Rakenteisen tekstin tallennusrakenteet

Hannu Toroskainen

12.6.2008

Joensuun Yliopisto
Tietojenkäsittelytiede
Pro Gradu –tutkielma

Tiivistelmä

Dokumentit eri muodoissaan ovat keskeinen osa organisaatioiden, ja sähköisten palveluiden lisääntyessä yhä enemmän myös yksityishenkilöiden, välistä tiedonvaihtoa. Jotta lisääntyvä tietomäärä olisi hallittavissa, käytettävissä ja siirtyisi järjestelmien välillä oikein, tarvitaan keinoja esittää tiedon rakenne niin, että erityyppisen tekstin tallennustarpeet tulevat huomioonotetuiksi. Dokumenttien siirrolle ja arkistoinnille on käytössä xml-standardi, mutta tekstin monimuotoisuuden takia rakenteen ohjelmalliselle hallinnalle on vaikeampi määritellä xml:ää tai relaatiomallia vastaavaa yleispätevää mallia, jota kaikentyyppiset rakenteiset dokumentit sujuvasti noudattaisivat. Tallennusratkaisua valittaessa joudutaankin tekemään valintoja tallennettavan tiedon tyypin ja sen käyttötarpeen mukaan.

Sisältö

1 Johdanto

Tässä tutkielmassa selvitetään, mitä tarkoitetaan rakenteisen tekstin tallennusrakenteilla ja miksi teksti ylipäänsä tarvitsee erillisen rakenteen määrittelyn. Lisäksi selvitetään, mitä tallennusratkaisuja on käytössä ja mitä tulee ottaa huomioon tallennusratkaisua valittaessa. Tarkastellaan kuitenkin aluksi, mitä on rakenteinen teksti.

1.1 Rakenteinen teksti

Teksti on luonnollista kieltä (Loeffen, 1994) tai kuten Clarke & al. (1994) sanovat, tekstillä on ”luonnollinen rakenne”. Esimerkiksi kirja koostuu nimestä, tekijöistä, kustantajasta, sisällysluettelosta, luvuista, alaluvuista, kappaleista, kuvista, lähdeluettelosta ja niin edelleen. Kirjeessä on päiväys, saaja, sisältö ja lähettäjä. Lukija tunnistaa kirjan tai kirjeen osan sen sijainnin tai merkintätavan perusteella. Esimerkiksi kirjan nimi on kirjan alussa ja luvun, joka alkaa kokonaisluvulla, perässä on otsikko fontin ollessa vahvennettu ja koon suurempi kuin varsinaisen tekstin fontti. Lukija tunnistaa kuvan 1 kirjeessä olevan päiväyksen 5.5.2008 kirjeen kirjoituspäiväksi. Asiayhteydestä erotettunakin se oletetaan päiväykseksi, mutta esimerkiksi muodossa 05052008 se voi olla vaikkapa asiakasnumero.

```
                                  Kontiolahti 5.5.2008

Hei!

Mitä kuuluu? Tänne kuuluu hyvää. Kesäloma alkaa jo painaa päälle
ja ennen sitä täytyy saada tämä projekti valmiiksi.
…
Hyvää kesänjatkoa!

Hannu
```

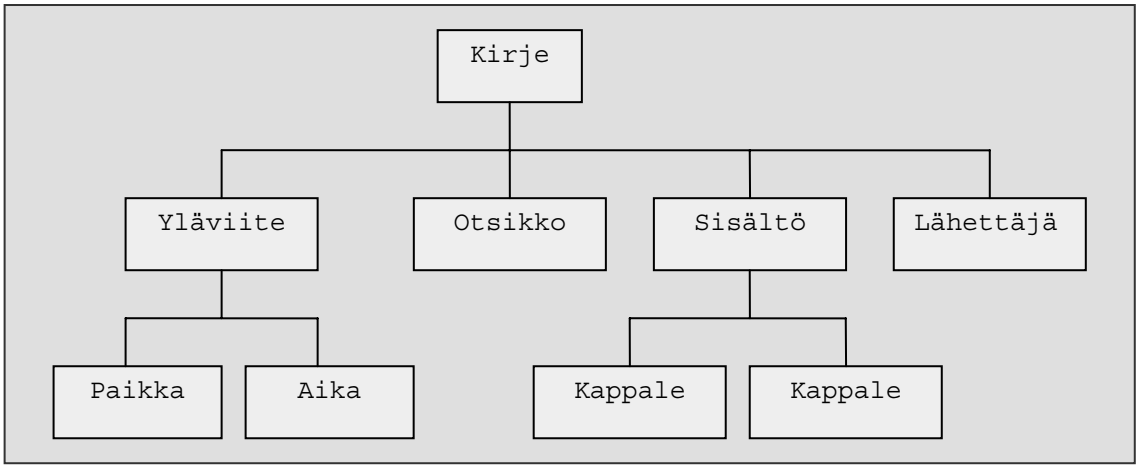
Kuva 1: Esimerkki rakenteettomasta tekstistä.

Dokumenttien ominaisuudet ja rakenteet vaihtelevat paljon jopa yhden dokumentin sisällä (Clarke & al., 1994). Lukuja ja alalukuja voi olla vaihteleva määrä, kirjoittajia voi olla useita tai he voivat olla tuntemattomia. Dokumentti voi olla kirjoitettu japaniksi kirjoitusmerkein tai suomeksi. Dokumentti voi olla osa suurempaa kokonaisuutta tai se voi olla itsenäinen kokonaisuus.

Ellei dokumentin rakennetta pystytä ohjelmallisesti käsittelemään, puhutaan *vapaamuotoisesta* tai *rakenteettomasta tiedosta* (Kuikka & al., 1999). Tällaisia dokumentteja ovat esimerkiksi tekstitiedostot, joissa tekstin osia ei ole erotettu toisistaan esimerkiksi *erottimin* eli *tagein*. Näissäkin rakenne on olemassa, mutta se on esitetty *typografisin* eli esitysmuodollisin keinoin kuten kirjan luvut edellä. Muuta rakenteetonta tietoa ovat esimerkiksi kuva-, ääni- ja videotiedostot. Jos dokumentin rakenneosat esitetään *eksplisiittisesti* eli niin selkeästi, että tietokoneohjelmat pystyvät ne tunnistamaan ja niitä käsittelemään riippumatta dokumentin ulkoasusta, voidaan puhua *rakenteisesta dokumentista* (Kuikka & Nikunen, 1994). Kuvassa 1 on esimerkki rakenteettomasta dokumentista ja kuvassa 2 vastaavasta dokumentista, jossa dokumentin osat eli elementit on erotettu toisistaan erottimin ja ovat siten ohjelmallisesti tunnistettavissa. Esimerkiksi päiväys tunnistetaan erottimien `<päiväys>` ja `</päiväys>` välisestä osasta. Kuvan 2 esimerkissä kirjeellä on hierarkkinen rakenne, joka voidaan kuvata kuvan 3 puumallisella kaaviolla.

```
<kirje>
<yläviite><paikka>Kontiolahti</paikka><päiväys>5.5.2008</päiväys></yläviite>

<otsikko>Hei!</otsikko>

<sisältö>
<kappale> Mitä kuuluu? Tänne kuuluu hyvää. Kesäloma alkaa jo painaa päälle
ja ennen sitä täytyy saada tämä projekti valmiiksi.</kappale>

<kappale> Hyvää kesänjatkoa!</kappale>
</sisältö>

<lähettäjä>Hannu</lähettäjä>
</kirje>
```

Kuva 2: Esimerkki tunnistimin varustetusta rakenteisesta tekstistä.



Kuva 3: Esimerkkikirjeen puumallinen esitys.

Kun kuvan 2 erottimin varustettu esimerkkitiedosto tallennetaan levylle odottamaan esimerkiksi siirtoa toiseen järjestelmään, on se tallennettu sinne *ulkoisen tallenusrakenteen* mukaisesti. Kun kyseinen tiedosto luetaan esimerkiksi rakenteisuutta tukevaan tekstinkäsittelyjärjestelmään, se organisoidaan sinne *sisäisen tallennusrakenteen* mukaisesti, kuten kuvassa 3 puumallisesti. Tallennusrakenteet eivät näy käyttäjille tällä tavoin, vaan tekstinkäsittelyjärjestelmä esittää dokumentin *käsitteellisessä* muodossa lukuina, alalukuina, kappaleina ja niin edelleen. Jos dokumenttia käytetään pelkästään järjestelmien välisessä tiedonsiirrossa, ei käsitteellistä muotoa tarvita, vaan esimerkiksi relaatiotietokannan tauluista muodostetaan xml-muotoinen siirtotiedosto.

Erottimin varustettua tiedostoa voidaan käsitellä sellaisenaankin esimerkiksi tekstieditorilla, mutta teksti on tällöin vaikealukuista. Sisällön lisäksi editoijan on huolehdittava rakenteesta, mikä on virhealtista sillä rakenteen osien tulee olla täsmälleen oikeissa paikoissa oikein nimettyinä.

1.2 Historiallista taustaa

Isojen keskuskoneiden aikaan tietokoneilla katsottiin olevan tärkeämpää tekemistä kuin tekstinkäsittely (Kuikka & Nikunen, 1994). Ensimmäiset tekstinkäsittelyyn tarkoitetut järjestelmät olivat niin sanottuja vuorovaikutteisia ladontajärjestelmiä, joissa teksti kirjoitettiin lähinnä ohjelmoijien käyttöön tarkoitetulla tekstieditorilla ja muotoilukomennot lisättiin tekstin sekaan. Editoija näki tekstin samanlaisena

raakatekstinä kuin oli muotoiluohjelman sisäinen esitysmuoto (Salminen, 1992) ja lopputulos nähtiin vasta paperitulosteesta, joten dokumenttia täytyi väliin muokata ja väliin tulostaa muotoiluohjelmaa käyttäen. Muotoiluohjelmia käytetään nykyäänkin, ja esimerkiksi LaTeX:ssa voidaan nimetä ja muotoilla dokumentin osia.

Mikrotietokoneiden ja henkilökohtaisten tietokoneiden (PC - Personal Computer) myötä 1980-luvulla ryhdyttiin puhumaan taitto-ohjelmista (desktop publishing) ja *wysiwyg*–editoreista (What Yoy See Is What You Get), joissa tekstin muotoilu tehdään editoinnin yhteydessä ja ulkoasu nähdään heti ruudulla. Näissäkin muotoilukomennot ovat tekstin seassa (*proseduraalinen merkkaus*), mutta - toisin kuin muotoiluohjelmissa - ne eivät näy käyttäjälle. Kukin ohjelma tallensi dokumentit omassa formaatissaan, usein binaarimuodossa (Ditch, 2007). Dokumentteja pystyttiin lukemaan ja käsittelemään vain kyseisellä ohjelmalla, ja kyetäkseen käsittelemään muualta tulleita dokumentteja jouduttiin pahimmassa tapauksessa hankkimaan useita ohjelmia. Markkinoiden kypsyessä 1980-luvulla jäi jäljelle muutama toimija, ja lopulta Microsoftin *doc-tiedostomuodosta* tuli tekstinkäsittelyn de facto-standardi. Käytännössä tämä tarkoitti sitä, että tarvittiin Microsoft Word-ohjelma, sillä doc-muodosta ei kuitenkaan tullut avointa standardia.

Ongelmat tiedostojen siirrettävyydessä ohjelmasta toiseen johtivat jo 1960-luvulla keskusteluun yleisen dokumenttiformaatin tarpeellisuudesta (Ditch, 2007). Vuonna 1986 julkaistiin Standard Generalized Markup Language (sgml) -kieli, jossa dokumentin rakenne pystyttiin merkkaamaan ja nimeämään niin, että rakenteita ja nimiä ei ole määritelty ennalta, vaan eri sovellusalueille määriteltiin omat rakenteet ja merkkaustavat (Salminen, 2006). Www-julkaisemista varten sgml-kielestä kehitettiin HyperText Markup Language-kieli (html). Internetin käyttö paisui 1990-luvulla kaikkialla läsnä olevaksi tietoverkoksi ja verkossa olevien ohjelmien tuli pystyä kommunikoimaan keskenään. Sgml:n säännöstö oli kuitenkin liian monimutkainen yleiseen käyttöön, joten siitä julkaistiin vuonna 1998 yksinkertaisempi versio *Extensible Markup Language*–kieli *(xml),* jossa on edellisiin verrattuna muitakin etuja kuten se, että sitä voidaan käyttää tiedonsiirrossa sovellusten kesken. Toisin kuin Wordin kaltaisissa editoreissa, xml-kielioppia käyttävissä, niin sanotuissa *deklaratiivisissa* järjestelmissä on ulkoasun määrittely erotettu sisällöstä ja tekstiä käsitellään rakenteisena.

Tekstinkäsittelyjärjestelmien ja sgml-standardin rinnalla kehiteltiin erillisiä rakenteisen tekstin käsittelyyn soveltuvia järjestelmiä. IBM (International Business Machines) kehitti jo 1950-luvun lopulla rakenteisen tekstin tallennukseen ja hakuun tarkoitetun STAIRS-järjestelmän (Storage And Information Retrieval System). Se oli tarkoitettu bibliografisen aineiston tallennukseen ja mahdollisti rakenteeseen perustuvan haun, mutta ei hierarkkisia tai sisäkkäisiä rakenteita eikä *vapaasanahakua* (Hiemstra & Baeza-Yates, 2008). *Tiedonhakujärjestelmiä* (*Information Retrieval - IR*) on käytetty laajemmin 1960-luvulta lähtien relevanttien dokumenttien hakemiseksi dokumenttikokoelmista (Kilpeläinen, 1992). Rakenteettomien elektronisten dokumenttien määrä alkoi tällöin lisääntyä voimakkaasti, eikä dokumentteja voitu tallentaa perinteisiin *taulukkomuotoisiin* (*table-style*) tietokantoihin, joita oli käytetty jo pitkään määrämuotoisen tiedon tallentamiseen (Bennett, 2004). Tiedonhakujärjestelmiin otettiin kuitenkin paljon piirteitä tietokantamaailmasta, ja vieläkin monissa tiedonhakujärjestelmissä on perusrakenteena relaatiomallin taulurakenne.

Yritysmaailmassa on määrämuotoisen tieto ollut perinteisesti etusijalla, ja kuten edellä keskuskoneiden yhteydessä todettiin, on tekstillä ollut toisarvoinen asema. Määrämuotoinen tieto, kuten asiakas- tilaus- ja laskutusrekisterit, tallennettiin alkujaan sovellusten sisäisesti. Tiedot olivat näin ollen vain kyseisen järjestelmän käytössä, minkä lisäksi rakenteen muuttaminen vaati sovelluksen uudelleenkoodauksen (Raymond & al., 1995). Järjestelmäriippuvuudesta haluttiin pois, mutta esimerkiksi tiedon siirrettävyydellä ei ollut oleellista merkitystä vaan riitti, että se on turvassa, päivitykset nopeita ja tieto oikeaa. Näin päädyttiin standardoimaan tiedon semantiikka, ja 1970-luvulla syntyi muun muassa *relaatiomalli*. Tekstimuotoiset dokumentit ovat vastaavasti olleet keskeisiä esimerkiksi oppilaitoksissa ja kirjapainoissa, joissa dokumentteja siirretään paikasta toiseen ja niiden täytyy olla luettavissa reaaliaikaisesti erilaisissa ympäristöissä. Dokumenttien tuottaminen oli alkujaan raskas prosessi, joka täytyi tehdä uudelleen, kun dokumenttia muutettiin tai tehtiin esimerkiksi järjestelmäpäivitys. Tästä haluttiin eroon ja päädyttiin standardoimaan tiedon esitysmuoto. Rakenteeseen ei otettu kantaa.

Pelkkä esitysmuoto ilman rakennetta ei kuitenkaan aina riittänyt, ja perinteiset tiedonhakujärjestelmät todettiin joustamattomiksi ja tehottomiksi moniin tarpeisiin, sillä ne oli tarkoitettu nimensä mukaisesti tiedon hakuun, ei muokkaukseen. Niinpä 1980-luvulla ryhdyttiin tutkimaan tietokantajärjestelmien soveltuvuutta rakenteisten dokumenttien hallintaan (Macleod, 1990). Relaatiomallissa oli pitkälle kehitetyt tiedon haku- ja muokkausominaisuudet sekä *sql-kieli (Strustured Query Language)*, jolla operaatiot toteutettiin. Relaatiomallissa oli lisäksi toteutettu muun muassa usean käyttäjän hallinta, käyttöoikeuksien hallinta ja toipumisominaisuudet, joita haluttiin hyödyntää myös tekstitiedon käsittelyssä.

Relaatiomallin puutteet esimerkiksi tekstin hierarkkisuuden esittämisessä johtivat tutkijat kehittämään 1980-luvulla relaatiomalliin pohjautuvia *käsitteellisiä* malleja, (esimerkiksi Macleodin taulukkomalli), joiden suunnittelun pääpaino oli tiedon semanttisten piirteiden esittämisessä (Macleod & Reuber, 1987). Myöhemmin 1990-luvulla relaatiomalliin on tehty erilaisia laajennuksia, kuten *sisäkkäiset relaatiot* (*nested relations*) ja *olio-laajennus*, sekä sql-kielen laajennuksia, kuten Güttingin sfql–kieli (Structured Fulltext Query Language) (Clarke & al., 1994) ja standardoitu *sql/xml*. Myös puhtaasti *hierarkkisia*- ja *oliotietokantoja* sekä erilaisia hajautettuja ratkaisuja on käytetty (Macleod, 1990).

Laajennuksista huolimatta perinteiset tietokantamallit olivat joustamattomia tekstin monimuotoisuuden hallintaan. Gonnet ja Tompa tutkivat 1980-luvun alussa kielioppipohjaista mallintamista, ja 1980-luvun lopulla julkaistiin elektroninen versio Oxford English-sanakirjasta (OED - Oxford English Dictionary) (Gonnet & Tompa, 1987). Tämä innoitti kielioppipohjaisten mallien laajempaan tutkimiseen, ja tuloksena oli muun muassa Burkowskin Containment-malli (Burkowski, 1992). Myös Gyssens, Paredaens ja Van Gucht (Gyssens & al., 1989) sekä Salminen ja Tompa (Salminen & Tompa, 1999) ovat monen muun ohessa tehneet tutkimusta tällä saralla. Syntaksin ohjaamaa tekstinkäsittelyä ovat tutkineet muun muassa Kuikka ja Salminen (Kuikka & Salminen, 1995).

1.3 Työn rakenne

Viime vuosina ovat niin sanotut xml-tietokannat olleet voimakkaan kehitystyön alla ja esimerkiksi Fuhr, Gövert, Bourret ja Lee ovat aktiivisesti tutkineet aihetta (Trotman, 2003). Kannat on tarkoitettu xml-muotoisen tiedon tallennukseen ja muokkaukseen, ja niissä tallennus tehdään järjestelmän sisäisen tallennusrakenteen mukaisesti. Dokumentteja myös siirretään järjestelmästä toiseen tai arkistoidaan xml-tiedostoina ulkoisen tallennusrakenteen mukaisesti. Tämän tutkielman tarkoituksena on selvittää, mitä tarkoitetaan sisäisellä ja ulkoisella tallennusrakenteella, mitä ratkaisuja niiden toteuttamiseen on käytössä ja mitä tulee ottaa huomioon tallennusratkaisua valittaessa. Vaikka ratkaisujen yhteydessä puhutaankin lähinnä xml-dokumenttien tallennuksesta, soveltuvat useimmat ratkaisut myös muuntyyppisten rakenteisten dokumenttien tallennukseen.

Luvussa kaksi käydään aluksi läpi, miksi tekstin tuottaminen rakenteisena on tullut niin tärkeäksi ja mitä etuja rakenteen lisääminen tuo tekstin käsittelyyn ja hallintaan. Lisäksi luvussa kaksi tarkastellaan, mitä muuta rakenteen lisäksi tarvitaan, jotta asioiden väliset yhteydet saadaan esimerkiksi hakukoneiden hyödynnettäviksi. Luvussa kolme selvitetään, mistä osista rakenteinen dokumentti koostuu ja mistä muodostuu ulkoinen ja sisäinen tallennusrakenne. Ulkoisen tallennusrakenteiden toteutuksessa käytettäviä keskeisimpiä tietomalleja tarkastellaan luvussa neljä. Luvussa viisi esitettyjen sisäisten tallennusratkaisujen yhteydessä esitän oman relaatiomallia ja *xml-tietotyyppiä* käyttävän *tyyppitauluihin* pohjautuvan ratkaisun rakenteisen tietokantavaraston malliksi. Luvussa kuusi arvioidaan esitettyjen mallien soveltuvuutta erityyppisten xml-dokumenttien tallennukseen.

Tallennusratkaisujen arvioinnissa käytetään sekä kirjallisuudesta haettua tietoa että omakohtaisia kokemuksia yli viidentoista vuoden ajalta erilaisista, suurienkin tietomassojen hallintaan liittyvistä tehtävistä it-yrityksissä ja teleoperaattorin palveluksessa käsittäen puhelu- ja laskutustietojen käsittelyä ja niistä raportointia. Muualla dokumentissa viitataan usein dokumenttien hallinnan ongelmiin. Niissä esitetyt kommentit perustuvat sekä omakohtaisiin kokemuksiin että käyttäjien antamiin palautteisiin järjestelmien käytöstä.

2 Rakenteisen tekstin käyttö

Elektronisessa muodossa olevan tiedon hallinta ja tekstin muotoiluun käytetty aika aiheuttavat käyttäjille ongelmia ja organisaatioille isoja kustannuksia. Rakenteisuuden käyttöönotolla saadaan ratkaistua monia ongelmia, mutta pelkästään se ei aina riitä.

2.1 Elektronisen tiedon hallinnan ongelmat

IDC:n (International Data Corporationin*)* mukaan vuonna 2006 massamuisteilla oli tietoa 161 eksatavua (1 eksatavu (EB)=10^{18} tavua) ja vuonna 2007 sitä arvioitiin olevan 255 eksatavua (Nikulainen, 2007). Tallennetun tiedon vuotuinen kasvu on 57 prosenttia, ja vuonna 2010 massamuisteilla arvioidaan olevan jo 988 eksatavua tietoa. Tallennettavan tiedon määrää yritysmaailmassa lisäävät esimerkiksi sähköpostiliikenteen lisääntyminen, internet-puheluiden ja *rfid*-tietojen (*Radio Frequency Identification* eli *Radiotaajuinen etätunnistus*) tallennustarpeet, sovellusten välinen tiedonsiirto sekä uudet säädökset ja direktiivit. Rfid-tunnisteen käyttömahdollisuudet ovat lähes rajattomat tavaroiden, eläinten ja jopa ihmisten kulun seuraamiseen. Tästä seuraa kuitenkin, että aina kun tunnisteella varustettu kulkee tunnistimen ohi, jää siitä jälki tietokantaan. Viranomaismääräyksenä on eduskunta hyväksynyt esityksen, jolla teleyritykset velvoitetaan tallentamaan teletunnistetiedot 12 kuukauden ajan alkaen viimeistään 15.3.2009 (Tietosuojalaki, 2008).

Suurten tietomassojen hallinta on usein työlästä, sillä tietojen täytyy olla tallennettu niin, että ne ovat tarvittaessa käytössä kuitenkin niin, että tietoja pääsevät katsomaan vain niihin oikeutetut henkilöt. Esimerkiksi teletunnistetietojen käsittelylle ja säilyttämiselle on annettu viranomaistaholta erittäin tiukat määräykset. Tiedot eivät saa joutua ulkopuolisten käsiin ja esimerkiksi puheluerittelyn tulostamisesta täytyy jäädä lokitieto. Suuren massan takia tietoja ei voida säilyttää pitkiä aikoja tietokannoissa, vaan ne on arkistoitava arkistokantaan tai esimerkiksi dvd-levylle poltettuina kassakaappiin. Tilinpäätöstiedoissa säilytysaika on vielä pidempi eli kymmenen vuotta. Tietojen arkistointi tietokantatiedostoina voi estää niiden myöhemmän käytön, sillä tietojen lukuun pystyvää ohjelmaa tai ohjelmaversiota ei välttämättä ole enää käytössä. Tästä syystä tallennus rakenteisena, esimerkiksi xml:nä, on turvallinen vaihtoehto, sillä kyseiset tiedot ovat tällöin luettavissa xml:ää

tukevaan ohjelmaan tai tietokantaan tai tiedostosta voidaan tehdä hakuja suoraan vaikkapa *xquerylla* tai *grep*:llä.

Esimerkki toisenlaisesta tiedosta ovat pankkien siirtoaineistot, joissa tietueet tallennetaan riveittäin siten, että yksittäiset tiedot ovat peräkkäin ilman erottimia. Tiedon löytäminen riviltä on vaikeaa, sillä yksittäisiä kenttiä ei eroteta toisistaan, vaan kentän pituus on tiedettävä ja kerrottava lukevalle ohjelmalle. Yksittäisen tiedon etsiminen esimerkiksi tarkastusmielessä on sekä virhealtista että vaikeaa. Kuvassa 4 on esimerkki pankkien maksuliikenteessä käytetystä maksutapahtumatietueesta, jossa siirrettävä summa kerrotaan merkeissä 78-87 rivin alusta laskien.

```
39000687009014008060608060606068X01579585180000000000123456789K
ALLEN PULTT 1J00000127040A
```

Kuva 4: Esimerkki pankin maksuliikennetapahtumasta.

On arvioitu (Kuikka & Nikunen, 1994; Virk, 2002), että 20 % tiedosta on rakenteista eli *määrämuotoista* ja loput 80 % rakenteetonta eli *vapaamuotoista* tai *puolirakenteista*. Rakenteisella tiedolla on täsmällinen rakenne ja tyyppi, ja se on saatu hallintaan tietokannoilla. Puolirakenteisella tiedolla, joita edustavat rakenteiset dokumentit[1], rakenne voi olla vapaampi ja tiedon tyyppi epämääräisempi, kuten esimerkiksi kuvan 2 kirjeessä. Rakenteeton ja puolirakenteinen tieto sijaitsee hajallaan tiedosto- tai webbipalvelimilla, työasemilla ja dokumenttienhallintajärjestelmissä useissa eri muodoissa (doc, pdf, html, xml). Levytilojen kasvun ja halpenemisen myötä on henkilökohtaisissa työasemissakin satoja gigatavuja levytilaa ja samojakin tiedostoja tallennetaan useisiin eri paikkoihin. Koska versionhallintaa ei ole, ei välttämättä tiedetä, mikä dokumentti on ”virallinen” ja ovatko kaikkien tekemät muutokset siinä mukana.

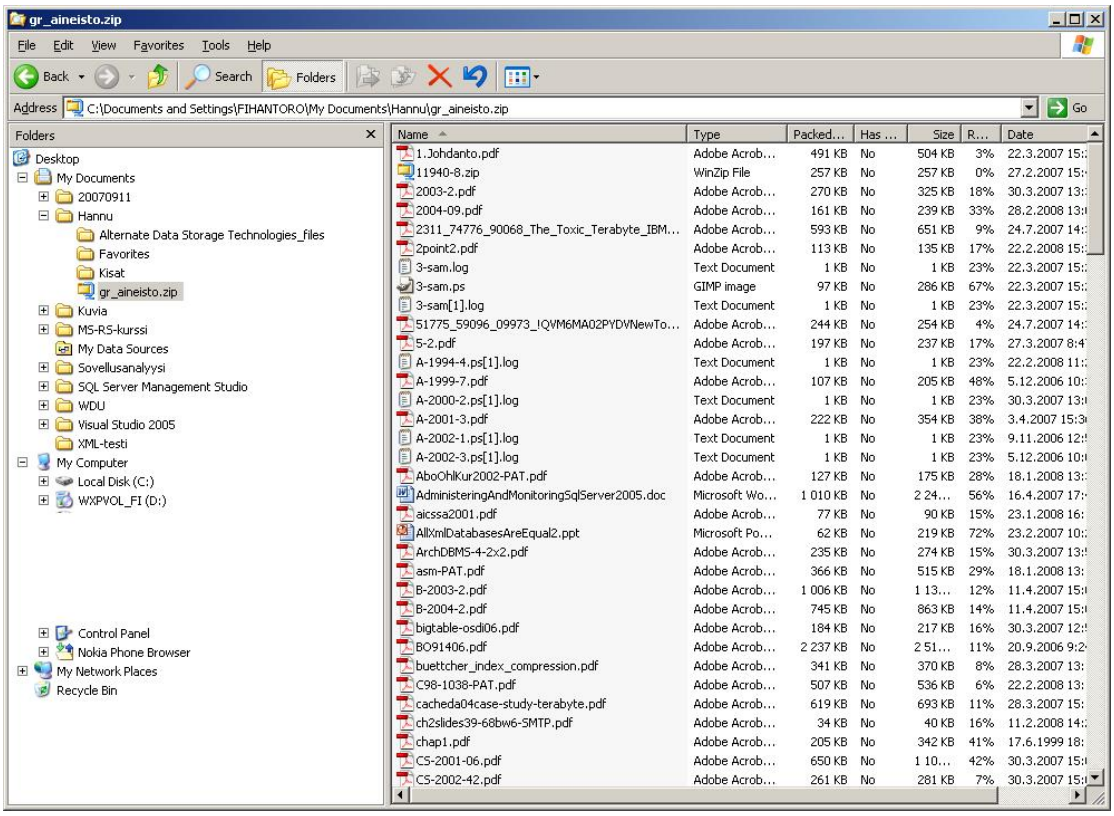
Rakenteettoman tiedon määrää ovat olleet kasvattamassa tekstinkäsittely- ja julkaisuohjelmat, joilla on helppo ja nopea tuottaa dokumentteja ulkoasun ollessa etusijalla. Koska rakennetta ei ole ohjelmallisesti määrätty, käyttäjä joutuu sen itse

[1] Rakenteinen tieto ja rakenteinen dokumentti tarkoittavat eri asiaa: rakenteinen dokumentti edustaa puolirakenteista tietoa kun taas tietokannoissa oleva tieto on rakenteista.

miettimään ja suunnittelu jää toissijaiseksi asiaksi. Rakenteen epäselvän merkkauksen vuoksi myös julkaiseminen eri formaateissa on hankalaa ja vaatii käsityötä. On arvioitu (Virk, 2002), että jopa 50 prosenttia julkaisuprosessista kuluu tekstin muotoiluun. Lisättäessä dokumenttiin esimerkiksi uusi kuva joudutaan loppupään numerointi usein korjaamaan käsin vastaamaan uutta tilannetta. Wordissa on kylläkin mahdollisuus kuvien automaattiseen numerointiin, mutta se on ainakin omasta mielestäni liian hankala käyttää. Lisäksi kuvan pitää olla nimenomaan kuva tai taulukko eikä esimerkiksi itse luotu kuvankaltainen `suorakolmio` tai `tekstikehys`, kuten tämän dokumentin kuva 4. Jos tällaisia on välissä, ei automaattisesta numeroinnista ole hyötyä, ellei niitä ensin tallenna kuvatiedostoina ja lisää sitten dokumenttiin.

IDC:n Yhdysvalloissa tekemän tutkimuksen mukaan tietotyöläinen kuluttaa viikossa 14,5 tuntia sähköpostin lukemiseen ja vastaamiseen, 13,3 tuntia asiakirjojen tekemiseen, 9,6 tuntia tiedon etsimiseen ja 9,5 tuntia tiedon analysointiin (Nikulainen, 2007). Yli tuhannen tietotyöläisen organisaatio menettää vuosittain 5,7 miljoonaa dollaria pelkästään tiedon muuntamiseen siirryttäessä sovelluksesta toiseen. Kun tietoa ei löydetä, maksaa se 5,3 miljoonaa dollaria vuodessa.

Kuvassa 5 on esimerkki hakemistosta, jossa sijaitsevien dokumenttien sisältöä on lähes mahdotonta päätellä niiden nimistä. Tunnistamista voidaan parantaa nimeämisellä (esimerkiksi AdministeringAndMonitoringSqlServer2005.doc), mutta sisältöön liittyviä hakuja voidaan tehdä tiedostomuodosta riippuen rajoitetusti. Nimeen voidaan liittää esimerkiksi viiteluettelossa käytetty tunniste, joka helpottaa dokumentin löytämistä käytettäessä vastaavaa viitettä viitatussa dokumentissa. Tiedosto 2004-09.pdf voidaan nimetä esimerkiksi nimelle 2004-09_Magnani-2004.pdf, jolloin sen löytäminen on huomattavasti helpompaa. Useimmiten dokumentit tallennetaan hajalleen tiedostopalvelimille tai omille työasemille ilman nimeämissääntöjä. Kokemuksieni mukaan voidaan pelkällä ohjeistuksella parantaa dokumenttien käytettävyyttä ja säästää siten työaikaa hyvinkin huomattavasti, kun esimerkiksi harvoin käytettyjä ohjeita ei jokaisen tarvitse tehdä itse eikä keksiä niihin omaa ratkaisua.



Kuva 5: Esimerkki dokumenttihakemiston sisällöstä.

Rakenteisuus auttaa esimerkiksi dokumenttien haussa ja hallinnassa, mutta ohjeistuksen täytyy silti olla kunnossa. Dokumentti voidaan tallettaa rakenteisena dokumenttitietokantaan tai dokumenttien hallintajärjestelmään, mutta käyttäjien on silti osattava nimetä dokumentit oikein, talletettava ne oikeaan hakemistoon, suojattava tarpeen mukaan ja annettava mahdolliset metatiedot kuvaamaan kyseistä dokumenttia. Dokumenttien hallinnan vaikeutta kuvaa hyvin se, että dokumentinhallintajärjestelmien rinnalle on ryhdytty ottamaan käyttöön sisällönhallintajärjestelmiä ja konsernikohtaisia hakukoneita.

2.2 Rakenteisuuden hyödyt

Dokumenttien hallinta käsittää Kar:n (1996) mukaan:

- dokumentin formaalin kuvaamisen,
- dokumentin käsittelyn,
- dokumentin kontrolloinnin.

Formaalin kuvaamisen tarkoituksena on tallentaa dokumentti sellaisessa muodossa, että sen käyttö on tehokasta ja luotettavaa. Dokumentin käsittely sisältää haun,

esittämisen, luonnin ja muokkaamisen. Kontrollointi pitää sisällään jakelun ja turvaamisen. Rakenteisuudella helpotetaan dokumenttien hallintaa, minkä lisäksi xml-muotoinen rakenne mahdollistaa (Virk, 2002):

- sisältöön liittyvät ”älykkäät” kyselyt,
- monikanavajulkaisemisen,
- asiakas-räätälöidyt dokumentit,
- ajan ja rahan säästön,
- dokumenttien ja niiden osien kierrätyksen,
- hajautetun luomisen ja turvallisuuden,
- sisällön jakelun,
- sisällön siirtämisen toiseen järjestelmään.

Rakenteisista dokumenteista voidaan tehdä kolmentyyppisiä kyselyjä (Kay, 2003):

- hakea dokumentteja,
- hakea faktatietoa,
- hakea informaatiota (kerro jotain X:stä).

Haut voivat kohdistua sekä rakenteeseen että sisältöön. Voidaan esimerkiksi hakea tuorein kirje jossa lähettäjänä on Hannu tai faktatietona päivä, jolloin kirje on kirjoitettu. Informaatiota haetaan tyypillisesti webin hakukoneilla useista eri lähteistä haun tulosten riippuessa sekä hakukoneen kyvystä analysoida tietoa että siitä, kuinka hyvin kontekstuaalinen eli yhteydestä ilmenevä informaatio on merkkaussäännöillä toteutettu (sekä hakijan kyvystä antaa hakuehtoja). Voitaisiin hakea esimerkiksi henkilöt, jotka ovat työelämässä ja kirjoittavat työsopimuksesta, kesälomasta tai projektista. Pelkän rakenteen ja sisällön perusteella tehdyt kyselyt eivät itse asiassa ole kovin älykkäitä sillä asioiden välisiä yhteyksiä ei ilman semantiikkaa voida esittää (ks. luku 2.3).

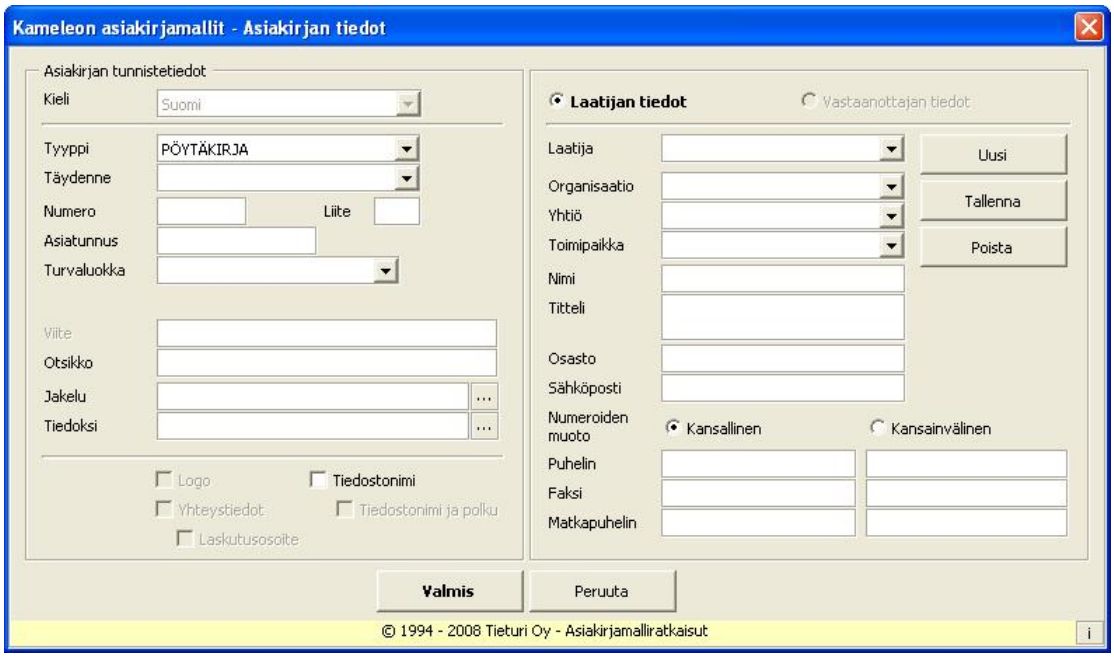
Monikanavajulkaiseminen tarkoittaa, että dokumentti kirjoitetaan kerran, mutta se voidaan julkaista useaan kertaan eri muodoissa kuten pdf:nä ja siitä edelleen paperille, wml:nä (Wireless Markup Language) esimerkiksi puhelimen näytölle tai html:nä julkaistavaksi webbisivuille. Asiakkaalle voidaan räätälöidä hänen haluamassaan muodossa (pdf, xml, doc) vain se osa dokumentista, jota hän tarvitsee.

Toimitusprosessit saneeraamalla säästetään aikaa ja rahaa tarjoamalla valmiit rakenteet käyttäjien huolehtiessa sisällöstä. Dokumentteja voidaan kierrättää käyttämällä niiden osia hyväksi muissa dokumentissa. Editoijalle ja loppukäyttäjälle voidaan antaa käyttöoikeus (luku, muokkaus, lisäys, poisto) tarpeen mukaan ja vain tarpeelliseen osaan dokumenttia. Näin dokumentilla voi olla useita yhtäaikaisia

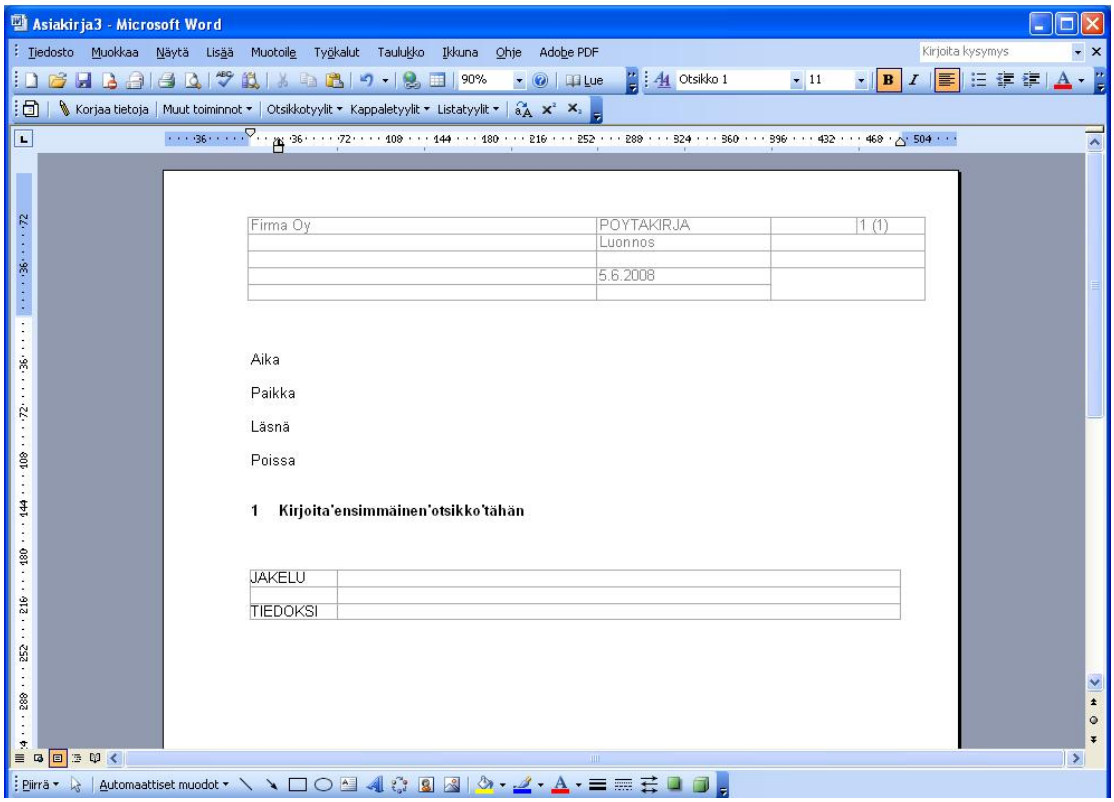
muokkaajia, ja jos jokin osa dokumentista sisältää arkaluontoista asiaa, voidaan pääsy kyseiseen osaan rajata. Xml-muotoisen tiedon sisällön jakelu voidaan hoitaa web-services–palveluna. Näiden lisäksi rakenteisuutta tukevilla järjestelmillä voidaan estää saman tiedon uudelleen tuottaminen. Ellei dokumenttia löydy, se tehdään uudestaan, ja samasta asiasta voi syntyä eri totuuksia, mikä voi olla sekä kallista että vaarallista. Uuden dokumentin tuottamiseen kuluu aikaa, ja vanhassa dokumentissa ollut elintärkeäkin huomio voi jäädä pois.

Edellä lueteltu lista käy yhteenvetona sille, miksi teksti ylipäänsä kannattaa tuottaa rakenteisena. Tallennusrakenteiden kannalta katsottuna xml:ssä on, standardin lisäksi, se etu, että se sopii hyvin rakenteettoman ja puolirakenteisen tiedon tallennusmuodoksi, sillä useimmilla dokumenteilla on puumainen rakenne, joka on rakenteena myös xml:ssä. Täytyy kuitenkin muistaa, että rakenteisuus vain mahdollistaa nämä ominaisuudet ja niiden toteutukseen tarvitaan järjestelmiä, jotka tukevat rakenteisen tekstin käsittelyä ja joissa on *läpinäkyvästi* mukana kyseiset ominaisuudet. Läpinäkyvyydellä tarkoitan sitä, että käyttäjän ei tarvitse itse huolehtia esimerkiksi rakenteen määrittämisestä, vaan hän valitsee rakenteen eli dokumentin mallin ja ohjelma huolehtii, että dokumentti täyttää rakenteen vaatimukset ja tallettaa dokumentin rakenteisena. Käyttäjä voi jopa itse määritellä rakenteen ohjelman huolehtiessa tekstin tuottamisesta rakenteisena ja ohjatessa prosessia niin, että dokumentin osat tulevat rakenteen mukaisessa järjestyksessä.

Asiakirjamallijärjestelmillä on pyritty standardin muotoisten asiakirjojen tuottamiseen. Esimeriksi Kameleon-järjestelmässä käyttäjä valitsee mallin valmiiden dokumenttipohjien joukosta, jonka jälkeen hänen tulee syöttää dokumentin perustiedot (kuva 6). Kun käyttäjä on syöttänyt vähintään pakolliset tiedot, avautuu dokumentti mallin mukaisena (kuva 7).



Kuva 6: Esimerkki Kameleon-järjestelmän aloitussivusta.



Kuva 7: Esimerkki valmiista dokumenttipohjasta.

Perusidea asiakirjamalleissa on hyvä, mutta rakenteen muuttaminen on työlästä eikä sovellu loppukäyttäjälle. Järjestelmät käyttävät yleensä doc-tiedostomuotoa eivätkä tallenna dokumenttia rakenteisena, minkä lisäksi ne ovat hankalia käyttää ja joustamattomia joten käyttö tyrehtyy yhden dokumentin luonnin jälkeen, ja jatkossa otetaan pohjaksi aikaisemmin luotu asiakirja.

Läpinäkyvyyden puute ja joustamattomuus on käsitykseni mukaan tärkein syy, miksi tekstiä tuotetaan edelleen rakenteettomana. Tekstinkäsittelyjärjestelmissä tulisi rakenteen luonnin olla perusominaisuutena niin, ettei käyttäjän tarvitsisi siitä erikseen huolehtia. Myöskään selviä etuja rakenteisuutta tukevien järjestelmien käyttöönotolle ei nähdä, sillä niiden käytöstä ei ole kokemuksia eikä hyötyjä siten osoitettavissa. Tutuissa tekstinkäsittely- ja julkaisuohjelmissa on monipuoliset muotoilu- ja taitto-ominaisuudet, ne ovat helppokäyttöisiä ja lopputulos nähdään heti. Dokumenttien jakelu internetin kautta on helppoa ja nopeaa, samoin pienimuotoinen julkaisutoiminta. Uusien rakenteisuutta tukevien järjestelmien käyttöönotto tarkoittaisi myös kallista uudelleenkoulutusta, joten tekstinkäsittelyjärjestelmien puolella näyttää siltä, että rakenteisuuden hyötyjä ryhdytään ottamaan käyttöön vähitellen uuden toimistotiedostyyppistandardin *ooxml* myötä, kun ei enää hallita kasvavaa dokumenttimassaa ja dokumenteissa oleva tieto halutaan mukaan tiedon analysointiprosessiin esimerkiksi *BI*:n (*Business Intelligence*) myötä.

2.3 Merkityksen esittäminen ja semanttinen web

Dokumenttien tuottaminen rakenteisena ei ratkaise kaikkia tiedonhallinnan ongelmia. Vaikka rakenne olisi merkattukin, asioiden väliset yhteydet eivät käy siitä ilmi, minkä ongelman Tim Berners-Lee toi esille jo 1990-luvun puolivälissä (Junkkaala, 2008). Esimerkiksi kuvassa 8 olevasta xml-kielisestä esityksestä eivät järjestelmät, kuten hakukoneet, ymmärrä mitä tarkoittaa "asiakas", "nimi" tai "katuosoite". Lukijalle asia on selvä, sillä hän osaa yhdistää esimerkiksi katuosoitteen paikkaan, jossa henkilö asuu tai missä yrityksen kotipaikka sijaitsee.

```
<?xml version=”1.0”?>
<asiakas>
    <nimi>Kallen Pultti Oy</nimi>
    <katuosoite>Kauppakatu 100</katuosoite>
    <pnro>80100</pnro</>
    <ptp>Joensuu</ptp>
</asiakas>
```

Kuva 8: Esimerkki XML-kielisestä tekstistä.

Ohjelma ei myöskään ymmärrä *sisäkkäisrakenteita*, sillä niille ei ole olemassa yleistä tulkintatapaa. Ohjelma ymmärtää vain, että kyseessä on puurakenne. Esimerkiksi kuvan 9 tilausrivit tarkoittavat samaa asiaa, mutta järjestelmät eivät sitä ymmärrä.

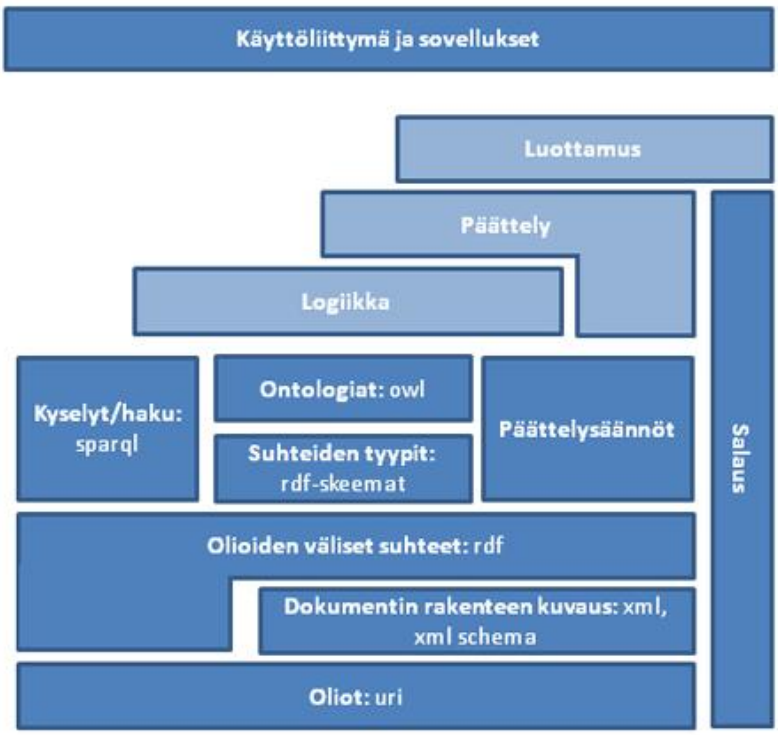
```
<Tilaus = ”Ruuvi 4*20”>
          <Asiakas>   Kallen   Pultti   Oy
</Asiakas>
</Tilaus>

<Asiakas = ”Kallen Pultti Oy”
          <Tilaus> Ruuvi 4*20 </Tilaus>
</Asiakas>
```

Kuva 9: Esimerkki XML:n sisäkkäisrakenteesta.

Kolmas ongelma on, että rakenteella ei pystytä kuvaamaan sanojen eroa eli ohjelma ei esimerkiksi ymmärrä, mikä ero on kalastukseen käytetyllä verkolla tai datayhteyksissä käytetyllä verkolla.

Rakenteen lisäksi tarvitaan siis keinot *semantiikan* eli merkityksen määrittelyyn. Junkkaala esittää artikkelissaan (2008) internetin yhteydessä käytettävän *semanttisen webin* perusidean (kuva 10). Semanttinen web rakentuu kerroksista alhaalta ylöspäin siten, että kerroksia voidaan käyttää vaikkeivät kaikki niistä ole vielä valmiina. Kerrokset ovat W3C:n[2] (World Wide Web Consortium) standardoimia tekniikoita, joista kuvan 10 vaaleat kerrokset ovat vielä vakiintumatta.

[2] Riippumaton web-standardien julkaisemiseen keskittyvä foorumi.



Kuva 10: Semanttisen webin tekniikat (Junkkaala, 2008).

Semanttinen web koostuu olioista, jotka yksilöidään *uri-tunnisteella* (*uniform resource identifier*). Olio voi olla esimerkiksi dokumentissa esiintyvä käsite tai kuva. Kuvaukset perustuvat sanojen sijasta käsitteisiin, jolloin voidaan erottaa samalla tavalla kirjoitetut, mutta erimerkityksiset sanat toisistaan. Dokumentin rakenne kuvataan *xml-skeemoilla* ja osat merkitään xml-kielellä. Olioiden väliset suhteet kuvataan xml:ään perustuvalla *rdf-tietomallilla* (*resource description framework*), joka tarjoaa yleiset säännöt suhteiden kuvaamiseen. Suhteiden tarkemmat tyypit kuvataan *rdf-skeemoilla*, joilla luodaan sanastoja resurssien välisten suhteiden ilmaisemiseen.

Tarkempien aihekohtaisten sanastojen määrittelyyn käytetään *owl-kieltä* (*web ontology language*), joka käyttää sekä uri-tunnisteita että rdf:n teknisiä puitteita tarjoten lisää tapoja ominaisuuksien ja luokitusten kuvailemiseen. Esimerkiksi suomalainen ontologiasivusto löytyy kansalliskirjaton sivuilta http://www.seco.tkk.fi/ontologies/yso/.

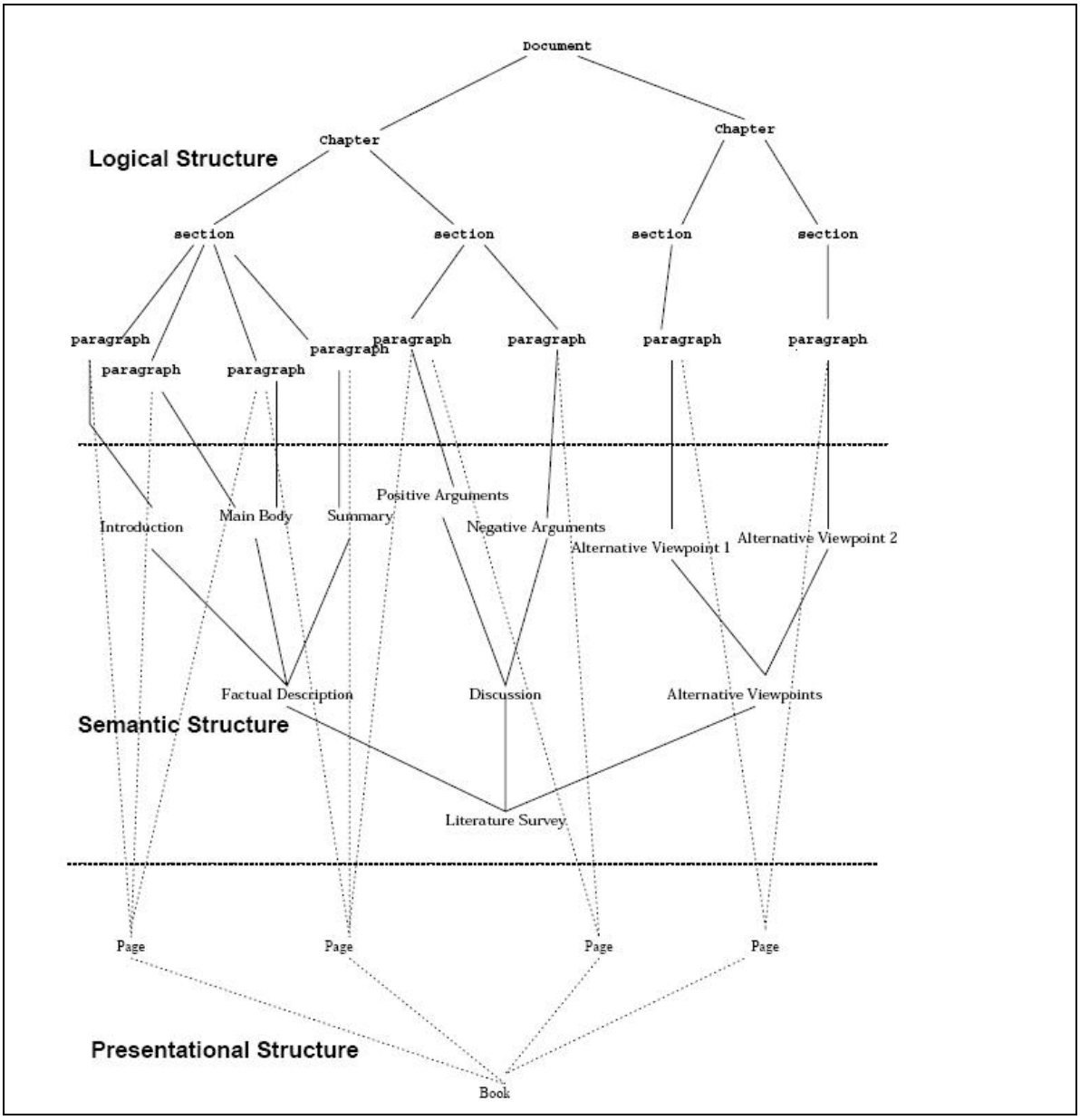
3 Dokumentin rakenteet

Rakenteisella dokumentilla on toisiinsa liittyviä rakenteita, jotka kuvaavat dokumentteja käsitteellisesti eri tasoilla. Lukija näkee dokumentin toisaalta hänelle merkityksellisinä lukuina, alalukuina ja kappaleina, mutta myös sivuina juoksevine numerointeineen. Jos sisäisenä rakenteena käytetään esimerkiksi puurakennetta, on siitä pystyttävä muodostamaan käyttäjälle näkyvä lukuihin, alalukuihin ja kappaleisiin perustuva esitysmuoto. Tässä luvussa esitetään dokumentin rakenteen kolme päätyyppiä ja selvitetään, mistä osista dokumentti koostuu sekä mistä muodostuu sisäinen ja ulkoinen tallennusrakenne.

3.1 Rakennetyypit

Salton & al. (1992) erottavat toisistaan kaksi dokumenttien toisiinsa liittyvää rakennetta: *abstraktin* rakenteen ja *semanttisen* rakenteen. Abstrakti rakenne määrittelee, kuinka tekstin osat sijoittuvat toisiinsa nähden, esimerkiksi kuinka teksti on jaettu lukuihin, kappaleisiin ja lauseisiin ja kuinka näistä osista kootaan koko teksti. Semanttinen rakenne esittää ajatukset, joita dokumentissa halutaan esittää siten, että koko dokumentin tarkoitus tulee esille yksittäisten osien kautta. Se mallintaa dokumentin korkeammalla ja käsitteellisemmällä tasolla kuin abstrakti rakenne ja on vaikeampi määritellä (Kar, 1996).

Abstrakti rakenne jakaantuu edelleen *loogiseen* rakenteeseen ja *ulkoasuun* eli *esitysmuodolliseen* rakenteeseen. Looginen rakenne kiinnittää dokumentin syntaktisen järjestyksen loogisella tavalla esimerkiksi lukuihin, osiin ja kappaleisiin kuten kirjoittaja on sen ajatellut lukijalle esittää. Esitysmuodollinen rakenne kuvaa, kuinka dokumentti muotoillaan ja esitetään esimerkiksi sivuina, riveinä ja sarakkeina. Kuvassa 11 on esimerkki teknisestä dokumentista kuvattuna rakennetyypeittäin ja niiden suhteista toisiinsa. Tallennusrakenteiden kannalta keskeinen on looginen rakenne, joka on dokumentin jäsennys tietyn kieliopin suhteen ja jakaa dokumentin loogisiin osiin.



Kuva 11: Esimerkki dokumentin rakenteen kolmesta päätyypistä ja niiden suhteista toisiinsa (Kar, 1996).

3.2 Dokumentin osat ja esitysmuodot

Rakenteen lisäksi dokumentti koostuu *sisällöstä* ja *muotoilusäännöistä* eli rakenteisella dokumentilla on kolme osaa:

- *sisältö* (mitä tietoa se on),
- *rakenne* (missä tieto on),
- *ulkoasu* (miltä tieto näyttää).

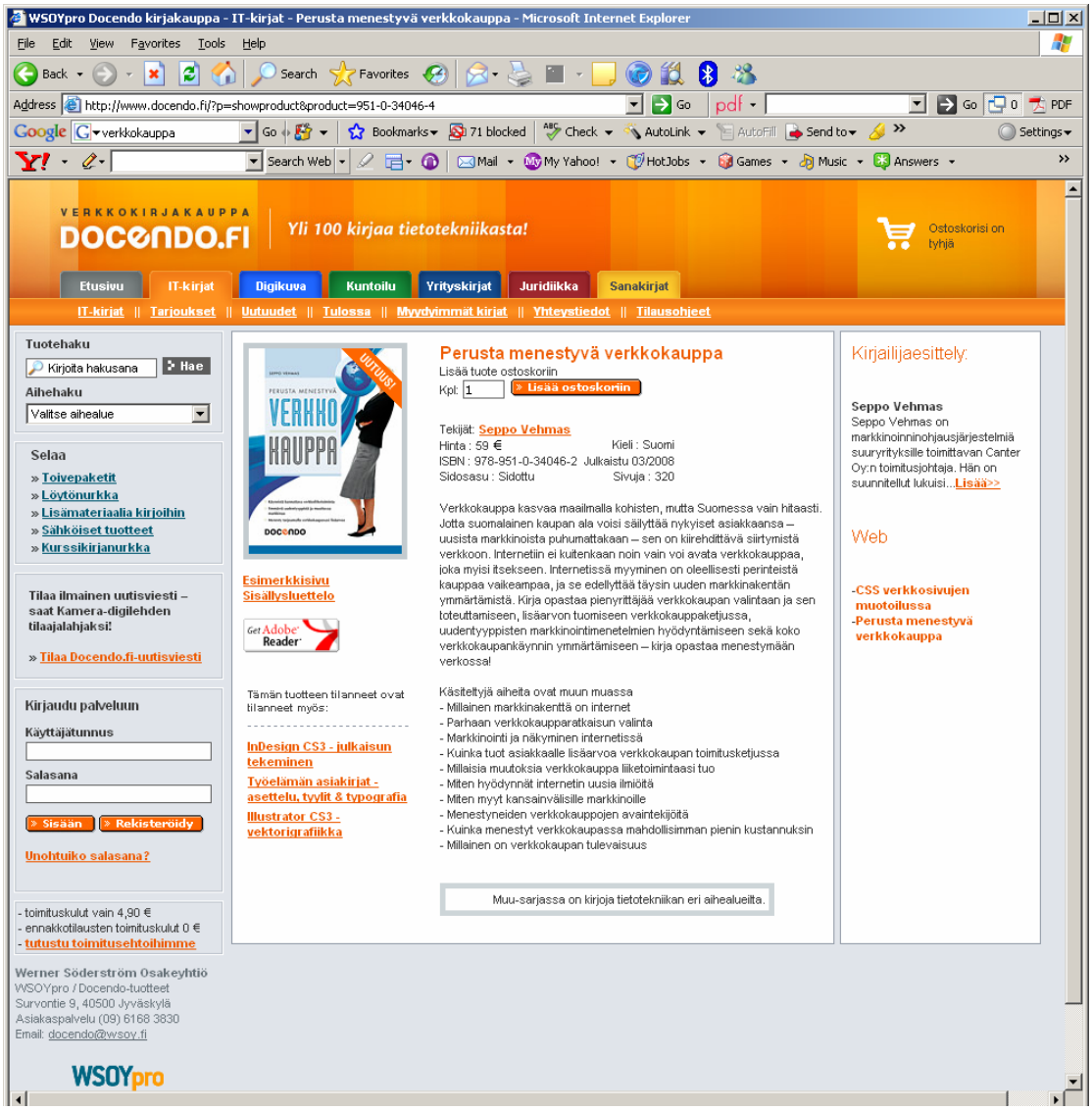
Esimerkiksi kuvan 2 kirjeellä on rakenteen osa `päiväys`, jonka sisältö tarkoittaa päivää, jolloin kirje on kirjoitettu. Päiväys muodostuu päivämäärästä, mutta se voitaisiin jakaa esimerkiksi päivä-, kuukausi- ja vuosikomponentteihin. Lisäksi sillä voi olla useita ulkoasukomponentteja, jotka liittyvät muihin komponentteihin. Päiväyksen ulkoasu voi olla esimerkiksi 3.4.2008 tai ***03042008,*** ja se voi olla riippuvainen käytettävästä kielestä, jolloin amerikkalaistyylinen esitysmuoto voikin olla 2008-03-04, eli päivä on vuoden jälkeen. Rakenteisen tekstin käsittelyn kannalta oleellista on ulkoasun ja muotoilusääntöjen erottaminen rakenteesta, jolloin esimerkiksi dokumentin jakelu eri formaateissa on mahdollista. Myös kappaleet voitaisiin päiväyksen tavoin jakaa edelleen lauseisiin, jolloin dokumentin rakenne saadaan hienorakeisemmaksi ja haku sekä käsittely voidaan tehdä lausetasolla.

Rakenteisella dokumentilla on *esitysmuotoja*, jotka vaihtelevat dokumentin käsittelyn eri vaiheissa (Kuikka & Salminen, 1995; Glushko & McGrath, 2002). Dokumentti voi olla päätelaitteella kuten näytöllä tai puhelimen ruudulla katseltavana tai se voi olla paperille tulostettuna. Dokumentti voi olla tallennettuna levylle tai siirrettävänä toiseen ympäristöön esimerkiksi xml:ää käyttäen. Näiden ulkoisten esitysmuotojen lisäksi dokumentti voi olla käsiteltävänä ohjelman sisäisen esitysmuodon, esimerkiksi *puurakenteen*, mukaisesti. Esitysmuodolla ei siis tarkoiteta pelkästään dokumentin käyttäjälle näkyvää muotoa. Elektronisessa muodossa levylle tallennettu xml-tiedosto on tallennettu sinne *ulkoisen tallennusrakenteen* eli xml-kieliopin mukaisesti. Kun se luetaan ohjelmaan sisäisen tallennusrakenteen mukaisesti, dokumentin esitysmuoto muuttuu xml-muodosta puumalliseksi ja tuloksena on dokumentin looginen jäsennyspuu.

Kun dokumentti siirretään esitysmuodosta toiseen, jokin tai kaikki sen osat (sisältö, rakenne, ulkoasu) voivat muuttua. Osan merkitys riippuu siitä, onko kyseessä *dokumentti*- vai *datakeskeinen* tieto. Dokumenttikeskeisessä mallintamisessa, jota edustavat esimerkiksi käyttöoppaat, esitteet, kirjat ja webbisivut, tiedolla on tyypillisesti hierarkkinen rakenne ja ulkoasulla on suurempi merkitys kuin datakeskeisessä mallintamisessa. Datakeskeisiä dokumentteja, kuten ostotilauksia, käytetään järjestelmien välisessä tiedonsiirrossa, ja niissä looginen rakenne on erittäin tärkeää, jotta tieto siirtyisi oikein (Glushko & McGrath, 2002).

3.3 Monimuotoiset dokumentit

Dokumenttien erilaiset käyttötarpeet asettavat haasteensa myös järjestelmien suunnittelulle ja käytölle. Dokumentti ei useinkaan ole joko data- tai dokumenttikeskeinen vaan sen välimuoto. Esimerkiksi nettikaupan tuotekatalogeissa on sekä rakenteella että ulkoasulla tärkeä merkitys. Esimerkiksi kuvassa 12 dokumentin ulkoasun eli käyttäjälle näytettävän webbisivun täytyy olla houkutteleva, jotta ostaja innostuisi siitä. Dokumentin rakenteen täytyy olla kuitenkin sellainen, että sen osat ovat tunnistettavissa ja yhdistettävissä kyseiseen tuotteeseen ja tilaajaan.



Kuva 12: Esimerkki verkkokaupan tuotesivusta.

Staattisen html-sivun tapauksessa riittäisi, että dokumentti avataan selaimella palvelimelta, mutta esimerkin kaltaisen dynaamisen dokumentin sisältö voi koostua useista eri lähteistä, kuten sql- tai xquery-kyselyillä tietokannoista haetuista riveistä, muista järjestelmistä tulleista xml-dokumenteista ja jopa vapaamuotoista tekstiä sisältävistä tiedostoista. Teoksia voidaan hakea hakusanojen perusteella, joten teosten perustiedot sijaitsevat esimerkiksi relaatiokannassa, josta haut ovat tehokkaita ja jonne teosten tietojen päivittäminen on joustavaa. Kun käyttäjä hyväksyy tilauksen, voidaan varmistus ja maksu tehdä verkkopankin kautta tai tilaajalle laitetaan erillinen lasku laskutusjärjestelmästä, jonne tilauksen tiedot lähetetään esimerkiksi xml-tiedostona. Tilaus voidaan lisäksi lähettää xml-muotoisena tilausrivinä varastonhallinta- ja logistiikkajärjestelmään. Esimerkiksi relaatiokannasta haetun datan konvertoiminen hierarkkiseksi xml-tiedostoksi tai päinvastoin ei kuitenkaan ole aina triviaali tehtävä. Luvussa 5 tarkastellaan tapoja ulkoisen xml-tiedoston tallentamiseksi relaatio- tai xml-tietokantaan, mutta varsinaisia muunnostekniikoita emme tässä tutkielmassa tarkastele.

3.4 Ulkoinen ja sisäinen tallennusrakenne

Joissain yhteyksissä rakenteisen tekstin *sisäisellä esitysmuodolla* tarkoitetaan tietokoneen käsiteltäväksi tarkoitettua esitysmuotoa ja *ulkoisella esitysmuodolla* ihmisaisteille tarkoitettua esitysmuotoa (Kuikka & Salminen, 1995). Xml-dokumenttien yhteydessä puhutaan dokumentin *loogisesta* ja *fyysisestä* rakenteesta (Salminen & Tompa, 2001). Looginen rakenne on dokumentin jäsennys tietyn kieliopin suhteen, jolloin tuloksena on dokumentin *jäsennyspuu*. Fyysinen rakenne on kokoelma (loogisen) dokumentin taustalla vallitsevia tiedostoja, tietorakenteita ja muita entiteettejä tietokoneen muistissa.

Tässä tutkielmassa *ulkoisella tallennusrakenteella* tarkoitetaan tallennusmuotoa, jossa dokumentti on tallennettu ulkoiseen muistiin esimerkiksi kovalevylle jatkokäsittelyä, arkistointia, tiedoston siirtoa tai muuta vastaava tarkoitusta varten. *Sisäisellä tallennusrakenteella* tarkoitetaan tietokoneen keskusmuistissa olevaa rakennetta silloin kun dokumentti on käsiteltävänä.

Ulkoisen ja sisäisen tallennusrakenteen ero ei ole aina aivan selvä. Dokumentti voidaan tallettaa esimerkiksi relaatiotietokannan tauluihin relaatioina. Fyysisesti tietokannan tiedot tallennetaan binaarimuotoisina tietokantatiedostoina levylle, jossa ne on tietokannan oman tallennusmallin mukaisesti järjestetty eikä malli ota tekstin rakenteisuuteen mitään kantaa. Fyysisiä tietokantatiedostoja ei voida myöskään käyttää tietojen siirtoon järjestelmien välillä, eivätkä ne ole, binaarimuodosta johtuen, soveltuvia arkistointiin, sillä niiden sisältö on sidottu kyseiseen tietokantaratkaisuun ja versioon. Tässä tapauksessa ulkoinen tallennusrakenne olisi esimerkiksi xml-muotoinen tekstitiedosto, jonka kuvaustiedoston (*dtd*, xml-skeema) mukaan luodaan relaatiokannan taulut ja tallennetaan dokumentti kantaan sisäisen tallennusrakenteen mukaisesti. Vastaavasti voidaan relaatiotaulujen sisällöstä muodostaa ulkoisen tallennusrakenteen mukainen xml-tiedosto. Tietokannassa olevaa dokumenttia voidaan käsitellä myös esimerkiksi tekstinkäsittelyohjelmalla, joka muodostaa relaatiomallin tauluihin tallennetuista dokumentin osista sisäisen, esimerkiksi puumallisen, rakenteen. Ohjelman näkökulmasta katsoen on puurakenne tällöin sen sisäinen tallennusrakenne ja relaatiokantaan tallennettu vastaava dokumentti ulkoinen tallennusrakenne.

Tässä yhteydessä on myös huomioitava, että ulkoisesti xml-muodossa esitetty dokumentti voidaan lukea ja tallentaa sisäisesti useilla eri tavoilla kuten relaatiotauluina, puurakenteena, xml:nä ja niin edelleen. Ulkoinen rakenne ei siis välttämättä sido käyttämään tiettyä sisäistä rakennetta tai päinvastoin.

3.5 Rakenteisen tekstin mallintaminen

Analysoimme reaalimaailman olioita, tapahtumia ja niiden suhteita luomalla malleja. Tietomalli koostuu tietotyypeistä, operaatioista ja eheyssäännöistä eli rajoituksista, ja sen tarkoituksena on esittää formalismi sille, miten reaalimaailman oliot, tapahtumat ja niiden suhteet esitetään, eli se kuvailee tietokannan rakenteen. Perinteiset tietokantamallit kuvaavat asioita suoraan, mutta tekstin kuvaamisessa käytettävien mallien tulee kuvata itse tekstiä (Salminen & Tompa, 1999). Esimerkiksi relaatiotaulun `asiakas`-tietue kuvassa 13 sisältää pelkästään datan, kun kuvan 8 xml-esimerkki sisältää sekä rakennekuvauksen että datan. Jälkimmäinen mahdollistaa haut sekä rakenneosien että sisällön perusteella.

Asiakas-tunnus	Nimi	Katuosoite	Pnro	Ptpt
12345	Kallen Pultti Oy	Kauppakatu 100	80100	Joensuu

Kuva 13: Esimerkki relaatiomallin asiakastietueesta.

Yksinkertaisin malli dokumentin esittämiseksi on *merkkipohjainen* malli, jossa teksti esitetään yhtenä pitkänä merkkijonona muotoilusääntöjen ollessa tekstin seassa (Kuikka, 1990). Näissä malleissa on dokumentin käsittely lineaarista toimenpiteiden kohdistuessa yksittäisiin merkkeihin tai merkkijonoihin eivätkä ne mahdollista rakenteen käyttöä missään editoinnin, tallennuksen tai siirron vaiheessa. Mallia käytetään tavallisissa tekstieditoreissa, ja sen etuna on, että tallennettaessa dokumentti esimerkiksi *ascii*-muodossa, on se jatkokäsiteltävissä lähes millä järjestelmällä tahansa, minkä lisäksi dokumentin sisällöstä voidaan tehdä *vapaasanahakuja*.

Lineaarimallissa dokumentti koostuu peräkkäisistä objekteista, jotka taas muodostuvat jonosta merkkejä. Toimenpiteet voivat kohdistua yksittäisten merkkien lisäksi nimettyihin objekteihin (kappaleisiin). *Pseudo-hierarkkisessa* lineaarimallissa objektit nimetään siten, että nimi määrittää luokan, johon objekti kuuluu (André & al., 1989). Mallia kutsutaan pseudo-hierarkkiseksi, koska kahden samannimisen objektin vierekkäisyys tarkoittaa, että ne kuuluvat yhteen. Mallia käytetään wysiwyg-editoreissa sen yksinkertaisen rakenteen vuoksi. Lineaarimalliin voidaan lisätä hierarkkisuutta esittämällä dokumentti sisäkkäisten objektien ja elementin sekvensseinä.

André & al. (1989) esittelevät teoksessaan ”Structured Documents” useita versioita hierarkkisista puumalleista. Niitä ei käydä tässä kuitenkaan läpi, sillä ne ovat rakenteisen tekstin hierarkkisuuden takia, ainakin käsitteellisellä tasolla, useimpien jatkossa esitettävien mallien ja tallennusratkaisujen pohjana.

Rakenteista tekstiä on mallinnettu perinteisten *hierarkkisen mallin, verkkomallin* ja *relaatiomallin* lisäksi *oliomallilla.* Näiden pohjalta on kehitetty lukuisia muita malleja kuten *olio-relaatiomalli,* jossa relaatiomalliin on lisätty hierarkkinen *oliopuu* (Bourret, 2005). Xml-kielen myötä on käyttöönotettu myös käsitteet *xml-tietomalli* ja

puolirakenteinen (semistructured) malli. Erottimin varustettu tiedosto levyllä edustaa puolirakenteista mallia ja ulkoista tallennusrakennetta. Kun tiedosto jäsennetään esimerkiksi xml–tietokannassa xml-tietomallien (*infoset*, *dom*, *xdm)* mukaisesti, on kyseessä sisäinen tallennusrakenne. Asiaa voidaan tarkastella myös niin, että sisäisen tallennusrakenteen mallit on tarkoitettu ulkoisen tallennusrakenteen mukaisesti tallennettujen, esimerkiksi xml-muotoisten, tiedostojen hallintaan.

4 Ulkoinen tallennusrakenne

Ulkoisessa muistissa olevat dokumentit ovat kaikki pohjimmiltaan lineaarisia tekstitiedostoja. Vaikka dokumentin rakenne olisi tekstin yhteyteen merkitty, ei sillä ole merkitystä, ellei lukeva ohjelma pysty sitä tulkitsemaan. Datakeskeisen tiedon tapauksessa voidaan tallennus tehdä yksinkertaisimmillaan niin sanottuna rivitason tallennuksena, jolloin kentät on erotettu toisistaan sovitulla erotinmerkillä, kuten tabulaattorilla tai puolipisteellä, tai ne ovat sovitun mittaisia kuten pankkisiirtoaineisto kuvassa 4. Tiedostoa käsittelevän ohjelman täytyy tietää kenttien järjestys pystyäkseen tulkitsemaan ne oikein. Ensimmäisellä, niin sanotulla otsikkorivillä, voidaan toki kertoa kenttien nimet, jolloin ohjelma voi tulkita siitä tiedon osat. Rakenteisen tekstin hierarkkisuutta ei tällä tavoin kuitenkin voida kuvata, ja sen käyttö soveltuu lähinnä tiedonsiirtoon esimerkiksi relaatiotietokantojen välillä. Erotinmerkki voi esiintyä myös varsinaisen datan seassa, jolloin rivin tulkinta menee väärin. Kuvassa 14 on esimerkki erotinmerkein varustetusta tiedostosta, jossa on kaksi tilausriviä.

```
Tilaus;Nimi;Katuosite;Pnro;Ptp;Tuote;Maara
100;Kallen Pultti Oy;Kauppakatu 100;80100;Joensuu;Ruuvi 4*20;10000
100;Kallen Pultti Oy;Kauppakatu 100;80100;Joensuu; Mutteri 5*30;15000
```

Kuva 14: Erotinmerkein varustettu rivitason tiedosto.

Jatkossa oletamme, että rakenteisella tekstillä on tunnistettavissa oleva looginen rakenne (Brown & al., 1997). Rakenne voidaan tunnistaa esimerkiksi erottimien avulla tai kuten xml-kielessä, erillisen *dtd-kuvaustiedoston (Data Type Definition)*, *xml-kaavion (XML Schema)* avulla tai kuten ohjelmointikielten yhteydessä, jäsentämällä teksti kieliopin mukaan.

4.1 Formaalit kieliopit rakenteisen tekstin määrittelyssä

Rakenteisen tekstin määrittelyyn voidaan käyttää ohjelmointikielten syntaksin kuvaamisesta tuttuja *formaaleja kielioppeja* (Salminen & Watters, 1992; Kuikka & al., 1999). Yhteysvapaiden kielioppien laajennetussa *Backus-Naur* (bnf) -esitysmuodossa voidaan säännön oikealle puolelle kirjoittaa toistoa ilmaisevia metasymboleja ’*’ (0 kertaa tai useammin), ’+’ (ainakin kerran) ja vaihtoehtoisuutta

’|’. Säännön vasen ja oikea puoli erotetaan merkinnällä ’->’ (tai ’::==’), vapaaehtoisuus hakasuluilla ’[]’ ja sulkuja '()' käytetään ryhmittelyyn. Esimerkiksi kuvan 3 kirje voitaisiin kuvata bnf-muotoisella kieliopilla kuvan 15 mukaisesti.

```
kirje -> yläviite otsikko sisältö lähettäjä
yläviite -> paikka päiväys
otsikko -> merkki{..50}+
sisältö -> kappale+
kappale -> merkki+
lähettäjä -> merkki{..25}+
paikka -> merkki{..25}+
päiväys -> numero{2}+ välimerkki numero{2}+ välimerkki numero{2}+
merkki -> numero | kirjain | erikoismerkki
numero -> ’0’|’1’|’2’|’3’|’4’|’5’|’6’|’7’|’8’|’9’
kirjain -> ’a’ | ’b’ | ’c’ | … | ’Y’ | ’Ä’ | ’Ö’
erikoismerkki -> ’.’ | ’,’ | ’;’ | …
```

Kuva 15: Esimerkkikirjeen BNF-muotoinen kielioppi.

Kieliopin perusteella voidaan teksti jäsentää ja muodostaa siitä sisäisen tallennusrakenteen mukainen jäsennyspuu, kuten ohjelmointikielten jäsentäjät tekevät jäsentäessään ohjelmakoodia. Jäsennettäessä teksti jaetaan rakenne-elementtien mukaisiin osiin (Young-Lai, 2001).

4.2 Rakenteisen tekstin standardit

Rakenteisen tekstin standardoinnilla on pyritty sekä oikean tiedon välitykseen että tekstin käsittelyn automatisointiin (Salminen, 1992). Dokumenttiin liittyy tällöin kieliopin kaltainen määrittely. Esimerkiksi smtp–muotoisen (Simple Mail Transfer Protocol) sähköpostisanoman muoto on kuvattu standardissa RFC 821 (Postel, 1982), joka on kehitetty vain tätä yhtä sovellusaluetta varten. Hieman yleisempi standardi on sähköisen tiedonsiirron kansainvälinen edifact–standardi (Electronic Data Interchange For Administration, Commerce and Transpor). Se on organisaatioiden välisessä tiedonsiirrossa (edi – Electronical Data Interchange) laajasti käytetty esitystavan standardi (Salminen, 1992).

Oda–standardi (*Open Document Architecture*) on tarkoitettu dokumenttien vaihtoon sovellusten kesken (Ditch, 2007). Standardissa esitetään dokumentin sisältö ja sen muoto kahdessa erillisessä rakenteessa (Kuikka, 1990). Sitä ei ole kuitenkaan otettu

laajempaan käyttöön johtuen standardin kehitystyön hitaudesta ja tukijoiden puutteesta. Sen on myös kritisoitu olevan liian rajoittunut ja liian sovelluskeskeinen (Raymond & al., 1995).

Edellä kuvatut standardit on tehty tiettyä sovellusaluetta varten. Jotta mikä tahansa dokumentti voidaan tallentaa rakenteisena siten, että rakenne on kaikkien tulkittavissa, tarvitaan rakenteen kuvaamiseksi yleinen standardi. Sgml ja siitä kehitetty xml on tehty tähän tarpeeseen (Ditch, 2007). Xml on *metakieli*, eli sillä määritellään standardin mukaisesti sovellusalueen bnf-muotoinen kielioppi.

Goldfarbin, joka on yksi sgml:n kehittäjistä, mukaan *yleisen merkintäsysteemin* määritelmä on (Kuikka & Nikunen, 1994):

1. Merkinnän (markup) tulee kuvata tekstin rakennetta eikä prosessia, joka tekstiin kohdistetaan myöhemmin ja joka täytyy tehdä vain kerran.
2. Merkinnän tulee olla niin huolellista ja tarkkaa, että tekstiä pystytään myöhemmin käsittelemään.

Tässä tulee ero Wordin kaltaisiin proseduraalisiin tekstinkäsittelyohjelmiin, joissa kerrotaan, kuinka tekstiä pitää käsitellä. Tekstin yhteyteen tallennettavilla merkinnöillä kerrotaan esimerkiksi, mistä lihavointi alkaa ja missä se loppuu.

4.3 Merkkauskieliopit ja XML

Xml:n kaltaisissa merkkauskielissä käytetään niin sanottua deklaratiivista merkkausta, joka kuvaa dokumentin loogisen hierarkkisen rakenteen. Sen avulla pystytään dokumentin osat tunnistamaan ohjelmallisesti tai niiden rakenne tai muoto pystytään muodostamaan automaattisesti (Kuikka & al., 1999). Myös ulkoasu erotetaan rakenteesta, mikä tekee dokumentista jaettavamman. Esimerkiksi kuvan 14 tilaus voidaan tallentaa xml-muotoisena kuvan 16 mukaisesti.

```
<?xml version="1.0"?>
<tilaus tunnus=”100”>
    <asiakas>
        <nimi> Kallen Pultti Oy </nimi>
        <katuosoite> Kauppakatu 100 </katuosoite>
        <pnro> 80100 </pnro>
        <ptp> Joensuu </ptp>
    </asiakas>
    <tilausrivi>
        <tuote> Ruuvi 4*20 </tuote>
        <maara> 10000 </maara>
    </tilausrivi>
    <tilausrivi>
        <tuote> Mutteri 5*30 </tuote>
        <maara> 15000 </maara>
    </tilausrivi>
</tilaus>
```

Kuva 16: Esimerkki XML-muotoisesta tilauksesta.

Dokumenttikeskeisen tiedon tapauksessa voi tekstin rakenne vaihdella huomattavasti. Esimerkiksi kuvan 17 esimerkissä voi, esityksestä riippuen, lukuja ja kappaleita olla vaihteleva määrä, tekijöitä voi olla useita ja niin edelleen.

```
<?xml version="1.0"?>
<esitys nro = ”1”>
  <otsikko> Rakenteisen tekstin tallennusrakenteet </otsikko>
  <tekijä> Hannu Toroskainen </tekijä>
  <sisältö>
      <luku>
            <tunnus> 1 </tunnus>
            <otsikko> Johdanto </otsikko>
            <teksti> Tässä luvussa selvitetään … </teksti>
            <alaluku>
              <alatunnus> 1.1 </alatunnus>
              <otsikko> Rakenteinen teksti </otsikko>
              <teksti> Teksti on luonnollista kieltä … </teksti>
            </alaluku>
            <alaluku>
              <alatunnus> 1.2 </alatunnus>
              <otsikko> Historiallista taustaa </otsikko>
              <teksti> Henkilökohtaisten tietokoneiden (PC) alkuaikoina …</teksti>
            </alaluku>
            …
      </luku>
      <luku>
            <tunnus> 2 </tunnus>
            <otsikko> Rakenteisen tekstin käyttö</otsikko>
            <teksti> IDC:n (International Data Corporationin) mukaan … </teksti>
            …
      </luku>
            …
  </sisältö>
</esitys>
```

Kuva 17: Esimerkki dokumenttikeskeisestä XML-tiedostosta.

Koska xml on *itsensäkuvaavaa*, voitaisiin kuvan 17 dokumentin rakenne tulkita suoraan. Ilman erillistä kuvausta eli kielioppimäärittelyä ei voida kuitenkaan päätellä, millaista vaihtelua kyseistä tyyppiä edustavat dokumentit voivat sisältää. Tekijöitä voi olla esimerkiksi useita, kappaleiden määrä vaihdella, kenttä voi olla numeerinen ja niin edelleen. Kuvauksen perusteella voidaan lisäksi tarkistaa dokumentin *validisuus* eli oikeellisuus. Esimerkiksi kuvan 17 dokumenttia vastaava dtd-määrittely on kuvan 18 mukainen.

```
<!DOCTYPE esitys [
<!ELEMENT esitys (otsikko, tekijä, sisältö)>
<!ATTLIST esitys nro ID #REQUIRED>
<!ELEMENT sisältö (luku+)>
<!ELEMENT tekijä (#PCDATA)>
<!ELEMENT luku (tunnus, otsikko, teksti, alaluku+)>
<!ELEMENT tunnus (#PCDATA)>
<!ELEMENT otsikko (#PCDATA)>
<!ELEMENT teksti (#PCDATA)>
<!ELEMENT alaluku (alatunnus,otsikko, teksti)>
<!ELEMENT alatunnus (#PCDATA)>
]>
```

Kuva 18: Dtd-määrittely kuvan 17 dokumentille.

Rakenteen kuvaus kertoo, millainen tietyntyyppinen rakenteiden dokumentti voi olla, mitkä rakenneosat ovat sallittuja ja missä järjestyksessä ne voivat esiintyä (Kuikka & Nikunen, 1994). Kuvan 18 esimerkki kertoo, että `esitys` koostuu `otsikosta`, `tekijästä` ja `sisällöstä`. Sisältö koostuu `luvuista`, joita voi olla yksi tai enemmän. Luvut jakautuvat edelleen pienempiin osiin. Lisäksi dokumentissa on oltava esityksen yksilöivä attribuutti `nro`. Kun tunnetaan dokumentin rakenne, voidaan ohjelmallisesti tarkistaa dokumentin oikeellisuus jäsentämällä se rakennekuvauksen mukaisesti. Näin saadaan ulkoisesta tallennusrakenteesta muodostettua ohjelman käyttämä sisäinen tallennusrakenne. Jäsentämisellä tarkoitetaan dokumentin osien tunnistamista ja useimmiten myös dokumentin muuttamista (*transformation*) esitysmuodosta toiseen. Kuten kuvista 16 ja 17 nähdään, soveltuu xml hyvin sekä data- että dokumenttikeskeisen tiedon ulkoiseksi tallennusrakenteeksi. Lisätietoja xml-kielestä löytyy W3C:n sivuilta http://www.w3.org/XML/.

Xml-kielessä määritellään kieliopin lisäksi sovelluskohtaisesti myös *merkkaussanasto*, jolloin tunnisteita voidaan käyttää dokumenttien kuvaamiseen kuten `<Asiakas>` kuvassa 16. Tunnisteet eli erottimet voidaan jakaa niiden käyttötavan mukaan kahteen kategoriaan (Yoshikawa & al., 1999):

- `rakennemerkkaus`,
- `sisältömerkkaus`.

Rakennemerkkauksella merkitään dokumentin looginen rakenne kuten `<tilaus>` kuvassa 16 ja `<esitys>` kuvassa 17. Sisältömerkkausta käytetään reaalimaailman olioita tai olioiden attribuuttien arvoja esittävien merkkijonojen erottamiseen kuten `<nimi>` kuvassa 16 ja `<tekijä>` kuvassa 17. Järjestelmien yhteydessä puhutaan yleensä vain rakenteen merkkaamisesta eikä sisältömerkkausta käsitellä erikseen. Yoshikawa & al. (1999) esittelevät *abstraktin tietotyypin* (*Abstract Data Type - adt*) nimeltään paratext, joita he käyttävät ParaDocs-järjestelmässä sisältömerkkauksen hallintaan.

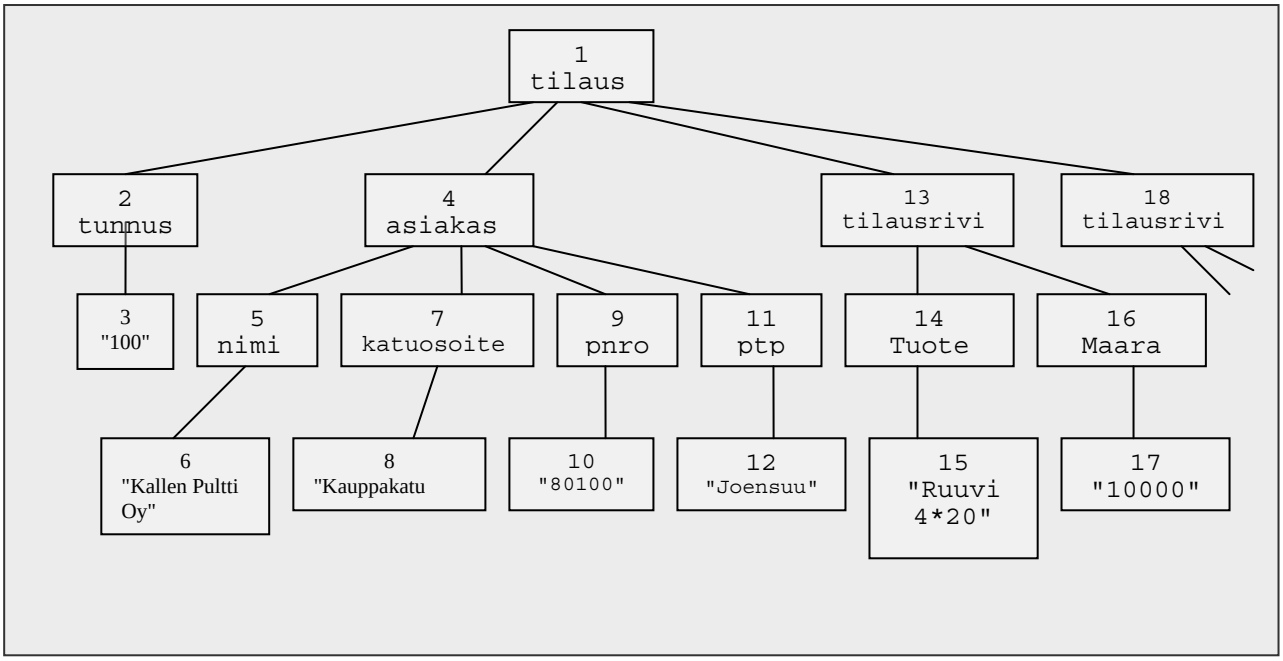
5 Sisäinen tallennusrakenne

Järjestelmien käyttämä sisäinen tallennusrakenne riippuu sekä tallennettavan tiedon tyypistä että sen käyttötarpeesta, joten sisäiselle tallennusrakenteelle on vaikeampi kehittää samanlaista yleispätevää standardia kuin xml-kielioppi on ulkoiselle tallennusrakenteelle tai relaatiomalli ja sql-kieli määrämuotoiselle tiedolle. Esimerkiksi datakeskeistä tietoa siirretään useimmiten relaatiotietokantojen välillä tiedon ollessa numeerista taloushallintoon, laskutukseen, budjetointiin tai muuhun vastaavaan liiketoimintoon liittyvää laskennallista tietoa. Dokumenttikeskeisen tiedon, esimerkiksi käyttöoppaiden, pääasiallinen käyttötarkoitus taas voi olla haku asiasanojen tai rakenneosien perusteella. Vaikka xml- tietomallit ja standardi xquery-kyselykieli ovatkin olemassa, täytyy sisäinen tallennusrakenne ratkaista tapauskohtaisesti.

Relaatiomalli ja xml-tietomalli sekä niiden yhdistelmät ovat vallitsevia sisäisen rakenteen tallennusmalleja, joten tässä luvussa selvitetään, kuinka niissä on toteutettu rakenteisen tekstin tallennus. Rakenteisen tekstin yleistämme jatkossa xml:ksi sen standardin ja yleisyyden vuoksi.

5.1 Jäsennyspuut

Rakenteisen tekstin sisäisenä tallennusrakenteena käytetään tekstin hierarkkisen luonteen vuoksi yleisesti *puurakennetta* (Kilpeläinen, 1992). Esimerkiksi kuvan 17 esityksen rakenne voidaan esittää kuvan 19 jäsennyspuuna.



Kuva 19. Rakenteisen dokumentin puumallinen esitys.

Tekstin yksi perusominaisuus on järjestys. Jäsennyspuu on *järjestetty puu*, jossa jokainen solmu on nimetty joko kieliopin *perussymbolilla* tai *välikesymbolilla* (Salminen, 1992). Välikesymboleita ovat haarasolmut, kuten puun juurisolmu `tilaus`. Perussymboleilla nimetään päätesolmut eli puun lehdet, esimerkiksi `"Kallen Pultti Oy"`. Jäsennyspuuhun liittyy aina kielioppi, ja siitä käyvät ilmi jäsennetyn tekstin osien suhteet. Erillistä kielioppimäärittelyä ei välttämättä ole konkreettisesti olemassa, jolloin jäsennyspuu muodostetaan samalla kun teksti käydään merkki merkiltä läpi ja osat tunnistetaan erottimien perusteella.

Jäsennyspuu voidaan muodostaa usealla tavalla, kuten luvusta 5.2 ilmenee. Jotkut mallit ovat luonnostaan puumallisia kuten xml-tietomalliin kuuluvissa dom:ssa (*Documnet Object Model*) ja xdm:ssä (*Xquery 1.0 and Xpath 2.0 Data Model*), ja joissain se on mukana käsitteellisellä tasolla kuten relaatiomalliin pohjautuvissa ratkaisuissa.

5.2 Relaatiomalliin pohjautuvat ratkaisut

Relaatiomallin taulukkorakenteella on vaikea kuvata tekstin hierarkkista puumallia sisäkkäisine rakenteineen (Sacks-Davis & al., 1994). Tekstissä on osien järjestyksellä tärkeä merkitys, kun taas relaatiomallin taulurakenne koostuu järjestämättömistä riveistä. Tekstin rakenne muuttuu usein, ja muutoksia on vaikea hallita relaatiomallin kiinteällä skeemalla. Relaatiokannoissa ovat tiedon rakenne, tyyppi ja koko etukäteen tiedossa. Tieto on määrämuotoista ja relaatiokantoja käyttävät sovellukset on rakennettu noudattamaan pysyvää tietomallia.

Relaatiokantojen yleisyyden vuoksi on relaatiomallia kuitenkin käytetty yleisesti myös rakenteisen tekstin mallintamiseen. Relaatiokannat soveltuvat lähinnä datakeskeisten dokumenttien hallintaan, mutta erilaisten tekniikoiden ja laajennusten ansiosta ne ovat käytettävissä myös dokumenttikeskeisen tiedon käsittelyyn. Rakenteinen tekstin tallennus relaatiokantaan sen laajennuksia käyttäen voidaan toteuttaa:

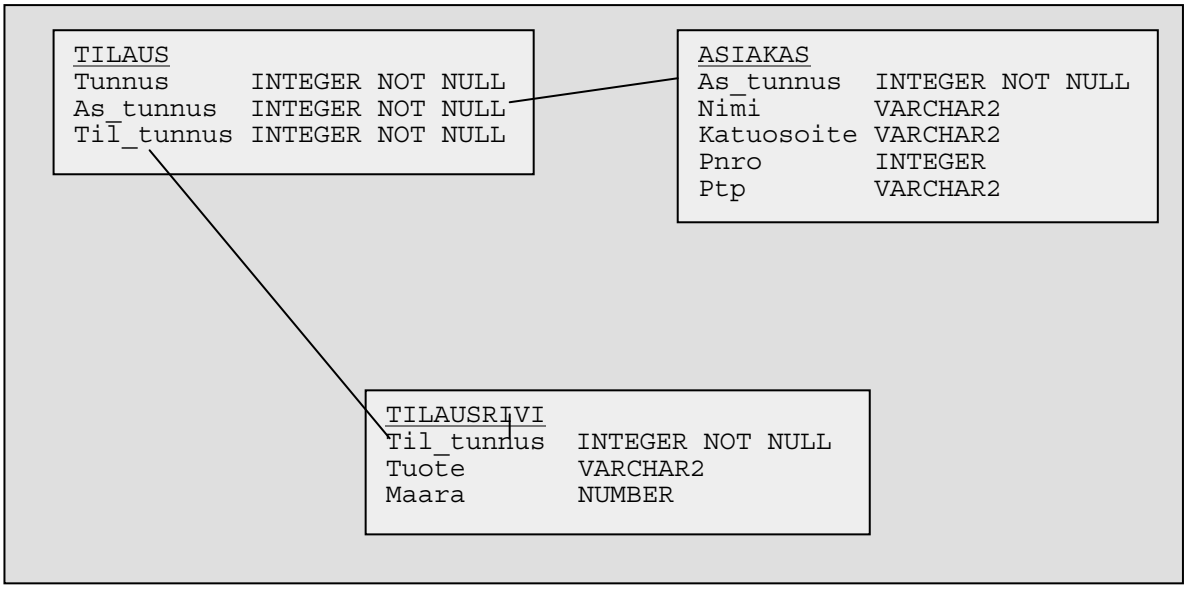
- `tauluihin jaolla,`
- `vaihtuvanmittaisilla tekstikentillä,`
- `sisäkkäisillä relaatioilla,`
- `virtuaalitauluilla,`
- `oliolaajennuksella.`

Lukuisia muitakin tapoja on olemassa, mutta tarkastelemme tarkemmin kolmea ensimmäistä. Oliolaajennusta emme käy tässä tarkemmin läpi, sillä sen tallennusrakenne on samankaltainen kuin xml-tietomallissa (kohta 5.3.1).

5.2.1 Tauluihin jako

Dokumentti voidaan jakaa rakenteensa mukaisesti osiin ja tallettaa osat relaatiokannan tauluihin. Esimerkiksi kuvan 16 tilausrivi voidaan jakaa tauluihin kuvan 20 mukaisesti. Rakennetunnisteita (kuten `tilaus`) käytetään tauluina ja sisältötunnisteita (kuten `maara`) sarakkeina. Koska esimerkin tiedostossa ei ole asiakastunnusta, täytyy asiakas etsiä nimen perusteella tai luoda asiakas aina uudestaan `asiakas`-tauluun. Molemmat vaihtoehdot ovat huonoja, sillä nimen perusteella asiakasta ei pystytä aukottomasti yksilöimään ja kirjoitusasu voi vaihdella. Saman asiakkaan lisääminen useaan kertaan taas sotii relaatiomallin periaatteita vastaan, kun samaa tietoa toistetaan. Satunnaisissa tilauksissa se ei ole ongelma, mutta

jos asiakaskunta on vakio, järkevämpi olisi välittää tilauksen mukana esimerkiksi y-tunnus, jolla asiakas yksilöidään. Ellei taulurakenne ole valmiina, jouduttaneenkin suorittamaan tietojen *normalisointi* esimerkiksi saman asiakkaan perustietojen esiintyessä tiedostossa useaan kertaan.



Kuva 20: Kuvan 16 xml-tiedosto jaettuna relaatiotauluihin.

Taulurakenne voidaan luoda myös dtd-kuvauksen tai xml-skeeman perusteella. Esimerkiksi kuvan 18 dtd:stä luotaisiin rakennetunnisteiden mukaiset taulut `esitys`, `luku ja alaluku` ja niille avainkentät. Vastaavasti sisältötunnisteista, joita ovat *pcdata*-tyyppiset elementit, luodaan taulujen sarakkeet siten kun ne esiintyvät määrityksessä. Kuvan 16 esimerkissä tauluihin tallennettiin dokumentin sisältö, ei elementtejä eikä järjestystä. Jos rakenne halutaan mukaan, on se tallennettava kantaan. Yksi mahdollisuus on käyttää *edge approach*-menetelmää (Laamanen, 2005), jossa dokumentti voidaan tallettaa yhteen *edge-tauluun* (kuva 21).

Id	Lähde	Järjestys	Nimi	Tyyppi	Arvo
1	1	1	Asiakas	Varchar2	NULL
2	2	1	Nimi	Varchar2	Kallen Pultti Oy
3	2	2	Katuosoite	Varchar2	Kauppakatu 100
4	2	3	Pnro	Varchar2	Joensuu
5	3	1	Tilausrivi		NULL
6	4	1	Tuote	Varchar2	Ruuvi 4*20
7	4	2	Maara	Integer	10000
8	5	1	Tuote	Varchar2	Mutteri 5*30
9	5	2	Maara	Integer	15000

Kuva 21: Tilaus-esimerkin tallennus edge-tauluun.

Edge-taulun `id-sarake` yksilöi tietueen ja `lähde`-sarake ilmaisee elementin hierarkkiatasoa dokumentin sisällä. `Järjestys`-sarakkkeessa kerrotaan samalla tasolla olevien elementtien keskinäinen järjestys. `Nimi`-sarakeessa kerrotaan elementin nimi, `tyyppi`-sarakkeessa sen tyyppi ja `arvo`-sarakkeessa arvo. `Tyyppi`-sarakkeen käyttö on vapaaehtoista. Edge approach-menetelmästä on useita versioita. *Binary Approach*-menetelmässä jokainen elementti ja attribuutti tallennetaan omaan tauluunsa.

Tauluihin jako ohjelmallisesti ei ole aina kovin yksinkertainen tehtävä ja siksi se soveltuu parhaiten datakeskeisten dokumenttien tallennukseen, kun taulurakenne on tiedossa eikä dokumenttien rakenne muutu. Dokumenttikeskeisten tiedostojen tapauksessa on järkevämpää käyttää muita ratkaisuja kuten xml-tyyppisiä tai vaihtuvanmittaisia kenttiä.

5.2.2 Vaihtuvanmittaiset kentät

Yksinkertaisin tapa tallettaa dokumentti tietokantaan on käyttää vaihtuvanmittaisia tekstikenttiä *varchar tai clob* (*Character Large Object*) tai binaarimuotoista *blob:ia* (*Binary Large Object*). Jatkossa näistä käytetään nimitystä *merkkijonokenttä*. Kenttien pituutta ei ole etukäteen rajoitettu, mutta yleensä niillä on jokin maksimiarvo, joka on kantakohtainen. Dokumentti tallennetaan kenttään sellaisenaan, ja käsittely tapahtuu sovellusohjelmissa. Tehtäessä rakenteeseen perustuvia hakuja täytyy dokumentti jäsentää jolloin haut ovat hitaita. Ratkaisu soveltuukin lähinnä dokumenttikeskeisen

tiedon tallennukseen, kun haut kohdistuvat kokonaisiin dokumentteihin. Kuvan 17 `esitys` voidaan tallettaa merkkijonokenttään kuvan 22 mukaisesti. Hakujen helpottamiseksi voidaan tauluun lisätä dokumenttia kuvaavia sarakkeita kuten `tekija`, `tallennuspvm` ja `tyyppi`.

Id	Tekija	Tallennuspvm	Tyyppi	Xml (clob)

1	Hannu Toroskainen	23.04.2008	Esitys	`<?xml version="1.0"?>`
				`<esitys nro = 1>`
				`<otsikko> Rakenteisen tekstin tallennusrakenteet </otsikko>`
				`<tekijä> Hannu Toroskainen </tekijä>`
				`<sisältö>`
				`<luku>`
				`<tunnus>1</tunnus>`
				`<otsikko> Johdanto`
				`</otsikko>`
				`<teksti> Tässä luvussa selvitetään … </teksti>`
				…
				</esitys>

2	Hannu Toroskainen	02.05.2008	Kirje	…
3	Kalle Kehveli	29.05.2008	Esitys	…
			…	

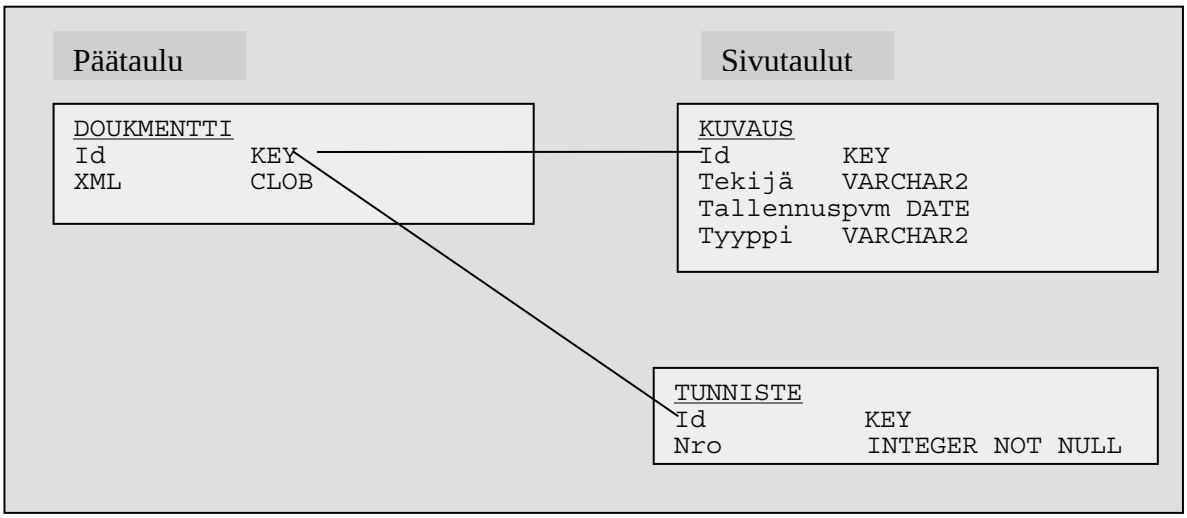
Kuva 22: Xml-dokumentti merkkijonokenttään tallennettuna.

Dokumenttien rakenne voi vaihdella paljon, joten luotu taulurakenne ei välttämättä sovellu kuin yhdentyyppisille dokumenteille. Tauluun voidaan luoda lisäkenttiä tarpeen mukaan, mutta erityyppisten dokumenttien lisääntyessä kasvaa taulun sarakkeiden määrä isoksi. Käyttötarpeesta riippuen voi erilaista skeemaa noudattavien dokumenttien tallennus erillisiin tauluihin tai jopa omiin instansseihin olla perusteltua. Dokumentin hakeminen esimerkiksi esityksen numeron perusteella tarkoittaa kaikkien dokumenttien läpikäyntiä ja dokumentin jäsentämistä. Esitys numero 1 voidaan hakea kyselyllä:

```
SELECT xml
FROM   dokumentti
WHERE  extract(xml, '/esitys/@nro') = 1;
```

Sivutauluja (*side tables*) käyttämällä saadaan haut tehokkaammiksi. Kuvan 22 taulu voidaan jakaa kolmeen tauluun siten, että varsinainen dokumentti jätetään niin sanottuun päätauluun (`dokumentti`) ja muodostetaan kaksi sivutaulua (`kuvaus` ja

tunniste). Toiseen sivutauluun viedään dokumenttia kuvaavat sarakkeet tekija, tallennuspvm ja tyyppi ja toiseen esityksen numero kuten kuvassa 23.



Kuva 23: Sivutaulujen käyttö.

Tallennusvaihe on luonnollisesti hitaampi sillä dokumentti joudutaan jäsentämään. Sivutaulut toimivat nyt eräänlaisina indekseinä, ja esitys numero 1 saadaan haettua suoraan sivutaulun tunniste avulla:

```
SELECT xml
FROM dokumentti d, tunniste t
WHERE  d.id = t.id
    AND t.nro = 1;
```

Eräs mahdollisuus on luoda erityyppisille dokumenteille omat taulunsa, kuten esitys- ja kirje-taulut. Esitän kohdassa 5.7 *tyyppitauluihin* perustuvan ratkaisuni, joka mahdollistaa joustavasti uusien dokumenttityyppien lisäämisen tietokantaan.

5.2.3 Sisäkkäiset relaatiot

Sisäkkäiset relaatiot (*nested relations*) mahdollistavat relaatiokannassa monimutkaisempien rakenteiden käytön, eikä erillisiin tauluihin jaettua dokumenttia tarvitse uudelleen julkaistessa jälleen yhdistellä kuten tauluihin jaettaessa (Kar, 1996). Esimerkiksi esityksen (kuva 18) skeema voidaan esittää sisäkkäisenä relaationa kuvan 24 mukaisesti.

```
esitys [
                nro             INTEGER,
                otsikko         STRING,
                tekijä          STRING,
                sisältö         LIST of luku
] KEY = (nro)

luku [
                tunnus          INTEGER,
                otsikko         STRING,
                teksti          STRING
                alaluvut        LIST of alaluku
] KEY = (tunnus)

alaluku [
                alatunnus       INTEGER,
                alaotsikko      STRING,
                teksti          STRING
] KEY = (alatunnus)
```

Kuva 24: Esimerkkiesityksen skeema sisäkkäisenä relaationa.

5.2.4 Virtuaalitaulut

Virtuaalitauluratkaisu on kieliopista riippumaton ratkaisu, jolla voidaan esittää yleisen xml-dokumentin rakenne. Mallissa luodaan kolme kappaletta niin sanottuja virtuaalitauluja (Kar, 1996):

- `text_nodes (nodeid, genid, content),`
- `text_attributes (nodeid, attr, value),`
- `text_structure (a_nodeid, d_nodeid, order).`

`Text_nodes`-taulussa on jäsennyspuun jokainen solmu. `Text_attributes`-taulu sitoo solmun sen attribuutteihin ja arvoihin, ja `text_structure`-taulussa kerrotaan solmun edeltäjä-seuraajasuhteet.

Yleinen tarkoittaa tässä sitä, että malli sopii kaikenlaisten xml-tiedostojen tallentamiseen. Xml-tietomallissa käytetään vastaavaa ratkaisua, joten emme käy ratkaisua tässä enempää läpi.

5.3 Natiivi xml–tietokanta

Natiivi xml–tietokanta käyttää sisäisenä tallennusrakenteena xml-tietomallia. Luvussa 5.3.2 katsotaan, kuinka tietomalli toteutetaan B-puuna natiivissa eXist-tietokannassa. Jatkossa xml-tietokannalla tarkoitetaan natiivia xml-tietokantaa tai xml-yhteensopivaa kantaa..

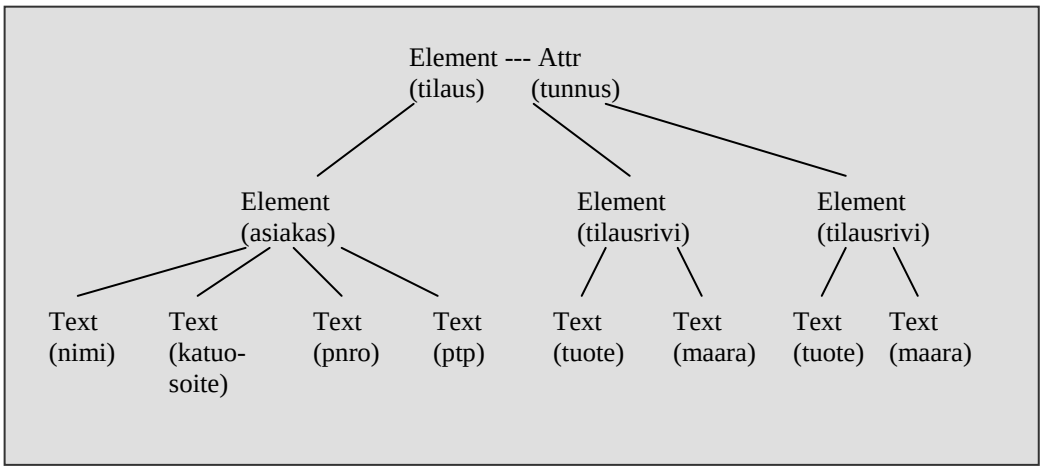
5.3.1 Xml-tietomalli

Natiivi xml–tietokanta määrittelee xml-dokumentin loogisen mallin, joka sisältää minimissään elementit, attribuutit, *pcdata-tietotyypin* (*Parser Character Data*) sekä järjestyksen (Bourret, 2005). Elementti tallennetaan pcdata-kenttään siten, että siinä on mukana alku- ja lopputunniste. Natiivi tarkoittaa, että dokumentit tallennetaan, indeksoidaan ja haetaan alkuperäisen xml-muodon mukaisesti. Käsittely koskee kaikkia dokumentin osia eli sisältöä, tunnisteita, attribuutteja, viittauksia ja järjestystä.

Loogisen rakenteen perusyksikkö on xml-dokumentti, kuten relaatiomallissa on taulun rivi. Dokumentti tallennetaan ja palautetaan xml-tietomallin mukaisesti, jolloin muunnosta mallista toiseen ei tarvita, kuten ei myöskään erillistä fyysisen rakenteen mallia. Kanta voidaan rakentaa esimerkiksi relaatio-, hierarkkisen- tai olio-relaatiomallin päälle tai käyttää indeksoituja ja kompressoituja tiedostoja omalla formaatilla.

On huomattava, että xml-yhteensopivissa kannoissa käytettävä oliomalli ei ole xml-tietomalli, sillä se mallintaa dokumentin sisällön ja on erilainen jopa samaa xml-skeemaa noudattavilla dokumenteilla (Bourret, 2005). Xml-tietomallit (*Infoset, dom, xdm*) taas mallintavat itse dokumentin. Kuvassa 25 dom-puu kuvaa kuvan 19 dokumenttia. Siinä puu muodostuu komponenteista `Element`, `Attr` ja `Text` kun oliomallissa kuvan 18 dtd:n mukainen dokumenttipuu muodostettaisiin objekteista `esitys`, `otsikko`, `tekijä`, `sisältö`, `luku` ja niin edelleen. Voidaan siis sanoa, että xml-tietomalli on kieliopista riippumaton ratkaisu, kuten luvussa 5.2.4 esitetty virtuaalitauluratkaisukin oli. Malliin voidaan siis lisätä minkämuotoinen xml-dokumentti tahansa. Mallit muodostavat dokumentin sisäisen, loogisen mallin, joka

voidaan toteuttaa esimerkiksi relaatiokannassa luomalla komponentteja vastaavat taulut kuten `elements`, `attributes`, `pcdata`, `entities` ja `entityReferences`.



Kuva 25: Dom-puu kuvan 19 jäsennyspuulle.

5.3.2 Jäsennyspuun toteuttaminen

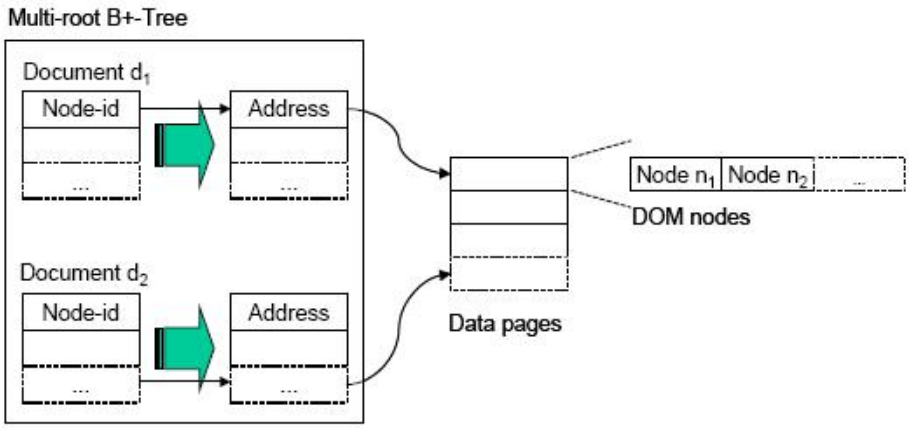
Järsennyspuun toteuttamiseksi voidaan käyttää *B-puita* (binaaripuut), joissa kaikki lehdet ovat samalla tasolla ja varsinaiset tiedot tallennetaan lehtisolmuihin. B+ -puussa matka juuresta lehteen on aina vakio. Tarkastellaan esimerkkinä natiivia eXist-kantaa, joka käyttää neljää B+ -puuhun pohjautuvaa indeksitiedostoa (Meier, 2002):

- `kokoelmahierarkkiat (collections.dbx),`
- `solmut (dom.dbx),`
- `elementit ja attribuutit (elements.dbx),`
- `sanat (words.dbx).`

Dokumentit tallennetaan hierarkkisiin kokoelmiin `kokoelmahierarkkiat`-indeksiin. Suorituskykysyistä dokumenttien kuvaukset tallennetaan vastaavan kokoelmaobjektin yhteyteen.

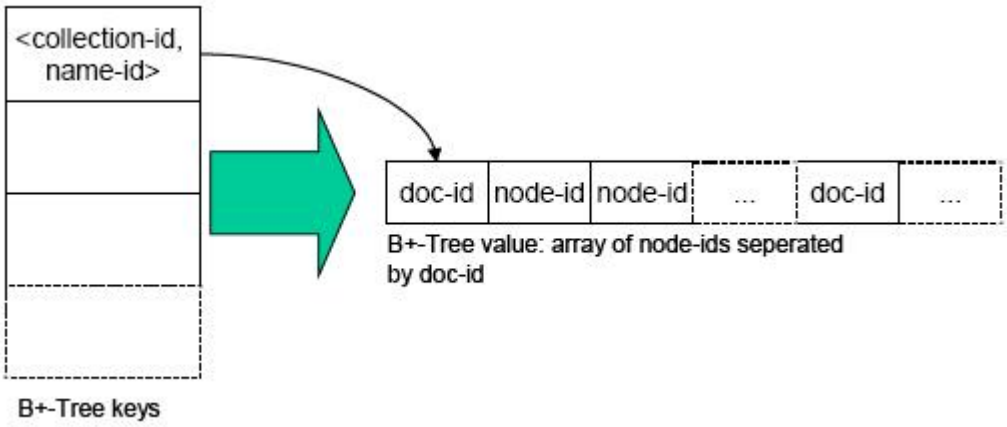
Kannan pääkomponentti on `solmut`-tietovarasto, johon dokumentin kaikki solmut tallennetaan dom-mallin mukaisesti (kuva 26). Vain ylätason elementit indeksoidaan B+-puuna. Attribuutit, tekstisolmut ja alemman tason solmut tallennetaan *Data Pages*-osioon lisämättä avainta B+-puuhun. Näiden solmujen haku tapahtuu

lähimmän edeltäjän kautta. Suurin osa xpath-operaatioista tehdään käyttämättä pääkomonenttia, joten sillä ei ole suorituskyvyn kannalta suurta merkitystä.



Kuva 26: Tiedon tallennusrakenne eXist-järjestelmässä (Meier, 2002)

Indeksit elementeille, atribuuteille ja avainsanoille muodostetaan dokumenttikokoelman, ei yksittäisen dokumentin mukaan. Esimerkiksi kaikki `"esitys"`-elementin esiintymät tallennettaisiin yhtenä `elementit`-indeksin indeksinä (kuva 27). Näin saadaan kannan kokoa pienemmäksi ja koko kokoelmaan tai useisiin kokoelmiin perustuvat haut nopeammaksi, sillä indeksi tarvitsee hakea vain kerran.



Kuva 27: eXist-järjestelmän indeksirakenne elementeille ja attribuuteille (Meier, 2002).

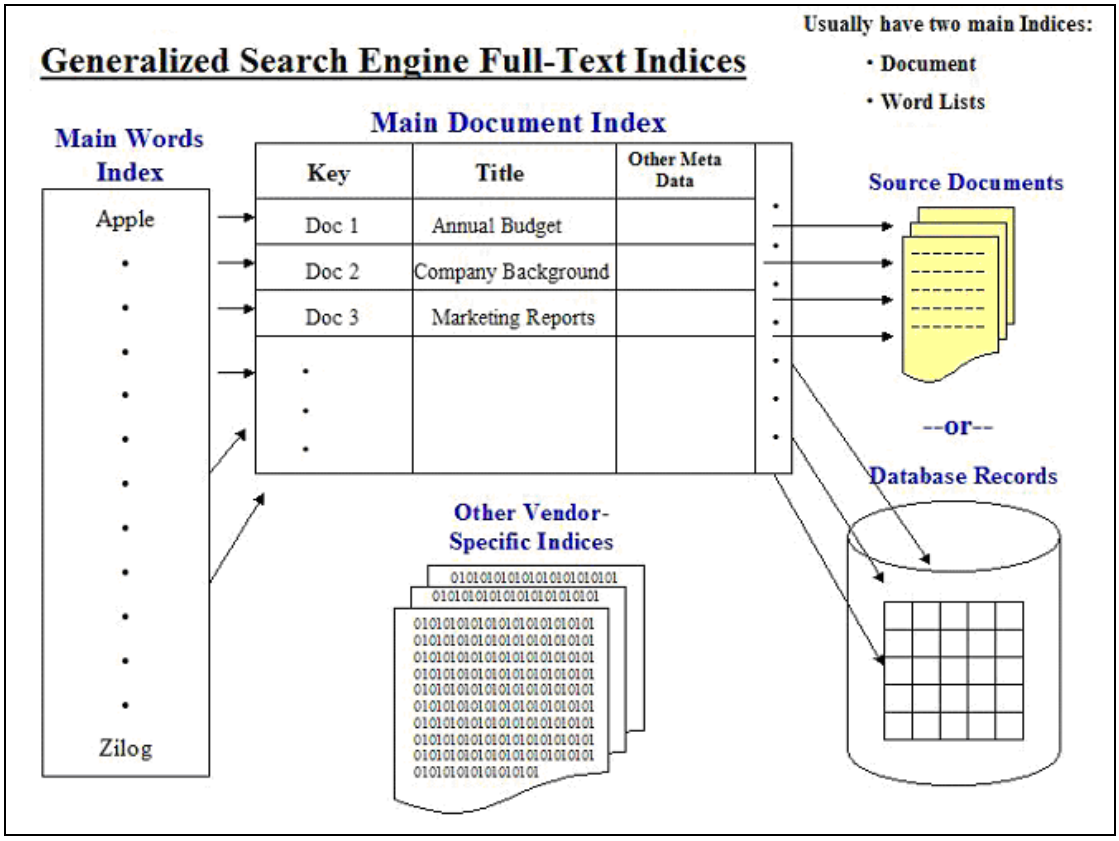
Sanat indeksoidaan käänteislistaksi, joka muistuttaa tiedonhakujärjestelmissä käytettyjä listoja (ks. kohta 5.6). eXist indeksoi oletusarvoisesti kaikki tekstisolmut ja attribuutit käyttäen samanlaista rakennetta kuin on elementeille käytettävä kuvassa 27. Jokainen <collection-id, keyword >-avainpari osoittaa teksti- tai attribuuttisolmuun, jossa avainsana esiintyy.

Natiivi xml-tietokannat voidaan jakaa *tekstipohjaisiin* ja *mallipohjaisiin* (Bourret, 2005). Tekstipohjaisessa mallissa xml-tiedosto tallennetaan tekstinä, joka voi olla tiedosto levyllä, blob relaatiokannassa tai valmistajan oma formaatti. Mallipohjaisessa kannassa käytetään sisäistä mallia, kuten domia. Edellä esitetty eXist on mallipohjainen kanta.

5.6 Tiedonhakujärjestelmät ja käänteislistat

Tiedonhakujärjestelmät pohjautuvat *käänteislistojen* (*inverted index*) käyttöön, ja alun perin niillä pystyttiin hakemaan *vapaasanahaulla* vain kokonaisia dokumentteja (Macleod, 1991). Toinen äärilaita on ollut rakenteellinen tieto, johon tietokantatutkimus on perinteisesti keskittynyt (Magnani & Montesi, 2004). Näiden välissä ollutta tyhjiötä täyttämään on tullut puolirakenteinen tieto, jonka käyttöä on xml-standardin myötä tutkittu molemmissa yhteisöissä.

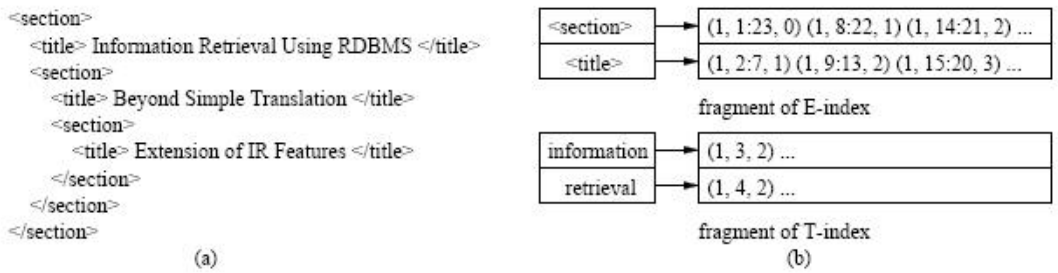
Käänteislistoissa dokumentin sanat ja sijainnit indeksoidaan erilaisiksi indeksilistoiksi (esimerkiksi termilista ja paikkalista). Kuvassa 28 näitä edustavat `Main Word Index` ja `Main Document Index`, joista jälkimmäiseen voidaan tallettaa metatietoa kuten dokumentin nimi. Termilistassa ovat kaikki dokumentin sanat mahdollisia *täytesanoja* lukuun ottamatta. Täytesanoja kuten ’ja’ ei kannata niiden yleisyyden vuoksi indeksoida, vaan niille voidaan luoda oma lista (*stopwords*), josta ne tunnistetaan.



Kuva 28: Tiedonhakujärjestelmän arkkitehtuuri (Bennett04)

Tiedonhakujärjestelmiin on otettu piirteitä tietokantamaailmasta, ja vieläkin monissa on perusrakenteena perinteinen relaatiokannan taulurakenne kuten kuvassa 28. Käänteislistat voidaan tallettaa levylle tekstitiedostoina ja lukea tarvittaessa tiedonhakujärjestelmään tai tietokantaan.

Käänteislistoja käytetään hakukoneissa ja xml–kannoissa dokumenttien ja niiden osien hakuun. Käänteislistoihin perustuvia rakenteisen teksin käsittelyyn pohjautuvia indeksointiratkaisuja ovat esimerkiksi *E-indeksi* ja *T-indeksi* (Zhang, 2001). T-indeksi vastaa perinteisissä tiedonhakujärjestelmissä tekstin sanat sisältävää listaa ja E-indeksi elementtilistaa (kuva 29). T-indeksi on muotoa `(dokumenttinumero, sananumero, taso)` ja E-indeksi muotoa `(dokumenttinumero, alku:loppu, taso)`.



Kuva 29: (a) Esimerkki XML-dokumentista, (b) sen käänteisindekseistä E-index ja T-index (Zhang, 2001).

Sekä relaatio- että xml-tietokannat soveltuvat hyvin monimutkaisiinkin hakuihin, kun tietokannan skeema on tiedossa ja kyselyt voidaan määritellä selkeästi (Al-Khalifa, 2003). Usein halutaan kuitenkin hakea vapaasanahaulla dokumentteja, jotka sisältävät luonnollista kieltä ja joiden skeema vaihtelee. Tiedonhakujärjestelmissä käytetyt tekniikat soveltuvat tämänkaltaisiin hakuihin, ja niitä, kuten *oleellisuusluokituksia* (*relevance ranking*) on otettu käyttöön esimerkiksi xml-tietokantojen vapaasanahakuominaisuuksissa ja hakukoneissa.

5.7 Tyyppitaulut

Esitän tässä merkkijonotallennusta vastaavan ratkaisuni, jota voidaan käyttää relaatio-ominaisuudet sisältävissä kannoissa. Kuten merkkijonotyyppisessä ratkaisussakin (luku 5.2.2), mallissa tallennetaan dokumentti kokonaisuudessaan yhteen *päätaulussa* sijaitsevaan blob- tai xml-tyyppiseen *dokumenttikenttään* (kuva 30). Suositeltavampaa on xml-tyyppisen sarakkeen käyttö, jolloin dokumenttia voidaan käsitellä suoraan rakenteisena. Jos halutaan tallettaa muitakin kuin xml-tiedostoja, niin sekä blob-kentän että xml-tyyppisen kentän käyttö on mahdollista. Jälkimmäisessä tapauksessa luodaan päätauluun yksi lisäkenttä, joka kertoo, onko kyseessä xml-dokumentti vai jokin muu kuten pdf-, doc- tai ascii –muotoinen tiedosto. Päätauluun luodaan `dokumenttikentän` lisäksi kaikille dokumenteille yhteiset sarakkeet kuten `tekijä`, `tallennuspvm` ja `tyyppi`.

Id	Kokoel-ma_id	Tekijä	Tallennus-pvm	Tyyppi	Dokumentti
1	1	Hannu Toroskainen	23.04.2008	Esitys	<?xml version="1.0"?> <esitys nro = 1> <otsikko> Rakenteisen tekstin tallennusrakenteet </otsikko> <tekijä> Hannu Toroskainen </tekijä> <sisältö> <luku> <tunnus>1</tunnus> <otsikko> Johdanto </otsikko> <teksti> Tässä luvussa selvitetään … </teksti> … </esitys>
2	2	Hannu Toroskainen	02.05.2008	Kirje	…
3	1	Kalle Kehveli	29.05.2008	Esitys	…
				…	

Kuva 30: Esimerkki päätäulusta 'dokumentit'.

Päätaulun lisäksi luodaan kullekin dokumenttityypille oma *tyyppitaulu*, jonka nimi on sama kuin dokumentin tyyppi eli päätaulun tyyppi-sarakkeen arvo kuten `esitys` tai `kirje`. Kuvassa 31 on esimerkki esitys-tyyppisten dokumenttien tyyppitaulusta jossa kyseistä tyyppiä kuvaavia sarakkeita edustavat `salaisuusaste` ja `versio`.

Dok_id	Kohderyhma	Salaisuusaste	Versio
1	IT-ammattilaiset	Julkinen	1.0
3	Opiskelijat	Julkinen	2.1
		…	

Kuva 31: Esimerkki 'esitys'-dokumenttien tyyppitaulusta.

Ratkaisu on joustava, sillä se mahdollistaa helposti uusien dokumenttityyppien lisäämisen. Tyyppisarakkeen ansiosta hauista voidaan tehdä dynaamisia ja tyyppitauluun voidaan lisätä helposti uusia kyseistä dokumenttityyppiä kuvaavia sarakkeita. Hakuja voidaan tehdä sql:llä ja xml-tyyppisestä kentästä xquerylla tai sql/xml:llä. Esimerkiksi esitykset, joiden kohderyhmänä ovat IT-ammattilaiset, voidaan hakea SQL-kyselyllä

```
SELECT d.xml
FROM dokumentit d, esitys e
WHERE d.id=e.dok_id AND e.kohderyma = “IT-ammattilaiset”;
```

Xml-tyyppisestä sarakkeesta voidaan tehdä rakenteeseen pohjautuvia kyselyjä, minkä lisäksi dokumentin päivitykset ovat mahdollisia. Esimerkiksi `esitys`-dokumentista voidaan xml-tyyppistä kenttää käytettäessä hakea IT-ammattilaisille tarkoitettujen esitysten otsikot

```
SELECT otsikko
FROM XMLTABLE (’$dokumentit’
    PASSING dokumnetit.dokumentti AS ”dokumentit” BY REF
    COLUMNS otsikko VARCHAR(20) PATH ’esitys/otsikko’)
    dokumentti d
WHERE d.tyyppi = 'Esitys';
```

Malli toimii myös dokumenttitietokantana ja tiedonhakujärjestelmänä, sillä dokumentteja voidaan hakea pää- ja tyyppitauluissa olevien metatietojen perusteella tai tehdä vapaasanahakuja xml-kentästä.

Mallia voidaan edelleen laajentaa dokumenttikokoelmaksi luomalla tyyppitaulua vastaava *kokoelmataulu* (kuva 32) ja lisäämällä päätauluun kokoelmatunnus `kokoelma_id`. Kokoelmat voivat olla esimerkiksi osasto- tai tytäryhtiökohtaisia.

Kokoelma_id	Organisaatio	Osasto	
1	Emoyhtiö	IT	1.0
2	Emoyhtiö	HR	2.1
3	Verkko	Suunnittelu	
	…		

Kuva 32: Esimerkki kokoelmataulusta.

6 Tallennusratkaisun valinta

Koska tekstin käyttötarpeet ovat mitä moninaisimpia, täytyy rakenteisen tekstin tallennusratkaisun käyttökelpoisuus arvioida sekä tallennettavan tiedon tyypin että sen tarpeen mukaan. Teksti voi olla esimerkiksi staattista ja tallennettu vain hakuja varten, se voi olla luotavana ja muokattavana usean yhtäaikaisen käyttäjän kesken tai se voi olla siirrettävänä järjestelmien välillä. Dokumentit voivat olla tyypiltään data- tai dokumenttikeskeisiä tai niiden välimuotoja.

Tässä luvussa käsiteltävät tietokantaratkaisut ovat nyt ja käsitykseni mukaan myös lähitulevaisuudessa vallitsevia rakenteisen tekstin tallennusratkaisuja. Näiden rinnalla käytetään tiedonhakujärjestelmiä, dokumentinhallintajärjestelmiä ja sisällönhallintajärjestelmiä, mutta erot järjestelmien välillä tulevat häviämään, kun tietokantoihin lisätään kyseisiä ominaisuuksia. Esimerkiksi SqlServer 2005:ssä voidaan dokumentit tallettaa xml-tyyppiseen kenttään ja käyttää xquery-kyselyä rakennehakuun tai indeksoida dokumentti ja käyttää vapaatekstihakua (XML Best Practices, 2006). Liitteessä 1 on yhteenveto tässä luvussa esitetyistä valintakriteereistä.

6.1 Puolirakenteinen malli ja xml-tiedostot

Xml-tiedostoja on käsitelty tässä tutkielmassa ulkoisena tallennusrakenteena. Jos ajatellaan, että tietokanta on kokoelma tietoa, voidaan ulkoista xml-dokumenttiakin pitää tietokantana (Bourret, 2005). Xml ja sitä ympäröivät teknologiat täyttävät tietokantaan oleellisesti liittyvän *tietokannanhallintajärjestelmän* (*DataBase Management System - DBMS*) vaatimuksen osittain, sillä XML tarjoaa muun muassa:

- tietojen varastoinnin,
- skeemat (dtd, xml-skeema),
- kyselykielet (xquery, xpath),
- ohjelmointirajapinnat (jdom, sax, dom).

Tietojen varastointi tapahtuu xml-tiedostoina, joiden rakenne voidaan kuvata dtd:llä tai xml-skeemalla. Xml:ssä voidaan määritellä linkkejä toisiin dokumentteihin ja muuntyyppiseen dataan kuten multimediatiedostoihin. Tietokantoihin verrattuna XML on lisäksi

- itsensäkuvaavaa (self-describing),
- siirrettävää (portable),
- se voi kuvailla datan puu- tai graafirakenteena.

XML:stä puuttuu ominaisuuksia (Bourret, 2005; Lapis, 2005), jotka löytyvät perinteisistä tietokannoista. Näitä ominaisuuksia ovat esimerkiksi

- tehokas ja pysyvä tallennus,
- indeksointi,
- turvallisuus,
- transaktioiden ja tiedon eheyden hallinta,
- usean käyttäjän hallinta,
- rajoitukset (triggers).

Xml-tietokantojen myötä nämä puutteet ovat poistumassa, mutta senkin jälkeen dokumentteja tallennetaan ja käsitellään ulkoisina tiedostoina, joihin kohdistetaan esimerkiksi hakuja. Xqueryyn sisältyvien uusien tekniikoiden ansiosta hakuja voidaan tehdä useasta dokumentista ja haut isoistakin xml-dokumenteista ovat suhteellisen nopeita. Xqueryssä voidaan käyttää erilaisia puskurointi- ja streamaustekniikoita, joilla hakuaikoja saadaan nopeutettua.

Esimerkkihaku DataDirect:n Xquery-moottorilla 47 MB:n kokoisesta, 27 353 riviä (tietuetta) sisältävästä laskutustiedostosta kesti yhden tietueen haku viisi sekuntia. Vastaavasti hakuajat 1,35 GB:n 830 872 riviä sisältävästä tiedostosta vaihtelivat puolentoista ja kahden minuutin välillä riippumatta tietueen sijainnista dokumentissa. Arkistointi- tai satunnaiskäytössä hakuajat ovat näin ollen riittäviä. Huomionarvoista oli sekin, että vaikka haut tehtiin kannettavalla tietokoneella, jossa oli 1 GB keskusmuistia ja 1,73 GHz:n prosessori, niin koneen muisti eikä suorituskyky loppunut hakujen aikana. Yksittäisiä hakuja voidaan tehdä järjestelmien omilla komennoillakin kuten Unixin grep:llä, mutta se palauttaa vain yhden rivin eikä esimerkiksi koko laskutietuetta, jos se on jaettu usealle riville.

Xml:stä on tullut rakenteisen tekstin standardi tallennus- ja siirtomuoto, joten se on käytännössä myös ulkoisen tallennusrakenteen standardi. Tiedon siirrettävyyden takia ulkoisen tallennusrakenteen standardi on iso kehitysaskel, sillä näin on järjestelmien välille saatu ”yhteinen kieli”.

6.2 Relaatiotietokanta

Tietokantajärjestelmiä on käytetty perinteisesti suurten tietomäärien tehokkaaseen ja turvalliseen hallintaan. Tietokannoissa oleva tieto on usein kriittistä, joten niissä on erittäin pitkälle kehitetyt suojaus- ja käsittelyominaisuudet, kuten käyttäjien ja käyttöoikeuksien hallinta sekä tietojen suojaamis-, varmistus-, palautus- ja replikointiominaisuudet. Nämä tietokantajärjestelmiin sisäänrakennetut ominaisuudet helpottavat sekä kehitys- että ylläpitotyötä ja ovat siten yksi hyvä syy tallettaa xml-muotoinen tieto relaatiokantaan, jos tiedon muoto sen helposti sallii. Lisäksi sql-kieli on monipuolinen ja yleisesti käytetty ja on lukuisista valmistajien omista murteistaan huolimatta helppo oppia ja käyttää.

Nykyiset tieto- ja raportointijärjestelmät on rakennettu pääsääntöisesti relaatiokantojen varaan. Jos järjestelmien toiminnallisuus ja xml-dokumenttien käsittelytarve ovat yhtenevät, voi olla järkevämpää hyödyntää järjestelmien valmiita ominaisuuksia kuin rakentaa uusi.

Vaikka tekstin hierakkisuutta on vaikea kuvata relaatiomallilla, on relaatiokanta kuitenkin harkinnan arvoinen valinta myös xml:lle, jos jokin seuraavista kriteereistä täyttyy (Lapis 2005):

- Tiedolla on luonnostaan taulukkorakenne.
- Tarvitaan korkeaa suorituskykyä.
- Tietoa käytetään myöhemmin yhdessä relaatiotiedon tai sitä käyttävän järjestelmän kanssa.
- Tieto ei ole hierarkkista eli ei kuvaa vanhempi-lapsi suhteita,

Relaatiotiedon ja xml:n yhdistämistä on ulkoisen xml-standardin myötä vaikea välttää. Tilaaja haluaa siirtotiedoston usein xml-muodossa tai tiedosto tulee järjestelmään xml:nä. Tulevan tiedoston lukeminen relaatiokantaan ei ole yhtä yksinkertaista kuin sen julkaiseminen sql:llä. Tallennusta varten on tehtävä erillinen jäsentäjäohjelma, joka erottaa datan rakenteesta ja tallettaa sen tauluihin ja sarakkeisiin.

Kohdassa 5.2 esitetyillä ratkaisuilla pystytään myös dokumenttikeskeistä tekstiä hallitsemaan relaatiomallilla, mutta parhaiten malli sopii datakeskeisen tiedon

tallentamiseen. Sama pätee, kun relaatiokannasta muodostetaan ulkoinen xml-tiedosto. Datakeskeisen tiedon (kuten tilausrivin) tapauksessa tiedosto on suhteellisen helppo muodostaa, mutta dokumenttikeskeisen tiedon tapauksessa rakenne voikin olla erilainen kuin sisäänluetun tiedoston.

6.2.1 Tauluihin jako

Varsinkaan dokumenttikeskeisten xml-tiedostojen jakaminen ja tallettaminen relaatiokannan tauluiksi ei ole aina triviaali toimenpide. Sitä kannattaa kuitenkin harkita, jos jokin seuraavista ominaisuuksista täyttyy (Lapis, 2005):

- Xml:ää käytetään vain tiedoston siirtoon eikä rakennetta tarvita tallennuksen jälkeen.
- Dokumentin osien järjestyksellä ei ole väliä.
- Tietoa käytetään yhdessä olemassa olevan relaatiotiedon kanssa.
- Rakenne on yksinkertainen ja siitä on helppo muodostaa relaatiomallin skeema.
- Xml-skeema on pysyvä tai muuttuu hyvin harvoin.
- Nykyiset sovellukset ja raportointijärjestelmät käyttävät vain relaatiokantaa.
- Relaatiokantaan viedystä tiedosta on tarve muodostaa uusi alkuperäisestä eroava xml-dokumentti.
- Haut ja päivitykset on helppo toteuttaa sql:llä. Muunnos xpath:sta sql:ään on helppo tai sitä ei tarvita ollenkaan.
- Tietoja päivitetään säännöllisesti suorituskyvyn ollessa kriittinen tekijä.
- Hakujen ja päivitysten suorituskyky on tärkeämpää kuin xml-muoto.

Joskus voi olla tarpeen jakaa suurempi xml-tiedosto pienempiin. Tauluihin jakoa ja sitä kautta pienempien dokumenttien julkaisemista voi tällöin harkita. Edellä luetellut kriteerit pätevät tällöinkin, sillä monimutkaisten, runsaasti yksittäisiä elementtejä sisältävien dokumenttien tallennus voi vaatia jopa satojen relaatiotaulujen luontia. Haut tällaisesta määrästä muodostuvat helposti hitaiksi, vaikka käytettäisiin indeksointia.

Relaatiokannasta on sql:llä suhteellisen helppo muodostaa xml-muotoinen tiedosto. Oletetaan, että kannassa on taulut, `asiakas`, `tilaus` ja `tilausrivi`, jolloin xml-tilaus voidaan muodostaa kyselyllä:

```
SELECT "<?xml version="||"""||"1.0"||"""||"?>",
          "<tilaus "||"tunnus = "||t.tunnus||">",
          "<asiakas>",
          "<nimi>"||asi.nimi||"</nimi>",
         "<katuosoite>"||asi.katuosoite||"</katuosoite>",
          "<pnro>"||asi.pnro||"</pnro>",
          "<ptp>"||asi.ptp||"</ptp>",
          "</asiakas>",
          "<tilausrivi>",
          "<tuote>"||tr.tuote||"</tuote>",
          "<maara>"||tr.maara"||"</maara>",
          "</tilausrivi>"
          "</tilaus>"
FROM asiakas asi, tilaus t, tilausrivi tr
WHERE asi.nimi="Kallen Pultti Oy" AND
          asi.asiakastunnus=t.asi_asiakastunnus AND
          t.tunnus=tr.til_tunnus;
```

Kuvan 16 esimerkkiin verrattuna kysely palauttaa rivit kuitenkin niin, että asiakastieto ja tilausrivi tulostuvat yhdelle riville relaatiomallin mukaisena *monikkona* (*tuple*). Alkuperäisen näköinen tulostekin saadaan aikaan, mutta se vaatii hieman enemmän työtä ja esimerkiksi aputaulujen käyttöä.

Dokumenttien julkaisemista varten on sql:ään tehty *sql/xml*-laajennus, jonka funktiot muodostavat xml-rakenteen automaattisesti. Esimerkiksi kuvan 16 dokumenttia vastaava xml-tiedosto saadaan haulla

```
SELECT XML2CLOB (XMLELEMENT(NAME "tilaus",
          XMLATTRIBUTES (t.tunnus),
          XMLELEMENT(NAME "asiakas",
              XMLFOREST (asi.nimi,asi.katuosoite,
                     asi.pnro,asi.ptp))
          XMLAGG( XMLELEMENT(NAME "tilausrivi",
              XMLFOREST (tr.tuote,tr.maara))))
) AS "tilaus"
FROM asiakas asi, tilaus t, tilaurivi tr
WHERE asi.nimi='Kallen Pultti Oy' AND
          asi.asiakastunnus = t.asi_asiakastunnus AND
          t.tunnus=tr.til_tunnus
GROUP BY tr.tuote;
```

Toisin kuin edellisessä kyselyssä tässä molemmat tilausrivit saadaan hierarkkisesti asiakkaan alle kuten kuvassa 16. Xmlagg-funktio luo *xml-elementtimetsän* `tilausrivi` muodostaen dokumenttia usealta riviltä löytyvistä xml-elementeistä.

6.2.2 Tallennus merkkijonokenttään

Dokumentin tallennus vaihtuvanmittaiseen merkkijonokenttään (varchar, clob tai blob) on nopeaa, sillä dokumenttia ei tallennusvaiheessa jäsennetä, vaan se on yhtenä

lineaarisena merkkijonoa. Myös kokonaisten dokumenttien hakeminen kannasta on nopeaa. Vastaavasti rakenteeseen perustuvat haut tai dokumentin päivitykset ovat hitaita, sillä jäsennys tehdään tässä vaiheessa.

Lapis (2005) esittää seuraavat kriteerit, joista jonkun täyttyessä merkkijonokenttään tallennus on perusteltua:

- Halutaan tallentaa ja hakea kokonaisia dokumentteja.
- Tallennuksessa ja haussa on suorituskyky ensiarvoisen tärkeää.
- Dokumenttien täytyy pysyä muuttumattomina esimerkiksi laillisuussyitä.
- Digitaalisten allekirjoitusten tulee säilyä.
- Päivityksiä ei tehdä tai tehdään hyvin harvoin.
- Dokumentti on liian monimutkainen jakaa tauluihin.
- Erilaisia skeemoja noudattavien dokumenttien ylläpito halutaan hoitaa yhdessä sarakkeessa.
- Dokumentit identifioidaan ja niitä haetaan muiden kuin sisältöön liittyvien tietojen perusteella.

Merkkijonokenttää tulee välttää, jos tarvitaan rakenteeseen perustuvia hakuja tai haetaan dokumentin osia. Myös skeemojen tulee olla pysyviä, sillä muuten dokumentti joudutaan jäsentämään ja validoimaan skeeman muuttuessa.

Dokumentteihin on mahdollista kohdistaa xquery-kyselyjä, mutta tällöinkin joudutaan tekemään jäsennys, sillä dokumentti ei ole xml-tietomallin mukaisesti kannassa. Hakuaikoja voidaan tehostaa käyttämällä sivutauluja kuten kuvassa 23. Jos dokumenttia joudutaan kuitenkin päivittämään, merkitsee se usein myös sivutaulujen päivittämistä.

6.5 Xml-tietokannat

XML–tietokannan tallennus- ja käsittelymalli on xml-tietomallin mukainen. Luettaessa tiedostoa tietokantaan tai luotaessa tietokannan tiedoista ulkoinen xml-tiedosto ei tarvita muunnosta tietomallista toiseen (Lapis, 2005). Muunnos ei ole aina aivan triviaali toimenpide, ja riskinä on tekstin rakenteen muuttuminen tai tietojen katoamista. Varsinkin dokumenttikeskeisen tekstin tapauksessa on tärkeää ja joskus välttämätöntä säilyttää dokumentin rakenne ja sisältö välimerkkejä myöten alkuperäistä vastaavana. Xml-tietokantaan tallennusta kannattaakin harkita jonkin seuraavista kriteereistä täyttyessä:

- Tarvitaan hyvää xquery/xpath-operaatioiden suorituskykyä.
- Xml-skeema on liian monimutkainen tai niitä on hyvin paljon erilaisia jotta muunnos esimerkiksi relaatiokannan skeemaksi olisi järkevää.
- Erilaisia skeemoja noudattavien dokumenttien ylläpito halutaan hoitaa yhdessä sarakkeessa.
- Dokumenttien sisäinen rakenne kuten järjestys täytyy säilyttää.
- Sovelluksissa käytetään xquery-hakuja, jotka on liian hankala esittää sql:nä.
- Relaatiokantaan viedystä tiedosta on tarve muodostaa uusi alkuperäisestä eroava dokumentti.

Useimpien perinteisten tietokantatoimittajien tuotteissa (Oracle, SQLServer, DB2) on mahdollisuus tallettaa xml-dokumentti xml-tyyppiseen kenttään. Sql-kielen laajennukset *xmlquery*, *xmlexist*, *xmltable* mahdollistavat haut xml-tyyppisestä kentästä ja yhdistämisen relaatiotiedon kanssa. Esimerkiksi xmltable muodostaa xml-dokumentista virtuaalitaulun, jota voidaan käyttää kuten relaatiotauluja (vaikkapa bi-raportoinnissa) eikä dokumenttia tarvitse jakaa relaatiotauluiksi (Lapis, 2005). Tallennettaessa kuvan 16 dokumentti xml-tyyppiseen kenttään, voidaan siitä hakea tilausrivit sql-kyselyllä:

```
CREATE TABLE tilausrivit (tilaukset XML);

SELECT tunnus, nimi, katuosoite, pnro, ptp,
    tr.nimi, tr.maara
FROM XMLTABLE (’$tilaukset/tilaus’
    PASSING tilausrivit.tilaukset AS ”tilaukset” BY REF
    COLUMNS tunnus INTEGER PATH ’@tunnus’,
           nimi    VARCHAR(20) PATH ’asiakas/nimi’,
           katuosoite VARCHAR(20) PATH’asiakas/katuosoite’
           pnro    VARCHAR(5) PATH ’asiakas/pnro’,
           ptp     VARCHAR(20) PATH ’asiakas/ptp’,
           tuote   VARCHAR(20) PATH ’tilausrivi/tuote’,
           maara   DESIMAL(5,2) PATH ’tilausrivit/maara’)
WHERE tunnus='100';
```

Myös xquerylla voidaan tehdä hakuja relaatiokannasta ja yhdistää sitä xml-muotoisen tiedon kanssa. Sekä sql/xml että xquery ovat standardeja xml-kyselykieliä, mutta soveltuvat parhaiten käytettäväksi erilaisissa ympäristöissä (Robie, 2001). Jos tiedot ovat pääasiassa relaatiomuodossa ja tarvitaan sql-kielen monipuolisia ja tehokkaita ominaisuuksia kuten päivityksiä, on sql/xml parempi vaihtoehto. Jos haut kohdistuvat pääasiassa xml-muotoiseen dataan tai on tarve yhdistellä erilaisista lähteistä tullutta tietoa xml-näkymien kautta, on xquery parempi vaihtoehto. Xquery käyttää xml-tietomallia, ja se kehitettiin alun perin yhdistämään erityyppistä dataa.

Varsinkin puolirakenteisen tiedon tapauksessa voi olla hankala päättää, minkä tyyppistä tietokantaa tulisi tiedon tallentamisessa käyttää. Ero perinteisten tietokantojen ja xml-tietokantojen välillä on kuitenkin häviämässä, kun perinteisiin kantoihin lisätään natiivien xml-kantojen ominaisuuksia ja natiivit xml-kannat tukevat dokumentin osien tallentamisen ulkoisiin relaatiotietokantoihin (Bourret, 2005).

7 Yhteenveto

Sekä rakenteisten että rakenteettomien dokumenttien määrä ja jakelutarve on kasvanut niin suureksi, että niiden hallinta tavanomaisin keinoin alkaa olla mahdotonta. Tekstin tuottaminen standardin mukaisesti rakenteisena säästää käsittelyyn ja hallintaan kuluvaa aikaa ja vaivaa. Rakenteisuus, relaatiokannan laajennukset ja xml-tietomalli tuovat mukanaan paljon muitakin etuja, kuten mahdollisuuden yhdistellä tekstiä ja tietokannoissa olevaa relaatiomuotoista tietoa esimerkiksi bi-järjestelmissä. Rakenteisten dokumenttien analysoinnissa voidaan hyödyntää myös *tiedonlouhintaa* (*data mining*), jolloin dokumenttien sisällöstä saadaan vielä enemmän irti.

Koska tietoa tulee olemaan jatkossakin useissa formaateissa ja tallennustarpeet ovat erilaisia, tuovat tietokannat, jotka mahdollistavat usean erityyppisen datan tallennuksen ja käsittelyn sekä niiden yhdistelyn, ratkaisun moneen tässä tutkielmassa esiintuotuun tiedonhallinnan ongelmaan. Myös rakenteeton teksti halutaan jatkossa mukaan tiedon integrointiprosessiin. Tämä voidaan tehdä lisäämällä tekstiin rakenne tai käyttämällä apuna esimerkiksi *tekstinlouhintaa* (*text mining*).

Kun myös toimistotiedostotyypiksi on saatu oma tallennusstandardi, voidaan myös tekstinkäsittelyjärjestelmien puolella ottaa käyttöön tässä tutkielmassa esiintulleita rakenteisuuden hyötyjä. Toimittajien omat tallennusformaatit hävivävät ja dokumentit ovat käytettävissä vuosikymmenienkin päästä.

Viitteet

Al-Khalifa S., Cong Yu, Jagadish H. V. (2003) Querying Structured Text in an XML Database. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (toim. Ives Z.), ACM Press, 4-15.

André J., Furuta R., Quint V. (1989) *Structured Documents*. Campridge University Press, New York, NY, USA.

Bennett M. (2004) *Contrasting Relational and Full-Text Engines*. New Idea Engineering, Inc. - Issue 9 - June 2004. http://ideaeng.com/pub/entsrch/issue09/article01.html (8.5.2007).

Bourret R. (2005) *XML and Databases*. http://www.rpbourret.com/xml/XMLAndDatabases.htm (12.2.2007).

Brown L, Consens M., Davis I., Palmer C, Tompa F (1998) A Structured Text ADT for Object-Relational Databases. *Theory and Practice of Object Systems, Special issue objects, databases, and the WWW*, **4**(4), 227-244.

Burkowski F. (1992) Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. *Proceedings of the 15th Annual International ACM SIGIR conference on Research and development in information* (toim. Belkin N., Ingwersen P, Pejtersen A. M.), ACM Press (112-125).

Clarke C.L.A, Cormack G.V., Burkowski F.J. (1995) An algebra for structured text search and a framework for its implementation. *Computer Journal*, **38**(1), 43-56.

Ditch W. (2007) *XML-based Office Document* Standards. http://www.jisc.ac.uk/media/documents/techwatch/tsw0702pdf.pdf (6.2.2008).

Glushko R.J., McGrath T. (2002) Document Engineering for e-Business. *Proceedings of the 2002 ACM Symposium on Document Engineering* (toim. Munson E.), ACM, 42-48.

Gonnet G., Tompa F. (1987) Mind Your Grammar: a New Approach to Modelling Text. *Proceedings of the 13th international Conference on Very Large Data Bases, Brighton 1987* (toim. Stocker P. M., Kent W.), Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 339-346.

Hiemstra D., Baeza-Yates R. (2008) *Structured Text Retrieval Models*. http://wwwhome.cs.utwente.nl/~hiemstra/papers/eds-structured-models-draft.pdf (18.1.2008).

Junkkaala J. (2008) *Semanttinen web on jo täällä*. TiVi TietoViikko, **26**(7), 10-11.

Kar Y.N. (1996) *The Use of a Combined Text/Relational Database System to Support Document Management*. Technical report CS-96-07, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (Saatavana myös: http://www.cs.uwaterloo.ca/research/tr/1996/07/CS-96-07.pdf, 30.3.2007).

Kay M (2003) *XML Databases*. http://www.xmlstarterkit.com/xmlzone/WP_XML_Databases_E.pdf (12.2.2007).

Kilpeläinen P. (1992) *Tree Matching Problems with Applications to Structured text Databases*. Ph.D. dissertation, Report A-1992-6, Department of Computer Science, University of Helsinki, Helsinki, Finland.

Kuikka E. (1990) *Syntax Directed Text Processing*. Lisensiaattitukimus, Kuopion Yliopisto, Luonnontieteiden ja Ympäristötieteiden tiedekunta, Tietojenkäsittelyoppi.

Kuikka E., Nikunen E. (1994) *Rakenteisten tekstien käsittelyjärjestelmistä.* Report A / 1994 / 4, Department of Computer Science and Applied Mathematics, University of Kuopio, Kuopio, FINLAND (Saatavana myös: http://www.cs.uku.fi/tutkimus/publications/reports/A-1994-4.pdf, 27.3.2007).

Kuikka E., Salminen A. (1995) Filtering structured documents in the SYNDOC environment. *Electronic Publishing* **8**(2&3), 181-193. (http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume8/issue2/2point2.pdf, 25.3.2007).

Kuikka E., Leinonen P., Penttonen M. (1999) *Approach to Document Structure Transformations*. Report A / 1999 / 7, Department of Computer Science and Applied Mathematics, University of Kuopio, Kuopio, FINLAND (Saatavana myös: http://www.cs.uku.fi/tutkimus/publications/reports/A-1999-7.pdf, 5.12.2006).

Laamanen H. (2005) *XML-verkkotietokantojen suorituskykyvertailu*. Diplomityö, Lappeenrannan Teknillinen Yliopisto, Tietotekniikan osasto.

Lapis G. (2005) *XML and Relational Storage–Are they mutually exclusive?* http://idealliance.org/proceedings/xtech05/papers/02-05-01/ (14.2.2007).

Loeffen A. (1994) Text Databases: A Survey of Text Models and Systems. *ACM Sigmod Record*, **23**(1), 97-106.

Macleod I., Reuber A. (1987) The Array Model: A Conceptual Modeling Approach to Document Retrieval. *Journal of the American Society for Information Science*, **38**(3), 162-170.

Macleod I., Reuber A. (1990) Storage and Retrieval of Structured Documents. *Information Processing & Management*, **26**(2), 197-208.

Macleod I. (1991) Text Retrieval and the Relational Model. *Journal of the American Society for Information Science*, **42**(3), 155-161.

Magnani M., Montesi D. (2004) *A Unified Approach to Structured, Semistructured and Unstructured Data*. Technical Report UBLCS-2004-9, Department of Computer Science, University of Bologna, Italy. (Saatavana myös: http://www.cs.unibo.it/pub/TR/UBLCS/2004/2004-09.pdf , 28.2.2008).

Meier W. (2002) *eXist: An Open Source Native XML Database.* http://exist-db.org/webdb.pdf#search=%22structured%20text%2Binner%20storage%20architecture%22 (9.9.2006).

Nikulainen K. (2007) *Tallennusmäärät kasvavat rajusti*. ITviikko, **8**(17), 12-13.

Postel J. (1982*) Simple Mail Transfer Protocol*. http://rfc.sunsite.dk/rfc/rfc821.html (28.2.2008).

Raymond R, Tompa F., Wood D. (1995) From Data Representation to Data Model: Meta-semantic Issues in the Evolution of SGML. *Computer Standards & Interfaces*, **18**(1), 25-36.

Robie J. (2001) *SQL/XML, Xquery and Native XML Programmin Languages*. http://www.xquery.com/docs/native_xml_programming.pdf (22.4.2008).

Sacks-Davis R., Arnold-Moore T., Zobel J. (1994) *Database Systems for Structured Documents*. http://www.pms.ifi.lmu.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/sacks94db.pdf (30.3.2007).

Salminen A. (1992) *Rakenteisen tekstin hallinta*. Tietojenkäsittelytieteen julkaisuja, Opetusmonisteita OM-3, Jyväskylän Yliopisto, Tietojenkäsittelyopin Laitos.

Salminen A., Watters C. (1992) A Two-Level Structure for Textual Databases to Support Hypertext Access. *Journal of the American Society for Information Science*, **43**(6), 432-447.

Salminen A., Tompa F.W. (1999) Grammars++ for Modelling Information in Text. *Information Systems*, **24**(1-24) (Saatavana myös: http://www.cs.uwaterloo.ca/research/tr/1996/40/CS-96-40.pdf, 30.3.2007).

Salminen A., Tompa F.W. (2001) Requirements for XML Document Database Systems. *Proceedings of the 2001 ACM Symposium on Document Engineering* (toim. Munson E.), ACM, 85-94 (Saatavana myös: http://www.cs.uwaterloo.ca/~fwtompa/.papers/xmldb-desiderata.pdf, 16.8.2007).

Salminen A. (2006) *Jakso 1: XML pähkinänkuoressa*. http://users.jyu.fi/~airi/opetus/xml/xml-kieli/xml-jakso1.pdf (6.11.2006).

Salton G., Buckley C., Allan S. (1992) Automatic structuring of text files. *Electronic Publishing*, 5(1), 1-17.

Suciu D. (2001) On Database Theory and XML. *SIGMOD Recrod*, **30**(3), 39-45 (Saatavana myös: http://xml.coverpages.org/SuciuDatabaseTheoryXML.pdf, 13.2.2007).

Tietosuojalaki (2008) *Hallituksen esitys Eduskunnalle laiksi sähköisen viestinnän tietosuojalain muuttamisesta/HE 48/2008*. http://www.finlex.fi/fi/esitykset/he/2007/20070158 (10.6.2008).

Trotman A. (2003) Searching Structured Documents. *Information Processing and Management: an International Journal*, **40**(4), 619-632.

Virk R. (2002) *"Why Convert Documents into XML?"*. http://www.cambridgedocs.com/resources/whitepapers/id35.htm (2.5.2007).

XML Best Practices (2006) *SQL Server 2005 Books Online*. http://microsoft... (31.8.2007).

Yoshikawa M, Kato H., Kinutani H. (1999) Design Framework of a Database for Structured Documents with Object Links. *IEICE Ttrans. Inf. & Syst. Special Issue on New Generation Database Technologies*. E82-D(1), 147-155 (Saatavana myös: http://electric.ee.psu.ac.th/ee/ieice/1999/pdf/e82-d_1_147.pdf, 22.2.2008).

Young-Lai M. (2001) *Text Structure Recognition using Region Algebra*. Technical Report Number: CS-2001-06, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

Zhang C., Naughton J., DeWitt D., Luo Q., Lohman G. (2001) On Supporting Containment Queries in Relational Database Management Systems. *Proceedings of the 2001 ACM SIGMOD international conference on Management of data* (toim. Mehotra S.), ACM, 425-436.

Liite 1: XML-tiedon tallennus – valintakriteerit

Ratkaisu	Soveltuvuus	Plussat	Miinukset
Ulkoinen xml-tiedosto	- Arkistokäyttö - Tiedonsiirto - Webbisivut	- Siirrettävää (unicode) - Itsensäkuvaavaa - Voi kuvata tiedon puuna tai verkkona. - Standardi metakieli. - Linkit kaikentyyppisiin tiedostoihin. - Luettavissa sellaisenaan.	- Usean käyttäjän hallinta - Turva- ja toipumisominaisuudet. - Monisanaisuus (verbose) - Hitaat haut - Päivitykset hankalia.
Relaatiokanta/blob, clob tai varchar	- Vaikeasti ositettavien dokumenttien talletus kun xml-tietotyyppiä ei käytössä. - Dokumenttikeskeiset dokumentit. - Kun tarvitaan pelkästään säilytysalustaa. - Kokonaisten dokumenttien haku.	- Yksinkertainen ratkaisu - Dokumentin alkuperäinen muoto säilyy. - Relaatiokantojen yleisyys	- Rakenteeseen kohdistuvat haut hankalia ja hitaita. - Relaatiodataan yhdistäminen hankalaa.
Relaatiokanta/tauluihin jako	- Datakeskeiset dokumentit, jotka helppo jakaa tauluihin. - Tarve yhdistää relaatiodatan kanssa.	- Hallintaominaisuudet - SQL-kieli - Liittymät tuotantosovelluksiin ja raportointijärjestelmiin - Relaatiokantojen yleisyys	- Hierarkkisen rakenteen hallinta hankalaa. - Järjestyksen hallinta vaikeaa.
Xml-kanta/xml-tietotyyppi	- Tieto on harvaa - Rakenne ei tiedossa - Rakenne voi muuttua - Rakenne on hierarkkista. - Järjestys tärkeää	- Rakenteeseen perustuvat haut nopeita. - Päivitys helppoa - Validointi mahdollista - Mahdollisuus tarkistaa, että dokumentti on hyvin muodostettu. - Yhdistäminen relaatiodatan kanssa helpohkoa. - Indeksointiominaisuus - Mahdollisuus tehdä SOAP-, ADO-, OLE DB–liittymiä.	
Xml-tietotyyppi + tyyppitaulut	Samat kuin relatiomalliin ja xml-kanta/xml-tietotyyppiratkaisussa, mutta lisäksi dokumenttikokoelmien hallinta.		