XML-pohjaiset kyselyt

Miikka Antikainen

21.05.2008

Joensuun yliopisto
Tietojenkäsittelytieteen laitos
Pro gradu -tutkielma

TIIVISTELMÄ

Rakenteisen tiedon käyttö kasvaa yhä edelleen nykyajan tietojärjestelmissä, joissa tiedon hakemisen, tallettamisen ja siirtämisen rooli on kasvanut yhä entisestään. XML ja siihen liittyvät tekniikat ovat olleet jo pitkään avustamassa nykypäivän tiedonvaihtamiseen liittyvissä arkirutiineissa niin tietojärjestelmien kehittäjiä kuin loppukäyttäjiäkin.

Kun informaation määrä kasvaa koko ajan, niin myös tiedon talletukseen käytettävälle järjestelmälle asetetaan uusia ja entistä haastavampia vaatimuksia koko ajan. Tieto täytyy olla helposti ja nopeasti saatavilla. Näiden vaatimusten täyttämiseen ovat heränneet myös tietokannanhallintajärjestelmiä kehittävät yhtiöt. Erilaisia rajapintoja ja tukea XML-kieleen onkin jo saatavilla usean eri valmistajan tietokantahallintajärjestelmästä. Yhtenä viimeisimmistä voidaan mainita tuki XML-pohjaisille kyselykielille.

Tässä tutkielmassa tutustutaan XML:n eri tekniikoihin ja niihin liittyviin kyselykieliin. Eri tekniikoita testataan myös käytännössä Oraclen XML-tietokannan kautta.

ACM-luokat (ACM Computing Classification System, 1998 version): D.3.2, H.2.3, H.2.4

Avainsanat: XML, tietokannanhallintajärjestelmät, kyselykielet

SISÄLLYSLUETTELO

1 JOHDANTO

Kautta aikojen eri organisaatiot ja yritykset ovat vaihtaneet tietoa keskenään. Perinteiset asiakirjat ovat yksi hyvä esimerkki siitä, miten tietynlainen rakenteisuus oli havaittavissa jo ennen Internetin valtakautta. Nykypäivän liiketoiminnassa ja tuotantoprosesseissa on yhä suurempi tarve erilaisen rakenteisen tiedon vaihtamiselle, hakemiselle ja tallettamiselle. Joillekin yrityksille tästä on tullut jopa ydinliiketoimintaa. Kun aiemmin dokumentit kirjoitettiin puhtaasti paperille ja niitä läheteltiin postitse vastaanottajalle, niin nykyään lähes kaikki informaatio on saatavissa sähköisessä muodossa, ja se on helposti ja nopeasti lähetettävissä myös eteenpäin. Sähköposti tavoittaa muutamassa minuutissa vastaanottajan mistä tahansa maailman kolkasta, eikä tiedon vastaanottaminen ja lähettäminen ole paikkaan tai aikaan sidottuja [19].

Pelkkä rakenteisen tiedon lähettämisen ja vastaanottamisen kasvaminen johti siihen, että tarvittiin yhä selkeämpiä yhteisiä toimintamalleja. Internetin yleistyessä tähän oman panoksensa antoi World Wide Web Consortium (W3C) -organisaatio, joka kehitti suosituksia ja standardeja kaikkien yhteiseksi hyväksi. Erilaiset Internetiin liittyvät tekniikat ja standardit, kuten HTML, XML, HTTP jne., saavuttivat heti suuren suosion niiden käyttäjien keskuudessa. Rakenteisten dokumenttien osalta XML oli varmasti suurin ilon aihe, sillä SGML-kieli oli havaittu syntaksiltaan liian haasteelliseksi normaalia päivittäistä käyttöä ajatellen.

Informaation määrän kasvaessa myös erilaisten haku- ja talletustapahtumien rooli on kasvanut entisestään. Tarvitaan yhä enemmän talletuskapasiteettia, nopeampia hakutoimintoja ja myös tiedonvälittämisen eteenpäin tulisi olla helppoa ja nopeaa. Tämän päivän tietokantavalmistajat painiskelevatkin näiden kysymysten kanssa päivittäin tyydyttääkseen asiakkaiden koko ajan kasvavaa vaatimustarvetta. Teratavun kokoiset tietokannat ovat jo normaalia arkipäivää ja talletettu tieto on yhä monimuotoisempaa. Myös tukea rakenteisille dokumenteille ja sitä kautta XML-muotoiselle tiedolle on kehitetty ja parannettu koko ajan. Suurimpien valmistajien tietokannanhallintajärjestelmistä löytyy jo oma tietotyyppinsä XML-pohjaiselle datalle. Myös erilaiset XML-pohjaisen tiedon hakemiseen ja tallettamiseen liittyvät

operaatiot ja tuki niihin liittyville rajapinnoille löytyy useimmista suurimpien valmistajien tietokantatuotteista.

Tämän tutkielman tavoitteena on tutustua rakenteisista kielistä tutun XML:n eri tekniikoihin ja siihen liittyviin ohjelmistorajapintoihin ja eri XML-hakutoimintoihin. XML-pohjaisen tiedon hakemiseen apuna käytetään Oraclen 10g XML-pohjaista tietokantaa ja niihin liittyviä eri rajapintoja. Osa esimerkeistä käydään läpi Java-ohjelmointikieltä apuna käyttäen.

Tutkielma etenee siten, että luvussa 2 käsittelen XML-kielen peruskäsitteitä ja niihin liittyviä tekniikoita, kuten DTD ja XML-skeema. Luvussa 3 tutustaan Oraclen XML-tietokannan tietotyyppiin XMLType ja XSU-tekniikkaan. Luvussa 4 keskityn eri XML-kyselytekniikoihin, kuten XPath ja XQuery. XML-dokumentti myös jäsennetään Java-ohjelmointikieltä apuna käyttäen ja lisäksi otan tuntumaa XSQL-nimiseen tekniikkaan, joka mahdollistaa XML-tiedonvaihdon dynaamisten Web-sivujen kautta. Luvussa 5 suoritan tutkielman yhteenvedon.

2 XML-PERUSKÄSITTEET

Tässä luvussa käsittelen XML-peruskäsitteet. Kohdassa 2.1 käyn läpi XML:n ja sen tekniikan ja kohdassa 2.2 tutustun DTD:n määrityksiin ja sen tekniikkaan. Viimeisenä kohtana tässä luvussa esitellään XML-skeema.

2.1 XML

XML (Extensible Markup Language) on SGML-kielestä periytyvä metakieli, jonka *W3C* (World Wide Web Consortium) hyväksyi standardiksi helmikuussa 1998. *SGML* (Standard Generalized Markup Language) –kieli kehitettiin jo vuonna 1986, jolloin tarkoituksena oli kehittää merkintäkieli rakenteisille dokumenteille. Muun muassa HTML-kieli on kehittynyt tämän seurauksena [3].

XML-kieli on SGML-kielen yksinkertaistettu versio ja se on myös tehty WWW-käyttöön sopivaksi. Se muistuttaa hiukan HTML-kieltä merkkauksineen, mutta sitä ei ole kuitenkaan tarkoitettu web-sivujen sivunkuvauskieleksi, vaan määrittelemään tiedon sisältö- ja rakennemuotoja. XML-kieli on siis SGML-kielen osajoukko.

XML-dokumentin rakenne voidaan jakaa *loogiseen* ja *fyysiseen rakenteeseen*. Fyysinen rakenne koostuu *entiteeteistä*, joita voidaan kuvailla rakenteisen dokumentin tiedon säilytyspaikoiksi. Dokumentin looginen rakenne jäsentää dokumentin elementtien mukaan eli jakaa sisällön sisäkkäisten elementtien mukaisiin osiin. Looginen rakenne myös määrittelee elementtien hierarkian ja niiden keskinäisen järjestyksen. Looginen rakenne edellyttää, että XML-dokumentti on ns. hyvin muodostettu. XML-dokumentti on silloin hyvin muodostettu, kun se pitää sisällään täsmälleen yhden *juurielementin*, jonka sisällä muut elementit ovat tasapainossa [22].

XML-dokumentti alkaa usein *esittelyosalla*, joka kertoo käytettävän XML-version ja käytettävän merkistön: `<?xml version="1.0" encoding="ISO-8859-1"?>`.

Esimerkki XML-tiedostosta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
 <book category="WEB">
   <title lang="en">Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
 </book>
</bookstore>
```

XML-dokumentin esittelyosassa voidaan määritellä myös standalone-määre. Määre standalone="yes" tarkoittaa, ettei XML-dokumentilla ole ulkoista dokumentin rakennemääritystä DTD (Document Type Definition). Mikäli ulkoinen rakennemääritys on olemassa, se esitellään XML-dokumentissa esimerkiksi seuraavasti:

```
<!DOCTYPE bookstore SYSTEM "bookstore.dtd">
```

2.1.1 Elementit

Elementit muodostavat XML-dokumentin hierarkkisen rakenteen. XML-dokumentti koostuu niin sanotuista juurielementistä, jolla voi olla muita *alielementtejä*, eli nk. lapsielementtejä. Elementillä on oltava aina *alku*- ja *lopputunniste*, joita myös nimitetään tageiksi. Elementti alkaa aloitustagilla esim. `<year>` ja loppuu lopetustagiin `</year>` [9]. Elementin sisältämä arvo on sijoitettu alku- ja lopputunnisteen väliin seuraavasti:

```
<year>2003</year>
```

Elementti voi olla myös tyhjä, jolloin se merkataan tyhjän elementin tunnisteella ja elementin nimellä esim. `<TYHJÄELEMENTTI/>`.

2.1.2 Attribuutit

Attribuutin tehtävänä on antaa lisätietoa elementistä [5]. Attribuutit ovat nimi-arvo –pareja, jotka sijoitetaan elementin alkutunnisteen sisään, esim. `<book category="WEB">`.

2.1.3 Entiteetit

XML-dokumentin fyysinen rakenne koostuu siis entiteeteistä. Entiteetit voivat sisältää esimerkiksi dokumentin osia, kuvia, linkkejä toisiin dokumentteihin, tekstiä, erikoismerkkejä, matemaattisia symboleita ja niin edelleen.

2.1.4 Kommentit

XML-kielen *kommentit* sijoitetaan aina <!-- ja --> tagien väliin. Kommentit voidaan sijoittaa mihin tahansa kohtaan XML-dokumenttia.

2.2 DTD

DTD (Document Type Definition) eli dokumentin rakennemääritys on eräs XML-kielen yhteydessä käytetyistä rakennemäärittelytavoista. DTD:n avulla XML-dokumentille voidaan määritellä elementtien sallitut ilmenemismuodot ja niiden esittämisjärjestys [9]. DTD:n avulla määritellään myös mahdolliset elementtien attribuutit ja sisältö.

DTD voi olla joko dokumentin sisäinen tai ulkoinen rakennemääritys. Ulkoinen DTD sijaitsee omassa tiedostossaan XML-dokumentin ulkopuolella.

Esimerkki dokumentin rakennemäärittelystä:

```
<!ELEMENT prices (book*)>
<!ELEMENT book (title, source, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

2.2.1 Elementtien määritykset

DTD:ssä esiintyvät elementtien määritykset kuvaavat säännöt XML-dokumentissa esiintyvälle elementille. Ylläolevassa esimerkissä elementtimäärääritys `<!ELEMENT prices`

`(book*)>` kuvaa, että elementti pricess sisältää nolla tai useampia book-elementtejä. Elementit title, source ja price sisältävät #PCDATA-tietoa (Parsed Character Data), eli ns. *jäsennettävää merkkitietoa* [8].

Elementtien esiintymiskertoja kuvaavat määritykset ovat:

? elementti voi esiintyä 0 tai 1 kertaa

+ elementti voi esiintyä 1 tai monta kertaa

* elementti voi esiintyä 0 tai monta kertaa

Mikäli merkintä puuttuu kokonaan, se tarkoittaa että elementti esiintyy yhden kerran.

2.2.2 Attribuuttien määritykset

Attribuuttien määritykset identifioivat, millä elementeillä voi olla attribuutteja, mitä attribuutteja niillä voi olla, mitä arvoja ko. attribuutit voivat saada ja mitkä niiden oletusarvot ovat. Jokaisella attribuuttimäärityksellä on kolme osaa: nimi, tyyppi ja mahdollinen oletusarvo. Alla on kuvattu attribuuttien tyypit:

CDATA	merkkijonotyyppi, jonka XML-prosessori jättää jäsentämättä.
ID	yksilöllinen elementti, kaikkien dokumentin ID-arvojen on oltava erilaisia
IDREF	yksittäisen ID-attribuutin arvo jollekin dokumentin elementeistä
ENTITY	yksittäisen entiteetin nimi
NMTOKEN	merkkijonon rajattu muoto

Esimerkiksi alla esitetyssä attribuuttimäärityksessä attribuutti year määritellään elementille book. Attribuutti year on tyyppiä CDATA ja se on pakollinen. Määreellä #IMPLIED ilmaistaan siis attribuutin valinnaisuutta ja määreellä #REQUIRED attribuutin pakollisuutta.

Määre #FIXED ilmaisee että attribuutilla on aina vakioarvo. Seuraava esimerkki ilmaisee book-elementin pakollisen year-attribuutin:

```
<!ATTLIST book  year CDATA  #REQUIRED >
```

2.2.3 Entiteettien määritykset ja notaatiot

Mikäli XML-dokumentissa halutaan julkaista esimerkiksi kuvia, tulee ne määritellä erikseen `<!ENTITY>` –osiossa. DTD:ssä käytetään `NOTATION` –määrettä ilmaisemaan XML-dokumenttiin tuotavan ulkoisen tiedon tiedostomuoto:

```
<!NOTATION jpg SYSTEM ”image/jpg”>
```

Määre `NDATA` ilmaisee, että entiteetti sisältää jäsentämätöntä dataa:

```
<!ENTITY kuva SYSTEM ”kukka.jpg” NDATA jpg>
```

2.2.4 Validointi DTD:n avulla

XML-standardi määrittelee, että XML-dokumentin tulee olla hyvin muodostettu ja validi. Hyvin muodostetun dokumentin täytyy siis alkaa XML-määrittelyllä, esim. <?xml version ”1.0”?>, ja jokaisella ei-tyhjällä elementillä tulee olla alku- ja lopputunniste. XML-dokumentin validointi onnistuu validoivan jäsentimen ja DTD:n avulla. Validoiva jäsennin vertaa siis XML-dokumenttia käytettyyn DTD-tiedostoon ja antaa virheilmoituksen, mikäli ko. dokumentti ei noudata DTD:tä. Jäsennin ilman validointi-ominaisuutta eli ns. ei-validoiva jäsennin, suorittaa ainoastaan tarkastuksen, onko dokumentti hyvin muodostettu [10]. XML-dokumentin validointi voidaan hoitaa myös XML-skeeman avulla.

2.3 XML-SKEEMA

XML-skeema on W3C:n (World Wide Web Consortium) kehittämä standardi XML-dokumentin rakenteen, sisällön ja semanttisuuden määrittämiseen [11]. XML-skeema on siis vaihtoehtoinen dokumentin rakennemäärittelytapa DTD:lle. Se eroaa DTD:stä siten, että se tarjoaa paljon tarkemman rakennemäärityksen XML-dokumentille [18]. XML-skeema on myös laajennettavissa tulevia lisäyksiä varten ja se tukee erilaisia nimiavaruuksia ja tietotyyppejä, kuten esimerkiksi `integer`, `string` ja `date`. XML-skeeman osia voidaan käyttää myös uudelleen toisissa skeemoissa [17].

XML-skeema on myös XML-dokumentti. Sen tunnistaa tiedostopäätteestä .xsd (xml schema description) ja sen juurielementti on <schema>. Jokaisella elementillä on tyypillisesti etuliite `xsd:`, mikä viittaa W3C:n nimiavaruuteen määrittelyllä xmlns:xsd= ”http://www-w3.org/2001/XMLSchema”. Täten sitä ei voida sekoittaa normaaliin XML-dokumenttiin. XML-skeemassa elementit määritellään käyttämällä xsd:element-määrettä:

```
<xsd:element name=”pvm”>
```

Elementin käyttämä tyyppi voidaan määritellä type määreen kautta:

```
<xsd:element name=”pvm” type=”xsd:date”>
```

Elementin esiintymiskerrat voidaan määritellä minOccurs ja maxOccurs määreillä:

```
<xsd:element name=”pvm” type=”xsd:date” minOccurs=”0”
maxOccurs=”1”>
```

Attribuutit määritellään XML-skeemassa määreellä xsd:attribute:

```
<xsd:attribute name=”paid” type=”boolean”/>
```

XML-skeema tukee myös erilaisia komplekseja tyyppejä ja attribuuttien ja mallien ryhmittelyä. Alla olevassa esimerkissä luodaan Address -niminen tyyppi:

```
<xsd:complexType name=”Address”>
       <xsd:sequence>
              <xsd:element name=”Street1”/>
              <xsd:element name=”Street2”/>
              <xsd:element name=”City”/>
              <xsd:element name=”State”/>
              <xsd:element name=”Zip”/>
```

```
        </xsd:sequence>
</xsd:complexType>
```

Tämän jälkeen Address –tietotyyppiä voidaan käyttää määrittelemään elementtejä:

```
      <xsd:element name=”mailingAddress” type=”Address”/>
      <xsd:element name=”billingAddress” type=”Address”/>
```

Oracle 10g -tietokanta tukee XML-skeemaa. Oracle käyttää XML-skeemaa mm. XML-tiedostojen validointiin. Kerran rekisteröity skeema mahdollistaa myös XMLType-tyyppisten taulujen, sarakkeiden ja näkymien luonnin ko. skeemaan perustuen [11].

Luodaan skeema Oraclen tietokantaan [24]:

```
SQL> declare doc varchar2(1000) := '<schema
targetNamespace="http://www.oracle.com/PO.xsd"
  2  xmlns:po="http://www.oracle.com/PO.xsd"
  3  xmlns="http://www.w3.org/2001/XMLSchema">
  4  <complexType name="PurchaseOrderType">
  5  <sequence>
  6  <element name="PONum" type="decimal"/>
  7  <element name="Company">
  8  <simpleType>
  9  <restriction base="string">
 10  <maxLength value="100"/>
 11  </restriction>
 12  </simpleType>
 13  </element>
 14  <element name="Item" maxOccurs="1000">
 15  <complexType>
 16  <sequence>
 17  <element name="Part">
 18  <simpleType>
 19  <restriction base="string">
 20  <maxLength value="1000"/>
 21  </restriction>
 22  </simpleType>
 23  </element>
 24  <element name="Price" type="float"/>
 25  </sequence>
 26  </complexType>
 27  </element>
 28  </sequence>
 29  </complexType>
 30  <element name="PurchaseOrder" type="po:PurchaseOrderType"/>
 31  </schema>';
 32  begin
 33  dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
 34  end;
 35  /

PL/SQL procedure successfully completed.
```

3 XML-TIETOKANTA

Tässä luvussa tutustutaan Oraclen XML-tietokantaan ja sen mahdollistamiin tekniikoihin. Kohdassa 3.1 käyn läpi Oraclen XMLType-tietotyypin ja kohdassa 3.2 esittelen Java-pohjaisen XSU:n (Oracle XML SQL Utility), joka on Oraclen XDK -komponentti.

3.1 XMLType

Oraclen tietotyyppi *XMLType* esiteltiin ensi kertaa 9i Release-version (9.0.1) yhteydessä. Tämä oli suuri helpotus sovelluskehittäjien työhön, sillä aiemmin he olivat joutuneet tallettamaan XML-dokumentin esimerkiksi CLOB-tyyppiseen sarakkeeseen ja jäsentämään sisällön taas uudelleen dokumentin käyttöä varten. XMLType on Oraclen natiivi tietotyyppi, aivan kuten esimerkiksi DATE tietotyyppi. XMLType:ä voidaan siis käyttää määrittämään yksittäistä saraketta, taulua tai esimerkiksi osana PL/SQL-lausekkeen muuttujaa [1].

XMLType tukee W3C:n määrittelemää XML-skeemaa, joten validointi skeeman avulla voidaan liittää esim. XMLType-tyyppiseen tauluun tai sarakkeeseen. XMLType pitää sisällään myös monia hyödyllisiä funktiota, joiden avulla XML-pohjaista tietoa voidaan esimerkiksi hakea, tallettaa tai indeksoida. Tietotyyppiä on mahdollista käyttää myös erilaisten ohjelmistorajapintojen (API) kautta kuten esim. PL/SQL- tai Java-ohjelmointikielen kautta.

XMLType-tietotyypin avulla kehittäjät saavat siis käyttöönsä relaatiotietokannan hyödyt ja XML-kielen ilmaisuvoiman.

Luodaan taulu po_ta, jossa on kenttä id, joka on tietotyypiltään number, ja kenttä po, joka on tietotyypiltään XMLType [24]:

```
SQL> create table po_tab
  2    (
  3    id number,
  4    po sys.XMLType
  5    )
  6    xmltype column po
  7      XMLSCHEMA "http://www.oracle.com/PO.xsd"
  8      element "PurchaseOrder";

Table created.
```

Taulun rakenteen sisältöä voidaan tarkastella DESC-komennolla:

```
SQL> DESC po_tab;
 Name                           Null?    Type
 ------------------------------ -------- ----
 ID                                      NUMBER
 PO                                      SYS.XMLTYPE
```

Viedään tauluun dataa INSERT-komennolla [24]:

```
SQL> insert into po_tab values (5,
  2   xmltype('<po:PurchaseOrder xmlns:po="http://www.oracle.com/PO.xsd"
  3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  4    xsi:schemaLocation="http://www.oracle.com/PO.xsd
  5   http://www.oracle.com/PO.xsd">
  6    <PONum>1001</PONum>
  7     <Company>Oracle Corp</Company>
  8    <Item>
  9     <Part>9i Doc Set</Part>
 10      <Price>2550</Price>
 11    </Item>
 12    <Item>
 13     <Part>8i Doc Set</Part>
 14      <Price>350</Price>
 15    </Item>
 16   </po:PurchaseOrder>'));

1 row created.
```

3.1.1 XMLType funktioiden käyttöä

Seuraavissa esimerkeissä käyn läpi Oraclen XMLType-tietotyypin sisältämien funktioiden getClobVal(), existsNode() ja extract() käyttöä.

Funktio getClobVal() palauttaa tulosjoukon nimensä mukaisesti CLOB-muodossa:

```
SQL> SELECT x.puhelin.getCLOBVal() FROM tyontekija x;

X.PUHELIN.GETCLOBVAL()
--------------------------------------------------------------------
<puhelin>013-235678</puhelin>
<puhelin>050-2662287</puhelin>
<puhelin>044-1234566</puhelin>
<puhelin><koti>013-345678</koti><tyo>013-88888</tyo></puhelin>
```

Funktio existsNode() palauttaa arvon yksi, mikäli kyselyssä annettu Xpath-lausekkeen solmu on olemassa. Mikäli solmua ei löydy, palauttaa funktio arvon nolla. Seuraava kysely löytää halutun solmun:

```
SQL> SELECT existsnode(value(x),'/bib/book/author/last') FROM testixml x;

EXISTSNODE(VALUE(X),'/BIB/BOOK/AUTHOR/LAST')
--------------------------------------------
                                           1
```

Mikäli kyselyn tulee palauttaa enemmän kuin yhden solmun arvo, tulee käyttää extract()-funktiota. Funktio extractValue() pystyy palauttamaan yksittäisen solmun tai attribuutin arvon XPath-lausekkeeseen perustuvassa kyselyssä. Seuraava kysely palauttaa last-elementtien arvot:

```
SQL> SELECT extract(value(x), '/bib/book/author/last') FROM testixml x;

EXTRACT(VALUE(X),'/BIB/BOOK/AUTHOR/LAST')
-----------------------------------------
<last>Stevens</last>
<last>Stevens</last>
<last>Abiteboul</last>
<last>Buneman</last>
<last>Suciu</last>
```

3.1.2 XMLType -näkymät

XMLType-näkymät ovat osa Oraclen XML-tietokantaa. XMLType-näkymät mahdollistavat perinteisen relaatiotiedon esittämisen XML-muotoisena näkymien kautta. Tästä on hyötyä etenkin niissä tapauksissa, joissa tarvitaan XML-muotoista esitystapaa, mutta tieto on talletettu tietokantaan perinteisessä muodossa. XMLType-näkymät mahdollistavat siis migraation tietokannan ja sovellusohjelman välillä siten, ettei alkuperäistä sovellusohjelman ohjelmakoodia tarvitse muuttaa [1].

3.2 XSU

Kun ajatellaan XML-tekniikkaa, niin se usein yhdistetään jollain tavoin Internetiin. Ajatusmalli on sinänsä luonnollinen sillä XML-dokumentin sisältö on puhtaasti tekstiä, jota on helppo siirtää läpi eri Internet-protokollien, eri käyttöjärjestelmien, palomuurien ja niin edelleen. Useimpien järjestelmien edut perustuvat kuitenkin yleensä niiden sisältämään dataan, joka useimmiten on talletettu johonkin tietokantaan. XML-tyyppinen tietokanta yhdistää nämä molemmat vaatimukset [12].

XML SQL Utility (XSU) on Oraclen XDK-komponentti, joka mahdollistaa SQL-kyselyn tulosjoukon muuntamisen XML:ksi ja päinvastoin. XSU on saatavilla Java- ja PL/SQL-versiona ja sitä voidaan käyttää Java- tai PL/SQL-ohjelmointikieltä käyttäen sekä Java-pohjaisen komentotulkin kautta [7]. XSU mahdollistaa mm. DTD:n dynaamisen generoinnin, XML-dokumentin muodostamisen joko String-tyyppisenä tai DOM-puumuotoisena, XML-skeemojen generoinnin SQL-kyselyistä [11] [14]. XSU tukee myös kaikkia Oraclen tukemia tietotyyppejä.

XSU mahdollistaa siis kyselyjen tekemisen relaatiotietokannan tauluihin ja näkymiin, jonka jälkeen tulosjoukko muunnetaan XML-muotoiseksi dokumentiksi. Myös datan tallentaminen XML-dokumentista tietokannan vastaavaan tauluun tai näkymään onnistuu XSU:n

avulla. XSU mahdollistaa myös tietokannan taulun päivittämisen ja tiedon poistamisen [7] [13][14].

Esimerkki XSU:n generoimasta XML-dokumentista:

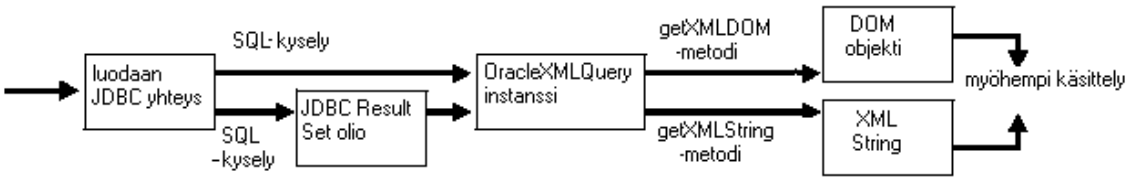
```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

Koko kyselyn tulosjoukko sisällytetään <ROWSET>…</ROWSET> -elementtien sisään. Kyseinen rivijoukko voi pitää sisällään tietokantataulun yhden tai useamman rivin tiedot. <ROWSET> -elementti on myös generoidun XML-dokumentin juurielementti [6] [7].

Jokainen <ROW> -elementti pitää sisällään yhden tietokantaulun rivin tiedot. Jokainen <ROW> -elementti sisältää siis yhden tai useampia elementtejä, jotka vastaavat ko. tietokantarivin sarakkeita. Elementit on nimetty taulun sarakkeiden mukaisesti, kuten esim. `<JOB>CLERK</JOB>`. Jokaiselle riville generoituu automaattisesti oma juokseva numeronsa attribuutti NUM, esim. `<ROW num="1">`.

Jokainen taulun sarakkeen mukaisesti nimetty elementti pitää sisällään kyselyn tuloksena syntynyttä dataa esimerkiksi `<SAL>800</SAL>`.

Kun tietokannan tauluun tai näkymään viedään dataa XML-dokumentista, XSU hankkii ensin metatietoa talletettavasta kohteesta. Tähän metatietoon perustuen XSU muodostaa talletuksessa käytettävän INSERT-lauseen. Tämän jälkeen XSU generoi tiedon XML-dokumentista ja yhdistää sen vastaaviin sarakkeisiin tai attribuutteihin. Lopuksi XSU suorittaa ko. INSERT-lauseen. Kuva 1 havainnollistaa kyselyn vaiheet.



Kuva 1. XSU-kyselyn vaiheet [2]

Seuraavassa esimerkissä tietokannan JASEN-taulusta generoidaan XML-dokumentti XSU:n avulla käyttäen Java-ohjelmointikieltä:

1. Luodaan yhteys

Jotta XML-dokumentin generointi onnistuisi, meidän täytyy ensiksi luoda yhteys tietokantaan. Yhteys voidaan luoda käyttämällä JDBC-rajapintaa. Rekisteröidään ensiksi JDBC-ajuri ja tämän jälkeen luodaan yhteys:

```
// tuodaan Oracle JDBC ajuri
import oracle.jdbc.driver.*;

// rekisteröidään ja ladataan Oracle JDBC ajuri
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

//Luodaan yhteys käyttäen JDBC thin ajuria
DriverManager.getConnection("jdbc:oracle:thin:@cs.joensuu.fi:1521:xml","u
sername","password");
```

Yhteyden muodostamiseen käytetään JDBC thin -ajuria, joka on kirjoitettu Java-ohjelmointikielellä. Määritellään thin-ajurille isäntäkoneen nimi (cs.joensuu.fi), portti (1521) ja Oraclen SID (xml), joka määrittelee käytettävän tietokantainstanssin.Yhteyden muodostamiseen voidaan käyttää myös Oracle:n OCI8 –rajapintaa (The Oracle Call Interface).

2. Luodaan OracleXMLQuery-luokkaa vastaava olio

Kun yhteys on luotu, luodaan SQL-kysely käyttäen apuna OracleXMLQuery-luokkaa.

```
//tuodaan kyselyluokka import -komennolla
import oracle.xml.sql.query.OracleXMLQuery;

// Luodaan kyselyluokka
OracleXMLQuery qry = new OracleXMLQuery(conn, "SELECT * FROM JASEN");
```

3. Muodostetaan kyselyn tulos

Kyselyn tulos voidaan muodostaa joko DOM puurakenteena tai XML string-muotoisena. DOM-olion tulostus voidaan muodostaa seuraavasti:

```
org.w3c.DOM.Document domDoc = qry.getXMLDOM();
```

XML string -tyyppinen tulostus voidaan muodostaa seuravasti:

```
String xmlString = qry.getXMLString();
```

Alla on esitetty ohjelmakoodi kokonaisuudessaan:

```
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;
// luokka stringin generoinnin testaukseen XSU:n avulla
class testXMLSQL {

   public static void main(String[] argv)
   {
     try{
      // luodaan yhteys
      Connection conn  = getConnection("scott","tiger");
      // Luodaan kyselyluokka
      OracleXMLQuery qry = new OracleXMLQuery(conn, "SELECT * FROM JASEN");
      // Muodostetaan tulosjoukko käyttäen getXMLString funktiota
      String str = qry.getXMLString();
      // Tulostetaan kysely XML muodossa
      System.out.println(" The XML output is:\n"+str);
      // Suljetaan kysely
     qry.close();
     }catch(SQLException e){
      System.out.println(e.toString());
     }
   }
```

```
   // Otetaan yhteys käyttäjätunnuksen ja salasanan avulla
   private static Connection getConnection(String username, String password)
     throws SQLException
   {
      // rekisteröidään JDBC ajuri
       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      // Luodaan yhteys käytten JDBC thin ajuria
       Connection conn = DriverManager.getConnection("jdbc:oracle:thin:
            @cs.joensuu.fi:1521:xml","username","password");
      return conn;
   }
}
```

Ohjelman suorituksen tuloksena ohjelma tulostaa tietokannan JASEN-taulun XML-muodossa:

```
> java testXMLSQL
 The XML output is:
<?xml version = '1.0'?>
<ROWSET>
   <ROW num="1">
      <ID>1234-111</ID>
      <ETUNIMI>MATTI</ETUNIMI>
      <SUKUNIMI>MEIKALAINEN</SUKUNIMI>
   </ROW>
   <ROW num="2">
      <ID>2222-222</ID>
      <ETUNIMI>TEPPO</ETUNIMI>
      <SUKUNIMI>TEIKALAINEN</SUKUNIMI>
   </ROW>
   <ROW num="3">
      <ID>3333-333</ID>
      <ETUNIMI>MAIJA</ETUNIMI>
      <SUKUNIMI>MEIKALAINEN</SUKUNIMI>
   </ROW>
   <ROW num="4">
      <ID>4444-444</ID>
      <ETUNIMI>SEPPO</ETUNIMI>
      <SUKUNIMI>SUOMALAINEN</SUKUNIMI>
   </ROW>
   <ROW num="5">
      <ID>5555-555</ID>
      <ETUNIMI>REIJO</ETUNIMI>
      <SUKUNIMI>RUOTSALAINEN</SUKUNIMI>
   </ROW>
</ROWSET>
```

4 XML-KYSELYT

Tässä luvussa käyn läpi erilaiset XML-kyselymenetelmät. Kohdassa 4.1 perehdyn jäsentämiseen tekniikkana ja suoritan jäsennyksen Java-ohjelmalla SAX-rajapintaa käyttäen. Kohdassa 4.2 käyn läpi XPath –kielen ja kohdassa 4.3 tutustun XML-kyselyihin Web-sivujen kautta XSQL-tekniikan avulla. Luvun päättää kohta 4.4 jossa käsittelen XQuery –kyselykieltä.

4.1 Jäsentäminen

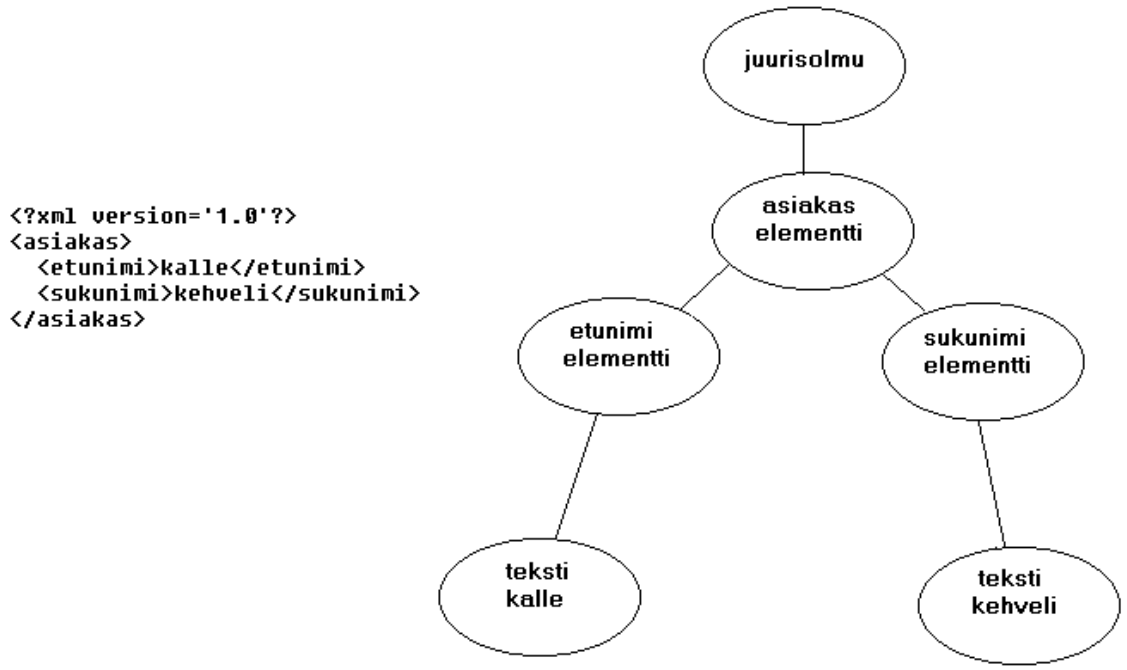
Oraclen XML-*jäsennin* lukee XML-dokumentin ja käyttää joko *DOM*- (Document Object Model) tai *SAX* (Simple Api for XML) -rajapintaa tarjoten ohjelmallisen pääsyn dokumentin sisältöön ja sen rakenteeseen [2]. Jäsennin on siis ohjelma, joka lukee XML-dokumentin rakenteen taso kerrallaan ja tarkastaa, että dokumentti noudattaa XML-kielioppia ja rakennemäärittelyä. Tätä tapahtumasarjaa kutsutaan siis jäsennykseksi. Normaalisti jäsennin tarkastaa, onko XML-dokumentti syntaksiltaan oikein, eli ns. hyvin muodostettu, mutta on myös olemassa ns. validoivia jäsentimiä. Validoiva jäsennin vertaa dokumenttia siihen liitettyyn DTD:hen tai skeemaan ja päättelee sen mukaan, onko ko. XML-dokumentti validi, eli oikein muodostettu [15].

Jäsennin lukee XML-dokumentin tietokoneen muistiin käyttäen apuna valittua rajapintaa eli API:a. Rajapinnan avulla XML-dokumentin sisältö ja rakenne saadaan myös muiden sovellusohjelmien käyttöön. XML-jäsentimeen on saatavilla useita eri rajapintoja mutta tärkeimmät tällä hetkellä käytettävistä ovat DOM ja SAX, joista SAX on ns. tapahtumaperustainen rajapinta ja DOM perustuu hierarkkiseen puurakenteeseen.

4.1.1 DOM

DOM (Document Object Model) on nimensä mukaisesti oliomalli, joka käsittelee XML-dokumentin osia olioina, tarjoten metodit niiden muokkaamiseen. DOM on myös W3C:n julkaisema standardi [16], joka on jaettu eri tasoihin kuten DOM Level 1, DOM Level 2

jne. DOM-rajapinta lukee kerralla koko XML-dokumentin tietokoneen muistiin ja muodostaa sen objekteista hierarkkisen puurakenteen. DOM tarjoaa siis kerralla kuvauksen koko XML-dokumentista, jossa elementit, attribuutit ja entiteteetit esitetään solmuina [21]. Elementtisolmuissa liikkuminen tapahtuu DOM:in tarjoamien metodien avulla, jotka mahdollistavat puun monipuolisen läpikäynnin. Esimerkiksi liikkumisen lapsisolmusta takaisin isäntäsolmuun on mahdollista. DOM mahdollistaa myös XML-dokumentin muokkaamisen, kuten elementin lisäämisen, entiteetin muokkaamisen ja poistamisen. Kuva 2 selventää XML-dokumentin suhdetta DOM-malliin.

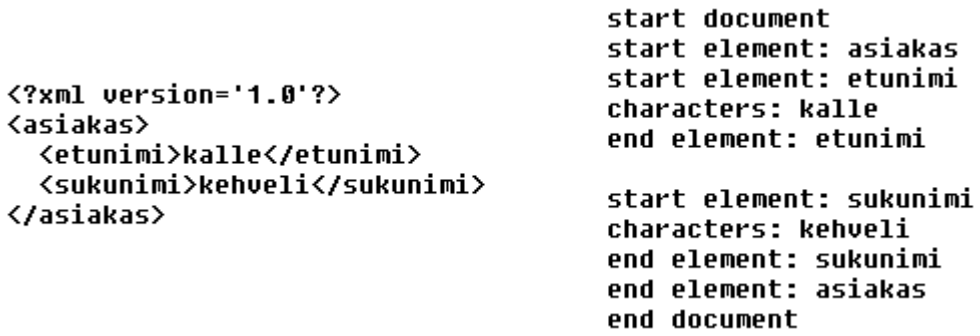


Kuva 2. XML-dokumentti ja sitä vastaava DOM-malli [18]

4.1.2 SAX

SAX on sarjallinen tapahtumapohjainen malli, jossa XML-dokumenttia käsitellään tapahtuma kerrallaan [20]. Käytännössä tapahtumaperusteisuus tarkoittaa siis sitä, että kun XML -dokumenttia luetaan ja määritelty elementti kohdataan, niin se käsitellään SAX:n avulla, jonka jälkeen se unohdetaan ja siirrytään seuraavaan elementtiin.

SAX on ns. ”Read-only” rajapinta, eli se mahdollistaa XML-dokumentin lukemisen, mutta ei sen muokkaamista. SAX ei salli myöskään sattumanvaraista hakua, eikä liikkumista dokumentissa taaksepäin. Mikäli dokumentissa halutaan liikkua takaisinpäin, joudutaan prosessi aloittamaan uudelleen alusta. Kuva 3 havainnollistaa XML-dokumentin suhdetta SAX-jäsennykseen.

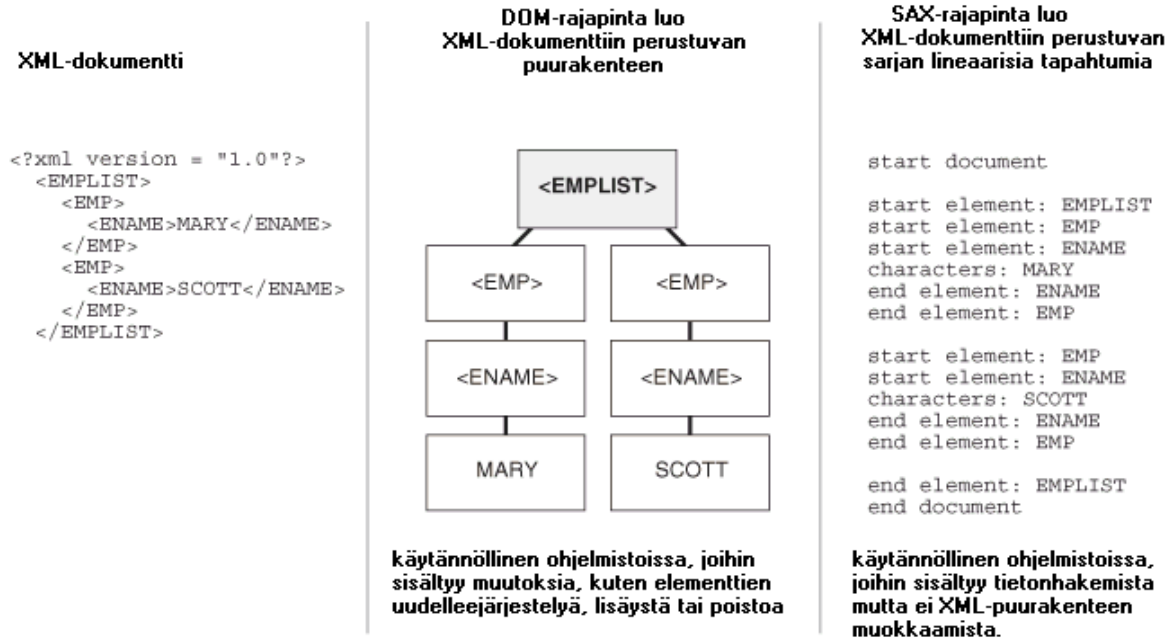


Kuva 3: XML-dokumentti ja sitä vastaava SAX-jäsennys [15]

4.1.3 DOM vs. SAX

Rajapinta kannattaa siis valita aina tilanteen ja käyttötarkoituksen mukaan. DOM sopii hyvin tilanteisiin, joissa tarkasteltavaa XML-dokumenttia saatetaan joutua muokkamaan ja joissa dokumentin koko ei ole liian suuri (dokumentin on mahduttava tietokoneen muistiin).

SAX-rajapinta soveltuu hyvin tilanteisiin, joissa joudutaan suorittamaan paljon haku-tyyppisiä toimintoja. SAX ei myöskään aseta rajoitteita dokumentin koolle. Kuvassa 4 on havainnollistettu ja vertailtu dokumenttia, joka koostuu kahdesta elementistä.



Kuva 4. DOM ja SAX –rajapintojen vertailu [2]

4.1.4 jäsentämisesimerkki SAX-rajapinnalla

Seuraavassa esimerkissä suoritan XML-dokumentin jäsentämisen SAX-rajapintaa ja Java-ohjelmointikieltä apuna käyttäen perustuen osaksi lähteeseen [24]. XML-dokumentti on talletettu Oraclen tietokannan tauluun `po_xml_tab`.

Java-luokka `Example` muodostaa tietokantayhteyden xml-tietokantaan JDBC thin -ajuria käyttäen. Luokka kutsuu luokkaa XMLParseri, jota apuna käyttäen XML-dokumentti jäsennetään ja tulostetaan näytölle. Alla on kuvattuna `Example`-luokan ohjelmakoodi:

```
import oracle.xdb.XMLType;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.xml.parser.v2.*;
import java.io.*;

public class Example
  {
  public static void main(String [] arg)
```

```
    {
    try {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
      Connection con = DriverManager.getConnection("jdbc:oracle:thin:
            @cs.joensuu.fi:"+"1521:xml","kayttajatunnus","salasana");
      OraclePreparedStatement stmt;
      stmt = (OraclePreparedStatement) con.prepareStatement("select e.poDoc
                  from po_xml_tab e");
      ResultSet rset = stmt.executeQuery();
      OracleResultSet orset = (OracleResultSet) rset;
      while (orset.next())
        {
        // get the XMLType
        XMLType poxml = (XMLType) orset.getObject(1);
        // get the XML as a string.
        String poString = poxml.getStringVal();
        System.out.println(poString);
        //==== jäsennys SAX-rajapinnalla.
        ByteArrayInputStream bis =
            new ByteArrayInputStream(poString.getBytes());
        InputStreamReader reader = new InputStreamReader(bis);
        XMLParseri p = new XMLParseri();
        p.parse(reader);
        }
      }
    catch (Exception e) {
     System.out.println(e);
      }
    System.out.println("______");
    }
  }
```

Luokka `XMLParseri` suorittaa luokalta `Example` saadulle syötteelle jäsennyksen ja tulostaa XML-dokumentin jäsennettynä näytölle. Alla on kuvattuna luokan `XMLParseri` ohjelmakoodi:

```
import org.xml.sax.*;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import org.w3c.dom.Node;
import java.io.*;
import java.net.*;

public class XMLParseri extends HandlerBase
  {

  public XMLParseri() {
    //Parser parser = new SAXParser();
    }

  public void parse(Reader reader) {
    Parser parser = new SAXParser();
```

```
    parser.setDocumentHandler(this);
    parser.setEntityResolver(this);
    parser.setDTDHandler(this);
    parser.setErrorHandler(this);
    try {
      parser.parse(new InputSource(reader));
      }
    catch(Exception e) {
      System.out.println(e);
      }
    }
  // Takaisinkutsumetodit
  public void startDocument() {
    System.out.println("Dokumentin luku alkaa.......");
    System.out.println();
    }
  public void startElement(String name, AttributeList atts)
    throws SAXException
    {
    System.out.println("Aloitus elementti:    " + name);
    for (int i = 0; i < atts.getLength(); i++) {
      String aname = atts.getName(i);
      String type  = atts.getType(i);
      String value = atts.getValue(i);
      System.out.println("Atribuutit: "+aname + "  " +value);
      }
    }

  public void characters(char[] cbuf, int start, int len)
    {
    System.out.print("Elementtien arvot:    ");
    System.out.println(new String(cbuf,start,len));
    System.out.println();
    }

  } //end
```

Kun käännetyt Java-luokat ajetaan, niin ohjelma tulostaa ensin näytölle haetun XML-dokumentin ja tämän jälkeen SAX-jäsennetyn version ko. dokumentista:

```
> java Example9_3
<?xml version="1.0"?>
    <PO pono="1">
      <PNAME>Po_1</PNAME>
      <CUSTNAME>John</CUSTNAME>
      <SHIPADDR>
        <STREET>1033, Main Street</STREET>
        <CITY>Sunnyvalue</CITY>
        <STATE>CA</STATE>
      </SHIPADDR>
    </PO>
```

```
Dokumentin luku alkaa.......

Aloitus elementti:    PO
Atribuutit:           pono  1
Aloitus elementti:    PNAME
Elementtien arvot:    Po_1

Aloitus elementti:    CUSTNAME
Elementtien arvot:    John

Aloitus elementti:    SHIPADDR
Aloitus elementti:    STREET
Elementtien arvot:    1033, Main Street

Aloitus elementti:    CITY
Elementtien arvot:    Sunnyvalue

Aloitus elementti:    STATE
Elementtien arvot:    CA
```

4.2 XPath

XML Path Language (XPath) on XML-dokumentin osien osoittamiseen kehitetty kieli, jonka W3C hyväksyi standardiksi marraskuussa 1999 [23]. XPath –kieltä voidaan siis käyttää tiedon etsimiseen XML-dokumentista esim. XSLT:n ja XPointer:in avulla. Dokumentin osien osoittaminen on mahdollista esim. viittaamalla elementtien ja attribuuttien nimiin tai niiden arvoihin. XPath käsittelee XML-dokumenttia puurakenteena, jossa on solmu jokaiselle elementille, attribuutille, tekstille, nimiavaruudelle, kommentille ja dokumentin juurielementille[9]. Solmun tai solmujoukon määrittämiseksi käytetään polkulauseketta, joka määrittää reitin kohteena olevaan solmuun. Polkulauseke koostuu yhdestä tai useammasta askeleesta, jotka erotetaan kauttaviivalla eli merkein ”/” tai ”//”. Yksittäinen / -merkki polku-lausekkeen alussa osoittaa puurakenteen juureen, jota voidaan kutsua myös absoluuttiseksi poluksi. Muutoin polkulauseke on suhteellinen ja se alkaa aktiivisesta solmusta [22].

Absoluuttinen polkulauseke on muotoa:

/askel/askel/…

Suhteellinen polkulauseke on muotoa:

askel/askel/…

Askelten avulla siis päästään XML-dokumentin solmusta yhteen tai useampaan toiseen solmuun. Jokainen askel muodostuu kolmesta komponentista: akselista, solmutestistä sekä mahdollisista predikaateista.

Polkulausekkeen syntaksi on muotoa:

akseli::solmutesti[predikaatit]

Esimerkiksi seuraavalla polkulausekkeella

```
child::class[position() <= 10] / descendant::student
```

valitaan kyseisestä XML-dokumentista kaikki student-elementit, jotka sisältyvät kymmeneen ensimmäiseen class-elementtiin.

Yleisimmät sijaintioperaattorit on esitetty taulukossa 1.

Taulukko 1. XPath sijaintioperaattoreita [23] [10]

Operaattori	**Kuvaus**
solmunimi	kaikki solmun lapsisolmut
/	juurielementti
//	kaikki jälkeläiset
.	valitsee tämänhetkisen solmun
..	valitsee solmun edeltäjän
@	valitsee attribuutit

Taulukossa 2 on kuvattu useimmin käytetyt Xpath -akselit. Akselit määrittävät suhteen isä- ja lapsisolmujen välillä liikkumiseen.

Taulukko 2. Useimmin käytetyt XPath –akselit [18] [10].

Akseli	**Kuvaus**
child	sisältää kontekstisolmun lapset
descendant	sisältää kontekstisolmun jälkeläiset (lapset, lapsenlapset)
parent	sisältää kontekstisolmun edeltäjän, mikäli sellainen on.
ancestor	sisältää kontekstisolmun edeltäjät aina juurisolmuun asti
following-sibling	sisältää kaikki kontekstisolmua seuraavat sisaret. Sisar on kontekstisolmun kanssa samantasoinen solmu.
preciding-sibling	sisältää kaikki kontekstisolmua edeltävät sisaret. Sisar on kontekstisolmun kanssa samantasoinen solmu.
following	sisältää kaikki dokumentissa olevat solmut, jotka seuraavat kontekstisolmua
preceding	sisältää kaikki solmut jotka edeltävät kontekstisolmua
attribute	sisältää kontekstisolmun attribuutit
namespace	sisältää kontekstisolmun nimiavaruussolmut
self	sisältää kontekstisolmun
descendant-or-self	sisältää kontekstisolmun ja sen jälkeläiset

Alla on esitetty esimerkkejä Xpath-kielen operaattoreiden käytöstä [23].

Lauseke:	Kuvaus:
kirjakauppa	valitsee kaikki kirjakauppa elementin lapsisolmut
/kirjakauppa	valitsee kirjakauppa juurielementin.
kirjakauppa/kirja	valitsee kaikki kirja elementit, jotka ovat kirjakaupan lapsielementtejä.
//kirja	valitsee kaikki kirja elementit riippumatta sijainnista
kirjakauppa//kirja	valitsee kaikki kirja elementit jotka ovat kirjakauppa elementin jälkeläisiä
//@kissa	valitsee kaikki kissa attribuutit

Predikaatteja käytetään ilmaisemaan tietty solmu tai sen sisältämä arvo. Esimerkiksi lauseke /kirjakauppa/kirja[hinta>35.00] valitsee kirjakauppa-elementin kaikki kirja-elementit, joiden hinta-elementin arvon on suurempi kuin 35.00.

Mikäli haettavan XML-elementin arvo on tuntematon, voidaan käyttää XPathin tukemia jokerimerkkejä. Esimerkiksi lauseke //kappale[@*] valitsee kaikki kappale-elementtien sisältämien attribuuttien arvot.

Valinta useampien polkujen väliltä onnistuu käyttämällä | operaattoria XPath -lausekkeessa. Esimerkiksi lauseke //kirja/nimi | //kirja/hinta valitsee kaikkien kirja-elementtien nimi- ja hinta-elementit.

Seuraavassa esimerkissä suoritetaan Xpath-muotoisen kysely käyttämällä XMLType-tietotyypin tarjoamaa funktiota extractValue():

```
SELECT PUHELIN FROM tyontekija P
  WHERE extractValue(PUHELIN, '/puhelin/koti') = '013-345678';

PUHELIN
--------------------------------------------------------------
<puhelin>
  <koti>013-345678</koti>
</puhelin>
```

4.3 XSQL

XSQL on Oraclen kehittämä tekniikka, joka mahdollistaa XML-pohjaisen tiedonvaihdon dynaamisten web-sovellusten ja tietokannan välillä. XSQL siis yhdistää XML:n, SQL:n ja XSLT:n tarjoamat mahdollisuudet julkaista dynaamisia web-sivustoja, joiden käyttämä tieto pohjautuu tietokantaan. XSQL on Java-servletti ja se muistuttaa paljon JSP (Java Server Pages) –tekniikkaa [7]. Juuri Javan laitteistoriippumattomuuden ansiosta XSQL integroituukin helposti erilaisiin palvelinympäristöihin ja muihin järjestelmiin.

Esimerkki XSQL-tiedostosta:

```
<?xml version="1.0"?>
<xsql:query connection="yhteys" xmlns:xsql="urn:oracle-xsql">
  SELECT * from JASEN order by ETUNIMI
</xsql:query>
```

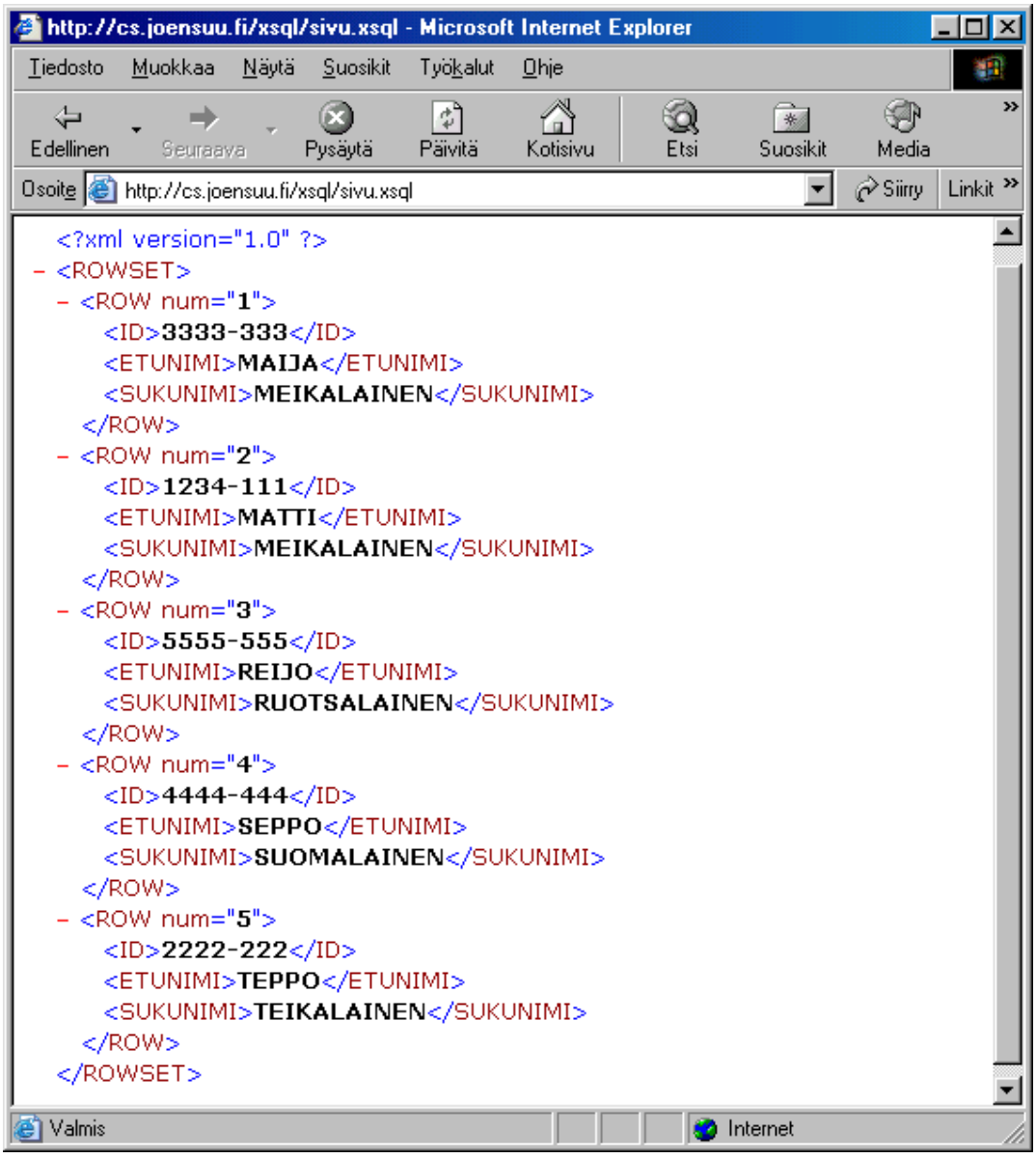
XSQL-tiedosto on siis XML-dokumentti, jossa SQL-kysely on sijoitettu <xsql:query . . . > tagin sisään. Ylläolevassa esimerkkikyselyssä relaatiotietokannan JASEN-taulusta haetaan tietoa etunimen mukaisessa järjestyksessä. Attribuutti connection määrittelee käytettävän yhteyden, joka viittaa XSQLConfig.xml -tiedostossa määriteltävään JDBC-yhteyteen.

Yhteys tietokannan ja XSQL-sivustojen välillä määritellään XSQLConfig.xml tiedostossa:

```
<connection name="DEMO">
    <username>user</username>
    <password>passwd</password>
    <dburl>jdbc:oracle:thin:@lcs.joensuu.fi:1521:xml</dburl>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
    <autocommit>false</autocommit>
  </connection>
```

Yhteys DEMO määrittelee JDBC-yhteyden tietokannan user/passwd -skeemaan. Yhteyden muodostamiseen käytetään Oraclen JDBC thin -ajuria, jonka avulla palvelin kuuntelee tulevia XSQL-pyyntöjä portin (1521) kautta.

XSQL-kyselyn <xsql:query> tulos käsitellään XSU:n tavoin. Tulos siis julkaistaan <ROWSET> -tagin sisältönä, jossa jokainen haettu tietokannan rivi sisältyy aina <ROW> -elementin sisään. Aikaisemman esim. mukaisen XSQL-sivun tulostus on esitetty kuvassa 5.



Kuva 5. XSQL-esimerkkituloste

Koska XSQL-tiedosto on validi XML-dokumentti, siihen voidaan myös sisällyttää tuloksen muotoiluun vaikuttava XSLT-tyylitiedosto.

XSLT-tyylitiedosto voidaan liittää seeuraavasti:

```
<?xml-stylesheet type="text/xsl" href="myXSL.xsl"?>
```

XSLT-tyylitiedoston avulla kyselyn tulos voidaan konvertoida esim. XML-muodosta HTML-muotoiseksi esitykseksi. Sama tulos on siis mahdollista esittää useilla eri tavoilla ja useissa erilaisissa järjestelmissä, kuten esimerkiksi eri selaimissa, mobiili- ja PDA-laitteissa.

XSQL-tekniikka mahdollistaa myös XSU:n tavoin tiedon päivittämisen ja tallentamisen XSQL-sivujen kautta tietokannan tauluihin ja näkymiin. XSQL-tagit <xsql:update-request>, <xsql:delete-request> ja <xsql:insert-request> määrittävät ko. toimenpiteen, eli tiedon päivittämisen, poistamisen tai tallentamisen tietokannan tauluun tai näkymään.

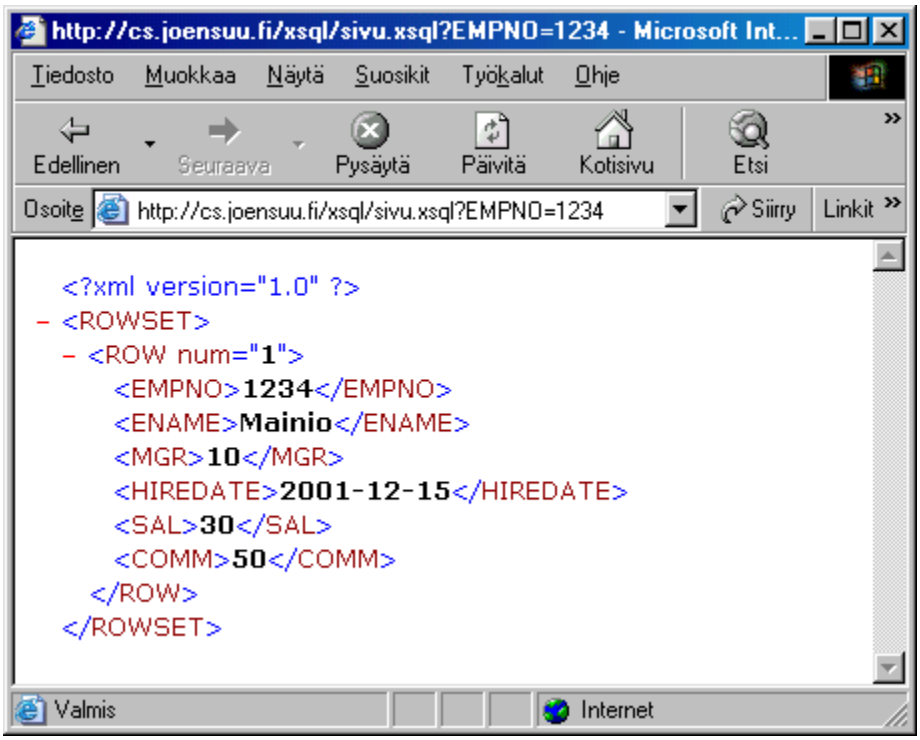
XSQL-tekniikka tukee myös parametrien välitystä XSQL-sivujen kautta tietokantapalvelimelle. Määre `bind-params` määrittää parametrina välitettävän tiedon, joka alla olevassa esimerkissä vastaa tietokannan emp-taulun empno-saraketta.

```
<?xml version="1.0"?>
<xsql:query connection="yhteys" bind-params="EMPNO"
xmlns:xsql="urn:oracle-xsql">
 SELECT * from emp where empno = ?
</xsql:query>
```

Parametri voidaan välittää web-sivun kautta seuraavasti:

```
http://cs.joensuu.fi/xsql/sivu.xsql?EMPNO=1234
```

Kyselyn tuloksena generoitu web-sivu näyttää kuvan 6 mukaiselta.

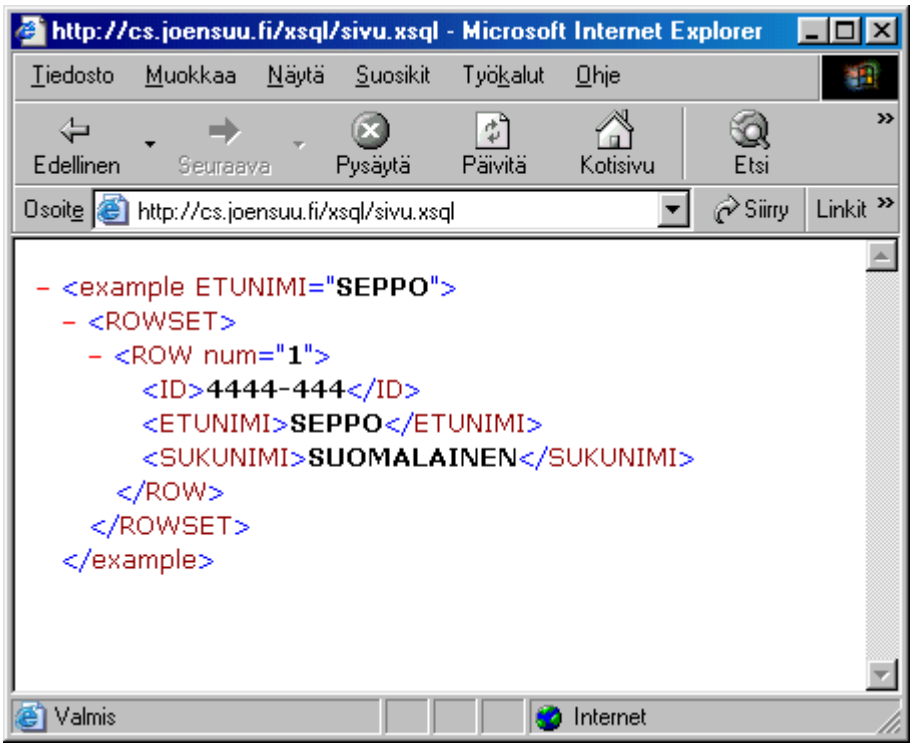


Kuva 6. XSQL-esimerkkituloste parametrilliseen kyselyyn

Myös parametrien alustus on mahdollista XSQL-sivuissa. Esimerkiksi alla oleva esimerkki hakee tietokannan JASEN-taulusta SEPPO-nimisen jäsenen tiedot, mikäli web-sivun kautta välitettävä parametri jätetään antamatta.

```
<example ETUNIMI="SEPPO" connection="yhteys" xmlns:xsql="urn:oracle-
xsql">
  <xsql:query bind-params="ETUNIMI">
    SELECT * FROM JASEN WHERE ETUNIMI = ?
  </xsql:query>
</example>
```

Tulostus näyttää siten seuraavalta kuvan 7 mukaiselta ilman sivun kautta välitettävää parametria.

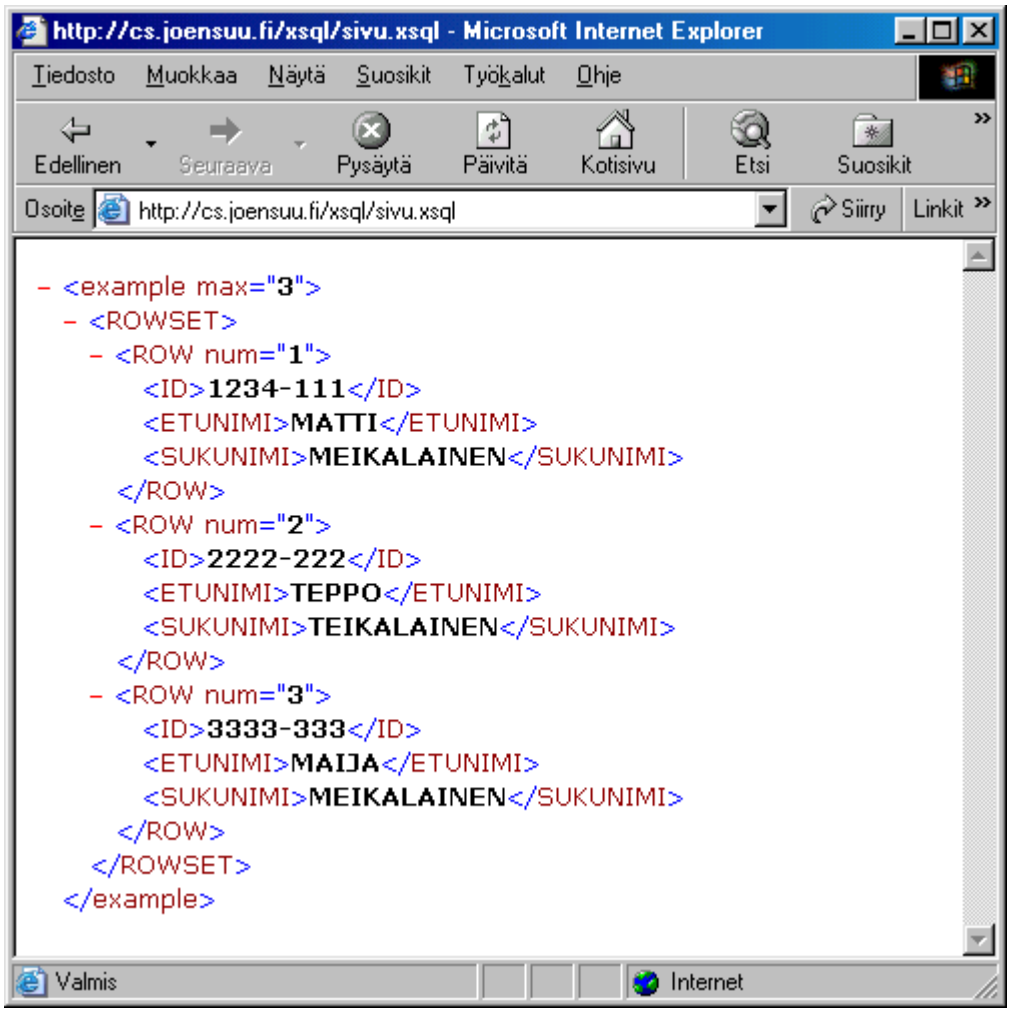


Kuva 7. XSQL-esimerkkituloste oletusparametrilla

Mikäli web-sivulle tulostuvien rivien määrää halutaan rajoittaa, niin se onnistuu määreellä `max-rows`. Kyseinen määre on järkevä etenkin niissä tapauksissa, joissa tiedämme, että haettavan datan määrä voi olla erittäin suuri, esim. yksittäinen taulu voi sisältää useita tuhansia rivejä. Näin voimme pitää huolen siitä, että web-sivulle tulostettavien tietojen määrä pysyy järkevissä mittasuhteissa. Alla olevassa esimerkissä haluamme tulostaa ainoastaan kolmen ensimmäisen rivin tiedot:

```
<example max="3" connection="yhteys" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}">SELECT * FROM JASEN</xsql:query>
</example>
```

Tulostus näyttää siis seuraavalta kuvan 8 mukaiselta.



Kuva 8. XSQL-esimerkkituloste rajoitettuun kyselyyn

4.4 XQuery

XML Query (XQuery) on W3C:n julkaisema standardi, joka mahdollistaa kyselyjen tekemisen XML-pohjaiselle datalle. Aivan kuten SQL on kyselykieli relaatiotietokannan sisältämälle tiedolle, XQuery on sitä XML-pohjaiselle tiedolle [9]. W3C:n tavoitteena on ollut suunnitella XQuery:sta kyselykieli, jonka syntaksi on tiivis ja helposti ymmärrettävä.

Yhtenä tavoitteena on myös ollut laajennettavuus ja mahdollisimman monipuolinen XML-tietolähteiden käyttö. XQuery sulautuukin helposti eri tietolähteisiin [18]. Kyselyjä voidaan tehdä niin XML-dokumentista kuin relaatiotietokannastakin. Myös monet muut tietolähteet ovat hyvin tuettuja, kuten esim. tiedonhakeminen Web Services –viesteistä [1].

XQuery on laajennus Xpath-kielestä ja siksipä standardit XQuery 1.0 ja XPath 2.0 jakavatkin saman tietomallin, funktiot ja operaattorit. XQuery on yhteensopiva useiden W3C-standardien kanssa kuten XML, nimiavaruudet, XSLT, XPath ja XML-skeema. XQueryn käyttämät tietotyypit ovat samoja XML Schema 1.0 -standardin kanssa [4].

XQuery-kyselykielen kehityksen myötä useat eri ohjelmistoyhtiöt ovat ymmärtäneet XQueryn merkityksen XML-pohjaisissa järjestelmissä ja täten he ovat kehittäneet tuen XQuerylle omaan tuotteeseensa. Esimerkiksi Oracle 10g -tietokanta sisältää tuen XQuery- kyselykielelle [1]. Markkinoilla on saatavilla useita eri rajapintoja XQuery-kyselyjen tekoon, joista yhtenä tunnetuista voidaan mainita Java-pohjainen Saxon-B. Saxon-B on XQuery-prosessori, joka mahdollistaa kyselyjen tekemisen XML-pohjaiselle tiedolle [18].

Käyn seuraavaksi läpi XQuery-kielen kielioppia esimerkkien kautta. Tarkastelussa hakutapahtumat perustuvat Liitteessä 2 esitettyyn books.xml –tiedostoon. Kyselyjen tekemiseen on käytetty sekä Oraclen XQuery XQLPlus-prototyyppirajapintaa että Saxon-B -rajapintaa.

4.4.1 Polkulausekkeet

XQueryssä käytettävät polkulausekkeet ovat siis tuttuja aikaisemmin käsitellyn XPath-kielen kautta. Funktio document( ) tai doc( ) palauttaa haettavan dokumentin juurielementin eli toisin sanoen doc() -funktiota käytetään ikään kuin avaamaan haettu dokumentti. Polkulausekkeen askeleita merkataan siis ”/” ja ”//” –merkeillä, joista yksittäinen ”/” merkki muodostaa absoluuttisen polun dokumentin juurielementtiin.

Esimerkki yksikertaisesta XQuery kyselystä:

```
document("books.xml")/bookstore/book/title
```

/bookstore -lauseke palauttaa siis bookstore elementin, /book valitsee kaikki book-elementit bookstore-elementin alta, ja /title valitsee kaikki tittle-elementit kaikkien book-elementtien alta. Kyselyn tulosena tulostuu seuraavaa:

```
> java oracle.xquery.XQLPlus kysely.xql

XQuery Command Line Tool
Version 1.0
Enter XQuery statements followed by /;

Result
--------------------------------------------------
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
>
```

4.4.2 Predikaattimääreet

Predikaattimääreiden avulla tulostusta voidaan rajata XQuery-kyselyssä. Esimerkiksi seuraava esimerkkikysely palauttaa kaikki bookstore-elementin alla olevat book-elementit, joiden hinta on pienempi kuin 30.

```
document("books.xml")/bookstore/book[price<30]

> java oracle.xquery.XQLPlus kysely.xql

XQuery Command Line Tool
Version 1.0
Enter XQuery statements followed by /;
```

```
Result
------------------------------------------------
<book category="CHILDREN">
   <title lang="en">Harry Potter</title>
   <author>J K. Rowling</author>
   <year>2005</year>
   <price>29.99</price>
</book>
```

4.4.3 FLWOR-lauseke

FLWOR-lauseke, jota usein myös kutsutaan nimellä ”flower” muodostuu lausekkeessa käytettävien komponenttien mukaan: **F**or, **L**et, **W**here, **O**rder by, ja **R**eturn [9]. Tätä voidaan ajatella XQuery:n vastineeksi SQL:stä tutulle SELECT-FROM-HAVING-WHERE –rakenteelle. FLOWR-lauseke muodostuu aina yhdestä tai useammasta FOR- ja/tai LET-lausekkeesta, joita seuraa aina pakollinen Return lauseke, joka päättää kyselyn. Where –osiota voidaan käyttää haun rajaamiseen ja Order by -osiolla tulostus voidaan hoitaa mieleisessä järjestyksessä. For-lausekkeella määritellään siis käytettävä hakusilmukka, ja Let-lausekkeen avulla esitellään ja alustetaan mahdollinen käytettävä muuttuja. Return-lauseke suoritetaan jokaista silmukan kierrosta kohden, mutta tulostus hoidetaan vasta viimeisen kierroksen jälkeen. Alla olevassa esimerkissä kysely hakee kaikki bookstore-elementin sisäiset book-elementit, joiden price-elementti on suurempi kuin 30. Return palauttaa kaikki title-elementit niiden mukaisessa järjestyksessä.

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Kyselyn tulos on siis seuraava:

```
> java net.sf.saxon.Query kysely.xql
<?xml version="1.0" encoding="UTF-8"?>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

4.4.4 Muut määreet

XQuery-kieli tukee suurta joukkoa erilaisia operaattoreita, funktioita ja tietotyyppejä. Myös perinteiset ohjelmoinnista tutut ehdolliset lauseet IF-THEN-ELSE kuuluvat XQuery:n tuen piiriin.

Esimerksi ehdollinen kysely

```
for $x in doc("books.xml")/bookstore/book
return  if ($x/@category="CHILDREN")
        then <child>{data($x/title)}</child>
        else <adult>{data($x/title)}</adult>
```

palauttaa seuraavan tulosjoukon:

```
java net.sf.saxon.Query kysely.xql
<?xml version="1.0" encoding="UTF-8"?>
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>XQuery Kick Start</adult>
<adult>Learning XML</adult>
```

Oracle mahdollistaa myös XQuery-kyselyjen tekemisen suoraan SQL*Plus komentotulkin kautta. Kyselyt kirjoitetaan XQUERY ja (/) lohkon väliin seuraavan esimerkin mukaisesti.

```
XQUERY for $i in ora:view("JASEN")
where $i/ROW/SUKUNIMI = "MEIKALAINEN"
return $i
/
```

Kun suoritamme esimerkkikyselyn, tuloksena saamme:

```
Result Sequence
-----------------------------------------------------------------
<ROW><ID>1234-
111</ID><ETUNIMI>MATTI</ETUNIMI><SUKUNIMI>MEIKALAINEN</SUKUNIMI></ROW>
<ROW><ID>3333-
333</ID><ETUNIMI>MAIJA</ETUNIMI><SUKUNIMI>MEIKALAINEN</SUKUNIMI></ROW>
```

5 YHTEENVETO

Tutkielman tavoitteena oli tutustua erilaisiin XML-kyselytekniikoihin ja niihin liittyviin standardeihin. Tavoitteena oli myös päästä testaamaan niitä käytännössä ja tässä myös onnistuttiin. Tulevaisuuden informaatiojärjestelmät tulevat näillä näkymin olemaan edelleenkin riippuvaisia XML-pohjaisesta tiedosta. Talletettavan tiedon sekä niihin kohdistuvien hakujen määrä tulee kasvamaan edelleen. XQuery:n tyyppisille kyselykielille on ollut selkeä tilaustarve jo kauan aikaa. Myös eri ohjelmistovalmistajat ovat huomanneet tarpeen tukea osaa W3C:n julkaisemista standardeista erilaisten rajapintojen, ajurien ja ohjelmiston lisäosien muodossa. On varmasti haasteellista ja jopa riskialtistakin uhrata resursseja jonkin tulevan standardin tukemiseen, joka hyvin usein on vasta kehitteillä, ei siis vielä valmis standardi.

Myös useat eri tietokannanhallintajärjestelmien valmistajat ovat julkaisseet oman tukensa XML-tyyppiselle datalle. XML-tietotyypit ja erilaiset XML-kieltä tukevat tekniikat ja standardit helpottavat nykyajan kehittäjien työtä huomattavasti. Vaikka tuettujen tekniikoiden käyttöönotosta on tullut entistä helpompaa, niin aina kannattaa kuitenkin muistaa muutama perussääntö. Käytettävä tekniikka tulee valita huolella aina tilanteen mukaan. Esimerkiksi DOM-mallin tarjoaman rajapintaan perustuva jäsennys ei toimi, mikäli käy-tettävän XML-dokumentin koko on erittäin suuri. Myös eri tekniikoiden käyttöönotto voi pahimmassa tapauksessa aiheuttaa ylimääräistä päänvaivaa varsinkin siinä vaiheessa, kun tuettavaa standardia ei vielä ole, vaan se on vasta kehitteillä. Kaiken kaikkiaan XML ja siihen liittyvät tekniikat sekä kyselykielet jättävät kuitenkin hyvin myönteisen kuvan tämän hetken tarjoamista mahdollisuuksista informaation hakuun liittyvien tekniikoiden saralla. Toivottavasti ko. tekniikoiden tulevaisuus jatkuu yhtä myönteisenä.

VIITELUETTELO

[1] *Oracle XML DB Developer's Guide 10g Release 2 (10.2)* WWW-sivusto, http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14259/toc.htm (26.4.2008)

[2] *Oracle XML Developer's Kit Programmer's Guide 10g Release 2* (10.2) WWW sivusto,http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14252/toc.htm (2.5.2008)

[3] W3C (2004)*Extensible Markup Language (XML)*, WWW-sivusto, http://www.w3.org/TR/xml/ (18.4.2008)

[4] W3C (2007) *XQuery 1.0: An XML Query Language*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/xquery/ (2.5.2008)

[5] *XML Tutorial*, WWW-sivusto, http://www.w3schools.com/xml/default.asp (3.5.2008)

[6] SQL/XML in JDBC Applications, DataDirect Networks (2004)

[7] Chaudhri A. B., Rashid A., Zicari R. (2003) *XML Data Management: Native XML and XML-enabled Database Systems*, Boston, USA

[8] Loney K., Koch G., (2002) *Oracle9i: The Complete Reference*

[9] Thomas M. Connolly, Carolyn E. Begg (2004) *Database Systems: A Practical Approach to Design, Implementation and Management,* Harlow, England

[10] Benz B., Durant J.(2004) *XML Programming Bible,* New York, USA

[11] Greenwald R., Stackowiak R., Dodge G., Klein D., Shapiro B, Chelliah C. G. (2005), *Professional Oracle Programming*.

[12] Pardalos P. M. (2004), *Supply Chain and Finance*. London, England

[13] *XML-SQL Utility (XSU).* WWW-sivusto, http://www.oracle.com/technology/tech/xml/xdk/doc/production/plsql/doc/plsql/xsu/xsu_userguide.html (18.4.2008)

[14] *Store and Retrieve XML from Databases with XSU.* WWW-sivusto, http://www.devx.com/xml/Article/32046 (21.4.2008)

[15] Tuikka, T., Kanala, S. (2001) *XML Basic*. Oy Edita Ab, Helsinki

[16] W3C (2005) *Document Object Model (DOM)*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/DOM/ (21.4.2008)

[17] W3C (2000) *XML Schema*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/XML/Schema (11.5.2008)

[18] Keogh J., (2005) *XML demystified.* McGraw-Hill Osborne Media.

[19] Walkama P., Laakkonen A. (2004) *XML-Skeema*. Edita Prima Oy, Helsinki

[20] Nachimovsky A., Myers T. (2002) *Inside Java ja XML*. Edita Prima Oy, Helsinki

[21] McLaughlin B. (2001), *Java & XML*, Gummerus Kirjapaino Oy, Jyväskylä

[22] Holzner S. (2001), *Inside XML*, Gummerus Kirjapaino Oy, Jyväskylä

[23] *XPath Tutorial*, WWW-sivusto, http://www.w3schools.com/xpath/default.asp (23.4.2008)

[24] Oracle9i XML Database Developer's Guide - Oracle XML DB Release 2 (9.2) WWW-sivusto, http://lbdwww.epfl.ch/f/teaching/courses/oracle9i/appdev.920/a96620/toc.htm (26.4.2008)

LIITE 1: bib.xml

```
<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>39.95</price>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for 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>
```

LIITE 2: books.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```