Spatiaaliset tietokannat

Riku Koffert

15.5.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu –tutkielma

TIIVISTELMÄ

Spatiaaliset tietojärjestelmät ja spatiaalista tietoa käsittelevät ohjelmistot ovat yleistyneet perinteistä tietoa tallentavien tietokantojen rinnalla navigaatio- sekä kartoitusjärjestelmien yleistyessä. Tässä tutkielmassa perehdytään spatiaalisen tiedon käsittelyn perusteisiin, spatiaalisiin tietomalleihin, spatiaalisten järjestelmien erikoispiirteisiin suhteessa perintei-siin tietokantajärjestelmiin sekä esitetään käytännön esimerkkejä spatiaalisen tiedon tallentamisesta sekä analysoinnista Oracle Spatial –järjestelmän avulla SQL-kyselykielen sekä Java-ohjelmointirajapinnan esimerkkien kautta.

ACM-luokat: (ACM Computing classification system, 1998 version): H.2.8

Avainsanat: spatiaalisuus, tietokantajärjestelmät, SQL, Java

Sisällysluettelo

1. Johdanto 1
2. Spatiaalisen tiedon varastointi 3
   2.1. Spatiaalisen tiedon peruskäsitteet ja mallit 3
      2.1.1 Spatiaalisen tiedon geometriatyypit 4
      2.1.2 Oracle Spatial -järjestelmän geometriatyypit 5
   2.2. Spatiaalinen tietomalli 6
      2.2.1 Objektien joukon geometrian kuvaminen 7
      2.2.2 Spagettimalli 8
      2.2.3 Verkkomalli 9
      2.2.4 Topologinen malli 10
      2.2.5 Spatiaaliset hierarkiat 12
      2.2.6 Käsitteellisen skeeman muuntaminen loogiseksi skeemaksi 15
      2.2.7 Oracle Spatialin hierarkkinen tietorakenne 17
   2.3. Indeksointi ja spatiaaliset tiedonsaantimenetelmät 19
   2.4. R-puu-indeksointi 22
      2.4.1 R-puun laatu 26
   2.5. Indeksointi Oracle Spatial -järjestelmässä 27
   2.6. Spatiaalisen tiedon tallennus Oracle Spatial -järjestelmässä 27
      2.6.1 Spatiaalisen tietomallin luonti Oracle Spatial -järjestelmässä 28
      2.6.2 Tietokantataulun luonti 28
3. Spatiaalisen tiedon hakeminen 31
   3.1. Spatiaalisen tiedon hakumenetelmät 32
   3.2. I/O-algoritmit 33
   3.3. Spatiaaliset liitokset 35
   3.4. Monimutkaiset spatiaaliset kyselyt 37
      3.4.1 Kyselysuunnitelma 37
      3.4.2 Ongelmat spatiaalisten hakujen käsittelyssä 38
   3.5. Spatiaaliset tiedonhakumenetelmät dynaamisissa järjestelmissä 39
   3.6. SQL-lausekkeet spatiaalisen tiedon hakemisessa 40
      3.6.1 Yksinkertaiset hakulauseet 41
      3.6.2 Spatiaaliset liitokset 47
   3.7. Hakulauseet Java-rajapinnan kautta Oracle Spatial –järjestelmässä 49
4. Yhteenveto 55

Viitteet 57

Liite 1 – SdoPrint-ohjelman lähdekoodi 60

1. Johdanto

Spatiaaliset tietokannat sekä niihin tallennetun sijaintitiedon käyttäminen joko analysoinnissa, päätöksenteossa tai paikannuksessa ovat yleistynyt kuluneen vuosikymmenen aikana. Tässä tutkielmassa perehdytään spatiaalisen tiedon käsittelyn perusteisiin, spatiaalisten järjestelmien erikoispiirteisiin suhteessa perinteisiin tietokantajärjestelmiin sekä esitetään käytännön esimerkkejä Spatiaalisen tiedon tallentamisesta sekä analysoinnista Oracle Spatial –järjestelmän (Oracle, 2003; Kothuri & al., 2004) avulla.

Luvussa 2 käsittelemme tarkemmin spatiaalisen tiedon peruskäsitteitä, kuten geometriaa ja spatiaalisten tietokantojen yhteydessä käytettäviä tietomalleja (Rigaux & al., 2005; Malinowski & Zimányi, 2005) sekä käsitteellisellä että konkreettisella tasolla. Tutustumme myös spatiaalisiin tietomalleihin, spatiaalisen tiedon indeksointiin sekä yleisesti spatiaalisissa järjestelmissä indeksirakenteena käytettyihin R-puihin. Luvun lopussa perehdytään tarkemmin spatiaalisen tietokannan luomiseen (Oracle, 2003) ja tiedon tallentamiseen Oracle Spatial -järjestelmässä esitettyjen esimerkkien avulla.

Luvussa 3 perehdymme spatiaalisen tiedon hakemiseen tietokannasta (Rigaux & al., 2003; Oracle, 2003). Luvun alussa tutustutaan spatiaalisten hakulauseiden käsittelyyn, sekä tutustutaan lyhyesti spatiaalisiin liitoksiin, jotka muodostavat spatiaalisten hakujen ytimen, ja spatiaalisessa tiedonkäsittelyssä käytettyihin algoritmeihin sekä I/O-algoritmien että hakualgoritmien osalta. Lisäksi käsitellään monimutkaisempien spatiaalisten hakulauseiden yhteydessä oleellisia kyselysuunnitelmia, sekä yleisimpiä spatiaaliseen tiedonhakuun liittyviä ongelmia ja spatiaalisten hakujen käsittelyä dynaamisissa ympäristöissä, kuten paikannusjärjestelmissä. Luvun loppupuolella esitetään esimerkkejä spatiaalisista hakulauseista Oracle Spatial -ympäristössä, sekä tutkitaan spatiaalista tiedonhakua ohjelmointirajapinnan (Kothuri & al., 2004) kautta Java-ympäristössä. Spatiaalisen tiedon visualisointi varsinaisissa sovelluskohteissa ja ohjelmistoissa on rajattu tämän tutkielman aihealueen ulkopuolelle.

Luvussa 4 luomme lopuksi yhteenvedon tutkielman keskeisistä kohdista sekä spatiaalisen tiedonkäsittelyn jatkotutkimusmahdollisuuksista ja tulevaisuudennäkymistä. Tässä viimeisessä luvussa esitetään myös lyhyesti tutkielman ulkopuolelle jätettyjä tekniikoita.

2. Spatiaalisen tiedon varastointi

Sijaintitietoa varastoidaan tietokantoihin perinteisen numeraalisen tiedon tapaan, mutta spatiaalisen tiedon tallentaminen vaatii tallennusjärjestelmältä, tietokantaskeemalta ja tietokannanhallintajärjestelmältä tiettyjä ominaisuuksia, jotka mahdollistavat spatiaalisen tiedon tehokkaan käsittelyn. Tässä luvussa perehdytään tarkemmin spatiaalisen tiedon peruskäsitteisiin, sen yhteydessä käytettäviin tietomalleihin ja spatiaalisen tiedon tallentamiseen liittyviin seikkoihin. Luvun loppupuolella esitetään myös esimerkkejä spatiaalisen tietokannan luomisesta ja tiedon tallentamisesta Oracle Spatial -järjestelmässä.

2.1. Spatiaalisen tiedon peruskäsitteet ja mallit

Avaruudellisia sekä ajallisia ominaisuuksia kuvaavat standardisoidut skeemat mahdollistavat tehokkaamman tiedon jakamisen ja siirron eri sovellusten ja järjestelmien välillä (OGC, 2003). Näitä skeemoja käyttävät sekä spatiaalista tietoa käsittelevät järjestelmät, että geograafisten informaatiojärjestelmien kehittäjät ja käyttäjät. Skeemojen avulla pyritään takaamaan ymmärrettävien ja oikeiden spatiaalisten tietojen saatavuus.

Geometria antaa mahdollisuudet kvantitatiiviseen kuvaamiseen koordinaattien ja matemaattisien funktioiden avulla. Näin voidaan esimerkiksi kuvailla erilaisia ominaisuuksia kuten ulottuvuuksia, sijaintia, kokoa, muotoa ja asentoa. Objektin geometrian kuvaamiseen käytettävät matemaattiset funktiot riippuvat spatiaalista sijaintia kuvaavasta koordinaattien viitejärjestelmästä. Geometria on siis ainoa muuttuva osa siirrettäessä informaatiota yhdestä viittaus-tai koordinaattijärjestelmästä toiseen.

Topologia käsittelee niitä geometristen muotojen ominaisuuksia, jotka säilyvät muuttumattomina, jos tilaa muutetaan joustavasti ja toistuvasti, kuten esimerkiksi silloin kun geograafista tietoa muunnetaan koordinaattijärjestelmien välillä. Geograa-

fisen tiedon kontekstissa topologiaa käytetään yleensä kuvaamaan n-ulottuvuudellisten graafien yhdistettävyyttä. Tämä ominaisuus on muuttumaton graafin muutoksista huolimatta. Laskennallinen topologia antaa tietoa niiden geometristen primitiivien liitettävyydestä, jotka voidaan johtaa näitä primitiivejä sisältävistä geometrioista.

2.1.1 Spatiaalisen tiedon geometriatyypit

Spatiaalisessa informaatiossa geometria on järjestetyistä kärkipisteistä koostuva sekvenssi, jotka on yhdistetty joko suorilla tai kaarevilla viivoilla (Oracle, 2003). Geometrian tyyppi määrittelee sen merkityksen. Spatiaaliset tietokannat tukevat useita primitiivisiä geometrioita, sekä näistä primitiiveistä koottuja kokoelmia. OGC:n referenssimallin (OGC, 2003) määritelmän mukaan geometrinen objekti on koordinaattigeometrian sekä koordinaattiviitejärjestelmän yhdistelmä, joukko *suorilla sijainneilla* (direct positions) ilmaistuja geometrisia pisteitä. Suora sijainti viittaa sijaintiin koordinaattien viitejärjestelmän sisällä.

Spatiaalisen tiedon kuvaamiseen käytettävien geometrioiden tyypit riippuvat käytettävän järjestelmän toteutuksesta, mutta perusperiaatteiltaan ne noudattavat ISO-19107 standardissa määriteltyjä geometriatyyppejä (ISO, 2003). ISO-19107 standardissa määritellään yhdenmukaistavat luokat 1-, 2- ja 3-ulotteisille primitiivisille geometriatyypeille.

Kaksiulotteisia objekteja tai pintoja sisältäviä objekteja käytetään useimmiten kuvattaessa entiteettejä, jotka sisältävät suuria yhtenäisiä alueita (Rigaux & al., 2002). Tällaisia kuvattavia kohteita ovat esimerkiksi paikannustietokantoihin tallennettavat maaalueet, tiet sekä muut geograafiset elementit. *Polygoni* on tärkein geometrinen tyyppi kuvattaessa edellä mainitun kaltaisten objektien geometriaa. Polygoni on *murtoviivalla* (polyline) rajattu alue tietyllä *tasolla* (plane). Tätä polygonia ympäröivää murtoviivaa kutsutaan sen *rajaksi* (boundary).

2.1.2 Oracle Spatial -järjestelmän geometriatyypit

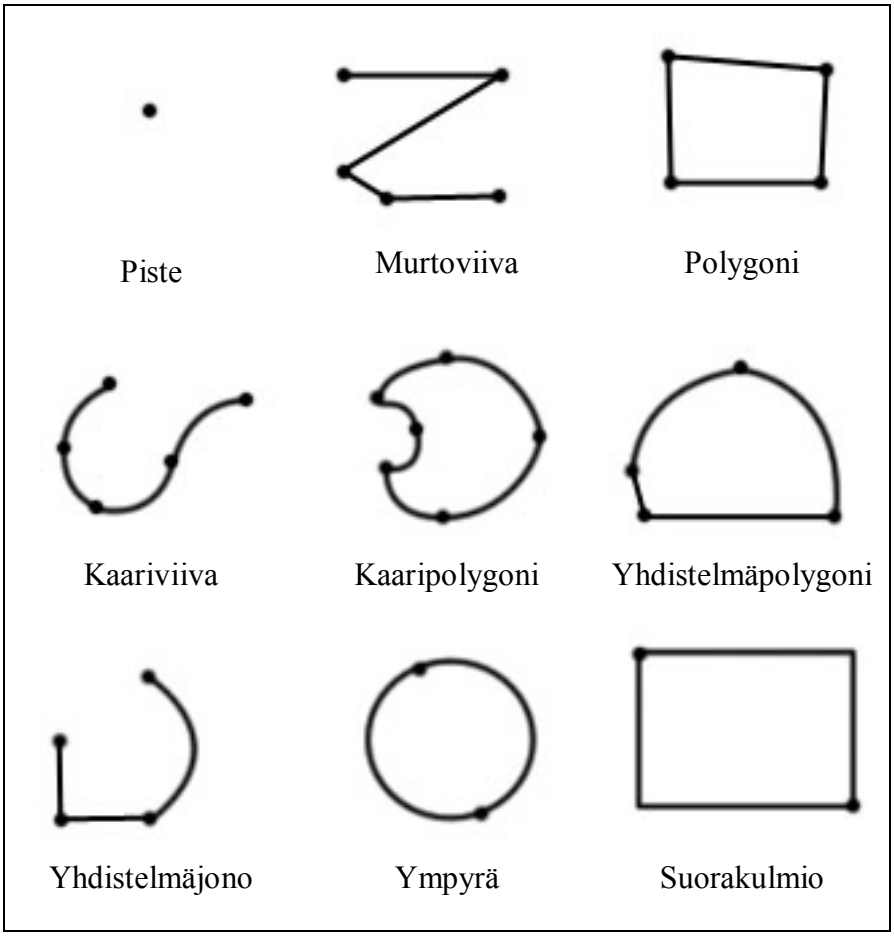
Oracle Spatial (Oracle, 2003) tukee useita erilaisia primitiivisiä geometriatyyppejä sekä näistä koottuja kokoelmia. Kaksiulotteiset pisteet muodostuvat X- ja Y-koordinaattiparista, jonka avulla ilmaistaan yleensä pituus ja leveysasteiden arvoja. Viivajonot muodostuvat yhdestä tai useammasta pisteparista, jotka määrittelevät viivasegmentin. Polygonit muodostuvat yhdistetyistä viivajonoista, jotka muodostavat suljetun kehän. Näiden primitiivisten tyyppien avulla kuvataan varsinaista spatiaalista tietoa. Esimerkiksi polygonia voidaan käyttää kuvaamaan paikoitusaluetta, tai pistettä kuvaamaan rakennuksen sijaintia.

Esimerkkejä Oracle Spatialin (Oracle, 2003) tukemista primitiiveistä sekä näistä koostuvista geometrioista ovat esimerkiksi 2-ulotteiset geometriatyypit:

- pisteet ja pisteryhmät
- murtoviivat
- polygonit
- kaariviivat
- kaaripolygonit
- yhdistelmäpolygonit
- kaarimurtoviivat
- ympyrät
- optimoidut suorakaiteet.

Oracle Spatial tukee myös kolmi- ja neliulotteisia geometrisia tyyppejä, joissa käytetään kolmea tai neljää koordinaattia kuvaamaan kutakin objektin pintaa. Useimmat spatiaaliset funktiot toimivat kuitenkin ainoastaan kaksiulotteisille spatiaalisille objekteille Oracle Spatial -järjestelmässä (Oracle, 2003). Kuvassa 1 on

esitetty esimerkkejä useimmista tavallisista spatiaalisissa järjestelmissä käytettävistä 2-ulotteisista geometriatyypeistä.



Kuva 1: Esimerkkejä 2-ulotteisista geometriatyypeistä.

2.2. Spatiaalinen tietomalli

Tietomalli voidaan määritellä yleisen tason kuvaukseksi joukosta entiteettejä, sekä näiden joukkoon kuuluvien entiteettien välisistä suhteista (Peuquet & Marble, 1990). Entiteetillä tarkoitetaan yksilöitävissä olevaa objektia, kuten vaikkapa spatiaalisissa tietojoukoissa esiintyvät tiet, rakennukset ja maa-alueet. Entiteettikokoelmat ovat kokoelma entiteettejä, joilla on yhteisiä piirteitä tai jokin yhdistävä tekijä. Spatiaalisessa tiedossa tällaisia kokoelmia ovat esimerkiksi kaupungit ja vuoristot. Tietomalliin tallennettavat suhteet sisältävät tietoa objektien välisistä etäisyyksistä, vierekkäisyydestä ja muista vastaavista tiedoista. Sekä entiteeteillä että niiden välisillä

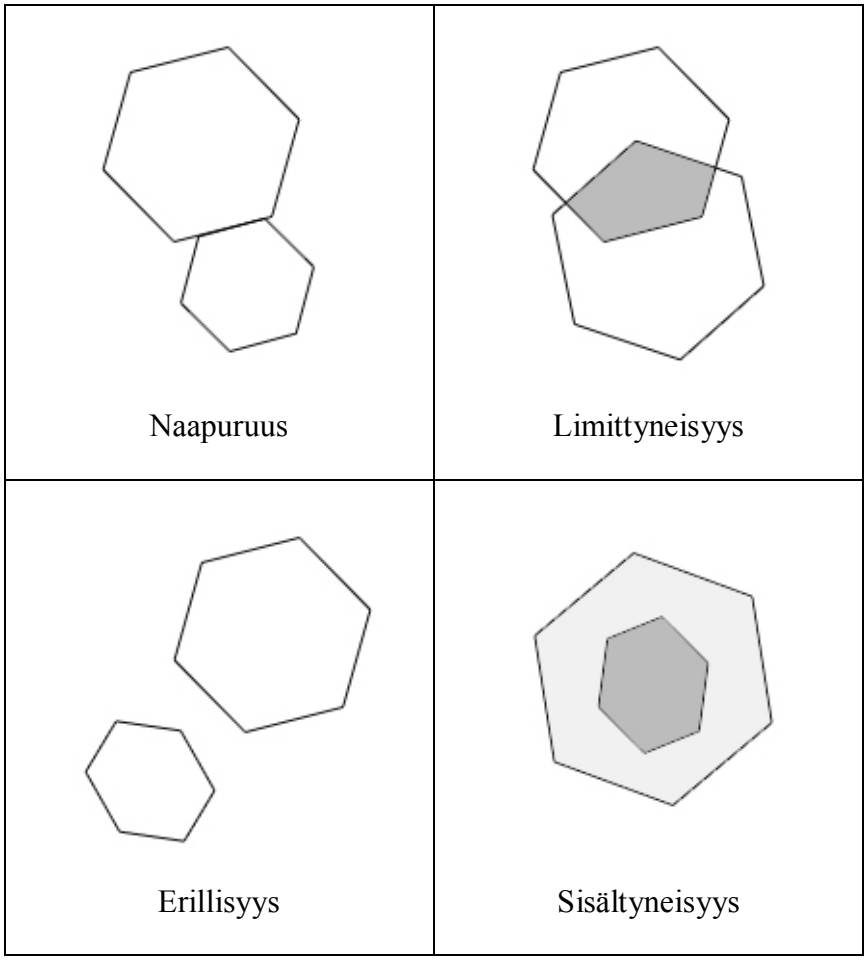
suhteilla voi olla monia erilaisia arvoja tai ominaisuuksia, kuten leveys, korkeus tai syvyys. Tietomallin peruspiirre on se, että malli on abstraktio todellisuudessa olemassaolevista objekteista ja näiden relaatioista, ja jokainen tietomalli kuvaa todellisuutta tietyllä tarkkuustasolla. Monet tietomallien suunnittelijat ovat ymmärtäneet, että tallennettua tietoa tulisi olla mahdollista tarkastella monella eri tasolla. Ei kuitenkaan ole olemassa yleisesti hyväksyttyä mallia siitä kuinka moneen abstraktiotasoon tietomalli tulisi erotella. Esimerkkinä voidaan mainita nelitasoinen malli (Peuquet & Marble, 1990), jossa abstraktio jaetaan seuraavalla tavalla :

- Todellisuus : Tilanne joka vallitsee todellisessa maailmassa sisältäen kaikki tarkasteltaessa ihmiselle näkyvät sekä näkymättömät osat.
- Tietomalli: Abstraktio todellisuudesta, johon sitältyvät vain ne osat, joita pidetään merkittävinä sovelluksen tai sovelluskohteen toiminnan kannalta.
- Tietorakenne: Tietomallin esitysmuoto, joka kuvataan yleensä listoina, kaavioina tai muina vastaavina kokoelmina, jotka voidaan tallentaa tietokoneen ymmärtämässä muodossa.
- Tiedostorakenne : Tiedon esitysmuoto fyysisellä tallennusmedialla.

2.2.1 Objektien joukon geometrian kuvaminen

Spatiaalisen tiedon kokoelmien kuvaamiseen käytettävistä malleista voidaan eritellä yleisimmät kolme mallia: *spagetti*-, *verkko*- ja *topologiamallit* (Rigaux & al., 2002). Nämä mallit eroavat toisistaan pääasiassa topologisten suhteiden kuvaustavan osalta. Objektien väliset topologiset suhteet ovat niitä suhteita, jotka pysyvät muuttumattomina topologian muuttuessa. Nämä suhteet siis säilytetään, kun spatiaalisia objekteja muutetaan esimerkiksi kääntämällä tai skaalaamalla niitä euklidisella tasolla. Topologisia suhteita ovat mm. *naapuruus*-, *limittyneisyys*-, *erillisyys*- ja *sisältyvyyssuhteet*, jotka muodostavat tärkeän luokan spatiaalisia riippuvuuksia. Tällaisten relaatioiden eksplisiittinen kuvaus spatiaalisessa tietokannassa lisää käytettävissä olevaa spatiaalista tietoa ja on hyödyllistä esimerkiksi arvioitaessa

spatiaalisia tietokantakyselyitä. Kuvassa 2 esitetään muutamia mahdollisia geometristen objektien välisiä topologisia suhteita.



Kuva 2: Esimerkkejä objektien välisistä topologisista suhteista.

2.2.2 Spagettimalli

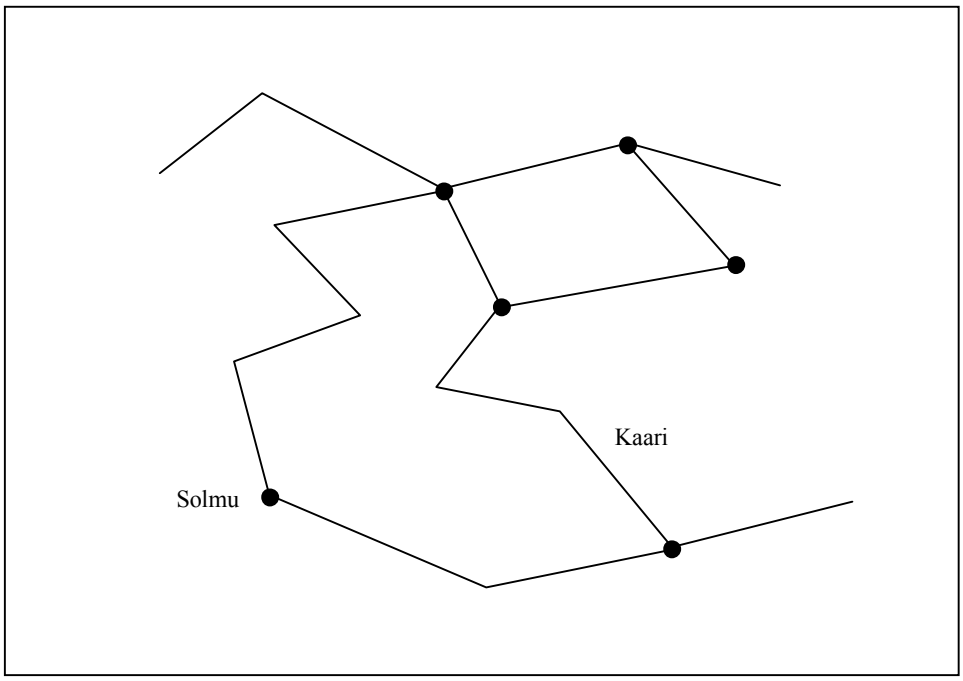
Spagettimallissa minkä tahansa kokoelmaan liittyvän spatiaalisen objektin tiedot kuvaillaan itsenäisesti riippumatta muista kokoelmaan liittyvistä objekteista (Rigaux & al., 2002). Tämän kaltaiset mallit eivät tallenna kokoelmaan liittyvää topologista tietoa, joten kaikki topologiset riippuvuudet tulee laskea suorituksen aikana niitä tarvittaessa. Tämä rakenne antaa myös ymmärtää, että malli sisältää ylimääräistä tietoa. Esimerkkinä kahden vierekkäisen alueen välistä rajaa kuvaava tieto esitetään kahdesti. Tämä yksinkertainen malli mahdollistaa heterogeenisen esitystavan, jonka

avulla pisteet, murtoviivat ja alueet voidaan sekoittaa yhteen kokoelmaan ilman rajoituksia.

Tämän lähestymistavan suurin hyöty on sen yksinkertaisuus. Malinowskin ja Zimányin (2005) mukaan mallin käyttökelpoisuutta lisää myös se, ettei käyttäjän tarvitse tietää uutta tietokantaobjektia lisättäessä mitään tietokannassa olevien tai sinne lisättävien objektien topologisista suhteista. Tämän vuoksi tiedon lisääminen onnistuu huomattavasti yksinkertaisemmin kuin esimerkiksi topologisessa mallissa. Toisaalta mallin suurimmat puutteet johtuvat eksplisiittisen relaatiotiedon puutteesta kokoelman objektien välillä, kuten objektien vierekkäisyys tai tieto siitä sisältääkö objekti jonkin toisen kokoelmaan kuuluvan objektin (Rigaux & al., 2002). Tämän takia mallissa ei ole esimerkiksi suoraviivaista tapaa selvittää sitä jakavatko kahden polygonin rajat saman pisteen. Edellä mainittu tiedon ylimäärä on toinen mahdollinen ongelma, jonka merkitys kasvaa käsiteltävän paikannustiedon määrän kasvaessa.

2.2.3 Verkkomalli

Verkkotyylinen spatiaalinen malli kehitettiin alunperin kuvaamaan verkkoja verkko-graafeihin perustuvissa ohjelmistoissa kuten liikenneyhteysohjelmistoissa (Rigaux & al., 2002). Tätä mallia käytettäessä tallennetaan kokoelmiin topologiset relaatiot pisteiden ja murtoviivojen välillä. Mallissa käytettävien geometristen tyyppien joukko on hieman monimutkaisempi kuin spagettimallissa. Mallin käyttöä varten tarvitaan kaksi uutta käsitettä jotka ovat solmut ja kaaret. Solmu on yksilöllinen piste, johon yhdistyy vaihteleva määrä kaaria. Kaari on murtoviiva joka alkaa ja päättyy solmuun. Kuvassa 3 esitetään yksinkertainen verkkomallin mukainen tietojoukko.



Kuva 3: Verkkomallin mukainen kuvaus topologiasta.

Toteutustavasta riippuen verkosto on joko planaarinen tai ei-planaarinen. Planaarisessa verkossa jokainen sivujen risteys tallennetaan solmuna, vaikka kyseinen solmu ei vastaisikaan geograafista objektia. Ei-planaarisessa verkossa sivut voivat ylittää toisensa muodostamatta risteystä. Esimerkkinä ei-planaarisista verkoista voidaan mainita esimerkiksi maakuljetusverkot, jotka sisältävät tunneleita ja tasoristeyksiä.

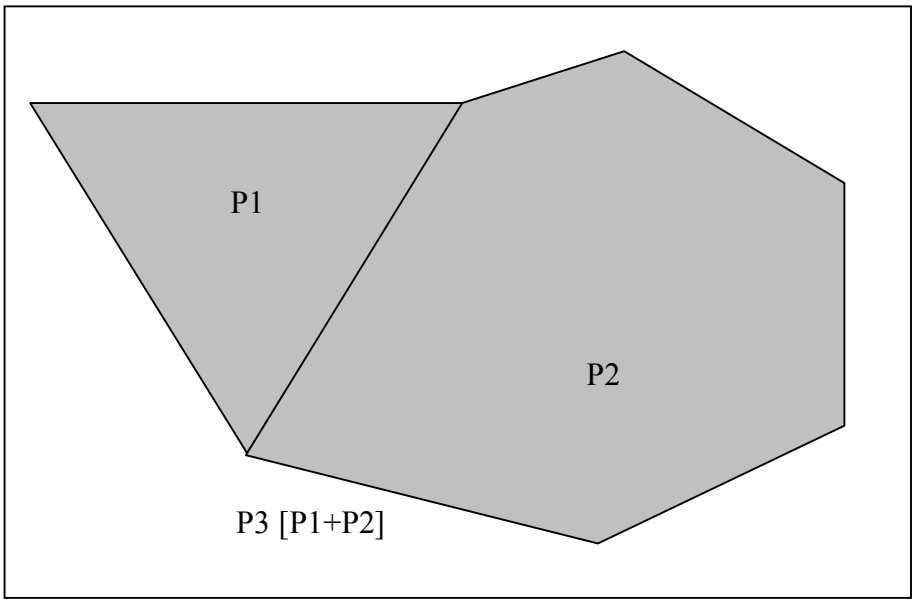
Alue kuvataan verkkomallissa verteksien järjestettynä listana polygonin rajaamalla alueella, eli samalla tavalla kuin spagettimallissa. Tämän mallin käytön hyviä puolia ovat sen esittämä luontainen kuvaus verkon muotoisesta topologiasta sekä yhteneväisyyskäsite, joiden avulla on esimerkiski mahdollista etsiä optimaalisia polkuja verkon läpi. Tämä malli ei tallenna 2-ulotteisten objektien välistä relaatiotietoa.

2.2.4 Topologinen malli

Topologinen malli on hyvin samankailtainen verkkomallin kanssa, mutta mallissa käytetty verkosto on planaarinen (Rigaux & al., 2002). Tämä verkosto aiheuttaa

planaarisen jakautumisen vierekkäisiksi polygoneiksi, joista jotkut eivät välttämättä vastaa todellista geograafista objektia. Verkkomallin mukaisesti solmu kuvaillaan topologisessa mallissa pisteenä, joka koostuu joukoista kaaria. Jos lista on tyhjä, on solmu eristyksissä muista verkon osista. Verkosta eristyneitä pisteitä käytetään kuvattaessa erilaisten objektien sijainteja kuten tornien tai toriaukioiden sijaintia. Kaaret sisältävät alku-ja loppupisteidensä lisäksi tiedon niistä polygoneista, joille kaari toimii yhteisenä osana.

Malli kuvaa polygonin kaarien joukkona, joista kukin kaari on jaettuna viereisen polygonin kanssa. Tehokkuuskysymyksien vuoksi mallissa esiintyy hieman tiedon ylimäärä näiden tietojen osalta, esimerkiksi polygoneja voidaan käsitellä joko polygoneina tai kaarina. Tallennetussa sijantitiedossa ei sen sijaan esiinny ylimäärää, koska kukin piste tai viiva tallennetaan vain kerran. Malinowskin ja Zimányin (2005) mukaan topologinen malli on huomattavasti monimutkaisempi ja joustamattomampi kuin spagettimalli, sekä vaatii enemmän tallennustilaa massamuistilta tietorakenteen tallentamiseen. Kuvassa 4 esitetään esimerkki polygonien esitystavasta topologisessa mallissa.



Kuva 4: Polygonien esitystapa topologisessa mallissa.

Eräs topologisen mallin eduista on topologisten kyselyiden nopea laskettavuus. Esimerkiksi kaikkien polygonin *P* viereisten polygonien etsiminen on suoraviivaista, koska voidaan suorittaa haku kaarista, jotka muodostavat polygonin *P*:n naapureina. Lisäksi tällaiseen malliin perustuvan tietojoukon säännöllinen päivittäminen ja ylläpitäminen on helpompaa objektien jakaman tiedon takia, toisin kuin esimerkiksi spagettimalliin perustuvassa tiedossa, jossa kahdelle polygonille yhteinen sivu tallennetaan kahteen kertaan. Tästä johtuen myös tietoa päivitettäessä muutokset tulee tehdä molempiin tietueisiin.

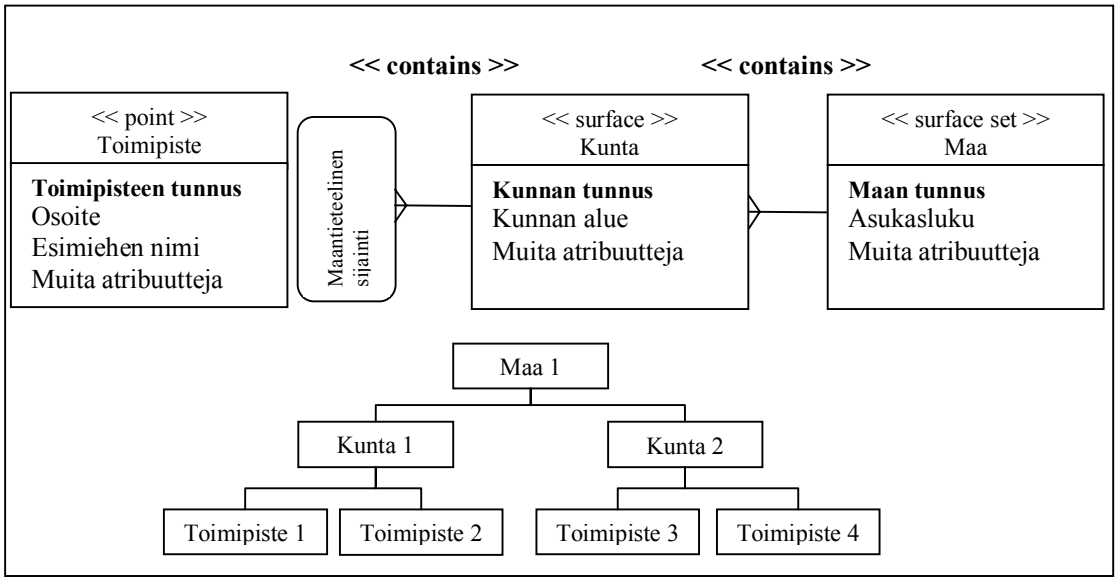
Topologisen mallin käytössä on muutamia ongelmia. Ensinnäkin osalla tietokannan spatiaalisista objekteista ei välttämättä ole semanttista vastinetta reaalimailman kanssa. Toinen ongelma on rakenteen monimutkaisuus, joka saattaa hidastaa joitain tiedon käsittelyyn liittyviä operaatioita. Monimutkaisten tietorakenteiden läpikäynti toistuvasti voi aiheuttaa suurta käsittelykuormaa. Lisäksi uuden objektin lisääminen tietojoukkoon edellyttää planaarisen graafin osittaista uudelleen laskemista.

2.2.5 Spatiaaliset hierarkiat

Malinowskin ja Zimányin (2005) mukaan yksinkertaiset spatiaaliset hierarkiat ovat sellaisia, joiden jäsenet voidaan kuvata käsitteellisessä mallissa puurakenteena. Nämä yksinkertaiset hierarkiat voidaan jakaa kolmeen alakategoriaan, jotka ovat symmetriset, epäsymmetriset ja yleistetyt spatiaaliset hierarkiat.

Symmetrisen spatiaalisen hierarkian skeemassa on ainoastaan yksi polku, jonka kaikki tasot ovat pakollisia. Esiintymätasolla rakenteen elementit muodostavat puurakenteen, jonka kaikki oksat ovat saman pituisia. Jokaisella vanhemmalla tulee olla vähintään yksi lapsisolmu, ja kukin lapsisolmu voi kuulua ainoastaan yhdelle vanhemmalle.

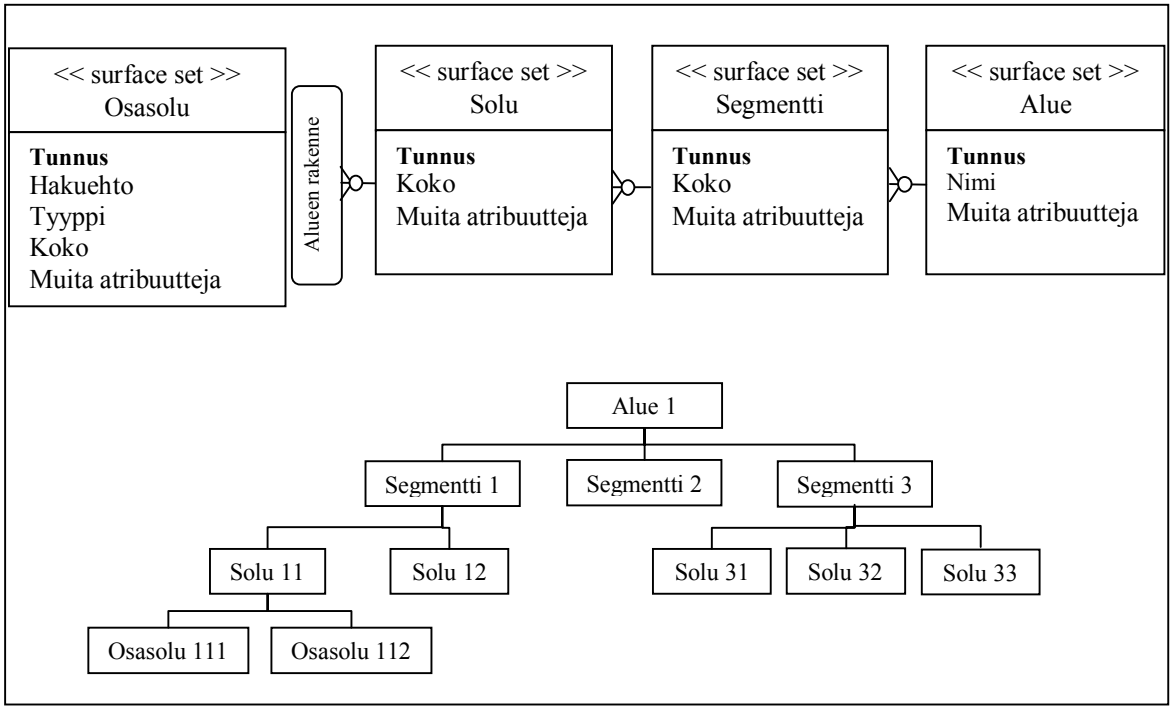
Kuvassa 5 esitetään symmetrisen hierarkian skeema sekä esimerkki skeeman mukaisesta puurakenteesta[1].



Kuva 5: Symmetrisen hierarkian skeema ja esimerkki puurakenteesta (Malinowski & Zimányi, 2005).

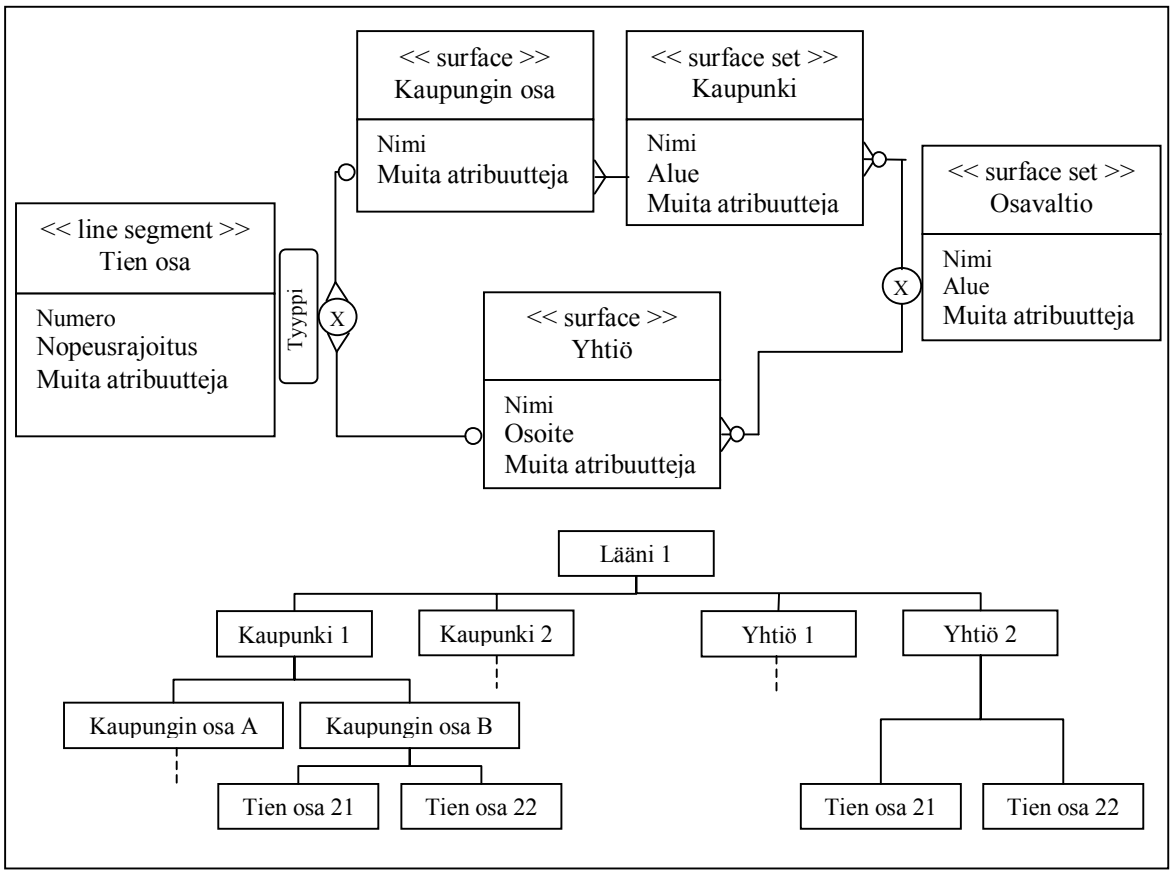
Asymmetrisissä spatiaalisissa hierarkioissa on myös ainoastaan yksi polku skeeman tasolla, mutta kardinaliteettien mukaan kaikki hierarkian alemmat tasot eivät ole pakollisia. Näin toteutustasolla puun jäsenet voivat muodostaa epätasapainoisen puun, jonka oksat eivät ole saman pituisia. Kuvassa 6 esitetään asymmetrisen hierarkian skeema, sekä esimerkki skeeman mukaisesta puurakenteesta.

[1] Tässä tutkielmassa osa Malinowskin ja Zimanyin [6] käyttämistä piktogrammeista on korvattu UML-notaation stereotyypeillä.



Kuva 6: Asymmetrisen hierarkian skeema ja esimerkki puurakenteesta (Malinowski & Zimányi, 2005).

Yleistetyt hierarkiat sisältävät skeematasolla monia toisensa poissulkevia polkuja, jotka jakavat joidenkin tasojen tietoja. Jokainen näistä poluista kuvaa yhtä hierarkiaa. Esiintymätasolla jokainen hierarkian jäsen kuuluu ainoastaan yhteen polkuun. Symboli Ⓧ ilmaisee sitä että polut ovat eksklusiivisia. Oheisessa kuvan 7 esimerkissä tämä tarkoittaa sitä, että teiden osat voivat kuulua sekä kaupungin omiin teihin että valtaväyliin. Kaupungin teiden osien huolto on paikallisen huollon vastuulla, kun taas valtavylien osien huoltotoimet on ulkoistettu tietyille yhtiöille.



Kuva 7: Yleistetyn hierarkian skeema ja esimerkki puurakenteesta (Malinowski & Zimányi, 2005).

2.2.6 Käsitteellisen skeeman muuntaminen loogiseksi skeemaksi

Malinowski ja Zimányi (2007) esittävät lisäksi artikkelissaan, kuinka suunniteltu käsitteellinen tietomalli voidaan muuntaa varsinaiseksi loogiseksi tietokantamalliksi. He viittaavat myös niihin etuihin, joita tämän kaltaisten muunnosten suorittamisessa saavutetaan käyttämällä spatiaaliset ominaisuudet jo sisältäviä tietokannanhallinta-järjestelmiä, kuten Oracle Spatialia, spagettitietomallin etuihin topologiseen malliin verrattuna, sekä semantiikan säilyttämisen tärkeyteen muunnettaessa käsitteellistä mallia loogiseksi malliksi.

Yleinen tapa toteuttaa spatiaalinen tietokanta loogisella tasolla perustuu joko relationaaliseen, olio-pohjaiseen tai oliorelaatio-pohjaiseen lähestymistapaan. Monimutkai-

sen ei-atomisen tiedon esittäminen on tunnetusti eräs relaatiomallin heikkouksista. Spatiaalisen tiedon tallentamisen yhteydessä tämä tarkoittaa esimerkiksi sitä, että polygonin tallentaminen relaatiomallin mukaisesti vaatii useita rivejä, jotka kuvaavat yksittäisen polygonin murtoviivoja näiden alku- ja loppupisteiden avulla, jättäen täten käyttäjän harteille kullekkin objektille kuuluvien tietojen ryhmittelyn. Olio-pohjainen lähestymistapa tiedon tallentamiseen mahdollistaa näiden ongelmien ratkaisemisen geometrioita kuvaavien olioluokkien avulla, mutta olioihin perustuvat tallennusjärjestelmät eivät ole vielä yleistyneet kovinkaan laajalti tietokantajärjestelmissä. Toisaalta olio-relaatiomalli sisältää relaatiomallin hyvät ominaisuudet, kuten nopean tiedon haun, sekä laajentaa sen ilmaisuvoimaa oliomallin avulla. Olio-relaatiomallin ominaisuudet ovat lisäksi mukana SQL:2003-standardissa sekä johtavissa tietokantaratkaisuissa, kuten Oracle tai Informix.

Malinowski ja Zimányi (2007) käyttävät omassa skeeman muunnoksessaan surrogaatteja käyttäjän määrittelemien avaimien sijaan. Tämä johtuu siitä, että esimerkiksi tietovarastoissa käyteään usein järjestelmän määrittelemiä avaimia, joka nopeuttaa liitosten suorittamista. Lisäksi surrogaatit eivät muutu ajan kuluessa, eli kaksi objektia, joiden surrogaatit ovat identtisiä, kuvaavat samaa entiteettiä mahdollistaen historiatietojen tallentamisen. SQL:2003-standardi sekä useat kaupalliset tietokantajärjestelmät, kuten Oracle 10g, tukevat surrogaatteja.

Spatiaalisen tiedon tallentamiseen voidaan käyttää joko kaksiosaista tai integroitua arkkitehtuuria. Ensimmäinen perustuu tekniikkaan, jossa spatiaaliselle ja ei-spatiaaliselle tiedolle käytetään erillisiä hallintajärjestelmiä. Integroiduissa järjestelmissä tietokantajärjestelmää on laajennettu spatiaalisilla tietotyypeillä ja funktioilla. Kaksiosaista arkkitehtuuria käytetään useimmiten traditionaalisissa GIS-järjestelmissä. Nämä järjestelmät vaativat erillisen tietomallin kuvaamaan sekä spatiaalista että ei-spatiaalista informaatiota, joka hankaloittaa mallintamista sekä lisää järjestelmän monimutkaisuutta. Spatiaalisilla ominaisuuksilla laajennetut tietokantajärjestelmät sisältävät spatiaalisten objektien käsittelyyn vaadittavat työkalut säilyttäen silti muut ominaisuudet kuten optimointityökalut.

Malinowski ja Zimanyi (2007) valitsivat omassa tutkimuksessaan spagettitietomallin mukaisen tallennusrakenteen skeemamuunnoksen suorittamisessa. Perustelut spagettimallin valinnalle olivat mallin yksinkertaisuus verrattuna topologiseen malliin, jonka tarjoama tallennusmalli on heidän mukaansa liian rajoittava ja joustamaton.

Eräs tärkeä osatekijä käsitteellisten mallien muuntamisessa loogisiksi malleiksi on semantiikan säilyttäminen. Osa käsitteellisessä mallissa määritellystä semantiikasta voidaan menettää, jos käyttöalustaksi valittu tietokannan hallintajärjestelmä ei tue konseptimallin sisältämiä rakenteita. Tässä tapauksessa voidaan käyttää eheysrajoitteita, kuten *laukaisimia* (triggers), joiden avulla voidaan varmistaa käsitteellisen ja loogisen skeeman ekvivalenssi. Laukaisin suorittaa tarvittavat eheyden tarkistukset aina tietokannan tietojen päivittyessä.

2.2.7 Oracle Spatialin hierarkkinen tietorakenne

Oracle Spatialin tietomalli on hierarkkinen rakenne, joka koostuu *elementeistä*, *geometrioista* sekä *kerroksista* (Oracle, 2003). Kerrokset muodostuvat geometrioista, jotka vastaavasti muodostuvat elementeistä. Oracle Spatialin hierarkkinen tietorakenne sisältää myös *koordinaattijärjestelmän* ja *toleranssin*.

Elementti (element) on geometrian perusosa. Oracle Spatialin tukemia elementin tyyppejä ovat pisteet, murtoviivat ja polygonit. Elementit voivat kuvastaa esimerkiksi tähtikuvioita (pisteryhmä) tai teitä (joukko murtoviivoja). Jokainen elementin koordinaatti tallennetaan X,Y koordinaattiparina. Yksittäisen pisteen tieto koostuu yhdestä koordinaatista. Viivan sisältämä tieto koostuu kahdesta koordinaatista, jotka kuvaavat elementin osaa. Polygonin tiedot koostuvat koordinaattipareista, yksi koordinaattipari polygonin kutakin sivua kohti.

Geometria (geometry) on jonkin spatiaalisen ominaisuuden esitysmuoto, joka on mallinnettu tiettyjen primitiivisten spatiaalisten elementtien järjestettynä joukkona.

Geometria voi koostua yksittäisestä elementistä, joka on jonkin tuetun primitiivisen tyypin ilmentymä, tai vaihtoehtoisesti se voi muodostua homogeenisestä tai heterogeenisestä elementtikokoelmasta. Homogeeninen kokoelma voi sisältää esimerkiksi tietyn maa-alueen geometriaa kuvaavat polygonit. Heterogeeninen kokoelma taas sisältää eri tyyppisiä elementtejä, kuten pisteitä ja polygoneja.

Kerros (layer) on kokoelma geometrioita, jotka jakavat samanlaisen attribuuttijoukon. GIS-järjestelmän yksi kerros voi esimerkiksi kuvata objektien topografisia ominaisuuksia, toisen tason kuvatessa asukastiheyttä. Kuhunkin kerrokseen kuuluvat geometriat ja näihin liittyvät spatiaaliset indeksit tallennetaan tavalliseen tietokantatauluun.

Koordinaattijärjestelmän (coordinate system) tai spatiaalisen viitejärjestelmän avulla määritellään koordinaatit tietylle sijainnille sekä muodostetaan suhteita näin määriteltyjen sijaintien välille. Se mahdollistaa todellisessa maailmassa sijaitsevan sijainnin esittämisen koordinaattijoukkona paikannusjärjestelmässä.

Toleranssia (tolerance) käytetään, kun spatiaaliselle tiedolle halutaan asettaa jokin tietty tarkkuusaste. Toleranssi kuvaa etäisyyttä, jonka verran objektit voivat poiketa toisistaan, mutta tulos tulkitaan edelleen sellaiseksi, että objektit ovat samassa pisteessä. Tämä on käytännöllistä esimerkiksi pyöristyksestä aiheutuvien tarkkuusvirheiden poistamisessa.

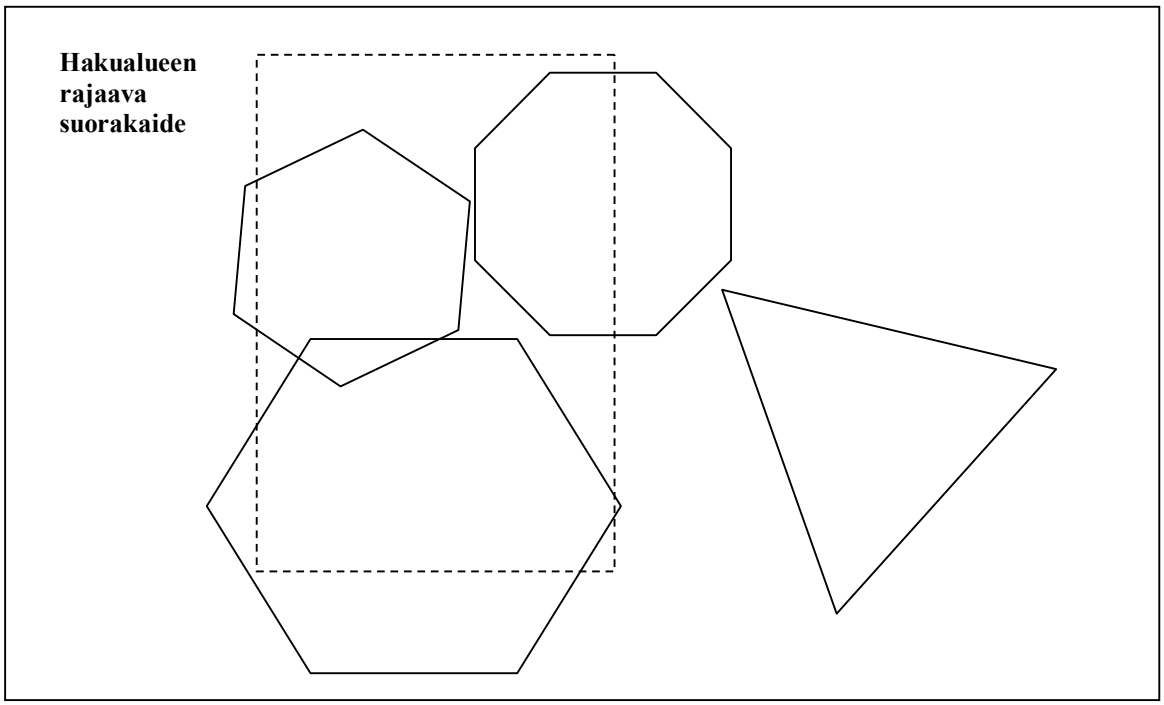
Oracle Spatialin hierarkkinen rakenne mahdollistaa spagettimallin käytön SDO_GEOMETRY-tietotyypin avulla (Oracle, 2003), sekä verkko- ja topologiamallin mukaisen tietomallin generoinnin (Oracle, 2005). Tässä tutkielmassa esitetyt Oracle Spatial –järjestelmän esimerkit on toteutettu spagettimallin mukaisella tallennusrakenteella.

2.3. Indeksointi ja spatiaaliset tiedonsaantimenetelmät

Moniulotteisten geograafisten objektien käsittelyä tehostavat tiedonsaantimenetelmät ovat ensisijaisen tärkeitä esim. CAD-suunnittelun sekä spatiaalista tietoa käsittelevien tietokantojen yhteydessä (Seeger & Kriegel, 1990). Tiedonsaantimenetelmät voidaan jakaa karkeasti pistetiedon saantimenetelmiin (*PAM, Point Access Methods*) sekä spatiaalisen tiedon saantimenetelmiin (*SAM, Spatial Access Methods*), joita käytetään haettaessa tietoa pisteistä tai spatiaalisista objekteista kuten polygoneista moniulotteisessa spatiaalisessa avaruudessa. Käytettävien tiedonsaantimenetelmien tulee olla dynaamisia, eli toisin sanoen niiden täytyy tukea toistuvaa tiedon lisäämistä, poistamista ja päivittämistä ilman tiedon globaalia uudelleenjärjestämistä tai suorituskyvyn heikkenemistä. Lisäksi niiden tulisi tukea tehokkaasti monia spatiaaliseen tietoon liittyviä hakutyyppejä, kuten lähimmän naapurin tai objektien välisen etäisyyden selvittämistä.

Spatiaalisten kyselyiden tehokas suorittaminen vaatii tietorakenteeseen soveltuvaa saantimenetelmää, jota kutsutaan indeksiksi (Rigaux & al., 2002). Saantimenetelmän tarkoituksena on nopeuttaa tiedonsaantia vähentämällä niiden objektien määrää, joiden tietoja haetaan tietokannasta hakua suoritettaessa. Perinteisissä tietokannoissa käytettävät yksiulotteiset indeksirakenteet eivät ole tehokkaita moniulotteista spatiaalista tietoa käsiteltäessä (Guttman, 1984). Arvojen absoluuttiseen vertailuun perustuvat rakenteet, kuten hajautustaulut, eivät ole käytännöllisiä, ja yksiulotteiseen avainarvojen järjestämiseen perustuvat rakenteet, kuten B-puut tai ISAM-indeksit, eivät toimi moniulotteisessa hakuavaruudessa.

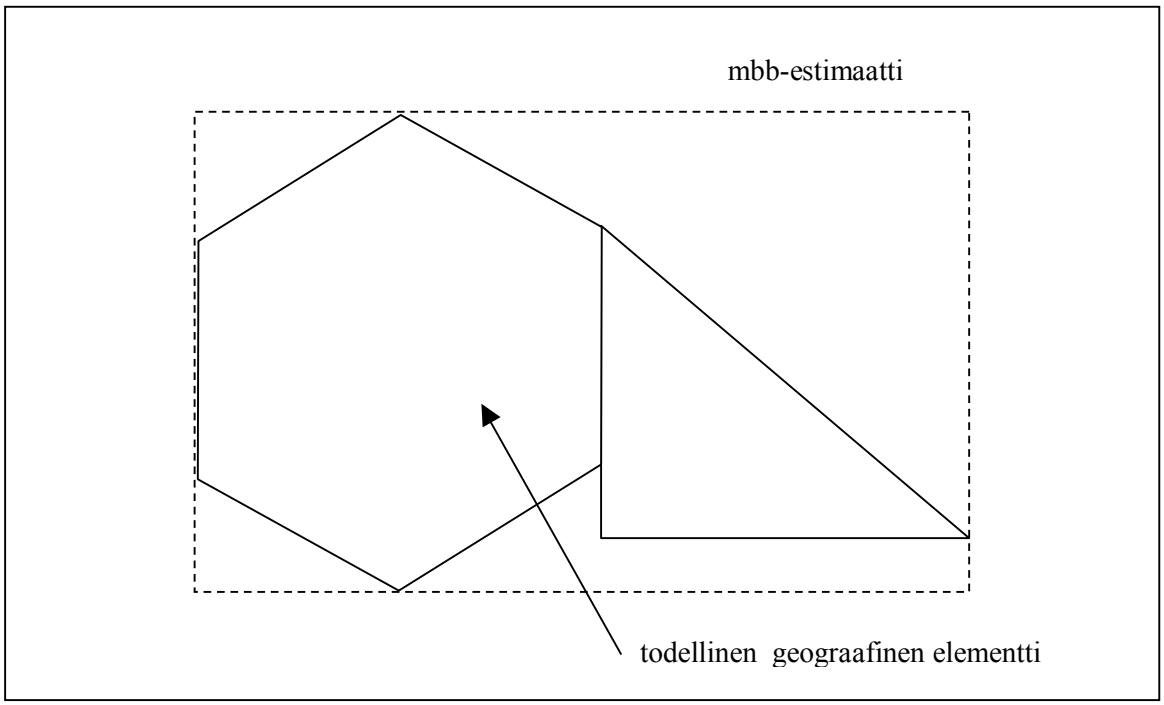
Esimerkkeinä spatiaalisista kyselyistä voidaan mainita piste-ja ikkuna-haut (Rigaux & al., 2002). Näissä kyselyissä etsitään objekteja, joiden geometria sisältää haetun pisteen, tai sijaitsee suorakaiteella rajatun alueen sisällä. Kuvassa 8 esitetään esimerkki ikkuna-haun toiminnasta käytännössä. *Spatiaalinen liitos* (spatial join) on myös eräs tärkeistä hakuesimerkeistä, jolla haetaan objekteja jotka kuuluvat haluttuun spatiaaliseen relaatioon kuten vierekkäisyys, päällekkäisyys tai sisältyvyys.



Kuva 8: Esimerkki ikkuna-hausta.

Spatiaalisen hakulauseen suoritus vaatii monimutkaisten ja suoritusaikaa vaativien geometristen operaatioiden suorittamista. Esimerkiksi hyvin tavallinen pisteen hakeminen suuresta joukosta objekteja vaatii monia hakuja fyysisestä levyjärjestelmästä, sekä useita geometristen predikaattien vertailuja. Näin ollen sekä aikaavievät geometriset algoritmit että suuret spatiaaliset datamäärät tallennusmedialla kannustavat yhä tehokkaampien spatiaalisten tiedonsaantimenetelmien kehittämiseen.

Useimmissa tapauksissa spatiaalinen tiedonsaantijärjestelmä käyttää eksplisiittistä rakennetta, jota kutsutaan spatiaaliseksi indeksiksi (Rigaux & al., 2002). Objektien varsinaisten geometrioiden sijaan indeksoidaan useimmiten yksinkertaistettu kuvaus objektista. Yleisimmin käytetty tapa luoda tällainen objektin arvioituun geometriaan perustuva indeksi on *mbb-suorakaiteen* (minimal bounding box) käyttö. Hauissa tarvittavien monimutkaisten geometristen predikaattien aiheuttamaa työkuormaa voidaan siis vähentää käyttämällä mbb:tä geometrisena avaimena spatiaalisen indeksin luomiseen, sillä mbb:hen kohdistettu geometrinen kysely on aikavaativuudeltaan vakioaikainen. Kuvassa 9 esitetään esimerkki geometriaan liitetystä mbb-estimaatista.



Kuva 9: Esimerkki mbb-estimaatista.

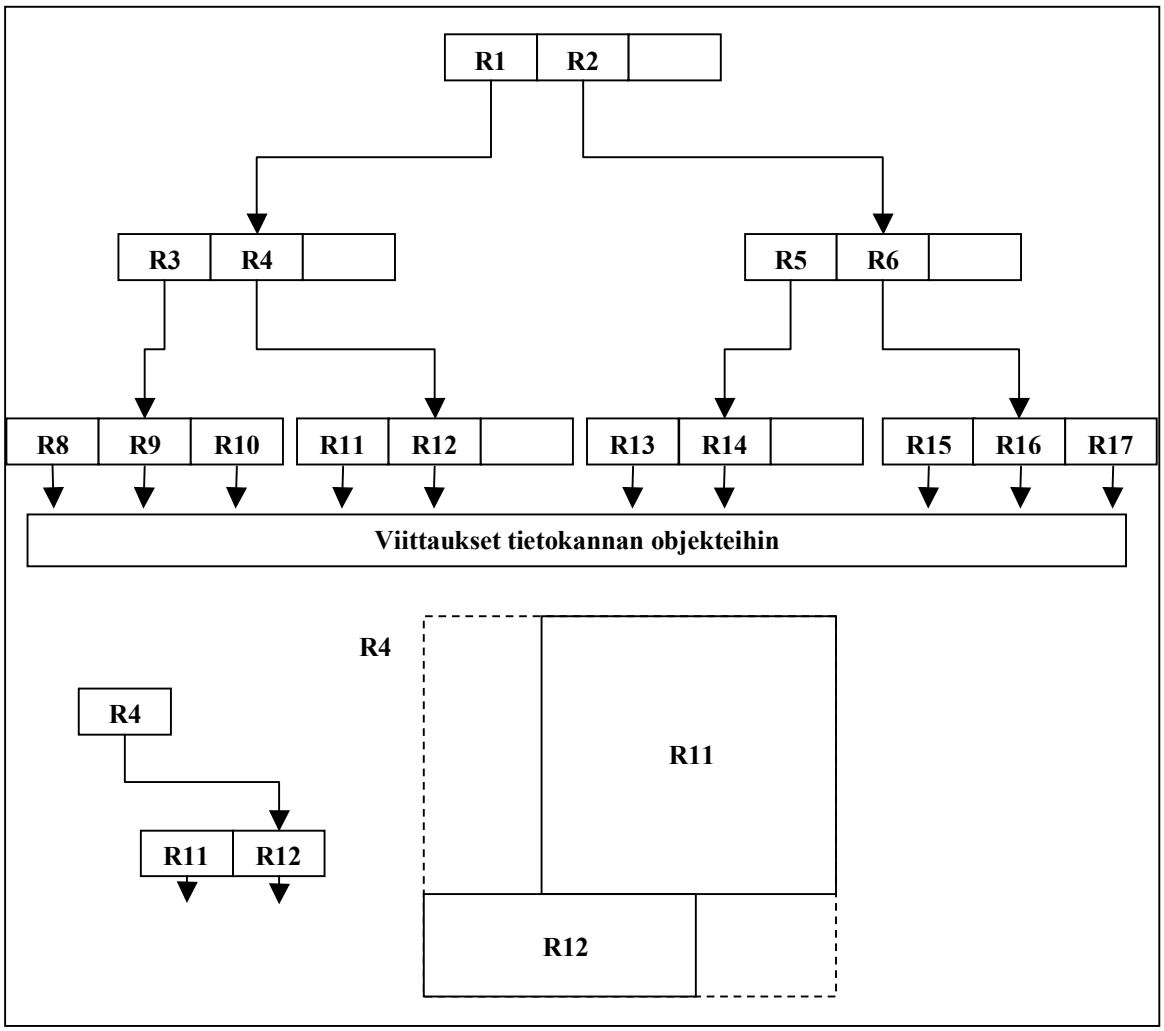
Spatiaalinen indeksi rakentuu kokoelmasta, joka sisältää geometristen objektien mbb-tiedot sekä objektin tunnuksen, joka kertoo mille objektille kyseinen mbb kuuluu (Rigaux & al., 2002). Spatiaalinen operaatio, jolla haetaan tietoa tällä periaatteella indeksoiduista objekteista, on kaksivaiheinen. Ensimmäisessä vaiheessa valitaan ne objektit, joiden mbb:hen suoritettu haku tyydyttää spatiaalisen hakulauseen ehdot, jonka tuloksena saadaan joukko objektien tunnuksia. Tätä vaihetta kutsutaan *suodatusvaiheeksi* (filter step).

Objektin mbb saattaa tyydyttää hakulauseen edellyttämät ehdot riippumatta siitä täyttyvätkö ne varsinaisen geometrisen objektin kohdalla. Toisessa *jalostusvaiheeksi* (refinement step) kutsutussa vaiheessa suodatuksella tulokseksi saadulle geometristen objektien joukolle suoritetaan alkuperäinen hakulause, kullekin spatiaaliselle objektille erikseen. Tämä vaihe on luonnollisesti edelleen aikaa ja raskasta geometrista laskentaa vaativa, mutta suodatuksen avulla käsiteltävää joukkoa saadaan rajattua.

2.4. R-puu-indeksointi

R-puu on yksi suosituimmista spatiaalisten tietokantojen kanssa käytettävistä indeksointirakenteista. Guttmanin (1984) alkuperäinen R-puu perustuu haun heuristiseen optimointiin haun edetessä puun alemmille oksille. Uudemmat R-puiden variantit kuten Beckmanin R*-puu (Beckmann & al., 1990) tai Greenen (1989) R-puu ovat tehokkuudeltaan optimoituja versioita Guttmanin alkuperäisestä R-puun rakenteesta.

R-puu on binääripuun kaltainen korkeudeltaan tasapainotettu puurakenne, jonka solmut sisältävät osoittimen indeksoitaviin geometrisiin objekteihin. R-puun rakenteen tarkoituksena on pienentää läpikäytävää tietojoukkoa spatiaalista tietokantakyselyä suoritettaessa. R-puu on dynaaminen, joten tiedon lisääminen, poistaminen ja hakeminen voidaan suorittaa ilman indeksin uudelleen järjestelyä (Guttman, 1984). Ne puun solmut jotka eivät ole lehtisolmuja sisältävät lapsisolmun osoitteen R-puussa, sekä suorakaiteen (mbb-estimaatin), joka rajaa kaikki solmun lapsisolmujen sisältämät alueet. Lehtisolmut sisältävät objektin ID-tunnisteen, jolla viitataan varsinaiseen tietokannassa olevaan geometriseen objektiin, sekä mbb-estimaatin kyseiselle objektille (Beckmann & al., 1990). Kuvassa 10 esitetään R-puun rakenne sekä geometriat, joihin indeksit viittaavat.



Kuva 10: R-puun rakenne (Guttman, 1984).

Oletetaan, että M on yhteen solmuun tallennettavien tietoparien maksimimäärä, sekä määritellään, että pienin solmuun tallennettavien tietojen määrä on m ($2 \leq m \leq M/2$). Tällöin R-puu toteuttaa seuraavat ehdot (Guttman, 1984) :

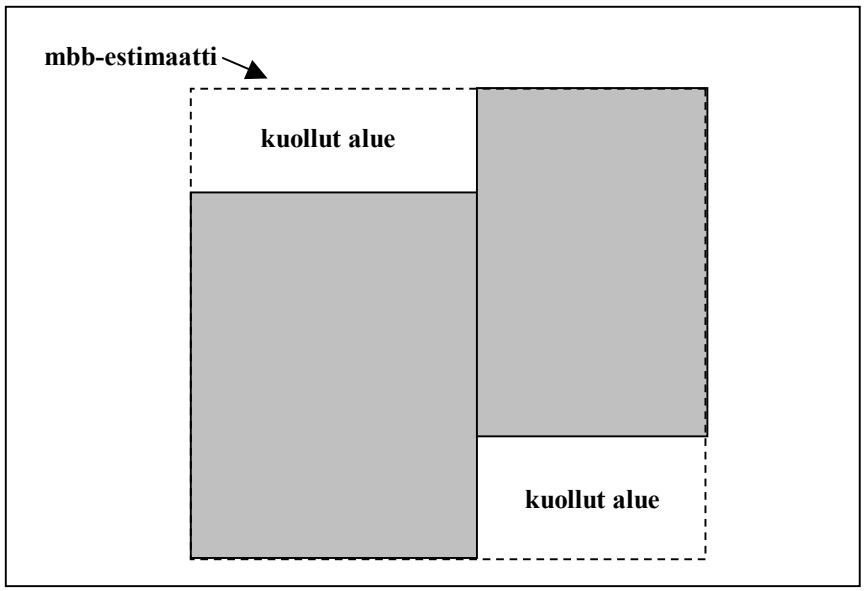
- Juurisolmulla on vähintään kaksi lapsisolmua, ellei se ole lehtisolmu.
- Jokaisella solmulla, joka ei ole lehtisolmu on vähintään m ja korkeintaan M lapsisolmua, ellei solmu ole juurisolmu.
- Kukin lehtisolmu sisältää vähintään m ja korkeintaan M tietoparia, ellei solmu ole juurisolmu.
- Kaikki lehtisolmut ovat samalla puun tasolla.

N indeksitietuetta sisältävän R-puun korkeus on korkeintaan $\lfloor \log_m N \rfloor - 1$, koska kukin solmu haarautuu vähintään m kertaa (Guttman, 1984). Puun solmujen maksimimäärä saadaan laskemalla $\lfloor N/m \rfloor + \lfloor N/m^2 \rfloor + 1$.

Beckmann & al. (1990) mukaan R-puu -rakenteen suurin ongelma on sen hakutoimintojen optimoinnin monimutkaisuus. Tietojoukosta tunnetut ominaisuudet, jotka vaikuttavat haun tehokkuuteen vaikuttavat toisiinsa erittäin monimutkaisella tavalla, joten yhden ominaisuuden optimointi saattaa loppujen lopuksi aiheuttaa suorituskyvyn heikkenemisen varsinaisessa hakutoiminnossa. Lisäksi yhden parametrin optimointiin keskittyvät menetelmät ovat erittäin epävarmoja johtuen indeksirakenteen dynaamisuudesta. Tästä johtuen optimoinnissa käytetään erilaisia mittaustuloksiin perustuvia heuristisia optimointimenetelmiä.

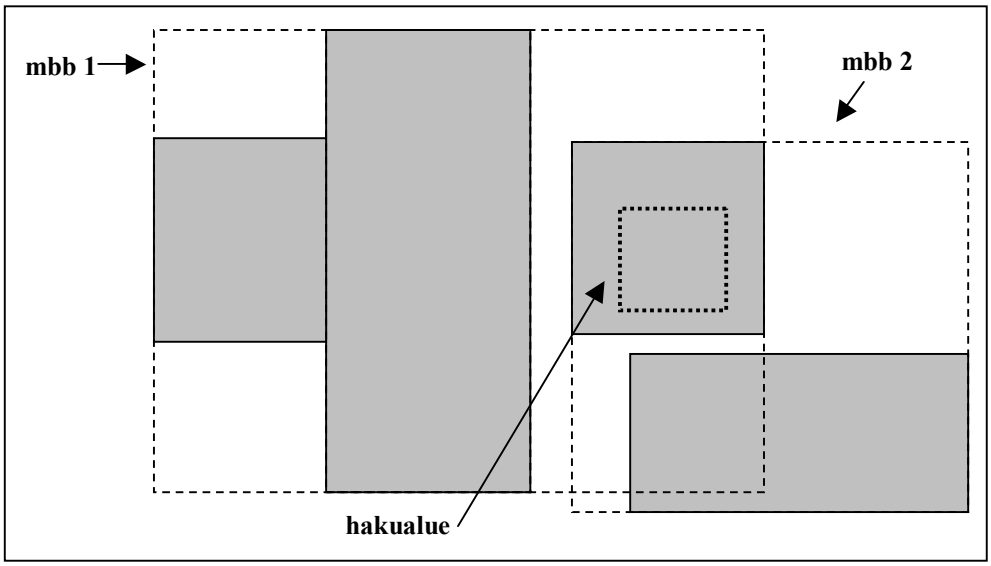
Hakutoimintojen tehokkuuden optimoinnin kannalta oleellisia parametreja ovat Beckmann & al. (1990) mukaan:

- *Rajaavien suorakaiteiden alueen tulee olla minimaalinen*. Toisin sanoen ylemmällä tasolla mbb-suorakaiteen rajaama alue, joka ei silti kuulu varsinaiseen alla olevaan geometriaan, on kuollut alue, jolla ei ole merkitystä hakutulosten kannalta. Hukkatilan vähentäminen tehostaa hakutoimintoja, koska tuloksen tuottava päätöspolku voidaan mahdollisesti saavuttaa ylemmillä tasoilla. Kuvassa 11 on esitetty esimerkki mbb-estimaatin sisältämästä kuolleesta alueesta.



Kuva 11: Esimerkki mbb-estimaatin sisältämästä kuolleesta alueesta.

- *Kahden tai useamman mbb-suorakaiteen päällekäisyydet tulisi minimoida*. Tämä vähentää hakutoiminnon aikana tutkittavien polkujen määrää. Esimerkiksi kuvan 12 mukaisessa tilanteessa hakualue sisältyy molempiin puun haaroihin, joten puun molemmat mahdollisen tuloksen sisältävät oksat on tutkittava erikseen. Kuvassa 12 on esitetty esimerkki hausta, jonka suoritus jakaantuu mbb-estimaattien päällekäisyyden vuoksi.



Kuva 12: Esimerkki päällekkäisyyden aiheuttamasta haun jakautumisesta.

- *Rajaavien suorakaiteiden marginaalin tulee olla minimaalinen*. Tässä yhteydessä marginaali tarkoittaa suorakaiteen sivujen pituuksien summaa. Kiinteän kokoista aluetta käsiteltäessä pienin marginaali saavutetaan neliön muotoisella alueella. Täten marginaalia optimoimalla rajaavat suorakaiteet muodostetaan ennemmin neliön muotoisiksi. Tästä optimoinnista hyötyvät eniten suuret, neliömäisen hakusuorakaiteen sisältävät hakutoiminnot. Marginaalin minimointi parantaa myös periaatteessa indeksin rakennetta, koska neliömäiset alueet ovat helpommin pakattavissa pieneen tilaan. Näin niiden tarvitsema ylemmän tason mbb-suorakaiteen vaatima alue on myös pienempi.

- *Indeksin tilankäyttö tulee olla optimoitua*. Olemassa olevan indeksiavaruuden käyttö tehokkaasti tehostaa hakutoimintoja, koska puun rakenne pysyy matalampana.

2.4.1 R-puun laatu

Huomattava määrä lisäys- tai poisto-operaatioita R-puu -indeksiin saattaa heikentää indeksipuun rakennetta (Oracle, 1999). Tämä voi vastaavasti vaikuttaa negatiivisesti hakutoimintojen suorituskykyyn. R-puun indeksirakenteen suorituskyky kyselylauseita suoritettaessa on riippuvainen puun indeksisolmujen määrästä sekä näiden etäisyydestä toisiinsa. Alimmalla tasolla olevat solmut sisältävät varsinaisten geometrioiden mbb-estimaatit, joten varsinaisen geometrian tietojen hakeminen alimmalta tasolta on riippuvainen alimman tason sekä ylimmän juurisolmun välisestä tietomäärästä.

2.5. Indeksointi Oracle Spatial -järjestelmässä

Spatiaalisten tiedon indeksointi on yksi Oracle Spatialin pääominaisuuksista (Oracle, 2003). Spatiaalinen indeksi mahdollistaa hakujen rajoittamisen, kuten muutkin vastaavat tietokantajärjestelmien indeksointimenetelmät. Spatiaalista tietoa käsiteltäessä indeksointi pohjautuu spatiaalisten kriteereiden arviointiin, kuten limittäisyys tai vierekkäisyys. Spatiaalista indeksiä tarvitaan esimerkiksi etsittäessä indeksoidusta tietojoukosta objekteja, jotka liittyvät haettavaan pisteeseen tai alueeseen, tai haettaessa spatiaalisesti toisiinsa liitoksissa olevia objekteja kahdesta indeksoidusta tietojoukosta. Spatiaalinen indeksi on käsitteelisellä tasolla looginen indeksi. Spatiaaliseen indeksiin tallennetut tiedot ovat riippuvaisia geometrioiden sijainnista koordinaattiavaruudessa, mutta varsinaiset indeksiarvot sijaitsevat erillisessä tietoavaruudessa.

Oracle Spatial käyttää oletusarvoisesti R-puu -indeksointia. Toisena vaihtoehtona on ”Quadtree”-indeksointi. On myös mahdollista käyttää molempia indeksointimenetelmiä yhtäaikaisesti. On kuitenkin suositeltavaa käyttää R-puuhun perustuvaa indeksointia, sillä ”Quadtree”-indeksi on Spatialin vanhentunut ominaisuus.

2.6. Spatiaalisen tiedon tallennus Oracle Spatial -järjestelmässä

Tässä kohdassa esitellään käytännön esimerkkien avulla spatiaalisen tietokannan luontiin, lisäämiseen ja ylläpitoon tarvittavat komennot Oracle Spatial -ympäristössä. Oracle Spatial -teknologia kuuluu oletusarvoisesti Oracle database server -ohjelmiston *standard* sekä *enterprise* malleihin, versioon 10.1.0.2 sekä tätä uudempiin versioihin (Kothuri & al., 2004). Database server-ohjelmiston peruslisenssi ei sisällä kaikkia luvussa esiteltyjä Spatialin funktioita. Näiden funktioiden käyttöön saamiseksi tulee hankkia erillinen Spatialin tuotelisenssi, joka sisältää tarvittavat funktiot.

2.6.1 Spatiaalisen tietomallin luonti Oracle Spatial -järjestelmässä

Käsitellään esimerkkinä yksinkertaisen spatiaalisen tietomallin luonti, joka koostuu sijaintitiedosta sekä tähän sidostusta geometrisesta objektista (Kothuri & al., 2004). Sijainti määrittelee objektin sijainnin moniulotteisessa koordinaattiavaruudessa. Geometrinen objekti pitää sisällään objektin geometrisen rakenteen, eli esimerkiksi pisteitä, murtoviivoja tai polygoneja. Esimerkissä on pyritty selkeyteen ja yksinkertaisuuteen, todellisissa järjestelmissä käytettävät tallennusmallit voivat luonnolisesti olla huomattavasti monimutkaisempia. Oracle Spatial tallentaa tietorivien sisältämät sijainti-sekä geometriatiedot *SDO_GEOMETRY*-tietotyypin mukaisena tietona, joka laajentaa *Oracle object*-tietotyyppiä. SDO_GEOMETRY-tietotyyppi on suunniteltu pitämään sisällään erilaisia geometrisia muotoja, kuten pisteitä, murtoviivoja, polygoneja tai edellämainittuista luotuja kokoelmia, eli se kykenee mallintamaan useimmissa spatiaalisissa sovelluksissa esiintyvää spatiaalista tietoa. Lisäksi tietomalli on yhteensopiva Open GIS Consortiumin (OGC, 2003) geometriamallin kanssa.

2.6.2 Tietokantataulun luonti

SDO_GEOMETRY-tietotyypin avulla voimme luoda tauluja, jotka sisältävät sijaintitietoa (Kothuri & al., 2004). Voimme esimerkiksi luoda kuvan 13 mukaisen *ravintolat*-tietokantataulun:

```
SQL>  CREATE TABLE      ravintolat
(
id          NUMBER,
poi_name    VARCHAR2(32),
location    SDO_GEOMETRY            -- Sijaintitiedon tallentava
sarake
);
```

Kuva 13: Tietokantataulun luonti (Kothuri & al., 2004).

Kuvan 13 esimerkillä olemme siis luoneet tietokantataulun, joka sisältää sijaintitiedot. Seuraavaksi esitellään kuinka luotuun tauluun voidaan lisätä tietoja. Koska SDO_GEOMETRY on objektityyppi, voidaan tätä tietotyyppiä käyttävät tietokanta-sarakkeet täyttää vastaavaan objektin *konstruktorin* avulla. Voimme esimerkiksi lisätä ravintolan sijainnin *ravintolat*-tauluun kuvan 14 osoittamalla tavalla.

SDO_GEOMETRY-objektin esiintymä luodaan siis objekti-luokan konstruktorin avulla (Kothuri & al., 2004). Luontilauseen ensimmäinen argumentti *2001* määrittelee että luotava objekti on 2-ulottainen piste. Vastaavasti murtoviivan luontiargumentti olisi *2002*, polygonin *2003*, sekä kokoelmatietotyyppiä käytettäessä *2004*. Neljäs argumentti määrittelee, että pisteen sijanti tallennetaan SDO_POINT-tyyppisenä objektina käyttäen SDO_POINT_TYPE-konstruktoria.

```
SQL>  INSERT INTO ravintolat  VALUES
(
      1,
      'RAVINTOLA 1',
      SDO_GEOMETRY
      (
            2001, -- SDO_GTYPE attribuutti: "2" attribuutissa 2001
                  määrittelee ulottuvuuksien määrän 2-ulotteiseksi
            NULL, -- muiden kenttien arvoksi asetetaan NULL.
            SDO_POINT_TYPE          --   määrittelee   koordinaatit
                  piste-objektille
            (
                  -87,  -- sijainti pituusasteikolla
                  38,   -- sijainti leveysasteikolla
                  NULL  --   mahdollinen   kolmannen   ulottuvuuden
                  sijainti
            ),
            NULL,
            NULL
      )
);
```

Kuva 14: Sijaintitiedon lisääminen *ravintolat*-tauluun (Kothuri & al., 2004).

Ohessa esitelty esimerkki käyttää tiedon syöttöön yhtä SQL:n INSERT-lauseketta. Tiedon syöttäminen voitaisiin toisaalta suorittaa myös isommissa erissä Oraclen työkalujen kuten *SQL*Loader*:in, *import*- tai *export*-toimintojen avulla tai käyttämällä ohjelmointirajapintaa, kuten *OCI*, *OCCI* tai *JDBC*.

3. Spatiaalisen tiedon hakeminen

Spatiaalisissa tietokannoissa käytettävä yleinen malli on johdettu relaatiotietokannan perusmallista, joka mahdollistaa geometristen objektien tallentamisen tietokantaan perinteisten relaatiotietokantaan tallennettavien atomisten muuttujien kuten numeromuuttujien sijaan (Paredaens & al., 1994). Vanhat perinteiset tietokannanhallintajärjestelmät eivät täytä geometristen informaatiojärjestelmien, karttajärjestelmien tai CAD-suunnitelussa käsiteltävän spatiaalisen tiedon käsittelyssä vaadittavia ominaisuuksia, koska ne perustuvat yksinkertaisempien tietojen käsittelyyn optimoituihin algoritmeihin (Orenstein, 1986). Samalla tavoin myös tiedon hakemisessa käytettävät algoritmit eroavat yksinkertaisemmilla tietotyypeillä käytettävistä algoritmeista.

Spatiaalisten tietokantojen monet erilaiset käyttökohteet edellyttävät erilaisia ominaisuuksia järjestelmän kyselykieleltä (Papadimitriou & al., 1999). Toisissa käyttökohteissa hakualueen ja tulosten tarkkuus ja sijainti on ensisijaisen tärkeää. Joissain sovelluksissa tarkastelun kohteena voivat olla ainoastaan tietokannan objektien väliset topologiset suhteet. Nämä erot käyttötarkoituksessa ja sovellusalueessa ovat oleellisia, sillä ne määrittelevät sen mikä tietojärjestelmämalli ja kyselykieli soveltuu parhaiten käyttöön, sekä vaikuttavat järjestelmän tehokkuuteen.

Tässä luvussa käsitellään spatiaalisen tiedon hakemiseen käytettäviä menetelmiä sekä spatiaalisten hakulauseiden käsittelyä tietokantajärjestelmässä. Luvussa perehdytään tarkemmin spatiaalisten tietokantojen yhteydessä käytettäviin I/O-algoritmeihin, joita kutsutaan myös ulkoisiksi algoritmeiksi, sekä spatiaalisiin liitoksiin, jotka ovat keskeinen osa spatiaalisten hakujen käsittelyä. Spatiaalisen tiedon hakemista havainnollistetaan Oracle Spatial -järjestelmän SQL-esimerkkien avulla kohdassa 3.6. Luvun lopussa tutustutaan spatiaalisen tiedon hakemiseen Java-rajapinnan kautta.

Traditionaalisten välittömästi suoritettavien spatiaalisten kyselyiden lisäksi kohdassa 3.5 käsitellään dynaamisia kyselyitä, eli niin sanottuja jatkuvia spatiaalisia tietokanta-

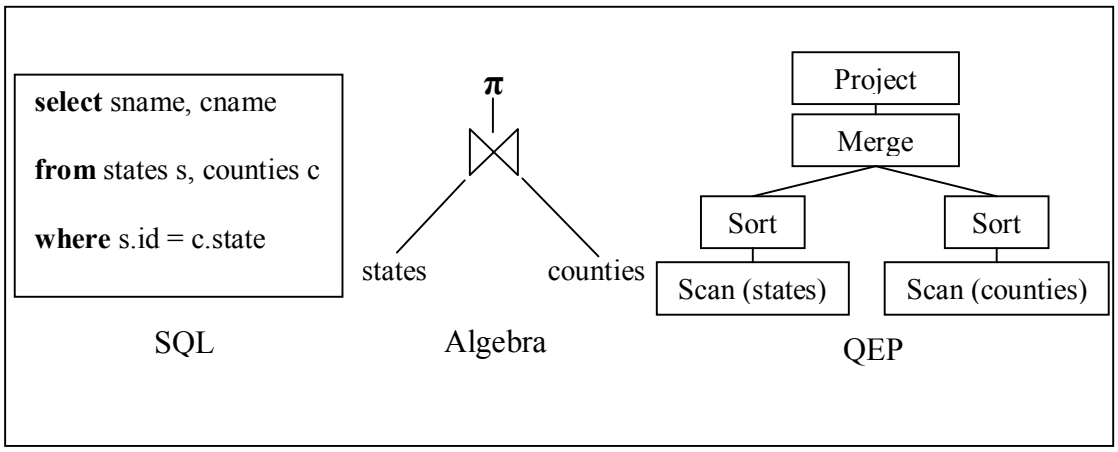
kyselyitä, jotka ovat oleellisia esimerkiksi nykyaikaisten navigaatiojärjestelmien toiminnan kannalta.

3.1. Spatiaalisen tiedon hakumenetelmät

Spatiaaliset tietokannat sisältävät useimmiten erittäin suuria tietomääriä, joka ylittää käytännössä lähes aina käytettävän järjestelmän käytettävissä olevan muistin määrän (Rigaux & al., 2002). Täten spatiaalisessa tiedonhaussa käytettävien algoritmit pyritään suunnittelemaan sellaisiksi, että fyysisen tiedon käsittelyoperaatioiden määrä voidaan pitää mahdollisimman pienenä. Algoritmien monimutkaisuutta mitataan toimintoon liittyvien kirjoitus- tai hakutoimintojen määrän perusteella.

Hakutoimintoja ohjaavan *hakumoottorin* (query execution engine) tehtävänä on ajoittaa fyysisen tiedon käsittelyä siten, että vaadittavien fyysisten tallennus- tai hakuoperaatioiden määrä jää mahdollisimman pieneksi. Tämä on erityisen tärkeää esimerkiksi *spatiaalisia liitoksia* (spatial join) sisältävissä kyselyissä, jotka sisältävät monimutkaista kahden syötteen sykronisointiin liittyvää laskentaa.

Hakulauseiden käsittelijä (query evaluator) sisältää tehokkaita I/O-algoritmeja kyselykielen lausekkeiden arviointia varten sekä tiedon hakemista ja esikäsittelyä varten. Näiden avulla pyritään päättelemään mahdollisimaan tehokas tapa yhdistellä erilaisia tarvittavia operaatioita monimutkaisten hakulauseiden suorittamiseksi. Tällaista ratkaisua kutsutaan yleensä *kyselysuunnitelmaksi* (Query Execution Plan, QEP), joka kuvailee sen kuinka tietyn hakulauseen vaatimat laskennalliset operaatiot suoritetaan. Kuvassa 15 on esimerkki kuinka SQL-kielinen hakulause (projektio ja liitos) voidaan muuntaa ja esittää QEP-muodossa.



Kuva 15: Deklaratiivisesta hakulauseesta QEP-muotoon (Rigaux & al., 2002).

Relaatiotietokannoissa käytettäville hakulauseille on tyypillistä, että kysely voidaan kuvata monilla eri lauseilla varsinaisessa relaatioalgebrassa. Lisäksi kutakin relaatioalgebran lausetta kohti voidaan luoda monia vaihtoehtosia kyselysuunnitelmia. Yksi tietokannan hallintajärjestelmän tärkeimmistä tehtävistä onkin valita paras suoritussuunnitelma monista vaihtoehdoista.

Hakutoimintojen prosessoinnin optimointi on spatiaalisten tietokantojen yhteydessä yhtä lailla tärkeää kuin perinteistä numeraalista tietoa tallentavien relaatiotietokantojen yhteydessä. *Hakuprosessorin* (QP, Query Processor) tehtävänä on muuntaa syötetyt hakulauseet yksinkertaisiksi hakutoiminnoiksi tietokannasta (Greiner, 1992). Hakutoiminnon prosessoinnin aikavaativuus on se keskimääräinen aika, joka hakuprosessorilta kuluu haun käsittelemiseen. Näiden operaatioiden vaativuus riippuu siitä, kuinka hakuprosessorin sisäinen käsittelystrategia käsittelee hakulauseita, ja kuinka se tämän johdosta muodostaa mallin tiedon hakemiseksi fyysisestä tietojärjestelmästä.

3.2. I/O-algoritmit

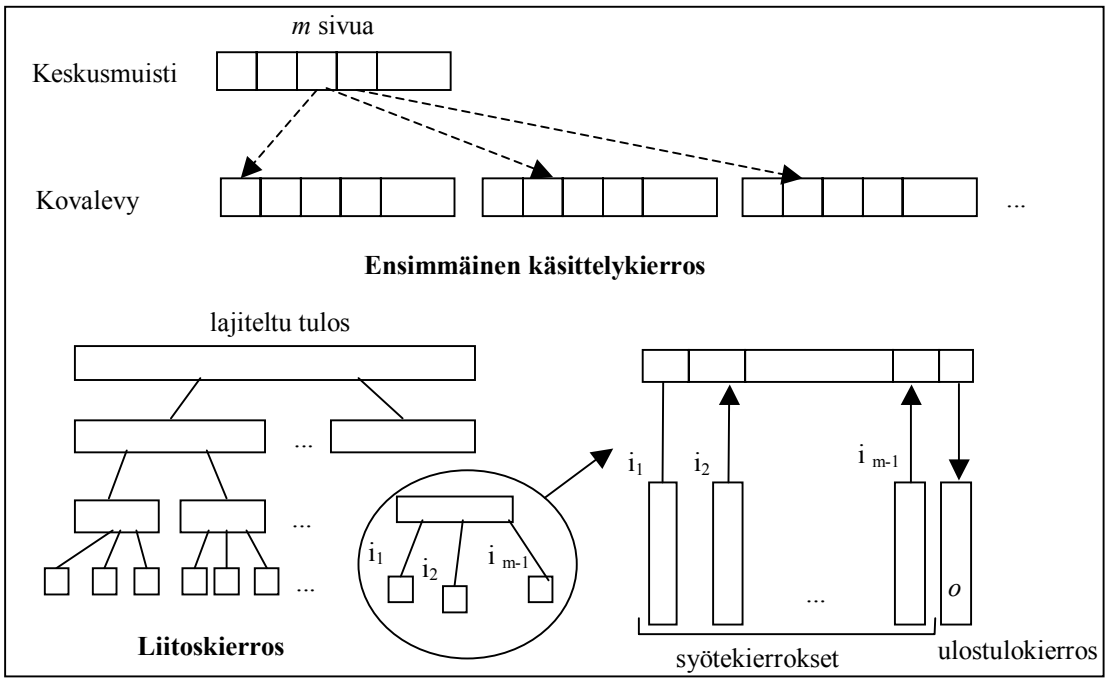
Seuraavaksi esitetään esimerkkejä spatiaalisen tiedonhaun yhteydessä käytettävistä I/O-algoritmeista (Rigaux & al., 2002). Selkeyden vuoksi käytetään yksinkertaista mallia, jossa käytössä on yhden fyysisen massamuistin sekä suorittimen sisältävä

järjestelmä. Esimerkeissä *n* kuvaa käsiteltävän tietojoukon kokoa, ja *m* kuvaa käytettävissä olevan muistin määrää.

Ensimmäisenä esimerkkinä voidaan mainita ulkoinen *lomitusjärjestäminen* (external Sort/Merge). Sitä käytetään yleensä esikäsittelyvaiheessa niiden algoritmien yhteydessä, jotka vaativat syötetietojen olevan järjestetyssä muodossa. Yksi tunnetuimmista algoritmeista tämän toteuttamiseksi on *järjestämislomitusliitos* (sort/merge-join), jota käytetään relaatiotietokannoissa kahden indeksoimattoman tietojoukon liitoksen muodostamisessa. Kahden järjestetyn relaation liitos on aikavaativuudeltaan logaritminen (O(log n)). Algoritmi alkaa partitiointivaiheella, jossa käsiteltävä tietojoukko jaetaan pienempiin osiin. Tämän jälkeen osajoukot järjestellään ja liitetään toisiinsa muodostaen yhä suurempia järjesteltyjä osajoukkoja. Kuvassa 16 esitetään algoritmin vaiheet.

Yleisin käytössä oleva ulkoisen järjestelyn toteuttava algoritmi on sort/merge-algoritmin muunnelma, joka toimii klassisella ”hajoita ja hallitse”-periaatteella. Ensimmäisessä lajitteluvaiheessa taulukkomuotoista syötetietoa jaetaan rekursiivisesti, kunnes se sisältää triviaalisti järjestettävissä olevia kahden tietoalkion pareja. Tämän partitioinnin tuloksena on binääripuu. Lomitusvaiheessa puun solmujen järjestellyt taulukot liitetään toisiinsa rekursiivisella algoritmilla, joka alkaa puun alimmista oksista.

Toinen esimerkkialgoritmi on *distribution sweeping*-algoritmi. Tätä tekniikkaa voidaan käyttää, kun halutaan saavuttaa optimaalinen *worst-case*-raja monissa geometriaa käsittelevissä laskentaoperaatioissa, jotka ovat välttämättömiä I/O-operaatioiden yhteydessä. Algoritmi on tärkeä myös siitä syystä, että sen avulla voidaan ratkaista eräs spatiaalisten liitosten aliongelmista, eli parillisten intersektioiden esiintymät kahden suuren suorakaidejoukon sisällä.



Kuva 16: Järjestämislomitusliitos algoritmin toiminta (Rigaux & al., 2002, s.273).

3.3. Spatiaaliset liitokset

Spatiaalinen liitos kahden relaation R1 ja R2 välillä muodostuu näiden relaatioiden tietoalkioiden pareista, jotka toteuttavat haussa määritellyn spatiaalisen predikaatin (Rigaux & al., 2002). Tällaisia ehtoja ovat aikaisemmissakin luvuissa mainitut spatiaaliset suhteet, kuten *päällekkäisyys*, *sisältyvyys* ja muut vastaavat topologiaa tai etäisyyttä koskevat predikaatit. Tyypillinen spatiaalinen liitos koostuu Jacoxin ja Sametin (2003) mukaan seuraavista osista: tiedon osiointi käsittelyä varten, spatiaalisten liitosten muodostaminen osajoukoille keskusmuistissa sekä tarkistusvaihe, jossa tarkastetaan kokonaisten polygonien risteävyydet keskenään. Jacoxin ja Sametin mukaan kaikki käytettävät tekniikat spatiaalisten liitosten muodostamiselle voidaan jakaa näihin edellä mainittuihin suoritusvaiheisiin.

Yleisesti hyväksytty menetelmä spatiaalisen liitoksen laskemiseksi sisältää kaksi vaihetta (Rigaux & al., 2002): *suodatusvaihe* ja *jalostusvaihe*.

Suodatusvaiheessa (filter step) valitaan ne spatiaaliset objektit, joiden mbb-suorakaiteet toteuttavat hakuehdon. Estimaattien yksinkertaisen rakenteen ansiosta tämän vaiheen avulla voidaan välttää raskaiden laskentaoperaatioiden suorittaminen varsinaisille geometrisille objekteille. Ne tietoparit, jotka valitaan suodatusvaiheessa, muodostavat *kandidaattijoukon*.

Jalostusvaiheessa kandidaattijoukon tietoparien sisältämät spatiaalisten objektien varsinaiset geometriat haetaan tietokannasta, ja haun vaatimat geometriset laskutoimitukset suoritetaan näille varsinaisille geometrisille objekteille.

On olemassa monia erilaisia strategioita ja algoritmeja spatiaalisten liitosten toteuttamiseksi. Käytettävä lähestymistapa riippuu siitä mitä spatiaalista tiedonsaantimallia järjestelmä käyttää. *Lineaarisissa rakenteissa* kukin relaatio on indeksoitu lineaarisen rakenteen avulla. Näiden rakenteiden kanssa voidaan käyttää esimerkiksi *z-järjestyspuita* (z-ordering trees). *R-puita* käytettäessä suosittu liitosalgoritmi on *synkronoitu puiden läpikäynti*. Yhden relaation osalta indeksoiduissa järjestelmissä (*single index)* yksinkertaisin lähestymistapa on *indexed nested loop* -algoritmi (INL). Tämä algoritmi tutkii indeksoimattoman relaation ja suorittaa haun toisen relaation indeksiin tietoalkioiden *mbb*-estimaattien avulla. Jos käytössä ei ole lainkaan indeksointi- järjestelmää, voidaan käyttää partitiointitekniikoita, joiden avulla jaetaan tietoparit joukkoihin. Tämän jälkeen voidaan käyttää jotain hajautusalgoritmia, kuten spatiaaliselle tiedolle räätälöityä *hash-join*-algoritmia.

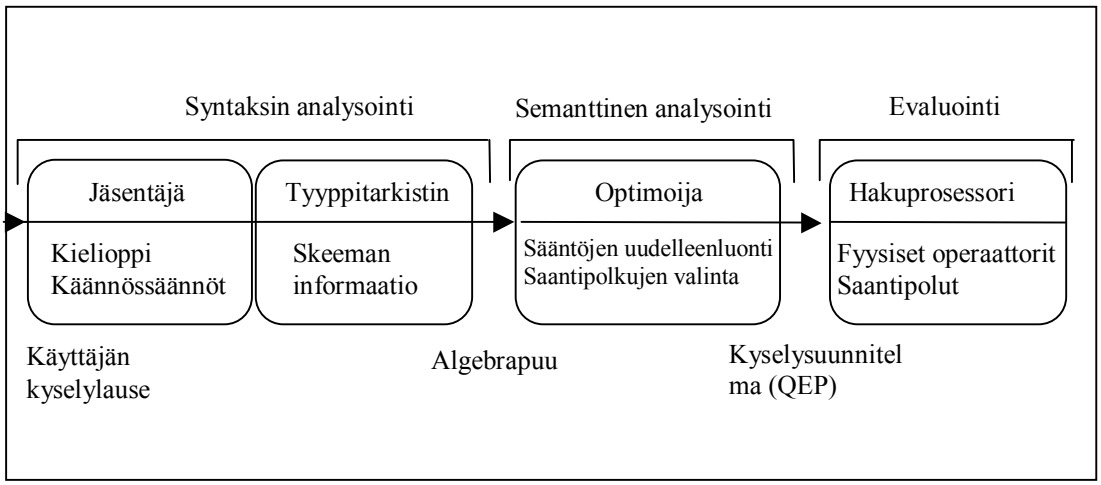
Esimerkkinä edellä mainittujen algoritmien käytännön sovelluksista voidaan mainita Oracle Spatial –järjestelmässä käytetty z-järjestelyalgoritmin rasterointiin perustuva muunnelma, joka on yksinkertainen, objektien dekompositioon perustuva versio monimutkaisemmista z-järjestelyalgoritmin versioista.

3.4. Monimutkaiset spatiaaliset kyselyt

Yhden spatiaalisen operaattorin sisältävien hakutoimintojen lisäksi voidaan suorittaa monimutkaisempia hakulausekkeita, jotka sitältävät useampia spatiaalisia operaatioita (Rigaux & al., 2002). Monimutkaisten hakuyhdistelmien suoritusta on tutkittu laajasti relaatiotietokantojen yhteydessä. Kohdassa 3.4.1 kerrotaan tarkemmin QEP-mallin mukaisesta kyselynkäsittelystä. Kohdassa 3.4.2 tutkitaan spatiaalisten hakulauseiden suorittamiseen liittyviä yleisiä ongelmia.

3.4.1 Kyselysuunnitelma

Ennen tietokantakyselyn suorittamista tietokannanhallintajärjestelmässä, tulee se ensin muuntaa ohjelmaksi, joka suorittaa tiedon hakemisen ja kyselyn prosessoinnin (Rigaux & al., 2002). Hakujen *kyselyoptimoija* (query optimizer) on yksi tärkeimmistä osista nykyisissä tietokantajärjestelmissä (Freytag, 1987). Optimoijan tehtävänä on muodostaa annetuista hakulauseista kyselysuunnitelma, jonka avulla haun tulokset saadaan laskettua mahdollisimman tehokkaasti. Kuvassa 16 esitetään tyypillinen järjestys hakulauseen käsittelyyn (Rigaux & al., 2002). Aluksi hakulause muunnetaan sitä vastaavaksi algebramuotoiseksi esitykseksi *jäsentäjän* (parser) avulla. *Tyyppi-tarkistin* (type checker) tarkastaa, että kyselyssä käytettävät relaatioiden nimet sekä attribuutit ovat valideja perustuen tietokantaskeemaan tallennettuun informaatioon. Lopputuloksena syntyvä kyselysuunnitelma on rinnastettavissa puurakenteeseen, jossa kukin oksa vastaa loogista spatiaalista operaatiota, kuten spatiaalista liitosta. Viimeinen vaihe on kyselyn optimointi, jossa mahdollisesti muokataan kyselyn sääntöjä tai suorituspolkuja. Kuvassa 17 on esitetty kyselyjen muuntaminen kyselylauseista QEP-muotoon.



Kuva 17: Kyselyjen arviointi ja muuntaminen (Rigaux & al., 2002).

QEP-suunnitelmaa kuvataan yleisesti binääripuuna, joka sisältää tiedon siitä missä järjestyksessä varsinaiset tietokantaoperaatiot suoritetaan. Lehtisolmuissa sijaitsevat operaatiot suoritetaan ensimmäisenä, eli puu käydään läpi pohjalta juurisolmuun. Koska monet loogiset operaatiot vaativat useampia fyysisiä operaatioita, kuten esimerkiksi spatiaalinen liitos ja siihen mahdollisesti vaadittavat *sort*-algoritmit, on kyselysuunnitelman muodostama puurakenne yleensä huomattavasti laajempi kuin kyselyn algebrallisen mallin sisältävä puurakenne.

3.4.2 Ongelmat spatiaalisten hakujen käsittelyssä

Rigauxin & al. (2002) mukaan spatiaalisten tietokantojen hakukäsittelyn yhteydessä ilmenee ainakin kaksi ongelmaa, jotka tulee ottaa huomioon spatiaalisten tietokanta-järjestelmien käytössä. Ensimmäinen on haun *jalostusvaihe* (refinement step). Koska spatiaaliset algoritmit ja tiedonsaantimenetelmät perustuvat mbb-estimaattien kautta tapahtuviin hakuihin, kyselyn suorituksen kannalta on välttämätöntä, että objektin todellinen geometrinen kuvaus on saatavilla.

Toinen ongelma on spatiaalisten operaatioiden raskaus. Toisin kuin perinteisten relaatiotietokantojen suhteen, jotka käsittelevät ainoastaan joukkoa yksinkertaisia predikaatteja vertailtaessa numeraalisten arvojen suhteita toisiinsa, spatiaalisissa

tietokannoissa suoritetaan aikavaativuudeltaan raskaita operaatioita spatiaaliselle tiedolle. Esimerkiksi *ikkuna-haku* R-puun osalta on huomattavasti raskaampi operaatio kuin binääripuun läpikäynti.

3.5. Spatiaaliset tiedonhakumenetelmät dynaamisissa järjestelmissä

Perinteiset spatiaaliset kyselyt ovat joissain tapauksissa käyttökelvottomia dynaamisissa järjestelmissä, joissa hakutulosten tulokset muuttuvat jatkuvasti tietokannan sisältämän informaation muuttuessa (Tao & Papadias, 2003). Näissä järjestelmissä myös tietokantakyselyiltä vaaditaan kykyä päivittää tietokantahakujen tuloksia ajan kuluessa. Tämän kaltaisia dynaamisia spatio-temporaalisia järjestelmiä ovat esimerkiksi ajoneuvojen GPS-navigaattorit, joissa hakutulosten päivittäminen on oleellista käyttäjän sijainnin muuttuessa jatkuvasti.

Mistä tahansa spatiaalisesta hakulauseesta voidaan luoda jatkuvasti päivittyvä vastine käytettäväksi dynaamisissa järjestelmissä. Jatkuvan kyselyn lopetusehto riipuu luonnollisesti sovellusalueesta ja ohjelmiston käyttötarkoituksesta. Esimerkkinä voidaan mainita ikkuna-tyyppinen tietokantahaku, joka päivittyy seuraavien viiden minuutin ajan, tai kunnes hakualueelta löydetään etsittävä objekti. Myös itse haku voidaan toteuttaa dynaamisena, esimerkiksi liikkuvana hakualueena muilta osin staattisessa ympäristössä.

Useimpien käytettyjen dynaamisten spatio-temporaalisten hakulauseiden ydin on *aikaparametroitu tietokantakysely* (time-parametrized query). Tällainen kysely palauttaa tuloksena ne spatiaaliset objektit, jotka täyttävät hakuehdot, tiedon siitä milloin kyselyn tulokset vanhenevat, sekä tiedon mahdollisista laukaisimista mitkä aiheuttavat hakulauseen tuloksen muuttuvan validista vanhentuneeksi. Aikaparametroidut kyselyt ovat oleellinen osa haettaessa tietoa dynaamisista spatiaalisista tietokannoista myös siitä syystä, että ne toimivat primitiivisinä rakennuspalikoina muille monimutkaisemmille hakulauseille.

Esimerkkinä perinteisestä staattisesta spatiaalista tietoa käsittelevästä hakulauseesta ja tästä muodostetusta dynaamisesta versiosta voidaan mainita *toistuva kNN*-algoritmi (Lee & al., 2005), joka on perinteisen staattisen kNN-algoritmin eli *nearest neighbour* -algoritmin muunnelma dynaamista hakua vaativiin sovelluskohteisiin. Erilaisia tapoja dynaamisen NN-hakutoiminnon suorittamiseksi on esitetty useita modernissa spatiaalisia algoritmeja koskevissa julkaisuissa kuten *continuous windowing kNN* (Iwerks & al., 2003), *NNS* (Raptopoulou & al., 2003), *time parametrized kNN* (Tao & Papadias, 2002) ja *plane sweeping*-tekniikka (Mokhtar & al., 2002). Ydintekniikka on joka tapauksessa samankaltainen kaikissa algoritmeissa, sillä dynaamisen kNN-haun tarkoituksena on iteratiivisesti tutkia kullakin hetkellä annetun NN-haun validiutta.

Edellä mainitun kaltaiset dynaamiset haut koostuvat kahdesta osasta, joista ensimmäisessä lasketaan hakutoiminnon alkamishetkellä ensimmäinen NN-arvo käyttäen olemassa olevia staattisia NN-algoritmeja (Lee & al., 2005). Toisessa vaiheessa tutkitaan niitä objekteja, jotka vaikuttavat haun päättymiseen sekä sitä milloin haku on ajoitettu päättymään. Hakutoiminnon aikavaativuus moninkertaistuu siinä tapauksessa että nämä objektit sijaitsevat esimerkiksi fyysisellä levyllä, jolloin levytoimintojen aiheuttamat viiveet voivat kasvaa merkittävän suuriksi. Kussakin algoritmissa käytetään erilaisia optimointitekniikoita näiden I/O- sekä laskentaviiveiden minimoimiseksi.

3.6. SQL-lausekkeet spatiaalisen tiedon hakemisessa

Tässä kohdassa käsitellään yksinkertaisten spatiaalisten hakulauseiden suorittamista SQL-kielen avulla Oracle Spatial –ympäristössä[1]. Tämän luvun esimerkit on toteutettu spagettimallin mukaisella tallennusrakenteella luotuun tietokantatauluun.

[1] Oracle Database Server -ohjelmiston versio 10.1.0.2 tai tätä uudempi versio

3.6.1 Yksinkertaiset hakulauseet

Tässä kohdassa esitellään esimerkkejä yksinkertaisista hakulauseista Oracle Spatial-ympäristössä SQL-kielen avulla (Oracle, 2003). Kuvassa 18 esitetään aluksi SQL -lauseet joiden avulla voimme luoda tietokantataulun. Tämä SQL –lause luo tietokantaan taulun nimeltä *cola_markets*, jonka pääavaimeksi määritellään numeerinen *mkt_id*. Lisäksi kantaan tallennetaan kunkin markkinointialueen nimi sekä geometrinen muoto, joka periytyy Spatialin SDO_GEOMETRY –geometrialuokasta.

```
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape SDO_GEOMETRY);
```

Kuva 18: Tietokantataulun luonti (Oracle, 2003).

Kuvassa 19 suoritetaan kaksi SQL –lausetta joiden avulla lisätään juuri luotuun *cola_markets* –tauluun kaksi uutta markkina-aluetta, *cola_a* ja *cola_b*. Molemmat luoduista markkina-alueista ovat tyypiltään 2-ulotteisia polygoneja. Ensimmäinen lisätty alue sisältää suorakaiteen geometrian, ja tarvitsee luotaessa tiedot suorakaiteen vastakkaisten kulmien koordinaateista. Jälkimmäinen markkina-alue luodaan polygoniksi joka sisältää viisi kulmapistettä.

```
INSERT INTO cola_markets VALUES(
  1,
  'cola_a',
  SDO_GEOMETRY(
    2003,  -- 2-ulotteinen polygoni
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- yksi suorakaide
    SDO_ORDINATE_ARRAY(1,1, 5,7) - Suorakaiteen määrittely vaatii
          -- ainoastaan 2 koordinaattia kahdelle vastakkaisille
          -- kulmille sekä Karteesisen koordinaatiston tiedot
  )
);



INSERT INTO cola_markets VALUES(
  2,
  'cola_b',
  SDO_GEOMETRY(
    2003,  -- 2-ulotteinen polygoni
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- yksi polygoni
    SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1)
  )
);
```

Kuva 19: Objektien lisääminen tietokantatauluun (Oracle, 2003).

Kuvassa 20 suoritetaan kaksi uutta SQL –lausetta, joiden avulla tietokantaan lisätään uudet markkina-alueet *cola_c* ja *cola_d*. Molemmat uusista geometrioista ovat jälleen tyypiltään 2-ulotteisia polygoneja. Alue *cola_c* on aiemmin luodun *cola_b*:n tavoin viidestä kulmasta koostuva polygoni. Alue *cola_d* on tyypiltään ympyrän muotoinen geometria.

```
INSERT INTO cola_markets VALUES(
  3,
  'cola_c',
  SDO_GEOMETRY(
    2003,  -- 2-ulotteinen polygoni
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- yksi polygoni
    SDO_ORDINATE_ARRAY(3,3, 6,3, 6,5, 4,5, 3,3)
  )
);




INSERT INTO cola_markets VALUES(
  4,
  'cola_d',
  SDO_GEOMETRY(
    2003,  -- 2-ulotteinen polygoni
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,4), -- yksi ympyrä
    SDO_ORDINATE_ARRAY(8,7, 10,9, 8,11)
  )
);
```

Kuva 20: Objektien lisääminen tietokantatauluun (Oracle, 2003).

Kuvassa 21 päivitetään juuri luodun tietokantataulun metatiedot, sekä luodaan spatiaalinen indeksi tietokantataululle. Metatietojen päivitys täytyy suorittaa ennen kuin tietokantataululle voidaan luoda indeksi. Metatietojen päivitys tulee suorittaa ainoastaan kerran kutakin kerrosta kohti, eli tässä tapauksessa metatietojen päivitys suoritetaan sarakeyhdistelmälle *COLA_MARKETS* ja *SHAPE*. Jälkimmäisessä SQL – lauseessa luodaan spatiaalinen indeksi *cola_spatial_idx* tietokantataulussa *cola_markets* sijaitseville geometrisille objekteille. Indeksin tyyppi määritellään R-puuksi, joka on Oracle Spatialin oletusarvoinen indeksirakenne.

```
INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'cola_markets',
  'shape',
  SDO_DIM_ARRAY(   -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
     ),
  NULL   -- SRID
);



-- Luodaan spatiaalinen indeksi
CREATE INDEX cola_spatial_idx
   ON cola_markets(shape)
   INDEXTYPE IS MDSYS.SPATIAL_INDEX;
-- Edeltävä lause määrittelee että uusi indeksi on tyypiltään R-puu
```

Kuva 21: Metatietojen päivittäminen sekä spatiaalisen indeksin luonti (Oracle, 2003).

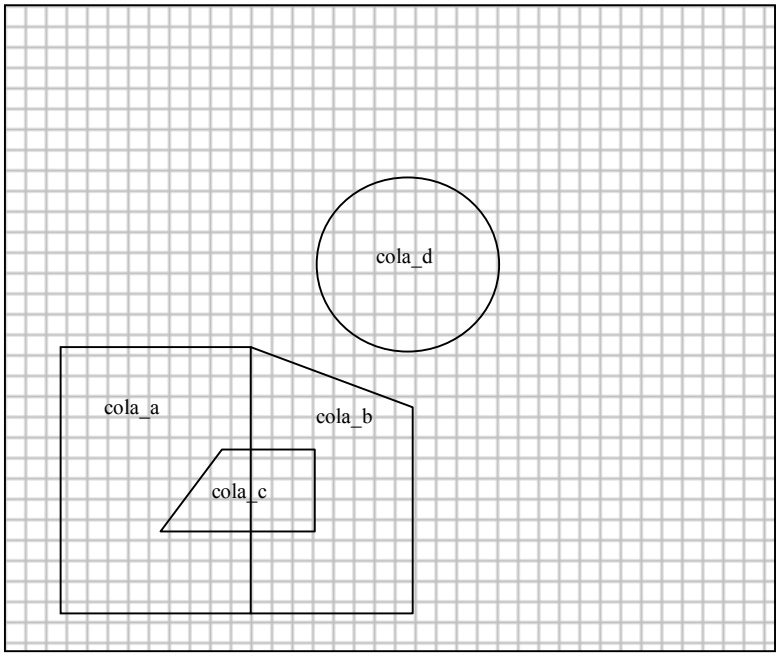
Kuvassa 22 suoritetaan kaksi yksinkertaista hakulausetta luotuun tietokantaan. Ensimmäinen hakulause palauttaa arvonaan ne alueet, jotka kuuluvat sekä *cola_a* että *cola_c* markkina-alueisiin SDO_GEOM.SDO_INTERSECTION –funktion avulla. Jälkimmäinen halulause palauttaa tiedon siitä onko alueilla *cola_b* ja *cola_d* minkäänlaisia topologisia spatiaalisia suhteita, kuten sisältyvyys tai limittyneisyys.

```
-- Palauttaa kahden geometrian topologisen intersektion
SELECT SDO_GEOM.SDO_INTERSECTION(c_a.shape, c_c.shape, 0.005)
   FROM cola_markets c_a, cola_markets c_c
   WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_c';


-- Onko valituilla geometrioilla spatiaalisia suhteita?
SELECT SDO_GEOM.RELATE(c_b.shape, 'anyinteract', c_d.shape, 0.005)
  FROM cola_markets c_b, cola_markets c_d
  WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';
```

Kuva 22: Yksinkertaisia hakulauseita Oracle Spatial-tietokannasta (Oracle, 2003).

Suoritettujen INSERT –lauseiden, metatietojen päivittämisen ja indeksin luonnin tuloksena olemme lisänneet tietokantaan neljä markkina-aluetta. Kuvassa 23 on yksinkertaistettu havainnollistava kuva siitä mitä geometrioita *cola_markets* –taulun geometria pitää sisällään.



Kuva 23: *Cola_markets* –taulun geometriat (Oracle, 2003).

Kuvassa 24 suoritetaan kolme hakulausetta. Ensimmäinen SQL –lause palauttaa arvonaan *cola_markets* –taulun kaikki geometriat toleranssiarvolla[1] 0.005. Toinen hakulause palauttaa arvonaan ainoastaan *cola_a* –alueen geometrian. Viimeinen hakulause palauttaa arvonaan markkina-alueiden *cola_b* ja *cola_d* välisen etäisyyden SDO_GEOM.SDO_DISTANCE –funktion avulla.

[1] Toleranssi kuvaa etäisyyttä, joka sallitaan kahdelle pisteelle, jotta ne tulkitaan edelleen samaksi pisteeksi.

```
-- Palauttaa kaikkien markkina-alueiden geometriat
SELECT name, SDO_GEOM.SDO_AREA(shape, 0.005) FROM cola_markets;

-- Palauttaa cola_a alueen geometrian.
SELECT c.name, SDO_GEOM.SDO_AREA(c.shape, 0.005) FROM cola_markets c
   WHERE c.name = 'cola_a';

-- Palauttaa kahden geometrian välisen etäisyyden
SELECT SDO_GEOM.SDO_DISTANCE(c_b.shape, c_d.shape, 0.005)
   FROM cola_markets c_b, cola_markets c_d
   WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';
```

Kuva 24: Yksinkertaisia hakulauseita Oracle Spatial-tietokannasta (Oracle, 2003).

Kuvassa 25 näemme kuvan 24 ensimmäisen hakulauseen tulosteen. Haku palauttaa kaikki *cola_markets* taulun sisältämät markkina-alueiden geometriat sekä näiden koot.

```
SQL> SELECT name, SDO_GEOM.SDO_AREA(shape, 0.005) FROM cola_markets;

NAME                           SDO_GEOM.SDO_AREA(SHAPE,0.005)
------------------------------ ------------------------------
cola_a                                                     24
cola_b                                                   16.5
cola_c                                                      5
cola_d                                               12.5663706

SQL>
```

Kuva 25: Ensimmäisen esimerkkihaun tuloste

Kuva 26 sisältää kuvan 24 toisen hakulauseen tulosteen. Hakulause palauttaa *cola_markets* –taulusta *cola_a* nimisten alueiden geometriat sekä alueiden koon, eli tässä tapauksessa yhden markkina-alueen.

```
SQL> SELECT c.name, SDO_GEOM.SDO_AREA(c.shape, 0.005) FROM
cola_markets c
   WHERE c.name = 'cola_a';
  2
NAME                           SDO_GEOM.SDO_AREA(C.SHAPE,0.005)
------------------------------ --------------------------------
cola_a                                                       24

SQL>
```

Kuva 26: Toisen esimerkkihaun tuloste

Kuvassa 27 nähdään kuvan 24 viimeisen esimerkkihaun tuloste, jossa haetaan markkina-alueiden *cola_b* ja *cola_d* välinen topologinen etäisyys. Haku palauttaa arvonaan näiden markkina-alueiden välisen etäisyyden.

```
SQL> SELECT SDO_GEOM.SDO_DISTANCE(c_b.shape, c_d.shape, 0.005)
   FROM cola_markets c_b, cola_markets c_d
   WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';
  2    3
SDO_GEOM.SDO_DISTANCE(C_B.SHAPE,C_D.SHAPE,0.005)
------------------------------------------------
                                      .846049894

SQL>
```

Kuva 27: Kolmannen esimerkkihaun tuloste

3.6.2 Spatiaaliset liitokset

Tässä kohdassa esitellään ilman taulumäärittelyjä Kothurin & al. (2004) esittämiä spatiaalisia liitoksia toteuttavia hakulauseita SQL-kielellä Oracle Spatial –järjestelmässä.

Kuvassa 28 suoritetaan tietokantahaku, jolla halutaan etsiä kaikki asiakkaat, jotka sijaitsevat 200 metrin sisällä kilpailevien yritysten sijainneista. Esimerkin liitos ei ole

varsinainen indeksoitu spatiaalinen liitos, vaan hakulause suoritetaan *nested loop* -toistorakenteessa, jolloin SDO_WITHIN_DISTANCE operaatio suoritetaan kullekkin riville *competitors*-taulussa.

```
SQL> SELECT COUNT (DISTINCT   ct.id)

FROM competitors comp, customers ct

WHERE SDO_WITHIN_DISTANCE

      (ct.location, comp.location, 'DISTANCE=200 UNIT=METER
')='TRUE';
```

Kuva 28: Spatiaalinen liitos SDO_WITHIN_DISTANCE-operaattorin avulla (Kothuri & al., 2004).

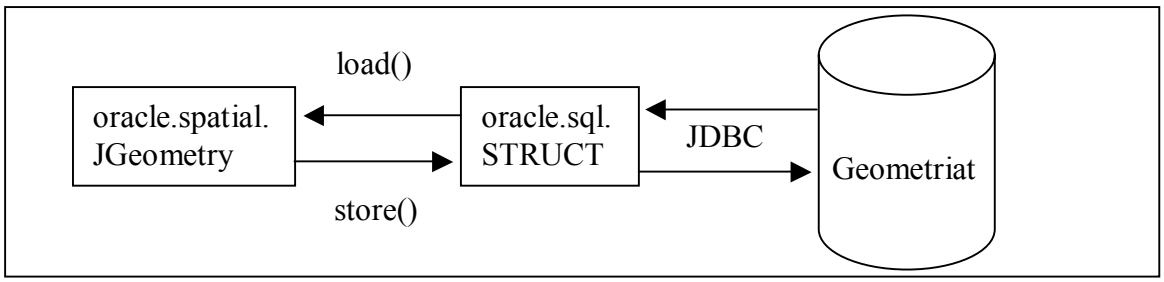
Kuvassa 29 esitetään indeksoitu spatiaalinen liitos SDO_join-funktion avulla, jossa haetaan kaikkien niiden asiakkaiden tiedot, joiden alueeseen kuuluva geometria sisältyy kilpailijoiden alueen geometriaan.

```
SQL> SELECT COUNT (DISTINCT ct.id)
FROM competitors_sales_regions comp, customers ct,
      TABLE
      (
            SDO_JOIN
            (
                  -- ensimmäinen taulu ja sarake
                  'competitors_sales_regions', 'geom',
                  -- toinen taulu ja sarake
                  'customers', 'location'
            )
      ) jn
WHERE ct.rowid=jn.rowid2 AND comp.rowid = jn.rowid1;
      COUNT(*)
```

Kuva 29: Spatiaalinen liitos jolla haetaan asiakkaat, joiden *mbb*-estimaatin alue sisältyy kilpailijan alueeseen (Kothuri & al., 2004).

3.7. Hakulauseet Java-rajapinnan kautta Oracle Spatial –järjestelmässä

Seuraavaksi perehdymme spatiaalisten hakulauseiden suorittamiseen Java-ohjelmointirajapinnan kautta Oracle Spatial -järjestelmässä. Suoritamme esimerkkihakuja kohdassa 3.6.1 luotuun tietokantatauluun *cola_markets*. Oracle Spatial-tietokannan käyttö Java-rajapinnan kautta vaatii JDBC-luokan käyttöä, joka mahdollistaa tietojen hakemisen tietokannasta (Kothuri & al., 2004). Lisäksi geometristen tietojen käsittely vaatii tietokannasta haettujen SDO_GEOMETRY-tyyppisten olioiden muuntamisen Java-ympäristön ymmärtämään muotoon JGeometry-luokan avulla, joka löytyy Oracle Spatialin asennuksen mukana levitettävästä Oracle Spatial Java API –kirjastosta *sdoapi.jar*. Kuvassa 30 esitetään malli, jonka mukaan geometrioiden haku Oracle Spatial-tietokannasta tapahtuu Java-rajapinnan avulla.



Kuva 30: Geometrioiden luku ja kirjoitus Java-rajapinnan kautta (Kothuri & al., 2004).

Hakulauseiden havainnollistamiseksi käytämme tulostustoiminnoiltaan kevennettyä versiota Kothurin & al. (2004) esittämästä SdoReader-ohjelmasta. Perehdytään aluksi tarkemmin varsinaiseen esimerkkiohjelmaan (ohjelman lähdekoodi liitteessä 1) JDBC-rajapinnan sekä geometrioiden käsittelyn selventämiseksi. Kuvassa 31 esitellään ohjelman toiminnan kannalta oleelliset *import*-lauseet. Java-kielen omista kirjastoista käytetään SQL-kielen sekä geometristen objektien käytön mahdollistavia kirjastoja. Oraclen Java-kirjastoista käytetään tietokantaliitännän tarjoavaa JDBC-ajuria

(ojdbc14.jar) sekä Oracle Spatial API-kirjastoa (sdoapi.jar) joka mahdollistaa tietokannasta noudetun spatiaalisen tiedon käsittelyn sekä tyyppimuunnokset.

```
import java.io.*;
import java.sql.*;
import java.util.*;
import java.awt.geom.*;
import java.awt.Shape;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.spatial.geometry.*;
```

Kuva 31: SdoReader-ohjelman käyttämät import-lauseet.

Kuvassa 32 esitellään ohjelman pääfunktio, jossa suoritetaan Oraclen JDBC-ajurin rekisteröinti ajurihallintaan, luodaan varsinainen tietokantayhteys, suoritetaan hakulause ja geometriatietojen tulostus printGeometries-funktion avulla ja suljetaan tietokantayhteys.

```
  public static void main(String args[]) throws Exception {
    String query = "";
    try {
      query = args[0];
    } catch (Exception e) {}
    //  Rekisteröidään Oraclen JDBC-ajuri
    DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
    //  Luodaan yhteys tietokantaan
    Connection dbConnection =
DriverManager.getConnection("jdbc:oracle:thin:@cs.joensuu.fi:1521
:sijainti", "username",
        "password");
    System.out.println("Got a connection:
"+dbConnection.getClass().getName());
    //  Suoritetaan SQL-lause
    printGeometries(dbConnection,query);
    //  Suljetaan tietokantayhteys
    dbConnection.close();
  }
```

Kuva 32: SdoReader-ohjelman pääfunktio.

Kuvassa 33 esitetään printGeometries-funktio, jonka avulla suoritetaan varsinainen SQL-kysely sekä tulostetaan kyselyn tuloksena saadun tietojoukon geograafisten objektien tiedot näytölle. Funktion alussa muodostetaan käyttäjän syöteparametrina antamasta SQL-kyselystä hakuoperaatio Statement-luokan avulla, sekä suoritetaan saatu kysely ja tallennetaan haun tulokset OracleResultSet-rakenteeseen. Tämän jälkeen saadut hakutulokset käsitellään rivi kerrallaan. Kunkin rivin sisältämä geometrinen objekti muunnetaan SDO_GEOMETRY-geometrialuokasta JGeometry-luokan objektiksi. Tämän jälkeen tiedot tulostetaan raakamuodossaan printGeom-funktion avulla. Funktion lopussa terminoidaan suoritettu kysely sekä tulostetaan yhteenveto tulostettujen tietojen määrästä.

```
static void printGeometries(Connection dbConnection, String query)
      throws Exception {
        long totalPoints = 0;
        long totalSize = 0;
        JGeometry geom;
        // Muodostetaan SQL-lause
        String sqlQuery = query;
        System.out.println("Executing query: '"+sqlQuery+"'");
        // Suorita kysely
        Statement stmt = dbConnection.createStatement();
        OracleResultSet ors = (OracleResultSet)
stmt.executeQuery(sqlQuery);
        //Käsittele tulokset
        int rowNumber = 0;
        while (ors.next()) {
          ++rowNumber;
          // Muunnetaan JDBC objekti tiedoista rakenteeseen
          STRUCT dbObject = (STRUCT) ors.getObject(1); // Rivi 18
          // Muunnetaan rakenne Geometry-objektiksi
          geom = JGeometry.load(dbObject); //Rivi 20
          System.out.println("Geometry # " + rowNumber + ":");
          // Tulosta geometria SDO_GEOMETRY formaatissa
          System.out.println (printGeom(geom));
        }
    stmt.close();
    System.out.println("");
    System.out.println("Done - "+rowNumber+" geometries extracted");
  }
```

Kuva 33: Hakukäsittely –ja tulostusfunktio printGeometries.

```
        static String printGeom (JGeometry geom) {
          String fg;
          // Hae geometrian tiedot
          int gType=geom.getType();
          int gSRID=geom.getSRID();
          int gDimensions=geom.getDimensions();
          boolean isPoint=geom.isPoint();
          // point
          double gPoint[]=geom.getPoint();
          // elementin tietotaulukko
          int gElemInfo[]=geom.getElemInfo();
          // Ordinaattitaulukko
          double gOrdinates[]=geom.getOrdinatesArray();
          // Formatoi JGeometry objekti SDO_GEOMETRY formaattiin
          int sdo_gtype=gDimensions * 1000 + gType;
          int sdo_srid=gSRID;
          fg = "SDO_GEOMETRY(" + sdo_gtype + ", ";
          if (sdo_srid == 0)
            fg = fg + "NULL, ";
          else
            fg = fg + sdo_srid + ", ";
          if (gPoint == null)
            fg = fg + "NULL), ";
          else {
            fg = fg + "SDO_POINT_TYPE(" + gPoint[0]+ ",
            "+gPoint[1]+", ";
            if (gPoint.length < 3)
              fg = fg + "NULL); ";
            else if (java.lang.Double.isNaN(gPoint[2]))
              fg = fg + "NULL), ";
            else
              fg = fg + gPoint[2]+"); ";
          }
          if (!isPoint & gElemInfo != null) {
            fg = fg + "SDO_ELEM_INFO_ARRAY( ";
            for (int i=0; i<gElemInfo.length-1; i++)
              fg = fg + gElemInfo[i]+", ";
            fg = fg + gElemInfo[gElemInfo.length-1] + "), ";
          }
          else
            fg = fg + "NULL, ";
          if (!isPoint & gOrdinates != null) {
            fg = fg + "SDO_ORDINATE_ARRAY( ";
            for (int i=0; i<gOrdinates.length-1; i++)
              fg = fg + gOrdinates[i]+", ";
            fg = fg + gOrdinates[gOrdinates.length-1] + ")";
          }
          else
            fg = fg + "NULL";
          fg = fg + ")";
          return (fg);
        }
}
```

Kuva 34: printGeom-tulostusfunktio.

Kuvassa 34 esitellään tulostusfunktio printGeom, joka on vastuussa kunkin tietokannasta haetun geometrisen objektin tietojen jäsentämisestä ja tulostamisesta. Funktion alussa geometrian tiedot haetaan muuttujiin JGeometry-luokan saantimetodien avulla. Tämän jälkeen tulostetaan geometrisen objektin tiedot. Käytettävä tulostusmuoto riippuu objektin tyypistä ja sen ominaisuuksista.

Kuvassa 35 suoritetaan hakulause, jossa haetaan kuvan 24 ensimmäisen hakuesimerkin mukaisesti kaikki *cola_markets*-tietokantataulun markkina-alueet SDO_GEOMETRY-tyyppisinä geometrioina, muunnetaan haetut geometriat JGeometry-muotoon sekä tulostetaan haettujen geometrioiden tiedot.

```
> java SdoPrint "SELECT shape FROM cola_markets"
Got a connection: oracle.jdbc.driver.T4CConnection
Executing query: 'SELECT shape FROM cola_markets'
Geometry # 1:
SDO_GEOMETRY(2003, NULL, NULL), SDO_ELEM_INFO_ARRAY( 1, 1003, 3),
SDO_ORDINATE_ARRAY( 1.0, 1.0, 5.0, 7.0))
Geometry # 2:
SDO_GEOMETRY(2003, NULL, NULL), SDO_ELEM_INFO_ARRAY( 1, 1003, 1),
SDO_ORDINATE_ARRAY( 5.0, 1.0, 8.0, 1.0, 8.0, 6.0, 5.0, 7.0, 5.0, 1.0))
Geometry # 3:
SDO_GEOMETRY(2003, NULL, NULL), SDO_ELEM_INFO_ARRAY( 1, 1003, 1),
SDO_ORDINATE_ARRAY( 3.0, 3.0, 6.0, 3.0, 6.0, 5.0, 4.0, 5.0, 3.0, 3.0))
Geometry # 4:
SDO_GEOMETRY(2003, NULL, NULL), SDO_ELEM_INFO_ARRAY( 1, 1003, 4),
SDO_ORDINATE_ARRAY( 8.0, 7.0, 10.0, 9.0, 8.0, 11.0))

Done - 4 geometries extracted
>
```

Kuva 35: Kaikkien markkina-alueiden haku Java-rajapinnan avulla.

Kuvassa 36 suoritetaan kuvan 24 toisen esimerkkihaun kaltainen haku, jossa haetaan tietokannasta *cola_a* alueen geometriatiedot sekä tulostetaan ne ruudulle.

```
> java SdoPrint "SELECT shape FROM cola_markets WHERE name='cola_a'"
Got a connection: oracle.jdbc.driver.T4CConnection
Executing query: 'SELECT shape FROM cola_markets WHERE name='cola_a''
Geometry # 1:
SDO_GEOMETRY(2003, NULL, NULL), SDO_ELEM_INFO_ARRAY( 1, 1003, 3),
SDO_ORDINATE_ARRAY( 1.0, 1.0, 5.0, 7.0))

Done - 1 geometries extracted
>
```

Kuva 36: *Cola_a* alueen geometrian haku Java-rajapinnan avulla.

4. Yhteenveto

Pyrin tutkielmassani luomaan katsauksen spatiaalisen tiedon käsittelyyn yleisellä tasolla. Tutkielman toisessa luvussa perehdyttiin spatiaalisen tiedon sekä geometrian peruskäsitteisiin. Käsittelimme spatiaalisen tietomallin ydinkohdat kuten spagettimallin, verkkomallin sekä topologiamallin mukaiset tietorakenteet, ja spatiaalisten hierarkioiden mallit. Tässä luvussa käsiteltiin myös spatiaalisissa tietokannoissa käytettäviä indeksejä, joista yleisin on jokin R-puun varianteista, sekä spatiaalisia tiedonsaantimenetelmiä. Lisäksi luvussa käsiteltiin Oracle Spatialin käyttämiä tietomalleja, sekä tiedon tallentamiseen Oracle Spatial –järjestelmässä käytännön esimerkkien avulla. Spagettimalliin perustuvien taulujen luonti on suhteellisen yksinkertaista, kun topologiamallin mukaisen kannan generointi olisi selkeästi vaikeampaa.

Kolmannessa luvussa käsiteltiin spatiaalisen tiedon hakemista ja analysointia, sekä spatiaalisten hakujen käsittelyn algoritmiikan että käytännön esimerkkien osalta. Luvussa käsiteltiin tarkemmin spatiaalisen tiedon haussa käytettäviä I/O-algoritmeja, hakulauseiden jäsentelyä QEP-suunnitelman avulla, spatiaalisia liitoksia sekä dynaamisia spatiaalisia hakuoperaatioita. Luvussa esitettiin lisäksi käytännön esimerkkejä spatiaalisen tiedon hakemisesta SQL-kyselykielen sekä Java-ohjelmointirajapinnan esimerkkien avulla. Kyselyiden suorittaminen vaatii tarvittavien spatiaalisten hakuoperaatioiden sekä Java-ohjelmointirajapinnan ominaisuuksien tuntemusta. Ohjelmistorajapintojen osalta tutkielmassa esiteltiin Oracle Spatial-järjestelmän avulla Java-ohjelmointirajapinnan käyttöä spatiaalisen tiedon analysoinnissa. Myös ohjelmointirajapintojen osalta vaihtoehtoisia toteutustapoja on monia, niin Java-rajapinnan kuin muiden kielienkin osalta, kuten *C++, C#, Visual Basic, Delphi* tai muut vastaavat yleiset ohjelmointikielet. Tutkielmassa esitetyt esimerkit ohjelmointirajapinnan sekä testiohjelman osalta ovat hyvin yksinkertaisia, eivätkä varsinaisesti anna oikeaa kuvaa siitä kuinka monimutkaista jäsentelyä spatiaalisen tietokannan tietojen käsittely oikeassa tuotantokäyttöön tarkoitetussa ohjelmistossa todellisuudessa vaatii.

Pääasiallisiksi ongelmiksi aihealueen käsittelyssä nousivat erilaisten tekniikoiden ja algoritmien valtava määrä sekä riittävän yksinkertaisten ja abstraktilla tasolla asioita käsittelevien lähdemateriaalien vähäisyys. Varsinkin spatiaaliseen tiedonkäsittelyyn

liittyvän algoritmiikan kenttä on laaja ja nopeasti kehittyvä. Tarkempi perehtyminen aihealueeseen vaatii sekä 20 vuotta vanhoihin tietokantateorioihin ja relaatiotietokantojen algoritmiikkaan sekä spatiaalisen tiedonkäsittelyn edellyttäviin spatiaalisen tiedonkäsittelyn erikoispiirteisiin perehtymistä.

Spatiaalisten tietokantojen ja spatiaalisen tiedon moninaiset käyttötarkoitukset ja aktiivinen tutkimustyö alueen algoritmiikassa sekä sovelluskohteissa lisäävät omalta osaltaan aiheen kiinnostavuutta. Varsinkin modernien karttajärjestelmien, paikannusjärjestelmien, mobiilisovellusten sekä www-käyttöön suunnattujen sekä hajautettujen paikannusjärjestelmien tekniikka tulee todennäköisesti kehittymään ja kasvamaan nopeasti, ja olisi täten sovelias alusta jatkotutkimukselle. Varsinkin kuluttajakäyttöön tarkoitettujen spatiaalista tietoa käsittelevien mobiililaitteiden, kuten navigaatio-ominaisuuksia sisältävien matkapuhelimien sekä autonavigaattoreiden käyttö on kasvanut räjähdysmäisesti kuluneen vuosikymmenen aikana.

Viitteet

Beckmann, N., Kriegel, H.P, Schneider, R., Seeger, B. (1990) The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. *ACM DIGMOD Record* 19(2), 322-331.

Freytag, J.C. (1987) A Rule Based View of Query Optimization. *ACM SIGMOD Record* 16(3), 173-180.

Greene, D. (1989) An Implementation and Performance Analysis of Spatial Data Access Methods. *Proceedings of 5th International Conference on Data Engineering*, 606-615.

Greiner, R. (1992) Learning efficient query processing strategies. *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 33-46.

Guttman, A. (1984) R-trees. A dynamic index structure for spatial searching. *Proceedings of the 1984 ACM SIGMOD international conference on Management of data,* 47-57.

ISO (2003) *ISO 19107:2003: Geographic information – Spatial schema.* International Standards Organisation, Geneva, Switzerland

Iwerks, G., Samet, S., Smith, K. (2003) Continuous K-Nearest Neighbor Queries for Continuous Moving Points with Updates. In proceedings of the international conference on very large databases, 512-523.

Jacox, E.H, Samet, H. (2007) Spatial Join Techniques. *ACM Transactions on Database Systems* 32(1), Article 7.

Kothuri, R., Godfrind, A., Beinat, E. (2004) *Pro Oracle Spatial*. Apress

Lee, K.C.K, Leong, H.V, Zhou, J., Si, A. (2005) An Efficient Algorithm for Predictive Continuous Nearest Neighbor Query Processing and Result Maintenance. *Proceedings of the 6th international conference on Mobile data management*, 178-182.

Malinowski, E., Zimányi, E. (2005) Spatial Hierarchies and Topological Relationships in the Spatial MultiDimER Model. *Lecture Notes in Computer Science: Database: Enterprise, Skills and Innovation*, 17-28. Springer, Berlin / Heidelberg

Malinowski, E., Zimányi, E. (2007) Logical Representation of a Conceptual Model for Spatial Data Warehouses. *Geoinformatica* 11/2007, 431-457.

Mokhtar, H., Su, J., Ibarra, O. (2002) On Moving Object Queries. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 188-198.

OGC (2003) *OGC reference model*. Open geospatial consortium Inc.

Oracle (2003) Spatial User's Guide and Reference, 10g Release 1 (10.1)

http://download-east.oracle.com/docs/html/B10826_01/toc.htm (9.5.2008)

Oracle (2005) Oracle Spatial Topology and Network Data Models, 10g Release 2 (10.2)

http://download.oracle.com/docs/pdf/B14256_01.pdf (13.5.2008)

Orenstein, J.A (1986) Spatial Query Processing in an Object-Oriented Database System. *Proceedings of the 1986 ACM SIGMOD international conference on Management of data,* 326-336.

Papadimitriou, C.H., Suciu, D., Vianu, V. (1999) Topological Queries in Spatial Databases. *Journal of Computer and System Sciences* 58(1), 29-53.

Paredaens, J., Van den Bussche, J., Van Gucht, D. (1994) Towards a Theory of Spatial Database Queries. *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 279-288.

Peuquet, D., Marble, D. (1990) *Introductory readings in geographic information systems.* CRC Press.

Raptopoulou, K., Papadopoulos, A.N., Manolopoulos, Y. (2003) Fast Nearest Neighbor Query Processing in Moving-Object Databases. *GeoInformatica*, 7(2), 113-137.

Rigaux, P., Scholl, M., Voisard, A. (2002) *Spatial databases with application to GIS.* Morgan Kauffman Publishers, San Fransisco.

Seeger, B., Kriegel, H.P. (1990) The buddy-tree: An efficient and robust access method for spatial database systems. *Proceedings of the Sixteenth International Conference on Very Large Data Bases*, 590-601.

Tao, Y., Papadias, D. (2002) Time-parameterized queries in spatial-temporal databases. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 334-345.

Tao, Y., Papadias, D. (2003) Spatial Queries in Dynamic Environments. *ACM Transactions on Database Systems,* 101-139.

Liite 1 – SdoPrint-ohjelman lähdekoodi

```
//SdoPrint ohjelman lähdekoodi (Kothuri & al., 2004)

import java.io.*;
import java.sql.*;
import java.util.*;
import java.awt.geom.*;
import java.awt.Shape;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import oracle.spatial.geometry.*;

public final class SdoPrint {

  public static void main(String args[]) throws Exception {

    String query = "";
    try {
      query = args[0];
    } catch (Exception e) {}
    //  Rekisteröidään Oraclen JDBC-ajuri
    DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
    //  Luodaan yhteys tietokantaan
    Connection dbConnection =
DriverManager.getConnection("jdbc:oracle:thin:@<server>:<port>:<datab
ase>", "<username>",
      "<password>");
    System.out.println("Got a connection:
"+dbConnection.getClass().getName());
    //  Suoritetaan SQL-lause
    printGeometries(dbConnection,query);
    //  Suljetaan tietokantayhteys
    dbConnection.close();
  }

  static void printGeometries(Connection dbConnection, String query)
throws Exception {

      long totalPoints = 0;
      long totalSize = 0;
      JGeometry geom;

      // Muodostetaan SQL-lause
      String sqlQuery = query;
      System.out.println("Executing query: '"+sqlQuery+"'");

      // Suorita kysely
      Statement stmt = dbConnection.createStatement();
      OracleResultSet ors = (OracleResultSet)
stmt.executeQuery(sqlQuery);


      //Käsittele tulokset
      int rowNumber = 0;
      while (ors.next()) {

        ++rowNumber;
```

```
        // Muunnetaan JDBC objekti tiedoista rakenteeseen
        STRUCT dbObject = (STRUCT) ors.getObject(1);

        // Muunnetaan rakenne Geometry-objektiksi
        geom = JGeometry.load(dbObject);


        System.out.println("Geometry # " + rowNumber + ":");

        // Tulosta geometria SDO_GEOMETRY formaatissa
        System.out.println (printGeom(geom));


      }
    stmt.close();
    System.out.println("");
    System.out.println("Done - "+rowNumber+" geometries extracted");

  }

      static String printGeom (JGeometry geom) {

        String fg;

        // Hae geometrian tiedot
        int gType=geom.getType();
        int gSRID=geom.getSRID();
        int gDimensions=geom.getDimensions();
        boolean isPoint=geom.isPoint();
        // point
        double gPoint[]=geom.getPoint();
        // elementin tietotaulukko
        int gElemInfo[]=geom.getElemInfo();
        // Ordinaattitaulukko
        double gOrdinates[]=geom.getOrdinatesArray();

        // Formatoi JGeometry objekti SDO_GEOMETRY formaattiin
        int sdo_gtype=gDimensions * 1000 + gType;
        int sdo_srid=gSRID;

        fg = "SDO_GEOMETRY(" + sdo_gtype + ", ";
        if (sdo_srid == 0)
          fg = fg + "NULL, ";
        else
          fg = fg + sdo_srid + ", ";
        if (gPoint == null)
          fg = fg + "NULL), ";
        else {

          fg = fg + "SDO_POINT_TYPE(" + gPoint[0]+ ", "+gPoint[1]+",
";
          if (gPoint.length < 3)
            fg = fg + "NULL); ";
          else if (java.lang.Double.isNaN(gPoint[2]))
            fg = fg + "NULL), ";
          else
            fg = fg + gPoint[2]+"); ";
        }

        if (!isPoint & gElemInfo != null) {
```

```
            fg = fg + "SDO_ELEM_INFO_ARRAY( ";
            for (int i=0; i<gElemInfo.length-1; i++)
              fg = fg + gElemInfo[i]+", ";
            fg = fg + gElemInfo[gElemInfo.length-1] + "), ";

          }
          else
            fg = fg + "NULL, ";
          if (!isPoint & gOrdinates != null) {

            fg = fg + "SDO_ORDINATE_ARRAY( ";
            for (int i=0; i<gOrdinates.length-1; i++)
              fg = fg + gOrdinates[i]+", ";
            fg = fg + gOrdinates[gOrdinates.length-1] + ")";
          }
          else
            fg = fg + "NULL";
          fg = fg + ")";

          return (fg);
        }

}
```