Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

UNIVERZITA KARLOVA V PRAZE

matematicko-fyzikální fakulta

Petr Davídek

Implementace a experimentální srovnání vybraných metod pro správu XML dat

Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Irena Mlýnková, Ph.D.

Studijní program: Informatika

2008

Na tomto místě bych rád poděkoval vedoucí své bakalářské práce RNDr. Ireně Mlýnkové, PhD. za poskytnutí kolekcí XML dat, za odborné rady a náměty, které přispěly k dokončení této práce. Dále své rodině a přítelkyni Lence za neskutečnou trpělivost. Díky.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce

V Praze dne 25.5.2008 Petr Davídek

Obsah

Název práce: *Implementace a experimentální srovnání vybraných metod pro správu XML dat*

Autor: *Petr Davídek*

Katedra: *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *RNDr. Irena Mlýnková, Ph.D.*

E-mail vedoucího: *irena.mlynkova(at)mff.cuni.cz*

Abstrakt: V této práci jsou popsány m*etody ukládání XML dokumentů do relační databáze (RDBMS) na bázi kódování prvků XML stromu. Dotazování je řešeno pomocí překladu XPath dotazu na SQL dotaz. Implementovanými metodami jsou XISS/R, XpathAccelerator, PathStack. Použité metody jsou experimentálně srovnány na reálných kolekcích XML dat.*

Klíčová slova: *XML, XPath, RDBMS, databáze, XISS/R, XpathAccelerator, PathStack, TwigStack, Derby, Apache, Xerces, SAX*

Title: *Implementation and experimental comparison of methods for managing XML data*

Author: *Petr Davídek*

Department: *Department of Software Engineering*

Supervisor: *RNDr. Irena Mlýnková, Ph.D.*

Supervisor's e-mail address: *irena.mlynkova(at)mff.cuni.cz*

Abstract: This document describes some methods for storing XML documents in relational dabase (RDBMS) based on encoding elements in XML tree. Selecting is done by translation XPath selects into SQL selects. Document also contains experimental comparison of selected methods with real XML data collections. Implemented methods are XISS/R, XpathAccelerator, PathStack.

Keywords: *XML, XPath, RDBMS, database, XISS/R, XpathAccelerator, PathStack, TwigStack, Derby, Apache, Xerces, SAX*

1.Úvod

1.1.Popis problematiky

XML (eXtensible Markup Language [1] je dnes velmi moderní a často používaný pojem. XML navazuje na mnohem starší standard pro vytváření "značkovaných" dokumentů – SGML [2]. Tato norma je však pro reálné použití zbytečně složitá a stejně tak složitá je i její implementace. Proto byl na základě SGML navržen nový standard, který zachovává většinu z výhod SGML (univerzálnost), ale zjednodušuje vytváření a zpracování takových dokumentů - XML.

XML dnes hraje klíčovou roli při integraci systémů, při výměně informací mezi subjekty v prostředí internetu. Je základem dnes velmi populárních WebServices [3] a SOAP protokolu [4]. Používá se pro ukládání konfiguračních souborů, slouží pro ukládání informací databázového typu (např. telefonní seznam, výpisy telefonních hovorů, databáze občanů apod).

Proč zrovna XML?

Jednou z jeho největších výhod je jeho „sebepopisnost.“ To znamená, že při čtení XML dokumentu je patrné k čemu informace v něm obsažené mohou být, na rozdíl např. od csv (comma seperated value) [5] souborů či binárních dat. XML dokument tedy data obsahuje a do jisté míry je i popisuje, vysvětluje, má svůj sémantický význam. A to bez toho, aby k XML souboru musela být připojena jakákoliv dokumentace a popis navíc.

Další výhodou XML je, že podporu pro něj dnes najdeme téměř ve všech programovacích jazycích.

V neposlední řadě je výhodou XML jeho jednoduchost – jedná se o

textový formát, může být tedy zpracován libovolným textovým editorem.

XML dokumenty mohou být i poměrně rozsáhlé. Mohou obsahovat data, která jsou svoji povahou databázová (na XML se dá pohlížet i tak, že se jedná o textovou databázi). Samozřejmě vyvstává požadavek zpracování informací v takto velkých datech. Lze to udělat (poměrně) jednoduše? Nedala by se místo („prostého“) textového prohledávání použít nějaká existující robustní a prověřená technologie, která by ukládání a prohledávání XML dokumentů usnadnila? Jednou z (více) možností je pokusit se XML dokument uložit do RDBMS (relační databáze). Na prohledávání XML dokumentů pak můžeme použít již existující nástroje v podobě SQL dotazů [6], databázových indexů apod. Většina z komerčních RDBMS (např. Oracle [7]) má své metody ukládání XML dokumentů z velké části hotové. Ale i tyto systémy mají svá omezení (např. na velikost textového uzlu, počet najednou zpracovávaných značek apod.). Některé relační databáze mohou mít pouze velmi jednoduše XML dokumenty uložené jako LOB (Large Objects). Dotazování nad takto uloženými daty pak může být velmi neefektivní. Některé relační databáze ještě nemusí mít tuto podporu vůbec.

1.2.Cíl práce

Pokusíme se implementovat a experimentálně zhodnotit vybrané metody ukládání XML dokumentů do relační databáze. XML dokument bude do databáze uložen ve formě jednoduchých datových typů (integer a varchar). Dalším cílem je pak implementovat dotazování nad takto uloženými daty (pomocí překladu XPath dotazů do SQL dotazů) včetně výpisu celého cílového elementu a opět srovnat na reálných kolekcích XML dat.

1.3.Rozsah práce

Práce je tematicky členěna do následujících kapitol

- První kapitola se zabývá úvodem a cílem práce.
- V druhé kapitole je uveden popis použitých technologií, zejména pak RDBMS Derby.
- Třetí kapitola je věnována popisu implementovaných metod pro ukládání XML dat (XISS [8], XpathAccelerator [9], PashStack [10]).
- Čtvrtá kapitola je věnována popisu implementace metod v jazyce Java (programátorská dokumentace).
- Pátá kapitola je věnována uživatelské dokumentaci a instalační dokumentaci.
- Šestá kapitola se zabývá srovnáním naměřených výsledků.
- V sedmé kapitole je závěr práce, hodnocení a návrh možných rozšíření.

2.Popis použitých technologií

V této kapitole jsou popsány jednotlivé technologie (resp. ty jejich části), kterými se práce zabývá. Jedná se především o standard XML [1]. Dále pak o standard dotazování v XML dokumentu XPath [11]. Pro parsing (rozdělení XML dokumentu na jednotlivé elementární části) dokumentů je použit „de facto“ standard SAX [12]. Konkrétně implementace v jazyce Java Xerces [13]. Pro ukládání XML dat je použita open source RDBMS Apache Derby [14]. Té je věnována podstatná část kapitoly. Na závěr je uveden přehled možných metod pro ukládání XML dokumentů a způsob jejich dotazování.

2.1.Jemný úvod do XML

Jak již bylo zmíněno v úvodu XML (eXtensible Markup Language) je tzv. značkovací jazyk vycházející ze standardu SGML [2]. XML tvoří jeho podmnožinu. Podobně i jazyk pro psaní webových stránek HTML vychází z tohoto standardu. Značkovací jazyk znamená, že dokument obsahuje sadu značek (tzv. tag). Oproti HTML, kde je význam jednotlivých značek předem dán (např. title je název stránky) a jejich počet je omezen definicí HTML standardu, umožňuje XML definovat libovolné množství značek a přiřadit jim libovolný význam.
Nejprve malá ukázka XML dokumentu

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<!DOCTYPE knihovna SYSTEM "file:/home/medvjed/mujxml.dtd">
<knihovna>
   <kniha nazev="XML">
      <autor>Standa</autor>
      <rok_vydani>2005</rok_vydani>
   </kniha>
   <kniha nazev="HTML">
      <autor>Pepa</autor>
      <rok_vydani>2004</rok_vydani>
   </kniha>
</knihovna>
```

Struktura XML dokumentu je definována následovně:

- obsah značky (element) musí začínat startovní značkou ve tvaru *<nazev_tagu>*
- konec elementu je vyznačen koncovou značkou *</nazev_tagu>*
- před ukončením elementu musí být ukončeny všechny jeho vnitřní elementy - každý element tedy musí mít koncovou značku uvnitř stejného elementu jako počáteční
- prázdný element vypadá takto *<nazev_tagu/>*

- element může obsahovat textovou hodnotu *<nazev_tagu>TEXT</nazev_tagu>*
- počáteční značka může obsahovat atributy (s hodnotou) *<nazev_tagu atr1="aaa" atr2="bbb">*
- prázdný element může obsahovat atributy *<nazev_tagu attr="attr1"/>*
- znaky, které jsou součástí textových elementů a mají speciální význam, musí být nahrazeny speciální sekvencí (tzv. escape) – (např. '<' se nahrazuje sekvencí '<')
- nejvyšší element v hierarchii elementů je nazýván kořenový element (root element) a může být pouze jeden
- XML dokumentu musí začínat hlavičkou, kde je uvedena verze xml standardu a kódování *<?xml version="1.0" encoding="utf8" ?>*
- XML dokument může (ihned za hlavičkou) obsahovat *<!DOCTYPE nazev_root_elementu SYSTEM "kontrolni_soubor" >,* kde *„kontrolni_soubor“* je odkaz na soubor, ve kterém je definována struktura XML souboru ve formě DTD [15]
- XML soubor může obsahovat binární data (např. obrázky), ale vzhledem k tomu, že XML je textový dokument, musí binární data kódována do textové podoby (např. pomocí kódování BASE64 [16])
- Elementy mohou patřit do tzv. jmenného prostoru (namespace) [17] Pomocí jmenných prostorů lze zamezit kolizi názvů značek. Předpona jmenného prostoru se nachází na začátku značky a od názvu značky se odděluje se dvojtečkou. Hodnota jmenného prostoru je zadána jako hodnota atributu xmlns a nejčastěji se používá identifikace pomocí URI (RFC 3986). Příklad XML dokumentu s použitím jmenných prostorů:

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<ns1:knihovna xmlns:ns1="muj.pekny.namespace">
   <ns2:kniha nazev="XML" xmlns:ns2="muj.hezci.namespace">
```

```
        <autor>Standa</autor>
        <rok_vydani>2005</rok_vydani>
    </ns2:kniha>
</ns1:knihovna>
```

2.2.Well-formed a valid XML dokument

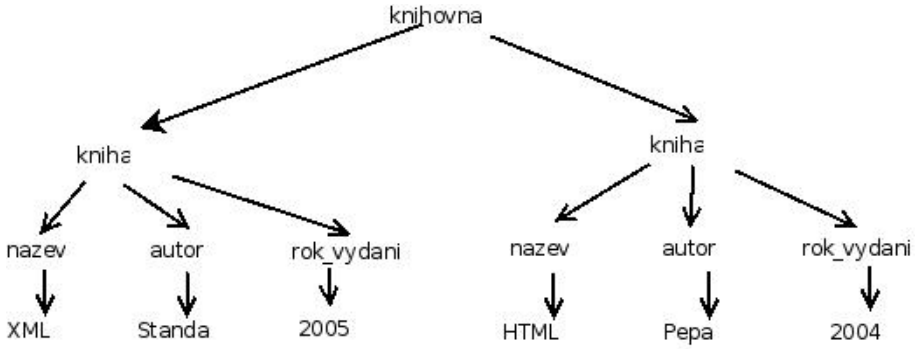
Pokud dokument splňuje všechny nutné podmínky, je tzv. well-formed. Pokud dokument navíc obsahuje odkaz na kontrolní soubor (tzv. schéma), může být provedena kontrola struktury dokumentu oproti tomuto souboru. Pokud dokument kontrolou projde je platný (valid). Pokud ne, tak je neplatný (invalid).

Nejčastějšími jazyky pro definici schémat jsou DTD [15] (Document Type Definition) a XSD [18] (XML Schema definition). Oba typy kontrolních souborů určují syntaxi XML dokumentu (jaké značky může dokument obsahovat, jak mají být strukturovány apod). DTD pochází ještě z dob SGML, ale postačuje pro jednoduché XML dokumenty. XML Schema oproti tomu má složitější a méně „průhlednou“ strukturu, ale umožňuje popsat libovolný XML dokument a každý XSD je současně XML dokumentem (tedy může být zpracováván naprosto stejným způsobem). Implementované metody v této práci předpokládají validitu XML dokumentu, a proto se platností dokumentů nezabývají (velmi jednoduše – viz programátorská dokumentace, kapitola 4– lze ovšem metody nastavit tak, aby se platností zabývaly). Na závěr malý příklad DTD dokumentu (k úvodnímu XML příkladu):

```
<!ELEMENT rok_vydani (#PCDATA)>

<!ELEMENT knihovna (#PCDATA | kniha)*>

<!ELEMENT autor (#PCDATA)>

<!ELEMENT kniha (#PCDATA | autor | rok_vydani)*>

<!ATTLIST kniha nazev CDATA #IMPLIED>
```

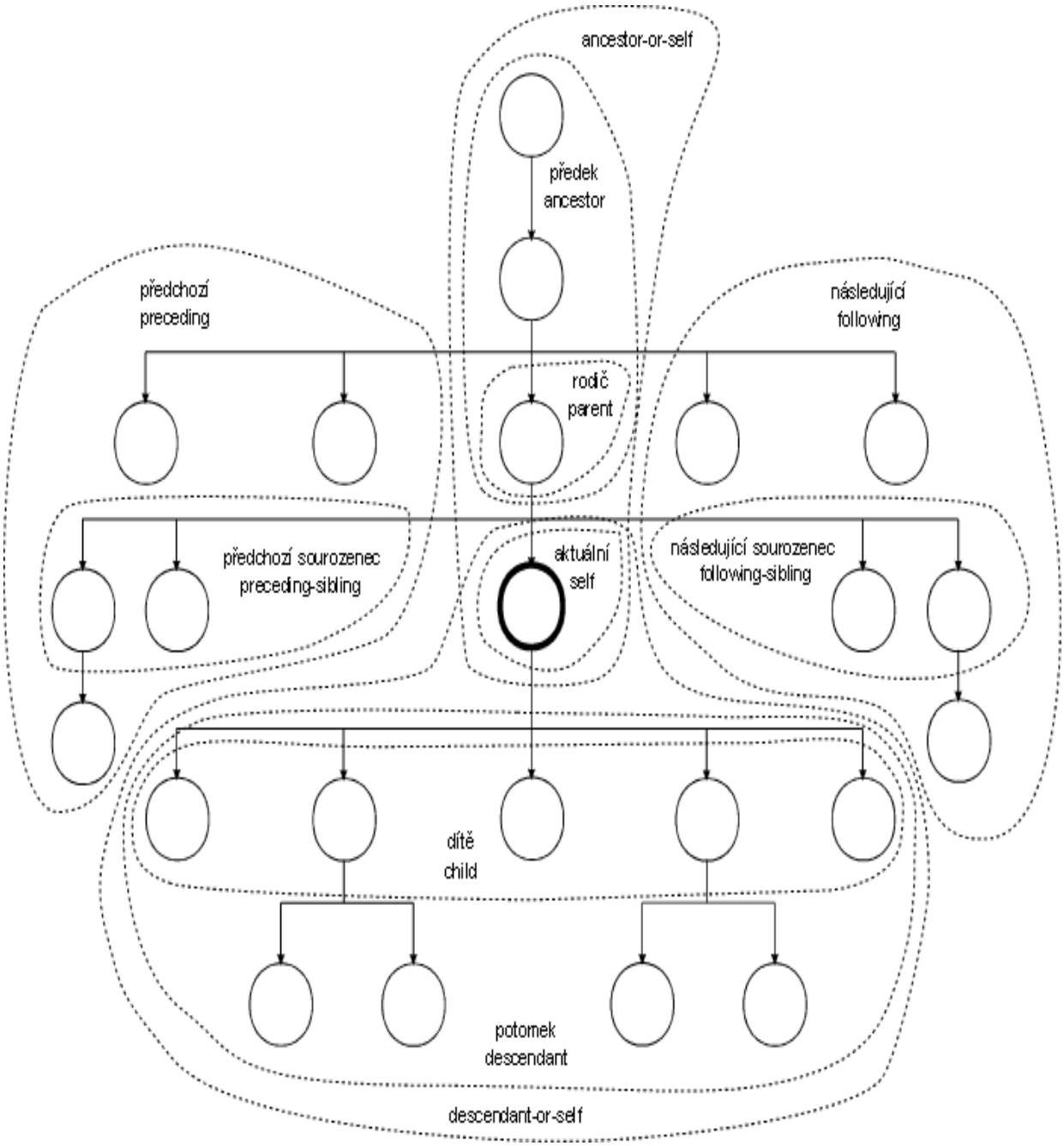
2.3.Osy v XML dokumentu a XPath dotazy

Na XML dokument je možné přirozeně pohlížet jako na stromovou strukturu (n-ární strom). Každý element je uzel, jeho přímí potomci jsou synové, nepřímí potomci se nacházejí v podstromu elementu, element bez potomků je listem, kořen je právě jeden, atributy (resp. hodnota atributu) jsou také listy atp. Strom z XML dokumentu lze jednoduše vybudovat tak, že při každém přečtení otevírací značky elementu se vytvoří nový uzel. Pokud se jednalo o kořenový element, vytvoří se kořen stromu. V ostatních případech se vytvoří vazba na poslední neuzavřený element. Při přečtení koncové značky elementu dojde k posunu do otce elementu. Stromovou strukturu nejlépe ilustruje obrázek 1.



Obrázek 1: XML strom

Xpath (jazyk na dotazování XML dokumentů) definuje pojem osa (axis). Jedná se o vyjádření vztahu mezi 2 uzly XML stromu, přičemž na jeden z nich se pohlíží jako na aktuální element a k němu se hledají elementy v dané ose. Typy a pojem os nejlépe osvětlí obrázek 2.



Obrázek 2: Osy (axis) v XML stromu

Specifikace jednotlivých os:

- self (aktuální element) – daný element
- parent (otec) – přímý nadřízený elementu
- child (syn) – přímý potomek elementu
- ancestor (předek) – element, který je jedním z předků elementu ve stromové struktuře

- ancestor-or-self – jako v minulém případě včetně aktuálního elementu
- descendant – libovolný potomek elementu
- descendant-or-self – jako v minulém případě včetně aktuálního elementu
- preceding – elementy, které byly (při sekvenčním čtení XML dokumentu) celé přečteny (včetně koncových značek) před aktuálním elementem
- preceding-siblings - jako v minulém případě, navíc elementy jsou bratry aktuálního elementu
- following – elementy, které byly (při sekvenčním čtení XML dokumentu) přečteny až po koncové značce aktuálního elementu
- following-siblings - jako v minulém případě, navíc elementy jsou bratry aktuálního elementu
- attribute – vybírá atributy (s hodnotou)

Specifikaci množiny XPath dotazů lze nalézt zde [11]. Tato práce se zabývá pouze jeho podmnožinou, která je uvedena dále (Tabulka 1). Pomocí XPath dotazu je možné z XML dokumentu vybrat množinu elementů (atributů či textových hodnot). Na úvod malý příklad k uvedenému vzorovému XML dokumentu.

- //autor – vybere z dokumentu všechny elementy, jejichž značka odpovídá (match) řetězci autor, tj výsledkem bude

```
<autor>Standa</autor><autor>Pepa</autor>
```

- /* - vybere kořenový element, tedy výsledkem bude

```
<knihovna>
   <kniha nazev="XML">
      <autor>Standa</autor>
      <rok_vydani>2005</rok_vydani>
   </kniha>
   <kniha nazev="HTML">
```

```
        <autor>Pepa</autor>
        <rok_vydani>2004</rok_vydani>
    </kniha>
</knihovna>
```

- //kniha[autor=Standa] – vybere element kniha (kdekoliv v XML dokumentu) takový, že obsahuje synovský podelement autor, který má textovou hodnotu Standa, tedy v našem případě

```
<kniha nazev="XML">
    <autor>Standa</autor>
    <rok_vydani>2005</rok_vydani>
</kniha>
```

Dalšími používanými prvky XPath dotazu, kromě výše uvedených os, jsou:

- * - odpovídá libovolnému řetězci XML dokumentu
- / - zkratka pro osu child
- // - zkratka pro osu descendant
- [] - slouží pro konstrukci podmínky v dané ose – např. pro porovnávání textových hodnot elementů
- ::node() - vybírá element
- ::text() - vybírá textovou hodnotu
- ::attribute() - vybírá atributy
- @ - zkratka pro ::attribute()
- @=* - vybírá atribut s libovolnou hodnotou
- nazev_tagu – vybírá element, který začíná (a končí) značkou nazev_tagu

2.4.Ukládání XML dokumentů do databáze

Aby bylo možné dokument uložit do databáze a následně nad ním pouštět množinu XPath dotazů překladem do SQL jazyka, je nejprve nutné dokument rozdělit na elementy, textové hodnoty, atributy. Pro toto rozdělení dokumentu se v této práci používá anglické slovo parse (parsing, naparsovat). Lépe označuje tuto činnost než slovo rozdělit.

Parsing dokumentů může být realizován 2 základními přístupy:

- DOM (Document Object Model) [19]
- SAX (Simple API to XML) [12]

2.4.1.Document Object Model (DOM)

DOM přečte XML dokument a vytvoří v paměti strom (např. obdobným způsobem, jaký je popsán v kapitole 2.3). Jeho nespornou výhodou je umístění v paměti (a tedy rychlý přístup k elementům dokumentu) a to, že odpovídá názorné představě stromu. Jeho velkou nevýhodou ovšem je opět uložení v paměti. Pokud bude dokument příliš velký, nemusí stačit VRAM (virtual RAM) přidělená procesu, který realizuje DOM. Vzhledem k zadání práce je třeba umět zpracovávat i velké dokumenty. Proto je použito rozhraní SAX, které toto omezení nemá.

2.4.2.Simple API to XML (SAX)

Tento přístup je vhodný pro zpracování (parsing) velkých XML dokumentů. Nevytváří totiž v paměti XML strom, ale čte postupně XML soubor a na základě přečtených dat spouští události. Událostí je přečtení značky, koncové značky a textového elementu. Nevýhodou může být to, že stromová struktura musí být „počítána“ v průběhu parsingu, tedy přístup klade větší nároky na program, který tento přístup používá.

2.5.DB Derby

Pro ukládání dat parsovaných XML dokumentů byla použita RDBMS Derby[14] od Apache Software Foundation. Důvodem je to, že se jedná o open source databázi napsanou v programovacím jazyce Java, která nativně podporuje JDBC rozhraní [20], tedy ukládání a dotazování je možné řešit z Javy standardní cestou. Další velkou výhodou je, že Derby podporuje tzv. hromadný (bulk) import dat do tabulek, což dost podstatně zkrátí dobu ukládání parsovaných dokumentů do databáze.
Základem projektu Derby je klon (snapshot) RDBMS IBM's Cloudscape [21]. IBM kód otevřela a umožnila komunitní vývoji Derby. Derby byla dlouhou dobu součástí Apache incubator projektu, což je jakási “líheň” pro nové produkty. Po nějaké době úspěšného „zrání“ v inkubátoru se projekt stane součástí rodiny standardních Apache projektů. Dnes je již Derby plnohodnotným projektem Apache Software Foundation. Vývoj DERBY je velmi bouřlivý (do konference derby-dev dojde každý den cca 50 mailů), aktuální stabilní verze je 10.4.

2.5.1.Trochu historie

- 1996 – Cloudscape, Inc startup – Oakland, CA
- 1997 – JBMS 1.0
- Duben 1999 – Cloudscape 2.0
- Prosinec 1999 – koupila firma Informix
- Červen 2001 – Cloudscape 4.0
- Červenec 2001 – koupila firma IBM
- Prosinec 2001 – IBM Cloudscape 5.0
- 2003 – IBM Cloudscape 5.1, 5.1FP1 & FP2
- Srpen 2004 – IBM otevírá kód pro komunitní vývoj

2.5.2.Derby Engine

Derby může běžet ve 2 módech – jako embedded databáze v nějaké aplikaci nebo v tzv. server módu. V server módu se o Derby “stará” nějaký aplikační server (AS) - pro něj je Derby embedded a AS vyřizuje požadavky webovských klientů. Derby je plnohodnotná relační databáze, umožňuje multiuživatelský/multivláknový přístup, obsahuje transakční model, umožňuje zamykání řádků, různé úrovně izolace, detekci zablokování (deadlock), recovery (obnova struktury databáze v případě pádu systému), zálohování a obnovu, tabulky, indexy, výjimky, apod.

2.5.3.Architektura Derby

Existují 2 úrovně pohledu na architekturu Derby

- Monitor

  Monitor je část kódu, která mapuje požadavky jednotlivých modulů na konkrétní implementaci. Např. používáme-li JDK 1.3, pak monitor pro implementaci použije Derby JDBC 2.0, zatímco u JDK 1.4 se použije JDBC 3.0, u JDK 1.6 JDBC 4.0. Monitor se také stará o startování a zastavování služeb.
- Modul

  Modul je (do jisté míry) nezávislá programová jednotka, mající na starosti konkrétní úkoly. Každý modul má nadefinované API - v jazyce Derby se mu říká protokol. Protokol je oddělen od jeho implementace.

RDBMS Derby je členěno do 4 vrstev

- JDBC

  Jedná se o nejvyšší vrstvu, poskytuje rozhraní (API) ke všem částem DERBY (implementace tříd java.sql and javax.sql pro

JDBC 2.0, 3.0, 4.0)

- SQL

  SQL vrstva se dělí na 2 hlavní oblasti – překlad (compilation) a běh (execution). SQL kompilace se děje v několika krocích - parsing dotazu (pomocí parseru vygenerovaného Javacc[22]) - výsledkem je strom objektů, dále namapování těchto objektů na konkrétní DB objekty (tabulky), optimalizace dotazu, vygenerování Java třídy (přímo v byte code), která reprezentuje "plán dotazu", a vytvoření instance této třídy. Plán dotazu je uložen do cache paměti a může být sdílen více připojeními. Běh SQL příkazu je pak volání metod instance této třídy. Výsledkem dotazu je objekt typu ResultSet, což není klasická JDBC třída, ale třída Derby ResultSet. JDBC vrstva pak vrací tento ResultSet aplikaci. Gramatika DERBY se nachází v java/engine-org/apache/derby/impl/sql/compile/sqlgrammar.jj ve zdrojových kódech Derby.

- Ukládací vrstva

  ResultSet je výsledkem práce ukládací vrstvy, která získává/ukládá data z/do DB. Dělí se na 2 hlavní podvrstvy – access a raw. Access vrstva představuje rozhraní k SQL vrstvě postavené na přístupu k tzv. conglomerates (tabulky, indexy) nebo konkrétním řádkům tabulek. Má na starosti prohledávání tabulek, indexů, vyhledávání indexů, indexování, třídění, uzamykání, transakce, míry izolace (stupeň zabezpečení vůči konfliktům). Access vrstva je nadvrstvou raw vrstvy. Raw vrstva má na starosti fyzické ukládání řádků do stránek a stránky do systému souborů (filesystem). Dále je zodpovědná za logování transakcí a jejich správu. JCE (nástroj pro kryptování DB dat) je integrován právě na úrovni stránek. Raw vrstva pracuje s API, které umožňuje ukládání dat na systém souborů, do jar souborů, jar souborů v daném class souboru apod. Derby ukládá tabulky a

indexy v kontejnerech, které jsou mapovány do souborů v adresáři seg0 konkrétní databázové instance. Každá DB instance má svůj vlastní adresář pojmenovaný dle názvu instance. Detaily ohledně způsobu ukládání dat lze nalézt zde [23].

- Služby

  Jedná se o kolekci modulů (např. správa zámků, správa cache, logování chyb, apod.) poskytujících nějakou konkrétní funkcionalitu. Dělí se na perzistentní (pevnou součástí DERBY) a dočasné (možno zapnout/vypnout).

2.5.4.Popis důležitých modulů

Všechny zdrojové kódy jsou uloženy v java/engine/org/apache/derby/ adresáři. Postup stažení zdrojových kódů lze nalézt zde [24]. Zde se uvádí důležité moduly a důležité adresáře zdrojových kódů.

- Autentikace – modul umožňující autorizované připojení do konkrétní DB. Umožňuje použití externích autentikačních metod jako LDAP, NIS +, Windows User Domain.
- Catalog – poskytuje informace o objektech v DB (tabulky, indexy, procedury, ..), synonymech, závislosti mezi objekty, poskytuje unikátní ID pro všechny objekty v DB.
- Database – poskytuje informace o konkrétní DB jako celku (readOnly), umožňuje manipulaci s DB jako celkem (backup, recovery, apod.).
- Diag – sada VTI (virtual table interface – virtuální tabulky nad systémovými logy) – dotazování nad transakcemi, systémovými chybami, SQLdotazy apod, informace o volném místě, o cachování, uzamčených objektech, apod.
- iapi, impl (iapi – API pro všechny třídy poskytované DERBY, impl – jejich implementace) – detailněji viz 2.5.5.

- io – základní ukládací funkce pro databázi, řeší pojmenování conglomerate souborů, poskytuje základní rozhraní pro ukládací vrstvu.
- JDBC – implementace standardního protokolu JDBC, správa připojení do DB, implementace XAConnection (typicky registrované přes JNDI - Java Naming and Directory Interface). XA je možné (zatím) použít pouze pro lokální databáze.
- loc – lokalizační soubory s chybovými hláškami, členěné dle modulů
- vti – rozhraní (interface) pro správu VTI (virtual table interface)

Detaily všeho popisovaného lze nalézt zde [14].

2.5.5.Implementační části API

Stručný popis modulů a adresářů implementační části Derby.

- load – import souborů do tabulek a export záznamů z tabulek do souboru (bude využito pro bulk import)
- io – ukládání perzistentních dat (tabulky), používáno DB enginem pro manipulaci s perzistentními daty, transakčním logem
- db – fyzická správa DB (uložená data, data v souborech), připojení do DB, backup, recovery
- jdbc – implementace JDBC pro DERBY (2.0, 3.0, 4.0)
- services/bytecode – překlad do bytecode (jazyk pro JVM), ladění
- services/JCE – šifrování/dešifrování dat v tabulkách
- services/cache – cache manager, podporuje stárnutí, růst a cache stránek, dotazů, otevřených souborů
- services/deamon – démon pro asynchronní IO operace, poskytuje asynchronní volání služeb
- services/locks – uzamykání prostředků, detekce deadlocků
- services/monitor – implementace monitoru, nahrávání jednotlivých služeb a modulů

- services/reflect – implementace classFactory (manipulace, správa Java tříd)
- services/stream – pro reportování informací (tracing, error)
- services/uuid – generování pseudonáhodných čísel (128bit)
- sql – kompletní správa SQL příkazů, parsing dotazu, příprava dotazu, spouštění dotazu
- store – ukládání a správa dat – vrstvy raw, access, počítání statistik (explain plan), implementace zámků

2.5.6.Stručný popis zajímavých utilit

Zdrojové kódy utilit se nacházejí v adresáři java/tools/org/apache/derby/tools

- sysinfo – informace o Javě a DERBY (verze, prostředí)
- dblook – zobrazuje informace o konkrétní DB instanci (DDL – data definition language)
- ij – interaktivní JDBC nástroj pro spouštění skriptů, SQL dotazů proti DB. Více viz uživatelská dokumentace, kapitola 5.12.

2.6.Teoretický základ pro implementaci XML technologie do RDBMS

V současnosti existuje poměrně pestrá škála metod, jak implementovat XML technologii do existující RDBMS. Tento přístup má nespornou výhodu v tom, že je možno využít robustního a propracovaného systému, který má některé rysy shodné s XML (na XML dokument je možno pohlížet i jako na textovou databázi). Je možno výhodně využít transakčního systému, indexů, optimalizací apod. Dotazování nad XML daty pak bývá řešeno překladem do SQL dotazu. Většinou nejsou podporovány elementy se smíšeným obsahem a zkoumání vztahů mezi elementy se omezuje na zkoumání vtahu předek <-> potomek.

2.6.1.Způsoby ukládání XML dokumentů

Nejjednodušším způsobem uložení XML dokumentu v databázi je uložení jako tzv. Large Object (BLOB/CLOB/BFILE). Dokument pak může být zpracováván nějakým XML parserem (např. XALAN [25]). Dotazování může být řešeno formou fulltextového (kontextového) vyhledávání, případně je možno použít regulárních výrazů (pokud je RDBMS podporuje). Úroveň tzv. roundrippingu (schopnost rekonstruovat XML dokument z DB úložiště) v tomto případě je 100%. Tato metoda ukládání patří do tzv. dokumentově orientovaných technik.
Tato práce se bude nadále věnovat pouze datově orientovaným metodám (kde úroveň roundtrippingu stačí nízká – např. tolik nezáleží na pořadí sourozenců v XML stromu, ignorují se sekce CDATA, komentáře, apod).

2.6.2.Přehled vybraných mapovacích metod

Nyní uvedeme některé mapovací metody – metody transformace XML z/do DB schématu a metody indexování těchto XML dokumentů. Není cílem podat vyčerpávající popis, jen zde uvést seznam metod pro čtenářovu představu. Detailnější informace lze nalézt zde [26] a zde [27].

- Table based mapping [28]– DB schéma je dáno přímo obsahem XML dokumentu (vhodné pro exporty DB)
- Generic tree mapping [29] – umožňuje uložit libovolný XML dokument pomocí tzv. hranového mapování. Hranové mapování lze implementovat následovně: Řádek tabulky představuje hrany XML stromu (opět představa XML dokumentu jako stromu). Každému uzlu je přiřazeno jednoznačné ID, získané průchodem stromu v preorder pořadí. Sloupce tabulky jsou tvořeny : ID, cílové ID, značka (markup), pořadí, obsah. Atributy mají tedy koncové ID=0. Pro rychlejší přístup je možno vytvořit indexy

(značka, obsah), (cílové ID), (ID, pořadí). Toto uložení má však problémy – netriviální překlad XPath do SQL, obtížné hledání vztahů předek <-> potomek. Dalšími metodami jsou atributové mapování, univerzální mapování, normalizované univerzální mapování, které představují různá rozšíření hranového mapování.

- Structure centered mapping [30] – uzly mají stejnou strukturu (typ, název, obsah, seznam potomků). Pro efektivní prohledávání vztahů existují různá tzv. číselná schémata , která každému uzlu přiřazují čísla, na základě kterých je možno určit tyto vztahy (předek <-> potomek). Použitelné metody jsou DF (deep first), SCF [31] (simple continued fraction), Dietzovo číslování [32], DLN [33] (dynamic level numbering).
- Simple path mapping [34] – hodí se pro XPath dotazy, ukládání celých cest z kořene až do listu, možno pak použít operátoru like, DB schéma obsahuje 4 tabulky (element, atribut, text, cesta).
- Schématem řízené mapování [26]– pokud existuje popis XML dokumentu (XML Schema, DTD) na jehož základě se vytvoří DB schéma. Podelementy s maximálním výskytem 1 jsou přidány do relace nadelementu (tzv. inlining), elementy s nepovinným výskytem jsou mapovány na sloupce s null hodnotami, podelementy s vícenásobným výskytem jsou mapovány do samostatných tabulek, vztah element <-> podelement je mapován pomocí klíčů a integritních omezení. Existuje řada algoritmů – např. basic, shared, hybrid.
- Constraints preserving mapping [35] – transformace XML dokumentu do EER (rozšířený ER-diagram – slouží pro návrh a modelování entit a jejich vztahů (kardinalit))
- LegoDB mapping [34] – vytvoří se (na základě nějakých zkušeností a pomocí heuristik) množina možných mapování XML schématu na DB schéma. Poté se na tuto množinu aplikují testy – vkládání a dotazování nějakého vzorku XML dat a jako

definitivní mapování je vybráno to, které z testu vyjde nejlépe.

- UB-trees [36](Universal B-trees) – indexovací metoda postavená na tzv. Z-ordering a B-stromech. Umožňuje pracovat i se smíšeným obsahem.

2.6.3.Metody dotazování a optimalizace dotazů XML dokumentů

- Schema-Free Xquery [37]- rozšíření Xquery, postavené na MLCAS (Meaningful Lowest Common Ancestor Structure), vhodné pro dotazování XML u kterého neznáme detailně strukturu
- Překlad Xquery/XPath dotazů do SQL – překlad výrazů do SQL pomocí metody dynamic interval encoding, Tree patterns, General tree patterns
- Structural Join Order Selection [38] – cost-based metody pro optimalizaci XQuery/XPath dotazů - Dynamic Programming, Dynamic Programming with Pruning, Dynamic Programming with Aggressive Pruning a jejich heuristiky
- Dotatování překladem XQuery/XPath do algebry Tree Logical Class (TLC) a rozšíření této algebry - Ordering and Duplicate specifications

Více informací lze opět nalézt zde [26] a zde [27].

2.7.XML a Derby

V současné vývojové verzi DERBY existuje pouze základní podpora pro ukládání XML. Sloupec v tabulce může být typu XML, pak je ukládán jako UTF8 řetězec do systému souborů. Základní třídy, které s tímto typem pracují jsou:

- org.apache.derby.iapi.types.XMLDataValue – rozhraní definující

základní metody pro práci s XML (xmlexists, xmlserialize, xmlparse)

- org.apache.derby.iapi.types.XML – implementace rozhraní XMLDataValue a DataType
- org.apache.derby.impl.sql.xml.XMLImp – implementace toho, jak je XML dokument čten, ukládán, dotazován, kopírován do paměti, apod.
- org.apache.derby.impl.sql.xml.XML_UTF8Impl – konkrétní implementace prvotního zpracování (ukládání apod.) XML v Derby. XML se chová téměř jako typ varchar, pro parsing je použit procesor Xerces, pro dotazování použit Xalan (součást JDK). Při manipulaci s XML je používáno tzv. impl_id, které představuje verzi XML ukládání

Tato práce se zabývá metodami, které jsou na databázovém „frontendu“. Tedy je použito standardního rozhraní (JDBC) pro ukládání a dotazování. Teoreticky by bylo možné tyto metody implementovat na databázovém „backendu“, tedy přímo v kódu databáze. To by mohlo vést ke zlepšení zejména dotazovacích technik, protože by se mohly použít speciální typy indexů, které by lépe vyhovovaly potřebám metod. Ačkoliv je to nad rámec této práce, krátce se zmíníme o tom, kde by se dalo začít s implementací metod do DB „backendu“:

- Upravit následující interface (org.apache.derby.iapi.types.XML) a v něm metody DataValueDescriptor, Storable, StreamStorable, XMLDataValue
- Založit novou třídu pro novou implementaci XML, která rozšíří (extenduje) org.apache.derby.iapi.types.XML a implementovat všechny potřebné metody
- Opravit dotazování, indexování, přidat nové výjimky, apod.

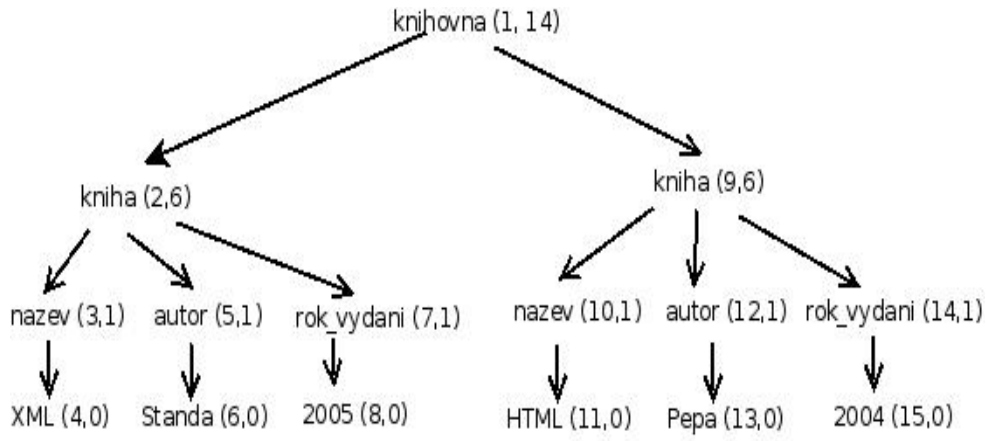
3.Popis implementovaných metod

V této kapitole budou postupně představeny klíčové myšlenky následujících metod pro ukládání a dotazování XML.

- XISS/R (XML Indexing and Storage systém using RDBMS)
- XpathAccelerator
- PathStack

3.1.XISS/R

Základem této metody je kódování prvků XML dokumentu, tedy elementů, (resp. jejich značek), atributů a textových hodnot. Kódování prvků znamená, že je každému prvku XML dokumentu přiřazena n-tice čísel. Použité kódování se nazývá „The Extended Preorder Numbering Scheme“[8]. Základem tohoto kódování je dvojice čísel <pořadí, velikost> (<order, size>). Pořadí prvku je pořadí jeho otevírací značky při sekvenčním čtení dokumentu od jeho počátku (průchod stromu v tzv. inorder pořadí). Velikost udává počet podprvků daného prvku. Tedy atributy a textové hodnoty mají velikost 0, listové elementy neobsahující textovou hodnotu ani atribut také velikost 0. Princip kódování nejlépe vysvětlí následující obrázek (použito XML z úvodu kapitoly 2.1).



Obrázek 3: XISS: kódování prvků

Z principu tohoto kódování plyne následující jednoduché tvrzení:

Pro libovolné 2 prvky XML dokumentu platí následující: Prvek y je potomkem prvku x právě tehdy když

$$order(x) < ordey(y) <= order(x) + size(x)$$

Tato informace plně dostačuje k nalezení os descendant a descendant-or-self, ancestor a ancestor-or-self. Dále je velmi jednoduše vidět, že platí i následující tvrzení:

Prvek x je following (resp. preceding) prvku y, právě tehdy když

$$order\ (x) > order\ (y)\ (resp.\ order(x) < order(y))$$

Tato informace stačí i k pohybu po osách preceding a following. Aby bylo možné nalézt syna a otce a tedy umožnit pohyb po ose parent a child, a preceding-sibling a following-sibling, je nutné kódovací dvojici rozšířit o další atribut – parent (pořadí otcovského elementu). Kořenový element může mít např. hodnotu parent = -1. Tedy platí další pomocné tvrzení:

Pro libovolné 2 prvky (kromě kořenového elementu) XML dokumentu platí následující: Prvek x je otcem prvku y, právě tehdy když

$$order(x) = parent(y)$$

Autoři metody dále rozšiřují tuto trojici o další atributy:

- attr – order prvního atributu elementu

- next – order příštího bratra
- child – order prvního syna

Počítání těchto atributů však vyžaduje o jeden přístup k elementu více než je třeba (je třeba nastavit order u předchozího bratra). A navíc je nutné držet bratra v paměti, dokud se nepřečte další bratr. Což u velmi rozvětvených XML stromů může působit problémy s velikostí VRAM. Proto uzpůsobíme atributy následovně:

- attr – order prvního atributu elementu (zůstává stejné)
- prev – order přechozího bratra
- child – order posledního syna

Pokud zpracováváme více dokumentů je ještě nutné přidat atribut *did,* který udává ID dokumentu. Pokud se v nadřízeném elementu uloží informace o počtu synů, dostaneme ekvivalentní informaci o struktuře XML jako autoři XISSu, ale za „menší cenu“. Attr se využije při konstrukci finálního elementu (pro rychlé zjištění, že element má atribut). Atributy child, prev a počet synů by bylo možné využít pro XPath dotazy, které se ptají na pořadí elementu v rámci nadelementu.

Metoda je použitelná ve 2 variantách (varianta A a B), podle toho, do kterých tabulek jsou data ukládána. DB schéma varianty (A), kterou implementuje tato práce, je uvedeno v programátorské dokumentaci (Obrázek 9). Základem DB modelu jsou 4 tabulky, do kterých se ukládají elementy, atributy, textové hodnoty a dokumenty. Navíc je přidána tabulka pro dlouhé textové hodnoty.

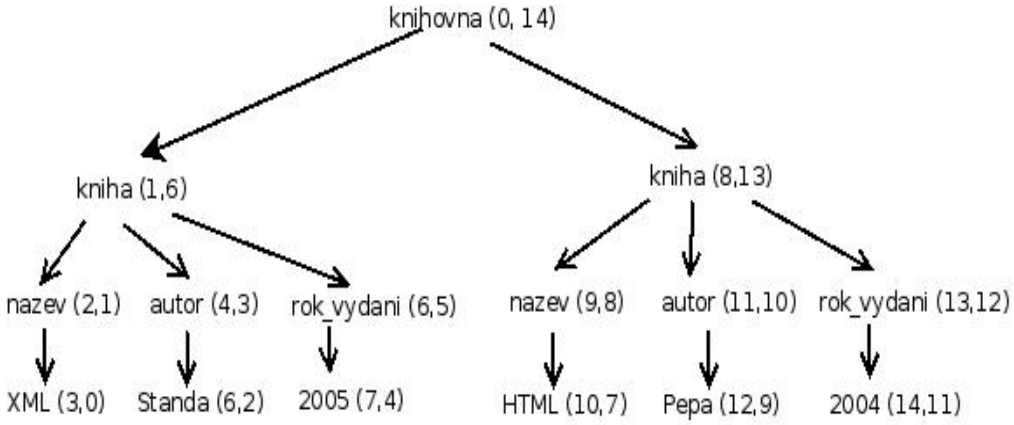
Druhá varianta (B) je velmi podobná té první, jen jsou data rozdělena (tzv. partitioning) podle názvu značek a atributů. Pokud RDBMS neumožňuje interně partitioning, je možné ho implementovat pomocí mnoha různých tabulek, jejichž název bude odvozen od názvu značky a atributu. Tabulka dokumentů a textových elementů je stejná jako ve variantě A.

Aplikace XPath dotazů je řešena překladem do SQL. V dotazu jsou

postupně nacházeny XPath osy a na základě uvedených vlastností kódování je skládán SQL dotaz. Více možno nalézt v programátorské dokumentaci v kapitole 4.2.2.

3.2.XpathAccelerator

Metoda XpathAccelerator je popsána zde [9]. Opět se zde používá kódování prvků XML dokumentu. Základem kódování je dvojice čísel (pre,post), které tvoří tzv. pre-post okénko. Atribut pre je počítán stejně jako v metodě XISS, tedy jeho hodnota je navyšována při inorder průchodu XML stromu při každé počáteční značce. Oproti tomu je hodnota post navyšována při každém uzavření značky opět při inorder průchodu stromem. Opět se rozlišují atributy, textové hodnoty, atributy s hodnotou, každý z těchto prvků má přiřazenu onu dvojici (pre, post). Kódování nejlépe osvětlí následující obrázek (použit XML dokument z úvodu kapitoly 2.1).



Obrázek 4: Xpath Accelerator: kódování prvků

Z principu tohoto kódování plyne následující jednoduché tvrzení:

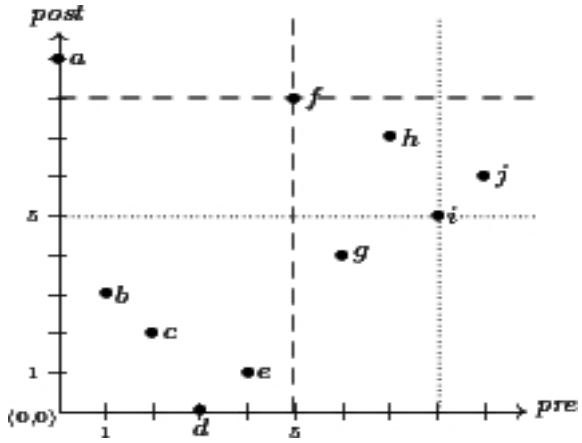
Pro libovolné 2 prvky XML dokumentu platí následující: Prvek y je potomkem prvku x právě tehdy když

pre(x) < pre(y) a zároveň post(y) < post(x)

Názorná představa okénka vznikne při zakreslení prvků do 2-rozměrného grafu se souřadnicemi pre (x-osa) a post (y-osa). Graf se rozdělí pomocí kolmých přímek ze všech bodů na oxy x a y. Pro každý prvek (bod) tak vznikne 6 oblasti (levá, pravá, levá horní, pravá horní, levá dolní a pravá dolní). V levé horní oblasti se nachází všechny ancestor prvky, v levé oblasti všechny preceding prvky, v pravé oblasti všechny following prvky a v pravé dolní oblasti všechny descendant prvky. Pokud budeme uvažovat následující XML strom,

Obrázek 5: XML strom

graf os bude vypadat následovně:



Obrázek 6: pre-post okénka

Z grafu a XML stromu je velmi jednoduše vidět, že platí i následující

tvrzení:

Prvek x je following (resp. preceding) prvku y, právě tehdy když

$$pre\ (x) > pre\ (y)\ (resp.\ pre(x) < pre(y))$$

Tato informace stačí i k pohybu po osách preceding a following. Aby bylo možné nalézt syna a otce a tedy umožnit pohyb po ose parent a child, a preceding-sibling a following-sibling, je nutné kódovací dvojici rozšířit o další atribut – parent, kde se bude ukládat pre atribut otcovského elementu. Kořenový element může mít např. hodnotu parent = -1. Tedy platí další pomocné tvrzení:

Pro libovolné 2 prvky (kromě kořene, který má parent=-1) XML dokumentu platí následující: Prvek x je otcem prvku y, právě tehdy když

$$pre(x) = parent(y)$$

Dalšími potřebnými atributy jsou

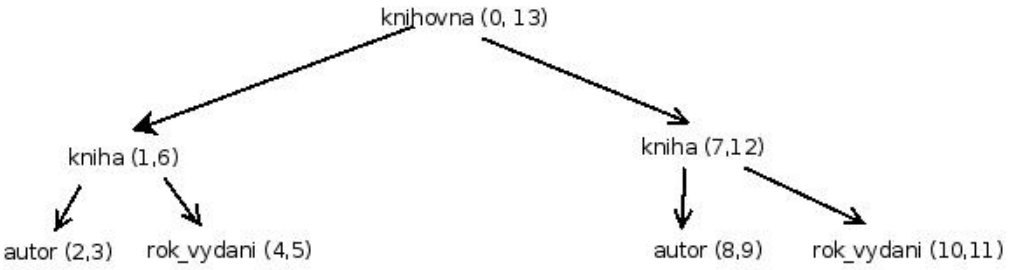
- did – ID dokumentu, které je nutné, pokud se zpracovává více XML dokumentů (pro jejich rozlišení)
- heigth – udává maximální výšku podstromu. Může být použito pro optimalizaci SQL dotazu a je použito pro zjišťování, zda se jedná o listový prvek. Textová hodnota má vždy výšku 0, atribut bez hodnoty také 0, atribut s hodnotou výšku 1.
- att – slouží pro rozlišení prvků, zda se jedná o značku, textovou hodnotu, atribut, atributovou hodnotu

Pro ukládání dat do DB je použita pouze jedna základní tabulka, kam se ukládají všechny prvky, dále tabulka pro dokumenty a tabulka pro

dlouhé texty. DB schéma je možno nalézt v kapitole 4.3.
Autoři metody uvádějí, že pro efektivní dotazování je možné použít speciální indexy (R-stromy, B-stromy)[9]. Vzhledem k tomu, že naše metoda je implementována na DB frontendu, používá klasické databázové indexy (tedy záleží na DB enginu jakým způsobem indexy vytváří a jak je prohledává). Jistě by bylo zajímavé metodu implementovat na DB backendu a použít vlastní indexy. To je však nad rámec této práce.
Aplikace XPath dotazů je řešena překladem do SQL jazyka. V dotazu jsou postupně nacházeny XPath osy a na základě uvedených vlastností kódování je skládán SQL dotaz. Více možno nalézt v programátorské dokumentaci v kapitole 4.3.2.

3.3.PathStack

Metoda PathStack je popsána zde [10]. Základem je velmi podobné kódování jako u XpathAcceleratoru - dvojice čísel (leftp, rightp). Atribut leftp je počítán stejně jako v metodě XISS, XpathAccelerator, tedy jeho hodnota je navyšována při inorder průchodu XML stromu při každé počáteční značce. Hodnota rightp je navyšována při každém uzavření značky opět při inorder průchodu stromem. Rozdíl je v tom, že hodnoty leftp a rightp jsou brány z jedné číselné řady a tato řada není ukončena ani při přečtení nového dokumentu. Možný úhel pohledu je ten, že všechny zpracované dokumenty leží pod uměle vytvořeným kořenovým elementem. Výhodou je, že není třeba se zabývat atributem *did* při hledání cílových elementů v rámci XPath dotazu. Dalším rozdílem oproti XpathAcceleratoru je, že textové hodnoty jsou uloženy ve sloupečku přímo u daného elementu a pro atributy je vytvořena zvláštní tabulka. Schéma DB modelu lze nalézt zde (Obrázek 11).
Kódování opět nejlépe osvětlí následující obrázek:

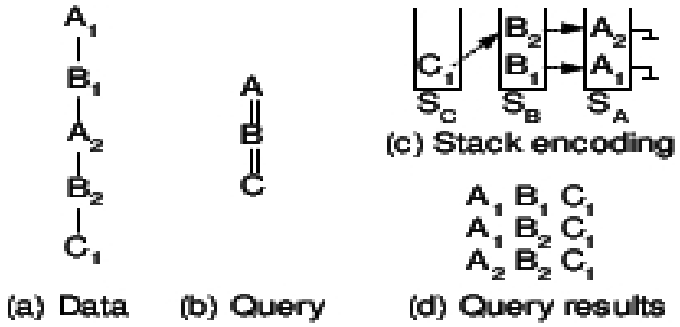


Obrázek 7: PathStack: kódování prvků

Dotazování je řešeno překladem XPath dotazů do SQL. Oproti předchozím metodám zde není použito DB enginu pro spojování výsledku, ale pouze pro dotažení dílčích výsledků pro jednotlivé elementy. Princip dotazování funguje následovně:
XPath dotaz se nejprve rozdělí tzv. QueryNody (QN) podle os (z důvodu jednoduchosti zde uvažujeme pouze osy child a descendant). Mezi QN se udržuje vazba otec <--> syn, dohromady tvoří tzv. QueryTree. Každému QN tak odpovídá jeden element (QN má přiřazenou značku, pokud byla v XPath dotazu zadána, nebo znak *). Na základě této značky se každému QN přiřadí databázový kurzor, který podle značky vybírá záznamy z databáze. Dále je každému QN přiřazen zásobník, na který se odkládají potencionální výsledky XPath dotazu a zároveň odkaz do zásobníku otce v QT stromu (to je kvůli správnému vypisování výsledků.
Detaily algoritmu a důkaz správnosti jsou uvedeny zde [10]. Princip algoritmu: Postupně se prochází QT, hledá se v něm uzel, který má minimální leftp na vrcholu svého zásobníku (postupuje se začátku XML dokumentu). Vybraná hodnota leftp je použita pro čištění zásobníků všech QN. Pokud je na vrcholu zásobníku hodnota rightp, která je menší než vybraná hodnota leftp, je jasné, že potencionální výsledek nepatří do množiny správných výsledků. Proto je ze zásobníku odebrán. U vybraného QN se na jeho zásobník uloží hodnota leftp (spolu s rightp a s odkazem na vrchol zásobník otce QN) a posune se

DB kurzor (QN) o další položku vpřed. Pokud je vybraný QN listem v QT, zavolá se funkce na výpis výsledků. Ta na základě vazeb mezi zásobníky jednotlivých QN vypíše možný výsledek.

Následující obrázek ukazuje stav zásobníků a vypsané výsledky pro konkrétní případ.



Obrázek 8: PathStack - příklad

Data ukazují příklad XML stromu, Query příklad XPath dotazů jako QT (XPath dotaz je zde /a//b//c). Stack encoding ukazuje stav zásobníků přiřazených k daným QN. Query results jsou výsledky. Z příkladu je patrné, že nebyl vypsán potencionální výsledek A2B1C1, protože ukazatel B1 v zásobníku SB neukazuje na A2, ale pouze na A1. Tedy je jasné. že A2 se nachází v XML dokumentu pod elementem B1.

Na závěr ještě uvádíme tabulku implementovaných typů XPath dotazů pro jednotlivé metody (Tabulka 1).

Dotaz/Medota	XISS	XpathAccelerator	PathStack
child	X	X	X
descendant	X	X	X
parent	X	X	
attribute()	X	X	
preceding-siblings	X	X	
following-siblings	X	X	
preceding	X	X	
following	X	X	
descendant-or-self	X	X	
ancestor-or-self	X	X	
node()	X	X	
text()	X	X	
Wild *	X	X	X
Tag_name	X	X	X

Tabulka 1: Typy XPath dotazů pro jednotlivé metody

4.Programátorská dokumentace

V této kapitole budou ukázány praktické příklady použitých technologií a dále detailnější popis toho, jak jsou metody naprogramovány.

4.1.Obecný úvod ke všem metodám

Pro detailnější informace o funkcích, metodách je možno použít

přiloženou JavaDoc, případně odkazujeme přímo do java kódu, je bohatě komentován. Tato dokumentace si neklade za cíl vysvětlení principu používaných algoritmů, pouze popisuje jejich průběh. Detaily algoritmů jsou popsány v předešlé kapitole 3.

Pro ukládání dat je použita relační DB Apache Derby. Testovány byly verze 10.2. a (zatím) poslední verze 10.4. Verze 10.4 již umí bulk import CLOB sloupečků.

Každá z popsaných metod má svůj specifický databázový model. Každá z metod má vytvořenu svoji instanci RDBMS, ve které je tento model vytvořen. Pokud daná databázová instance neexistuje, je vytvořena na začátku importu dat (pasingu XML dokumentů) - při volání JDBC connect (pomocí parametr *create=true*). Připojení do DB vypadá takto:

Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();
DriverManager.getConnection("jdbc:derby:<instname>;create=true");

Tímto voláním vznikne v aktuálním adresáři nový adresář s názvem databázové instance (instname). V tomto adresáři se nacházejí všechny DB objekty dané instance. Pokud instance existuje, dojde pouze k připojení k této instanci.

Pro ukládání a dotazování je použito rozhraní JDBC, které má Derby interně implementováno. Ukládání dat do DB je realizováno následovně:

- Pro každou tabulku daného DB modelu je vytvořen dočasný csv soubor. Struktura cvs souboru musí odpovídat sloupečkům tabulky.
- Pro každý soubor je vytvořen zapisovač s vyrovnávací pamětí (*BufferedWriter).* To tohoto bufferu jsou zapisována naparsovaná

data a buffer si sám řídí ukládání do souboru. Před nahráním souboru do tabulky je nutné buffer uzavřít.

- Pro nahrání dat do DB je použita interní funkce Derby pro hromadný import dat do tabulek (tzv. bulk import) - (*SYSCS_UTIL.SYSCS_IMPORT_TABLE*). Použití je následující:

```
stmt=conn.prepareStatement("CALL
SYSCS_UTIL.SYSCS_IMPORT_TABLE (?,?,?,?,?,?,?)");
stmt.setString(1,"<schema>");
stmt.setString(2,"<tab_name>");
stmt.setString(3,"<file_name>");
stmt.setString(4,"<col_separator>");
stmt.setString(5,"<text_separator>");
stmt.setString(6,<character_encoding>);
stmt.setInt(7,1);
stmt.execute();
```

Nejprve se připraví volání funkce, pak se nastaví parametry:

- DB schéma (defaultně APP)
- Název tabulky
- Soubor se zdrojovými daty
- Oddělovač sloupců v csv souboru
- Ohraničení položek v csv souboru
- Kódování souboru
- Příznak, zda data přidat to tabulky (1), či tabulku přepsat (0). V naší implementaci algoritmu můžeme použit obě hodnoty, protože tabulky jsou při každém spuštění importu smazány (drop).

Pro dotazování jsou použity metody *createStatement* a *executeQuery*. *CreateStatement* vytvoří SQL dotaz na základě textového řetězce a *executeQuery* tento dotaz pustí proti danému schématu. Výsledky jsou přiřazeny do struktury *ResultSet*. Vytvořením *ResultSetu* v podstatě

vytvoříme databázový kurzor.

ResultSet result = stmt.executeQuery(<select>);

Výsledky jsou zobrazeny postupným procházením množiny výsledku (DB cursoru):

while (result.next()) printResults;

Pro potřeby této práce byla zvolena implementace SAX od Apache Software Foundation Xerces2 [13]. Důvod je ten, že poskytuje SAX rozhraní pro programovací jazyk Java, nejedná se o komerční parser a plně vyhovuje potřebám této práce. Mezi další implementace SAX patří např. Fusion XML SAX Parser [39], nebo standardní GNU knihovna libxml2.

Xerces je SAX parser napsaný v Javě. Parser funguje následovně: čte proud dat na vstupu (XML soubor) a pokud nastane událost (např dojde k přečtení značky), je vyvolána příslušná obslužná funkce. Parser hlídá, zda je dokument well-formed.
Nejprve je nutné nastavit použití parseru:

String parser = "org.apache.xerces.parsers.SAXParser";
XMLReader xml = XMLReaderFactory.createXMLReader(parser);

Pro každý dokument je vytvořena nová instance třídy *XML_SAX_READER*, která má parsing na starosti. Tato třída je potomkem třídy *DefaultHandler*, která je definována ve standardním rozhraní Xerces2.

Parsing jednoho konkrétního dokumentu je realizován následovně:

XML_SAX_READER handler = new XML_SAX_READER();

Při běhu parseru rozlišujeme 3 události (obsloužené 3 funkcemi):

- začátek elementu (byl přečten počáteční tag) – zavolá se metoda *startElement*. Tato metoda zároveň obsluhuje i atributy, které jsou dostupné jako pole párů „atribut:hodnota“
- konec elementu (přečten koncový tag) – zavolá se metoda *endElement*
- přečtení textové hodnoty – zavolá se metoda *characters.* Textové hodnoty jsou ve všech metodách rozlišovány na krátké (délka < 200 znaků) a dlouhé. Důvod je ryze praktický – XPath dotazy typicky neobsahují dotazy, které by specifikovaly tak dlouhé texty. Navíc i indexace dlouhých textů by byla značně náročná. Dlouhé texty jsou tedy používány pouze při výpisu výsledných XML fragmentů. Ve všech uvedených algoritmech lze jednoduše toto omezení odstranit, ale bude to na úkor rychlosti dotazů, které obsahují dotaz na textový element.

Tyto obslužné funkce se pro konkrétní metodu (XISS, XPathAccelerator, PathStack) samozřejmě liší. Každá z metod obsahuje 2 balíky – Import a XPath:

- Balík import čte XML dokumenty z daného adresáře, pomocí Xerces parseru je parsuje, vytváří si v paměti struktury, které následně uloží do pomocných souborů. Po dokončení běhu parseru jsou data ze souborů bulk importem nahrána do databáze do příslušných tabulek. Pokud dojde k chybě při parsingu XML dokumentu (document not well-formed), je tato chyba vypsána, a pokračuje se dalším souborem. Parser nepoužívá validaci oproti schématům (DTD, XSD). To je zajištěno nastavením těchto tzv. features:

  xml.setFeature("http://apache.org/xml/features/nonvalidating/load-

external-dtd" , *false);*
xml.setFeature("http://xml.org/sax/features/validation" , *false);*

- xpath naparsuje zadaný XPath dotaz a transformuje ho na odpovídající select, který je následně puštěn nad danou databází. Tím jsou nalezeny všechny cílové elementy. Pro každý z takto nalezených elementů je provedena jeho zpětná rekonstrukce.

4.2.Metoda XISS

Základem této metody je specifické kódování značek, atributů a textových elementů. Kódováním rozumíme přiřazení číselných identifikátorů každému prvku XML dokumentu (prvek XML je tag/text/ atribut). Na základě tohoto kódování je možno určit kdo je čí otec, kdo čí syn, potomek, předek apod.

Základem použitého kódování je atribut *order* (unikátní ID každého prvku XML dokumentu, které se zvětšuje o +1 při každém přečteném prvku). Spolu s *did* (*document_id*) tvoří složený primární klíč tabulek *attr_tab*, *elem_tab*, *text_tab*. Dále se používá atribut *depth* (hloubka zanoření prvku v XML stromu), *parent_id* (*order* otce prvku), *size* (mohutnost prvku, tj. kolik obsahuje „podprvků“). Dále pak pomocné atributy *child_id* (order prvního potomka prvku), *prev_id* (*order* předchozího bratra – bratr je prvek, který má stejné *parent_id* (a stejnou *depth*)). A nakonec *attr_id* který odkazuje na *order* atributu.

XML dokumenty jsou uloženy do následujícího DB modelu:

- V tabulce **DID_TAB** jsou uloženy dvojice ID dokumentu (*did*) a název dokumentu (*name*).

- V tabulce **ELEM_TAB** jsou uloženy elementy jako n-tice (*name, did,*

order, size, depth, parent_id, child_id, prev_id, attr_id). Atribut *name* je název tagu, kterým element začíná.

- V tabulce **TEXT_TAB** jsou uloženy texty do velikosti 256 znaků jako n-tice (*did, order, depth, parent_id, value*). Atribut *value* představuje textovou hodnotu elementu.

- Texty, které jsou delší než 255 znaků jsou ukládány do tabulky **LONGTEXT_TAB** jako tzv. LOB (Large Object), konkrétně CLOB (Character LOB). Vkládanou n-tici tvoří stejně jako v minulém případě (*did, order, depth, parent_id, value*), kde *value* je textová hodnota elementu.

DID_TAB

Column	Type	Nullable	Default	Comments
DID	**INTEGER**			**ID dokumentu**
NAME	VARCHAR2(128)	Y		nazev dokumentu

Key	Column(s)	Type
PK_XDOC	DID	P

Index	Column(s)	Type
PK_XDOC	DID	unique

ELEM_TAB

Column	Type	Nullable	Default	Comments
NAME	VARCHAR2(256)	Y		nazev elementu
DID	**INTEGER**			**ID dokumentu**
NID	**INTEGER**			**node ID elementu**
size	INTEGER	Y		pocet podprvku elementu
DEPTH	INTEGER	Y		hloubka zanoreni elementu v XML str
PARENT_ID	INTEGER	Y		nid otce elementu
CHILD_ID	INTEGER	Y		nid posledniho bratra
PREV_ID	INTEGER	Y		
ATTR_ID	INTEGER	Y		nid prvniho atributu elementu

Key	Column(s)	Type
PK_XELEM	DID, NID	P

Index	Column(s)
IND_XELEM	DID, NID, size

LONGTEXT_TAB

Column	Type	Nullable	Default	Comments
DID	**INTEGER**			**ID dokumentu**
NID	**INTEGER**			**node ID textu**
DEPTH	**INTEGER**			**hloubka zanoreni textu v**
PARENT_ID	INTEGER	Y		nid otce textu
NAME	CLOB	Y		textova hodnota

Index	Column(s)
IND_XLTEXT	DID, NID, PARENT_ID

TEXT_TAB

Column	Type	Nullable	Default	Comments
DID	**INTEGER**			**ID dokumentu**
NID	**INTEGER**			**node ID textu**
DEPTH	**INTEGER**			**hloubka zanoreni textu v**
PARENT_ID	INTEGER	Y		nid otce textu
NAME	VARCHAR2(256)	Y		textova hodnota

Key	Column(s)	Type
PK_XTEXT	DID, NID	P

Index	Column(s)
IND_XTEXT	DID, NID, PARENT_ID

ATTR_TAB

Column	Type	Nullable	Default	Comments
NAME	VARCHAR2(128)	Y		nazev atributu
DID	**INTEGER**			**ID dokumentu**
NID	**INTEGER**			**node ID atributu**
DEPTH	INTEGER	Y		hloubka zanoreni atributu v XM
PARENT_ID	INTEGER	Y		nid otce atributu
VALUE	VARCHAR2(1024)	Y		textova hodnota atributu

Key	Column(s)	Type
PK_XATTR	DID, NID	P

Index	Column(s)
IND_XATTR	DID, NID, PARENT_ID

Obrázek 9: XISS DB model

4.2.1.XISS.Import

Balík slouží pro nahrání XML souborů do DB instance *xmldb_xiss*

Balík obsahuje třídy

- *elem* – třída pro ukládání vlastností elementů
- *pomzasstruct* – pomocná „zásobníková“ třída pro odkládání odkazů na ještě ne úplně zpracovaných elementů.
- *XML_SAX_READER* – třída implementující SAX parser (a obslužné metody pro zpracování XML)

Třída *pomzasstruct* tvoří základ pro zásobník, na který se odkládají elementy s atributy do doby než jsou zpracovány všechny podelementy daného elementu. Zásobníky jsou realizovány pomocí typu *Vector.*

Funkce *main* třídy *Import* nejprve zavolá funkci *newconn*, která se pokusí pomocí JDBC připojit do DB instance (*xmldb_xiss*). Pokud tato instance neexistuje, je vytvořena. V embded módu umožňuje instance pouze single-user připojení, tedy, je-li někdo k dané instanci připojen (např. utilitou ij), připojení se nezdaří a program je ukončen. Následně se vytvoří dočasné soubory, kam se budou ukládat naparsovaná data. Každé tabulce odpovídá jeden soubor a každému souboru jeden zapisovač (*BufferedWriter*). Pro tabulku **ELEM_TAB** je použit soubor *elem.dat*, pro tabulku **ATTR_TAB** soubor *attr.dat*, pro tabulku **TEXT_TAB** soubor *text.dat*, pro tabulku **LONGTEXT_TAB** soubor *ltext.dat* a nakonec pro tabulku **DID_TAB** soubor *doc.dat*. Dále jsou postupně čteny soubory ze zadaného adresáře a pro každý soubor se vytvoří nová instance třídy XML_SAX_READER, čímž se provede vlastní parsing XML dokumentu.

V průběhu parsingu jsou rozlišovány 3 výše uvedené události a jsou nadefinovány následující metody:

- Při vyvolání funkce *startElement* se nejprve přečte vrchol zásobníku *pomzastruct* (kde máme uložený nadřízený element právě zpracovávaného elementu), zjistí se z něj potřebné informace (*child_id, parent_id*) a založí se nová instance třídy *elem* a tato instance se uloží na zásobník *pomzasstruct*. Poté se zpracují všechny atributy tohoto elementu a uloží se do souboru *attr.dat* pomocí volání funkce *store_attr.*
- Při vyvolání funkce *endElement* je nejprve obsloužena potencionální textová hodnota (která se nám nahromadila v proměnné *text*). Text je prvek podobný elementu, jen nemá žádné potomky ani bratry ani atributy. Text je pomocí funkce *store_text* uložen do souboru *text.dat*. Pokud je text delší než 200 znaků je v plné délce uložen do souboru ltext.dat. Následuje vyzvednutí vrcholu zásobníku *pomzasstruct*, dopočítání hodnoty *size* a propagace této hodnoty do nadřízeného elementu. Pak je element pomocí volání funkce *store_elem* uložen do souboru *elem.dat*.
- Při vyvolání funkce *characters* se pouze upravuje (concat) proměnná text. Důvod je ten, že texty, které jsou v XML dokumentu odděleny znakem LF (konec řádku), nejsou považovány rozhraním SAX za jeden text, ale za několik textů.

Po dokončení běhu tohoto parseru se vypíše statistika toho, jak dlouho trvalo zpracování souboru a počty elementů, atributů a textů.

Po zpracování všech XML souborů v adresáři (těch které mají příponu xml) se nejprve vymažou (drop) existující tabulky (z případného předešlého běhu importu) a znovu se vytvoří. Pak se

data z dočasných souborů bulk importem vloží do připravených tabulek. Na závěr jsou vytvořeny indexy a primární klíče. A nakonec se vypíše statistika importu všech XML souborů s informacemi, která část importu jak dlouho trvala.

4.2.2.XISS.xpath

Balík slouží pro dotazování XML dokumentů uložených v DB instanci *xmldb_xiss*

Balík obsahuje tyto třídy

- *Query* – slouží pro uložení fragmentů XPath dotazu (fragmenty jsou zde ty části, které vzniknou rozdělením XPath dotazu podle znaků '/' a '//').
- *Elem* – slouží pro parsing jednotlivých fragmentů a k výpočtu hodnot, které budou použity pro složení finálního SQL dotazu
- *xpath* – hlavní třída, která to vše řídí

Jediným vstupem procedury *main* třídy *xpath* je XPath dotaz v uvozovkách. Procedura xpath dotaz rozdělí na fragmenty nejprve podle '//' a tyto fragmenty dále podle '/'. Výsledné fragmenty jsou uloženy ve *Vectoru query* ve formě instancí třídy *Query*. Tento *vector* je následně zpracováván jako fronta fragmentů. Pro každý fragment je vytvořena nová instance třídy *Elem* a zavolán parser dotazu (funkce *parse_elem*).

Funkce *Parse_elem* takto získaný fragment nejprve rozdělí podle znaku '::'. Následně se kontroluje, zda fragment obsahuje znaky [] (a tedy výsledný SQL dotaz bude obsahovat dotaz na textovou hodnotu). Dále se zjišťuje, zda fragment obsahuje znak @ (a tedy výsledný SQL dotaz bude obsahovat dotaz na atribut, případně na atribut s hodnotou). Na základě zjištěných informací se upravují hodnoty *et, at, tt*. Tyto hodnoty představují počty tabulek, které se

zúčastní výsledného SQL dotazu. *Et* udává počet tabulek **ELEM_TAB**, *at* počet tabulek **ATTR_TAB** a *tt* počet tabulek **TEXT_TAB**. Funkce zároveň skládá where podmínka. Pokud bude třeba rozšířit množinu XPath dotazů, které metoda umí, upraví se příslušně tato funkce.

Skládání dílčích where podmínek podle již zjištěných informací mají na starosti 2 metody třídy *xpath* – *get_where_et* (skládá dílčí podmínku pro tabulky **ELEM_TAB** a **TEXT_TAB**) a *get_where_at* (skládá dílčí podmínku pro tabulku **ATTR_TAB**). Pro optimalizaci SQL dotazů je možno tyto funkce příslušně upravit.

Jsou-li již zpracovány všechny fragmenty (fronta *query* je prázdná), poskládá se předmět selectu, from část a where část. Tím je hotov výsledný SQL dotaz (select). Program se dále připojí do DB instance (*xmldb_xiss*) a pustí dotaz. Pak jsou do vyčerpání *resultSetu* čteny výsledky a vypisovány na obrazovku. Výsledný dotaz pouze vybere cílový prvek XPath dotazu. Je-li cílovým prvkem element, je třeba vypsat celý XML fragment, jehož je cílový element kořenem (root). To má za úkol funkce *print_results*.

Funkce *print_results* si otevře kurzory nad tabulkami **ELEM_TAB, TEXT_TAB, LONGTEXT_TAB** a **ATTR_TAB** a postupně je prochází a vypisuje strukturovaně výsledný XML fragment. Opět je použit zásobník pro odkládání elementů, které ještě nebyly ukončeny. Element je vypsán a odebrán ze zásobníku na základě hodnot *order* a *size.* Texty a atributy jsou vypisovány na základě hodnot *parent* a *attr_id* – element, kterému patří daný text či atribut, má *order* rovnu hodnotě *parent* textové hodnoty či atributu. Nakonec je vypsána doba běhu SQL dotazu a výpisu výsledků.
Na závěr uvádíme příklad XPath dotazu a jemu odpovídající SQL dotaz.

//f/descendant-or-self::node()/ancestor-or-self::node()

select distinct (select name from did_tab where did = et2.did) DOC_NAME, et2.did DOC_ID, et2.name, et2.nid, et2.size from elem_tab et0, elem_tab et1, elem_tab et2 where et0.nid>=0 and et0.name = 'f' and et1.DID = et0.DID and et1.nid >= et0.nid and (et1.nid+et1.size) <= (et0.nid +et0.size) and et2.DID = et1.DID and et2.nid <= et1.nid and (et2.nid+et2.size) >= (et1.nid +et1.size) order by DOC_ID, et2.did

4.3.Metoda XpathAccelerator

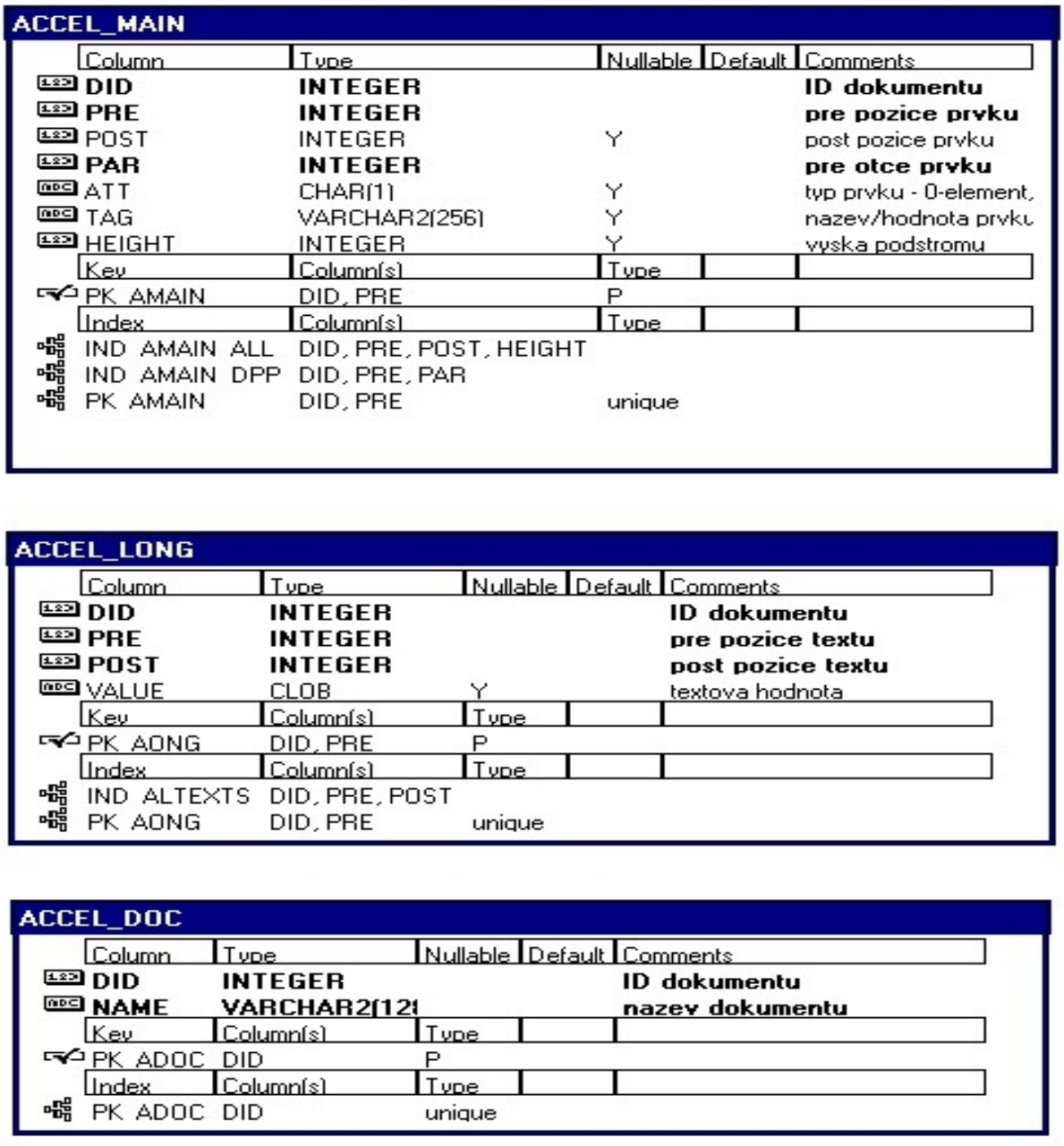
Základem této metody je opět specifické kódování elementů, atributů a textových elementů. Metoda je podobná metodě *XISS*. Na rozdíl od ní je zde použito tzv. *pre/post* okénko a prvky jsou uloženy pouze v jedné tabulce (**ACCEL_MAIN**). Pro *pre/post* okénko platí toto: prvek b je potomkem prvku a, jestliže *pre(a)< pre(b) && post(b)<post(a)*. Při přečtení počátku prvku je zvětšen atribut *pre*, při přečtení konce zase atribut *post*. Zároveň se počítá atribut *height*, který udává hloubku podstromu daného elementu. Atribut *pre* spolu s *did* (ID dokumentu) tvoří složený primární klíč tabulky **ACCEL_MAIN**. (Klíčem může být i dvojice *post, did* – klíče jsou si ekvivalentní). Dále se ještě počítá atribut *par* - otec prvku.

XML dokumenty jsou uloženy do následujícího DB modelu:

- V tabulce **ACCEL_DOC** jsou uloženy dvojice ID dokumentu (*did*) a název dokumentu (*name*).

- V tabulce **ACCEL_MAIN** jsou uloženy prvky jako n-tice (*name, did, pre, post, height, att*). Atribut *name* je název tagu. Atribut *att* slouží pro rozlišení typu prvku
  - 0-element
  - 1-atribut
  - 2-text
  - 3-hodnota atributu

- Texty, které jsou delší než 255 znaků, jsou ukládány do tabulky **ACCEL_LONG** jako tzv. LOB (Large Object), konkrétně CLOB (Character LOB). Vkládanou n-tici tvoří stejně jako v minulém případě (*did, pre, post, value*), kde *value* je textová hodnota elementu.



Obrázek 10: XPathAccelerator DB model

4.3.1.XPathAccelerator.Import

Balík slouží pro nahrání XML souborů do DB instance *xmldb_accel*

Balík obsahuje 3 třídy

- *elem* – třída pro ukládání vlastností prvků (do paměti)
- *dbmanip* – třída pro ukládání prvků (instancí třídy elem), dokumentů a dlouhých textů do pomocných importních souborů
- *XML_SAX_READER* – třída implementující SAX parser (a obslužné metody pro zpracování XML dat)

Funkce *main* třídy *Import* (po kontrole vstupních parametrů) vytvoří instanci třídy *dbmanip* a vytvoří připojení do DB instance *xmldb_accel*. Při vytváření instance třídy dbmanip jsou zároveň založeny pomocné importní soubory – *elem.dat a doc.dat* a jim odpovídající zapisovače (*BufferedWriters).* Pokud se připojení nezdařilo (např. někdo jiný je do instance připojen), je program ukončen. Následně jsou smazány (drop) existující tabulky (pomocí funkce *manip.dropold*) a vytvořeny nové, prázdné (pomocí funkce *manip.crenew*).

Následuje čtení souborů ze zadaného adresáře a pro každý soubor se vytvoří nová instance třídy *XML_SAX_READER*. O zbytek (tedy parsing a uložení dat do DB) se pak už stará pouze tato třída.

Stejně jako u metody XISS jsou předefinovány obslužné metody vyvolané parserem při událostech začátek/konec elementu a přečtení textové hodnoty

- Při vyvolání funkce *startElement* se vytvoří nová instance třídy *elem* – odpovídá novému elementu. Zároveň se navýší hodnota proměnné *pre*. Pokud má tag nějaké atributy, použije se na jejich

zpracování funkce *make_leaf*. Ta zpracovává nejen atributy, ale i textové hodnoty (viz dále). Nově vzniklý element je uložen na zásobník elementů (*elems*). To je nezbytné kvůli dopočítávání atributu *height*.

- Při vyvolání funkce *endElement* se nejprve obslouží textová hodnota. Jeho hodnota byla postupně ukládána do proměnné *text (*voláním metody *characters)*. Na textovou hodnotu se zavolá také funkce *make_leaf*. Pak se vyzvedne vrchol zásobníku *elems* a dopočítá se hodnota atributu *post*. Takto upravená instance třídy *elem* je uložena do pomocného importního souboru *elem.dat* (zavolá se funkce *db.insert()*) a výška tohoto elementu je propagována do otcovského elementu (na aktuální vrchol zásobníku *elems*).

- Při vyvolání funkce *characters* se pouze počítá (řetězí) proměnná text, tj. postupně se skládá z přečtených textových fragmentů mezi 2 tagy.

- Funkce m*ake_leaf* slouží ke zpracování (a ukládání) listů. Listy jsou v textové hodnoty nebo atributy (případně atributy s hodnotou). Funkce vrací hloubku uloženého stromu. V případě textové hodnoty je to vždy 1, v případě atributu bez hodnoty také 1, v případě atributu s hodnotou 2. Funkce vytvoří novou instanci třídy *elem* a vloží ji do do importního souboru *elem.dat* (pomocí funkce *db.insert()*).

Pro každý soubor je zároveň vypsána statistika toho, jak dlouho která operace trvala.

Po zpracování všech souborů v adresáři (tedy těch, které mají příponu xml) je zavolána výše zmiňovaná interní funkce pro tzv.

bulk import (hromadný import) - *SYSCS_UTIL.SYSCS_IMPORT_TABLE*. Funkce je postupně zavolána na soubory *elem.dat, doc.dat, text.dat*. Výsledkem volání této funkce jsou data v příslušných tabulkách (**ACCEL_MAIN**, **ACCEL_DOC** a **ACCEL_LONG**).

Po nahrání dat do tabulek dojde ještě k dodatečnému vytvoření primárních klíčů a indexů, které budou použity pro SQL dotazy při XPath dotazování. To se vykoná pomocí funkce *db.creind()*;

Nakonec je vypsán souhrnný čas importu.

4.3.2.XPathAccelerator.xpath

Balík slouží pro dotazování XML dokumentů uložených v DB instanci *xmldb_accel*

Balík obsahuje tyto třídy

- *Query* – slouží pro uložení fragmentů XPath dotazu (fragmenty jsou zde ty části, které vzniknou rozdělením XPath dotazu podle znaků '/' a '//').
- *Elem* – slouží pro parsing jednotlivých fragmentů a k výpočtu hodnot, které budou použity pro složení finálního SQL dotazu
- *xpath* – hlavní třída, která to vše řídí

Jediným vstupem procedury *main* třídy *xpath* je XPath dotaz v uvozovkách. Stejně jako v minulém případě procedura xpath dotaz rozdělí na jednotlivé fragmenty nejprve podle znaku '//' a tyto fragmenty dále podle '/'. Výsledné fragmenty jsou uloženy ve *vectoru query* ve formě instancí třídy *Query*. Tento *vector* je následně zpracováván jako fronta fragmentů. Pro každý fragment je vytvořena nová instance třídy *Elem* a zavolán parser (*parse_elem*).

Funkce *parse_elem* fragment nejprve rozdělí podle znaku '::'. Dále se kontroluje, zda fragment obsahuje znaky [], zda fragment obsahuje znak @ (a tedy výsledný SQL dotaz bude obsahovat atribut, případně atribut s hodnotou). Za základně zjištěných informací se upravuje hodnota *et* (počet tabulek, které se zúčastní SQL dotazu). A zároveň se (podle typu osy) skládá i where podmínka.

Pokud budeme chtít rozšířit množinu XPath dotazů, které metoda umí, upravíme příslušně tuto funkci (*parse_elem*).

Skládání where podmínky má starosti metoda *get_where_et*. Pro optimalizaci SQL dotazů pomocí změny pořadí where podmínek je možno tuto funkci příslušně upravit.

Po zpracování celé fronty fragmentů je poskládán předmět SQL dotazu, jeho „from“ část a „where“ část a tím pádem je k dispozici celý výsledný SQL dotaz. Program se dále připojí do DB instance (*xmldb_xiss*) a pustí dotaz. Pak jsou do vyčerpání *resultSetu* čteny výsledky a vypisovány na obrazovku. Pro každý cílový prvek je ještě zavolána funkce *print_results*.

Funkce *print_results* si otevře kurzory nad tabulkami **ACCEL_MAIN**, **ACCEL_LONG**. Postupně je prochází a vypisuje strukturovaně výsledný XML fragment. Opět se používá pomocného zásobníku pro odkládání neukončených elementů. Na základě hodnot pre a post se pozná, že element má být ukončen, je vypsána koncová značka a element odebrán ze zásobníku. Nakonec je vypsána doba běhu SQL dotazu.
Na závěr opět uvádíme příkras SQL dotazu, který vznikl překladem XPath dotazu.

//following-sibling::attribute()*

select distinct (select name from accel_doc where did = et3.did) DOC_NAME, et3.did DOC_ID, et3.tag t3, et3.pre p3, et3.post po3 from accel_main et0, accel_main et1, accel_main et2, accel_main et3 where et0.pre=0 and et0.att = '0' and et1.par = et0.pre and et1.DID = et0.DID and et1.att = '0' and et2.DID = et1.DID and et2.par = et1.par and et2.pre > et1.pre and et2.post > et1.post and et3.par = et2.pre and et3.att = '1' and et3.height=1 order by DOC_ID, et3.pre

4.4.PathStack

Základem této metody je opět specifické kódování elementů a textových elementů. Podobně jako u předchozí metody XPathAccelerator je zde použito kódování pomocí 3 atributů – *leftp, rightp a level*, a prvky jsou uloženy v jedné tabulce (**TWIG_MAIN**). Rozdíl oproti XPathAccelerator je v tom, že hodnota *leftp* a *rightp* je brána ze stejné číselné řady (sekvence) a navíc se tato sekvence nenuluje při parsingu dalšího XML dokumentu. I pro tuto metodu platí následující tvrzení: prvek d je potomkem prvku c, jestliže *leftp(c) < leftp(d) < rightp(d)<rightp(c)*. Atribut *level* udává hloubku zanoření elementu v XML stromu. Při přečtení počátku prvku je zvětšen atribut *leftp*, při přečtení konce zase atribut *rightp*. Zároveň počítáme atribut *level* podle úrovně zanoření v XML stromu. Atribut *leftp* tvoří primární klíč tabulky **TWIG_MAIN**.

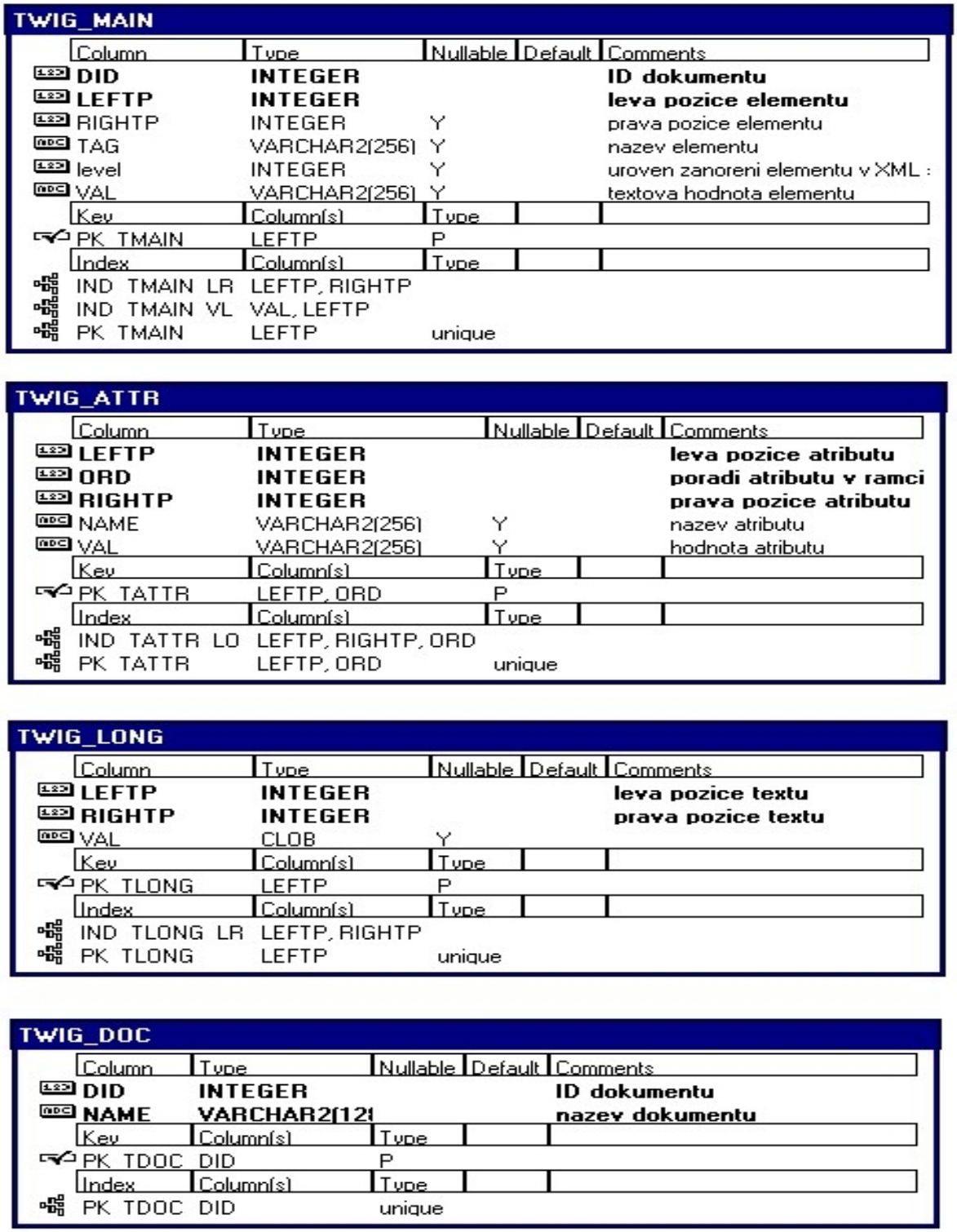
XML dokumenty jsou uloženy do následujícího DB modelu:

- V tabulce **TWIG_DOC** jsou uloženy dvojice ID dokumentu (*did*) a název dokumentu (*name*).

- V tabulce **TWIG_MAIN** jsou uloženy prvky jako n-tice (*name, did, leftp, rightp, tag, level, val*). Atribut *tag* je název tagu. Atribut *val* je textová hodnota elementu

- Texty, které jsou delší než 255 znaků jsou ukládány do tabulky

TWIG_LONG jako tzv. LOB (Large Object), konkrétně CLOB (Character LOB). Vkládanou n-tici tvoří (*leftp, rightp, val*), kde *val* je textová hodnota elementu.



TWIG_MAIN

Column	Type	Nullable	Default	Comments
DID	**INTEGER**			**ID dokumentu**
LEFTP	**INTEGER**			**leva pozice elementu**
RIGHTP	INTEGER	Y		prava pozice elementu
TAG	VARCHAR2(256)	Y		nazev elementu
level	INTEGER	Y		uroven zanoreni elementu v XML :
VAL	VARCHAR2(256)	Y		textova hodnota elementu

Key	Column(s)	Type
PK_TMAIN	LEFTP	P

Index	Column(s)	Type
IND_TMAIN_LR	LEFTP, RIGHTP	
IND_TMAIN_VL	VAL, LEFTP	
PK_TMAIN	LEFTP	unique

TWIG_ATTR

Column	Type	Nullable	Default	Comments
LEFTP	**INTEGER**			**leva pozice atributu**
ORD	**INTEGER**			**poradi atributu v ramci**
RIGHTP	**INTEGER**			**prava pozice atributu**
NAME	VARCHAR2(256)	Y		nazev atributu
VAL	VARCHAR2(256)	Y		hodnota atributu

Key	Column(s)	Type
PK_TATTR	LEFTP, ORD	P

Index	Column(s)	Type
IND_TATTR_LO	LEFTP, RIGHTP, ORD	
PK_TATTR	LEFTP, ORD	unique

TWIG_LONG

Column	Type	Nullable	Default	Comments
LEFTP	**INTEGER**			**leva pozice textu**
RIGHTP	**INTEGER**			**prava pozice textu**
VAL	CLOB	Y		

Key	Column(s)	Type
PK_TLONG	LEFTP	P

Index	Column(s)	Type
IND_TLONG_LR	LEFTP, RIGHTP	
PK_TLONG	LEFTP	unique

TWIG_DOC

Column	Type	Nullable	Default	Comments
DID	**INTEGER**			**ID dokumentu**
NAME	**VARCHAR2(12**			**nazev dokumentu**

Key	Column(s)	Type
PK_TDOC	DID	P

Index	Column(s)	Type
PK_TDOC	DID	unique

Obrázek 11: PathStack DB model

4.4.1.PathStack.Import

balík slouží pro nahrání XML souborů do DB instance *xmldb_twig*

Balík obsahuje 3 pomocné třídy

- *elem* – třída pro ukládání vlastností prvků (do paměti)
- *dbmanip* – třída pro ukládání prvků (instancí třídy elem) do pomocných importních souborů
- *XML_SAX_READER* – třída implementující SAX parser (a obslužné metody pro zpracování XML)

Funkce *main* třídy *Import* (po kontrole vstupních parametrů) vytvoří instanci třídy *dbmanip* a vytvoří připojení do DB instance. Při vytváření instance třídy dbmanip jsou zároveň založeny pomocné importní soubory – *elem.dat a doc.dat* a jim odpovídající zapisovače. Pokud se připojení nezdařilo, je program ukončen. Instance DB se v tomto případě jmenuje *„xmldb_twig“*. Funkce smaže (drop) existující tabulky (pomocí funkce *manip.dropold*) a vytvoří nové tabulky (pomocí funkce *manip.crenew*). Dále čte postupně soubory z daného adresáře a pro každý soubor vytvoří novou instance třídy *XML_SAX_READER*. O zbytek (tedy parsing a uložení dat do DB) se pak už stará pouze tato třída.

I zde jsou předefinovány obslužné funkce pro SAX parser:

- Při vyvolání funkce *startElement* se vytvoří nová instance třídy *elem* – odpovídá novému elementu. K tomuto elementu jsou uloženy i atributy s hodnotami. Nově vzniklý element je uložen na zásobník elementů (*elems*). To je nezbytné kvůli dopočítávání hodnoty atributu *rightp*.

- Při vyvolání funkce *endElement* se nejprve obslouží textová

hodnota. Jeho hodnotu je uložena v proměnné *text* – uloží se k aktuálně přečtenému elementu. Pak je vyzvednut vrchol zásobníku *elems* a dopočítán atribut *rightp*. Takto upravený *elem* je uložen nakonec souboru elem.dat (zavolá se *db.insert()*). Tato fukce (*db.insert*) ukládá i atributy do souboru *attr.dat* a *text.dat*.

- Při vyvolání funkce *characters* se pouze počítá proměnná text.

Pro každý soubor je vypsána statistika jak dlouho která operace trvala.

Po zpracování všech souborů v adresáři (tedy těch které mají příponu xml) je zavolána výše zmiňovaná interní funkce pro hromadný import dat do tabulky - *SYSCS_UTIL.SYSCS_IMPORT_TABLE*. Funkce je postupně zavolána na soubory *elem.dat* a *doc.dat*, *attr.dat* a *text.dat*. Výsledkem volání této funkce jsou data v příslušných tabulkách (**TWIG_MAIN TWIG_DOC**, **TWIG_ATTR**, **TWIG_TEXT**).

Následně jsou ještě vytvořeny primární klíče a indexy, které budou použity pro SQL dotazy. To se vykoná pomocí funkce *db.creind()*;

Nakonec se vypíše statistika celého importu.

4.4.2.PathStack.xpath

Balík slouží pro dotazování XML dokumentů uložených v DB instanci *xmldb_twig*

Balík obsahuje tyto třídy

- *Stack* – slouží pro ukládání pozic prvků dokumentu na zásobník. Na tomto zásobníku se počítají výsledné odpovědi
- *QueryNode* – slouží pro uložení značek XPath dotazu, jejich

zásobníků, proudu dat (datastreamu) a dalších vlastností
- *xpath* – hlavní třída

Tato dotazovací metoda se oproti minulým o něco liší. Rozdíl spočívá v tom, že je využito databáze pouze pro vrácení záznamu pro jednotlivé elementy. Spojování výsledků (tzv. „join“) těchto záznamů řeší algoritmus sám.

Jediným vstupem procedury *main* třídy *xpath* je XPath dotaz v uvozovkách. Xpath je opět rozdělen na fragmenty podle řetězců '//', '/', '[]', 'and'. Pro každý takto vznikly fragment je založena instance objektu *QueryNode*. Jednotlivé *QueryNode* jsou mezi sebou provázány vazbou otec-syn, tvoří *QueryTree*. Při založení nové instance *QueryNode* dojde mimo jiné k otevření kurzoru, který odpovídá názvu elementu (tagu) daného *QueryNoda* - přiřadíme k němu tzv. *datastream* (proud dat, který může být výsledkem XPath dotazu). Tedy otevřený kurzor tvoří *datastream* příslušný jistému *QueryNodu*.

Je-li vytvořený QT, program ho začne zpracovávat v cyklu. Cyklus končí, pokud všechny zásobníky na listech QT jsou prázdné. V každém kroku se nejprve vybere takový QN, který má minimální *leftp* na vrcholu svého zásobníku (minimum vybírá funkce *getMinSource*). Poté dojde k „vyčištění“ všech zásobníků od elementů, které nemohou být ve výsledku vypsány (nejsou potomky). U vybraného QN se vloží na vrchol zásobníku aktuální hodnoty *leftp, rightp a docid*. A přečte se další hodnota z datového proudu (cursoru, datastreamu), který přísluší tomuto QN. Pokud jako minimální QN je vybrán nějaký list, algoritmus se pokusí vypsat řešení dotazu (*printResults*) a odstraní vrchol zásobníku.

Pokud funkce *printResults* vypíše nějaké řešení, je ještě zavolána

funkce print_elem, která vypíše celý XML fragment obdobným způsobem popsaným v kapitole 4.3.2.

Po vypsání všech řešení dotazu je vypsána doba běhu.

5.Uživatelská dokumentace

5.1.Úvodem

Překlad, testování (import, dotazování) je dělán pomocí bash skriptů. Vyvíjeno a testováno na Debian GNU/Linux lenny/sid.

Potřebné GNU utility – sed, tee, diff, awk, tail, head, bc. Není k dispozici žádný configure script, protože se předpokládá, že všechny tyto utility jsou na každém linuxovém (správném) stroji.

Obsah DVD (adresáře)

- docs – tato práce, html dokumentace tabulek, javadoc
- xiss, accel, path - jednotlivé testované metody zpracování XML, každá obsahuje adresáře :
  - test - pro testování referenčních XPath dotazů oproti referenčním XML dokumentům
  - results – naměřené výsledky (ukládání XML dat a dotazování nad nimi)
- java - obsahuje instalaci JDK 1.5.11 a 1.6.6.
- derby - obsahuje instalaci derby DB ve verzi 10.2.0 a 10.4.0.
- lib - přídavné jar knihovny (zatím pouze xerces)

Pro ostré testování byla použita JDK 1.5.11, která se ukázala býti

rychlejší při SQL dotazování. Verze derby byla použita 10.4.0. Ta již umí bulk import tabulek, které obsahují sloupeček typu CLOB.

5.2.Jak nainstalovat

Stačí nakopírovat obsah přiloženého DVD na filesystém (např. do domovského adresáře) a v souboru .derby.profile (v rootu CD) je třeba změnit obsah proměnné INSTALL_PATH. Ta udává místo na filesystému, kde bylo DVD nakopírováno.

5.3.Jak nastavit prostředí shellu

Pro překlad zdrojových kódů či importu souborů nebo pouštění XPath dotazů musí být nejprve správně nastaveno prostředí (environment). K tomu slouží soubor .derby.profile. Měl by být spustitelný (pokud není, je potřeba pomocí příkazu chmod +x přidat právo execute). Soubor spustíme:

. .derby.profile

Tím se nastaví proměnné shellu - PATH, JAVA_HOME, DERBY_HOME a CLASSPATH. Pro otestování je možné použít příkaz env.

5.4.Jak kompilovat

Je třeba mít nastavené prostředí (environment). Podle toho, kterou metodou budeme pracovat, je třeba se pomocí příkazu *cd* vystavit do jejího adresáře. Pro překlad Importu je třeba zadat příkaz

javac Import.java

Pro překlad XPath dotazovače

javac xpath.java

Výsledkem překladu je, že se v adresáři vytvoří Java binární kód (classa). Co třída ve zdrojovém kódu, to jeden samostatný class soubor. Pro kompilaci obou balíků současně slouží skript compile.sh. Pouští se následovně:

./compile.sh

5.5.Jak pustit Import XML souborů

Opět je třeba mít nastavené prostředí, vystavíme se do příslušného adresáře. Import xml souborů do DB se pouští příkazem

java Import <path_to_xml_dir>

kde *path_to_xml_dir* je adresář s XML soubory (soubory v adresáři musí mít příponu XML nebo xml). V průběhu importu jsou zobrazovány statistiky, které jsou zároveň ukládány do adresáře results (viz dále). Při prvním spuštění Importu vznikne v adresáři podadresář xmldb_<metod_name> (podle toho, s jakou metodu pracujeme – xmldb_accel, xmldb_xiss, xmldb_twig). Tento adresář bude obsahovat všechny DB objekty včetně databazávých souborů, indexů apod. Představuje jednu databázovou instanci RDBMS Derby.

5.6.Jak pustit Xpath dotaz

Opět je třeba mít nastavené prostředí, vystavíme se do adresáře

příslušné metody. Pokud ještě neexistuje podadresář xmldb_<metod_name>, je nejprve třeba naimportovat nějaká XML data (howto run Import). Pro spuštění XPath dotazu se zadá příkaz

java xpath "<*xpath dotaz*>"

Následně je zobrazen výsledek. Výsledkem je formátovaný výstup o 3 sloupcích – název souboru, výsledek XPath dotazu a interní identifikátory elementů (atributů, textů). Na další řádce pod názvem souboru je vypsán XML fragment cílového elementu.

5.7.Jak na testování

Pro kontrolu zavlečenosti chyb při úpravách kódu je dobré si ověřit, že Import i xpath vracejí stále stejné výsledky. K tomu slouží adresář test. Obsahuje adresář xml (s několika malými vzorovými xml soubory), soubor xpath.example (vybrané vzorové testovací xpath dotazy) a referenční soubor reffile (ten obsahuje výsledky XPath dotazů). Test se pustí následujícím příkazem:

./make_test.sh

Ten nejprve naimportuje referenční XML data do DB a poté spustí testovací sadu XPath dotazů, výsledky si poznamená do pomocného souboru a nakonec si udělá rozdíl (diff) referenčního souboru a pomocného souboru. Pokud jsou oba soubory stejné, je celkem velká šance, že nedošlo k žádné zavlečené chybě. Pokud jsou soubory stejné, odpovědí je „Seems OK“, jinak výstup pomocného programu (utility) diff (see man diff).

5.8.Opakování pokusů

Pro přesnější měření časů (a do budoucna možná i pro statistická data) je dobré, pokud stejný pokus opakujeme vícekrát. Pokusem zde je myšlen jeden Import adresáře (v případě importu) a jedna sada XPath dotazů (v případě dotazů) do připravené DB. K těmto účelům jsou připraveny skripty (v adresáři příslušné metody) *make_import* a *make_xpath*.

./make_import.sh <xml_dir> [count]

kde *xml_dir* je cesta k adresáři s XML dokumenty a count je počet opakování pokusu. Pokud není hodnota count zadána, bere se hodnota 1. Na základě tohoto příkazu je count-krát proveden import XML souborů do DB a výsledek importu je zapsán do adresáře results do souboru *<xml_dir>.import.<iteration>*. Na závěr je ze všech pokusů vypočítána průměrná doba běhu importu.

Pro hromadné dotazování je třeba vytvořit soubor s XPath dotazy (nebo použít přiložený xpath.example pro testování). Struktura souboru s xpath dotazy je následující: každý XPath dotaz začíná na nové řádce a je „obalen“ znakem uvozovka (např. "//@*"). Sadu XPath dotazů lze pustit proti DB pomocí příkazu

./make_xpath.sh <xpath_file> [count]

kde *xpath_file* je cesta k souboru, který obsahuje sadu XPath dotazů. Pokud není count zadáno, bere se hodnota 1. Na základě tohoto příkazu jsou count-krát provedeny všechny XPath dotazy ze souboru. Výstupem jsou časové údaje běhu dotazů. A výsledky dotazů jsou taktéž uloženy do adresáře results do souboru <xml_dir>.query.<pořadí dotazu v souboru>.<iterace>.

5.9.Hromadné operace

Pro větší „pohodlí“ jsou dále připraveny jednoduché skripty, které dělají nějakou akci pro všechny implementované metody. Jsou umístěny v kořenovém adresáři přiloženého CD. Pro překlad všech zdrojových kódů lze použít příkaz:

./compile_all.sh

Pro postupný import XML souborů pomocí všech metod je možné použít příkaz

./make_import_all.sh <xml_dir> [count]

kde *xml_dir* je adresář s XML dokumenty a count je počet opakování pokusu. Pokud není count zadáno, bere se hodnota 1. Průběh importu je vypisován na obrazovku a zároveň jsou výsledky ukládány do adresářů results příslušných metod úplně stejně jako v případě „nehromadných“ operací.

Pro spuštění sady XPath dotazů oproti všem databázím (xmldb_xiss, xmldb_accel, xmldb_twig) je možno použít příkaz

./make_xpath_all.sh <xpath_file> [count]

kde *xpath_file* je cesta k souboru, který obsahuje sadu XPath dotazů. Pokud není count zadáno, bere se hodnota 1. Na základě tohoto příkazu jsou count-krát provedeny všechny XPath dotazy ze souboru oproti všem databázím. Výstupem jsou časové údaje běhu dotazů a zároveň jsou výsledky ukládány do adresářů results příslušných metod

úplně stejně jako v případě „nehromadných“ operací.

5.10.Příklady XPath dotazů

Seznam je možné nalézt v adresáři test v souboru *xpath.example*
Zde uvedeme typy XPath dotazů, které metody umí.

5.11.Výsledky pokusů - adresáře results

Zde jsou výsledky měření importu a dotazu. Názvy souborů odpovídají názvům adresářů na DVD s testovacími daty, přípona .import udává, že se jednalo o import, .query, že o dotaz. Poslední přípona (číslo) udává pořadové číslo pokusu. v případe XPath dotazů má soubor query 2 číselné přípony – první udává pořadové číslo XPath dotazu v souboru dotazů, druhé pak pořadové číslo pokusu.

5.12.Utilita ij

Derby podobně jako většina RDBMS (snad kromě M$SQL) obsahuje příkazovou utilitu ij. Jedná se o nástroj na spouštění SQL skriptů a spouštění (tzv. adhoc) SQL dotazů. Je třeba mít nastavené prostředí. Utilita se spouští příkazem:

ij

Všechny příkazy zadávané v ij musí být ukončeny znakem středník. Pro připojení do konkrétní DB instance je nutné (už v prostředí ij) zadat

connect 'jdbc:derby:<instaname>';

Upozornění: pro připojení k DB instanci musí být ij spuštěno v adresáři, který obsahuje adresář s názvem databáze (a s DB objekty)

Pro vypsání tabulek z DB katalogu, stačí zadat příkaz

show tables;

Pro vypsání struktury dané tabulky slouží příkaz

describe <tabname>;

Konkrétní příklady

XISS

- connect 'jdbc:derby:xmldb_xiss';
- select substr(name,1,20), nid, size, depth, parent_id, child_id, prev_id, attr_id from ELEM_TAB;

XpathAccelerator

- connect 'jdbc:derby:xmldb_accel';
- select substr(tag,1,20), pre, post, par, att, height from accel_main;

PathStack

- connect 'jdbc:derby:xmldb_twig';
- select substr(tag,1,20), did, leftp, rightp, level, substr(val,1,20) from twig_main;

6.Výsledky měření a porovnání

Měření výsledků probíhalo na PC s procesorem AMD Sempron(tm) 2500+, taktovací frekvence 1435.247 MHz, vyrovnávací paměť procesoru 256KB, RAM 512MB (DDR2, 333MHz). Databázové soubory (datafiles) byly uloženy na RAID1 poli (2x250 HDD Seagate, 250GB, vyrovnávací paměť 8MB, rychlost vystavení hlaviček 9ms, 7200 otáček za minutu, ultra ATA rozhraní). Přesnějších výsledků je možno dosáhnout opakováním pokusů, jak je popsáno v uživatelské dokumentaci, kapitola 5.8. Testovaná data jsou na přiloženém disku. Naměřené výsledky je též možné nalézt na přiloženém disku v adresářích results.

Počet dokumentů	301
Počet KB	111547
Počet elementů	435921
Počet textových hodnot	402321
Počet atributů	1829

Tabulka 2: Arthurs Classic Novels statistiky

Metoda	XISS	XpathAccel	PathStack
Doba importu (s)	151	51	52
Doba dotazu /* (s)	1162	133	1023
Doba dotazu //argsd (s)	3	6	4

Tabulka 3: Arthur Classic Novels naměřené hodnoty

Počet dokumentů	2
Počet KB	8506
Počet elementů	57677
Počet textových hodnot	32359
Počet atributů	55421

Tabulka 4: Bible v XML statistiky

Metoda	XISS	XpathAccel	PathStack
Doba importu (s)	26	34	23
Doba dotazu /* (s)	21	18	21
Doba dotazu //argsd (s)	1	2	1

Tabulka 5: Bible v XML naměřené hodnoty

Počet dokumentů	41
Počet KB	14547
Počet elementů	227948
Počet textových hodnot	0
Počet atributů	193781

Tabulka 6: Jan Bosák statistiky

Metoda	XISS	XpathAccel	PathStack
Doba importu (s)	49	52	43
Doba dotazu /* (s)	120	53	99
Doba dotazu //argsd (s)	2	3	2

Tabulka 7: Jan Bosák naměřené hodnoty

Počet dokumentů	1
Počet KB	593314
Počet elementů	8407051
Počet textových hodnot	6714952
Počet atributů	0

Tabulka 8: DNA v XML statistiky

Metoda	XISS	XpathAccel	PathStack
Doba importu (s)	1564	2178	1249
Doba dotazu /* (s)	2452	1081	2544
Doba dotazu //argsd (s)	42	65	38

Tabulka 9: DNA v XML naměřené hodnoty

Počet dokumentů	1210122
Počet KB	1298243
Počet elementů	23132542
Počet textových hodnot	21820832
Počet atributů	33660745

Tabulka 10: FreeDB statistiky

Metoda	XISS	XpathAccel	PathStack
Doba importu (s)	42132	45640	40468

Tabulka 11: FreeDB naměřené hodnoty

Testy ukázaly, že metody lze použít nejen pro zpracování malých XML dokumentů, kde se metody chovají rozumě (celkem rychle), ale i pro zpracování rozsáhlých dat.

Co se týká importní části, tak jsou metody schopné zpracovat velmi rozsáhlé XML dokumenty nebo i velmi mnoho menších dokumentů. Při zpracování mnoha (1000000) dokumentů však už dochází k velkému zpomalení.

U XPath dotazů musíme konstatovat, že rychlost není úplně optimální, jistě by se dalo dosáhnout vyšší rychlosti jinými indexovacími technikami. Dalším limitem pro XPath dotaz je počet os, ze kterých se dotaz skládá. Při velkém počtu elementů a větším počtu os (3 a více), kdy dochází ke spojování více tabulek, odpověď trvá velmi dlouho.

Velké dokumenty také občas působí problémy JVM. Při testech se několikrát stalo, že JVM (Java Virtual Machine) spadla na internal error. Opakovaný pokus pak proběhl. Ale to není problém metod, ale JVM (a pravděpodobně způsobu, jakým spravuje paměť).

Z implementovaných metod nejlepší časy vykazovala metoda XpathAccelerator při XPath dotazování, naopak nejdéle trval import dat pomocí této metody. To je dáno skladbou indexů, déle se vytvářejí, ale lépe se dotazují. Import u XISS metody trvá nejdéle z toho důvodu, že je třeba importovat větší množství tabulek. PathStack naopak vykazoval nejlepší časy při hledání neexistujícího elementu.

Podstatnou část času importu zabírá vytváření indexů (u menšího množství dokumentů i více než polovinu). Pokud bychom na vytváření indexů rezignovali, urychlíme importní část, ale zpomalíme XPath dotazování.

6.1.Lehké porovnání s Oracle 10gR2

Na závěr jsme prováděly experimenty s DB Oracle verze 10gR2. Pokud do DB vložíme (např. pomocí dbms_lob.loadFromFile) XML soubor, jehož textový řetězec je delší než 64KB, tak při xpath dotazu oracle ohlásí výjimku.

SQL> select extractValue(xml,'/') from did_tab;*

ERROR at line 1:

ORA-31167: XML nodes over 64K in size cannot be inserted

Námi implementované metody tento problém nemají.

Dalším pokusem bylo nahrání souboru DNA.xml (Tabulka 11). Load do DB trval sice jen 30 vteřin, ale XPath dotaz na kořenový element po více než 5 hodinách (na stroji s 2GB paměti, 2xCPU 1800MHz) spadl na výjimku.

20:31:15 SQL> select extractValue(xml,'/') from did_tab;*

ERROR at line 1:

ORA-04030: out of process memory when trying to allocate 1000 bytes (qmxlu subheap,qmemNextBuf:alloc)

Tedy ukazuje se, že v mezních případech (obrovské XML dokumenty) jsou implementované metody lepší než komerční software.

7.Závěr

Konstatujeme, že cíle práce bylo dosaženo. Provedli jsme implementaci metod pro ukládání (rozsáhlých) XML dokumentů do RDBMS a metody jsme srovnali na kolekcích reálných dat. Implementované metody i přes svoje nedostatky ukázaly svoji použitelnost, zejména co se týká rozsáhlých XML dokumentů. Obzvláště importní část, kde bylo s výhodou použito bulk importu dat je poměrně rychlá. V porovnání s komerčním produktem firmy Oracle metody umí zpracovat a dotazovat rozsáhlejší XML dokumenty. Nicméně metody nejsou samozřejmě ideální, možné rozšíření vidíme zejména v implementaci dalších typů Xpath dotazů (např. dotazy na počty podelementů, pozice elementů, u metody PathStack pak i zbývající osy). Co se týká rychlosti XPath dotazů, zde jsou také určitě rezervy. Otázkou zůstává, zda je možné vyšší rychlosti dosáhnout na databázovém frontendu další optimalizací používaných indexů, nebo bude nutné (pro lepší výsledky) se posunout o úroveň níže na databázový backend a zde případně implementovat speciální indexové struktury, které budou lépe vyhovovat uvedeným metodám.

Seznam obrázků

Seznam tabulek

Seznam literatury

1: W3C Recommendation , Extensible Markup Language (XML) 1.1 (Second Edition), 2006, http://www.w3.org/TR/2006/REC-xml11-20060816/
2: ODA, Standard Generalized Markup Language, 1986, http://www.w3.org/MarkUp/SGML
3: W3C Working Draft, Web Services Architecture, 2003, http://www.w3.org/TR/2003/WD-ws-arch-20030808/
4: W3C Recommendation (Second Edition), Simple Object Access Protocol (SOAP), 2007, http://www.w3.org/TR/soap/
5: Kasper B. Graversen, CSV specification, 2007, http://supercsv.sourceforge.net/csvSpecification.html
6: Federal InformationProcessing Standards Publication 127-2, Database Language SQL (FIPS PUB 127-2), 1992, http://www.itl.nist.gov/fipspubs/fip127-2.htm
7: Oracle, Oracle home page, http://www.oracle.com/
8: Philip J Harding, Quanzhong Li, Bongki Moon, XISS/R: XML Indexing and Storage System Using RDBMS, September 2001
9: Torsten Grust, Accelerating XPath Location Steps,
10: Nicolas Bruno, Nick Koudas, Divesh Sristava, Holistig Twig Joins: Optimal XML Pattern Matching
11: W3C Recommendation, XML Path Language (XPath), November 1999, http://www.w3.org/TR/xpath
12: SAX (Simple API for XML), April 2004, http://www.saxproject.org/
13: Apache, Xerces2 Java Parser, http://xerces.apache.org/xerces2-j/
14: Apache, The Apache DB Project, http://db.apache.org/derby/
15: W3C, Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1, http://www.w3.org/XML/1998/06/xmlspec-report.htm
16: Network Working Group, The Base16, Base32, and Base64 Data Encodings (RFC 3548), 2003
17: XML namespaces, http://www.w3.org/TR/REC-xml-names/

18: XML Schema, http://www.w3.org/XML/Schema
19: Document Object Model (DOM), http://www.w3.org/DOM/
20: Sun, JDBC Specification
http://java.sun.com/products/jdbc/download.html
21: IBM, IBM Cloudscape RDBMS,
http://www-306.ibm.com/software/data/cloudscape/
22: Javacc, https://javacc.dev.java.net
23: Derby On Disk Page Format,
http://db.apache.org/derby/papers/pageformats.html
24: Apache Derby: Source Code,
http://db.apache.org/derby/dev/derby_source.html
25: Xalan, http://xml.apache.org/xalan-j/
26: RNDr. Irena Mlýnková, PhD., XML Schema a jeho implementace v prostředí relační databáze
27: Mlýnková, Pokorný, Richta, Toman K, Toman V.: Technologie XML
28: Ronald Bourret, Mapping DTDs to Databases, 2001,
http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html
29: Irena Mlýnková, Přehled a možná vylepšení technik pro zpracování XML dokumentů v (O)RDBMS
30: Various, Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, 2003
31: Vadim Tropashko, Nested Intervals Tree Encoding with Continued Fractions
32: Nan Tang, Jeffrey Xu Yu and Kam-Fai Wong, Fast XML Structural Join Algorithms by Partitioning, 2008
33: Timo Böhme, Erhard Rahm, Supporting Efficient Streaming and Insertion of XML Data in RDBMS
34: Irena Mlýnková, Jaroslav Pokorný, Information Systems Development,
35: Denilson Barbosa Juliana, Freire Alberto, O. Mendelzon, Designing Information-Preserving Mapping Schemes for XML, 2005
36: Michal Krátký, Jaroslav Pokorný, Václav Snášel, Indexing XML Data with UB-trees, 2002

37: Yunyao Li, Cong Yu, H. V. Jagadish, Schema-Free XQuery, 2004
38: Yuqing Wu, Jignesh M. Patel, H. V. Jagadish, Structural Join Order Selection for XML Query Optimization
39: Fusion XML SAX parser, http://www.unicoi.com/fusion_web/fusion_xml_sax_microparser.htm

Příloha A

Obsah přiloženého disku

- docs – tato práce, html dokumentace tabulek, javadoc
- xiss, accel, path - jednotlivé testované metody zpracování XML, každá obsahuje adresáře :
  - test - pro testování referenčních XPath dotazů oproti referenčním XML dokumentům
  - results – naměřené výsledky (ukládání XML dat a dotazování nad nimi)
- java - obsahuje instalaci JDK 1.5.11 a 1.6.6.
- derby - obsahuje instalaci derby DB ve verzi 10.2.0 a 10.4.0.
- lib - přídavné jar knihovny (zatím pouze xerces)
- test_data – adresář s testovacími XML dokumenty