VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

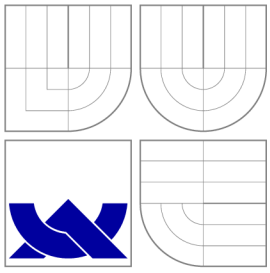
VÝVOJ SQL/XML FUNKCIONALITY V DATABÁZI POSTGRESQL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

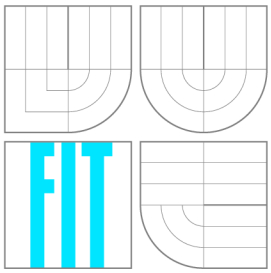
AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ POSPÍŠIL

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VÝVOJ SQL/XML FUNKCIONALITY V DATABÁZI POSTGRESQL

DEVELOPMENT OF SQL/XML FUNCTIONALITY IN POSTGRESQL DATABASE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. TOMÁŠ POSPÍŠIL
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR CHMELAŘ
SUPERVISOR

BRNO 2011

Abstrakt

Hlavním cílem této diplomové práce je navrhnout a implementovat chybějící XML funkcionalitu v databázovém systému PostgreSQL. Druhá kapitola práce teoreticky rozebírá XML paradigma spolu se souvisejícími technologiemi jako jsou Xpath nebo XQuery. Třetí kapitola diskutuje ISO SQL normy a následně rozebírá aktuální úroveň implementace v XML nativních databázích oproti tradičním relačním databázím. Poslední část se zaměřuje na rozdíly v přístupu a navrhuje řešení implementace XML API do PostgreSQL, které validuje XML dokumenty vůči XSD, DTD a RelaxNG schématům. Dalším bodem je přehled indexačních technik XML a návrh nového indexu pomocí GiST.

Abstract

The aim of this thesis is to propose a way to implement the missing XML functionality for the database system PostgreSQL. The second chapter discusses the theoretical paradigm with an XML-related technologies like Xpath or XQuery. The third chapter discusses the ISO SQL standards and describes the current level of implementation of native XML databases, versus traditional relational databases. The last part focuses on different approaches and it proposes a solution to implement the XML API to PostgreSQL, which validates XML documents against XSD, DTD and RelaxNG schemes. Next point is focused on XML indexing techniques and proposal of new index based on GiST.

Klíčová slova

PostgreSQL, XML, LibXML, RelaxNG, DTD, XSD, SQL/XML, ISO SQL:2003, ISO SQL:2008, ISO SQL:20nn, GiST, GIN, ltree, XPath, XQuery, ORDBMS

Keywords

PostgreSQL, XML, LibXML, RelaxNG, DTD, XSD, SQL/XML, ISO SQL:2003, ISO SQL:2008, ISO SQL:20nn, GiST, GIN, ltree, XPath, XQuery, ORDBMS

Citace

Tomáš Pospíšil: Vývoj SQL/XML funkcionality v databázi PostgreSQL, diplomová práce, Brno, FIT VUT v Brně, 2011

Vývoj SQL/XML funkcionality v databázi PostgreSQL

Prohlášení

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

.....
Tomáš Pospíšil
23. května 2011

Poděkování

Chtěl bych poděkovat Ing. Petru Chmelařovi za poskytnuté rady a odborné vedení, jež byly cenným vodítkem při realizaci tohoto projektu.

Obsah

Kapitola 1

Úvod

Se vzrůstajícím množstvím XML dokumentů vyvstává otázka jejich persistence a hromadného zpracování pro potřeby dalšího použití. Jelikož se typicky jedná o textové soubory, tak je nasnadě jejich perzistence do struktury souborového systému. Takto spravované soubory ale přináší z pohledu údržby nemalé komplikace, tudíž se stejně jako u ORDBMS (Object-Relational Database Management System) přešlo na centralizované úložiště dokumentů.

Vedle databázových systémů, které přidaly podporu pro nový datový typ XML, vznikly i nativní XML databázové systémy, které mimo centralizovaný způsob uložení XML dokumentů řeší mapování dat mezi typicky relačním způsobem práce s daty v ORDBMS a dokumentovým přístupem, jež je vlastní XML. Meta jazyk XML, jakožto další člen do rodiny univerzálních značkovacích jazyků SGML, byl navrhnut s ohledem na snadnou čitelnosti lidmi. Tato jeho vlastnost je ovšem vykoupena velikostí výsledného dokumentu a tím zvýšenými časovými a datovými nároky při strojovém zpracování. Další komplikací je zanesení hierarchizace XML dokumentu, kterou Coddův relační model nereflektuje a tudíž je třeba pracovat s XML dokumenty rozdílným způsobem oproti ostatním atomickým datovým typům. Tento problém eskaluje při návrhu indexačních technik, které musí být obrazem jak strukturální informace, tak i adresace obsahové části.

Primárním účelem XML je snadná výměna dat mezi multiplatformními informačními systémy. Nejlépe se pro tyto účely hodí datově zaměřené XML dokumenty, které naproti dokumentově zaměřenému paradigmatu přináší výhodu v podobě snadné kontroly dat pomocí validačních schémat. Z historického důvodu má zde důležité místo DTD, dále XSD a nejnovější RelaxNG, pomocí nichž se XML dokumenty validují a jedná se tedy i o integritní omezení. Validace spolu s modifikujícími operacemi nad XML dokumentem začaly společnosti vyvíjející databázové systémy řešit vlastním způsobem. Některá z těchto řešení se staly de facto standardy, ostatní pokrývá norma ISO SQL:2003 a její následníci včetně nejnovější pracovní revize normy ISO SQL:20nn.

Tato práce teoreticky rozebírá vlastnosti SQL/XML částí normy ISO:SQL, které definují operace nad XML dokumenty pomocí jazyka SQL. Dále provádí rešerši těchto vlastností u největších komerčních databázových tvůrců jako je Oracle, IBM, Microsoft, ale i open-source poskytovatelů PosgreSQL a MySQL. V přehledu je kladen důraz na způsob indexace, který je použit u výše jmenovaných databázových systému nad XML datovým typem.

Hlavní část práce se soustředí na návrh a následnou implementaci chybějícího rozšíření XML funkcionality do databázového systému PostgreSQL 9.0.3. V prvé řadě se jedná o přidání XML validačních funkcí podporujících DTD, XSD, RelaxNG validační schémata. Druhým hlavním cílem je navrhnout způsob indexace XML datového typu s využitím existujících procedur a vytvořit rozhraní pro následnou implementaci XQuery.

Současný stav v PostgreSQL umožňuje vytvořit index pouze při definici XPath dotazu, kdy se výsledek ukládá pro pozdější použití, není tedy možné obecně akcelerovat dotazování nad XML daty pro všechny dopředu neznámé dotazy. Tato komplexní otázka indexace XML datového typu byla přijata do projektu Google Summer of Code 2011, kde budu pod dohledem mentora (Gregory Star, MIT) implementovat zvolený index s ohledem na specifika daná PostgreSQL.

1.1 Cíl práce

Jelikož některé podstatné operace nad XML v PostgreSQL stále nejsou implementovány, tak je cílem této práce navrhnout a doplnit tuto funkcionalitu. Jmenovitě se jedná o skupinu validačních funkcí a návrh indexační struktury pro datový typ XML. Důraz je kladen na implementaci splňující normu ISO SQL:2008, případně aktuální pracovní draft normy ISO SQL:20nn. Chování, která nejsou definována normou, jsou determinována aktuální implementací v Oracle 11g, případně IBM DB2. Implementačním jazykem je ISO C 99 a dále procedurální rozšíření SQL používané v PostgreSQL PL/pgSQL.

Struktura práce

Obsah práce navazuje na předchozí semestrální projekt, jehož součástí byly kapitoly: 2, 3 a 5, které byly pro potřebu diplomové práce adekvátně rozšířeny. Práce je rozdělena následujícím způsobem:

- Druhá kapitola (2) pojednává o historii XML jazyka, jeho hlavních rysech důležitých při zpracování. Dále teoreticky rozebírá problematiku zpracování a validace XML dokumentů pomocí nejčastěji používanými schématy DTD, XSD a RelaxNG. Stručně popisuje technologii XPath a XQuery spolu s jazyky, které využívají.

- Třetí kapitola (3) se zaměřuje na vazbu XML - SQL v prostředí relačních (případně objektově-relačních) databází. Diskutuje definice použité v ISO normách, stejně tak i aktuální pracovní draft normy SQL:20nn.

- Čtvrtá kapitola (4) popisuje základní metody indexace XML dokumentů. Rozděluje je do několika kategorií dle použitého datového modelu a následně popisuje typické zástupce těchto kategorií. V neposlední řadě je uveden návrh indexu XLABEL, který sice nebyl nikdy implementován, ale jedná se o případovou studii prezentovánu u předchozí verze PostgreSQL 8.3 a některé z charakteristických rysů jsou použity i ve vlastním řešení indexu FastX.

- Pátá kapitola (5) reflektuje aktuální míru implementace XML podpory u nejrozšířenějších komerčních i open-source zástupců objektově-relačního přístupu k perzistenci dat. Hlavní důraz je kladen na PostgreSQL, kde jsou popisovány použité implementační detaily.

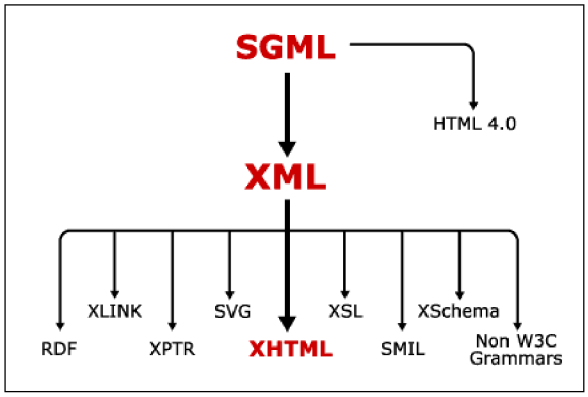
- Šestá kapitola (6) rozebírá návrh implementace pro určenou chybějící funkcionalitu v PostgreSQL 9.03. Popisuje problémy, které vznikly v průběhu vlastní práce a řešení, které bylo zvoleno.

Kapitola 2

Vlastnosti XML

2.1 Historie vzniku XML jazyka

Jazyk XML (Extensible Markup Language) [18] patří do skupiny značkovacích jazyků, které formálně popisují strukturu dokumentu. XML patří do početné rodiny jazyků, které jsou definovány ze SGML (Standard Generalized Markup Language) ISO normou 8879 z roku 1986 viz [39], tak jak ilustruje obrázek 2.1 do níž mimo jazyk XML patří také HTML, XHTML, SVG a jiné.



Obrázek 2.1: Rodina SGML jazyků. Převzato z [10].

XML dokument viz ukázka kódu 2.1 přidává k obsahu ještě další metadata, která hierarchizují strukturu souboru. Zároveň sémantiku těchto metadat nedefinuje a je tak pouze na konzumentovi daného dokumentu, jak jej interpretuje. XML samo o sobě nedefinuje ani seznam všech použitelných značek v dokumentech, jak to dělá například HTML. Jedná se tedy o meta-jazyk pro popis značkovacích jazyků, kdy pouze definuje syntaktickou strukturu značek a vazbu mezi nimi. Hlavním účelem vzniku XML dle [18] bylo:

- usnadnit komunikaci programů přes internet
- použít pouze nezbytnou syntaxi SGML ale zachovat zpětnou kompatibilitu
- kód dokumentu by měl zůstat lidem čitelný

XML je tedy definováno jako profil SGML (Standard Generalized Markup Language) a organizace W3C (World Wide Web Consortium) [45] standardizovala v roce 1996 jeho základní

podobu spolu a následně jeho nejnovější verzi XML 1.0 (pátá edice) která byla schválena 26.listopadu 2008. Dále existuje standard XML 1.1 (druhá edice) [17], který byl přijat 29.listopadu 2006.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<person id="1" career="inventor">
  <name title="Ingenieur">
     <surname>Cimrman</surname>
     <firstname>Jaroslav</firstname>
  </name>
  <address correspondent="true">
     <housenumber>10</housenumber>
     <street>Mariahilfestrasse</street>
     <town>Wien</town>
     <postcode>LAX 2YZ</postcode>
     <country main="false">AT</country>
  </address>
</person>
```

Kód 2.1: Základní struktura XML dokumentu. Pro přehlednost bez jmenného prostoru.

2.2 XML jmenné prostory

XML jmenné prostory [16] slouží pro jednoznačné určení definice uzlu. Pokud existují elementy se stejným názvem, tak pomocí jmenného prostoru lze jednoznačně určit do kterého oboru definic patří. V hlavičce dokumentu lze definovat výčtem použité jmenné prostory a přiřadit k nim odpovídající URI (Uniform Resource Identifier). Tímto způsobem se definují odkazy na validační schémata, která se mohou využít při kontrole struktury dokumentu. Jelikož se jedná o URI, tak lze dokument adresovat jak na lokálním úložišti tak i na síti, čímž vzniká potřeba stažení těchto definic pro potřebu validace. Tento problém je nastíněn v implementační části (6), jelikož z pohledu bezpečnosti není vhodné stahovat nezabezpečený obsah dokumentu v průběhu validace.

Každý XML dokument může obsahovat několik jmenných prostorů, stejně tak nemusí obsahovat žádný. Právě jeden jmenný prostor může být vybrán jako výchozí. Zápis elementů je poté ve formátu `xmlns:prefix`.

Norma XML 1.1 umožňuje místo URI použití IRI (Internationalized Resource Identifier), která dovoluje v adrese použít i specifických symbolů pro jednotlivé znakové sady, případně symboly mimo ASCII tabulku, jako je např. π. Většina dnešních XML parserů ovšem tento formát nepoužívá a stále pracuje s URI, případně pouze s URL.

Jelikož jmenné prostory nebyly původní součástí XML 1.0 definice, bylo třeba ve verzi XML 1.1 přidat zpětně kompatibilní rozšíření. Proto starší XML 1.0 parsery pracují se jmenným prostorem tak, jako by byl vlastní součástí názvu elementu. Je to zapříčiněno tím, že dvojtečka je legální symbol v názvu elementu i atributu. Parser prostě pracuje s větším názvem obsahujícím i jmenný prostor např. pro element `<ui:insert />` se použije jako identifikátor `ui:insert`. Identifikátor poté nezapříčiní problémy při načtení dokumentu. Parser, který pracuje se jmennými prostory následně kontroluje, jestli všechny prefixy jsou

mapovány na URI. Odmítne dokumenty, které používají nenamapované předpony (s výjimkou `xmlns` které jsou uvedeny v XML specifikaci).

2.3 Validace XML dokumentů

Přestože je v XML syntaxe zanoření značek flexibilní, existuje omezení pomocí gramatiky založené na striktním určení pořadí spolu s pojmenování použitých značek i atributů v nich. Dále neumožňuje křížové zanoření značek přes sebe a značky musí být vždy v páru, případně ve zkrácené formě `<tag/>` atd. Všechny XML dokumenty musí splňovat tyto omezení a jsou tedy zpracovatelné nějakým XML parserem jako je např. LibXML [26], ale to samo o sobě není dostatečné proto, aby byly validní i z pohledu zpracovatele.

Právě tento problém řeší validace vůči validačním schématům jakými jsou např.: Document Type Definition (DTD), XML Schema Definition (XSD) a RelaxNG [29], kde pomocí těchto definic lze determinovat, které elementy mohou být obsaženy v XML dokumentu, případně které mohou být zanořeny do sebe. Dále definují implicitní hodnoty atributů, jejich povolené datové typy.

Pokud jsou XML dokumenty tímto způsobem zvalidovány, tak se dají prohlásit za korektní z pohledu syntaxe i sémantiky.

2.3.1 DTD

DTD (Document Type Definition) [39] je vlastní podmnožinou SGML DTD. Tudíž DTD obsahuje množinu vlastností a elementů, které mohou být obsaženy v XML dokumentu a kontext ve kterém mohou být použity. DTD dokument definuje všechny elementy, atributy a entity, které dokument používá a kontexty, v nichž je používá. Způsob definicí v DTD viz ukázkový dokument 2.2 funguje na premise, která zakazuje vše, co není explicitně povoleno. Tudíž celý XML dokument musí odpovídat prohlášení v DTD. V neposlední řadě DTD může obsahovat definice elementů, které nesmí být v XML dokumentu obsaženy.

Existuje ovšem mnoho omezení které, DTD přináší. Mezi nejdůležitější patří následující:

- Nelze definovat kořenový element.
- Nelze specifikovat počet instancí daného elementu.
- Nedá se určit struktura datového typu.
- Nelze přiřadit sémantický význam elementu, např. zda obsahuje datum nebo jméno osoby.

DTD lze umístit přímo do XML dokumentu, ale díky benevolentním omezením na velikost nelze určit maximální délku dokumentu. Parser XML dokumentu není povinen při zpracování dokumentu kontrolovat jeho platnost vůči DTD. Pokud ji ovšem kontroluje, může chyby ignorovat. Vlastní validace je prováděna až explicitně a chyby, které se v dokumentu objeví nemusí vždy znamenat kritické zastavení. V některých případech, jako je vykreslování webových stránek, může být chyba ignorována a prohlížeč ve většině případů zobrazí stránku správně. Na druhou stranu datově striktní rozhraní jakým je například vkládání záznamů do databáze, vyžaduje přesný formát zdrojových dat, proto jakákoliv chyba v XML dokumentu je kritická.

DTD je tedy mnohem vhodnější pro dokumentově zaměřené XML oproti datově zaměřenému XML. Hlavní příčinou tohoto omezení je slabá podpora datových typů, ty totiž nejsou

potřebné ve strukturovaném textu. Naproti tomu u datově zaměřených XML dokumentů se jedná o esenciální součást, pomocí níž se dodržují deklarativní omezení daná datovými typy. Přesto lze tento způsob validace použít. Jeho největší devizou je velké množství nástrojů, které s ním umí pracovat.

```
<!ELEMENT address ( housenumber, street, town, postcode, country ) >
<!ATTLIST address correspondent NMTOKEN #REQUIRED >
<!ELEMENT country ( #PCDATA ) >
<!ATTLIST country main NMTOKEN #REQUIRED >
<!ELEMENT firstname ( #PCDATA ) >
<!ELEMENT housenumber ( #PCDATA ) >
<!ELEMENT name ( surname, firstname ) >
<!ATTLIST name title NMTOKEN #REQUIRED >
<!ELEMENT person ( name, address ) >
<!ATTLIST person career NMTOKEN #REQUIRED >
<!ATTLIST person id NMTOKEN #REQUIRED >
<!ELEMENT postcode ( #PCDATA ) >
<!ELEMENT street ( #PCDATA ) >
<!ELEMENT surname ( #PCDATA ) >
<!ELEMENT town ( #PCDATA ) >
```

Kód 2.2: Ukázka struktury DTD dokumentu pro vzorový XML dokument 2.1.

2.3.2 XSD

XML Schema Definition (XSD) [24] vzniklo jako odpověď na nedostatky DTD, od nějž se ve velké míře odlišuje. Původní návrh pochází od společnosti Microsoft, ale následně byl plně převzat konzorciem W3C a schválen jako standard [24]. Jazyk XML schema definition je sám založen na XML, tudíž je na rozdíl od DTD dynamicky rozšiřitelný a podporuje jmenné prostory. Dále má lepší podporu pro datové typy, protože obsahuje celou škálu primitivních datových typů jakými jsou: `string, decimal, integer` atd. V neposlední řadě lze definovat komplexní datové typy složením z primitivních datových typů. Tyto vlastnosti činí XSD mnohem použitelnější pro popis datově zaměřených dokumentů. V neposlední řadě plyne výhoda v samotné struktuře XSD, jelikož se jedná o XML dokument, tak se dá validovat vůči referenčnímu XSD a není při tom potřeba zvláštního zacházení jako u DTD, které se svou syntaxí vymyká XML struktuře.

Definice XSD je stejně jako ostatní jazyky z rodiny XML spravována organizací W3C a je rozdělena do 3 částí viz [24] zabývající se základními koncepty použitými v XSD definici, [41] se zabývá použitými strukturami, omezeními a použití jmenných prostorů a poslední část [14] je plně věnována datovým typům.

Jednoduchá struktura XSD schématu viz příloha C, zobrazuje definice pro XML dokument 2.1. Tag s označením `sequence` je uspořádaná kompozice podelementů. Dále element definující adresu obsahuje maximální i minimální omezení na počet možných instancí.

Stejně jako u DTD je hlavním cílem XSD definice platných XML dokumentů. Umožňuje použití těchto konstrukcí:

- Definice elementů které se mohou v XML dokumentu objevit.
- Definice atributů které se mohou v XML dokumentu objevit.

- Definice předchůdce a následníka v stromové struktuře elementů.
- Definice pořadí následných elementů.
- Omezení počtu následných elementů.
- Omezení na obsah elementu.
- Definice datových typů pro atributů i elementy.
- Definice výchozích and fixních hodnot pro elementy a atributy.

2.3.3 RelaxNG

RelaxNG je ve své struktuře velmi podobný XSD a je založen na Relax Core a TREX [29]. Stejně jako u XSD se jedná o XML dokument, tudíž ho zle dynamicky rozšiřovat a validovat vůči referenčnímu RelaxNG dokumentu. Druhým použitím je syntaxe podobná DTD s nímž je zpětně kompatibilní. Jeho hlavní devizou by měla být zjednodušená syntaxe oproti XSD. Naproti tomu největším nedostatkem je jeho menší rozšíření a tím daný nižší počet nástrojů, jež s ním umí pracovat. Tento nedostatek ale nahrazuje možnost převedení RelaxNG schématu na XSD.

Hlavním důvodem většiny použití je jeho funkce tzv. `pivot formátu`, kdy je tento referenční formát dále transformuje do jiných jazyků. Je rozšířen mezi vývojáři, čemuž přispívá i ISO standard ISO/IEC 19757-2:2008.

Jednoduchá struktura RelaxNG schématu viz příloha C zobrazuje definice pro XML dokument 2.1.

2.3.4 Validační model

Vlastní validace je nejčastěji prováděna pomocí konečného automatu (KA).

$$KA = (Q, \Sigma, \delta, g_0, F)$$

1. Q je konečná množina stavů
2. Σ je konečná vstupní abeceda
3. δ je přechodová funkce ve tvaru $\delta : Q \times \Sigma \to 2^Q$
4. $g_0 \in Q$ je počáteční stav
5. $F \subseteq Q$ je množina koncových stavů

Tento KA pracuje s XML dokumentem v serializované podobě. Využívá se vstupně/výstupní abstrakce *stream*, která sekvenčně čte jednotlivé znaky pouze v dopředném směru. Během validace není třeba budovat DOM, případně jiné datové modely nad XML dokumentem, protože validační schémata jej nevyužívají, pouze kontrolují bezkontextovost vlastního dokumentu. Všechny validační přístupy jsou ve své podstatě bezkontextové gramatiky [44], dá se tedy pomocí zásobníkového automatu (ZKA) simulovat jejich běh a tím provést vlastní validaci.

Při implementaci se častěji používá KA místo ZKA, ale pro zvýšení vyjadřovací síly se přidá pomocné počítadlo zanoření.

Strukturální pohled na validační schémata

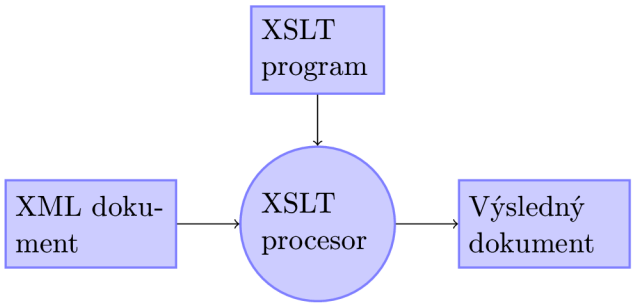
Pro potřebu zavedení matematické konvence se nejčastěji pohlíží na XML dokument jako na orientovaný strom, kde jednotlivé uzly nesou hodnotu v podobě řetězce nad konečnou abecedou Σ a představují tak jednotlivé značky (tag) v XML dokumentu.

Validační schémata (DTD, XSD, RNG) se dají modelovat jako dvojice (d, s), kde d je funkce mapující symboly z abecedy Σ na sekvenci znaků (představující regulární výraz) nad stejnou abecedou Σ a $s \in \Sigma$ je startovací symbol. Jedná se tedy o stromovou strukturu, kde kořenem je s a jednotlivé uzly (a) mají ohodnocení $a_1 \dots a_n$, které představuje regulární výraz odpovídající funkci $d(a)$. Jazyk těchto schémat je často označován jako *local tree language* [44].

2.4 XSLT

XSL (Extensible Stylesheet Languages) je skupina jazyků [31] používaných pro transformaci XML dokumentů do jiného požadovaného formátu. V současnosti je v platnosti verze XLS 2.0, kterou používá norma XPath 2.0 (popsána v kapitole 2.6). Vlastní norma se rozděluje do dvou částí:

- XSLT (XSL Transformace), popisující transformace
- XSL-FO (XSL Formatting Objects), popisující vzhled



Obrázek 2.2: Schéma transformace XSLT.

XSLT je XML aplikace pro určení pravidel, podle kterých je jeden XML dokument transformován do jiného dokumentu XML. XSLT dokument je vlastně šablona, která obsahuje pravidla pro transformaci. Procesor XSLT porovnává prvky a ostatní uzly ve vstupním dokumentu s XSLT šablonou. Při průchodu vstupu se aplikací pravidel vytváří výstupní strom, případně se serializuje do požadovaného formátu. XSLT stejně jako ostatní jazyky určené pro práci s XML využívá ostatních norem, jmenovitě XPath, XLink atd.

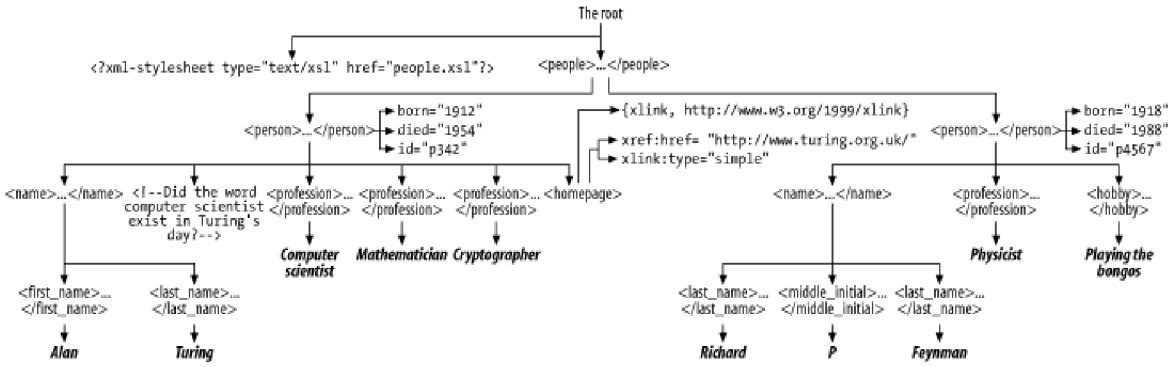
2.5 XPath 1.0

W3C definovala v [20] nový jazyk jímž identifikuje jednotlivé části XML dokumentu, sám svou syntaxí ovšem neodpovídá XML struktuře. Pracuje nad stromovou strukturou dokumentu (*DOM*) a vzdáleně připomíná regulární výrazy identifikující jednotlivé podelementy elementů. Oproti regulárním výrazům umožňuje XPath vytvářet odkazy, které odkazují

na část podstromu, či jednotlivé atributy. XPath identifikuje uzly podle pozice, vzájemné polohy, typu, obsahu a několika dalších kritérií.

XPath výrazy mohou také reprezentovat čísla, řetězce nebo booleovské operace. Tato vlastnost je extenzivně využívána u XSLT stylů při provádění jednoduchých aritmetických operací pro účely, jakými jsou například číslování, či křížové odkazy u obrázků, tabulek a rovnic. Manipulace s řetězci v XPath umožňuje XSLT provádět operace jakými jsou např.: převedení názvu kapitoly na velká písmena nebo výběr posledního dvojčíslí z roku.

XSLT používá XPath výrazy k výběru konkrétního prvku ve vstupním dokumentu pro zpracování a kopírování do výstupního dokumentu. XPointer používá XPath výrazy k identifikaci určitého místa nebo jeho části XML dokumentu, na které XLink odkazuje. W3C XML Schema používá XPath výrazy pro definování jedinečnosti a omezení identity. XForms spoléhá na XPath při kontrole vazby formuláře na instance dat, omezení vstupů zadaných uživateli a výpočet hodnoty, které jsou závislé na jiných hodnotách.



Obrázek 2.3: Stromová struktura XPath parseru. Převzato z [28].

XPath datový model viz diagram 2.3 má několik neobvyklých vlastností. Za prvé, kořenový uzel stromu XPath (dále již kořenový uzel stromu) není stejný jako kořenový element XML dokumentu. Kořenový uzel stromu obsahuje celý XML dokument včetně prvního elementu, stejně jako případné komentáře a instrukce pro zpracování, které se objevují před kořenovým elementem. V ukázkovém diagramu kořenový uzel obsahuje `xml-stylesheet` instrukce pro zpracování, stejně jako `<people>..</people>` kořenový element.

Nicméně, XPath datový model nezahrnuje vše, co je v XML dokumentu obsaženo. Zejména deklarace XML, deklarace DOCTYPE a vnitřní části DTD nejsou adresovatelné pomocí XPath. Jedinou vyjímku tvoří definice defaultních hodnot pro atributy, ty XPath podporuje. V neposlední řadě `xmlns` prefixy u atributů nejsou považovány za identifikátory uzlů. Nicméně, jmenný prostor je brán v patrnost jako rozlišovací prvek.

Výsledkem zpracování XPath dotazu může být:

- Množina uzlů.
- Boolean hodnota.
- Numerická hodnota (v XML je defaultně používán datový typ float)
- Textový řetězec.

Pří skládání XPath výrazu se dají použít některé zkracující syntaxe u identifikátorů os a testů uzlu viz tabulka níže.

Ukázka XPath výrazu pro vzorový XML dokument 2.1 viz kód 2.3.

zkratka	popis
`//`	Zkratka pro `/descendant-or-self::node/`
`@`	Reprezentuje osu `attribute`
`*`	Reprezentuje jakékoliv pojmenování (elementu, atributu, ...)
`.`	Reprezentuje identifikátor osy `self`
`..`	Reprezentuje identifikátor osy `parent`
`text()`	Vybírá textové uzly

Tabulka 2.1: Výběr nejpoužívanějších zkrácených syntaxí v XPath 1.0.

```
/person/name/surname
```

Kód 2.3: Vzorový XPath výraz pro XML dokument 2.1.

Další možnosti omezení jsou pomocí podmínky na hodnotu `[housenumber > 400]`, případně symbolu `"@"` pro identifikaci atributu.

2.6 XPath 2.0

XPath 2.0 je vlastním rozšířením XPath 1.0, které je v některých částech specifikace omezené díky použitému datovému modelu DOM. Proto v roce 2007 byla vydána specifikace XPath 2.0 [6], která definuje nový model XDM. Jazyk je zpětně kompatibilní, avšak některé dílčí specifikace jsou ponechány na implementátorech. XDM datový model viz hierarchická struktura typů (obrázek 2.4), která je společná jak pro XPath 2.0, tak i pro XQuery 1.0.

Stejně jako DOM, tak i XDM reprezentuje XML dokument jako strom. Každý uzel obsahuje buď hodnotu, nebo element. Hodnota je reprezentována některým z atomických typů (např. `xs:string`), případně typem odvozeným.

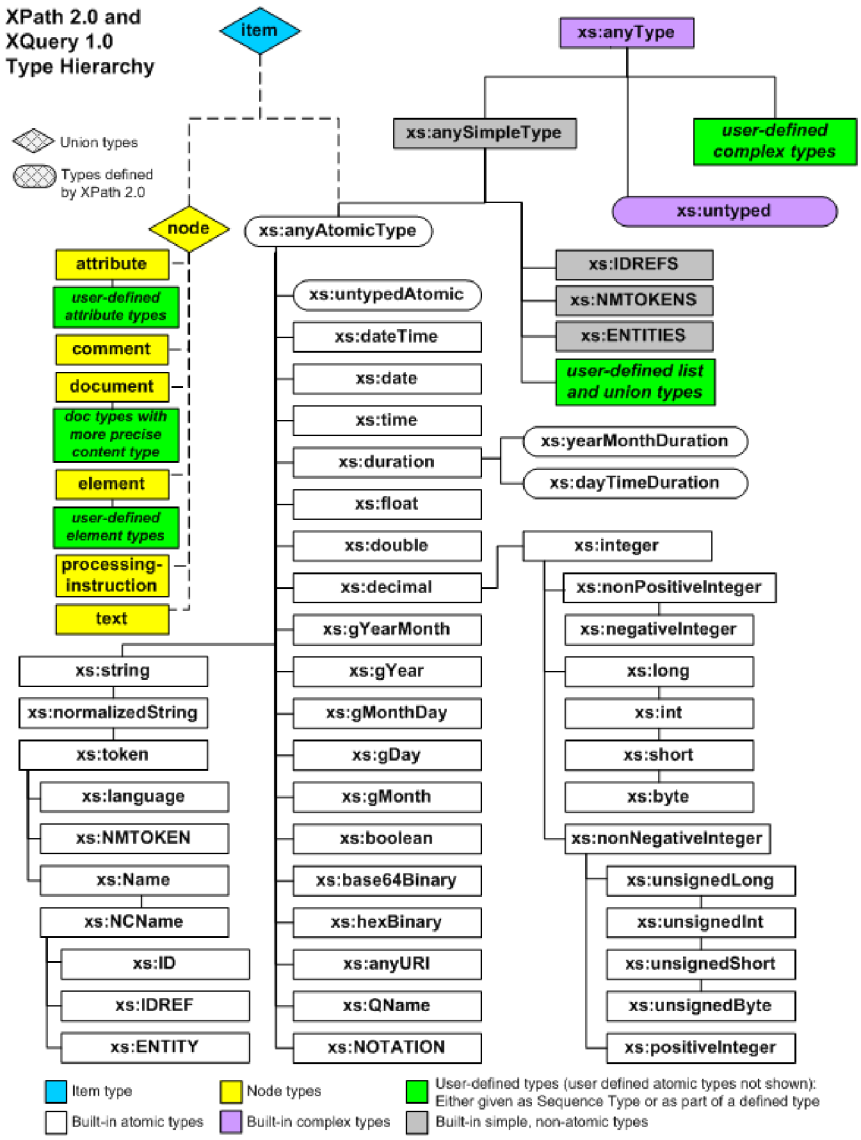
Vyhodnocení XPath výrazu probíhá ve dvou fázích. První statická fáze připravuje XDM instanci vyhodnocením názvů funkcí, proměnných a jmenných prostorů. Druhá dynamická fáze přiřazuje hodnoty do připravených proměnných a vytváří vlastní výstup. Vyhodnocená funkce je předána uživateli jako serializovaný výsledek bez jakékoliv výstupní transformace.

2.7 XQuery

Mezinárodní organizace W3C definovala normou jazyk XQuery [15] jakožto standardní dotazovací jazyk pro XML. Využívá jazyka XPath pro adresaci a práci nad XML dokumentem. Definice XQuery 1.0 dokonce zahrnuje XPath 2.0. pro navigaci nad stromovou strukturou dokumentu.

XQuery bylo definováno až po SQL normě, přesto se od ní do značné míry odlišuje. Je to zapříčiněno odlišným přístupem k reprezentaci dat. Dále lze nad prováděnými dotazy zároveň provádět transformace (agregace, filtrování, ...). Ukázka dotazu pomocí XQuery viz kód 2.4.

Proměnné se označují pomocí "$"a řetězce se uvozují pomocí jednoduchých, případně dvojitých uvozovek. Jazyk XQuery také povoluje podmíněné výrazy (if then else) spolu v omezením na hodnotu pomocí binárních operátorů ($<, \leq, =, \neq, \geq, >$), nebo nově zavedených operátorů `lt, le, gt, ge, eq` pro porovnání nad neatomickými datovými typy.



Obrázek 2.4: XDM hierarchická struktura typů. Převzato z [36].

```
for $x in doc("person.xml")/person/address
where $x/housenumber>5
order by $x/town
return $x/name
```

Kód 2.4: Ukázka XQuery dotazu nad vzorovým XML dokumentem 2.1.

Používá se základní členění do 5 kategorii (*FLWOR*): `for`, `let`, `where`, `order by`, `return`:

- Klauzule `for` je podobná svému jmenovci v SQL, specifikuje totiž proměnné, případně rozsah nad nímž je prováděn dotaz. Pokud není specifikován, tak se použije kartézský součin všech možných hodnot, nad nimiž je prováděn výpočet stejně jako u SQL.

- Klauzule `let` umožňuje uložit pod-dotaz do proměnné a pracovat s ní následujících částech dotazu.
- Klauzule `where` funguje podobně jako v SQL provádí dodatečnou restrikci nad n-ticemi definovanými v klauzuli `from`.
- Klauzule `order by` podobně jako v SQL provádí seřazení nad výsledkem dotazu.
- Pseudoproměnná `return` se používá pro uložení výsledku jakožto XML dokumentu.

Kapitola 3

SQL normy definující operace nad XML dokumenty

Mezinárodní standardizační organizace ISO spolu s ANSI vydávají standardy, jimiž se řídí tvůrci databázových systémů. Finální verze norem není bezplatně dostupná, avšak poslední pracovní draft je volně šiřitelný. První norma týkající se SQL byla dle [38] vydána v roce 1986 organizací ANSI a následně o rok později v nezměněné podobě i organizací ISO pod názvem SQL-87. Její hlavní část definuje základní podobu relačního přístupu k datům. Pojmenování SQL norem dodržuje konvence SQL:rok_vydání a je tak snadné jednotlivé normy od sebe odlišit.

Od normy ISO SQL:2003 [3] jsou diskutovány nejdůležitější body týkající se práce s XML dokumenty. Všechny SQL normy jsou vytvářeny inkrementálním způsobem, tedy například norma ISO SQL:2003 vychází z ISO SQL:1999. Pokud dojde k redefinici, případně kolizi v použití je tato skutečnost explicitně uvedena ve zvláštním oddílu normy a vždy se uvádí kolize vůči předchůdci aktuální normy.

3.1 ISO SQL:2003

SQL:2003 je pátou v pořadí norem definujících SQL a je první normou definující operace nad XML, veškeré definice týkající se XML přebírá z definic W3C a je s nimi tedy kompatibilní.

Norma se skládá z těchto částí, které definují jednotlivé oblasti:

- ISO/IEC 9075-1:2003 - Framework (SQL/Framework)
- ISO/IEC 9075-2:2003 - Foundation (SQL/Foundation)
- ISO/IEC 9075-3:2003 - Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4:2003 - Persistent Stored Modules (SQL/PSM)
- ISO/IEC 9075-9:2003 - Management of External Data (SQL/MED)
- ISO/IEC 9075-10:2003 - Object Language Bindings (SQL/OLB)
- ISO/IEC 9075-11:2003 - Information and Definition Schemas (SQL/Schemata)

- ISO/IEC 9075-13:2003 - SQL Routines and Types Using the Java Programming Language (SQL/JRT)
- ISO/IEC 9075-14:2003 - XML-Related Specifications (SQL/XML)

Nejdůležitější je poslední část, která definuje práci s XML dokumenty. Datovým typům použitým v XML přiřazuje ekvivalentní datové typy v programovacích jazycích jmenovitě pro jazyky: ADA, C, Cobol, Fortran, NUMPS, Pascal a PL/I. Dále norma definuje převod datových typů definovaných v SQL na datové typy používané v XML.

Jelikož W3C pro XML definuje jedinou pracovní znakovou sadu UTF-8 a SQL umožňuje použití většího množství znakových sad, tak je třeba provést mapování SQL znakových literálů na UTF-8 znakovou sadu. Používá se homomorfního přiřazení každého znaku kladnému číslu v rozsahu datového typu `32b integer`. Pokud dojde při mapovaní k neplatnému přiřazení mimo stanovený rozsah, tak SQL norma definuje chybový kód.

Stěžejní části normy je mapování SQL tabulek, SQL schémat (databázové struktury tabulek) a SQL katalogů (metadat nad databázovou strukturou) na odpovídající XML reprezentace. Pro tabulky se mapování provádí vytvořením schématu z definicí sloupců a následného naplnění daty ze všech řádků v tabulce uvozením `<row>` tagu. Pokud sloupec může obsahovat `NULL` hodnotu, objeví se u odpovídajícího elementu v XML dokumentu atribut `xsi:nil="true"`.

Mapování SQL schématu ve svém důsledku vytvoří 2 XML dokumenty. Pro data z databáze je vytvořen jeden XML dokument a následně je vygenerován XML dokument definující XSD schéma pro tato data. Kvůli zabezpečení jsou do mapování zahrnuty pouze ty tabulky v SQL schématu, pro které má uživatel právo `SELECT`.

Mapování SQL katalogu na XML dokument je podobné jako u SQL schémat. Až na odlišnost kdy se některé datové typy použité v XML Schema a elementy v SQL katalogu mapovat pomocí SQL metadata, kvůli zachování nepřímé vazby na XML. Jedná se o implementačně závislé řešení, které norma striktně nevynucuje.

Jmenné prostory jsou použity základní, které jsou převzaté od W3C, dále přibyl jmenný prostor `sqlxml` viz tabulka 3.1. Implementace jmenných prostorů není vyžadována, ale je doporučena spolu s použitím základních jmenných prostorů a jejich odpovídajícím URI.

XML namespace	**XML namespace URI**
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
xq	http://www.w3.org/2002/11/xquery-functions
sqlxml	http://standards.iso.org/iso/9075/2003/sqlxml

Tabulka 3.1: Jmenné prostory a jejich příslušející URI definované dle ISO normy [3].

Důležitou částí SQL:2003 je deklarace kompatibility s SQL/XML definovaném v první části normy [2]. SQL/XML obsahuje sadu deklarací funkcí nad XML dokumenty a je jedna ze základních funkčních bloků na které se staví další definice. Jedná se o rozšířená jazyka SQL které umožňuje export dat do XML dokumentu.

XML datový typ

Před vydáním normy ISO SQL:2003 se XML dokumenty ukládaly jako hodnoty typů `VARCHAR` případně `CLOB`. Druhý způsob uložení dokumentu spočíval v tzv. *schredding*

dekompozici XML dokumentu do relačního modelu. Až norma ISO SQL:2003 zavádí dedikovaný datový typ pro XML dokumenty, kde jeho vlastní implementace je ponechána na tvůrcích databázových systémů. Sloupec typu `XML` může obsahovat: XML dokument, jednotlivé elementy, strom elementů a semistrukturovaný obsah. Atributy se mohou vyskytovat pouze v rámci elementů, samy o sobě nejsou legální XML hodnotou. V rámci normy SQL:2003 byly zakázány komentáře, toto omezení je součásti úprav v rámci aktuálnějších SQL norem.

3.1.1 SQL/XML funkce

Funkce pro základní práci s XML definují mapování mezi databázovými tabulkami a XML dokumentem. Kdy může jít o jednu tabulku, případně kolekci tabulek, či celou databázi. Operace je prováděna pod určitým uživatelem, tudíž jsou vyžadována oprávnění na `SELECT`.

Vytváření XML fragmentů

Bylo zavedeno několik operací, které vytváří jednotlivé části XML dokumentu.

- `XMLELEMENT` operace vytváří XML element, není omezena počtem argumentů a dá se tudíž vytvořit hierarchická struktura XML dokumentu viz ukázka SQL kódu 3.1 a výsledek volání v tabulce 3.2.

```
SELECT table.id, XMLELEMENT (NAME "tag_name",
       XMLATTRIBUTES (table.column AS "attribute1"),
       "unstructured text",
       XMLELEMENT (NAME "tag_name2")
       ) AS "result"
FROM table;
```

Kód 3.1: Ukázka použití funkce `XMLELEMENT` a `XMLATTRIBUTES`.

id	result
1	<tag_name attribute1='some_value'> unstructured text <tag_name2></tag_name2></tag_name>
2	<tag_name attribute1='other_value'> unstructured text <tag_name2></tag_name2></tag_name>

Tabulka 3.2: Výsledek volání funkce `XMLELEMENT`.

- `XMLFOREST` vytváří strom elementů z každého prvku definovaného v argumentech viz ukázka SQL kódu 3.2 a výsledek volání funkce ve výpisu 3.3.

```
SELECT table.id, XMLFOREST (table.column AS "element1",
       table.column2 AS "element2",
       ) AS "result"
FROM table;
```

Kód 3.2: Ukázka použití funkce `XMLFOREST`.

id	result
1	<element1>table-column-data</element1> <element2>table-column-data</element2>
2	<element1>other-table-column-data</element1> <element2>other-table-column-data</element2>

Tabulka 3.3: Výsledek volání funkce `XMLFOREST`.

- `XMLGEN` vytváří XML dokument vyplněním hodnot do šablony tak jak ukazuje SQL kód 3.3, kde {$PROMENNA} představuje zástupný symbol (place holder) pro stejně pojmenovaný sloupec v rámci dotazu a je následně nahrazen výslednou hodnotou. Výsledek volání `XMLGEN` prezentuje tabulka 3.4.

```
SELECT table.id,
       XMLGEN ('<tag_name atribute="{\$COLUMN_NAME}" />'
       ) AS "result"
FROM table;
```

Kód 3.3: Ukázka použití funkce `XMLGEN`.

id	result
1	<tag_name attribute='some_value' />
2	<tag_name attribute='other_value' />

Tabulka 3.4: Výsledek volání funkce `XMLGEN`.

- `XMLCONCAT` vytváří strom elementů konkatenací elementů, tak jak ukazuje volání funkce v SQL kódu 3.4 a výsledek volání v tabulce 3.5.

```
SELECT table.id,
       XMLCONCAT (XMLELEMENT (NAME "first", table.column),
              XMLELEMENT (NAME "second")
       ) AS "result"
FROM table;
```

Kód 3.4: Ukázka použití funkce `XMLCONCAT`.

id	result
1	<first>data</first><second></second>
2	<first>other_data</first><second></second>

Tabulka 3.5: Výsledek volání funkce `XMLCONCAT`.

- `XMLAGG` je agregační funkce vytvářející strom elementů z kolekce elementů. Hodnoty agregátů jsou předány pomocí vnější SQL agregační funkce viz kód SQL 3.5 a výsledná tabulka po volání funkce 3.6.

```
SELECT table.id,
       XMLCONCAT (XMLELEMENT (NAME "first"),
              XMLAGG (
                     XMLELEMENT (NAME "second", table.agg_column)
              )
       ) AS "result"
FROM table;
```

Kód 3.5: Ukázka použití funkce `XMLAGG`.

id	result
1	<first><second>agg_value1</second> <second>agg_value2</second></first>

Tabulka 3.6: Výsledek volání funkce `XMLCONCAT`.

Mapování relačních dat na XML dokument

Data z tabulky 3.7 se mapují na XML dokument tak, že pro každý řádek se vytvoří element `<row>...</row>` pomocí funkce `XMLELEMENT`, která jako vnitřní atributy volá znovu funkce `XMLELEMENT`, vytvářející elementy pro každý sloupec ve zdrojové tabulce. Výsledek transformačních funkcí je XML dokument 3.6.

id (INT)	data1 (TEXT)	data2 (NUMERIC(9,4))
1	'textual data1'	123.45
2	'textual data2'	67890.23

Tabulka 3.7: Ukázková tabulka `Example` se 3 sloupci `(INT, TEXT, NUMERIC(9,4))`.

```
<Example>
       <row>
              <id>1</id>
              <data1>textual data1</data1>
              <data2>123.45</data2>
       </row>
       <row>
              <id>2</id>
              <data1>textual data3</data1>
              <data2>67890.23</data2>
       </row>
</Example>
```

Kód 3.6: Výsledek mapování ukázkové tabulky 3.7 na XML dokument.

Kořenový element je pojmenován podle názvu tabulky. Každý řádek v tabulce je uzavřen v elementu `<row> ... </row>`. Hodnoty obsažené v tabulce jsou mapovány na odpovídající

XML datové typy a pokud je v tabulce obsažen řádek, který obsahuje `NULL` hodnotu je mapován do XML jako `xsi:nil="true"`.

3.2 ISO SQL:2008

SQL:2008 je sedmou v pořadí norem definujících SQL a v současnosti je nejaktuálnější platnou normou. Stejně jako předchozí revize, vychází ze zdrojové normy ISO SQL:2006 [4] a tu doplňuje o nové definice. Revize z roku 2008 používá novější definice vycházející z XQuery 1.0 a jinak je v části týkající se XML shodná s ISO SQL:2006. Byla tedy upřednostněna před popisem revize z roku 2006. Skládá se také z 9 částí.

- ISO/IEC 9075-1:2008 Framework (SQL/Framework)
- ISO/IEC 9075-2:2008 Foundation (SQL/Foundation)
- ISO/IEC 9075-3:2008 Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4:2008 Persistent Stored Modules (SQL/PSM)
- ISO/IEC 9075-9:2008 Management of External Data (SQL/MED)
- ISO/IEC 9075-10:2008 Object Language Bindings (SQL/OLB)
- ISO/IEC 9075-11:2008 Information and Definition Schemas (SQL/Schemata)
- ISO/IEC 9075-13:2008 SQL Routines and Types Using the Java TM Programming Language (SQL/JRT)
- ISO/IEC 9075-14:2008 XML-Related Specifications (SQL/XML)

9. část přidává oproti předchozí normě podporu pro XML schémata (XSD) a jejich registraci pro pozdější validace. Definice jmenných prostorů pro XML dokumenty zůstává shodná s předchozí normou, ale jmenné prostory jsou nyní implicitně registrovány do centralizovaného seznamu registrovaných jmenných prostorů. Z bezpečnostních důvodu není dovoleno použít pro validaci jiných než předem registrovaných validačních schémat.

Největší změnou v XML části je aktualizace XML datového typu. Došlo k rozdělení na několik podtypů a to jmenovitě na:

- `XML(SEQUENCE)` je kolekce XML dokumentů (nemusí být správně strukturována) a dalších dat bez bližšího určení. Tento datový typ může nabývat hodnoty `NULL`.
- `XML(ANY CONTENT)` obsahuje XML dokument, který nemusí být validní ani správně strukturovaný. Tento datový typ může nabývat hodnoty `NULL`.
- `XML(UNTYPED CONTENT)` obsahuje XML dokument bez explicitně přiřazených datových typů u jednotlivých elementů a atributů. Také nemusí být správně strukturován.
- `XML(ANY DOCUMENT)` obsahuje XML dokument, kdy každý element má maximálně jeden synovský podelement.
- `XML(UNTYPED DOCUMENT)` je XML dokument, kdy každý element má maximálně jeden synovský podelement. Rozdíl oproti `XML(ANY DOCUMENT)` je ten, že může nabývat hodnoty `NULL`.

Další podstatnou změnou je silná typová kontrola, která je daná použitým datovým modelem `XQuery Data Model` viz kapitola o XQuery 2.7. Také byl přidán predikát `VALID`, který indikuje validní XML dokument vůči registrovaným schématům.

Norma zavedla nová klíčová slova:

- `XMLCOMMENT` - Funkce vytvářející komentáře `<!-- comment -->`.
- `XMLPI` - Funkce vytvářející tzv. procesní rozšíření XML `<? target expression ?>`, které se používá pro vytvoření hlavičky XML dokumentu, deklaraci XSLT šablony aj.
- `XMLQUERY` - Vyhodnocuje XQuery dotaz nad zvoleným XML dokumentem.
- `XMLCAST` - Konverzní funkce mezi jednotlivými typy XML dokumentu (`XML(SEQUENCE)`, `XML(ANY CONTENT)`, ...
- `XMLEXISTS` - Funkce testuje zvolený XML dokument na platnost XQuery dotazu.
- `XMLITERATE` - Bez přiřazené sémantiky, určeno pro budoucí použití.

3.3 ISO SQL:20nn pracovní draft

Aktuální pracovní draft ze dne 3.1.2011 přidává definice pro serializaci a deserializaci hodnot v XML dokumentu. Dále přidává podporu pro operaci `MERGE` (spojení) nad XML dokumenty. V poslední revizi nebyla nalezena nekompatibilita s předchozí verzí normy, stejně tak nepřidává žádné klíčové slovo. Jedná se o pracovní verzi, tudíž lze očekávat úpravy v definicích. Předpokládaný rok vydání je 2011.

SQL datový typ	XML ekvivalent
CHARACTER	xs:string
BINARY LARGE OBJECT	xs:hexBinary
NUMERIC	xs:decimal
INTEGER	xs:integer
REAL	xs:float
BOOLEAN	xs:boolean
DATE	xs:date

Tabulka 3.8: Přehled reprezentativních SQL datových typů a jejich ekvivalentů v XML.

Norma definuje mapování SQL datových typů na XML datové typy. Mapování je surjektivní, tedy pro množinu datových typů v SQL existuje většinou jediný v XML, kde se pomocí dalších atributů specifikují omezení velikosti a rozsahu.

Pro datový typ CHARACTER se přidává specifikace délky pomocí atributu `xs:length`, případně pro CHARACTER VARYING a CHARACTER LARGE OBJECT se definuje `xs:maxLength`. Pro binární formáty jsou definovány dva způsoby mapování buď zmíněný `xs:hexBinary`, nebo `xs:base64Binary`.

Datové typy určené pro reprezentaci měn (NUMERIC, DECIMAL) definují omezení pomocí `xs:precision` a `xs:scale`.

Pokud se SQL datový typ pro desetinná čísla (REAL, DOUBLE PRECISION, FLOAT) dají prezentovat do méně než 24 bitů, tak se použije `xs:float` pro větší rozsah se použije `xs:double`.

Kapitola 4

Indexace XML dokumentů

Indexy jsou datové struktury, které se primárně používají pro určení lokace specifických dat. Klasickým příkladem může být index na konci knihy, který sumarizuje výskyty jednotlivých hesel na stránkách. Tento přístup si klade za cíl snížit časovou náročnost, která by byla jinak třeba při sekvenčním průchodu celou knihou.

V případě databází, kde jsou data typicky prezentována pomocí relačního modelu a jednotkou uspořádaných `n-tic` nad potenční množinou 2^N je tabulka, určují indexy pozice jednotlivých řádků a není třeba provádět sekvenční průchod všemi záznamy (tzv. `table scan`). Akcelerace dotazování je vykoupena dodatečným prostorem, který je třeba alokovat v trvalém úložišti pro vlastní index.

XML dotazovací jazyky, jakými jsou XPath nebo XQuery používají specifických výrazů pro identifikaci cesty při navigaci po logické struktuře XML dokumentů. Při konvenčním přístupu pak XPath procesor obvykle používá některý ze dvou známých způsobů průchodech DOM a to buď shora-dolů nebo zdola-nahoru a současně přitom vyhodnocuje XPath výraz. Příkladem těchto implementací jsou například Jaxen, DOM4J, nebo LibXML.

Tento čistě stromový přístup je velmi neefektivní pro velké sbírky XML dokumentů. Uvažujme XPath dotaz, který bude vybírat všechny knihy, jejichž název obsahuje řetězec `XML` vypadá následovně: `/knihy//obálka[contains (název, 'XML')]`

Při konvenčním top-down přístupu provádění dotazovacího plánu musí XPath procesor procházet všechny následníky uzlu `knihy` a zkontrolovat, jestli obsahují následníka, případně jestli se nejedná o hledanou sekci `obálka`. To znamená průchod každým uzlem v dokumentu stromu pod elementem `kniha`, protože neexistuje způsob, jak zjistit případné umístění následníka předem.

Strojový čas nutný pro tyto nezbytné průchody ovšem není úzkým místem algoritmu. Největší problém činí počet I/O operací a to primárně pokud je celý dokument umístěn na disku. Navíc, celý postup se musí opakovat pro každý jednotlivý dokument ve sbírce, což činí tuto metodu nepoužitelnou pro velké sbírky XML dokumentů, jež se typicky nacházejí v databázích.

Z těchto důvodů vyvstává potřeba použití indexovací techniky, která by zvýšila efektivitu dotazování nad XML daty a to i pro předem neznámé rozsáhlé dotazy operující nad neomezeným množstvím XML dokumentů. Použití indexu by tedy mělo snížit počet přístupů do trvalého úložiště a adekvátně s tím zvýšit rychlost zpracování pokládaných dotazů.

Pro jednoduchý XPath dotaz `///obálka[contains (název, 'XML')]` by měl indexovací systém poskytovat dvě hlavní funkce. Za prvé by místo procházení podstromů měl okamžitě vrátit adresy všech prvků s identifikátorem `obálka`. To lze provést například tak,

že se namapují názvy elementů na seznam odkazů, které odkazují na odpovídající DOM uzly v uloženém XML dokumentu. Za druhé by index měl být alespoň částečně adaptabilní a reagovat na změny ve zdrojovém dokumentu bez potřeby sestavit celý index znovu.

Mezi další požadované vlastnosti patří znalost potenciálního předchůdce a následníka uzlu, kde ovšem tato informace není sama o sobě dostatečná pro obecné určení vztahu mezi libovolnou dvojicí uzlů. Proto je nutná druhá funkce, která pomocí informace obsažené v hierarchickém rejstříku provede rychlou identifikaci vztahů mezi jednotlivými uzly, které jsou relevantní v průběhu zpracování dotazu.

Další problém, který je třeba brát v úvahu při implementaci XML indexu je jeho celková velikost. Příliš rozsáhlý index způsobí vyšší počet přístupů do trvalého úložiště a tím se zpomalí celkové zpracování. Dále je třeba brát v patrnost datové typy obsažené v XML a jejich mapování na příslušné SQL datové typy.

4.1 Datový model

Indexy, které byly dosud navrženy se dají dle [8] rozdělit dle použitého datového modelu do 4 hlavních kategorií a to na: strukturální, sekvenční, numerické a indexy založené na klíčových slovech, kdy se každá kategorie indexu hodí pro jiný účel.

4.2 Základní indexační struktury

Existuje velké množství základních indexačních struktur, které se používají u nestrukturovaných dokumentů. V této podkapitole budou popsány nejčastější přístupy, které se používají jak u typicky relačního přístupu, tak i při reprezentaci dokumentu pomocí stromu.

Tyto základní indexační přístupy jsou základem i pro komplexnější přístupy používané u strukturovaných datových typů.

Invertovaný soubor

Invertovaný soubor je základní indexovací technika, která je velmi podobná klasickému indexu v knihách. Ukládá se každé klíčové slovo a spolu s ním odkazy na všechny výskyty v textu viz 4.1. Vlastní odkazy mohou být adresovány jako offset od začátku/konce dokumentu, stránka dokumentu, případně jinak.

klíčové slovo	adresa výskytu
`"index"`	&1;&6
`"prales"`	&0;&3;&7
`"praděd"`	&0;&3;&9
`"proces"`	&2

Tabulka 4.1: Invertovaný soubor.

Document/term matice

Jedná se o dvoudimenzionální matici (k, d), kde pro každé klíčové slovo (term) k a každý dokument d obsahuje příslušné pole ohodnocení `1`, pokud je k obsaženo v d, jinak obsahuje hodnotu `0`, jak ukazuje tabulka 4.2. Vlastní index je do velké míry podobný invertovanému

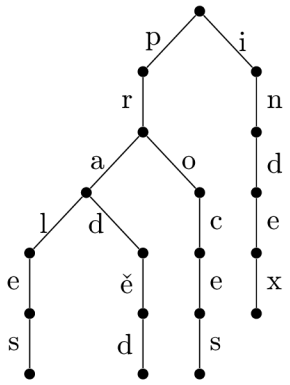
souboru, kdy jsou udržovány reference klíčových slov spolu s jejich výskyty. Hlavním rozdílem je vyšší granularita dotazování, kdy lze provádět vyhledávání podle klíčových slov a jako výsledek je vrácen seznam dokumentů, které ho obsahují. A druhý způsob dotazovaní kdy lze k hledanému klíčovému slovu snadno identifikovat všechna ostatní klíčová slova spolu s výskyty ve všech ostatních dokumentech.

klíčové slovo	d_0	d_1	d_2
`"index"`	0	1	0
`"prales"`	1	0	1
`"praděd"`	1	1	1
`"proces"`	0	1	1

Tabulka 4.2: Document/term matice. d_x pro $x \in < 0, 2 >$ představují dokumenty.

Trie

Trie [25] je stromová struktura, kde hrany mezi jednotlivými uzly jsou ohodnoceny symbolem z použité abecedy, jak ukazuje obrázek 4.1. Indexovaný řetězec se dá rekonstruovat průchodem stromu od kořene po koncový symbol. Společné předpony klíčových slov jsou uloženy pouze jednou. Vlastní offsety klíčových slov jsou pro přehlednost vynechány.



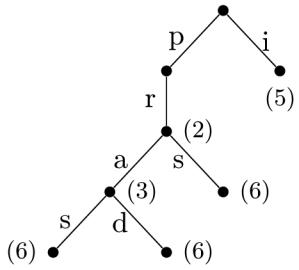
Obrázek 4.1: Struktura Trie.

Patricia Trie

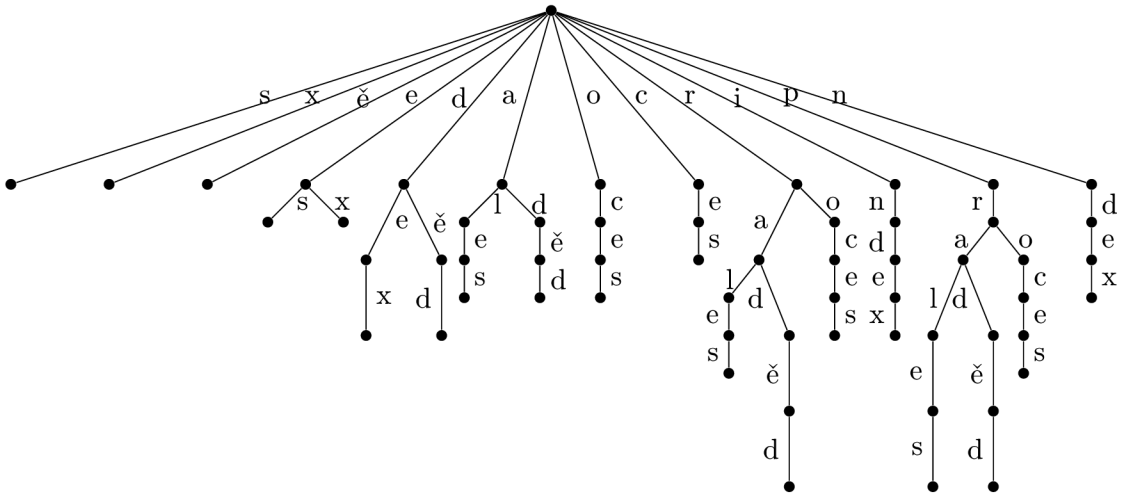
Patricia Trie [34] je rozšířením stromové struktury Trie. Hlavní nevýhodou struktury Trie je její prostorová náročnost, kdy pro každý symbol existuje samostatná hrana. Patricia Trie vynechává hrany, které se v žádném následníkovi nedělí. Jak ukazuje obrázek 4.2, kdy je celá cesta `index` zakóduje do jediné hrany s ohodnocením `i` a koncový uzel nese hodnotu reprezentující délku celého klíčového slova. Obdobným způsobem se pracuje u slov, jež mají společnou předponu.

Suffix tree

Suffix tree je struktura Trie, vybudována nad všemi sufixy daných klíčových slov. Strom se tedy skládá z původního Trie stromu a přidaných všech sufixů, které obsahuje. Obrázek 4.3.



Obrázek 4.2: Struktura Patricia Trie.



Obrázek 4.3: Struktura Suffix tree.

Na rozdíl od struktury Patricia Trie obsahuje plnou cestu dle klíčových slov, ale existuje varianta PAT tree [27], která používá výše zmíněnou Paricia kompresi.

4.3 Datové modely strukturovaných dokumentů

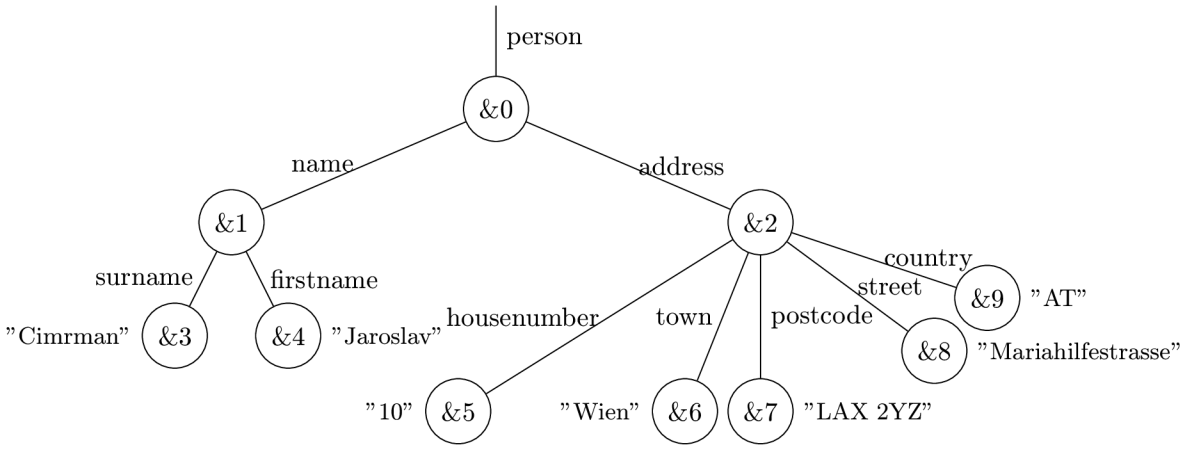
Pro semistrukturované dokumenty existuje několik datový modelů viz ([7], případně [40]). Každý z těchto přístupů řeší rozdílným způsobem formalizmus pro dotazování, indexaci, transformaci a prezentaci výsledných dat. Strukturální indexy vždy pracují nad stromovou strukturou XML dokumentu.

Datový model pro strukturované dokumenty se liší od klasického jedno-rozměrného přístupu, kdy se mapuje lineární sekvence slov do lineárního modelu datového úložiště. Druhý rozměr strukturovaných dokumentů musí být reflektován a při vytváření indexu brán v potaz. Výsledný 2 dimenzionální dokument je stejně serializován do jednorozměrného proudu symbolů pro potřeby perzistence.

Obrázek 4.4 zobrazuje analogii mezi fyzickou prezentací XML dokumentu 2.1 a jeho logickým obrazem. Každý uzel obsahuje hodnotu reprezentující adresu v rámci XML dokumentu. Používá se jednoznačný identifikátor v podobě kladného celého čísla.

Přehled hierarchických konceptů:

1. Label path je sekvence značení (labels), které přísluší jednotlivým hranám daného



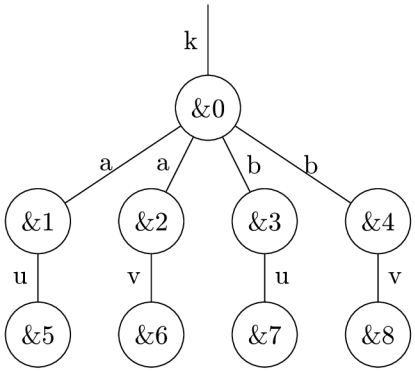
Obrázek 4.4: Logická struktura XML dokumentu.

grafu. Například `/person/name/surname` je label path, který se vyskytuje v 2.1 pouze v jedné instanci.

2. ID path je sekvence identifikátorů (ID), která přísluší uzlům daného grafu. Pro XML dokument 2.1 je ID path např. `&0/&2/&5/`

4.3.1 Deskriptivní schéma

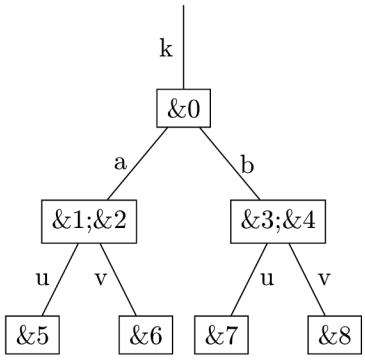
Při provádění dotazu nad kolekcí strukturovaných dat se dá úspěšně použít navigačních indexů, kde navigační indexy jsou obrazem aktuálního stavu databáze (tzv. deskriptivního schématu).



Obrázek 4.5: Stromová reprezentace label path.

Uvažujme kolekci strukturovaných dokumentů (reprezentovaných stromovou strukturou v logickém pohledu), kde P představuje množinu rozlišitelných cest. Pak deskriptivní schéma pro stromy dokumentu je také strom, kde je každý prvek z P obsažen právě jednou. Pro databázový pohled, jež reprezentuje obrázek 4.5 odpovídá schéma na obrázku 4.6.

Jednotlivé uzly v deskriptivního schématu představují ekvivalenční třídy $[n]_{\equiv_p}$, jež jsou generovány relací $\equiv_p$. Relace je definována nad množinou uzlů N, kde pro $\forall n_1, n_2 \in N$,



Obrázek 4.6: Deskriptivní schéma pro obrázek 4.5.

platí:

1. Uzel $n_1 \equiv_p n_2$, pokud je n_1 dosažitelný po stejné cestě jako n_2
2. Pokud $n_1 \neq n_2$, pak $[n]_{\equiv_p}$ je dosažitelná po stejné cestě a uzel je anotován referencí na všechny adresy $[n]_{\equiv_p}$.

Dá se dokázat viz definice v knize [43] (strana 30), že každý uzel v stromové reprezentaci label path 4.5 je odkazován právě jednou cestou v deskriptivním schématu. Některá literatura [30], značí uvedenou třídu ekvivalence pomocí $n_1 \approx^p n_2$ a označuje ji "bi-similar" relace.

Vlastní dotazování je nejprve prováděno nalezením shody v deskriptivním schématu, čímž se určí příslušnost do stromové reprezentace label path, popřípadě vlastní existence hledaného uzlu.

4.3.2 Indexace nad strukturovanými dokumenty

Práce se strukturovaným dokumentem je odlišná oproti plochému modelu použitému v relačních databázích. Je třeba brát v potaz nejen obsah, ale i strukturální informaci. Indexační techniky často využívají více druhů klasických indexů pracujících nad obsahem dokumentu spolu se strukturálním indexem.

Při provádění dotazů se jednotlivé indexy používají, záleží tedy na implementaci jestli bude vlastní index používat jednotlivé samostatné podindexy, které se budou skládat pomocí operace JOIN, případně se všechny nutné složky zkombinují do jedné struktury jako u IndexFabric 4.4.2.

4.3.3 Modely indexů pro strukturované dokumenty

Existuje celá řada indexů pro strukturované dokumenty mezi něž patří i XML. V této kapitole budou nejznámější přístupy prezentovány a zhodnoceny dle zvolených kritérií jež jsou následující:

- Způsob identifikace jednotlivých uzlů.
- Způsob prohledávání stromu (prohledáváním grafu, použití JOIN operací, porovnáním ID identifikátorů).

- Způsob vyhledání při porovnání na shodu (průchod B-tree, table scan).
- Aktualizace indexu při změně XML dokumentu (inkrementální přístup).
- Celková velikost indexu (velikost jednotlivých záznamů).

V případě indexování XML dat existují dva základní způsoby jak uložit původní XML dokument:

- Uložit celý XML dokument beze změny.
- Uložit pouze množinu struktur, která umožňuje rekonstrukci původního XML. dokumentu.

Při použití první metody se původní XML dokument využívá pro adresaci v rámci indexu. Je vhodná pro operace nad celým dokumentem, ale méně při operacích nad částmi dokumentu. Její největší nevýhodou je vyšší datová náročnost. Naproti tomu u druhé metody se musí XML dokument pokaždé rekonstruovat, ovšem velikost použité paměti je menší na úkor výpočetní složitosti.

Datový model vlastního indexu

Všechny indexy, které jsou v této práci zmíněny operují nad jedním z dvojice datových modelů. Prvním je klasický přístup pomocí tabulek a druhý je orientovaný graf. Je třeba odlišovat logický model od reálné implementace. Například indexy založené na grafu mohou být implementovány pomocí tabulky. Stejně tak sekundární indexy, které jsou sestaveny nad primárními indexy mohou používat odlišný logický model dat.

4.3.4 Identifikace uzlů

Indexační techniky, které pracující s jednotlivými uzly, musí tyto uzly jednoznačně identifikovat. Je tedy třeba vytvořit bijektivní zobrazení mapující $id : NodeIds \longrightarrow DocNodes$, kde $NodeIds$ je množina identifikátorů uzlů a $DocNodes$ je množina všech uzlů. Vlastní identifikátor uzlu může obsahovat dodatečné informace. Nejčastěji se jedná o hierarchickou strukturu předcházející tento uzel. Případně je jedná o informaci o ID předchůdce a následníka. Toto kódování vztahu rodič/potomek je užitečné pro efektivní vyhodnocení dotazu bez nutnosti prohledávat celou strukturu obecného vztahu předchůdce/následník.

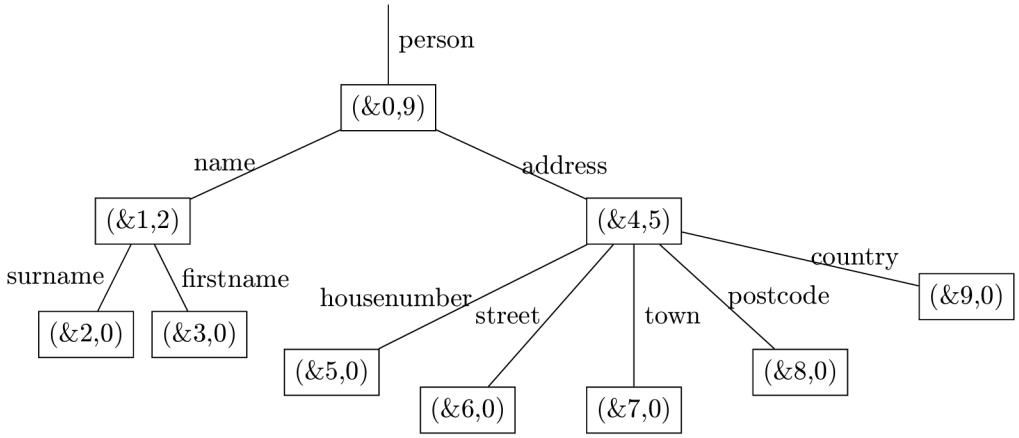
Jednoznačné přiřazení identifikátoru uzlu musí být zachováno i po operacích nad nosným dokumentem. V případě vložení/smazání nového uzlu dochází k zneplatnění množiny již přiřazených identifikátorů a je třeba tento stav reflektovat. Některé metody proto mapují identifikátory s využitím rezerv tzv. *DummyID*, které představují virtuální uzly, čímž se sníží potřeba přemapování množiny neplatných ID po každé úpravě zdrojového dokumentu.

Pre/post order kódování uzlu

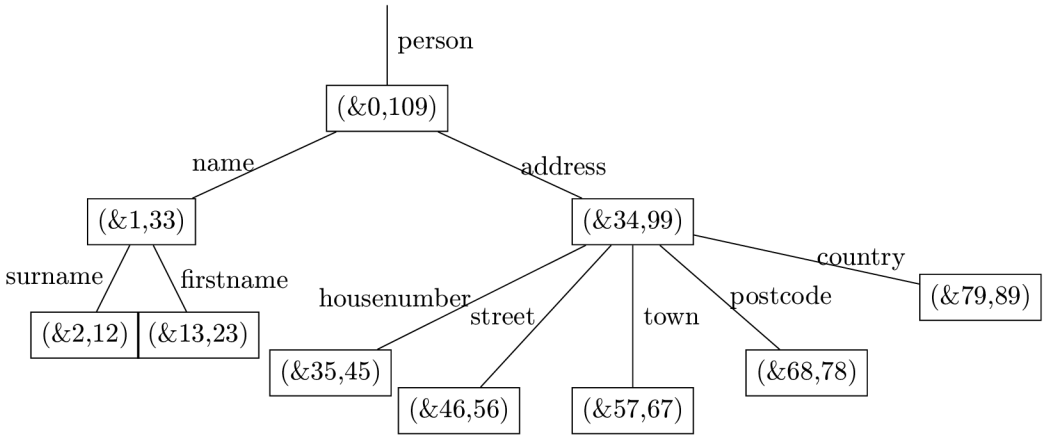
Pre/post order kódování [22] mapuje vztah předchůdce/následník mezi jednotlivými uzly dokumentu. Identifikátor uzlu je uspořádaná dvojice *(pre,post)*, kde *pre* je hodnota určená při preorder průchodu stromem a *post* je hodnota při průchodu postorder. Využívanou vlastností při tomto způsobu kódování je jednoduché určení vztahu předchůdce/následníka a to pomocí výrazu: n_2 je následníkem n_1 tehdy a jen tehdy, pokud hodnota n_1 *pre* je menší než *pre* hodnota n_2 a *post* hodnota n_1 je větší než *post* hodnota n_2.

Kódování uzlu pomocí intervalu

Identifikace uzlu pomocí kódování intervalu [33] je stejně jako pre/post order kódování založeno na průchodu stromem v pořadí preorder. Každý uzel je anotován pomocí uspořádané dvojice *(pre,size)*, kde *pre* je hodnota v pořadí při průchodu stromem preorder a *size* je ofset vůči nejvyšší hodnotě *pre* u následníků uzlu viz obrázek 4.7. V rámci optimalizace indexu pro aktualizační operace nad nosným dokumentem se přiřazují hodnoty *pre* s δ rozdílem o určité předem známé hodnotě například 10 jak ukazuje obrázek 4.8.



Obrázek 4.7: Identifikace uzlů pomocí metody kódování intervalu.



Obrázek 4.8: Optimalizovaná metoda identifikace uzlů pomocí kódování intervalu.

Sekvenční číslování uzlů

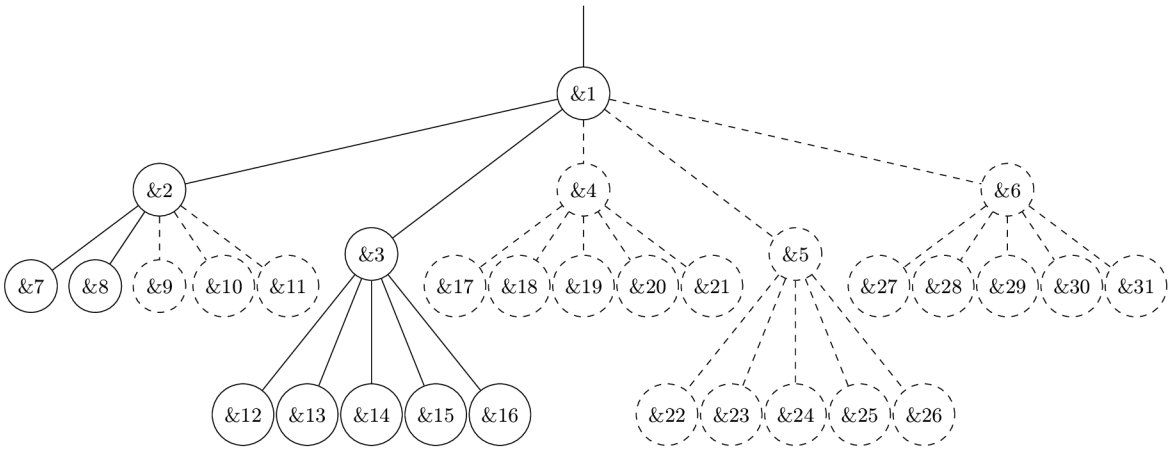
Indexační techniky nejenom v databázích často využívají sekvenčního číslování uzlů, jež vychází z návrhů definovaných v [46]. Základní myšlenka tohoto přístupu je použití uspořádaných n-tic k definici počáteční a koncové pozice uzlu v dokumentu a jeho jednoznačné identifikaci v rámci všech uzlů. Existuje tedy zobrazení jedna ku jedné a lze zpětně rekonstruovat celý XML dokument.

Začátek a konec pozice je jednoznačně determinován pozicí od počátku danou počtem předchozích slov. Tudíž je prvek identifikován trojicí `(dokument ID, počáteční_pozice : koncová_pozie, úroveň vnoření)`. Pomocí těchto trojic se dá definovat vztah typu předek-potomek následujícím způsobem: uzel x s definovaný trojicí $(D1, S1 : E1, L1)$ je potomkem uzlu y s trojicí $(D2, S2 : E2, L2)$ tehdy a jen tehdy, pokud $D1 = D2$, $S1 < S2$ a $E2 < E1$. Přes tento dostatečný matematický aparát se zmíněná varianta indexování často nepoužívá, je to zapříčiněno vysokou datovou náročností tohoto řešení.

Každý XML dokument je pak reprezentován jako strom s identifikátory, kde jejich číselná hodnota může bát dána průchodem pre-order, či post-order

Virtuální uzly

Identifikace uzlů pomocí úplného stromu, jak je prezentováno v [32], kde kóduje relaci předchůdce/následníka přímo do hodnoty aktuálního uzlu. Každý uzel je číslován tak, jako by šlo o úplný strom s aritou a. Obrázek 4.9 představuje strom s aritou $a = 5$ pro ukázkový dokument 2.1, kde přerušovaná čára identifikuje virtuální uzly.



Obrázek 4.9: Identifikace uzlů pomocí úplného stromu.

$$parent(i) = \left\lfloor \frac{(i-2)}{a} + 1 \right\rfloor \tag{4.1}$$

$$child(i) = a(i-1) + k \ + 1 \tag{4.2}$$

Definice [32] vyžaduje inicializaci čítače na hodnotu 1. Výhodou tohoto přístupu je možnost provádění lokálních změn ve zdrojovém dokumentu bez potřeby po každé této změně aktualizovat celý index. Další výhodou je snadné určení otcovského/synovského uzlu v konstantním čase pomocí těchto dvou funkcí: $parent : NodeIDs \longrightarrow NodeIDs$ pro výpočet otcovského ID a $child : NodeIDs \times \mathbb{N} \longrightarrow NodeIDs$ pro identifikaci k-tého synovského uzlu, kde $k \in \mathbb{N}$, tak jak ukazuje rovnice 4.1 a 4.2.

4.4 XML indexy

4.4.1 XLABEL

Na konferenci pgCon 2007 byl navrhnut nový XML index XLABEL tehdejším studentem Moskevské univerzity Nikolay Samokhvalovem [37]. Avšak nebyl plně implementován a je uveden pouze pro přehled, protože jeho návrh byl předlohou pro vlastní řešení.

XLABEL je rozšířením GiST viz 5.2.5, které umožňuje indexovat strukturu XML dokumentu. Jedná se o podobnou strukturu jakou je `ORDPATH` použitý u SQL Serveru 5.2.3, který místo referencí na hodnoty značek používá zkrácený identifikátor.

Dále se používá podpůrných tabulek `xnames` a `xdata`, kde první jmenovaná obsahuje atributy všech elementů obsažených v XML datech. Druhá jmenovaná obsahuje výpis všech elementů. Tyto tabulky (4.3 a 4.4) jsou specifické pro každou databázi zvlášť.

xname_id	xname_name
1	person
2	name
3	firstname
4	surname
5	address
...	...

Tabulka 4.3: Obsah tabulky `xname` při použití `XLABEL` indexu.

tid	xlabel	node_type	xname_id	value
1	a	1 (elem.)	1	null
1	a.b	1 (elem.)	2	null
1	a.b.c	1 (elem.)	3	Cimrman
...	...	...	...	...

Tabulka 4.4: Obsah tabulky `xdata` při použití `XLABEL` indexu.

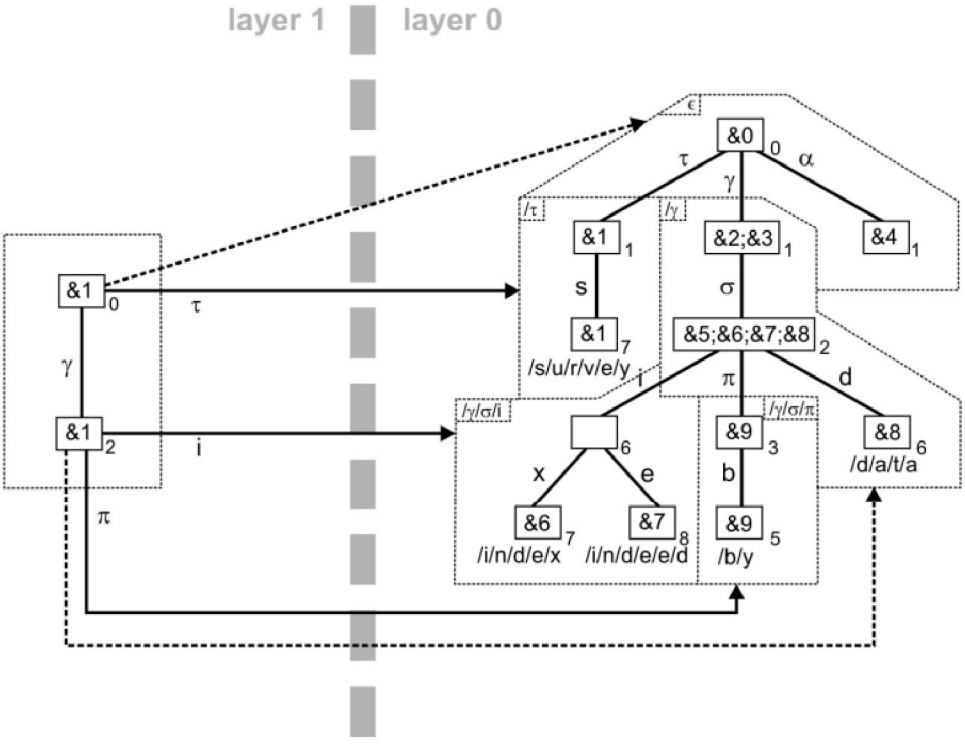
4.4.2 IndexFabric

Indexační technika IndexFabric [21] je navrhnuta pro efektivní adresaci strukturální i obsahové informace v rámci XML dokumentu. Jedná se o vyváženou více úrovňovou stromovou strukturu (obrázek 4.10), která obsahuje v rámci jednotlivých úrovní podstromy typu PATRICIA Trie (definováno v kapitole 4.2).

Nejhorší případ vyhledávání cesty v indexu způsobí výpadek maximálně 2 bloků v rámci jedné úrovně. Časová složitost je tedy lineární a je daná výškou stromu, respektive počtem úrovní, které je potřeba během zpracování indexu projít.

```
<?xml version="1.0" ?>
<book>
  <title>survey</title>
  <chapter>
    <section>index</section>
    <section>
      <paragraph>by</paragraph>
    </section>
  </chapter>
  <chapter>
    <section>indeed</section>
    <section>data</section>
  </chapter>
  <appendix />
</book>
```

Kód 4.1: Ukázkový XML dokument pro IndexFabric.



Obrázek 4.10: Struktura IndexFabric indexu pro XML dokument 4.1. Převzato z [43].

Výhodou IndexFabric je jeho stránkovací přístup, kdy jednotlivé PATRICIA Trie podstromy jsou vytvářeny do té výšky, aby se celé vešly do jedné stránky disku. Je to umožněno tím, že PATRICIA Trie podstromy jsou ve své podstatě ztrátová komprese, proto se tímto přístupem optimalizuje počet přístupů na disk na opravdové minimum. Nevýhodou tohoto přístupu je nutná následná kontrola na shodu celého dotazu.

Vkládání nového záznamu předchází prohledání tzv. *designator* tabulky, která obsahuje unikátní názvy všech elementů a přiřazuje jim speciální kód viz tabulka 4.5.

element	**kód**
appendix	α
chapter	γ
paragraph	π
section	σ
title	τ

Tabulka 4.5: *Designator* tabulka s kódy elementů pro XML dokument 4.1.

Obsah jednotlivých elementů je pro `<book><title>survey</title></book>` mapován následujícím způsobem:

1. V prvním kroku se vytvoří chybějící položky v *designator* tabulce.
2. V druhém kroku dojde k mapování na odpovídající cestu na obsah $/\tau/\mathrm{s}/\mathrm{u}/\mathrm{r}/\mathrm{v}/\mathrm{e}/\mathrm{y}$.

Vyhledání obsahu v IndexFabric je díky hybridnímu systému B-tree a PATRICIA Trie značně efektivní, ale není vhodný pro dokumentově zaměřené XML. Další nevýhodou je opomenutí indexace atributů v jednotlivých uzlech. Tyto nedostatky se snaží doplnit návrh indexu FastX v kapitole 6.1.5.

Kapitola 5

Podpora XML v databázových systémech

5.1 XML nativní databáze

Databázové systémy, které primárně operují nad XML daty se ve značné míře odlišují od standardního objektově-relačního přístupu. Stejně jako formát XML byl původně určen pro univerzální výměnu informací na internetu, tak i tyto ORDBMS v sobě často kombinují nejen databázové úložiště, ale i webový server spolu s komunikačními protokoly jakými jsou např. SOAP, XML-RPC.

Mezi hlavní představitele těchto databází dle [13] lze považovat komerční Tamino XML DB a open-source zástupce eXist Native XML Database. Důležité principy práce s XML dokumenty v těchto databázových systémech jsou rozebrány v následujících podkapitolách.

Nativní XML databázové systémy poskytují výhodu rychlejší práce s XML a to z důvodu přístupu k těmto datům přes nativní DDL (Data Definition Language). Dále tedy není třeba řešit složitá mapování XML obsahu na relační přístup, který je nutný pro SQL dotazování. V neposlední řadě odpadá nenormalizovaný přístup k datům, v případě zahrnutí XML dokumentů přímo do tabulky relačního přístupu, což odporuje již první normální formě. Většina XML dokumentů totiž není navrhována pro zpracování v databázích.

Nevýhoda použití XML nativních databází spočívá v zavedení nového systému, který se bude muset spravovat. A v neposlední řadě se tento systém musí začlenit do stávajícího IS, kde je povětšinou využíváno ORDBMS.

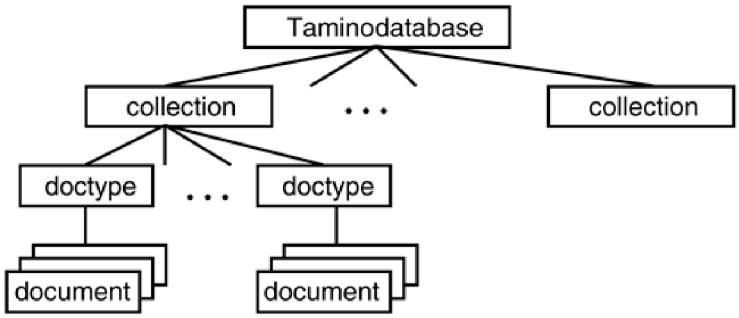
5.1.1 Tamino XML DB

Jednu z prvních nativních databázových aplikací vytvořila německá společnost *Software AG* již v roce 1999. Přístup do databáze je řešen pomocí protokolu HTTP, kdy se využívá primárně RESTfull[42] přístupu, který je blízký webovým aplikacím, pro které je tento systém navrhnut.

Tamino XML pracuje jednotně jak s datově zaměřenými XML, tak i s dokumentově zaměřenými XML. Datově zaměřené dokumenty mají pravidelnou strukturu a smíšený obsah se zde nevyskytuje. Jedná se o typ informací, které jsou obvykle uloženy v relační nebo objektově-relační databázi. Dokumentově zaměřené dokumenty se vyznačují méně pravidelnou strukturou, vyskytuje se zde smíšený obsah a je významné pořadí prvků v dokumentu. Tamino XML Server zpracovává tyto druhy XML dokumentů jednotně a je jeho cíle zpracovávat XML dokumenty efektivně bez ohledu na jejich strukturu. Dále Tamino XML

umožňuje ukládat jiné typy dat (např. obrázky, HTML soubory, atd.), které jsou relevantní pro webové informační systémy.

Perzistence XML dat v Tamino database je řešena způsobem znázorněným na obrázku 5.1. Kde databáze se skládá z množství tzv. kolekcí. Tyto kolekce definují vazby mezi skupinami dokumentů, kdy každý dokument je uložen pouze jednou a může se vyskytovat pouze v jedné kolekci. Kolekce má svou sadu popisů XSD. V každém XSD schématu může být definováno pomocí Tamino-specifické notace rozšíření oblasti W3C XML Schema (`appinfo element`). Dále `doctype` identifikuje jeden z globálních prvků deklarovaných ve W3C XML Schema na kořenový element. V kolekci je každý dokument uložen jako člen právě jedné `doctype`.



Obrázek 5.1: Způsob uložení XML dokumentů v DB firmy *Software AG*. Převzato z [13].

Validační schémata, jež mohou být přiložena k XML souboru, jsou uložena v samostatném bloku a je na ně veden odkaz z XML dokumentu, který ho vyžívá. V některých případech bývá uložena pouze část těchto schémat, kterážto pak validují pouze definovanou podčást. Tyto schémata ale nejsou striktně vyžadována pro práci nad XML dokumenty.

Primárním způsob dotazování je *XQuery* [15], jakožto jedné z norem W3C pro práci nad XML. Databáze podporuje základní principy ACID (atomicity, consistency, isolation, durability), spolu s transakcemi.

Indexace XML v Tamino XML DB

Indexy jsou nepostradatelné v databázových systémech, jinak se totiž nelze efektivně dotazovat nad velkým množstvím dat. Tamino XML Server podporuje tři typy indexů, které jsou udržovány i v průběhu typicky databázových operací `INSERT, UPDATE, DELETE`. Jelikož se jedná o komerční řešení, tak není známa kompletní specifikace použitých indexů, pouze základní přehled použitých technik.

Standardně používaný index je hashovacího typu mapující hodnotu na klíč, stejně jako v relačních databázích. Slouží k rychlému vyhledávání při dotazu na konkrétní elementy, či atributy, který mají určité hodnoty (např. pro dotazy hledající všechny knihy od autora s příjmením Hruška). Všechny indexy jsou definovány v referenčním schématu Tamino, opět s využitím rozšíření W3C XML Schema.

Indexy jsou typované, tudíž pro číselné hodnoty je vrácen správný výsledek (tj. $5 < 10$), indexy textového typu jsou seřazeny lexikograficky (”5”>”10”).

Textové indexy jsou nutným ale nedostačujícím předpokladem pro efektivní full-textové vyhledávání. Indexace slova obsaženého v elementu nebo atributu je prováděna tak, aby se zrychlilo jeho další zpracování během dotazování. Textové indexy nejsou definovány pouze na listových elementech, ale také na nadstromech, tudíž lze efektivně dotazovat podstromy,

či celý dokument.

Výpočet textového indexu je založen na výsledcích jež vrací volání funkce `text()` definované v XPath normě [20]. Tento výsledek je rozdělen na tokeny a každý token je zvlášť zahrnut do indexu. Tokenizace je netriviální úkol a je třeba brát v potaz jazyk pro který se rozdělení používá. Tamino XML Server obsahuje vlastní sadu těchto slovníku.

Problém textových indexů je při lexikografickém řazení pro národní sady, jelikož tato funkcionalita není W3C definována, tak se používá obecně známý způsob řazení, který se definuje pomocí metadat.

K dispozici je také XML-specifický typ indexu, tzv. index struktury. Existují dvě verze tohoto indexu. První verzí je komprimovaná konstrukce indexu, která udržuje informace o všech cestách, které se vyskytují pro každý element. To může v mnoha případech urychlit provádění dotazu (např. pro dokumenty bez souvisejících schémat, nebo když je v dotazu použit `xs:anyAttribute`).

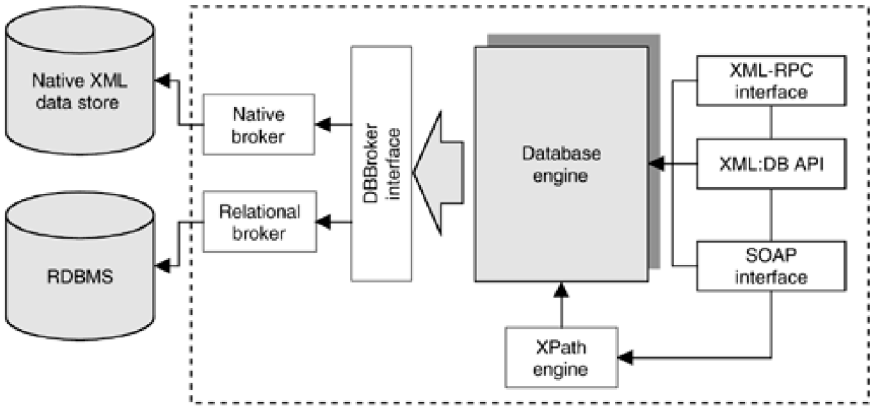
Druhá verze indexu udržuje úplné záznamy nejen pro existenci cesty v `DOCTYPE`, ale vede odkazy na všechny dokumenty, ve kterých existuje stejná cesta. Této vlastnosti lze využít při optimalizaci dotazu, který požádá o nepovinné části `DOCTYPE` (např. prvky nebo atributy s `minOccurs = 0`, případně celý podstrom). Tento strukturální index také pomáhá při změně schématu a to tím způsobem, že platnost provedených změn lze vyhodnotit bez případné kontroly nad celým `DOCTYPE`.

Pokud neexistuje ani jeden index nad danou kolekcí XML dokumentů, může to vést až k sekvenčnímu prohledávání a tím se velmi znatelně degraduje výkonnost. Tamino XML standardně nevytváří indexy nad danou kolekcí XML dokumentů.

5.1.2 eXist

Open-source alternativa *eXist* byla založena v roce 2001. Její implementace je úzce svázaná s *Apache Cocoon* projektem a je implementována čistě v jazyce Java. Používá stejné prostředky dotazování jako *Tamino database*, ale přidává vlastní rozšíření, která svou syntaxí připomínají SQL.

Největší odlišnost oproti *Tamino database* je v plné podpoře relačního přístupu, kde jsou udržovány 2 úložiště jak ukazuje obrázek 5.2. Existence vlastního úložiště pro XML přináší výhodu v systematickém přístupu k datům různých typů.



Obrázek 5.2: Architektura ORDBMS *eXist*. Převzato z [13].

Databáze eXist byla navržena tak, aby prováděla co nejrychleji XPath dotazy a vyžívala při tom v největší možné míře indexy a to pro všechny elementy i atributy. Velké množ-

ství dotazů lze díky těmto vlastnostem zodpovědět pouze pomocí dat uložených v indexu. Vlastní uzly jsou zpracovávány pouze v nezbytných případech, jako je např. při zobrazení výsledků dotazu.

Obdobně jako u jiných nativních systémů umožňuje eXist ukládání dokumentů XML bez připojeného schématu pro validaci. Jedinou podmínkou je, aby dokumenty byly validní z pohledu XML syntaxe.

Vnitřní struktura uložených XML dokumentů kopíruje adresářovou strukturu známou ze souborových systémů. Kolekce dokumentů mohou být libovolně vnořeny a jelikož není vyžadováno schéma, tak daná kolekce může obsahovat XML dokumenty různých datových tříd.

Databázový systém eXist byl primárně navrhnut pro podporu dokumentově zaměřených XML. Dokumentově zaměřená XML jsou obvykle lépe čitelná uživateli a obsahují ve velké míře smíšený obsah spolu s delšími úseky textu bez potřeby složitě zpracovávat údaje v nich uložené. Naproti tomu je dotazování nad těmito dokumenty problematické a to z důvodu slabé selektivity pomocí jazyka XPath. Databáze eXist tak nabízí celou řadu rozšíření která řeší problém fulltextového dotazování. Využívá se pomocná indexovací struktura, která udržuje informace o jednotlivých slovech. Pro operace nad fulltextovým indexem jsou definovány speciální operátory pracující nad uzly, stejně jako nad hodnotami atributů.

Indexace XML v eXist

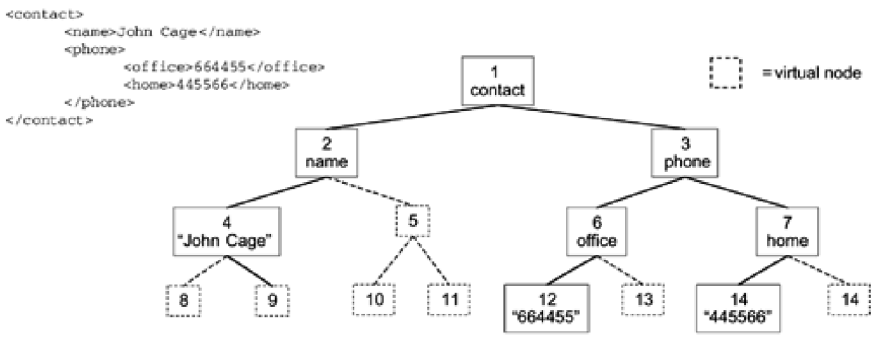
Otevřenost zdrojových kódů systému eXist přináší obrovskou výhodu ve znalosti použitých technik. Nejvíce se to projevuje při studiu implementace indexů nad XML dokumenty. Používají se hierarchické indexy obdobně jako Tamino XML DB 5.1.1 a přidává se numerické číslování pro identifikaci jednotlivých uzlů XML v indexu. Index tedy odkazuje nejen na aktuální DOM uzly v XML dokumentu, ale poskytuje také rychlou identifikaci všech možných vztahů mezi uzly ve stromové struktuře dokumentu (např. předchůdce-potomek aj.). Na základě těchto vlastností lze za pomocí optimalizátoru urychlit algoritmy určené pro vyhodnocení XPath výrazů. Konvenční přístupy, použité např. v knihovně LibXML, jsou obvykle založeny na sekvenčním průchodu stromem a to buď shora-dolů, nebo zdola-nahoru. Bylo prokázáno, že využití těchto indexačních technik znatelně urychluje dotazování, viz důkaz v publikaci [1].

Indexace je aplikována na všechny uzly v dokumentu a to včetně elementů, atributů ale i textového obsahu spolu s komentáři. Na rozdíl od jiných přístupů, není nutné explicitně vytvářet indexy, protože všechny indexy jsou řízeny automaticky DBMS. Je však možné omezit automatické vytváření fulltextového indexování vymezením pouze naa specifické části dokumentu.

Indexace v eXist DB je inspirována [32]. Jedná se indexovací model, který je založený na k-nární stromové struktuře dokumentu, kde k je rovno maximálnímu počtu podelementů k danému uzlu. Je tedy zřejmé (viz diagram 5.3), že každý nelistový uzel ve stromu má stejný počet podelementů. Číselný identifikátor je přiřazen každému uzlu v pořadí v jakém je uveden dle jednotlivých úrovni stromu. Pro libovolný uzel s méně než k podelementy platí, že všechny virtuální podřízené uzly odkazují na prázdnou hodnotu. Jedná se o nadbytečné uzly s identifikátory, které jsou ale nezbytné pro správné vyvážení úplného k-nárního stromu.

Takto generovaný identifikátor má několik důležitých vlastností. V prvé řadě lze vždy z hodnoty identifikátoru vypočítat ID svého rodiče, sourozence, a (možného virtuálního) podelementu.

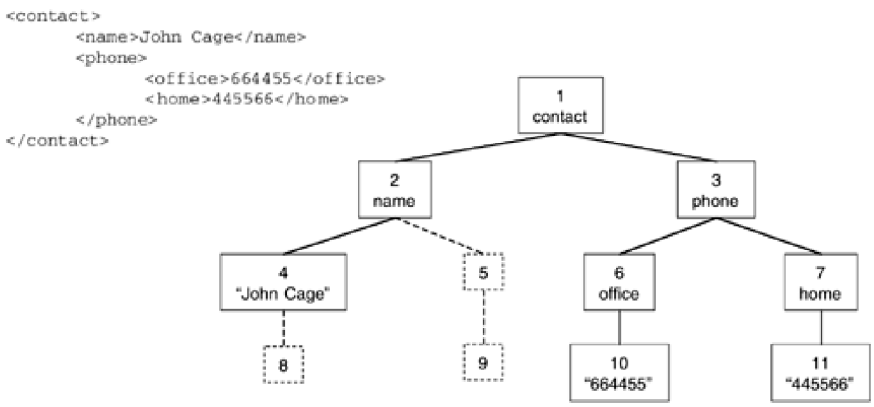
Implementace DOM operací určených pro práci nad každým uzlem je pak snadná. Na-



Obrázek 5.3: Úplný k-nární strom použitý pro indexování v databázi *eXist*. Převzato z [13].

příklad pro určení rodiče daného uzlu se jednoduše spočítá identifikátor rodiče, který je následně použit k získání skutečné hodnoty rodičovského uzlu. Všechny osy navigace jsou podporovány stejně jak to požaduje W3C DOM norma [11]. Významně se tím snižuje potřebný prostor pro indexaci jednoho uzlu. Nemusí se totiž ukládat odkazy na rodiče, sourozence a podřízené uzly, protože tato informace je implicitně obsažena v generovaném identifikátoru. Velikost indexu ses pohybuje pro každý prvek mezi 4 a 8 byty v závislosti na maximálním počtu podelementů.

Nedostatkem tohoto přístupu je ovšem rychlé zvyšování hodnoty klíče indexu, takže se velmi rychle vyčerpá stavový prostor daný 4 byty. Dále existuje omezení dané maximální velikostí XML dokumentu, který lze indexovat a to ze stejného důvodu vyčerpání stavového prostoru. V praxi totiž mnoho dokumentů obsahuje podstromy, které jsou dosti nevyvážené. Například typický článek má omezený počet nejvyšších úrovní prvků, jakými jsou kapitoly a oddíly, zatímco většina uzlů je tvořena z odstavců. V nejhorším případě, kdy jeden uzel v některé hluboce stromu dokumentu má maximální počet podelementů, pak musí být doplněny na všech úrovních stromu virtuální uzly pro splnění omezení úplnosti. Typický příklad jsou identifikátory generované pro kompletní 10-narní strom o výšce 10, který se nevejde do 4 byte integeru a jedná se přitom o velmi malý XML dokument.



Obrázek 5.4: Hybridní stromová struktura použitá pro indexování v databázi *eXist*. Převzato z [13].

Tento nedostatek odstraňuje rozšíření, kdy se nepoužívá úplného k-nárního stromu, ale počet uzlů v jednotlivých úrovních je determinován maximálním počtem podelementů v dané úrovni. Dále je nutné pro každou úroveň uchovávat tabulku maximálního počtu

podelementů, kterou obsahuje a pak lze obdobným způsobem jako u úplného k-nárního stromu počítat požadované indexy viz diagram 5.4.

5.2 Práce s XML v ORDBMS

Nedlouho poté co se stalo XML populární, začaly renomované databázové společnosti přidávat podporu do svých produktů. Dnešní moderní ORDBMS by nebyla na trhu konkurenceschopná, pokud by nepodporovala XML formát. Hlavním úkolem je tedy využít dosavadního přístupu ORDBMS spolu s ACID transakcemi, CRUD a standardními programovacími API (např. JDBC, ODBC) a přidat podporu pro XML. V neposlední řadě využít potenciálu XML v strukturování dat pro inter-business komunikaci mezi informačními systémy.

Komerční databázové systémy mají podporu XML na velmi dobré úrovni jak ukazuje tabulka v příloze C.1. Avšak většina systémů stále obsahuje nedostatky v implementaci SQL:2003 normy obzvláště v sekci věnované XML.

Devizou moderních ORDBMS je jejich vysoká penetrace a škálovatelnost. Naproti tomu vyvstává otázka zpracování strukturovaných dat jakými je XML v rámci relačního modelu a zajistit přitom dostatečnou robustnost a rychlost. Využívá se různých technik, které jsou popsány v následující podkapitole. V příloze C.1 je přehledová tabulka práce s XML datovým typem u diskutovaných komerčních databází.

5.2.1 Oracle 11g

Oracle jakožto jeden z nejznámějších a zároveň největších poskytovatelů databázových serverů, zavedl podporu formátu XML již od verze 8i v roce 1999. V nejnovější verzi 11g, ale stále dle [9] neposkytuje úplnou podporu normy SQL:2003.

Podpora XML dokumentů je zde implementována dvojím způsobem. Kde první je použití datového typu XMLType, nebo využít samostatný datový sklad pro správu XML dat. Každý přístup má své výhody a omezení, které jsou reflektovány níže.

XMLType

XMLType používá pro vnitřní reprezentaci CLOB, případně objektově-relační přístup. Od verze 11g je přidána podpora pro binární formát. CLOB poskytuje výhodu zachování identické reprezentace a to včetně whitespace znaků, zatímco v druhém případě se reprezentace blíží DOM. Objektově relační reprezentace vytváří neviditelné sloupce, ve kterých jsou uloženy všechny potřebné informace pro zpětnou rekonstrukci XML dokumentu. Tyto metadata ale nejsou přirozená pro relační model. Přístup k těmto sloupcům je zajištěn přes poskytované API. Je možné změnit typ uložení dat z objektově-relačního na CLOB (případně obráceně) bez dopadu na existující kód aplikace.

Indexační metody použité pro XMLType se liší dle zvoleného typu uložení. Pro CLOB se vytváří klasický textový (případně fulltextový) index, který je přizpůsobivý změně schématu a umožňuje spravovat dokument přesně tak, jak byl pořízen. Naproti tomu objektově-relační přístup používá modifikaci B-stromu, kde existuje silná vazba mezi použitým schématem a XML dokumentem, dále je vypuštěno odřádkování a formátovací znaky.

Mezi další výhody datového typ XMLType patří podpora XML schémat umožňující validaci vložených dokumentů. Dále částečný XML update, kdy lze specifikovat konkrétní elementy a atributy, které mají být modifikovány. V neposlední řadě lze s XMLType pra-

covat jako s objektem (definovat konstruktor, dědit), ale také vytvářet celé tabulky typu XMLType.

XML Repository

Sklad dokumentů pro správu umožňuje vedle vlastnických práv přiřadit vlastní seznam přístupových práv, které zvyšují granularitu omezení přístupů. Dále lze namapovat adresářovou strukturu do databáze a řídit přístup k XML dokumentům tímto způsobem. V neposlední řadě existuje podpora pro hierarchické indexy umožňující rychlé prohledání cesty v rámci XML dokumentu, případně cesty k danému souboru. Tyto vlastnost spolu s využitím JNDI a přístupem pomocí Servletů umožňuje snadné provázání na technologii Java EE, kterou firma Oracle v roce 2009 koupila spolu se společností SUN.

Dotazy nad oběma způsoby správy XML dokumentů lze provádět pomocí XPath dotazů a od verze 10g Release 2 i pomocí XQuery rozšíření. Množství dalších funkcí pro práci s XML je obsaženo v XML Developer's Kit (XDK), které umožňuje provázání s nejznámějšími procedurálními jazyky jako je Java, C++, ... spolu s PLSQL (procedurálnímu rozšíření SQL v systému Oracle).

5.2.2 IBM DB2 9.7

Druhá nejrozšířenější databázová platforma obsahuje podporu XML pomocí dvou nástrojů a to starší XML Extender a nahrazující pureXML.

pureXML

XML dokumenty ukládá ve stromové struktuře, která odpovídá XML datovému modelu. Nepoužívá tedy žádného mapování XML na relační struktury, ale vytváří nový datový typ XML, stejný jako všechny ostatní SQL typy. Dále definuje společné úložiště pro validační schémata, která mohou být následně použita pro kontrolu vstupních XML dokumentů.

Hybridní systém dotazování nad XML

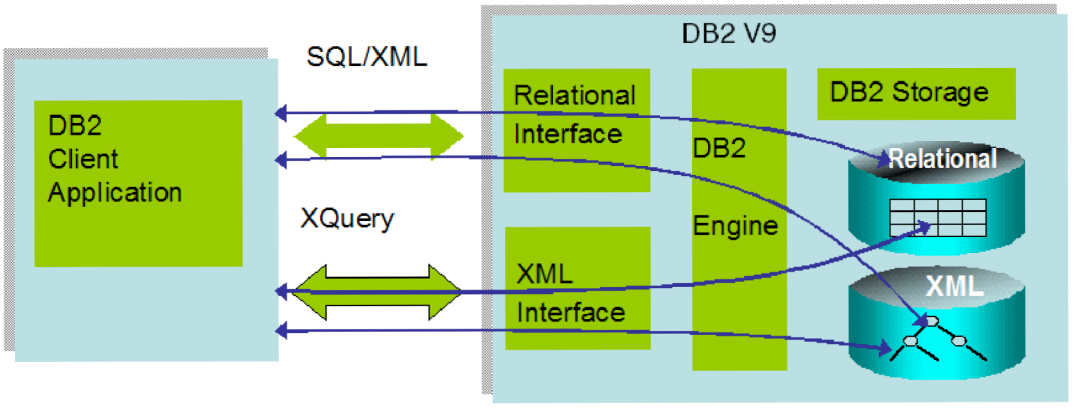
Databázový systém DB2 používá hybridního způsobu dotazování nad XML dokumenty. Jedná se o vzájemné provázání SQL a XQuery viz obrázek níže 5.5. Dotaz typu SQL má odlišný parser od dotazu typu XQuery, ale používají shodný exekuční plán, který je z těchto definic vytvořen.

Indexace XML

Index se vytváří nad konkrétním sloupcem obsahujícím XML data. Nelze vytvořit index používající více sloupců, byť by všechny byly správného typu. Definice je přípustná pouze pro specifikovaný seznam XPath. Při vytváření indexu se zadaný seznam XPath vyhodnotí a vytvoří se odkazy pro rychlou navigaci v rámci hierarchické struktury XML.

DB2 9 také podporuje full-textový index na XML dokumentem využívající již stávající implementaci pod názvem Net Search Extender.

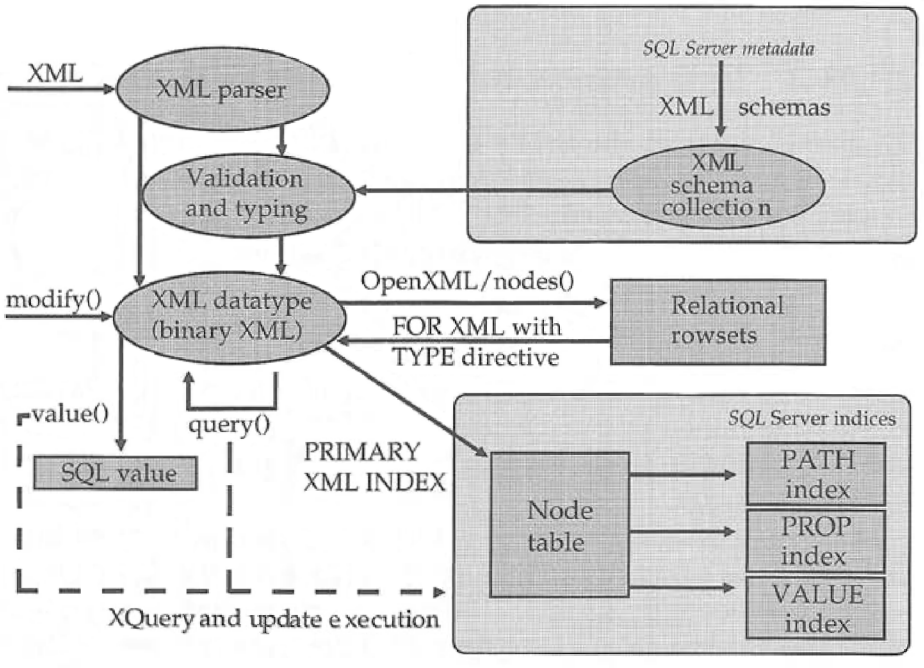
Množství podpůrných nástrojů pro práci s XML je na srovnatelné úrovni jako u Oracle. Podporuje běh na Unix-like operačních systémech, ale i na Windows.



Obrázek 5.5: Integrovaný způsob dotazování pomocí XQuery a SQL v DB2. Převzato z [19].

5.2.3 Microsoft SQL Server 2008

Nejrozšířenější databázový systém na platformě Windows podporuje práci s XML od verze SQL Serveru 2000. Avšak až ve verzi SQL Server 2005 přidal podporu XML datového typu. Viz obrázek 5.6, zobrazující schéma práce SQL Serveru s XML dokumenty.



Obrázek 5.6: Architektura pohled na podporu XML v SQL Serveru 2008. Převzato z [13].

Pro dotazování je použit jazyk XQuery 1.0 případně XPath 2.0 a obdobně jako u databázových systémů DB2 a Oracle 11g, tak i SQL Server udržuje kolekci validačních schémat (DTD, XSD) v centrálním úložišti dostupném pro všechny uživatele.

Indexace XML

Podporu indexace XML dokumentů funguje naprosto odlišným způsobem na rozdíl od již zmiňovaných databázových systémů. Hlavním důvodem odlišného zacházení je interní reprezentace XML. SQL Server totiž mapuje hierarchickou strukturu XML do relačního modelu. Výhodou tohoto přístupu je následné dotazování pomocí standardních dotazovacích technik, pro které je databázové úložiště nejlépe optimalizované.

Následující tabulka 5.1 zobrazuje zjednodušený náhled na způsob reprezentace XML indexu.

PK	ORDPATH	TAG	NODE	NODETYPE	VALUE	HID
1	1	1 (Resume)	ELEMENT	12 (Resume)	null	#Resume
1	1.1	2 (Name)	ELEMENT	13 (NameT)	null	#Name#Resume
1	1.1.1	3 (Prefix)	ELEMENT	2 (xs:string)		#Prefix#Name#Resume
1	1.1.3	4 (First)	ELEMENT	2 (xs:string)	Jaroslav	#First#Name#Resume
1	1.1.5	5 (Middle)	ELEMENT	2 (xs:string)	da	#Middle#Name#Resume
1	1.1.7	6 (Last)	ELEMENT	2 (xs:string)	Cimrman	#Last#Name#Resume
1	1.1.9	7 (Suffix)	ELEMENT	2 (xs:string)		#Suffix#Last#Name#Resume

Tabulka 5.1: Zjednodušená pohled na XML index v SQL Serveru.

Význam jednotlivých sloupců:

- `PK` je primární klíč odkazující na vlastní XML dokument
- `ORDPATH` reprezentuje strukturu XML dokumentu
- `TAG` je hodnota značky obsažené v elementu
- `NODE` číselná reprezentace uzlu (v závorce obsažená hodnota)
- `NODETYPE` typ uzlu (element, atribut)
- `VALUE` hodnota uzlu
- `HID` reprezentace cesty v XML stromu pomocí tokenů

Vlastní index vychází z návrhu definovaného v [35].

XML index má několik omezení. Prvně ho nelze vytvářet nad pohledy, či proměnnými. Lze vytvořit pouze jeden a to primární index nad daným sloupcem obsahujícím XML dokument.

Sekundární index nad XML dokumentem

Pokud je známa přesná podoba XQuery dotazu, tak lze urychlit vykonání SQL plánu pomocí sekundárního indexu. Sekundární index pracuje nad primárním XML indexem, používá sloupce `PK`, `ORDPATH`, `VALUE` a `HID`. Sekundárních indexů může být více a mohou se vzájemně doplňovat například vytvořením indexů pro dotazování hodnot, atributů, nebo optimalizaci XPath. Pokud je třeba zrušit primární XML index, musí se nejdříve odstranit všechny navázané sekundární indexy. Je to způsobeno silnou závislostí sekundárních XML indexů, které obsahují odkaz na primární index a došlo by tím k porušení integrity dat.

Full-text index

Obdobně jako Oracle přidává SQL Server podporu pro full-textový index nad XML dokumenty. Používá vlastní slovník pro určení hranic jednotlivých slov a jazykové odlišnosti.

Podpůrné nástroje v podobě API pro práci s XML jsou poskytovány hlavně pro platformu .NET. Z komerčních databázových systémů implementuje nejmenší část SQL:2003 normy, ale poskytuje největší optimalizaci běhu na Windows, jakožto nejrozšířenější běhové platformě, čímž se snaží tento nedostatek vyvážit.

5.2.4 MySQL 6.0

Jakožto jeden z hlavních představitelů open-source databázových systémů neposkytuje velké množství XML funkcionality. Nepodporuje nativní XML typ a pracuje se všemi XML dokumenty jako s textem. Z tohoto přístupu plyne celá řada optimalizačních problémů v podobě nevhodného indexačního přístupu nereflektující hierarchickou strukturu XML.

Podporovaná funkčnost přidaná od verze 5.1 je následující:

- Export a následný import databáze do XML dokumentu.
- Základní operace pomocí XPath, jmenovitě `ExtractValue(), UpdateXML()`
- Použití externí knihovny `lib_mysqludf_xql` pro formátování výsledků dotazů do XML.

Chybějící podpora dotazování XQuery a veškeré další manipulace s XML dokumenty jsou tudíž prováděny v business logice programu.

5.2.5 PostgreSQL 9.03

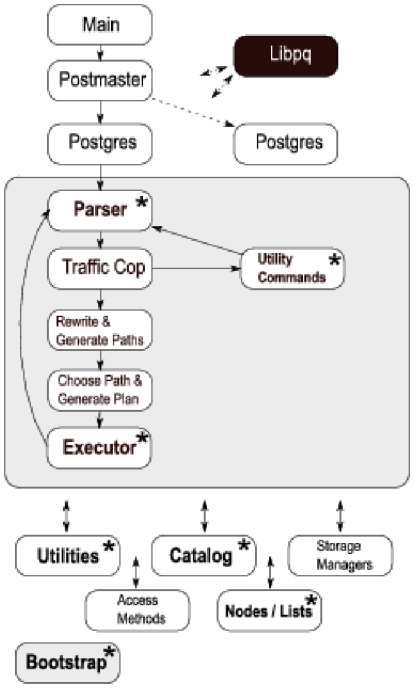
PostgreSQL je jedním z nejstarších open-source databázových systémů. Byl vyvinut profesorem Stonebrakerem na univerzitě Berkeley v Kalifornii. Jeho přímým předchůdcem byl databázový systém Ingres, který sloužil pro studijní účely. Následně byl jeho kód uvolněn a byl přejmenován na Postgres z důvodu zdůraznění post relačního přístupu k datům. Po celém světě má aktivní komunitu vývojářů přesahující 1000 členů a aktivně komunikujících uživatelů na `irc #postgresql` je přes 500. Podstata komunitního vývoje posouvá vývoj rychle kupředu, ovšem má i své stinné stránky v podobě zdlouhavé disputace nad implementačními problémy. Nelze totiž vyhovět všem zúčastněným stranám a tak PostgreSQL podporu pouze částečně normu SQL:2003.

V průběhu vývoje bylo třeba upravit jednotlivé moduly spolupracující v průběhu provádění dotazu nad XML. Změněné moduly jsou označeny hvězdičkou viz obrázek 5.7.

Podpora práce s XML dokumenty

Částečná podpora datového typu XML byla přidána ve verzi 8.3. Původně byla přidána sada XPath a XSLT funkcí pracující nad datovým typem VARCHAR. Následně byl přidán zmíněný datový XML typ jako logické rozšíření datového typu VARCHAR/TEXT. V pozdějších verzích se přešlo na vnitřní reprezentaci pomocí CLOB.

Většina funkcí z balíku SQL/XML jsou již implementovány. Jmenovitě `XMLPARSE, XMLSERIALIZE, XMLELEMENT, XMLATTRIBUTES, XMLFOREST, XMLAGG, XMLCONCAT, XMLCOMMENT, XMLPI, XMLROOT`, kde výše jmenované funkce transformují relační model dat uložených v databázi na XML formát (popis funkcí byl uveden v kapitole 3).



Obrázek 5.7: Zpracování dotazu a jednotlivé moduly, které bylo třeba upravit pro přidání podpory XML datového typu a SQL/XML funkcí. Převzato z [23].

Pro dotazování nad XML se používá sada XPath, XQuery není v současné chvíli podporováno. Hlavní příčinou tohoto stavu je chybějící implementace v knihovně LibXML, která zaštiťuje veškeré operace nad XML dokumenty.

Některé podpůrné funkce zatím chybějí. Jmenovitě validace pomocí validačních schémat není implementována stejně tak chybí podpora datových typů pro DTD, XSD i RelaxNG. Dále chybí XQuery data model [36].

Indexace XML

Vkládání a dotazování celého XML dokumentu je díky textové reprezentaci extrémně rychlé, ovšem u vyhledávání a práce s částmi dokumentů tomu tak není, proto byly zavedeny základní typy indexů.

Nejjednodušším indexem je B-tree index definovaný nad XPath funkcí viz kód 5.1. Při vytvoření tohoto indexu odpadá potřeba parsování XML a vyhodnocení XPath výrazu, čímž se znatelně urychlí celý dotaz. Nevýhodou je nutná dopředná znalost XPath výrazu. Byl navrhnut i další typ index viz kapitola 4.4.1, ale ten nebyl plně implementován.

```
CREATE INDEX xpath_index
       ON users
       USING btree(
              xpath_array(xmldata, '//address/@town')
       );
```

Kód 5.1: Vytvoření B-tree indexu nad XPath výrazem. Převzato z [23].

Full-text index nad XML dokumenty je implementován pomocí modulu `tsearch2`. Pro plnou podporu je ale nutné nejdříve vybrat podčásti, které se mají indexovat. Pro výběr těchto pod částí lze použít XPath výraz. Dále lze jednotlivým slovníkovým výrazům přiřadit rozdílné váhy a tím některé upřednostnit při výběru.

GiST

Abstraktní stromová struktura (Generalized Search Tree) [12] je podpůrná datová struktura, která se v PostgreSQL používá při vývoji nových indexů nad různými datovými typy.

Jedná se vyvážený strom obsahující v každém uzlu dvojici `(key, pointer)`, kde `key` je hodnota specifická pro zadaný ADT. Vlastní implementace využívající tohoto generického přístup jsou například: B-tree, R-tree (používán při prostorových datech v PostGIS), RD-tree (indexuje množinu položek např. při full-textovém indexu).

GIN

GIN (Generalized Inverted Index) je stejně jako GiST podpůrná datová struktura, která ovšem pracuje na bázi invertovaného souboru (definován v kapitole 4.2). V rámci indexu je ukládána dvojice *(key, seznam_vyskytu)*, kde klíč je volitelný a plně závisí na implementátorovi jakou zvolí funkci a *seznam_výskytu* obsahuje id řádků na nichž se klíč vyskytuje. Klíč nemusí být unikátní a stejné položky v seznamu se mohou vyskytovat v rámci různých seznamů.

Výhodou tohoto generického přístupu je již existující akcelerace vyhledávání a tudíž stačí implementovat pouze několik potřebných funkcí pracující nad klíči a seznamem výskytů.

Interní vlastnosti úložiště

XML data jsou v PostgreSQL ukládána jako TOAST (The Oversized Attribute Storade Technique), pro kterou je poskytována sada funkcí pro převody mezi touto datovou strukturou (uzpůsobenou pro optimální uložení na disku) a ADT (Abstract Data Type) definovanými v databázi. Maximální velikost je při současné implementaci 1 GB.

Kapitola 6

Návrh implementace rozšíření do PostgreSQL 9.03

Tato kapitola se zaměřuje na popis použitých technik a problémů, které se během implementační fáze vyskytly. Nejedná se však o úplný výpis všech zdrojových kódů. Pro lepší pochopení souvislostí lze nahlédnout do programové nápovědy, která je začleněna v rámci dokumentace k PostgreSQL. Nachází se na přiloženém CD v adresáři `/postgresql-9.0.3/doc/src/sgml/html/functions-xml.html` pod heslem `xml_validate`. Případně lze nahlédnout přímo do zdrojového kódu XML validace a indexu FastX, který je umístěn ve složce `postgresql-9.0.3/contrib/xml2`.

Jako první část implementace chybějící funkčnosti jsem zvolil přidání podpory validace XML dokumentů vůči DTD, XSD a RelaxNG. Podrobným studiem interní dokumentace PostgeSQL byla zjištěna silná závislost na knihovně LibXML, která je používána pro většinu operací nad XML daty. Tato závislost determinovala další postup vývoje, kdy v rámci studia programového API knihovny, byla zjištěna úroveň podpory jednotlivých validačních schémat a následně způsob použití.

Knihovna LibXML poskytuje několik přístupů zpracování vstupního XML dokumentu. Z pohledu validace je optimální varianta používající vstupně/výstupní abstrakci vstupu (stream). Výhodou tohoto přístupu je nízká paměťová náročnost, způsobená odstraněním nutnosti načítat celý XML dokument do hlavní paměti. Na druhou stranu je hlavní nevýhodou chybějící podpora některých podstatných částí ve zpracování validace (hlavně u RelaxNG), proto byl zvolen klasický způsob práce s XML dokumentem. Jedním z argumentů pro toto rozhodnutí byla politika PostgreSQL, která poskytuje podporu nejen pro nejnovější verzi databázového systému, tudíž nasazené instance neobsahují potřebnou verzi LibXML. Dalším důvodem byla rychlost začleňování nových verzí LibXML do hlavních distribucí Linuxu, která je značně restriktivní.

Stávající implementace PostgreSQL kontroluje při překladu ze zdrojových kódu přítomnost knihoven LibXML. Vlastní kontrola je zajištěna pomocí skriptu `configure` a v případě nenalezení knihovny LibXML v potřebné verzi nedojde k definici konstanty `USE_LIBXML`. Tato konstanta je kontrolována pomocí makra během překladu, což má za následek vypuštění zdrojového kódu závislého na LibXML viz příloha D.1.

V případě chybějící knihovny, případně překladu bez podpory LibXML budou sice všechny funkce pracující nad XML fungovat, ale v omezeném režimu. Vlastní datový typ XML bude plně funkční, jelikož při jeho importu/exportu se knihovna LibXML nepoužívá.

Jednou z problematických částí implementace byla rozdílná práce s textovými literály

v rámci PostgreSQL a LibXML, která je dána samotnou normou XML [18], jež předpokládá použití UTF-8 datového typu. LibXML se této normy striktně drží a používá vlastní datový typ `XMLCHAR` nahrazující `char` používaný v jazyce `C`. Tento přístup přináší komplikace v případě jiného kódování v rámci databáze než je UTF-8.

Navržené řešení spočívá v implementaci konverzní funkce `pg_do_encoding`, jak ukazuje kód 6.1, který konvertuje `char*` na `XMLCHAR*` v UTF-8 kódováním. Vlastní konverzní funkce je součástí zdrojových kódů PostgreSQL a pokud má dojít ke konverzi mezi stejnými kódováními, tak je funkce ukončena s návratovou hodnotou odpovídající nepřekódovanému vstupu.

```
utf8xsd = pg_do_encoding_conversion(xsd,
        lenxsd,
        GetDatabaseEncoding(),
        PG_UTF8);
```

Kód 6.1: Ukázka volání konverzní funkce.

Systémový katalog u PostgreSQL je navržen s ohledem na budoucí snadné rozšíření. Bylo tedy snadné zakomponovat nové funkce implementované v programovacím jazyku `C` (viz ukázka části kódu v příloze D). SQL norma z roku 2008 definuje přesný formát jaký musí validační funkce splňovat (viz gramatika 6.2).

```
<XML validate> ::=
  XMLVALIDATE <left paren>
    <document or content or sequence>
    <XML value expression>
    [ <XML valid according to clause> ]
    <right paren>
<document or content or sequence> ::=
  <document or content>
  | SEQUENCE
```

Kód 6.2: Gramatika volaní validační funkce dle ISO normy [5].

6.1 Validace XML dokumentů

6.1.1 Vytvoření nových datových typů DTD, XSD, RelaxNG

Pro potřeby validace byly vytvořeny nové datové typy reprezentující validační schémata. Dle [23] lze definovat nový datový typ složením z již existujících datových typů, viz gramatika pro vytvoření nového datového typu 6.3.

```
CREATE TYPE name AS
    ( attribute_name data_type [, ... ] )

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )
```

```
CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
)

CREATE TYPE name
```

Kód 6.3: Vytvoření nového datového typu.

Pro potřeby složitějších datových typů, jakými jsou validační schémata, je třeba definovat potřebné konverzní a vstupně/výstupní funkce v jazyce `C`. Funkce mají speciální hlavičku (viz D.1).

```
PG_FUNCTION_INFO_V1(dtd_in);
PG_FUNCTION_INFO_V1(dtd_out);

Datum dtd_in(PG_FUNCTION_ARGS);
Datum dtd_out(PG_FUNCTION_ARGS);
```

Kód 6.4: Hlavičky vstupně/výstupních funkcí pro typ DTD.

V rámci zdrojových kódů PostgreSQL se vyskytuje velké množství definic pomocných maker. Tyto preprocesorové konstrukce zdrojový kód do jisté míry znepřehledňují a tím prodlužují dobu nutnou pro celkové pochopení všech procesů probíhajících na pozadí. Na druhou stranu se tímto způsobem deklarací zapouzdřují platformě závislé operace a unifikují přístupové metody k datovým reprezentacím v rámci interního úložiště atd. Z těchto důvodů je v ukázkové definici hlaviček 6.4 použito makro `PG_FUNCTION_INFO_V1`, které se používá pro deklaraci funkce přístupné z prostředí SQL konzole.

Dále je netradiční použití návratové hodnoty `Datum`. Jedná se o univerzální datový typ, který se dá použít pro všechny interní datové reprezentace. O jaký datový typ se přesně jedná se definuje v SQL skriptu, který zavádí dané funkce implementované v jazyce `C` a přeložené do podoby sdílené knihovny.

Argumenty funkce `PG_FUNCTION_ARGS` se chovají stejně jako seznam univerzálních datových typů. Tedy obdobně jako hlavička funkce `main(int argc, char** argv)`. Dají se

předávat hodnotou nebo ukazatelem, kde makra pro ukazatele mají příponu `_P`. Stejně jako u návratové hodnoty funkce se vlastní datový typ deklaruje pomocí SQL skriptu, který implementovanou funkci zavádí do databáze.

Kontrola korektnosti datových typů

Implementované funkce (viz ukázka implementace vstupně/výstupních funkcí pro DTD v příloze D.1) používají funkcionalitu danou LibXML. Provádí kontrolu správné struktury zadaného validačního schématu a pokud jsou dodržena syntaktická pravidla, tak jej perzistují do trvalého úložiště. V opačném případě dojde k chybovému výstupu s kódem SQL chyby `ERRCODE_NOT_AN_DTD_DOCUMENT`. Obdobný přístup k chybovým hlášením je aplikován pro všechny implementované datové typy.

Zavedení funkce do databáze

PostgreSQL pracuje se *databázovými schématy*, které zavádí zvýšenou míru granularity nad vlastní databází, kde každá databáze může mít `1..n` přiřazených schémat. Vlastní databázové objekty se nachází v rámci těchto schémat.

Uživatelsky definované funkce mohou být instalovány buď do globálního schématu `postgres.default`, čímž se stanou dostupnými všem databázím, případně mohou být instalovány do zvoleného schématu dané databáze, čímž se stanou lokálními. Instalaci DTD datového typu ilustruje SQL skript 6.5.

```
-- DTD data type
CREATE TYPE dtd;

CREATE FUNCTION dtd_in(cstring)
    RETURNS dtd
    AS 'MODULE_PATHNAME'
    LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION dtd_out(dtd)
    RETURNS cstring
    AS 'MODULE_PATHNAME'
    LANGUAGE C IMMUTABLE STRICT;

CREATE TYPE dtd (
   input = dtd_in,
   output = dtd_out);
```

Kód 6.5: Instalace DTD datového typu do databáze.

6.1.2 Validační funkce

Vlastní funkce provádějící validaci využívá nově definovaných datových typů, kdy se dle zvoleného validačního schématu použije odpovídající varianta funkce. Norma jazyka C nedefinuje přetížené funkce, proto je použito procedurální rozšíření PostgreSQL PL/pgSQL, které dle použitých parametrů při volání funkce vybere správnou implementaci.

Algoritmus validačních funkcí pracuje následujícím způsobem:

- Prvním krokem v průběhu validace je kontrola správného formátování vstupního XML dokumentu. Pokud se jedná o platnou instanci, tak algoritmus pokračuje dalším krokem, v opačném případě ukončí validaci s návratovou hodnotou `false` a SQL chybou, která odpovídá sémantice nalezeného nedostatku.
- Druhým krokem je kontrola validačního schématu, které musí být také syntakticky korektní. V případě špatného formátu dojde k ukončení validace s příslušným SQL chybovým hlášením.
- Posledním krokem je vlastní validace XML dokumentu. Pokud odpovídá struktura XML dokumentu zadanému validačnímu schématu, tak je návratová hodnota rovna `true` (případně `false` pokud tomu tak není).

Vlastní proces validace je prováděn pomocí funkcí z knihovny LibXML, která používá interního číselníku pro popis chyb, proto bylo třeba vytvořit funkci pro transformaci chybových hlášení do reprezentace používané PostgreSQL. V rámci studia zdrojových kódu PostgreSQL byla nalezena podobná funkce (`xml_errorHandler`), která byla pro potřeby validačních funkcí upravena.

```
static void
xml_error_handler(void *ctxt, const char *msg,...)
{
        /* Append the formatted text to xml_err_buf */
        for (;;)
        {
                va_list  args;
                bool  success;

                /* Try to format the data. */
                va_start(args, msg);
                success = appendStringInfoVA(xml_error_buf, msg, args);
                va_end(args);

                if (success)
                        break;

                /* Double the buffer size and try again. */
                enlargeStringInfo(xml_error_buf, xml_error_buf->maxlen);
        }
}
```

Kód 6.6: Převedení chybových hlášení z LibXML do PostgreSQL.

6.1.3 Práce s dynamicky alokovanou pamětí

PostgreSQL používá vlastních funkcí pro práci s dynamicky alokovanou pamětí. Nejsou používány klasické přístupy pomocí `malloc` a `free`, ale speciální funkce `palloc` a `pfree`, které jsou určeny pro alokaci a dealokaci paměti během jedné *session*, trvající od započetí do ukončení databázové transakce.

PostgreSQL automaticky spravuje paměť, která je alokována pomocí `palloc` funkce a po ukončení transakce dojde k jejímu automatickému uvolnění. Explicitně lze vyvolat uvolnění paměti pomocí zmíněné funkce `pfree`, kdy je žádoucí provádět dealokaci po větších blocích. Ve většině případových studií nastává tento stav na konci transakce.

6.1.4 Použití datových typů a validačních funkcí

Pro vytvoření a naplnění tabulky s novými datovými typy se využívá následující SQL kód 6.7

```
CREATE OR REPLACE TABLE schema_data_types(
        id INT,
        dtd DTD,
        xsd XSD,
        rng RNG);
INSERT INTO schema_data_types VALUES (1, -- vlozeni validacnich schemat
        --DTD
        '<!ELEMENT name ( surname, firstname ) >
         <!ATTLIST name title NMTOKEN #REQUIRED >
         <!ELEMENT surname ( #PCDATA ) >
         <!ELEMENT firstname ( #PCDATA ) >',

        --XSD
        '<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
          <xs:element name="my">
           <xs:complexType mixed="true"/>
          </xs:element>
         </xs:schema>',

        --RNG
        '<element name="my" xmlns="http://relaxng.org/ns/structure/1.0"/>
            '
        );
```

Kód 6.7: SQL kód vytvářející tabulku s definovanými typy (DTD,XSD a RNG), následně je naplněna ukázkovými daty.

```
SELECT xmlvalidate_dtd('<?xml version="1.0" ?>', 'some valid DTD');
SELECT xmlvalidate_xsd('<?xml version="1.0" ?>', 'some valid XSD');
SELECT xmlvalidate_rng('<?xml version="1.0" ?>', 'some valid RNG');
```

Kód 6.8: Ukázka volání validačních funkcí.

6.1.5 Prototyp indexu FastX nad XML datovým typem

Navržený XML index je kombinací indexu XLABEL (popsán v kapitole 4.4.1) a IndexFabric (definice v kapitole 4.4.2). Výhodou použití tohoto indexu by mělo být snížení počtu přístupů k disku, kdy by se v rámci sestavení DOM pracovalo pouze s nezbytnou částí XML dokumentu. Rozhraní FastX je navrženo pro snadnou implementaci pomocí GIN a GiST generických indexů, kde PATRICIA Trie je vhodná pro GiST a obsahové tabulky 6.1 se dají modelovat pomocí GIN.

element_id	element_name
1	person
2	name
3	firstname
4	surname
5	address

element_id	attribute_name
1	id
1	career
2	title
5	correspondent

Tabulka 6.1: Obsah tabulky *xelems* a *xattr* při použití `FastX` indexu.

Tabulky *xelems* a *xattr* jsou vhodnými kandidáty pro implementaci upraveného GIN indexu, který místo odkazů na čísla řádků na nichž se dané klíče vyskytují, definuje adresaci pomocí dvojice *(rowid, offset)*, kde offset určuje polohu jednotlivých elementů/atributů v rámci daného XML dokumentu.

6.1.6 Testování validačních funkcí na vzorku dat

Testování bylo prováděno na PC s následující konfigurací:

- CPU: Intel Core2 Duo T9800 (2.93 GHz)
- HDD: 160 GB, 7200 ot./min.
- x86_64 GNU/Linux, kernel 2.6.32-31-generic
- PostgreSQL ve verzi 9.03, překlad s O2 optimalizací
- gcc 4.4.3

První fáze testování spočívala v ověření funkčnosti nově implementovaných datových typů, pro které byl sestaven korpus reprezentativních souborů. Následně byl vytvořen SQL skript, střídavě plnící tabulky různě velkými validačními schématy. V průběhu testu byl měřen čas potřebný pro vložení jednotlivých schémat i celé dávky souborů viz tabulka 6.2.

Před spuštěním každé sady testů byl server PostgreSQL restartován a po restartování byl spuštěn příkaz `vacume`, který čistí databázi od neplatných záznamů. Dále nebyla použita žádná integritní omezení (primární, cizí klíč), která by zvyšovala režii během zavádění dat. Těmito operacemi byl snížen vliv cache a dalších optimalizačních technik na minimum.

Ve druhé fázi testování byl sestaven nový korpus obsahující XML dokumenty a k nim odpovídající validační schémata. Zvolená metrika měření je obdobná jako u datových typů, tj. doba potřebná pro vložení nového záznamu a celková doba nutná k provedení vlastní validace viz tabulka 6.3.

Výsledky provedených testů spolu s celkovými časy prováděných operací jsou přiloženy na CD.

DTD	zpracování (ms)	XSD	zpracování (ms)	RNG	zpracování (ms)
1	7,905	1	8,160	1	8,726
2	8,528	2	8,183	2	8,170
3	7,909	3	8,260	3	8,158
4	8,275	4	8,413	4	8,330
5	8,262	5	8,249	5	8,422
6	8,294	6	8,191	6	8,298
7	8,307	7	8,278	7	8,154
8	8,252	8	8,255	8	8,293
9	8,259	9	8,238	9	8,308
10	8,311	10	8,194	10	8,190
11	8,175	11	8,283	11	8,375
12	8,175	12	8,318	12	8,219
13	8,471	13	8,277	13	8,342
14	8,346	14	8,288	14	8,278

Tabulka 6.2: Tabulka s časy zpracování DTD, XSD a RNG během vložení nového řádku.

XML	DTD (ms)	XML	XSD (ms)	XML	RNG (ms)
1	32,868	1	23,110	1	1,894
2	20,134	2	25,801	2	1,451
3	1,314	3	32,234	3	31,898
4	40,030	4	26,045	4	32,266
5	1,410	5	1,868	5	30,109

Tabulka 6.3: Tabulka s časy validace pomocí DTD, XSD a RNG.

6.2 Zhodnocení výsledku

Testy ukázaly, že doba zpracování středně velkých validačních schémat je v řádu jednotek ms, tudíž zvolená implementace datových typů XSD, RNG a DTD je úspěšná.

Testy nad validačními funkcemi dosahují řádu desítek ms, což se také dá označit za úspěšnou implementaci. Během validačních testů docházelo k výkyvům naměřených hodnot, které jsou způsobeny chybným schématem, případně XML dokumentem. V těchto případech je ukončen průběh validace dříve, proto jsou v tabulce 6.3 takto rozdílné hodnoty.

Nově vytvořené datové typy DTD, XSD, RNG fungují bez zjevných problémů až na omezení dané `INSERT` operací, kdy nelze vložit příliš dlouhý řádek. Řešením je použít místo operace `INSERT` příkaz `COPY`.

Index FastX nebyl otestován, protože je ve stádiu prototypu. V současné chvíli je předmětem dalšího výzkumu. Jako potenciálně zajímavé řešení se jeví použití modulu *ltree* případně *tsvector*, které jsou svým určením velmi podobné problematice indexovaní XML.

6.3 Možná rozšíření

V případě schválení implementovaného rozšíření se jeví jako logické pokračování registrace validačních schémat do centrálního databázového katalogu a následná validace vůči všem zaregistrovaným schématům, které jsou obsaženy v hlavičce XML dokumentu. Jako další nástavba by byla vhodná funkce implementující predikát `VALID`, který je definován v ISO normě a diskutován v kapitole 3.2. Výhodou tohoto přístupu by byla implicitní podpora jmenných prostorů.

Případné další rozšíření by mohlo přidat podporu pro práci s validačními schématy do grafické nástavby SQL interpreta `PgAdmin`, který nedokáže editovat uživatelsky definované typy a tudíž lze tabulky se sloupci typu XSD, DTD případně RNG pouze zobrazit.

Kapitola 7

Závěr

V rámci této diplomová práce bylo třeba podrobně prostudovat zdrojové kódy open-source databázového systému PostgreSQL, které samy o sobě svým rozsahem reflektují o jak komplexní dílo se jedná. V rámci vlastní implementace bylo zvoleno téma validace XML dokumentů, které chybí v současné verzi PostgreSQL.

Během studia podporovaných operací nad XML u předních poskytovatelů komerčních databázových systémů jsem musel rozšířit své znalosti ohledně problematiky XML. Následně se podrobně seznámit s různými druhy validačních schémat a způsobem použití dalších přidružených technologií jakými jsou: XPath, XQuery a XMLNS, které jsou využívány během procesu validace. Popis těchto technologií spolu s přístupy k XML datovému typu u stávajících databázových systémů je hlavní částí teorie rozebírané v první polovině této diplomové práce.

Jelikož nejvyšších rychlostí ve zpracování XML dokumentů dosahují nativní XML databáze, byla provedena rešerše za účelem identifikace rizik spojených s implementací v objektově relačních databázích. Dalším důvodem bylo možné přenesení indexačních technik do relačního modelu.

Vlastní implementace validačních funkcí využívá poskytovaných rozhraní v knihovně LibXML, která je také šířena pod otevřenou licencí, ale hlavně je používána i pro jiné operace v rámci stávajícího kódu PostgreSQL. Nedochází tedy k zanesení nových závislostí a tím potenciálních chyb. V průběhu řešení byly implementační detaily důsledně diskutovány s vývojářskou komunitou (pgsql-hackers) a případné poznámky byly zaneseny do finální podoby kódu. V současné chvíli je mnou implementované rozšíření odesláno k připomínkování.

Jelikož implementace validačních funkcí a nových datových typů nezabrala celé časové období vyhrazené pro řešení, byl po konzultaci s vedoucím diplomové práce započat návrh nového indexu, který by podporoval XML datový typ. Jedná se o oblast zájmu velkého množství výzkumných skupin nejen na akademické půdě, což s sebou přináší velké množství přístupů a s tím související informační neucelenost.

Z těchto důvodů byl v rámci diplomové práce proveden výzkum nejpoužívanějších indexačních technik ve stávajících implementacích a navrhnut prototyp indexu FastX, jež je předmětem budoucího vývoje v rámci programu Google Summer of Code 2011. Vlastní návrh indexu spolu s napojením na stávající kód je diskutována s přiděleným mentorem (Gregory Stark, MIT).

Literatura

[1] Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, Washington, DC, USA: IEEE Computer Society, 2002.

[2] Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). 2003, ISO/IEC 9075-1.

[3] ISO International Standard (IS) Database Language SQL. 2003, ISO/IEC 9075:2003.

[4] ISO International Standard (IS) Database Language SQL. 2006, ISO/IEC 9075:2006.

[5] ISO International Standard (IS) Database Language SQL. 2008, ISO/IEC 9075:2008.

[6] XML Path Language (XPath) 2.0 (Second Edition). `http://www.w3.org/TR/xpath20/`, 2010, dostupné dne 7.1.2010.

[7] Abiteboul, S.: Querying semi-structured data. *Database Theory—ICDT'97*, 1997: s. 1–18.

[8] Ahn, C.; Li, Q.; Elmasri, R.; aj.: A Survey of Three Types of XML Indexing Techniques. *ACM Transactions on Computational Logic*, 2005: str. 12.

[9] Alapati, S. R.; Kim, C.: *Oracle Database 11g: New Features for DBAs and Developers*. Apress, 2007, 563p s., iSBN 978-1-59059-910-5.

[10] Altheim, M.; Austin, D.: Extending XHTML with Compound Documents. `http://www.w3.org/TR/1999/xhtml-modularization-19990406/extending.html`, 1999, dostupné dne 7.1.2010.

[11] Apparao, V.; Byrne, S.; Champion, M.; aj.: Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/DOM/DOMTR`, 2003.

[12] Bartunov, O.; Sigaev, T.: GiST for PostgreSQL. 2003.

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

[14] Biron, P. V.; Beech, D.: XML Schema Part 1: Datatypes. `http://www.w3.org/TR/xmlschema-2/`, 2004, Dostupné dne 24.4.2011.

[15] Boag, S.; Chamberlin, D.; Fernández, M. F.: XQuery 1.0: An XML Query Language (Second Edition). `http://www.w3.org/TR/xquery/`, 2010, dostupné dne 7.1.2010.

[16] Bray, T.; Hollander, D.; Layman, A.; aj.: Namespaces in XML 1.0 (Third Edition). `http://www.w3.org/TR/xml-names/`, 2009.

[17] Bray, T.; Paoli, J.; Sperberg-McQueen, C.; aj.: Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/xml11/`, 2006.

[18] Bray, T.; Paoli, J.; Sperberg-McQueen, C.; aj.: Extensible markup language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/REC-xml/`, 2008.

[19] Chen, W.-J.; Chun, J.; Ngan, N.: *DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET*. International Technical Support Organization, 2006, 330p s., iSBN 0738496758.

[20] Clark, J.; DeRose, S.: XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath/`, 2010, dostupné dne 7.1.2010.

[21] Cooper, B.; Shadmon, M.: The index fabric: Technical overview. *RightOrder Inc*, 2001.

[22] Dietz, P.: Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, ACM, 1982, s. 122–127.

[23] Douglas, K.; Douglas, S.: *PostgreSQL: The comprehensive guide to building, programming, and administering PostgreSQL databases, Second Edition*. Sams Publishing, 2005, 1032p s., iSBN 0-672-32756-2.

[24] Fallside, D. C.; Walmsley, P.: XML Schema Part 0: Primer. `http://www.w3.org/TR/xmlschema-0/`, 2004, Dostupné dne 24.4.2011.

[25] Fredkin, E.: Trie memory. *Communications of the ACM*, 1960: s. 490–499, ISSN 0001-0782.

[26] Gnome project: XML C parser and toolkit. `http://xmlsoft.org/index.html`, 2011.

[27] Gonnet, G.; Baeza-Yates, R.; Snider, T.: Lexicographical indices for text: Inverted files vs. PAT trees. 1991.

[28] Harold, E. R.; Means, W. S.: *XML in a Nutshell*. O'Reilly Media, 2004, 712p s., iSBN 0-596-00764-7.

[29] James Clark, M. M.: RELAX NG Specification. `http://www.relaxng.org/spec-20011203.html`, 2001.

[30] Kaushik, R.; Shenoy, P.; Bohannon, P.; aj.: Exploiting local similarity for indexing paths in graph-structured data. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, IEEE, 2002, ISBN 0769515312, s. 129–140.

[31] Kay, M.: XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/xslt20/`, 2007, Dostupné dne 24.4.2011.

[32] Lee, Y. K.; Yoo, S.-J.; Yoon, K.; aj.: Index structures for structured documents. In *Proceedings of the first ACM international conference on Digital libraries*, DL 96, New York, NY, USA: ACM, 1996, ISBN 0-89791-830-4, s. 91–99, doi:http://doi.acm.org/10.1145/226931.226950.
URL http://doi.acm.org/10.1145/226931.226950

[33] Li, Q.; Moon, B.: Indexing and querying XML data for regular path expressions. In *Proceedings of the International Conference on Very Large Data Bases*, Citeseer, 2001, s. 361–370.

[34] Morrison, D.: PATRICIA—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 1968: s. 514–534, ISSN 0004-5411.

[35] Pal, S.; Cseri, I.; Seeliger, O.; aj.: Indexing XML data stored in a relational database. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, VLDB Endowment, 2004, ISBN 0-12-088469-0, s. 1146–1157.

[36] Recommendation, W.: XQuery 1.0 and XPath 2.0 Data Model (XDM). `http://www.w3.org/TR/xpath-datamodel/`, 2010.

[37] Samokhvalov, N.: XML Support in PostgreSQL. `http://www.pgcon.org/2007/schedule/events/14.en.html`, 2007, Dostupné dne 24.4.2011.

[38] Silberschatz, A.; Korth, H. F.; Sudarshan, S.: *Database System Concepts*. New York: McGraw Hill, 2006, iSBN 978-0-07-295886-7.

[39] Smith, J.; Stutely, R.: *SGML: the user's guide to ISO 8879*. Halsted Press New York, NY, USA, 1988, ISBN 0470211261.

[40] Suciu, D.: An overview of semistructured data. 1998.

[41] Thompson, H. S.; Beech, D.: XML Schema Part 1: Structures. `http://www.w3.org/TR/xmlschema-1/`, 2004, Dostupné dne 24.4.2011.

[42] Webber, J.; Parastatidis, S.; Robinson, I.: *REST in Practice*. O'Reilly Media, 2010, iSBN 978-0-596-80582-1.

[43] Weigel, F.; des Fortgeschrittenenpraktikums, P.: A Survey of Indexing Techniques for Semistructured Documents. *Project Thesis, University of Munich, Computer Science Institute, Teaching and Research Unit "Programming and Modelling Languages", August*, 2002.

[44] Weigel, F.; Meuss, H.; Schulz, K.; aj.: Content and structure in indexing and ranking XML. In *Proceedings of the 7th international Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004*, ACM, 2004, s. 79–96.

[45] World Wide Web Consortium: Mezinárodní standardizační organizace. `http://www.w3.org/Consortium/`, 2011.

[46] Zhang, C.; Naughton, J.; DeWitt, D.; aj.: On supporting containment queries in relational database management systems. *SIGMOD Rec.*, May 2001: s. 425–436, ISSN 0163-5808, doi:http://doi.acm.org/10.1145/376284.375722.
URL http://doi.acm.org/10.1145/376284.375722

Příloha A

Obsah CD

Na přiloženém CD se nachází následující obsah:

- Zdrojové kódy databáze PostgreSQL 9.03.
- Patch přidávající podporu nových datových typů DTD, XSD a RNG.
- Patch s implementací funkcí pro validace XML dokumentů.
- Patch implementující prototyp FastX indexu.
- SQL skript pro zavedení těchto implementací do databáze PostgreSQL 9.03.
- Korpus vzorových XML dokumentů a jejich validační schémata.
- Sada SQL skriptů použitá pro testování funkčnosti DTD, XSD a RNG datových typů.
- Sada SQL skriptů s testy validačních funkcí.
- XML dokumenty použité v technické zprávě.
- Validační schémata použitá v technické zprávě.
- Technická zpráva v programu LaTeX spolu s diagramy vysázenými pomocí balíčku Ti*k*Z.

Příloha B

Manuál

Na přiloženém CD se nacházejí zdrojové kódy open source ORDBMS PostgreSQL 9.03, pro úspěšné přeložení (předpokládá se platforma Linux) je třeba nainstalovat několik závislých balíčků jmenovitě:

- GNU make, doporučená verze alespoň 3.79.1 a výše. Jiné make distribuce nejsou podporovány.
- ISO/ANSI C překladač, minimálně (C89 kompatibilní) doporučuje se GCC 4.4, ale není striktně vyžadován.
- GNU Readline knihovna, případně libedit pod BSD licencí, která se používá v psql (PostgreSQL interpret SQL).
- GNU Flex 2.5.31 a GNU Bison 1.875 v minimálních verzích, pro překlad lexikální a syntaktické části SQL interpretu.
- Perl 5.8 a novější. Během překladu se využívá pro generování SQL instalačních skriptů.
- LibXML 2.6.23 a novější. Knihovna nutná pro implementovanou část DP.
- LibXSLT 2.6.23 a novější. Knihovna bývá často součástí LibXML2 balíčku. Stejně jako LibXML je vyžadována pro správný chod implementované části DP.
- OpenJade 1.4 a novější spolu s SGML knihovnami jsou nutné pro překlad dokumentace, která je psaná v jazyce SGML a následně pomocí transformací převáděna na HTML, PDF, případně MAN nápovědu a programovou dokumentaci. Použití validačních funkcí je popsáno v rámci této programové dokumentace.

Pro vlastní běh databáze je nutné alespoň 100 MB volného prostoru pro přeložení zdrojových kódů. Databázový cluster zabírá minimálně 35 MB a pro spuštění regresních testů je potřeba dalších 150 MB.

Instalace

V kořenovém adresáři $postgresql$ je doporučeno vytvořit složku určenou pro vlastní překlad a to následujícím způsobem:

```
mkdir build_dir
cd build_dir
../configure --with-libxml --with-libxslt
gmake world
gmake install
```

Konfigurační skript otestuje systém na přítomnost požadovaných knihoven a následně je spuštěn vlastní překlad. Během sestaví se vytváří i programová dokumentace, která je psána v jazyce SGML. Po překladu se spustí `make install` a dojde k instalaci do zvoleného umístění. Implicitně se jedná o složku `/usr/local/pgsql`.

Nastavení systému pro běh databáze

Pro přihlášení do databáze je vhodné vytvořit nový uživatelský účet (`useradd postgres; passwd postgres`), kde tento účet bude pozdějším vlastníkem databázového clusteru.

Dalším konfiguračním bodem je inicializace databázové clusteru. Ta se spustí pomocí příkazu `/usr/local/pgsql/initdb -D /usr/local/pgsql/data` (vyžadují se práva uživatele root). Po úspěšné inicializaci je třeba změnit vlastníka nově vytvořeného clusteru a to následujícím způsobem `chown -R postgres:postgres /usr/local/pgsql/data`. Vlastní spuštění databáze (nutná práva uživatele root) se provede pomocí příkazu `/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data`.

Po úspěšném spuštění běží PostgreSQL na lokáním systému a naslouchá na portu 5432. Spojení s databází je ale kvůli bezpečnostní politice povoleno pouze z adresy `localhost` a jenom uživateli `postgres`, proto je třeba spustit klientský program `psql` pod tímto uživatelem.

Zavedení validačních funkcí

V adresáři `$postgresql$/contrib/xml2` jsou umístěny zdrojové kódy validačních funkcí spolu s prototypem indexu. Pro překlad a instalaci do cílového umístění použijte příkaz `make; make install`. Následně je třeba spustit interpret SQL `psql` pod uživatelem `postgres`. Po spuštění dojde k připojení do implicitní databáze se stejným jménem jako uživatel tj. `postgres`. Příkazem `\i /usr/local/pgsql/share/contrib/pgxml.sql` (spuštěným v interpretu) dojde k instalaci nových datových typů spolu s validačními funkcemi do databáze. Pro případné odinstalování je připraven skript `\i /usr/local/pgsql/share/contrib/uninstall_pgxml.sql`.

Příloha C

Validační schémata a XML u ORDBMS

Ukázka XSD schématu

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="person" type="personType" />
  <xsd:complexType name="personType">
    <xsd:sequence>
      <xsd:element name="name" type="nameType" />
      <xsd:element name="address" type="addressType" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int" />
    <xsd:attribute name="career" type="xsd:string" />
  </xsd:complexType>
  <xsd:complexType name="addressType">
    <xsd:sequence>
      <xsd:element name="housenumber" type="xsd:int" />
      <xsd:element name="street" type="xsd:string" />
      <xsd:element name="town" type="xsd:string" />
      <xsd:element name="postcode" type="xsd:string" />
      <xsd:element name="country" type="countryType" />
    </xsd:sequence>
    <xsd:attribute name="correspondent" type="xsd:boolean" />
  </xsd:complexType>
  <xsd:complexType name="countryType">
    <xsd:attribute name="main" type="xsd:boolean" />
  </xsd:complexType>
  <xsd:complexType name="nameType">
    <xsd:sequence>
      <xsd:element name="surname" type="xsd:string" />
      <xsd:element name="firstname" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="title" type="xsd:string" />
```

```
  </xsd:complexType>
</xsd:schema>
```

Ukázka RelaxNG schématu

```
<?xml version="1.0" encoding="utf-8"?>
<rng:grammar xmlns:rng="http://relaxng.org/ns/structure/1.0" ns=""
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
        <rng:start combine="choice">
                <rng:ref name="person"/>
        </rng:start>
        <rng:define name="person">
                <rng:element name="person">
                        <rng:ref name="personType"/>
                </rng:element>
        </rng:define>
        <rng:define name="personType">
                <rng:element name="name">
                        <rng:ref name="nameType"/>
                </rng:element>
                <rng:element name="address">
                        <rng:ref name="addressType"/>
                </rng:element>
                <rng:optional>
                        <rng:attribute name="id">
                                <rng:data type="int"/>
                        </rng:attribute>
                </rng:optional>
                <rng:optional>
                        <rng:attribute name="career">
                                <rng:data type="string"/>
                        </rng:attribute>
                </rng:optional>
        </rng:define>
        <rng:define name="addressType">
                <rng:element name="housenumber">
                        <rng:data type="int"/>
                </rng:element>
                <rng:element name="street">
                        <rng:data type="string"/>
                </rng:element>
                <rng:element name="town">
                        <rng:data type="string"/>
                </rng:element>
                <rng:element name="postcode">
                        <rng:data type="string"/>
                </rng:element>
```

```
                <rng:element name="country">
                        <rng:ref name="countryType"/>
                </rng:element>
                <rng:optional>
                        <rng:attribute name="correspondent">
                                <rng:data type="boolean"/>
                        </rng:attribute>
                </rng:optional>
        </rng:define>
        <rng:define name="countryType">
                <rng:optional>
                        <rng:attribute name="main">
                                <rng:data type="boolean"/>
                        </rng:attribute>
                </rng:optional>
        </rng:define>
        <rng:define name="nameType">
                <rng:element name="surname">
                        <rng:data type="string"/>
                </rng:element>
                <rng:element name="firstname">
                        <rng:data type="string"/>
                </rng:element>
                <rng:optional>
                        <rng:attribute name="title">
                                <rng:data type="string"/>
                        </rng:attribute>
                </rng:optional>
        </rng:define>
</rng:grammar>
```

Vlastnosti		**IBM DB2 9.7**		**SQL Server**	**Oracle 11g**
		PureXML	XMLExtender		
XML technologie	DTD	✓	✓		✓
	XML Schemas	✓		✓	✓
	Xpath	✓	✓	✓	✓
	Xquery	✓		✓	✓
	XSL	✓	✓	✓	✓
Způsob uložení	BLOB		✓	✓	✓
	CLOB		✓		✓
	VARCHAR		✓		
	Native	✓			✓
	Schreded			✓	✓
XML datový typ	XMLType				✓
	XML	✓		✓	
	XMLVarchar		✓		
	XMLClob		✓		
	XMLFile		✓		
Sloupce XML typu		✓	✓	✓	✓
Tabulky XML typu					✓
XML Validace	DTD		✓		✓
	XSD	✓		✓	✓
	RelaxNG				
Složení XML dok.		✓	✓	✓	✓
Složení XML sch.				✓	✓
XML Mapping		✓	✓	✓	✓
XML Indexing		✓	✓	✓	✓
SQL/XML		✓	✓		✓
XQuery		✓	✓	✓	✓
XML Repository		✓			✓

Tabulka C.1: Podpora XML u IBM DB2 9.7, Microsoft SQL Server 2008 a Oracle 11g, podle jednotlivých programových dokumentací.

Příloha D

Ukázka implementace vstupně/výstupní funkce pro nový datový typ DTD v PostgreSQL

```
// typedef of new data types
typedef struct varlena dtdtype;
typedef struct varlena xsdtype;
typedef struct varlena rngtype;

/**
 * Help macros for TOASTING/DETOASTING
 */
#define DatumGetDTDP(X)  ((dtdtype *) PG_DETOAST_DATUM(X))
#define DTDPGetDatum(X)  PointerGetDatum(X)
#define PG_GETARG_DTD_P(n) DatumGetDTDP(PG_GETARG_DATUM(n))
#define PG_RETURN_DTD_P(x) PG_RETURN_POINTER(x)

/**
 * Declaration of IN/OUT functions
 */
Datum  dtd_in(PG_FUNCTION_ARGS);
Datum  dtd_out(PG_FUNCTION_ARGS);

PG_FUNCTION_INFO_V1(dtd_in);
PG_FUNCTION_INFO_V1(dtd_out);

/**
 * input function for DTD type, need cstring at first parameter,
 * check if DTD is well formed
 * @return toasted POINTER(x)
 */
Datum
dtd_in(PG_FUNCTION_ARGS)
```

```
{
#ifdef USE_LIBXML
        char *s = PG_GETARG_CSTRING(0);
        dtdtype *vardata;
        xmlDtdPtr dtd = NULL;

        vardata = (dtdtype *) cstring_to_text(s);

        /*
         * Parse the data to check if it is well-formed XML data.
         * Assume that ERROR occurred if parsing failed.
         */
    dtd = xmlIOParseDTD(NULL,
                xmlParserInputBufferCreateMem(s, strlen(s),
                        XML_CHAR_ENCODING_UTF8),
                        XML_CHAR_ENCODING_UTF8);

    if (dtd == NULL)
    { // schema itself is not valid
                ereport(ERROR,
                        (errcode(ERRCODE_NOT_AN_DTD_DOCUMENT),
                         errmsg("invalid DTD content")));
                return -1;
    }

        xmlFreeDtd(dtd);

        PG_RETURN_DTD_P(vardata);
#else
        NO_XML_SUPPORT();
        return 0;
#endif
}

/**
 * output function for DTD type
 * @return cstring
 */
Datum
dtd_out(PG_FUNCTION_ARGS)
{
        PG_RETURN_CSTRING(
                text_to_cstring((text *) PG_GETARG_POINTER(0))
        );
}
```

Kód D.1: Implementace nového datového typu DTD.