T A R T U Ü L I K O O L
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool
Rakendusinformaatika eriala

Tiit Kaeeli

XML dokumentide andmebaasisüsteemid

Diplomitöö

Juhendaja: Jaak Vilo, PhD

Autor: ..... “.....” mai 2004
Juhendaja: ..... “.....” mai 2004
Õppetooli juhataja: ..... “.....” 2004

TARTU 2004

Sisukord

Peatükk 1

Sissejuhatus

Arvutiteaduses leiab nii andmeedastuse, dokumendiesituse kui ka andmesalvestusega seonduvates valdkondades järjest enam kasutust XML formaat. [xmla]. Mitmete rakenduste puhul on sellest kujunenud de facto standard. XML formaadiga on seotud terve hulk erinevaid tehnoloogiaid ja standardeid mis igaüks sobivad mingi kindla ülesannete hulgaga tegelemiseks. Näiteks dokumentide struktuuri kirjeldamiseks DTD [xmla] ja Schema [sch], esitusviisi kirjeldamiseks XSL [xsla], teisendamiseks XSLT [xslb], programseks töötlemiseks SAX [sax] ja DOM [dom] jne.

XML on oma olemuselt hierarhiline ja srtuktrueeritud andmeformaat, seetõttu on loomulik et sageli on tarvis sooritada päringuid mingi väiksema andmeüksuse kätte saamiseks või muutmiseks suuremas dokumendis. Seda võimaldavad mitmed praeguseks olemas olevad või välja töötatavad standardid nagu XPath [xpa], XQuery [xqu], XUpdate [xup].

Et töö XML formaadiga oleks efektiivne, peaksid ka selle salvestamiseks kasutatavad vahendid nimetatud standardeid toetama. XML formaadi eripäradest tulenevalt ei ole see ühegi varasema andmesalvestustehnoloogia puhul võimalik. Seetõttu on asutud välja töötama täisesti uusi lahendusi.

Praeguseks on XML formaadis andmete salvestamisel välja kujunenud mitu erinevat suunda. Olulisemateks neist, mis ka käesolevas töös käsitlemist leiavad on algupärased XML andmebaasid (native XML database) , XML formaati toetavad relatsioonilised andmebaasisüsteemid ning relatsioonilised andmeteisendajad [Bou03]. Lisaks neile võimaldavad XML formaadis dokumente salvestada ka mitmed dokumendihaldus- (Document Management System) ning püsiva dokumendi objektimudeli (Persistent DOM) süsteemid. Esimesed neist on mõeldud eelkõige tööks dokumentide kui sellistega, mitte andmetega üldisemalt. Teised sobivad kasutamiseks peamiselt ühe süsteemi piires. Seetõttu me neid käesolevas töös lähemalt ei kajasta.

Mõnes mõttes erinevad tehnoloogiad täiendavad teineteist, sobides erinevate

probleemide lahendamiseks, samas ei ole piirid nende vahel kuigi täpselt välja kujunenud ning mitmes osas konkureerivad erinevad tehnoloogiad omavahel.

Käesolevas töös käsitlemegi erinevaid võimalusi XML formaadis andmete säilitamiseks.

Kuna kogu XML dokumentide salvestamisega seonduv valdkond on suhteliselt noor (XML formaat ise on praeguseks ainult 8 aastat vana), iseloomustab seda suur dünaamilisus ja kiire areng, sellest tulenevalt aga ka vähene standardiseeritus. Väga paljud tooted omavad spetsiifilisi ja teiste toodetega ühildumatuid võimalusi. Sellest tulenevalt on antud töö mahtu arvestades võimatu anda ülevaadet kõikidest olemasolevatest tehnoloogiatest. Valiku tegemisel on eelkõige arvestatud seda milliseid laiemalt aktsepteeritud standardeid või tõenäoliselt standardiks kujunevaid tehnoloogiaid mingid tooted toetavad, nende funktsionaalsust aga samuti ka tuntust ja kasutatavust. Mõnes mõttes on valik ka subjektiivne. Eesmärgiks on anda võimalikult hea ülevaade valdkonna hetkeseisust ning tähtsamatest arengusuundadest.

Teksti mõistmiseks peaks lugeja omama üldisi eelteadmisi XML formaadist ja objekt- relatsioonilistest andmebaasisüsteemidest ning SQL keelest.

Lähem huvi antud valdkonna vastu on tingitud praktilisest vajadusest leida sobiv meetod konkreetsete XML formaadis andmete salvestamiseks.

Meditsiinilise uurimistööga tegelevatel inimesel on tarvis läbi töötada paljusid teadusajakirjades ilmunud artikleid, neid süstematiseerida ning koguda ja säilitada oluliste artiklite terviktekste. Meditsiinialaste teadusartiklite kohta peab laialdast ja üsna täielikku andmebaasi USA Rahvuslik Meditsiiniraamatukogu (U.S. National Library of Medicine), lühendatult NLM. [nlm] Andmebaas on tuntud nimetuse PubMed all. [pub]

PubMedi andmebaas sisaldab artiklite kohta niisuguseid andmeid nagu abstraktid, autorid, olulised märksõnad, andmed ajakirja kohta kus artikkel on ilmunud jne. Kuna paljudes ajakirjades ilmunud artiklite suhtes kehtivad autorikaitsega seotud piirangud, ei võimalda PubMed otsest ligipääsu artiklite terviktekstidele.

Vajalikud terviktekstid tuleb hankida muudest allikatest, sageli vastava tasu eest. Kui artiklite tekste koguneb palju, võib tekkida probleem nende haldamisega. Kui kasutaja säilitab artiklite tekste failisüsteemis, on tema jaoks ainsaks viiteks artikli sisu kohta faili nimi ja asukoht kataloogipuus. PubMedi andmebaas võimaldab küll vajaliku artikli mitmesuguste parameetrite põhjal üles leida, kuid puudub võimalus sealsete kirjete vastavusse viimiseks terviktekstidega kasutaja arvutisüsteemis.

NLM pakub võimalust PubMedi kirjete pärimiseks XML formaadis [med]. See võimaldab kasutjal koguda ja säilitada koos artiklite tekstidega ka neile vastavaid PubMedi kirjeid. Selleks et niisugust võimalust maksimaalselt ära kasutada, tuleks luua infosüsteem, mis seoks artiklite terviktekstid PubMedi kirjetega ning võimaldaks sooritada kirjete suhtes mitmesuguseid päringuid.

Et päringuid oleks võimalik sooritada efektiivselt, tuleks kirjeid säilitada mingis andmebaasis. Seega on infosüsteemi loomisel peamiseks küsimuseks see, millist andmebaasisüsteemi oleks kõige otstarbekam kasutada PubMedist pärinevate XML formaadis andmete säilitamiseks.

Peatükk 2

Ülevaade XML formaadis dokumentide säilitamise võimalustest

XML (eXtensible Markup Language) on struktueeritud dokumendikirjelduskeel, mis põhineb SGML (Standard Generalized Markup Language) keelel [sgm]. Kuna ka HTML [htm] põhineb SGML keelel, siis on XML ja HTML üsna sarnased, kuid siiski mitmes osas erinevad. Kui HTML on keel, mille abil kirjeldada seda, kuidas mingeid andmeid või dokumenti kasutajale esitada, siis XML on ette nähtud ainult dokumendi või andmete sisemise struktuuri kirjeldamiseks. See kuidas andmeid või dokumenti kasutajale esitada, jääb teiste tehnoloogiate, näiteks XSL (eXtensible Stylesheet Language) [xsla] lahendada. Erinevalt HTML keelest on XML tõstutundlik (case sensitive) . Tänapäeval on HTML keele edasiarendusena loodud spetsiaalsel DTD- l (Document Type Definition) põhinev XML lahendus XHTML [xht] eesmärgiga viia HTML vastavusse XML standardiga võimaldamaks HTML dokumentide töötlemiseks kasutada XML põhiseid tööriistu.

XML formaadi peamiseks eeliseks on see, et ta on piisavalt üldine, võimaldades kirjeldada laia hulka andmeid ja dokumente, eraldades samas selgelt üksteisest sisu ja esitusega seonduvad küsimused. See võimaldab ühe ja sama sisu põhjal erinevaid esitusviiside kirjeldusi (nt. XSL stiililehed) kasutades saada erinevaid väljundeid. Samuti on olemas võimalused ühe XML dokumendi teisendamiseks teistsuguse struktuuriga XML dokumendiks või ka hoopis mingis muus, näiteks PDF formaadis dokumendiks. Kõik see võimaldab luua dokumente, mis on mõistetavad paljudele osapooltele, ainseks eelduseks on see et kõik asjasse puutuvad osapooled omavad vahendeid XML formaadiga töötamiseks.

Teiseks eeliseks on see, et XML keele süntaks on lihtne ja ühtaegu kergesti nii programselt töödeldav, kui ka inimestele mõistetav. Tänu sellele on inimesel vajaduse korral võimalik dokumente ka käsitsi lugeda, muuta ja koostada.

XML rakendused võib jagada laias laastus kaheks: dokumendipõhised ja andmepõhised [Bou03]. Dokumendipõhised rakendused esitavad mingit dokumenti, mis on lõpptulemusena ette nähtud inimestele lugemiseks, näiteks veebileht, raamat või artikkel. Andmepõhise rakenduse korral esitatakse XML formaadis mingeid andmeid, näiteks aadressid, telefoninumbrid, arved, statistilised andmekogumid. Sellised dokumendid võivad olla ette nähtud ainult tarkvarakomponentide vaheliseks andmevahetuseks. Kuigi ka paljusid andmepõhiseid XML dokumente on võimalik esitada inimesele mõistetaval kujul, ei muuda see veel nende olemust. Erinevus saab selgeks, kui võrrelda erinevat tüüpi dokumentide omadusi.

Dokumendipõhiste XML dokumentide puhul on oluline, millises järjekorras paiknevad mingi elemendi alamelemendid, eriti sama tüüpi alamad. Näiteks kui peatükk koosneb mitmest lõigust, siis on äärmiselt oluline, et need lõigud paikneksid õiges järjestuses, samas kui arve puhul ei ole oluline, millise toote kohta käivad andmed paiknevad XML dokumendis eespool. Juhul kui järjekord on oluline, võidakse selle määramiseks kasutada sobivat elementi või atribuuti, näiteks arvel rea numbrit.

Dokumendipõhiste XML dokumentide struktuur ei ole eriti täpselt piiritletud. Näiteks võib raamatus olla igas peatükis suvalises järjestuses tekstilõike, alampeatükke, tabeleid, jooniseid ja muid struktuurielemente. Andmepõhiste rakenduste, nagu arvete puhul on aga dokumendi struktuur väga rangelt määratud. Näiteks peab igal arvel olema üks esitaja, üks vasuvõtja, üks kuupäev, üks maksetähtaeg ning üks või mitu kauba nimetust, millest igaühe kohta on märgitud nimetus, kogus, ühiku hind ja summa.

Andmepõhised dokumendid on kõrgema granulaarsusega. Vähimaks loogiliseks andmeüksuseks on üks ainult teksti sisaldav element või atribuut. Väga vähe või üldse mitte esineb segatüüpi elemente.

Andmepõhiste ja dokumendipõhiste XML dokumentide eristamine ei ole praktikas sageli nii lihtne. Näiteks võib muidu andmepõhine dokument sisaldada dokumendipõhist osa, nagu kauba või teenuse kirjeldus hinnakirjas. Ometi on niisugune jaotamine oluline selleks, et leida iga konkreetse dokumentide hulga jaoks optimaalne viis nende säilitamiseks.

XML dokumente on võimalik kasutada ka andmete pikaajalisemaks säilitamiseks failisüsteemis. XML formaadi eeliseks mõne muu failiformaadi kasutamise ees on suur hulk olemasolevaid ja läbiproovitud vahendeid XML dokumentide töötlemiseks, mida on võimalik valmis kujul kasutada uue formaadi ja vajalike parserite väljatöötamise asemel. Samuti formaadi avatus, mis võimaldab kõikidel võimalikel osapooltel selle sisust õieti ja täpselt aru saada. Sellise kasutusviisi näiteks on muuhulgas XML formaadis konfiguratsioonifailid.

Mõnes mõttes võib selliselt kasutatavaid XML dokumente koos lisatehnoloogiatega käsitleda kui andmebaasi, sest neil on olemas mõningad andmebaasidele omased tunnused. Näiteks on olemas on võimalused andmete struktuuri kirjelda-

miseks (DTD, Schema) ning päringukeeled (XQuery, Xpath).

Samas kasutatakse suhteliselt palju andmemahtu struktuuri (Tagide) kirjeldamiseks, ligipääs andmetele ei ole mingil kombel optimeeritud ning puuduvad muud andmebaasidele omased võimalused nagu ligipääsu kontroll, transaktsioonid. Nende puuduste kõrvaldamiseks tuleks failisüsteemi asemel kasuta mingit muud salvestusviisi, mis võimaldaksid kasutada andmebaasisüsteemidest tuttavaid lisavõimalusi. Olemasolevad andmebaasisüsteemid selleks vahetul kujul ei sobi. Need on ette nähtud relatsiooniliste tabelikujulise struktuuriga andmete salvestamiseks, XML dokumentidele omast hierarhilist struktuuri aga ei ole üldisel juhul võimalik relatsioonilistesse raamidesse suruda.

Viimastel aastatel on välja töötatud mitmesuguseid andmebaasisüsteeme, mis on mõeldud spetsiaalselt XML formaadis andmete salvestamiseks. Olulisemateks neis on XML laiendused olemasolevatele relatsioonilistele või objekt- relatsioonilistele andmebaasisüsteemidele, algupärased XML andmebaasid ning relatsioonilised andmeteisendajad.

XML laiendused on olemasolevatele andmebaasisüsteemidele lisatud vahendid, mis võimaldavad neid kasutada ka XML formaadis andmete salvestamiseks. Need võib omakorda jaotada kaheks:

1. Süsteemid, mis kasutavad dokumentide salvestamiseks kasutaja poolt defineeritud andmebaasiskeemi

2. Süsteemid, mis loovad dokumentide salvestamiseks vajaliku andmebaasiskeemi automaatselt.

Suurem osa relatsiooniliste andmebaaside XML laiendusi võimaldavad dokumente salvestada kahel erineval moel. Esimesel juhul hoitakse tervet dokumenti tervikuna ühes sobiva pikkusega tekstilist andmetüüpi veerus.

Teisel juhul jaotatakse dokumendid osadeks, seades igale XML dokumendi struktuurile vastavusse mingi andmebaasiskeemi ning salvestades XML dokumendid osade kaupa erinevatesse andmetabelitesse. Saadud tabelite struktuur võib erineda kardinaalselt salvestatavete dokumentide struktuurist, kuna andmebaasiskeemi koostamisel lähtutakse salvestatavate andmete, aga mitte andmete edastamiseks kasutatava XML dokumendi struktuurist. Vastava andmebaasiskeemi koostamiseks on vajalik, et salvestatavate dokumentide struktuur oleks korrapärane ning schema või DTD abil formaalselt määratletud. Oluline on mitte ainult schema või DTD olemasolu, vaid ka see, et võimalike dokumentide struktuur oleks selle abil täielikult paika pandud. Näiteks ei tohi olla `ANY` tüüpi elemente. See on vajalik, et salvestusskeemi koostamisel oleks võimalik ette näha kõikide võimaike dokumentide struktuuri, mida antud skeemi kasutades võib olla tarvis salvestada. Dokumentide vastavus schemale või DTD- le on nende salvestamiseks

möödapääsmatu. Seega on dokumendid alati rangelt valideeritud. Saadud tabelitega opereerimiseks on võimalik kasutada standardseid relatsiooniliste andebaaside poolt pakutavaid vahendeid.

Kõiki XML dokumente on võimalik salvestada terviklikul kujul. Osadeks jaotatud kujul salvestamiseks vajalikele eeltingimustele vastavad üldjuhul ainult andmepõhised dokumendid.

Kui dokumendid salvestatakse tervikuna, kasutataks päringute optimeerimiseks mitmesuguseid spetsiaalseid indekseid. Sõltuvalt süsteemist on võimalik indekseerida nii PCDATA- elemente, atribuute kui ka dokumentide struktuuri (tage) ning teid dokumendi puustruktuuris. Päringute formuleerimiseks on võimalik kasutada nii XML põhiseid päringukeeli, nagu XQuery ja XPath kui ka andmebaasisüsteemi SQL keelt.

Kui dokument jaotatakse salvestamisel osadeks ei ole harilikult hiljem võimalik andmebaasist enam täpselt sama dokumenti uuesti tagasi saada. Kaduma lähevad kommentaarid, töötlusjuhised (processing instructions) , samuti info selle kohta, kas mingi osa dokumendist oli otsene dokumendi osa või kasutati mõnda sisemist või välimist üksust (Entity). Mõningatel juhtudel läheb kaduma ka informatsioon samal tasemel paiknevate elementide järjestuse kohta. Paljude andmepõhiste rakenduste puhul ei ole selline infokadu probleemiks, dokumendipõhiste XML dokumentide salvestamisel aga ei ole see harilikult vastuvõetav.

Algupärased XML andmebaasid on andmebaasisüsteemid mis on spetsiaalselt loodud XML dokumente säilitamiseks. Nende peamiseks eripäraks on see, et erinevalt XML laiendustest, kus tulenevalt kasutatava andmebaasisüsteemist iseärasustest on säilitusühikuks andmetabeli rida kasutatakse algupärastes XML andmebaasides säilitusühikuna XML dokument. Tabeli rolli täidab kogum (collection) , mis on oma olemuselt hulk mingil kombel seotud XML dokumente. Kogumite loomisel võib aluseks olla vastavus samale schemale või DTD- le, kuid paljud süsteemid võimaldavad koondada ühte kogumisse ka mitmele erinevale schemale või DTD- le vastavaid dokumente. Algupärased XML andmebaasid säilitavad dokumente kui selliseid, see tähendab elemente, atribuute, PCDATA elemente, dokumentide struktuuriandmeid, mitte ainult andmeid, mida need dokumendid sisaldavad. Kasutavad salvestusmeetodid on sellised, mis võimaldavad salvestada suvalisi XML dokumente. Seetõttu sobivad algupärased XML andmebaasid hästi ka selliste dokumentide salvestamiseks, mille jaoks DTD või Schema puudub. Schema või DTD olemasolul on võimalik määrata, kas dokumentide valideerimine on vajalik või mitte. Mõningate andmebaaside puhul, nagu näiteks Software AG Tamino [tam] , on võimalik ka osaline valideerimine. Sellisel juhul on schema abil piiritletud ainult dokumendi mingi osa struktuur. Valideerimisel kontrollitakse, et schemale vastaksid dokumendi need osad, mis on schemas defineeritud, kuid dokumendid võivad sisaldada ka komponente, mille jaoks definitsioon puudub.

Algupärased XML andmebaasid jagunevad andmete salvestamise viisi põhjal

sellisteks, mis salvestavad dokumente kas tervikuna või mingite osade kaupa tekstiliselt ning sellisteks, mis salvestavad dokumente mudelina, näiteks DOM- puudena. Andmeid võidakse salvestada mingis spetsiaalselt selleks otstarbeks välja töötatud failiformaadis, aga võidakse kasutada ka relatsioonilisi andmebaasisüsteeme. Relatsioonilise andmebaasisüsteemi olemasolu on sel juhul kasutaja eest täielikult varjatud. Algupäraste XML andmebaaside poole pöördumiseks saab kasutada ainult XML põhiseid vahendeid ja päringukeeli. Andmebaasi kasutamisel salvestatakse sinna kas terveid dokumente või nende osi tekstilist andmetüüpi veergudesse või kasutatakse mingit hulka tabeleid struktuurielementide salvestamiseks.

Mõnes mõttes võib ka relatsiooniliste andmebaaside XML laiendusi, mis pakuvad võimalusi tervikuna salvestatud dokumentide XML põhiseks kasutamiseks pidada algupärasteks XML andmebaasideks.

Päringute optimeerimiseks kasutatakse algupärastes XML andmebaasides mitmesuguseid sisemisi või ka kasutaja poolt defineeritud indekseid.

Algupärased XML andmebaasid võimaldavad saada päringute abil uuesti kätte sama dokumendi, mis sinna sisestati. Kui süsteem salvestab dokumente tekstina, siis on võimalik tagasi saada sisestatuga identne dokument. Kui salvestamiseks kasutatakse mudelipõhist lähenemist, on tagastatav dokument sisestatuga samaväärne selles osas, millist osa dokumendi struktuuris võimaldab antud mudel esitada.

Millist salvestusviisi mingite XML dokumentide puhul kasutada sõltub igal konkreetsel juhul paljudest asjaoludest ja igaks juhtumiks sobivaid reegleid ei ole praeguseni välja töötatud. Siiski võib tuua mõningad üldised juhised .

Kõigepealt on dekomponeeritud salvestusviisi relatsioonilistes andmebaasisüsteemides võimalik kasutad ainult siis, kui kõikide salvestatavate dokumentide võimalik struktuur on eelnevalt täpselt teada. Muudel juhtudel tuleb kasutada terviklikku salvestusviisi või algupärast XML andmebaasis. Isegi kui dokumentide struktuur on rangelt määratletud, võib algupäraste XML andmebaaside kasutamine või dokumentide salvestamine tervikuna relatsiooniliste andmebaaside XML lainedustes olla sobivam, kui dokumentide struktuuri ei ole eriti korrapärane. Sellisel juhul tekiks relatsioonilistes andmebaasides kas väga palju NULL väärtustega tabeliridu või siis väga palju erinevaid tabeleid. Algupärast XML andmebaasi võiks eelistada ka siis, kui päringute hulgas domineerivad sellised, mis tagastavad terveid dokumente või suuri osi üksikutest dokumentides. Dokumentide jaotamine relatsioonilistesse tabelitesse on sobivam siis, kui dokumendid on selge struktuuriga ning on oodata palju päringuid, mis kasutavad väikest hulka andmeid suurest hulgast erinevatest dokumentidest (Näiteks päring, mis leiab kõikide nende tellimuste esitajad, millel on märgitud mingi konkreetne toode).

Peatükk 3

XML päringukeeled

Koos XML formaadi laialdase levikuga tekkis vajadus päringukeelte järele, mille abil oleks võimalik leida vajalikku informatsiooni üksikutest XML dokumentidest või ka tervetest dokumendihulkadest. Kuna XML on erinev varasematest informatsiooni säilitamiseks kasutatavatest formaatidest, selgus et olemasolevate päringukeelte laiendamise või kohandamaise asemel on otstarbekam välja töötada täiesti uued tehnoloogiad. Niisuguseid päringukeeli on mitmeid, kuid kõige olulisemateks neist on XPath ja XQuery.

XML päringukeeli on võimalik rakendada kõikidele XML dokumentidele, sõltumata sellest kus ja millisel kujul neid säilitatakse.

Nagu kõikides andmebaasisüsteemides on ka algupärastes XML andmebaasides üheks olulisemaks osaks päringukeeled. Kuna algupärastes XML andmebaasides on salvestusühikuks XML dokument kasutatakse nimetatud keeli suurema osa algupäraste XML andmebaaside puhul peamiste päringukeeltena.

3.1 XPath

Kõikide XML päringukeelte üheks olulisemaks osaks on vahendid dokumendi struktuurist vajalike osade, nagu elemendid ja nende väärtused, alampuud või atribuudid leidmiseks. Valdava osa tänapäevaseid päringumootoreid kasutavad selleks otstarbeks XPath keelt. XPath on W3C konsortsiumi poolt välja töötatud ja standardiseeritud. Esimene standardi versioon, XPath 1.0 pärineb 16. novembrist 1999 a. Praeguseks hetkeks on välja töötamisel uus versioon, XPath 2.0. Versioon 1.0 on loodud pidades silmas eelkõige dokumentide esituse-, teisendamise- ja dokumentide osadele viitamisega seonduvaid küsimusi lähtuvalt XSL, XSLT ja XPointer keelte vajadustest. Kuna kõik nimetatud keeled vajasid sarnaseid vahendeid dokumentide sisus navigeerimiseks ja osade leidmiseks, otsustati et selle asemel, et iga keele jaoks luua eraldi spetsiifilised vahendid, on otstarbeks välja

töötada uus ühtne standard, mis rahuldaks kõikide olemasolevate keelte vajadused, ning mida oleks võimalik kasutada ka uutes loodavates standardites. Sellest tulenevalt võib XPathi mõnes mõttes lugeda XSL, XSLT ja XPointer standardite osaks.

Oma algse ülesande täitis XPath hästi, kuid XML päringukeele XQuery välja töötamise käigus ilmnesid siiski mitmed puudused. Nimelt oli vahepeal valminud XML Schema standard, mis võimaldas defineerida ja kasutada erinevaid andmetüüpe. XPaht 1.0 aga Schemat ega andmetüüpide kontrolli ei toetanud. Lisaks sellele ilmnes XQuery seisukohast veel terve rida muid puudusi. Kuna XPath 1.0 oli loodud eeskätt dokumentide esitust silmas pidades, oli prioriteediks seatud igal võimalikul juhul mingisugusegi tulemuse kätte saamine. Päringute korrektsusele osutati vähem tähelepanu. Kui tekkis olukord, kus päring ei olnud päris korrektne, kuid mingi vastus oli sellele siiski võimalik leida ja tuli valida, kas tagastada mitte päris korrektne vastus või anda veateade ja töö katkestada. XPath 1.0 puhul eelistati veateadet vältida. Selline lähenemine võib olla õigustatud inimesele suunatud dokumendiesituses, kuid mitte andmetele orienteeritud päringukeeltes. Niisuguste probleemide lahendamiseks alustati uue XPath standardi versiooni 2.0 välja töötamist. Selle valmimist on oodata samaaegselt XQuery enda standardiseerimisega. Uus standard on suures osas eelmise versiooniga ühilduv, lisades sellele uusi süntaksielemente ja funktsioone, kuid mitte päris täielikult. Mitteühildumeine on seotud mõningate operaatorite erineva käitumisega, peamiselt just nimetatud eelistuse osas veateate või mitte täielikult korrektse vastuse puhul. Vajaduse korral täieliku ühilduvuse tagamiseks on XPath 2.0 puhul võimalik määrata, et töö toimub tagasiühilduvas reziimis. Sellisel juhul käituvad muutunud operaatorid samamoodi nagu versioonis 1.0. Seda võimalust ei ole võimalik kasutada XQuery, küll aga XSLT ja muude standardite puhul. [KCD+03]

XPath keele eesmärgiks on võimaldada leida XML dokumentidest väiksemaid osi nagu elementide ja atribuutide väärtused ning alampuud. Selleks kasutatakse spetsiaalseid teeavaldisi (path expressions) , mis oma süntaksilt ei põhine XML keelel. Päringuid teostatakse mitte XML dokumentide tekstilise struktuuri, vaid spetsiaalsel puukujulisel XPath andmemudeli (XPath data model) suhtes. Igal sammul on nii sisendiks kui ka tulemuseks kogum elemente.

Avaldisi on kahte liiki: teeavaldised mis leiavad dokumendi osi ning predikaadid, mis kontrollivad leitud dokumendi osade vastavust mingile tingimusele.

Teeavaldised koosnevad mitmest sammust, igal sammul leitakse igast sisendtipust, mida nimetatakse ka kontekstipuks, lähtudes uus tipp, mis on järgmise sammu jaoks uueks kontekstipuks. Sammud eraldatakse kaldkriipsudega `/` . Iga sammu jaoks on määratud suund (axe) , vaikimisi suund on järglaste leidmine. Näiteks järgmise avaldise korral (`doc("books.xml")` on sisendfunktsioon, mis leiab antud failis salvestatud XML dokumendi ruuttipu.):

```
doc("books.xml")/bib/book
```

Leiab samm `/bib` kõik ruuttipu `bib` tüüpi alamad, `/book` omakorda kõikide leitud `bib` tüüpi tippude `/book` tüüpi alamad. Tulemuseks on kogum `book` tüüpi elemente. Paljude suundade märkimiseks on olemas nii lühendatud kui ka täielik süntaks. Eelmises näites on kasutatud lühendatud süntaksit, täieliku süntaksi puhul näeks sama avaldis välja selline:

```
doc("books.xml")/child::bib/child::book
```

Teisteks suundadeks on järglase suund `//` , (täieliku süntaksiga `/descendant::`), vanema leidmine `/..` (`/parent::`) ning hulk ainult pika süntaksi abil väljendatavaid suundi nagu eelmine kolleeg (`/preceding-sibling::`), järgmine kolleeg (`/next-sibling::`), esivanem (`/ancehestor::`), esivanem või kontekstitipp (`/anchestor-or-self::`,), järeltulija või kontekstitipp (`/descendant-or-self::`). Lisaks tippudele on olemas suunad ka atribuutide leidmiseks: `/@` või pika sünataksiga `/attribute::`

. Võimalik on esitada nõudmisi leitud tipu tüübile, näiteks nõuda et see oleks kommentaar. Selleks kasutatavad avaldised on `processing-instruction()` , `comment()` , `text()` ja `node()` .

Lisaks teeavaldistele on võimalik kasutada predikaatavaldisi, mille abil saab kitsendada leitavate tippude hulka. Neis on võimalik kasutada programmeerimiskeeltest tuttavaid operaatoreid ja funktsioone. Predikaatavaldis paikneb vastavate tippude leidmiseks kasutatava teeavaldise järel kandilistes sulgudes, näiteks:

```
doc("books.xml")/bib/book/author[last="Stevens"]
```

leiab vastavat teeavaldist rahuldavate tippude hulgast ainult sellised, millel on alam `last` sisuga `Stevens` . Üheks eriliseks predikaadiks on tipu positsioon tema kolleegide hulgas. Selleks saab kasutada `position()` funktsiooni, aga olemas on ka lühike süntaks, mille puhul kandilistesse sulgudesse pannakse ainult vastav number. Järgmine avaldis leiab näiteks kõigi raamatute esimesed autorid:

```
doc("books.xml")/bib//author[1]
```

Pika süntaksi korra oleks sama asi:

```
doc("books.xml")/bib/book/author[position() = 1]
```

Lisaks sellele, et XPathil on oluline osa XQuery standardis, on see mõningate algupäraste XML andmebaasisüsteemide, nagu Xindice puhul ka üldse ainsaks päringukeeleks. Samuti on puhtal kujul XPath päringute teostamise võimalus olemas mitmetes andmebaasisüsteemides muude päringukeelte kõrval.

3.2 XQuery

Dokumentide osade leidmise kõrval, mida võimaldab XPath, on mõnikord tarvis erinevaid dokumente omavahel kombineerida, dokumentide struktuuri muuta või muid keerulisemaid päringuid sooritada. Seda kõike võimaldab välja töötatav universaalne XML päringukeel XQuery.

XQuery ei ole esimene selline keel. Erinevaid XML päringukeeli on välja pakutud alatest XML- i enda tekkimisest. Olulisemateks ja tähelepanuväärsemateks neist võiksid olla QUILT, XQL ja X-Query. QUILT on sisuliselt XQuery otsene eelkäija, sealt on pärit mitmed olulised konstruktsioonid, nagu FLWOR- avaldised. Siiski ei standardiseeritud seda W3C poolt, vaid otsustati jätakata tööd keele täiustamiseks. XQL keelt on kasutatud mitmetes algupäraste XML andmebaasisüsteemides päringukeelena, kuid praeguseks on tema levik vähenenud. X-Query on Software AG poolt välja töötatud ja nende Tamito serveris kasutatud päringukeel, mis pakub analoogseid võimalusi XQuery keelele, kuid mis võeti praktilisest vajadusest tingituna ühepoolselt kasutusele, kuna XQuery oli sel ajal reaalse kasutuse jaoks alles liiga varajases loomisjärgus.

XQuery üheks oluliseks osaks on XPath 2.0 siiski ei saa XQuerit pidada ainuüksi XPath keele laienduskes, kuna olulisi põhimõttelisi lisavõimalusi on selleks liiga palju. XQuery loomist on tugevalt mõjutanud ka relatsioonilistes andmebaasides kasutatav SQL keel, selle tunnistuseks on FLWOR- avaldised, mis oma olemuselt on üsna lähedased relatsioonilistes andmebaasides kasutatavale `SELECT FROM WHERE` konstruktsioonile.

Praeguseks ei ole W3C konsortsiumi poolt välja töötatav XQuery 1.0 standard veel valminud. Sellest hoolimata on olamas juba suur hulk selle realisatsioone. Valdav osa arenevatest ja elujõulistest algupärastest XML andmabaasisüsteemidest kas omavad juba praegu XQuery toetust päringukeelena või on selle lisandumine lähemal ajal plaanis. Kõigi eelduste kohaselt saab XQuerist selle standardiseerimisel peamine XML päringukeel.

XQuery poolt pakutavad peamised võimalused on lisaks dokumendi osade leidmisele XPath avaldist abil uute dokumentide või elementide genereerimine olemasolevate põhjal ning dokumentide restruktureerimine ja kombineerimine FLWOR-avaldiste abil. Kõiki keele konstruktsioone on võimalik omavahel kombineerida. Olamas on suurt hulka sisemiselt defineeritud funktsioone ja operaatoreid, samuti on kasutajal võimalik luua uusi funktsioone ja funktsioonide teeke ning kasutada muutujaid. Nii nagu XPath 2.0, on ka XQuery tüübitundlik. Nagu näha sarnaneb XQuery mitmete omaduse poolest programmeerimiskeeltele.

XML elementide genereerimiseks on kaks võimalikku konstruktsiooni. Esimene on sobivam siis, kui suurem osa genereeritavast elemendist on konstantne. Kasutatakse XML keele süntaksit, dünaamiliste osade sisestamiseks lisatakse vastavad avaldised loogeliste sulgude vahele. Näiteks järgmise päringu:

```
<example>
    <p> Here is a query. </p>
    <eg> doc("books.xml")//book[1]/title </eg>
    <p> Here is the result of the above query.</p>
    <eg>{ doc("books.xml")//book[1]/title }</eg>
</example>
```

tulemus oleks niisugune:

```
<example>
    <p> Here is a query. </p>
    <eg> doc("books.xml")//book[1]/title </eg>
    <p> Here is the result of the above query.</p>
    <eg><title>TCP/IP Illustrated</title></eg>
</example>
```

Teine võimalik konstruktsioon algab võtmesõnaga, milleks on kas `element` , `attribute`, `document`, `text`, `processing-instruction`, `comment` või `namespace` , sellele järgneb nimega komponentide nagu element, atribuut töötlusinstruktsioon või nimeruum puhul komponendi nimi, mille järel paikneb loogelistes sulgudes selle sisu. Kasutada on võimalik kõiki XQuery keele vahendeid.

```
element book {
    attribute year { 1977 },
    element author {
        element first { "Crockett" },
        element last { "Johnson" }
    },
    element publisher {"HarperCollins Juvenile Books"},
    element price { 14.95 }
}
```

Ka komponendi nimi võib olla dünaamiliselt määratud loogelistes sulgudes pikneva avaldise abil:

```
element { translate-element-name("publisher", "German") } {
    "HarperCollins Juvenile Books"
}
```

XQuery keele üheks võimasaimaks konstruktsiooniks on FLWOR- avaldised. Nimetus tuleneb võimalike klauslite For, Let, Where, Order by ja Return esitähtedest nende tüüpilises kasutamise järjekorras.

`for` ja `let` klauslite abil seotakse muutujate väärtused avaldistega ning tekitatakse muutuja- väärtus paaride hulkade järjend. Lihtsaimal juhul koosneb `for` klausel ühest muutujast ja ühest avaldisest. Sellisel juhul väärtustatakse avaldis ja antakse muutujale järgemööda iga väärtus tekkinud väärtuste hulgast ning täidetakse avaldise ülejäänud klauslid. Seejärel saab muutuja järgmise väärtuse ja tsükkel kordub. Tekib tsükkel üle avaldise väärtuse komponentide hulga. `for` klausel võib sisaldada ka mitut muutujat. Sellisel juhul tekib tsükkel üle avaldiste väärtuste komponentide hulkade korrutise. Näiteks `for` klausel

`for $i in (1, 2), $j in (3, 4)`

Tekitab järgmised muutuja- väärtus paaride hulgad toodud järjekorras:

```
($i = 1, $j = 3)
($i = 1, $j = 4)
($i = 2, $j = 3)
($i = 2, $j = 4)
```

Iga muutujaga `for` klauslis võib siduda positsioonilise muutuja võtmesõna `at` abil, näiteks:

```
for $car at $i in ("Ford", "Chevy"), $pet at $j in ("Cat",
"Dog")
```

Positsiooniline muutuja on samuti osaks muutuja- väärtus paaride hulgast ja selle väärtus näitab vastavale `for` tsükli muutujale antud juhul omistatud väärtuse järjekorranumbrit alustades ühest. Eelmise näite puhul saame järgmised muutuja-väärtus paaride hulgad:

```
($i = 1, $car = "Ford", $j = 1, $pet = "Cat")
($i = 1, $car = "Ford", $j = 2, $pet = "Dog")
($i = 2, $car = "Chevy", $j = 1, $pet = "Cat")
($i = 2, $car = "Chevy", $j = 2, $pet = "Dog")
```

`let` klausel seob muutuja vastava avaldise väärtusega ilma tsüklit tekitamata. Muutuja esineb koos oma väärtusega igas `for` klausli poolt genereeritud muutuja-väärtus paaride kulgas. Nii `for` kui ka `let` klausleid võib olla mitu, kusjuures eelmistes seotud muutujate väärtusi võib järgmistes kasutada.

`where` klausel eemaldab muutuja- väärtus paaride hulgast sellised, mis ei vasta ette antud tingimusele, järele jäävad hulgad, mille puhul avaldise väärtuseks on `true` . Sarnane vastavale klauslile SQL keeles.

`order by` klausel võimaldab muuta muutuja- väärtus paaride hulkade järjekorda, mis muidu on `for` klausli poolt tekitatuna vastavuses elementide järjestusega dokumentides. Võimalik on kasutada mitmeid järjestamise tunnuseid, kasvavat või kahanevat järjekorda. `order by` klausel on samuti lähedane oma analoogile SQL keeles.

Kogu FLWOR konstruktsiooni tulemus tekitatakse `return` klausli abil. See väärtustatakse järjest iga muutuja- väärtus paaride hulga kohta pärast seda kui kõik teised klauslid on oma töö teinud. Iga hulga kohta saadakse üks väljundfragment, kogu konstruktsiooni tulemuseks on nende fragmentide konkatenatsioon. `return` klauslis on võimalik kasutada kõiki XQuery võimalusi, sageli on selleks elemendi konstruktor. Sel moel on FLWOR avaldiste abil võimalik teisendada XML dokumente, muuta elementide järjekorda, kombineerida omavahel andmeid mitmetest dokumentidest.

Järgmises näites muudetakse elementide järjekorda dokumendis, luues bibliograafia põhjal autorite nimekirja milles iga autori kohta on märgitud tema osalusel kirjutatud raamatute nimekiri. Sisendandmeteks on järgmine XML fragment:

```
<bib>
    <book>
        <title>TCP/IP Illustrated</title>
        <author>Stevens</author>
        <publisher>Addison-Wesley</publisher>
    </book>
    <book>
        <title>Advanced Unix Programming</title>
        <author>Stevens</author>
        <publisher>Addison-Wesley</publisher>
    </book>
    <book>
        <title>Data on the Web</title>
        <author>Abiteboul</author>
        <author>Buneman</author>
        <author>Suciu</author>
    </book>
</bib>
```

Kasutame sellist FLWOR konstruktsiooni, kusjuures `fn:distinct-values()` on funktsiooni, mis eemaldab oma parameetrite hulgast dublikaadid:

```
<authlist>
{
    for $a in fn:distinct-values($books)//author
    order by $a
    return
        <author>
            <name>
                { $a/text() }
            </name>
            <books>
            {
                for $b in $books//book[author = $a]
                order by $b/title
                return $b/title
            }
            </books>
        </author>
}
</authlist>
```

Tulemuseks saame:

```
<authlist>
    <author>
        <name>Abiteboul</name>
        <books>
            <title>Data on the Web</title>
        </books>
    </author>
    <author>
        <name>Buneman</name>
        <books>
            <title>Data on the Web</title>
        </books>
    </author>
```

```
    <author>
        <name>Stevens</name>
        <books>
            <title>TCP/IP Illustrated</title>
            <title>Advanced Unix Programming</title>
        </books>
    </author>
    <author>
        <name>Suciu</name>
        <books>
            <title>Data on the Web</title>
        </books>
    </author>
</authlist>
```

XQuery võimaldab kasutada ka if- then- elese tingimuslausesid. Sünataks on analoogiline paljudele teistele programmeerimiskeeltele. Näiteks järgmise sisend-dokumendi korral:

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="1992">
        <title>Advanced Programming in the UNIX Environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>65.95</price>
    </book>

    <book year="1999">
        <title>
            The Economics of Technology and Contentfor Digital TV
        </title>
        <editor>
            <last>Gerbarg</last><first>Darcy</first>
            <affiliation>CITI</affiliation>
        </editor>
        <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>
</bib>
```

Annab päring:

```
for $b in doc("books.xml")//book
return
    <book>
        { $b/title }
        {
            for $a at $i in $b/author
            where $i <= 2
            return <author>{string($a/last), ", ",string($a/first)}</author>
        }
        {
        if (count($b/author) > 2)
        then <author>et al.</author>
        else ()
        }
    </book>
```

Sellise tulemuse:

```
<book>
    <title>TCP/IP Illustrated</title>
    <author>Stevens, W.</author>
</book>
<book>
    <title>Advanced Programming in the Unix Environment</title>
    <author>Stevens, W.</author>
</book>
<book>
    <title>Data on the Web</title>
    <author>Abiteboul, Serge</author>
    <author>Buneman, Peter</author>
    <author>et al.</author>
</book>
<book>
    <title>The Economics of Technology and Content for
    Digital TV</title>
</book>
```

Tänu oma paindlikkusele ja suurele hulgale võimalustele on XQuery sobivaks päringukeeleks mida kasutatakse paljudes algupärastes XML andmebaasisüsteemides ja mille kasutamise edasist laienemist on oodata pärast esimese standardi valmimist.

Peatükk 4

Algupärased XML andmebaasid

4.1 Mis on algupärane XML andmebaas

Algupärased XML andmebaasid on välja töötatud spetsiaalselt XML formaadis andmete salvestamiseks. Termin älgupärane XML andmebaas"(native XML database) leidis esmakordselt kasutust Software AG poolt loodud andmebaasisüsteemi Tamino reklaamikampaanias. Kuna algselt on tegemist reklaamiterminiga, siis ei ole selle tähendus täpselt defineeritud. Kirjanduses paistab olevat laialdaselt aktsepteeritud XML:DB grupi poolt välja pakutud definitsioon [xmlc] , mis ütlebjärgmist:

Algupärane XML andmebaas:

1. Defineerib XML dokumendi, mitte dokumendis esitatud andmete jaoks (loogilise) mudeli ning salvestab ja tagastab dokumente selle mudeli alusel. Mudel peab sisaldama vähemalt elemente, atribuute, PCDATA- t ja elementide järjekorda. Näideteks sellistest mudelitest on XPath andmemudel, XML infoset ning mudelid, mis tulenevad DOM- ist ja SAX- sündmustest.

2. Kasutab minimaalse (loogilise) säilitusühikuna XML dokumenti, nagu relatsiooniline andmebaas kasutab minimaalse (loogilise) säilitusühikuna tabelirida.

3. Kasutatav füüsiline salvestusmudel ei ole oluline. Kasutada võidakse näiteks relatsioonilist, hierarhilist või objekt- orienteeritud andmebaasi või ka spetsiaalset salvestusformaati nagu indekseeritud ja pakitud failid.

Nagu definitsioonist võib järeldada, keskenduvad algupärased XML andmebaasid dokumentide kui selliste, mitte dokumentides sisalduvate andmete säilitamisele, andmebaasi sisestatakse ja sealt saadakse tagasi XML dokumente. Algupärane XML andmebaasisüsteem võib põhineda ka mõnel olemasoleval relatsioonilisel andmebaasisüsteemil. Tuntumad algupärased XML andmebaasid on

näiteks Software AG Tamino, Apache Software Foundation Xindice, eXist, X-Hive/DB. Täielikum nimekiri on võimalik leida aadressilt [Bou04]

4.2 Andmete füüsiline säilitamine

Andmete säilitamiseks native XML andmebaasides kasutatakse kahte erinevat lähenemist. Nendeks on mudelipõhine ja tekstipõhine andmesäilitus.

4.2.1 Mudelipõhine andmete säilitamine

Mudelipõhise andmete säilitusviisi puhul ei salvestata XML dokumente tervikliкul kujul, vaid need jaotatakse mingist mudelist, nagu näiteks W3C DOM lähtudes komponentideks ja säilitatakse tekkinud komponente mingis sobivas formaadis kas failisüsteemis või siis mõnes relatsioonilises andmebaasis.

Järgnevalt käsitleme lähemalt tehnoloogiat, mida kasutatakse dokumentide salvestamiseks vabavaralises algupärases XML andmebaasisüsteemis eXist [Mei02].

eXist kasutab andmete salvestamiseks spetsiaalselt selleks otstarbeks välja töötatud failiformaati ning indekseid, mis võimaldavad kiirendada vastuse leidmist päringutele.

XML päringukeeled nagu XPath ja XQuery kasutavad vajalike osade leidmiseks XML dokumentide puustruktuurist teeavaldisi (path expression) . Näiteks leiab avaldis

```
book//section/title
```

kõik `title` elemendid, mis on sellise `section` elemendi lapsed, mille esivanemaks on `book` element.

Päringu tulemuseks on komplekt elemente selles järjestuses, milles nad asuvad päritavas dokumendis.

Lisaks struktuuripõhistele päringutele on võimalik tulemuse täpsustamiseks kasutada predikaatpäringuid, näiteks:

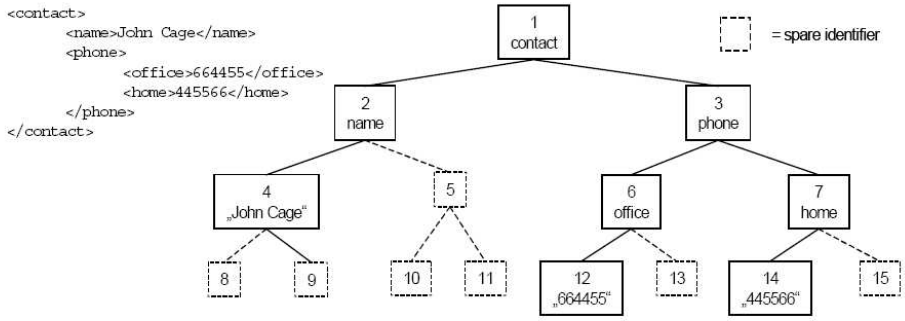
```
book//section[contains (title, 'XML')]
```

Erinevalt struktuuripõhistes päringutest mis kasutavad tulemuse leidmiseks ainult infot dokumendi struktuuri kohta, vajavad predikaatpäringud informatsiooni elementide ja atribuutide sisu kohta. Lähemalt käsitleme XPath keelt XML päringukeeli käsitlevas osas.

Kõige lihtsam meetod selliste päringute lahendamiseks eeldab dokumendi täielikku läbivaatust, et otsida üles kõik vajalikku tüüpi elemendi ning kontrollida nende väärtusi. Kuigi selline meetod on oma olemuselt lihtne, muutub vastuse leidmine suurema hulga dokumentide puhul aeganõudvaks. Dokumentide täieliku läbivaatuse vältimiseks on kasutusele võetud indeksid, mis võimaldavad leida

lahendusi nii dokumentide struktuuri, kui ka elementide ja atribuutide väärtuste kohta käivatele päringutele.

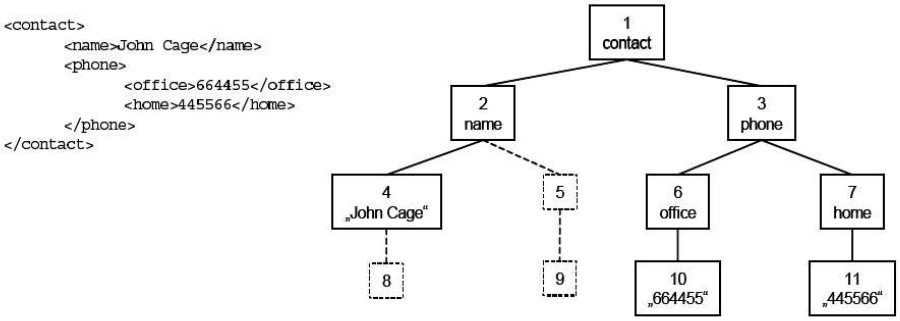
Väärtuste indekseerimiseks on võimalik kasutada olemasolevaid traditsioonilisi meetodeid, nagu näiteks B- puud, kuid struktuuriandmete indekseerimiseks need meetodid ei sobi. Tarvis on meetodit, mis võimaldaks kiiresti leida elementidevahelisi suhteid nagu näiteks kas üks element on teise laps või järeltulija või kumb element paikneb dokumendistruktuuris eespool. Selle probleemi lahendamiseks on välja pakutud mitmesuguseid meetodeid, kuidas oleks otstarbekas nummerdada dokumentide elemente. Nummerdamise puhul määratakse igale elemendile unikaalne identifikaator, mida on võimalik kasutada indeksites elemendi asemel viitena vastavale elemendile. eXist poolt kasutatava nummerdamismeetodi aluseks on järgmine skeem: Dokumente kujutatakse täielike puudena, mille iga tipu, mis ei ole leht, alamate arv on võrdne maksimaalse alamate arvuga mis leidub mingil tipul vastava XML dokumendi puustruktuuris. Selleks, et puu saaks täielik, lisatakse sinna vajalikul hulgal lisatippe. Saadud puu tipud nummerdatakse eesjärjestuses. Joonisel 4.1 on toodud lihtne XML dokument ja sellele vastav täielik puu koos identifikaatoritega.



Joonis 4.1:

Saadud identifikaatoritel on selline omadus, et iga identifikaatori põhjal on võimalik arvutada tema vanema ja järglaste (millest osad võivad olla fiktiivsed), identifikaatorid. Kirjeldatud meetodi puhul on oluliseks puuduseks see, et kuna puu peab olema täielik, tuleb sinna lisada palju lisatippe. See muudab identifikaatorite väärtused ülemäära suureks.

Selle puuduse kõrvaldamiseks kasutatakse kirjeldatud meetodi edasiarendust, mille puhul asendatakse puu täielikkuse nõue nõudega, et tippude alamate arv peab olema võrdne ainult puu ühe taseme piires. Nõutav alamate arv igal tasemel salvestatakse koos dokumendipuuga. Selline puu on kujutatud joonisel 4.2.



Joonis 4.2:

Sellise täiustuse tulemusena muutub lisatavate tippude koguhulk märgatavalt väiksemaks, kuid säilib võimalus identifikaatorite põhjal määrata elementidevahelisi suhteid dokumendi struktuuris. Lisaks sellele tekib võimalus lisada uusi elemente ja vastavaid tippe puu sügavamatele tasemetele, ilma et oleks vaja mingil kombel muuta puu ülemist osa.

eXist kasutab dokumentide ja indeksite salvestamiseks nelja faili: `collections.tbx` , `dom.tbx` , `elements.tbx` ja `words.tbx` . Kõikide indeksite puhul kasutatakse B- puid. Indekseerimine toimub mitte üksiku dokumendi, vaid terve dokumentide kogumi (collection) piires, kuna suur osa päringuist hõlmab tervet või isegi mitut dokumentide kogumit. Kokkuvõttes väheneb vajalike indeksi lugemiste arv.

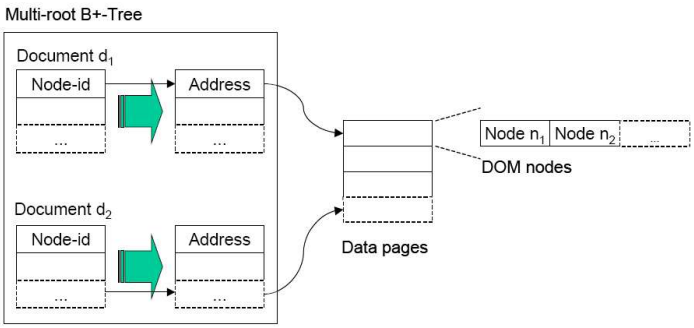
`Collections.tbx` faili kasutatakse selleks, et seostada dokumente kogumitega, kuhu mingi dokument kuulub. Seal on kirjas kogumite ja dokumentide identifikaatorid.

`Dom.tbx` faili on lehekülgedeks jaotatud fail, mida kasutatakse dokumentide reaalseks salvestamiseks. Sinna on salvestatud kõik dokumentides leiduvad elemendid vastavalt DOM mudelile. Lisaks elementidele hoitakse seal ka B- puud, mis seab tippude identifikaatoritele vastavusse faili lehekülgede aadressid, kuhu vastava tipu sisu on salvestatud.

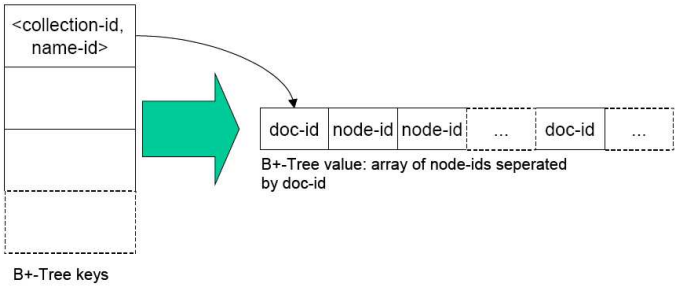
`dom.tbx` faili struktuur on kujutatud joonisel 4.3.

Faili elements.tbx kasutatakse selleks, et seada elementide ja atribuutide nimed vastavusse tippude identifikaatoritega. Iga indeksi element koosneb võtmest, milleks on kogumi identifikaatori ja elemendi või atribuudi nime paar ning väärtusest, milleks on massiiv, mis sisaldab järjestatud kujul vastavaid dokumendi- ja tipu identifikaatoreid.

Viimane indeksifail `words.tbx` sisaldab pööratud indeksit, mis seab doku-



Joonis 4.3:



Joonis 4.4:

mentide elementides või atribuutide väärtustes leiduvatele märksõnadele vastavusse dokumendi- ja tipu identifikaatorid. Märksõnad saadakse, kui väärtuse tekst jagatakse üksikuteks sõnadeks või sõnaosadeks. Sisuliselt on tegemist täisteksti indeksiga. Faili struktuur on sama, mis elements.tbx faili puhul, ainult võtmeks on paar kogumi identifikaatorist ja märksõnast.

Kirjeldatud indekseid kasutades on võimalik leida elemendi või attribuudi nime või nende tekstis esinevate märksõnade põhjal vastavad tipu identifikaatorid, ning tipu identifikaatorite põhjal aadressid, kuhu vastavad elemendid on füüsiliselt salvestatud. Seda kasutab ära päringumootor efektiivseks XPath päringute lahendamiseks. XPath päringud on eriti olulised, kuna nad on peamiste algupäraste XML andmebaaside päringukeelte nagu XQuery ja XPath 2.0 aluseks.

4.2.2 Tekstipõhine andmete säilitamine

Teiseks andmete salvestamise võimaluseks algupärastes XML andmebaasides on tekstipõhine salvestus. Sellist meetodit kasutavad näiteks andmebaasisüsteemid Tamino ja Xindice. Tekstipõhise salvestusmeetodi korral säilitatakse XML dokumente tervikliku tekstina, harilikult mingis spetsiaalses formaadis ja pakitud kujul, kuid neid ei jaotata elementideks, nagu mudelipõhise lähenemise puhul. Selleks, et optimeerida päringuid, kasutatakse mitmeid indekseid. Kui mudelipõhise lähenemise korral olid indeksid juba andmebaasisüsteemi sisse ehitatud, siis tekstipõhise lähenemise puhul jääb indeksite defineerimine harilikult kasutaja või andmebaasi administraatori ülesandeks. See võimaldab indeksite loomisel arvestada seda, milliseid päringuid ja millistes mahtudes on loodava andmebaasi puhul oodata ning sellest lähtuvalt luua igal konkreetsel juhul kõige sobivamad indeksid.

Andmebaasisüsteem Tamino kasutab kolme tüüpi indekseid. [CRZ03] Esimeseks indeksiks on väärtusepõhine indeks, mis võimaldab leida antud atribuudi või elemendi väärtuse põhjal dokumendid, kus vastavad elemendid või atribuudid esinevad. Seda indeksit on võimalik kasutada ka väärtuste põhiseid predikaate sisaldavate päringute lahendamisel, näiteks leida arved, mille summa on suurem kui ette antud väärtus. Indeksi võtmetena kasutatakse elementite või atribuutida väärtusi tervikuna. Kui indekseeritakse elemente, millel leidub alamaid, siis kasutatakse elemendi väätrusena elemendi enda ja kõikide tema alamelementide väärtusti, mis on järjestatud vastavalt dokumendi struktuurile.

Teiseks indeksiks on täisteksti indeks, mis võimaldab ette antud märksõna järgi leida dokumente, kus antud märksõna esineb. Märksõnad saadakse, kui kõikide dokumendis leiduvate elementide ja atribuutide väärtuste tekst jaotatakse sõnadeks. Täisteksti indeksi eriliigina on võimalik kasutada sõnaosade indeksit.

Kolmandaks indeksiks on XML struktuuri indeks, mida kasutatakse arvestuse pidamiseks dokumentides esinevate teede kohta. Selline indekseerimine võib olla osaline, täielik või üldse välja lülitatud. Osalise indekseerimise puhul säilitatakse ainult informatsiooni selle kohta, millised teed esinevad mingile dokumenditüübile vastavate dokumentide hulgas, kuid mitte seda, millistes dokumentides mingid teed täpselt esinevad. Täieliku struktuuriindeksi puhul säilitatakse ka informatsiooni selle kohta, millistes dokumentides mingid teed esinevad.

Sellistest struktuuriindeksitest on kasu dokumentide puhul, mille jaoks puudub DTD või schema või mille schema ei määratle dokumendi struktuuri täielikult. Näiteks võib esineda ANY- elemente või selliseid elemente, mille osad alamad on valikulised või mille esinemissagedus ei ole täpselt määratletud.

4.3 Andmete muutmine algupärastes XML andmebaasides

Igasugused andmekogumid, nii ka algupärased XML andmebaasid vajavad võimalusi selleks, et lisaks pärimisele saaks ka lisada, eemaldada ning muuta nendes salvestatud informatsiooni. Kuna algupärastes XML andmebaasides on peamiseks andmeüksuseks dokument, räägime järgnevalt XML dokumentide lisamisest, eemaldamisest ja uuendamisest.

Dokumentide lisamine ja eemaldamine on hädavajalikud, aga ka üsna triviaalsed operatsioonid. Seetõttu on nende jaoks kõikides andmebaasisüsteemides võimalused olemas. Keerulisem probleem on dokumentide muutmine.

Selleks, et dokumendi muutmisel oleks üldse mingit mõtet, peab süsteem sisemiselt säilitama dokumente osadena mudelipõhiselt ja omama võimalusi mudeli uuendamiseks. Kui dokumente säilitatakse tervikuna, kasutatakse üldjuhul dokumendi uuendamiseks täielikku asendamist vana kustutamise ja uue lisamise teel. Paljudel juhtudel varjatakse see kasutaja eest, kellele pakutakse ikkagi näilist võimalust dokumente muuta.

Dokumentide uuendamiseks ei ole praeguseks välja kujunenud ühtset standardit. Suurem osa algupäraseid XML andmebaasisüsteeme kasutavad tootespetsiifilisi dokumentide uuendamise keeli.

Üheks enam levinud ja laiemalt kasutatavaks dokumentide uuendamise keeleks on XML:DB initsiatiivi poolt välja pakutud XUpdate [xup]. Kuigi XUpdate ei ole praeguseks veel standardiseeritud, kasutavad seda vähemalt andmebaasid Tamino, eXist, Xindice ja X-Hive/DB. Järgnevalt lühike ülevaade XUpdate keele poolt pakutavatest võimalustest.

4.3.1 XUpdate

XUpdate on XML:DB initsiatiivi poolt välja töötatav XML dokumentide uuendamise keel. Iga selles keeles väljendatud muudatus on süntaksi poolest XML dokument. Muudatusi kirjeldavad elemendid ja atribuudid on pärit nimeruumist http://www.xmldb.org/xupdate. Iga muudatus on määratud elemendiga `xupdate:modifications`, millel on kohustuslik atribuut `version`, mis näitab millist XUpdate keele versiooni on muudatuse kirjeldamiseks kasutatud. Element `modification` võib sisaldada järgmisi võimalikele operatsioonidele vastavaid elemente:

```
xupdate:insert-before
xupdate:insert-after
xupdate:append
xupdate:update
xupdate:remove
```

`xupdate:rename`
`xupdate:variable`
`xupdate:value-of`
`xupdate:if`

Elemente `xupdate:insert-before` ja `xupdate:insert-after` kasutataks uute osade lisamiseks dokumenti. Neil elementidel on kohustuslik atribuut `select`, mille väärtuseks on XPath avaldis, mis määrab ära kontekstipud dokumendile vastavas andmemudelis, mille ette või järele uus tipp või tipud lisatakse. `xupdate:insert-before` lisab uue tipu kontekstipu ette eelmiseks kolleegiks, `xupdate:insert-after` aga kontekstipu järele järgmiseks kolleegiks. Uue lisatava elemendi defineerimiseks võivad `xupdate:insert-before` ja `xupdate:insert-after` elemendid sisaldada elemente:

`xupdate:element`
`xupdate:attribute`
`xupdate:text`
`xupdate:processing-instruction`
`xupdate:comment`

mille abil on võimalik luua vastavalt uusi elemente, atribuute, tekstitippe, töötlusinstruktsioone või kommentaare. Näiteks loob järgmine fragment:

```
<xupdate:element name="address">
    <xupdate:attribute name="id">
        2
    </xupdate:attribute>
    <town>
        San Francisco
    </town>
</xupdate:element>
```

Sellise elemendi:

```
<address id="2"/>
    <town>
        San Francisco
    </town>
</address>
```

`xupdate:append` element sarnaneb insert elementidega, kuid loob uue tipu kontekstipu alamana. Element võib omada atribuuti `child` , mis määrab mitmendaks alamaks loodav element saab. Vaikimisi lisatakse loodav element kontekstipu viimaseks alamaks.

`Xupdate:update` elementi kasutatakse selleks, et asendada kontekstipu sisu antud elemendi poolt defineeritud uue tipuga. Elemente `xupdate:remove`

ja `xupdate:rename` vastavalt selleks, et eemaldada või nimetada ümber kontekstipp. `Xupdate:rename` puhul peab kontekstipuks olema element või atribuut.

`Xupdate:variable` võimaldab defineerida muutujaid, muutuja nimi määratakse atribuudiga `name`.

Järgnevalt näide, mis lisab algdokumenti:

```
<?xml version="1.0"?>
<addresses version="1.0">

    <address id="1">
        <fullname>Andreas Laux</fullname>
        <born day='1' month='12' year='1978'/>
        <town>Leipzig</town>
        <country>Germany</country>
    </address>

</addresses>
```

XUpdate abil uue aadressi esimese aadressi järele:

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">

    <xupdate:insert-after select="/addresses/address[1]" >
        <xupdate:element name="address">
            <xupdate:attribute name="id">2</xupdate:attribute>
            <fullname>Lars Martin</fullname>
            <born day='2' month='12' year='1974'/>
            <town>Leizig</town>
            <country>Germany</country>
        </xupdate:element>
    </xupdate:insert-after>

</xupdate:modifications>
```

Tulemus oleks niisugune:

```
<?xml version="1.0"?>
<addresses version="1.0">
    <address id="1">
        <fullname>Andreas Laux</fullname>
        <born day='1' month='12' year='1978'/>
        <town>Leipzig</town>
        <country>Germany</country>
    </address>
    <address id="2">
        <fullname>Lars Martin</fullname>
        <born day='2' month='12' year='1974'/>
        <town>Leizig</town>
        <country>Germany</country>
    </address>
</addresses>
```

4.4 Algupäraste XML andmebaaside programmeerimisliidesed

Suurem osa kõikidest andmebaaside poole pöördumistest toimud muude programmide vahendusel, mitte nii, et lõpkasutaja suhtleb otse andmebaasi kliendi kasutajaliidesega. Teisi programme kasutatakse selleks, et andmebaasiga suhtlemine inimesel mugavamaks teha ja varjata tema ees mitmesugused tehnilised üksikasjad, nagu näiteks kasutatavate päringute süntaks. Paljudel juhtudel ei pruugi kasutaja üldse arugi saada, et tarkvara, mida ta kasutab on kuidagi ka andmebaasidega seotud.

Sellest tulenavalt on igasuguste andmebaasisüsteemide puhul vältimatuks osaks süsteemist liidesed mille abil on teistel tarkvarakomponentidel võimalik nendega suhelda.

Programmeerimisliideste puhul on väga oluline, et need oleksid erinevate andmebaaside puhul ühtlustatud, võimaldamaks kirjutadad tarkvara, mis oleks võimeline töötama koos erinevate andmebaasisüsteemidega. Relatsiooniliste andmebaasisüsteemide puhul on mitmete programmeerimiskeelte jaoks olemas sellised liideste standardid, nagu ODBC, JDBC, Perl DBI. Ka algupäraste XML andmebaaside tootjad on sellise standardi vajalikkust mõistnud, kuid seniajani laialdaselt aktsepteeritud standard puudub. Kõige lähemal on sellele XML:DB initsiatiivi poolt välja töötatav XML:DB liides. [xmlb] Arvestades seda, et XML:DB initsiatiivi kuulub suur osa algupäraste XML andmebaaside tootjatest ja et järjest suuremale osale andmebaasidele lisatakse XML:DB toetus, võib loota, et sellest kujuneb programmeerimisliidese standard algupäraste XML andmebaaside jaoks. Praegu toetavad seda vähemalt niisugused andmebaasisüsteemid nagu Tamino, eXist, Total XML, X-Hive/DB, Xindice.

4.4.1 XML:DB Tootest sõltumatu liides

XML:DB liides on oma ülesehituselt modulaarne. Liidese funktsioonid on jaotatud moodulitesse, moodulid omakorda erinevatele põhitasemetele (core level).

Selleks, et mingi realisatsioon vastaks antud tasemele, peab see realiseerima kõik sellel tasemel nõutavad moodulid koos kõigis neis moodulites defineeritud liidesefunktsioonidega. Realiseeritud võib olla ka muid mooduleid, mis ületavad toetatud taseme piire. Nendeks võivad olla mõningad olulised moodulid kõrgematelt põhitasemetelt või ka tootespetsiifilised moodulid, mida standardis ei ole üldse ette nähtud. Näiteks andmebaasisüsteem eXist pakub lisaks standardis defineeritutele teenust `XQueryService`. Niisugused võimalused on vajalikud, kuna algupärased XML andmebaasid arenevad ja muutuvad väga kiiresti ning praeguseks ei ole veel päris täpselt selge, milliseid liideseid on nendega töötamiseks tarvis.

See võimaldab igal XML:DB liidest toetaval andmebaasil määrata, millist liidese taset see toetab, ning samuti igal rakendusel määrata, millisele liidese tasemele vastavat andmebaasisüsteemi antud rakendus oma tööks vajab. XML:DB liidese kirjeldamiseks kasutatakse programmeerimiskeelest sõltumatut OMG IDL keelt. Programmeerimiskeeltest on praegu toetatud Java ja Python.

Esimeseks põhitasemeks on tase 0, selle taseme peab realiseerima iga XML:DB liidest toetav andmebaasisüsteem. Nõutavad moodulid on API Base ja XMLResource. Tasemel 1 lisandub neile XPathQuery service.

Järgnevalt lühike näide Java programmist, mis leiab filmid pealkirjaga `Music Man` . Kasutatav andmebaas peab toetama XML:DB liidese taset 1.

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;

/**
 * Simple XML:DB API example to query the database.
 */
public class Example1 {
    public static void main(String[] args) throws Exception {
        Collection col = null;

        try {
            String driver = "org.vendorx.xmldb.DatabaseImpl";
            Class c = Class.forName(driver);

            Database database = (Database) c.newInstance();
            DatabaseManager.registerDatabase(database);
            col = DatabaseManager.getCollection(
                "xmldb:vendorx://db.xmlmovies.com:2030/movies"
            );

            String xpath = "//movie[@title='Music Man']";
            XPathQueryService service =
                (XPathQueryService) col.getService("XPathQueryService", "1.0");
            ResourceSet resultSet = service.query(xpath);

            ResourceIterator results = resultSet.getIterator();
            while (results.hasMoreResources()) {
                Resource res = results.nextResource();
                System.out.println((String) res.getContent());
            }
        }
        catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " + e.errorCode);
        }
        finally {
            if (col != null) {
                col.close();
            }
        }
    }
}
```

Peatükk 5

XML formaati toetavad relatsioonilised andmebaasisüsteemid

Valdav osa tänapäevaseid infosüsteeme kasutab vajalike andmete säilitamiseks relatsioonilisi andmebaasisüsteeme. Tuntumate relatsiooniliste andmebaasisüsteemide arendajateks on tugevad ja kõrge teaduspotentsiaaliga kompaniid. Seoses XML põhiste tehnoloogiate järjest laiema levikuga on ka relatsioonilisete andmebaasisüsteemide tootjad mõistnud vajadust kohandada oma süsteeme tööks selle formaadiga. Praeguseks on sellistel andmebaasisüsteemidel nagu Oracle 10g, IBM DB2, Microsoft SQL server, Sybase ASE ja teised olemas mitmeid võimalusi tööks XML formaadis dokumentidega ning selles vallas toimub ka edasine intensiivne arendustegevus.

Relatsiooniliste andmebaaside XML laiendused tegelevad kahe peamise ülesandega, milleks on XML dokumentide genereerimine olemasolevate relatsiooniliste andmete põhjal ning juba eelnevalt XML formaadis olevate dokumentide salvestamine relatsioonilistesse andmebaasisüsteemidesse. Sealjuures pakutakse mitmeid mugavaid lisavõimalusi, nagu XML vaadete loomine relatsioonilistele andmetele ja relatsiooniliste vaadete loomine XML formaadis andmetele, dokumentide teisendamine XSLT stiililehti kasutades, võimalused muuta andmebaasi sisu kättesaadavaks http protokolli kasutades. XML formaadis dokumentidest on praegu võimalik päringuid sooritada XPATH keelt kasutades, kuid lähemal ajal on mitmetele relatsioonilistele andmebaasisüsteemidele oodata ka XQuery toetust.

XML formaadis andmete salvestamiseks kasutatakse kahte erinevat lähenemist. Nendeks on dokumentide säilitamine tervikuna kas mõnda olemasolevat tekstilist andmetüüpi veergudes või siis spetsiaalselt selleks otstarbeks mõeldud XML tüüpi veergudes ning dokumentide salvestamine osade kaupa erinevates relatsioonilistes andmetabelites.

Kui dokumente säilitatakse terviklikul kujul, on päringute optimeerimiseks võimalik kasutada indekseid. Selline lähenemine oma funktsionaalsuselt lähedane algupäraste XML andmebaaside poolt pakutavaga. Samas on aga erinevalt algupärastest XML andmebaasidest võimalik rangelt strutureeritud XML formaadis andmeid salvestada ka osadeks jaotatuna tavalistesse relatsioonilistesse andmetabelitesse, mis võimaldab kasutada nende töötlemiseks kõiki relatsioonilistes andmebaasisüsteemides olemas olevaid traditsioonilisi võimalusi.

Seega on relatsiooniliste andmebaaside XML laiendused tugevaks konkurendiks algupärastele XML andmebaasidel. Peale selle on algupäraste XML andmebaaside arendamisega tegelevad ettevõtted relatsiooniliste andmebaasisüsteemide tootjatega võrreldes suhteliselt väikesed ja noored ning sellest tulenevalt mõnevõrra ebakindlama tulevikuga. Samuti ei ole nende tuntus võrreldav suuremate relatsiooniliste andmebaaside arendajatega. Lõpkokkuvõttes on viimased isegi mõnevõrra soodsamas positsioonis. Oma osa mängib ka see, et paljud organisatsioonid kasutavad juba pikka aega relatsioonilisi andmebaasisüsteeme ning vajavad neid ka edaspidi. Kui nende abil on võimalik rahuldada ka XML dokumentide salvestamisega seotud vajadused, ei olda huvitatud sellest, et kulutada lisaresursse muude eraldiseisvate süsteemide juurutamiseks.

Nagu algupäraste XML andmebaaside, nii ka relatsiooniliste andmebaaside XML laienduste puhul on erinevate tootjate ja süsteemide poolt pakutavad võimalused ning kasutatavad tehnoloogiad suuresti erinevad ja omavahel ühildumatud, mis muudab raskeks kasutatavast andmebaasisüsteemist sõltumatute tarkvarakomponentide loomise ning nõuab spetsialistidelt ühelt andmebaasisüsteemilt teisele üleminekul märgatavat ümberkohanemist. Nende probleemide kõrvaldamiseks on asutud välja töötama uusi standardeid. Üheks olulisemaks valminud standardiks antud valdkonnas on järjekordne SQL keele laiendus "Information technology - Database Languages - SQL - Part 14: SQL/XML"ehk SQL/XML, mille esimene versioon standardiseeriti ISO poolt 15.12.2003. [sql]

SQL/XML standard käsitleb XML dokumentide genereerimist olemasolevate relatsiooniliste andmete põhjal ning XML andmetüübi kasutamist dokumentide salvestamiseks terviklikul kujul, kuid mitte dokumentide osadeks jaotamist ning relatsioonilistesse andmetabelitesse salvestamist. Selleks kasutavad erinevad andmebaasisüsteemid mitmesuguseid spetsiifilisi lahendusi. Oluliseks osaks SQL/XML standardist on ka reeglid identifikaatorite ja andmetüüpide vastavusse viimiseks XML dokumentide ja andmebaasisüsteemide vahel.

Järgnevalt vaatleme lähemalt relatsiooniliste andmebaaside XML laienduste poolt pakutavaid võimalusi XML formaadis andmetega töötamiseks.

5.1 XML dokumentide genereerimine relatsioonilis-te andmete põhjal

Suur osa olemasolevatest elektroonilisel kujul säilitatavatest andmetest on salvestatud relatsioonilistesse andmebaasisüsteemidesse. Andmetevahetuses leiab aga järjest enam kasutust XML formaat. Seetõttu on oluline, et relatsioonilistes andmebaasides paiknevaid andmeid oleks võimalik esitada XML dokumentidena.

SQL/XML standard defineerib terve rea funktsioone, mis võimaldavad genereerida relatsiooniliste andmete põhjal XML formaadis dokumente või dokumendifragmente, kasutades selleks SQL SELECT lause süntaksit. Tulemuseks on aga tabelikujulise andmehulga asemel üks või ka mitu XML formaadis dokumendifragmenti. [FMR02] Niisugused funktsioonid on järgmised:

`XMLElements()` genereerib ühe XML elemendi.

`XMLAttributes()` võimaldab lisada elemendile atribuute.

`XMLForest()` on lühendatud variant mitme järjestikuse elemendi genereerimiseks `XMLElements()` asemel.

`XMLConcat()` liidab mitu erinevat XML fragmenti kokku üheks fragmendiks, võimaldab luua kogumi järjestikkuseid XML elemente, mis ei oma ühist vanemat. Samuti lühendatud variant mitme järjestikuse `XMLElement()` konstruktsiooni asemel.

`XMLAgg()` on agregeeriv funktsioon, mis võimaldab luua XML dokumendis mitme sama tüüpi alamaga elemente mitme `SELECT` lause poolt genereeritud tabelirea põhjal. Kui `SELECT` lauses kasutatakse `GROUP BY` klauslit, koondab `XMLAgg()` kokku ühte gruppi kuuluvad read, vastasel korral kõik `SELECT` lause poolt leitud read. Tulemuseks on see, et "väljaspool"`XMLAgg()` konstruktsiooni on iga grupi või ka terve tabeli kohta üks komplekt elemente, `XMLAgg()` sees aga üks komplekt iga rea kohta. Kui kõikide muude funktsioonide korral on tulemuseks üks kõige kõrgema taseme element iga `SELECT` lause abil leitud rea kohta siis `XMLAgg()` funktsiooni kasutamisel iga grupi või terve tabeli kohta. Elementide järjekorda on võimalik määrata `ORDER BY` klausliga.

Kuigi funktsioone ei ole eriti palju, on neid sobivalt kombineerides võimalik luua peaaegu igasugust mõeldavat XML dokumenti. Järgnevalt mõned lihtsad näited.

Päringule:

```
SELECT XMLELEMENT(
    "Emp",
    XMLELEMENT("name", e.fname ||' '|| e.lname ),
    XMLELEMENT ("hiredate", e.hire)
)
FROM employees e
WHERE employee_id > 200 ;
```

Saame vastuseks:

```
<Emp>
    <name>John Smith</name>
    <hiredate>24-MAY-00</hiredate>
</Emp>
<Emp>
    <name>Mary Martin</name>
    <hiredate>01-FEB-96</hiredate>
</Emp>
```

Järgmine näide demonstreerib `XMLAgg` funktsiooni kasutamist:

```
SELECT XMLELEMENT(
    "Department",
    XMLAttributes(deptno AS "deptno"),
    XMLAgg(
        XMLElement("Employee", e.job||' '||e.ename)
    )
)
FROM scott.emp e
GROUP BY e.deptno;
```

Tulemuseks saame:

```
<Department deptno="10">
    <Employee>MANAGER CLARK</Employee>
    <Employee>PRESIDENT KING</Employee>
    <Employee>CLERK MILLER</Employee>
</Department>
<Department deptno="20">
    <Employee>CLERK SMITH</Employee>
    <Employee>ANALYST FORD</Employee>
    <Employee>CLERK ADAMS</Employee>
    <Employee>ANALYST SCOTT</Employee>
    <Employee>MANAGER JONES</Employee>
</Department>
```

Selline XML dokumentide genereerimise viis on ühine kõigile SQL/XML standardit toetavatele andmebaasisüsteemidele. Lisakas sellele on mitmetes andmebaasisüsteemides, mille puhul dokumente on võimalik salvestada osade kaupa relatsioonilistesse andmetabelitesse ja nende muutmiseks on võimalik kasutada SQL vahendeid, võimalik defineerida XML dokumendi ja relatsioonilise andmebaasiskeemi vaheline vastavus ka selliste andmete jaoks, mis on eelnevalt andmebaasi sisestadud XML tehnoloogiaid kasutamata. Selliselt on samuti võimalik teisendada relatsioonilisel kujul olevaid andmeid XML formaati. Niisuguseks andmebaasisüsteemiks on näiteks IBM DB2 versioon 8.1. Lähemalt räägime sellest dokumentide salvestamist käsitlevas osas.

5.2 XML dokumentide salvestamine relatsioonilistesse andmabaasisüsteemidesse

Lisaks XML dokumentide genereerimisele olemasolevate relatsiooniliste andmete põhjal võimaldavad XML formaati toetavad andmebaasisüsteemid salvestada andmebaasi ka eelnevalt XML formaadis olevaid dokumente.

Suure osa XML dokumentide salvestamist võimaldavate relatsiooniliste andmebaasisüsteemide puhul on dokumente võimalik salvestada kahel erineval moel: terviklikul kujul tekstina või siis jaotades dokumendid osadeks ja salvestades osad traditsioonilisi relatsioonilisi või objekt- relatsioonilisi meetodeid kasutades. Oluline on sealjuures, et osadeks jaotamise aluseks on dokumendis sisalduvate andmete, aga mitte dokumendi enda struktuur. Kõige paremini avalduvad relatsiooniliste andmebaasisüsteemide eelised just dekomponeeritud salvestusviisi juures, kuna sel juhul on võimalik ära kasutada kõiki andmebaasisüsteemi olemasolevaid võimalusi.

Probleem on aga selles, et relatsioonilised andmebaasisüsteemid võimaldavad säilitada tabelikujulise struktuuriga andmeid, ning ka seoseid erinevate tabelite vahel. Objekt- relatsioonilised andmebaasid veel lisaks keerukama struktuuriga objekte. Kumbadki aga ei võimalda vahetul kujul salvestada hierarhilisi puukujulisi andmestruktuure, nagu on XML dokumendid. Seetõttu on andmebaasisüsteemil tarvis mingit võimalust, kuidas iga konkreetse dokumendi korral otsustada, kuidas on seda kõige otstarbekam salvestada. Kuna relatsioonilised andmebaasid on ette nähtud suure hulga sama struktuuriga andmete salvestamiseks ühtsel kujul (ridadena tabelis), siis on ka sarnaseid XML dokumente otstarbekas salvestada ühtsel kujul. XML dokumentide grupeerimisel on sobivaks kriteeriumiks dokumentide vastavus samale DTD- le või schemale. Seega on tarvis salvestusformaati mitte igale dokumendile eraldi, vaid igale schemale või DTD- le vastavate dokumentide grupile.

Relatsiooniliste või objekt- relatsiooniliste andmebaaside puhul on andmete struktuur andmebaasis vastava andmebaasiskeemi poolt rangelt paika pandud ning andmed, mida on võimalik mingisse andmebaasi salvestada peavad antud skeemile täpselt vastama. XML dokumendid seevastu on oma olemuselt väga paindliku struktuuriga. Paratamatult ei ole võimalik leida üks- ühest vastavust kahele andmeformaadile, mis on üksteisest niivõrd erinevad vaid tuleb seada mingid kitsendused. Õnneks on olemas küllalt suur hulk, valdavalt andmepõhiseid XML dokumente, mille struktuur on rangelt määratletud ning formaalselt DTD või schema abil kirjeldatud. Sageli on just selliste dokumentide puhul tarvis sooritada suurt hulka erinevaid dokumente hõlmavaid päringuid mingite dokumendi osade põhjal ning võimalus kasutada relatsioonilisi vahendeid annaks kõige suurema eelise. Nagu selgub, on selliste rangelt struktureeritud dokumentide puhul juba võima-

lik luua sobiv relatsiooniline või objekt- relatsiooniline andmebaasiskeem, mis vastaks antud schemale või DTD- le, ning kuhu oleks võimalik salvestada kõiki sellele schemale või DTD- le vastavaid dokumente.

Pannes paika reeglid, kuidas seada igale võimalikule konstruktsioonile XML dokumendis vastavusse mingi konstruktsioon relatsioonilises andmebaasis, on põhimõtteliselt võimalik saavutada olukord, kus igasugust XML formaadis dokumenti on võimalik ilma kasutajapoolse täiendava sekkumisete salvestada dekomponeeritud kujul relatsioonilisse andmebaasisüsteemi. Selline võimalus on kasutaja jaoks paljudel juhtudel lihtne ja mugav. Samas ei pruugi automaatselt genereeritud andmebaasiskeem olla kõikide dokumentide jaoks optimaalne. Seetõttu puudub osade andmebaasisüsteemide puhul automaatse andmebaasiskeemi genereerimise võimalus täielikult. Ka sellistes andmebaasisüsteemides kus niisugune võimalus on olemas, on kasutajal vajaduse korral siiski võimalik loodava andmebaasiskeemi struktuuri mõjutada.

Sellest, kas dekomponeeritud dokumendisalvestusel kasutatava andmebaasiskeemi loomine toimub automaatselt või tuleb seda teha käsitsi lähtub veel terve rida muid olulisi erinevusi andmebaasisüsteemide vahel. Kui andmebaasiskeem on automaatselt genereeritud, siis ei saa ka mingite muude operatsioonide puhul eeldada, et kasutaja arvestab andmebaasi füüsilise struktuuriga.

Sellest tulenevalt erineb nimetatud kahte gruppi kuuluvate andmebaasisüsteemide puhul oluliselt ka viis kuidas toimub päringute ja muudatuste tegemine salvestatud dokumentides.

Tegelikkuses on suured erinevused ka sama andmebaasiskeemi loomise viisi kasutavate, kuid erinevate tootjate arendatavate andmebaasisüsteemide vahel. Seni puuduvad sisuliselt antud valdkonda reguleerivad standardid. Selle tulemusena on raske luua rakendusi, mis oleksid ilma suuremate muudatusteta ühilduvad rohkem kui ühe XML formaati toetatava relatsioonilise andmebaasisüsteemiga.

Kuigi SQL/XML standard käsitleb dokumentide terviklikul kujul salvestust, on ka sellisel juhul andmebaasisüsteemide vahel suured erinevused selles osas, kuidas toimub kasutatavate tabelite, veergude, andmetüüpide, indeksite ja muude salvestusega seonduvate üksikasjade kirjeldamine. Valdav osa andmebaasisüsteeme kasutab nii tervikliku kui ka dekomponeeritud salvestuse puhul samas formaadis kirjeldust, milleks on tihti kas eraldiseisev XML dokument, või siis kasutatakse schema standardis defineeritud võimalusi tootespetsiifiliste laienduste lisamiseks annotatsioonide kujul.

Seetõttu käsitlemegi järgnevalt eraldi kahte erinevat XML formaati toetavat relatsioonilist andmebaasisüsteemi, milleks on Oracle 10g ning IBM DB2 versioon 8. Neist esimene esindab automaatset andmebaasiskeemi genereerimist kasutavat lähenemist. Teise puhul tuleb kasutajal vajalik andmebaasiskeem ning selle vastavus XML dokumentidele enne andmete sisestamist käsitsi defineerida. Andmebaasiskeemi ja XML dokumentide vahelise vastavuse kirjeldamiseks kasutatak-

se eraldiseisvat XML formaadis dokumenti, mida nimetatakse Document Access Definition- iks või lühendatult DAD- ks.

5.2.1 Kasutaja defineeritud andmebaasiskeemil põhinev dokumendisalvestus

Osade XML formaati toetavate relatsiooniliste andmebaasisüsteemide puhul tuleb kasutajal või andmebaasi administraatoril enne dokumentide andmebaasi sisestamist luua käsitsi vajalikud andmebaasi tabelid ning defineerida dokumentide ja tabelite vaheline vastavus. Näiteks sellisest andmebaasisüsteemist on IBM DB2 versioon 8. XML formaadi toetus on selles andmebaasisüsteemis realiseeritud laienduspaketina olemasolevat funktsionaalsust ja laiendusvõimalusi ära kasutades. Andmebaasisüsteemi keskne mootor on jäänud endiseks ega toeta otseselt XML formaati. Selle jaoks paistavad kõik XML formaadis andmed nagu tavalised relatsioonilised andmed. Kogu XML spetsiifiline funktsionaalsus on koondatud vastavasse laienduspaketti, mida nimetatakse IBM DB2 XML Extenderiks, ning mis on realiseeritud komplekti kasutaja defineeritud funktsioonide (User Defined Function, UDF), kasutaja defineeritud andmetüüpide (User Defined Type, UDT) ja salvestatud protseduuridena (Stored procedure) . XML extender võimaldab salvestada XML dokumente nii terviklikul kui ka dekomponeeritud kujul, toetab dokumentide valideerimist DTD ja schema põhjal (kusjuures neid mõlemaid on samuti võimalik säilitada andmebaasis), sooritada XPATH päringuid ning teisendada dokumente XSLT stiililehtede abil. [IBM]

Järgnevalt näide sellest, kuidas toimub dokumendi salvestus kõigepealt terviklikul kujul ning seejärel dekomponeeritult. Kasutame XML dokumente, mis vastavad järgmisele DTD- le:

```
<!xml encoding="US-ASCII"?>
<!ELEMENT Order (Customer, Part+)>
<!ATTLIST Order key CDATA #REQUIRED>
<!ELEMENT Customer (Name, Email)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Part (key, Quantity, ExtendedPrice, Tax, Shipment+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT ExtendedPrice (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Part color CDATA #REQUIRED>
<!ELEMENT Shipment (ShipDate, ShipMode)>
<!ELEMENT ShipDate (#PCDATA)>
<!ELEMENT ShipMode (#PCDATA)>
```

Toodud DTD- le vastab näiteks järgmine dokument:

```
<?xml version="1.0"?>
```

```
<!DOCTYPE Order SYSTEM "dxx_install/samples/db2xml/dtd/getstart.dtd">
<Order key="1">
    <Customer>
        <Name>American Motors</Name>
        <Email>parts@am.com</Email>
    </Customer>
    <Part color="black ">
        <key>68</key>
        <Quantity>36</Quantity>
        <ExtendedPrice>34850.16</ExtendedPrice>
        <Tax>6.000000e-02</Tax>
        <Shipment>
            <ShipDate>1998-08-19</ShipDate>
            <ShipMode>BOAT </ShipMode>
        </Shipment>
        <Shipment>
            <ShipDate>1998-08-19</ShipDate>
            <ShipMode>AIR </ShipMode>
        </Shipment>
    </Part>
    <Part color="red ">
        <key>128</key>
        <Quantity>28</Quantity>
        <ExtendedPrice>38000.00</ExtendedPrice>
        <Tax>7.000000e-02</Tax>
        <Shipment>
            <ShipDate>1998-12-30</ShipDate>
            <ShipMode>TRUCK </ShipMode>
        </Shipment>
    </Part>
</Order>
```

Kõigepealt vaatleme, kuidas niisuguseid dokumente on võimalik salvestada andmebaasi terviklikul kujul. Selleks tuleb läbida järgmised sammud:

1. Otsustada millist andmetüüpi kasutada dokumendi salvestamiseks.
2. Otsustada milliseid dokumendi osi tuleks indekseerida.
3. Salvestada andmebaasis DTD- d.
4. Luua andmebaasiskeem ja võimaldada sinna XML dokumentide salvestamine
5. Luua DAD kirjeldus.
6. Luua vajalikud indeksid.
7. Sisestada andmebaasi XML dokumendid.

DB2 XML extender defineerib kolm XML andmetüüpi, milleks on XMLVARCHAR, XMLCLOB ja XMLFILE. Kaks esimest on sisuliselt samad, mis VARCHAR ja CLOB andmetüübid, ainult et on määratletud, et praegusel juhul

on sinna salvestatud terviklik XML dokument, XMLFILE võimaldab dokumente salvestada failisüsteemi väljaspoole andmebaasi. Kuna antud juhul on dokumendid suhteliselt lühikesed, kasutame XMLVARCHAR andmetüüpi. Dokumentide salvestamiseks kasutame tabelit SALES_TAB, mille struktuur on toodud tabelis 5.1:

Column name	**Data type**
INVOICE_NUM	CHAR(6) NOT NULL PRIMARY KE
SALES_PERSON	VARCHAR(20)
ORDER	XMLVARCHAR

Tabel 5.1: SALES_TAB

Selleks, et päringute tegemisel XML dokumentide suhtes ei oleks alati tarvis kõiki dokumente täielikult läbi vaadata, on võimalik sobivaid dokumendi osi indekseerida. Et indekseerimine oleks optimaalne, peab olema eelnevalt teada, milliste dokumendi osade suhtes on oodata kõige rohkem päringuid või millised päringud võivad võtta kõige rohkem aega. Praegusel juhul eeldame, et indekseerida tuleks dokumentide osad, mis on toodud tabelis 5.2

Data	**Location path**
order key	/Order/@key
customer	/Order/Customer/Name
price	/Order/Part/ExtendedPrice
shipping date	/Order/Part/Shipment/ShipDate

Tabel 5.2:

IBM DB2 XML Extender võimaldab indekseerimiseks kasutada nn. "kõrvaltabeleid"(Side tabels). See milliseid kõrvaltabelid mingite dokumentide puhul kasutatakse defineeritakse DAD kirjelduses. Need tabelid tuleb kasutajal eelnevalt käsitsi luua, samuti tuleb luua neile indeksid. Edaspidi haldab kõrvaltabelid andmebaasisüsteem iseseisvalt. Kasutaja ei tohiks teha neisse otseselt mingisuguseid muudatusi, kuigi see võimalus on täiesti olemaset. Uute dokumentide lisamisel ning olemasolevate muutmisel või eemaldamisel hoolitseb kõrvaltabelite uuendamise ees andmebaasisüsteem.

Käesolevas näites kasutame kõrvaltabeleid, mis on toodud joonistel 5.3 kuni 5.5:

Selleks, et panna paika dokumentide salvestusega seotud detailid nagu see, millist salvestusviisi kasutatakse, millisesse tabelisse ja millisesse veergu dokumendid salvestatakse, milliseid kõrvaltabeleid kasutatakse ja millised dokumendi

Column name	Data type	Location path	Multiple occurring?
ORDER_KEY	INTEGER	/Order/@key	No
CUSTOMER	VARCHAR(16)	/Order/Customer/Name	No

Tabel 5.3: ORDER_SIDE_TAB

Column name	Data type	Location path	Multiple occurring?
Column name	Data type	Location path	Multiple occurring?
PRICE	DECIMAL(10,2)	/Order/Part/ExtendedPrice	Yes

Tabel 5.4: PART_SIDE_TAB

osad sinna tuleb salvestada, kasutatakse IBM DB2 XML Extenderi puhul DAD kirjeldust. Meie näite puhul näeks DAD kirjeldus välja niisugune:

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
    <dtdid>
        "dxx_install/samples/db2xml/dtd/getstart.dtd"
    </dtdid>
    <validation>YES</validation>

    <Xcolumn>
        <table name="order_side_tab">
            <column name="order_key" type="integer"
                path="/Order/@key" multi_occurrence="NO"/>
            <column name="customer" type="varchar(50)"
                path="/Order/Customer/Name" multi_occurrence="NO"/>
        </table>
        <table name="part_side_tab">
            <column name="price" type="decimal(10,2)"
                path="/Order/Part/ExtendedPrice" multi_occurrence="YES"/>
        </table>
        <table name="ship_side_tab">
            <column name="date" type="DATE"
                path="/Order/Part/Shipment/ShipDate" multi_occurrence="YES"/>
        </table>
    </Xcolumn>
</DAD>
```

Loodud kirjelduse salvestame faili `getstart_xcolumn.dad` . Kasutatav element `Xcolumn` määrab, et kasutatavaks salvestusmeetodiks on dokumentide terviklik salvestamine XML andmetüüpi veergu.

Enne kui andmebaasi on võimalik salvestada XML dokumente, tuleb luua vajalikud andmebaasitabelid ja indeksid, ning määrata XML andmetüüpi veeru kohta, et seda kasutatakse XML dokumentide salvestamiseks.

Selleks, et kasutada ära kõrvaltabeleid ja indekseid, tuleb nendega päringu

Column name	**Data type**	**Location path**	**Multiple occurring?**
Column name	Data type	Location path	Multiple occurring?
DATE	DATE	/Order/Part/Shipment/ShipDate	Yes

Tabel 5.5: SHIP_SIDE_TAB

tegemisel ilmutatud kujul arvestada. Järgmine päring kasutab ainult XML dokumenti sisaldavas tabelis ja kõrvaltabelites olevaid relatsioonilisi andmeid:

```
DB2 "SELECT DISTINCT SALES_PERSON
FROM SALES_TAB S, PART_SIDE_TAB P
WHERE PRICE > 2500.00 AND S.INVOICE_NUM=P.INVOICE_NUM"
```

Kui mingit päritavat dokumendi osa ei ole kõrvaltabelitesse salvestatud, tuleb kasutada dokumendi läbivaatus. Dokumendi osi aderesseeritakse XPath süntaksi abil, kasutades spetsiaalseid funktsioone nagu `extractVarchar()`, `extractInteger()`, `extractDate()` jne. Seekordses näites kasutatame tabelite ühendamise asemel XML tüüpi veeru aktiveerimisel loodud vaikimisi vaadet.

```
SELECT extractVarchar(Order, '/Order/Customer/Name')
FROM sales_order_view
WHERE price > 2500.00
```

Kui süsteemi on installeeritud IBM DB2 Text Extender, on seda võimalik kasutada täisteksti otsinguks XML dokumentidest. Lähemalt sellel siin praegu ei peatu.

Dokumentide muutmisel on lisaks täielikule asendamisel `UPDATE` käsuga võimalik kasutada funktsiooni `Update(xmlobj, path, value)`, mis uuendavad ainult mingi osa dokumendist, näiteks:

```
UPDATE sales_tab
SET order = Update(order, '/Order/Customer/Name', 'IBM')
WHERE sales_person = 'Sriram Srinivasan'
```

Teiseks viisiks, kuidas IBM DB2 XML Extender võimaldab dokumente salvestada on dokumentide jaotamine osadeks ja salvestamine relatsioonilistesse andmetabelitesse. Selleks tuleb kõigepealt välja töötada sobiv andmebaasiskeem, kuhu oleks võimalik dokumentide osad salvestada. Praegusel juhul kasutame tabeleid ORDER_TAB, PART_TAB ja SHIP_TAB:

DAD, mis kirjeldab, kuidas salvestada XML dokumente nendesse tabelitesse, oleks selline:

Column name	Data type
ORDER_KEY	INTEGER
CUSTOMER	VARCHAR(16)
CUSTOMER_NAME	VARCHAR(16)
CUSTOMER_EMAIL	VARCHAR(16)

Tabel 5.6: ORDER_TAB

Column name	Data type
PART_KEY	INTEGER
COLOR	CHAR(6)
QUANTITY	INTEGER
PRICE	DECIMAL(10,2)
TAX	REAL
ORDER_KEY	INTEGER

Tabel 5.7: PART_TAB

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
    <dtdid>dxx_install/samples/db2xml/dtd/getstart.dtd </dtdid>
    <validation>YES</validation>
    <Xcollection>
        <prolog>?xml version="1.0"?</prolog>
        <doctype>
            !DOCTYPE Order SYSTEM "dxx_install/samples/db2xml/dtd/getstart.dtd"
        </doctype>

        <root_node>
            <element_node name="Order">
                <RDB_node>
                    <table name="order_tab"/>
                    <table name="part_tab"/>
                    <table name="ship_tab"/>
                    <condition>
                        order_tab.order_key=part_tab.order_key AND
                        part_tab.part_key=ship_tab.part_key
                    </condition>
                </RDB_node>

                <attribute_node name="Key">
                    <RDB_node>
                        <table name="order_tab"/>
                        <column name="order_key"/>
                    </RDB_node>
                </attribute_node>

                <element_node name="Customer">
                    <element_node name="Name">
```

Column name	Data type
DATE	DATE
MODE	CHAR(6)
COMMENT	VARCHAR(128)
PART_KEY	INTEGER

Tabel 5.8: SHIP_TAB

```
                    <text_node>
                        <RDB_node>
                            <table name="order_tab"/>
                            <column name="customer_name"/>
                        </RDB_node>
                    </text_node>
                </element_node>
                <element_node name="Email">
                    <text_node>
                        <RDB_node>
                            <table name="order_tab"/>
                            <column name="customer_email"/>
                        </RDB_node>
                    </text_node>
                </element_node>
            </element_node>
            <element_node name="Part">
                <attribute_node name="Key">
                    <RDB_node>
                        <table name="part_tab"/>
                        <column name="part_key"/>
                    </RDB_node>
                </attribute_node>

                <element_node name="...">
                    ...

                </element_node>

                ...


                ...

            </element_node>
        </element_node>
    </root_node>
</Xcollection>
</DAD>
```

<xcollection> element näitab, et kasutatakse dekomponeeritud salvestusviisi. XML dokumendi koostisosad on tähistatud elementidega <element_node> , <attribute_node> ja <text_node> . Neist igaüks sisaldab <RDB_node> elementi, mille alamad <table> ja <column> määravad kuhu on vastav komponent andmebaasis salvestatud. <condition> element määrab ära tingimu-

sed tabelite ühendamisel. Dokumendi juurelement peab sisaldama `<table>` elemente kõikide kasutatavate tabelite kohta ning juhul kui tabeleid on rohkem kui üksi, `<condition>` elementi mis määrab ära seosed tabelite vahel. Tingimused peavad vastama primaarvõtme- välisvõtme suhetele tabelites. `<prolog>` ja `<doctype>` elemendid määravad ära vastavad parameetrid dokumentide genereerimisel. Kui DAD kirjeldus on kasutatav ainult dokumentide salvestamiseks, võib need ka ära jätta.

Dokumentide salvestamiseks tuleb kasutada mõnda selleks mõeldud salvestatud protseduuri, näiteks `db2xml.dxxShredXML()`. Sellele antakse parameetriteks DAD kirjeldus ning XML dokument, protseduur tagastab staatuse koodi ning tekstilise teade.

```
db2 CALL db2xml.dxxShredXML(
    db2xml.XMLVarcharFromFile
        ('dxx_install/samples/db2xml/dad/getstart_xcollection.dad'),
    db2xml.XMLVarcharFromFile
        ('dxx_install/samples/db2xml/xml/getstart.xml'),
    ?,?
)
```

XML dokumentide genereerimine relatsiooniliste andmete põhja DAD kirjeldust kasutades toimub XML Extenderi poolt defineeritud salvestatud protseduuride abil. Võimalikud protseduurid on `dxxGenXML()`, `dxxGenXMLCLOB()`, `dxxRetrieveXML()` ja `dxxRetrieveXMLCLOB()`. Nimetatud protseduurid salvestavad genereeritud XML dokumendid ette antud tabelisse, mille kasutaja peab olema eelnevalt loonud ja mis peab sisaldama täpselt ühte XML andmetüüpi veergu. Parameetriteks on DAD kirjeldus ja tabeli nimi, kuhu tulemus salvestada.

Protseduur tagastab genereeritud dokumentide arvu, staatuse koodi ning veateate.

5.2.2 Automaatselt genereeritud andmebaasiskeemil põhinev dokumendisalvestus

Mitmed tänapäevased relatsioonilised andmebaasisüsteemid võimaldavad salvestada objektikujulisi andmestruktuure. Neid nimetatakse objekt- relatsioonilistes andmebaasisüsteemideks. Selliste andmebaasisüsteemide edasisel täiustamisel võimaldamaks kasutada neid ka XML formaadis dokumentide säilitamiseks on võimalik ära kasutada olemasolevaid võimalusi objektide salvestamiseks.

XML formaadis dokumendid kujutavad endast põhimõtteliselt puukujulist objektide hierarhiat. Pannes paika üldised reeglid milline objektistruktuur seada vastavusse igale võimalikule konstruktsioonile XML dokumentides, on võimalik saavutada olukord, kus iga dokumendi jaoks on võimalik luua sellele üks- üheselt

vastav objektide hierarhia. Kuna objekt- relatsioonilised andmebaasisüsteemid võimaldavad salvestada suvalisi objekte, tekib võimalus salvestada ilma kasutajapoolese sekkumisete ka kõikvõimalikke XML formaadis dokumente.

Näiteks sellist lähenemist kasutavast objekt- relatsioonilisest andmebaasisüsteemist on Oracle 10g XML DB. Edaspidi vaatlemegi süsteemselt genereeritud andmebaasiskeemil põhineva dokumendisalvestuse eripärasid lähemalt selle andmebaasisüsteemi näitel.

Erinevalt IBM DB2 XML Extenderist on Oracle XML DB puhul XML formaadi toetus integreeritud andmebaasi kesksesse mootorisse. [Ora03] Selline arhitektuur annab mitmeid eeliseid, võimaldades varjata kasutaja eest dokumentide salvestamisega seotud üksikasju ning seega muuta töö kasutaja jaoks lihtsamaks. XML dokumentide salvestamiseks kasutatakse XMLType andmetüüpi, mis on oma olemuselt objektitüüp, nii et seda võivad kasutada üksikud veerud, aga võimalik on luua ka terveid XMLType tüüpi andmebaasitabelid. Dokumentide grupeerimisel võetakse aluseks nende vastavus samale XML schemale. Iga XMLType tüüpi tabeli või veeruga seotakse kindel XML schema, mille põhjal andmebaasisüsteem otsustab, millist objektistruktuuri kasutades antud schemale vastavaid dokumente salvestada. Vajadusel on kasutajal võimalik dokumentide salvestamise viisi mõjutada, kasutades selleks spetsiaalseid schema laiendusi (annotations) . Võimalik on kasutad nii terviklikku kui struktureeritud salvestusviisi. Kui schema on olemas ja kirjeldab dokumentide struktuuri piisavalt rangelt, eelistatakse vaikimisi struktureeritud salvestusviisi. Muudel juhtudel kasutatakse terviklikku salvestust. Dokumente, millel schema üldse puudub on võimalik salvestada ainult terviklikul kujul. Päringute ja muudatuste tegemine dokumentidesse toimub kõikidel juhtudel ühtemoodi, sõltumata sellest, kuidas dokumendid on füüsiliselt salvestatud. Erinevate salvestusviiside omapärasid arvestab andmebaasisüsteem, optimeerides päringuid sellele vastavalt. Isegi relatsioonilist salvestusviisi kasutavate dokumentide puhul ei kasutata relatsioonilisi päringuid, vaid üldisi XML spetsiifilisi päringuid mille andmebaasisüsteem teisendab automaatselt vastavateks relatsioonilisteks päringuteks.

Dokumentide salvestamiseks Oracle XML DB andmebaasi tuleb läbida järgmised sammud:

1. Luua schema.
2. Lisada vajaduse korral schemale annotation- id
3. Registreerida schema.
4. Luua andmebaasitabelid.
5. Lisada vajalikud indeksid.

6. Sisestada dokumendid andmabaasi.

XML dokumentide salvestamisel Oracle XML DB andmebaasisüsteemi mängib keskset rolli dokumentide schema. Lisaks dokumentide valideerimisele kasutatakse seda ka dokumentide salvestamisega seonduvate detailide paika panemiseks. Struktureeritud salvestuse aluseks on dokumentide struktuuri jaotamine vastavateks SQL objektitüüpideks. Iga schemas leiduva `complexType` tüübi kohta luuakse sobiv SQL objektitüüp, atribuutidest ja `PCDATA` tüüpi elementidest saavad objektide jaoks attribuudid. Kui mingi element võib esineda dokumendi struktuuris mitmekordselt, luuakse nende salvestamiseks andmebaasis `VARRAY` tüüpi objekt. Vajalikud objektitüübid luuakse, kui toimub vastava XML schema registreerimine. Edaspidi on kõiki antud schemale vastavaid dokumente võimalik salvestada, kasutades selleks loodud SQL objektitüüpe.

Edaspidistes näidetes kasutame järgmist dokumenti.

```
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=
            "http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
    <Reference>SBELL-20021009123336O1PDT</Reference>
    <Actions>
        <Action>
            <User>SVOLLMAN</User>
        </Action>
    </Actions>
    <Reject/>
    <Requestor>Sarah J. Bell</Requestor>
    <User>SBELL</User>
    <CostCenter>S30</CostCenter>
    <ShippingInstructions>
        <name>Sarah J. Bell</name>
        <address>
            400 Oracle Parkway Redwood Shores CA 94065 USA
        </address>
        <telephone>650 506 7400</telephone>
    </ShippingInstructions>
    <SpecialInstructions>
        Air Mail
    </SpecialInstructions>
    <LineItems>
        <LineItem ItemNumber="1">
            <Description>A Night to Remember</Description>
            <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
        </LineItem>
        <LineItem ItemNumber="2">
            <Description>The Unbearable Lightness Of Being</Description>
            <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
        </LineItem>
        <LineItem ItemNumber="3">
            <Description>Sisters</Description>
            <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
        </LineItem>
    </LineItems>
</PurchaseOrder>
```

W3C XML Schema võimaldab dokumendi schemasse lisada rakenduse spetsiifilise laiendusi, nn annotatsioone. Oracle XML DB kasutab seda võimalust, et kasutajal oleks võimalik määrata ja mõjutada mingi schema põhjal genereeritavate objektidega seonduvaid parameetreid. Lisades schemale annotatsioone on kasutajal võimalik määrata loodavate objektide ja atribuutide nimesid, muuta nende struktuuri ning määrata tabel, kuhu antud schemale vastad dokumendid vaikimisi salvestatakse. XML DB spetsiifilised annotatsioonid kasutavad nimeruumi `http://xmlns.oracle.com/xdb`.

Järgnevalt näide sellest, milline võib välja näha annoteeritud schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xdb="http://xmlns.oracle.com/xdb" version="1.0"
        xdb:storeVarrayAsTable="true">
    <xs:element name="PurchaseOrder" type="PurchaseOrderType"
        xdb:defaultTable="PURCHASEORDER"/>
    <xs:complexType name="PurchaseOrderType" xdb:SQLType="PURCHASEORDER_T">
        <xs:sequence>
            <xs:element name="Reference" type="ReferenceType" minOccurs="1"
                xdb:SQLName="REFERENCE"/>
            <xs:element name="Actions" type="ActionsType"
                xdb:SQLName="ACTIONS"/>
            <xs:element name="Reject" type="RejectionType" minOccurs="0"
                xdb:SQLName="REJECTION"/>
            <xs:element name="Requestor" type="RequestorType"
                xdb:SQLName="REQUESTOR"/>
            <xs:element name="User" type="UserType" minOccurs="1"
                xdb:SQLName="USERID"/>
            <xs:element name="CostCenter" type="CostCenterType"
                xdb:SQLName="COST_CENTER"/>
            <xs:element name="ShippingInstructions" type="ShippingInstructionsType"
                xdb:SQLName="SHIPPING_INSTRUCTIONS"/>
            <xs:element name="SpecialInstructions" type="SpecialInstructionsType"
                xdb:SQLName="SPECIAL_INSTRUCTIONS"/>
            <xs:element name="LineItems" type="LineItemsType"
                xdb:SQLName="LINEITEMS"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="LineItemsType" xdb:SQLType="LINEITEMS_T">
        <xs:sequence>
            <xs:element name="LineItem" type="LineItemType" maxOccurs="unbounded"
                xdb:SQLName="LINEITEM" xdb:SQLCollType="LINEITEM_V"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="LineItemType" xdb:SQLType="LINEITEM_T">
        <xs:sequence>
            <xs:element name="Description" type="DescriptionType"
                xdb:SQLName="DESCRIPTION"/>
            <xs:element name="Part" type="PartType" xdb:SQLName="PART"/>
        </xs:sequence>
        <xs:attribute name="ItemNumber" type="xs:integer"
            xdb:SQLName="ITEMNUMBER" xdb:SQLType="NUMBER"/>
    </xs:complexType>
    <xs:complexType name="PartType" xdb:SQLType="PART_T">
        <xs:attribute name="Id" xdb:SQLName="PART_NUMBER" xdb:SQLType="VARCHAR2">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:minLength value="10"/>
```

```
                    <xs:maxLength value="14"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="Quantity" type="moneyType" xdb:SQLName="QUANTITY"/>
        <xs:attribute name="UnitPrice" type="quantityType" xdb:SQLName="UNITPRICE"/>
    </xs:complexType>

    <xs:complexType name="...">
        ...

    </xs:complexType>

    ...


    ...

</xs:schema>
```

Kasutatud annotatsioonidest määrab `defaultTable` , et antud schemale vastavad dokumendid salvestatakse tabelisse nimega `PurchaseOrder`. `SQLType` ja `SQLName` annotatsioonid võimaldavad kasutajal määrata nimesid genereeritavatele SQL objektitüüpidele ja nende atribuutidele.

XML dokumentide sisestamisel andmebaasi otsustatakse millisesse tabelisse dokument salvestada selle alusel, millisele schemale dokument vastab. Dokumenti on võimalik schemaga siduda andes selle salvestamiseks kasutatavale protseduurile või `XMLType` tüübi konstruktorile vajaliku schema identifikaatori parameetrina ette. Teise võimalusena peab dokumendis endas olema vastava schema identifikaator määratud, kasutades juurelemendil standardseid `xsi:noNamespaceSchemaLocation` või `xsi:schemaLocation` parameetreid.

Dokumentide pärimisel XMLType tüüpi tabelites kasutatakse `object_value` võtmesõna, mis asendab päringus veeru nime, näiteks:

```
SELECT object_value
FROM PURCHASEORDER;

OBJECT_VALUE
--------------------------------------------------------------------------------
--------------------------------------------------
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=
        "http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
    <Reference>SBELL-20021000912333601PDT</Reference>
    <Actions>
        <Action>
            <User>SVOLLMAN</User>
...

1 row selected.
```

Päringute ja uuenduste tegemiseks XML dokumentidele kasutab Oracle XML DB XPath süntaksit toetavaid funktsioone. Päringute tegemiseks kasutatavad funkt-

sioonid on `extract()`, `extractValue()`, `existsNode()` ja `XMLSequence()`. `Extract()` leiab XPath avaldisega määratud osa või osad XML dokumendist ning tagastab need XMLType tüüpi objektina. `ExtractValue()` leiab dokumendis ühe teksitipu või atribuudi väärtuse ning tagastab selle kasutades sobivat skalaarset SQL andmetüüpi. Kui kasutatud XPath avaldis vastab mõnele mittetekstilisele tipule, tagastatakse veateade. `ExistsNode()` võimaldab kontrollida, kas antud dokumendis leidub mingile XPath avaldisele vastav tipp. `Extract()` funktsioon leiab paljudel juhtudel mitmest omavahel mitte seotud elemendist koosneva dokumendifragmendi. Sellisel juhul ei ole saadud tulemusele võimalik rakendada `ExtractValue()` funktsiooni. Appi tuleb `XMLSequence()`, mis teisendab ühe mitmest elemendist koosneva XMLType tüüpi objekti mitmeks `XMLType` tüüpi objektiks, millest igaüks sisaldab ühte elementi. Saadud objektidest on `Table()` funktsiooni abil võimalik luua virtuaalne tabel, millest on võimalik vajalikud väärtused `ExtractValue()` abil kätte saada.

Järgmises näites leitakse kõikide tellitud kaubaartiklite kirjeldused ühest tellimusest:

```
SELECT extractValue(value(d), '/Description')
FROM purchaseorder p,
    table (xmlsequence(extract(p.object_value,
        '/PurchaseOrder/LineItems/LineItem/Description'))) d
WHERE existsNode(
    p.object_value, '/PurchaseOrder[Reference="SBELL-20021009123336O1PDT"]') = 1;

EXTRACTVALUE(VALUE(D),'/DESCRIPTION')
--------------------------------------------------------------------------------
A Night to Remember
The Unbearable Lightness Of Being
Sisters

3 rows selected.
```

Kasutades eespool toodud funktsioone on võimalik luua XML formaadis andmetele relatsioonilisi vaateid:

```
CREATE OR REPLACE view PURCHASEORDER_MASTER_VIEW (REFERENCE, REQUESTOR,
    USERID, COSTCENTER, SHIP_TO_NAME,
    SHIP_TO_ADDRESS, SHIP_TO_PHONE, INSTRUCTIONS
)
AS SELECT
    extractValue(value(p),'/PurchaseOrder/Reference'),
    extractValue(value(p),'/PurchaseOrder/Requestor'),
    extractValue(value(p),'/PurchaseOrder/User'),
    extractValue(value(p),'/PurchaseOrder/CostCenter'),
    extractValue(value(p),'/PurchaseOrder/ShippingInstructions/name'),
    extractValue(value(p),'/PurchaseOrder/ShippingInstructions/address'),
    extractValue(value(p),'/PurchaseOrder/ShippingInstructions/telephone'),
    extractValue(value(p),'/PurchaseOrder/SpecialInstructions')
FROM PURCHASEORDER p;

View created.
```

```
DESCRIBE PURCHASEORDER_MASTER_VIEW
Name Null? Type
----------------------------------------------------------------------
REFERENCE VARCHAR2(30 CHAR)
REQUESTOR VARCHAR2(128 CHAR)
USERID VARCHAR2(10 CHAR)
COSTCENTER VARCHAR2(4 CHAR)
SHIP_TO_NAME VARCHAR2(20 CHAR)
SHIP_TO_ADDRESS VARCHAR2(256 CHAR)
SHIP_TO_PHONE VARCHAR2(24 CHAR)
INSTRUCTIONS VARCHAR2(2048 CHAR)
```

Dokumentide uuendamine toimub samuti XPath süntaksit kasutades `updateXML()` funktsiooni abil. Kasutaja jaoks toimub nii tervikuna kui ka dekomponeeritud kujul säilitatavate dokumentide muutmine samamoodi, Salvestusviisist tulenevate eripärade eest hoolitseb jällegi täielikult andmebaasisüsteem. Terviklikul kujul säilitatavate dokumentide muutmiseks luuakse kõigepealt olemasolevast dokumendis DOM puu, rakendatakse sellele vajalikud uuendused ning salvestatakse muudetud puu andmebaasi tagasi, dekomponeeritud kujul salvestatavate dokumentide puhul genereeritakse vajalikud SQL `UPDATE` käsklused ning rakendatakse need vajalikele andmetele.

```
UPDATE PURCHASEORDER
SET object_value = updateXML(
    object_value,
    '/PurchaseOrder/LineItems/LineItem/Description[text()="Sisters"]/text()',
    'The Wizard of Oz'
)
WHERE existsNode(object_value,
    '/PurchaseOrder[Reference="SBELL-20021009123336O1PDT"]') = 1;

1 row updated.
```

Oracle XML DB toetab kolme tüüpi indekseid:

1. Tekstipõhised indeksid
2. Funktsioonipõhised indeksid
3. Traditsioonilised B- puu indeksid.

Neist kahte esimest on võimalik kasutada kõikide salvestusviiside puhul, B-puu indekseid on aga võimalik kasutada ainult struktureeritud kujul salvestatud dokumentide indekseerimiseks. Nii nagu dokumentide andmebaasi sisestamisel ja päringute tegemisel, varjab ka indeksite loomisel andmebaasisüsteem kasutaja eest salvestusega seotud iseärasused. Tema jaoks toimub kõikide indeksite loomine ühtemoodi. Indeksite loomisel kasutatakse päringute tegemise funktsioone, näiteks `extractValue()`.

```
CREATE INDEX PURCHASEORDER_USER_INDEX
ON PURCHASEORDER (extractValue(object_value,'/PurchaseOrder/User'));
```

5.3 Relatsioonilised andmeteisendajad

XML dokumente on võimalik relatsioonilistesse andmebaasidesse salvestada ka kasutades relatsioonilisi andemeteisendajaid. Need on andmebaasisüsteemidest eraldiseisvad tarkvarakomponendid, mille abil on võimalik salvestada rangelt struktureeritud XML dokumente dekomponeeritud kujul relatsioonilistesse andmebaasisüsteemidesse. Harilikult on nad ühendatud teekide kujul loodava tarkvarakomponendi koosseisu. Kasutatava andmebaasisüsteemi suhtes on ainsaks nõudeks see, et ta toetaks vajalikku liidese standardit nagu JDBC, ODBC või Perl DBI. Kogu suhtlus andmebaasisüsteemiga toimub SQL keelt kasutades, XML spetsiifika ees hoolitseb täielikult andmeteisendaja.

Mitmed siia kategooriasse kuuluvad tooted on tegelikult võimsad andmeteisendajad mis toetavad muude formaatide hulgas ka XML dokumente ja relatsioonilisi andmebaasisüsteeme. Samas võib relatsioonilisteks andmeteisendajateks lugeda ka ise arendatud tarkvarakomponendid, mis võimaldavad mingil konkreetsel ja täpselt piiritletud juhul XML formaadis andmeid andmebaasi salvestada või andmebaasipäringute põhjal XML dokumente luua.

Valdava osa relatsiooniliste andmeteisendajate puhul tuleb XML dokumentide skeemi ja andmebaasiskeemi vaheline vastavus eelnevalt käsitsi defineerida. Selleks kasutatakse mingit spetsiifilist kirjelduskeelt, mis on harilikult samuti XML formaadis. Mitmed tooted sisaldavad selle vastavuse loomiseks ka graafilisi tööriistu.

Realisatsiooni poolest on eristatavad kahte liiki relatsioonilised andmeteisendajad. Esimesel juhul jaotatakse dokumendid vastavuse kirjeldust aluseks võttes osadeks ning salvestatakse saadud osad otse andmebaasi. Teisel juhul toimub teisendamine kahes osas. Kõigepealt luuakse XML dokumendi põhjal vastavad Java- või mõne teise objekt- orienteeritud programmeerimiskeele objektid, seejärel põlistatakse (persistence) saadud objektid relatsioonilist andmebaasi kasutades. Vastavuse kirjeldust kasutatakse sel juhul peamiselt objektide ja andmebaasiskeemi vahelise seose määratlemiseks, objektide genereerimine XML dokumentide põhjal võib sõltuvalt kasutatavast tootest toimuda automaatselt.

Kuigi funktsionaalsuselt on relatsioonilised andmeteisendajad lähedased tabelipõhist salvestusviisi kasutavatele XML formaati toetavatele relatsioonilistele andmebaasisüsteemidele, jäävad nad viimastele jõudluse poolest alla, seda eriti juhul kui on tervis sooritada dokumentidest XML põhiseid päringuid XPath või XQuery vahendeid kasutades. Kuna tegemist on andmebaasist eraldiseisvate ning sageli ka erineval arvutil töötavate tarkvarakomponentidega, tuleb kõikide pärita-

vate dokumentide jaoks vajalikud andmed üle võrgu andmebaasisüsteemist kohale toimetada ja neist dokumendid kokku panna. Alles seejärel on võimalik sooritada dokumentide suhtes päringuid. Kokkuvõttes tuleb üle kanda tunduvalt suurem infohulk, samuti ei ole võimalik kasutada päringute optimeerimiseks indekseid. Samas on andmeteisendajad sobivaks lahenduseks juhul, kui valdav osa päringuid on relatsioonilised. XML põhiste päringute ülekaalu puhul tuleks nende asemel tõsiselt kaaluda algupäraste XML andmebaasi kasutamist.

Relatsioonilisi andmeteisendajaid kasutades on võimalik rangelt struktureeritud XML dokumente salvestada ka sellistesse andmebaasisüsteemidesse, millel endal XML formaadi toetus puudub. Eelkõige on sellisteks andmebaasisüsteemideks vabavaralised tooted nagu MySQL ja PostgreSQL. Ka relatsiooniliste andmeteisendajate hulgas leidub mitmeid vabavaralisi tooteid. Seega kui nõudmised loodava lahenduse jõudlusele ei ole väga kõrged ning päringud on valdavalt relatsioonilised, võimaldab nende kasutamine hoida kokku kulutusi komertsiaalsete andmebaasisüsteemide hankimiselt, kuid kasutada dokumentidega töötamisel siiski relatsiooniliste andmebaasisüsteemide eeliseid.

Peatükk 6

Kokkuvõte

XML formaadis andmete salvestamiseks on olemas terve rida erinevaid võimalusi, alustades lihtsatest failidest failisüsteemis ning lõpetades keerukate algupäraste XML andmebaasidega. Sellele vaatamata ei ole sobiva tehnoloogia valik mingi konkreetse vajaduse rahuldamiseks triviaalne ülesanne. Et leida mingil konkreetsel juhul sobivaim viis andmete salvestamiseks, tuleb võtta arvesse dokumentide struktuuri ja päringuid mida antud dokumentide suhtes on tarvis sooritada. Kuna valdkond on suhteliselt vähe standardiseeritud, tuleb lisaks mingi toote poolt pakutavale funktsionaalsusele arvestada ka selle ühilduvust muude kasutatavate komponentidega. Näiteks on mitmete toodete ja standardite puhul piiratud toetatavate programmeerimiskeelte nimekiri. Paljud tooted on omavahel ühildumatud. Seetõttu saab kogu loodav infosüsteem paratamatult tihedalt seotud kasutatava tootega ning hilisem üleminek mõnele muule süsteemile võib olla vägagi valulik. Eriti algupäraste XML andmebaaside puhul on maastik vägagi dünaamiline. Mõnigi toode mis praegu tundub arenev ja elujõuline võib mõne aja möödudes sellest hoolimata alla käia ning kaduda. Valikul relatsiooniliste andmebaaside ja algupäraste XML andmebaaside vahel on määrava tähsusega see, kas kogu ifosüsteemi arhitektuur põhineb XML põhistel tehnoloogiatel või mitte.

Lisaks komertsiaalsetele süsteemidele on olemas ka mitmeid arvestatavaid vabavaralisi lahendusi.

Peatükk 7

Abstract

Ever spreading use of XML format in data management and representation has created need for efficient storage and querying technologies for XML documents. To accomplish these needs, new kinds of solutions, such as native XML databases, XML- enabled relational database management systems and XML middleware products have been developed during the last years. Fast impovement in this area is expected to continue.

In this work, review of such new tecnologies is given. First, we review XML related standard like XQuery, XPath, SQL/XML, XUpdate and XML:DB API. Next, the native XML databses in general and products like eXist and Tamino in particular are reviewd. Last, we review XML- enabled relational database management systems Oracle 10g and IBM DB2.

Motivation for the work comes from the practical needs to store NLM Medline cientific article abstracts that are distributed in XML format. During writing the current thesis we experimented with several of the above systems.

Kirjandus

[Bou03] Ronald Bourret. XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm, 2003.

[Bou04] Ronald Bourret. XML Database Products. http://www.rpbourret.com/xml/XMLDatabaseProds.htm, 2004.

[CRZ03] Akmal B. Chaudhri, Awais Rashid, and Roberto Zicari. *XML Data Management: Native XML and and XML- Enabled Database Systems*. Addison-Wesley, 1st edition, 2003.

[dom] Document Object Model (DOM). http://www.w3.org/DOM/.

[FMR02] J. E. Funderburk, S. Malaika, and B. Reinwald. XML programming with SQL/XML and XQuery. http://www.research.ibm.com/journal/sj/414/reinwald.pdf, 2002.

[htm] HyperText Markup Language (HTML). http://www.w3.org/MarkUp/.

[IBM] IBM. *XML Extender Administration and Programming*.

[KCD+03] Howard Katz, Don Chamberlin, Denise Draper, Mary Fernandez, Michael Kay, Jonathan Robie, Michael Rys, Jerome Simeon, Jim Tivy, and Philip Wadler. *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison-Wesley, 1st edition, 2003.

[med] Copyright Information & Downloading National Library of Medicine Data. http://www.nlm.nih.gov/databases/download.html.

[Mei02] Wolfgang Meier. eXist: An Open Source Native XML Database. http://exist-db.org/webdb.pdf, 2002.

[nlm] U.S. National Library of Medicine. http://www.nlm.nih.gov/.

[Ora03] Oracle. *Oracle XML DB Developer's Guide 10g Release 1 (10.1)*, 2003.

[pub] PubMed. http://www.ncbi.nlm.nih.gov/pubmed.

[sax] SAX project. http://www.saxproject.org/.

[sch] W3C XML Schema. http://www.w3.org/XML/Schema.

[sgm] Overview of SGML Resources. http://www.w3.org/MarkUp/SGML/.

[sql] SQL/XML. http://www.sqlx.org/.

[tam] Tamino. http://www2.softwareag.com/Corporate/products/tamino/default.asp.

[xht] XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). http://www.w3.org/TR/xhtml1/.

[xmla] Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/2004/REC-xml-20040204.

[xmlb] XML:DB API. http://www.xmldb.org/xapi/.

[xmlc] XML:DB Initiative. http://www.xmldb.org.

[xpa] XPath. http://www.w3.org/TR/xpath.

[xqu] XQuery. http://www.w3.org/XML/Query.

[xsla] Extensible Stylesheet Language (XSL) Version 1.0. http://www.w3.org/TR/xsl/.

[xslb] XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt.

[xup] XUpdate. http://www.xmldb.org/xupdate.