Koosteet tietovarastojärjestelmissä ja koosteiden toteutus materialisoiduilla näkymillä

Reetta-Leena Salmelainen

12.6.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Karkeasti jaotellen 1900 -luku oli operatiivisten tietojärjestelmien nousukautta ja 2000 -luvulle siirryttäessä oman nousukautensa ovat saaneet operatiivisista järjestelmistä tietoa kokoavat tietovarastojärjestelmät. Tietovarastojärjestelmiin kohdistuvia analysointitarpeita tukemaan kehitetyt tiedonmallinnusmenetelmät tähti- ja lumihiutalemalleineen omaksuttiin nopeasti järjestelmäkehittäjien keskuudessa. Sen sijaan tiedonkäsittelyn suorituskyvyn yhdeksi avaintekijäksi nykyään tunnustettu tietojen koostamiseen perustuva menetelmä on järjestelmäkehittäjien keskuudessa edelleen vaihtelevasti tiedostettu. Tässä tutkielmassa tarkastellaan kirjallisuuden perusteella tämän koosteiden hallintana tunnetun menetelmän merkitystä tietovarastojärjestelmien suorituskyvyn hallinnassa sekä Oracle 10g -tietokannanhallintajärjestelmän tarjoamaa materialisoituihin näkymiin perustuvaa tekniikkaa koosteiden toteuttamiseksi. Tutkielma on tarkoitettu erityisesti ohjelmistokehittäjille, jotka ovat siirtymässä tapahtumankäsittelyjärjestelmien kehitystyöstä tietovarastojärjestelmien toteutuksen pariin. Tutkielman luettuaan ohjelmistokehittäjällä tulisi olla käsitys koosteiden erilaisista käyttötarkoituksista, käsitys koosteisiin liittyvästä kyselyjen uudelleenkirjoituksesta sekä valmius lähteä etsimään syventävää tietoa koosteiden hallinnan toteutustekniikoiden osa-alueisiin liittyen.

ACM-luokat (ACM Computing Classification System, 1998 version): D.2.2, D.2.11, E.1, E.5, F.2.2, H.1, H.2, H.3, H.4

Avainsanat: tietovarastojärjestelmä, kooste, materialisoitu näkymä, kyselyjen uudelleenkirjoitus

Sisältö

1 Johdanto

Analyysien ja raporttien tuottamiseen kuluvan ajan väheneminen luetaan tietovarastojärjestelmien kriittisten tekijöiden joukkoon arvioitaessa tietovarastojärjestelmään kohdistuneesta investoinnista saatavaa hyötyä (Kimball *& al.*, 2008). Yksittäisten raportointi- ja analysointitarpeiden määrittäminen tavoitteellisine vasteaikoineen onkin olennainen osa tietovarastojärjestelmän vaatimusten keruuta. Tiedon saatavuuden parantuminen yhdistettynä tiedon paikkaansapitävyyden ja ajantasaisuuden kanssa ovat olennainen osa tietovarastojärjestelmän perimmäistä tarkoitusta mahdollistaa paremmat lähtökohdat ihmisistä ja prosesseista riippuvalle päätöksenteolle.

Tässä tutkielmassa tarkastellaan ensiksi luvussa 2 riittävän vasteajan käsitettä tietovarastojärjestelmissä sivuten samalla vasteaikahaasteiden taustalla vaikuttavia tekijöitä. Seuraavaksi siirrytään luvussa 3 tarkastelemaan tutkielman keskeisimpää asiasisältöä käymällä läpi koosteiden käyttötarkoituksia tietovarastojärjestelmissä sekä koosteiden hallinnan eri osa-alueita. Osa-alueiden tarkastelu suoritetaan perehtyen luvussa 4 Oracle 10g -tietokannanhallintajärjestelmän tarjoamaan materialisoituihin näkymiin perustuvaan koosteiden toteutustekniikkaan. Luvussa 5 muistutetaan, että kaikella saavutettavalla hyödyllä on myös hintansa. Kuten tietokantatauluille luotavien indeksien kanssa, koosteiden määrään ja sisältöön vaikuttavat päätökset pitää tehdä harkitusti. Luku 6 sisältää yhteenvedon tutkielman sisällöstä.

Tutkielman lukijan oletetaan hallitsevan relaatiotietokantojen tiedonmallintamisen perusteet myös tietovarastojärjestelmille tyypillisen dimensionaalisen mallintamisen osalta. Yhtä lailla relaatiotietokantoihin liittyen oletetaan lukijan hallitsevan SQL -kyselykielen keskeisen sisällön.

2 Vasteaikapaineet tietovarastosovellusten kulmakivenä

Epäilisin suurimman osan erilaisia tietojärjestelmiä käyttäneiden henkilöiden joskus käyttäneen sanaa hidas kuvatessaan yhteistyötä järjestelmän kanssa. Yleensä hitaudella viitataan pettymykseen, joka käyttäjälle on muodostunut tietojärjestelmän alittaessa käyttäjän odotukset järjestelmän nopeudesta vastata käyttäjän järjestelmälle antamaan ärsykkeeseen.

Ohjelmistojen kehitysprosessin alkuvaiheeseen kuuluvana loppukäyttäjien tärkeimmät olettamukset järjestelmien toiminnoista ja niiltä odotettavista vasteajoista muotoillaan vaatimuksiksi ohjaamaan suunnittelijoiden työtä. Tapahtumankäsittelyjärjestelmien odotetaan palvelevan tehokkaasti operatiivisen toiminnan eri tilanteissa. Tietovarastojärjestelmät kokoavat yleensä tietoa useasta tapahtumankäsittelyjärjestelmästä, ja tietovarastojärjestelmien perimmäisenä tarkoituksena mielletään mahdollisuus koostaa ja analysoida järjestelmän sisältämää tietomassaa eri näkökulmista.

Tässä luvussa pohditaan riittävän vasteajan käsitettä tietovarastosovelluksen kannalta sekä hahmotetaan tietovarastosovelluksien vasteaikoihin vaikuttavien tekijöiden kokonaisuutta.

2.1 Riittävät vasteajat

Tämän luvun johdannossa viitattiin tapahtumankäsittelyjärjestelmien ja tietovarastojärjestelmien erilaiseen lähtökohtaan järjestelmien perimmäisestä käyttötarkoituksesta. Tapahtumankäsittelyjärjestelmien sidonnaisuudesta liiketoimintaprosessin yksittäisiin vaiheisiin voidaan järjestelmän käyttötilanteisiin liittää ominaisuuksia kuten rutiininomaisuus, nopeus sekä verrattavan pienien tietomäärien hakeminen järjestelmästä ja vieminen järjestelmään yhdellä kertaa. Järjestelmän yksittäisessä toiminnossa käsiteltävät tiedot liittyvät usein vain kyseisen liiketoimintaprosessin vaiheen tarvitsemaan tietokokonaisuuteen.

Tapahtumankäsittelyjärjestelmien toimintojen rutiininomaisuudesta seuraa kasvava paine yksittäisten toimintojen vasteaikojen osalta. Yleensä yksittäisten toimintojen vasteaikavaatimukset liikkuvat sekunneissa tai sekunnin osissa, joiden vaikutus rutiinin sujuvuuteen on kuitenkin ilmeinen.

Esimerkkinä rutiininomaisesta käyttötilanteesta voi ajatella potilastietojärjestelmän sisältämän toiminnon potilaan tietojen hakemiseksi ja päivittämiseksi. Potilaan eri tutkimustulosten hakeminen ja potilastietojen päivittäminen ei voi muodostua perinteisessä 20 minuutin lääkärin vastaanottoajassa olennaiseksi osaksi tapaamista. Liiketoimintaprosessin sujuvuuden takaamiseksi tämän kaltaiset käyttäjän taidoista, järjestelmästä ja muista tilannetekijöistä riippuvat tekijät on tarpeen mukaan jopa eliminoitu. Esimerkkimme tapauksessa lääkärien käyttämä sanelutekniikka eliminoi 20 minuutin vastaanottoajalta suuren osan tietojen päivittämiseksi vaaditun lääkärin ja järjestelmän välisen yhteistyön sisältämistä riskeistä. Liiketoimintaprosessia tarkasteltaessa voidaan tilanne tulkita niin palveluntarjoajan kuin vastaanottajankin hyödyksi. Lääkäri voi keskittyä asiantuntijuutensa hyödyntämiseen ja asiakkaan saama vastine 20 minuutin lääkäriajasta suoritettavalle palvelumaksulle riippuu vähemmän lääkärin tekstinkäsittelytaidoista ja tietojärjestelmään liittyvistä tekijöistä.

Siirryttäessä tarkastelemaan vasteaikavaatimuksia tietovarastojärjestelmien näkökulmasta tarkastelutaso muuttuu. Laajennetaan esimerkkiä potilastietojärjestelmästä kansalliseksi tietovarastojärjestelmäksi, johon eri potilastietojärjestelmien sisältämiä tietoja koottaisiin kansallisen terveydentilan tarkastelemiseksi. Perimmäisenä tavoitteena tietovarastojärjestelmässä ei enää ole tarkastella nimenomaan yksittäisten yksilöiden tietoja, vaikkakin ideaalitilanteessa tietovarastojärjestelmälläkin tulisi olla mahdollisuus tarjota *porautuminen* tietojen yksityiskohtaisemmallekin tasolle. Yleensä tietovarastojärjestelmällä tavoitellaan kuitenkin erilaisten tietokantakyselyjen ja raporttien avulla muodostaa yhteenvetotietoja kokonaiskuvan hahmottelemiseksi eri tarkastelunäkökulmista.

Edellä mainittu tietovarastojärjestelmien ominaisuus koota tietoa monista eri järjestelmistä sekä lisäksi niiden ominaisuus säilyttää historiatietoja tapahtumankäsittelyjär-

jestelmiä kauemmin johtavat siihen, että tietovarastojärjestelmissä säilytettävän tiedon määrä saavuttaa nykyään usein jo teratavujen lukemat. Kun tämä ajatus tietomassasta yhdistetään tietovarastojärjestelmien perimmäisen tarkoituksen kanssa, päästään pohtimaan riittävän vasteajan käsitettä tietovarastojärjestelmien yhteydessä.

Ralph Kimball vertaa kirjoittamassaan esipuheessa tietovarastojärjestelmälle asettamaansa tavoitetta Google -tiedonhakupalvelun välittämään viestiin ”Voit hakea yksittäisen tiedon koko Internetin sisällöstä alle sekunnissa” (Adamson, 2006). Kimball asettaa tietovarastojärjestelmille yhtä lailla tavoitteen ”Sinun pitäisi pystyä olettamaan välittömiä tuloksia tietovarastojärjestelmään kohdistetuille tiedonhauille”. Muistuttaen tietovarastojen hakukoneita monimutkaisemmista tietorakenteista, vaaditusta tiedonkäsittelystä sekä loppukäyttäjille suunnattujen business intelligence (BI) -sovellusten yleisestä kehittymisasteesta, Kimball tulkitsee tähän mennessä otetun vain osittaisia kehitysaskelia kohti asettamaansa tavoitetta.

Käsite välitön on subjektiivisesti arvioitavissa oleva asia. Usein tietovarastojärjestelmien tietoja erilaisilla BI -välineillä koostavat käyttäjät ovat asennoituneet erilaisin odotuksin järjestelmän välittömään palautteeseen kuin tapahtumankäsittelyjärjestelmien käyttäjät. Väittäisin, että vain harvat BI -käyttäjät edes olettaisivat tietovarastojärjestelmän vasteajan olevan sekunnissa tai sen osissa, kuten Shneidermanin (1984) raportissa tapahtumankäsittelyjärjestelmien osalta todetaan loppukäyttäjien odottavan. Poikkeuksen edelliseen tietovarastojärjestelmienkin osalta muodostaa valmiiksi työstetyiksi tiedetyt raportit, joiden vasteaika tulisi koostua lähinnä raportin noutamisesta tietovarastojärjestelmästä.

Pendse (2008) on kerännyt yhteen määrityksiä sekä moniulotteisen OLAP (*On Line Analytical Processing*) -raportoinnin että sen tuottamiseksi käytettyjen sovelluksien *OLAP -välineiden* ominaisuuksia. Pendse (2008) yhteenvetää ominaisuudet niin sanottuun FASMI -testiin (*Fast Analysis of Shared Multidimensional Information*). FASMI -testi antaa ohjeellisia aikoja vasteaikojen subjektiivisen arvioinnin tueksi. Pendsen (2008) määrittelyn mukaan moniulotteiseen raportointiin tarkoitetun tietovarastojärjestelmän tulisi pääsääntöisesti pystyä palauttamaan tulokset käyttäjälle noin viiden

sekunnin sisällä. Toisaalta kaikkein yksinkertaisimpien tiedonhakujen tulisi tulokset palauttaa sekunnin kuluessa ja vain harvat tiedonhaut voisivat viedä kauemmin kuin 20 sekuntia.

Pendse (2008) viittaa myös riippumattomaan alankomaalaiseen tutkimukseen loppukäyttäjien reagoimisesta vasteaikoihin. Tutkimuksessa havaittiin käyttäjien tulkitsevan järjestelmän suorittaman tiedonhaun epäonnistuneen, jos järjestelmä ei palauttanut tuloksia alle 30 sekunnissa. Käyttäjien alttiutta turvautua suorituksen keskeyttämiseen Ctrl-Alt-Del -näppäinyhdistelmällä laski järjestelmän etukäteen antama varoitus pidempään kestävien tiedonhakujen yhteydessä. Mielenkiintoinen maininta on myös se, että vaikka käyttäjät saattoivat aluksi innostua ennen päiväkausia käsityötä vaatineen laskennan suorittumisesta minuuteissa, niin jonkin ajan jälkeen minuuttien vasteajat koettiin analysointia heikentäväksi tekijäksi. Odottelemiseen kyllästyneet käyttäjät kokivat paremmaksi vaihtoehdoksi vasteaikojen parantamisen lähemmäksi toivottua tasoa jopa yksittäisten analyysien laajuudesta tinkimällä.

Eri liiketoimintaprosessien tarkastelunäkökulmat varastoituun tietoon poikkeavat toisistaan. Vastaavasti loppukäyttäjien rooli vaikuttaa analysoinnille ja raportoinnille asetettuihin vasteaikavaatimuksiin. Kuten yllä tietovarastojärjestelmiin kohdistuvia loppukäyttäjien odotuksia vasteajoista tarkasteltiin, eivät odotukset vasteajan osalta ole kuitenkaan merkittävästi pienempiä tapahtumankäsittelyjärjestelmiltä odotettaviin vasteaikoihin verrattuna. Tapahtumankäsittelyjärjestelmistä merkittävästi eroten tietoa haetaan kuitenkin jopa teratavujen kokoisesta tietomassasta. Suuria tietomassoja koskevien tiedonhakujen lisäksi, tietoa tulisi muutamien sekuntien tai kymmenien sekuntien sisällä eri tavoin vielä koostaa ja käsitelläkin.

Tämä tasapainoittelu loppukäyttäjäkohtaisten odotuksien, tekniikan asettamien rajoituksien ja käsiteltävänä olevan tiedon määrän kanssa luo kokonaisuuden, jossa vasteaikatavoitteita ei kyetä tavoittelemaan enää samanlaisin suunnittelumenetelmin kuin tapahtumankäsittelyjärjestelmien yhteydessä. Tämän luvun seuraavassa kohdassa luodaan katsaus tähän kokonaisuuteen. Kokonaisuudesta tullaan erottamaan tekijä, joka monille tapahtumankäsittelyjärjestelmille on tuntematon, mutta tietovarastojärjestel-

mille sen on todettu olevan jopa välttämätön.

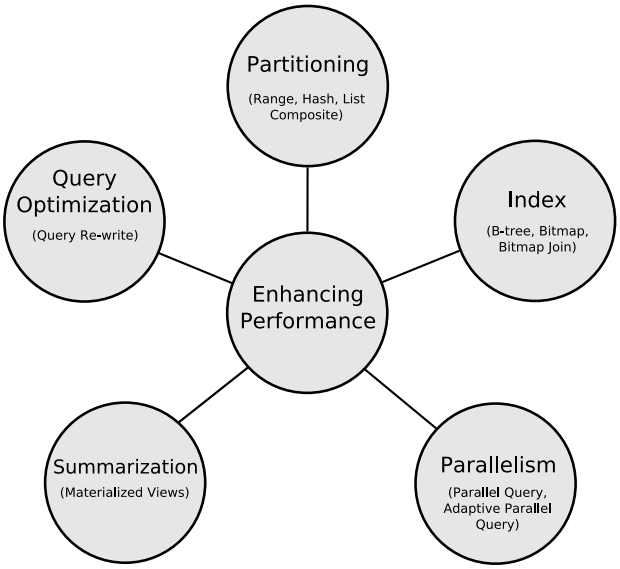
2.2 Koosteet vasteaikatavoitteiden saavuttamisessa

Tapahtumankäsittelyjärjestelmien sisältämä tietomäärä pysyy usein tietokannanhallintajärjestelmien kapasiteettiin nähden sen verran kohtuullisena, että ohjelmistokoodiin jääneet suorituskykyä heikentävät ohjelmointi- ja arkkitehtuuriratkaisut eivät muodostu ongelmaksi kuin haastavimmissa raporteissa tai massiivisimmissa eräajoissa. Muuten tietokannanhallintajärjestelmän optimointikeinoja sisältävät tietojenkäsittelyratkaisut voivat riittää tasapainoittamaan kömpelöimpienkin tietokantakyselyjen tapauksessa tilanteen siihen pisteeseen, että kyselyjen tulokset on saavutettu kaikesta huolimatta riittävässä vasteajassa.

Tietovarastojärjestelmissä lähtökohdat ovat haasteellisemmat. Tietovarastojärjestelmään kerättävien analysointien perustana olevien tietojen kokonaisuudet voivat sisältää satoja tuhansia tai miljoonia tietokantataulujen rivejä. Näihin kokonaisuuksiin kohdistuvat tarkastelunäkökulmat ovat järjestelmän kehitysvaiheessa usein vain osittain tiedossa. Lisäksi yhtä varmasti kuin organisaatioiden toimintaympäristössä tapahtuu muutoksia, tarkastelunäkökulmat ja tietojen koostamistarpeet päätöksentekoa tukevassa järjestelmässä myös muuttuvat.

Indeksointi nopeuttaa tietovarastojärjestelmissäkin tiedon etsintään ja rajaamiseen kuluvaa aikaa. Tiedonhaun muodostuessa yhteenkoostamisen myötä laajaa tietojoukkoa koskevaksi, muodostuu todelliseksi uudeksi haasteeksi tietojoukon käsittelyyn kuluva I/O -aika. Vaikka tiedon järjestely auttaisi tiedon rajaamisessa, ei suurempia tietomassoja käsittelevissä kyselyissä voida tavoitella riittäviä vasteaikoja kuin pienentämällä käsiteltävää tietomäärää, lisäämällä laskentatehoa tai tuottamalla kyselyjä, jotka hyödyntävät edellä mainittuja tekijöitä. Näistä tekijöistä syntyy kuvan 1 esittämä kokonaisuus suorituskyvyn parantamiseen liittyvistä tekijöistä tietovarastojärjestelmissä.

Kuvan 1 rinnakkaissuorituksella (*Parallelism*) viitatan mahdollisuuteen suorittaa tiedonkäsittelyyn liittyviä työjonoja rinnakkain joko yhden tai useamman suorittimen



Kuva 1: Suorituskyvyn parantaminen tietovarastojärjestelmissä (Stackowiak, 2007).

avulla. Varsinaisesta laskentatehon lisäämisestä voidaan puhua, kun suorittimien määrää tai olemassaolevien suorittimien tehoja kasvatetaan työjonojen rinnakkaisen käsittelyn parantamiseksi. Näennäisestä rinnakkaissuorituksesta puhutaan, kun yksittäisen suorittimen sallitaan jakaa palveluaikaansa vuorotellen eri tiedonkäsittelyjen kesken. Tästä käytetään myös nimitystä moniajo. Moniajoon perustuvan rinnakkaissuorituksen vasteaikavaikutukset ovat kuitenkin suhteellisia. Mitä useammalle työjonolle suoritin palveluaikaansa vuorotellen jakaa, sitä kauemmin yksittäisen työjonon valmistuminen kestää.

Suorittimien määrän kasvattamisen esteeksi voi nousta lisääntyvät kustannukset niin laitteiston kuin mahdollisten suorittimien määrään sidottujen lisenssimaksujen muodossa. Monikäyttäjäympäristön omaavissa tietovarastojärjestelmissä vähintään moniajon mahdollistaminen katsotaan kuitenkin välttämättömäksi tekijäksi. Kehitystyöhön sisältyy oma haasteensa. Ohjelmistokehittäjät kehittävät järjestelmiä ja testaavat tuotoksiaan usein järjestelmätoimittajan omassa kehitysympäristössä. Kehitysympäristö muodostuu usein *dedikoiduksi* ympäristöksi, jossa suorittimen palveluajasta ei järjestelmäkehittäjän työjonojen kanssa kilpaile kukaan muu. Kun tuotoksena syntyneitä viimeisimmät tuotannon luvut laskevia raportteja alkavat yhteisessä *jaetussa* tuotantoympäristössä tulostaa useampi samaan kokoukseen osallistuja, vasteajat kipuavat ulottu-

mattomiin järjestelmän suorittaessa kullekin käyttäjälle erikseen suuria tietomassoja käsittelevät laskennat.

Oletetaan edellä mainitussa tosielämän tilanteessa kunkin kokoukseen osallistujan raportin suorittuvan yhden suorittimen varassa toimittaessa kahdessa tunnissa. Jos järjestelmäkehittäjän tulisi tähän asti läpikäydyillä tiedonhaun tehostamiskeinoilla saavuttaa vasteaikaparannuksia, tarvitsisi järjestelmäkehittäjän tarkastaa indeksoinnin tilanteeseen sopivuus ja perustaa tekninen ratkaisu mahdollisimman tehokkailla laitteistoresursseilla varustetun moniprosessorikoneen tai -koneiden varaan. Käytännössä ratkaisu perustuisi prosessorien määrän kasvattamiseen suhteessa käyttäjämääriin. Loppujen lopuksi tämä lähestymistapa olisi yhtä viisasta kuin julkisen liikenteen lopettaminen ja investoiminen jokaisen työssäkäyvän ”autollistamiseen” sillä perusteella, että jokaisen yksilön työmatkaan kuluva aika optimoitaisiin. Todellisuudessa ratkaisu sisältäisi uusia hidastavia tekijöitä, kuten julkisen liikenteen esimerkin tapauksessa lisääntyneet liikenneruuhkat ja kokoukseen liittyvän esimerkin tapauksessa laitteiston ylläpidon lisäämät niin rahalliset kuin ajalliset kustannukset.

Tietomassojen paljoudesta johtuen tietovarastojärjestelmissä useimmiten kohdataan edellä kuvattu ”katto”, jossa rinnakkaissuorituksenkaan avulla ei saavuteta toimintatilanteille hyväksyttäviä vasteaikoja. Ainoaksi vaihtoehdoksi jää tehostaa vasteaikoja pienentämällä kunkin tiedonhaun tietomassan läpikäyntitarvetta tai käsittelyihin liittyvien laskentojen määrää. Edellä mainittuihin tavoitteisiin pääsemiseksi tarvittavia tehostuskeinoja ovat kuvan 1 esittämät partitiointi (*Partitioning*), koosteiden hallinta (*Summarization, Aggregation*) ja kyselyjen optimointi (*Query Optimization*).

Pienentämällä kyselyissä tarvittavien tietojen määrää vähennetään I/O -aikakustannuksien osuutta tiedonhaun kokonaisvasteajasta. Lähtökohtana on varmistaa, että etukäteen laaditut tiedonhakuja tuottavat tietokantakyselyt eivät sisällä suorituskykyongelmia aiheuttavia kyselyratkaisuja. Ohjelmistokehittäjän tulisi osata käyttää kyselykieltä ja kyselyn suorituksen kuvaavia työvälineitä tavalla, joka minimoi muodostettavien SQL -kyselyjen I/O -aikakustannukset rajaamalla ensin kyselyyn liittyvät tiedot ja suorittamalla taulujen väliset liitokset vasta rajauksien jälkeen.

SQL -kyselykieleen kohdistuvien heuristiikkojen joukkoon luettavaa tavoitetta rajata ensin tietojoukko ja sen jälkeen suorittaa vasta tietokantataulujen väliset liitokset havainnollistetaan liitteessä 1.

Vastaavaan kyselyjen I/O -vasteajan vähentämiseen tähtää myös partitiointi, jossa yhteenkuuluva samaan tietokantauluun säilöttäväksi tarkoitettu tietomassa voidaan jakaa fyysisesti toisistaan erillään sijaitseviin kokonaisuuksiin. Tietovarastojärjestelmissä tämä voi tarkoittaa esimerkiksi suuren faktataulun jakamista vuosi -tiedon perusteella eri levyille. Kyselyjen vaatimat I/O -aikakustannukset vähenevät, kun muodostettavissa kyselyissä tietokantataulujen liitoksissa käytettävien tietokokonaisuuksien koko pienenee. Edellinen perustuu tietokannanhallintajärjestelmän sisältämään tekniikkaan tunnistaa ja ottaa mukaan vain ne partitiot, joita tiedonhaku koskee. Partitiointia käytetään usein yhdessä rinnakkaissuorituksen kanssa muodostaen kokonaistulos eri partitioissa rinnakkaisesti laskettujen tulosten perusteella. Partitiointi yhdistettynä rinnakkaissuorittamisen kanssa on kyselyjen vasteaikojen nopeuttamisen ohella tärkeässä roolissa tietokannan tietoja päivitettäessä. Eri osioita faktataulusta voidaan päivittää usean suorittimen avulla samanaikaisesti. (Stackowiak, 2007)

Tietovarastojärjestelmien sisältämistä laskennoista löytyy useimmiten joukko kaikista massiivisimpia laskentoja, joissa vasteaikoja ei kaikkien edellisten tekijöiden muodostamaa kokonaisuutta säätämällä saada riittäviksi. Jäljelle jääväksi ratkaisevaksi tekijäksi kohoaa kuvan 1 koosteiden hallinta. Koosteiden hallinnassa lähtökohtana on koota ja laskea tietomassasta esille kokonaisuuksia, josta haasteellisimmat jaetussa suoritusympäristössä suoritettavat tiedonhaut voivat noutaa tietoa etukäteen valmiimmaksi työstetyssä muodossa. Käytännössä tämä tarkoittaa osan faktatauluihin kohdistuvien laskentojen suorittamista ennalta ja tulosten tallentamista erillisiin tietokantatauluihin. Usein nämä laskennat suoritetaan tuotantoympäristön off-line -aikana ladattaessa uusia tietoja tietovarastojärjestelmään. Off-line -ajalla järjestelmän ympäristö voidaan dedikoida ja varmistaa tarvittavien laskentojen suorittaminen tehokkaimmassa mahdollisessa ajassa. Tehokkaasti suoritetuista valmiista laskentatiedoista hyötyvät kaikki samoja tietoja tiedonhakuihinsa tarvitsevat järjestelmän loppukäyttäjät. Useimmiten koosteiden avulla raskaimmatkin tuntien vasteajat vaatineet tiedonhaut saadaan tehostettua mi-

nuuttitasolle. Lisäksi tuotettuihin koostetauluihin kohdistuvaa tiedonkäsittelyä voidaan tehostaa edelleen kohdistamalla koostetauluihin kuvan 1 muita tehostamiskeinoja.

Koosteiden avulla saavutettujen vasteaikaetujen kustannukset muodostuvat koosteita säilyttävien tietokantataulujen viemästä levytilasta ja koosteiden ylläpidon vaatimasta ajasta. Koosteisiin liittyvien kustannusvaikutuksien vuoksi, jokaisen koosteen käytön tulee olla harkittua. Yleensä lähtökohtana on varmistaa ensin vasteaikaongelmien ratkaistavuus muita optimointikeinoja käyttäen. Tarkasteluun nousee usein ensiksi indeksointi ja SQL -kyselyjen analysointi. Alan kirjallisuudessa koosteet mainitaan nykyään kuitenkin tehokkaimpana keinona parantaa tietovarastojärjestelmän suorituskykyä (Adamson, 2006; Kimball *& al.*, 2008; Lightstone *& al.*, 2007).

Suorituskyvyn optimointi muodostaa tietovarastojärjestelmässä laajan kokonaisuuden, jossa mikään tekijä ei ole yksinään toistaan parempi. Tätä kokonaisuutta Kimball *& al.* (2008) vertaavat orkesteriin, jossa solisti yksinään ei pysty kokonaisuutta muodostamaan. Henkilökohtaisesti en voi olla ajattelematta asiaa taloustieteen nyrkkisääntöjä lainaten markkinointimaailman peruskilpailukeinoihin liittyvän markkinointimix -ajattelumallin kautta (Borden, 1964). Jos oikean markkinointimix:n lähtökohtana on oikea tuotemix, hintamix ja saatavuusmix , liittyy tietovarastojärjestelmiin suorituskykymix koostuen osatekijöistä tietomix, kustannusmix ja saatavuusmix. Tuloksena on molemmisssa tapauksissa arvontuotanto, joka on saavutettavissa vain kaikkia tekijöitä tarkoituksenmukaisesti eri osa-alueilla yhdistelemällä.

Seuraavassa luvussa tarkastellaan tarkemmin koosteiden tarjoamia mahdollisuuksia tiedonhakujen vasteaikaongelmien ratkaisemiseksi. Myöhemmissä luvuissa tarkastellaan koosteiden teknistä toteutusta ja koosteiden käyttöön liitettyä valintakysymystä.

3 Koosteiden monet käyttötavat

Edellisessä luvussa tuotiin esille koosteiden rooli merkittävänä tekijänä suuria tietomassoja koskevien tiedonhakujen vasteaikoja parannettaessa. Koosteiden ratkaisuna vasteajan parantamiseen todettiin olevan tiedonhakuun tarvittavan tietomassan pienentäminen. Tämän todettiin erityisesti olevan välttämätöntä tiedonhauissa, joissa tiedonhaku kattaa laajoja tietomassoja sekä vaatii tietomassan tietojen käsittelyä esimerkiksi yhteenlaskemalla tietoja eri tarkastelunäkökulmista.

Tässä luvussa käsitellään tarkemmin koosteiden keskeisimpiä käyttötapoja. Samalla esille nousee näkymättömyyden käsite ja siihen liittyvä mahdollisuus sallia tietokannanhallintajärjestelmän uudelleenkirjoittaa tietokantakyselyjä. Lopuksi tarkastellaan koosteiden osuutta OLAP -raportoinnissa. Varsinaista koosteiden teknistä toteuttamista käsitellään seuraavassa luvussa.

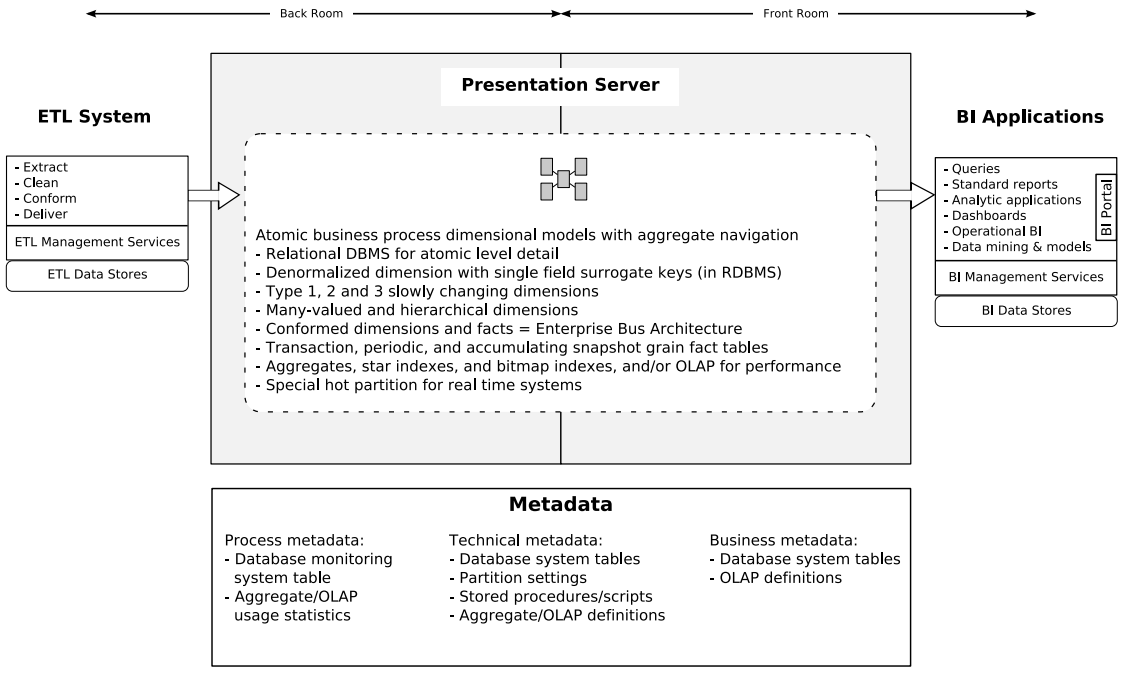
3.1 Lähtökohtana atomitason tiedonkeruu

Tietovarastoinnin toteutustavoista puhuttaessa kuullaan paradigmat jaettavan Bill Inmonin ja Ralph Kimballin lähestymistapoihin. Inmonin lähestysmistavassa tietovarasto nähdään koko yrityksen tai organisaation yhteisenä tietolähteenä, jossa tieto on varastoituna operatiivisille tietokannoille tyypilliseen tapaan kolmannessa normaalimuodossa. Yhteisestä tietolähteestä johdetaan yrityksen eri raportointi- ja analysointitarpeita varten erityisiä paikallisvarastoja (*data mart*), joiden tieto on dimensionaalisesti mallinnettu. Kimballin lähestymistavassa tietovaraston on sen sijaan kuvattu muodostuvan erikseen toteutettujen paikallisvarastojen kokonaisuudesta, jossa tieto on lähtökohtaisesti vain dimensionaalisesti mallinnettu. (1keydata, 2008)

Edellä mainittu karkea jaottelu on ollut harhaanjohtava. Kimball *& al.* (2008) välttävät uusimmassa teoksessaan kokonaan käyttämästä paikallisvarastoon viittaavaa käsitettä, koska he kuvaavat käsitteen saaneen liikaa heidän todellisista tarkoitusperistään harhaanjohtavia mielikuvia. Heidän mukaansa alkuperäistä yksittäisten paikallisvarasto-

jen kautta tapahtuvaa kehitystyömenetelmää tulkittiin väärin, mikä on johtanut paikallisvarastoihin liittyviin mielikuviin liian itsenäisistä osastokohtaisista tietovarastoista, joissa tieto usein mielletään myös olevan jo lähtöjärjestelmään nähden karkeammalle tasolle koostettua. Pahimpana uhkakuvana näiden mielikuvien kautta on käytännön toteutuksissa jouduttu tilanteeseen, jossa yli osastorajojen eri liiketoimintaprosesseja samanaikaisesti kattava analysointi eikä porautuminen tiedon atomisemmalle tasolle ole ollut enää tietovarastosta mahdollista. Nämä vaatimukset ovat kaksi Kimballin *& al.* (2008) mainitsemasta kolmesta tietovaraston sisältämälle tiedolle useimpien liiketoimintojen asettamista perusvaatimuksista. Kolmannen vaatimuksen muodostaa vaatimus yhdestä yhtenäisestä tietolähteestä.

Paikallisvarastojen sijaan Kimball *& al.* (2008) nostaa kolmen perusvaatimuksen täyttymisen edellytykseksi *liiketoimintaprosessien atomitasoisen dimensionaalisen mallintamisen*. Kuvassa 2 on esitetty esityspalvelimen arkkitehtuuri, josta käy ilmi atomitasoiseen mallintamiseen kytkeytyvät asiat.



Kuva 2: Esityspalvelimen arkkitehtuurimalli (Kimball *& al.*, 2008).

Ensimmäisenä esille nousee vaatimus löytää eri liiketoiminnoille yhteiset tarkastelunäkökulmat. Vaikka tietovarastojärjestelmän kehitystyö tapahtuisi iteratiivisesti osastoit-

tain, ei osastokohtaisesti muodostetut tähtimallit sovellu useita liiketoimintaprosesseja kattavaan analysointiin ellei osastojen erityistarpeisiin muodostetut tähtimallit kytkeydy toisiinsa yhteisten dimensioiden ja faktatietojen (*conformed dimensions and facts*) välityksellä. Toisekseen tietoihin porautumisen mahdollistamiseksi tulisi osastokohtaisissakin tiedoissa kerätä tietovarastoon tiedot tarkimmalla mahdollisella tasolla. Tämä atomitasoinen tiedonkeruu kasvattaa kuitenkin tietovarastojen tähtimallien tietomäärää, jolloin kolmanneksi avainasiaksi muodostuu edellisen luvun perustelujen pohjalta koosteiden hallinta. (Kimball *& al.*, 2008)

Inmon (1996) korostaa, ettei tapahtumankäsittelyjärjestelmien tavoin tietovarastojärjestelmien kehitystyössä vaatimukset ole koskaan vastaavasti määritettävissä etukäteen. Inmonin (1996) mukaan vaatimuksia voidaan tarkentaa riittävälle tasolle vasta sitten, kun loppukäyttäjä on päässyt kokeilemaan raportointi- ja analysointitoimintoja ensimmäisten tietovarastoon poimittujen tietojen perusteella. Jos tietovarastojärjestelmän kehitystyön aikaiset vaatimuksetkaan eivät ole etukäteen määritettävissä, on ymmärrettävää, että vielä vähemmän pystytään järjestelmäkehityksen aikana ennustamaan kaukaisemman tulevaisuuden vaatimuksia. Atomitasoinen tiedonkeruu voidaan siksi nähdä merkittävänä tekijänä varmistettaessa tietovarastojärjestelmän elinkaaren jatkuvuutta. Vaikka Inmon (1996) ei korosta dimensionaalista mallintamista, tuo hän esille yhtä lailla atomitasoisen tiedonkeruun tarpeen sekä atomitasoisista tiedoista muuttuvien vaatimuksien perusteella johdettavat yhteenvedot.

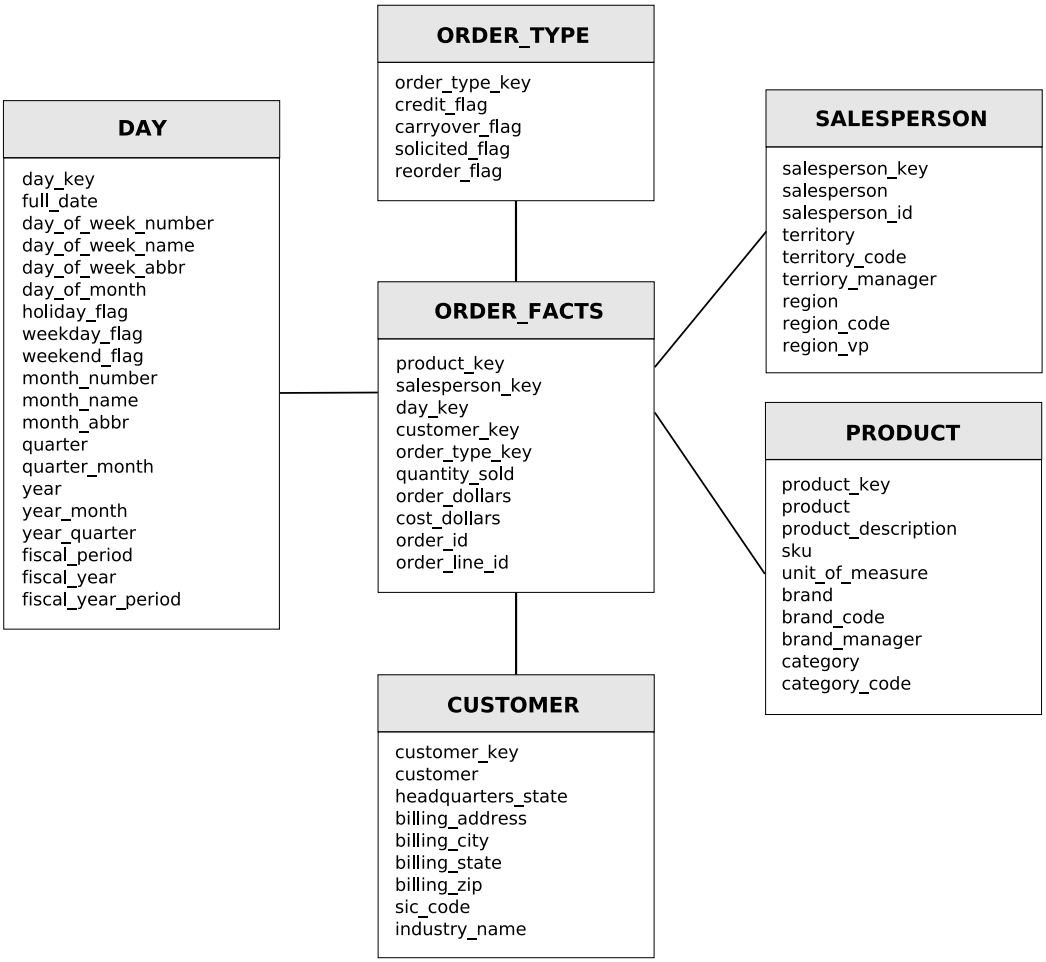
Inmonin ja Kimballin lähestymistavat ovat siis mielestäni tiedon mallintamisen tapaa lukuunottamatta lähellä toisiaan. Kummassakin tapauksessa tietovarastojärjestelmän tulee nykyään toimia yhtenäisenä kokonaisuutena eri BI -välineiden tiedonlähteenä. Seuraavassa kohdassa käsitellään koosteiden eri käyttötapoja tietovarastojärjestelmissä. Vaikka käsittely kohdistuukin tähtimallin kyselyjen optimointiin, on samoja periaatteita sovellettavissa myös haluttaessa koostaa tietoja Inmonin lähestymistavan mukaisesta kolmanteen normaalimuotoon mallinnetusta relaatiotietokannasta.

3.2 Koosteet tähtimallin kyselyjen optimointitekijänä

Perustuen edellä kuvattuun tietovaraston tarpeeseen sisältää sekä atomitason tietoa että valmiiksi koostettua tietoa, Adamson (2006) esittelee jaon tähtimalliin mallinnettavista atomitasoisista perusskeemoista (*base schema*) sekä koostetuista tiedoista koostuvista koosteskeemoista (*aggregate schema*). Edellinen jaottelu kaavioiden atomitasoisen tiedon ja koostetiedon välillä on osittain kärjistetty, sillä muodostettujen koostetaulujen sijoittuminen koosteskeemaan riippuu myös siitä, kuinka hyvin kyseinen koostetaulu noudattaa näkymättömyyden periaatetta. Koosteiden näkymättömyyttä ja sijoittumista koosteskeemaan käsitellään tarkemmin seuraavassa kohdassa. Tässä kohdassa käydään sitä ennen läpi Adamsonin (2006) esittelemiä erilaisia keinoja tiedon koostamiseksi.

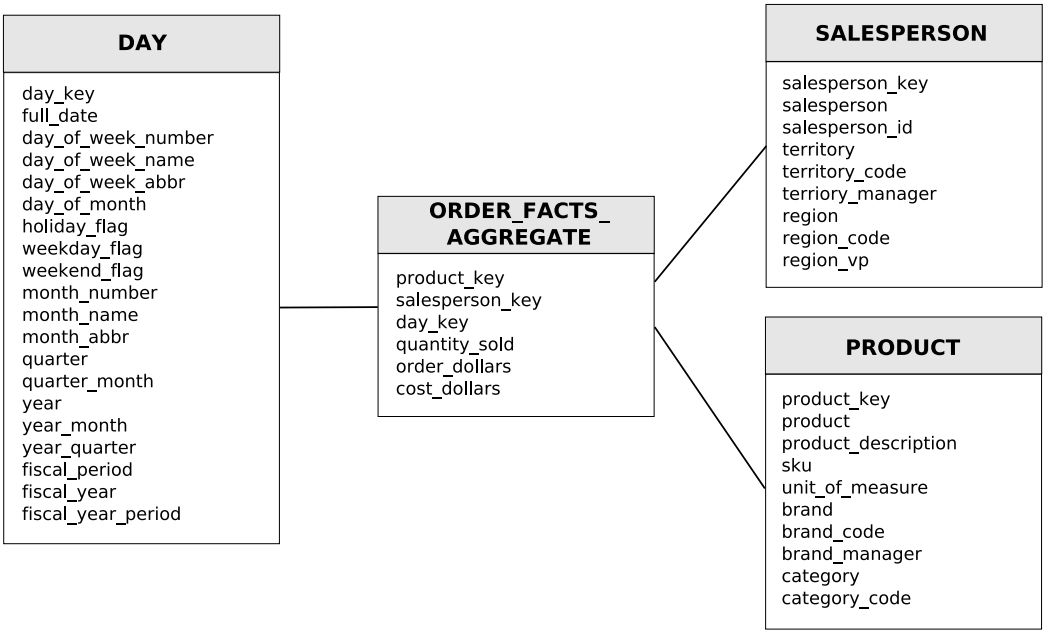
Kuvassa 3 on esitetty dimensionaalinen tähtimalli tilaustenhallinnan liiketoimintaprosessiin liittyen. Kuvatun tähtimallin faktataulu sisältää tilausten tiedot atomitasolla tarkimmassa mahdollisessa muodossaan.

Tietojen koostamistarpeeseen ja -tapaan vaikuttaa BI -välineillä suoritettavaan analysointiin tarvittava tietojen käyttötarve. Tilaustenhallinnan tapauksessa saatettaisiin analyyseissa ja raporteilla olla kiinnostuneita esimerkiksi päivätasoisesta tiedosta. Loppukäyttäjää saattaisi kiinnostaa kuinka paljon yhtenä päivänä on kaiken kaikkiaan myyty ja kuinka paljon myynneistä on yhtenä päivänä aiheutunut organisaatiolle kustannuksia. Jos kyseisiä raportteja tuotettaisiin järjestelmästä tiheästi, saattaisi sekä kyseisten tiedonhakujen suorittumisen että muiden samassa jaetussa ympäristössä suoritettavien tiedonhakujen kannalta olla järkevää koostaa tietoa valmiimpaan muotoon. Tällöin kuvan 3 perusskeemaan kuuluneesta tähtimallista voitaisiin muodostaa kuvan 4 havainnollistama koostetaulu, jossa faktataulun tiedoista on koostetauluun valmiiksi laskettu yhteen päiväkohtaiset myyntimäärät ja kustannukset. Jos perusskeeman faktataulussa rivejä myyntipäivää kohden olisi ollut esimerkin tapauksessa 10 000, voitaisiin koosteen avulla päästä esimerkiksi 1 000 riviin päivää kohden. Tämä vastaavasti tarkoittaisi sitä, että tietokannanhallintajärjestelmän tarvitsisi tiedonhaun suorittamiseksi läpikäydä koosteen kautta 1/10 osa siitä tietomäärästä, jonka se tiedonhaun suorittamiseksi joutuisi läpikäymään suoraan atomitason tiedoista. Adamson (2006) muistuttaa tietoa



Kuva 3: Atomitasoisen tiedon tähtimalli tilaus -prosessista (Adamson, 2006).

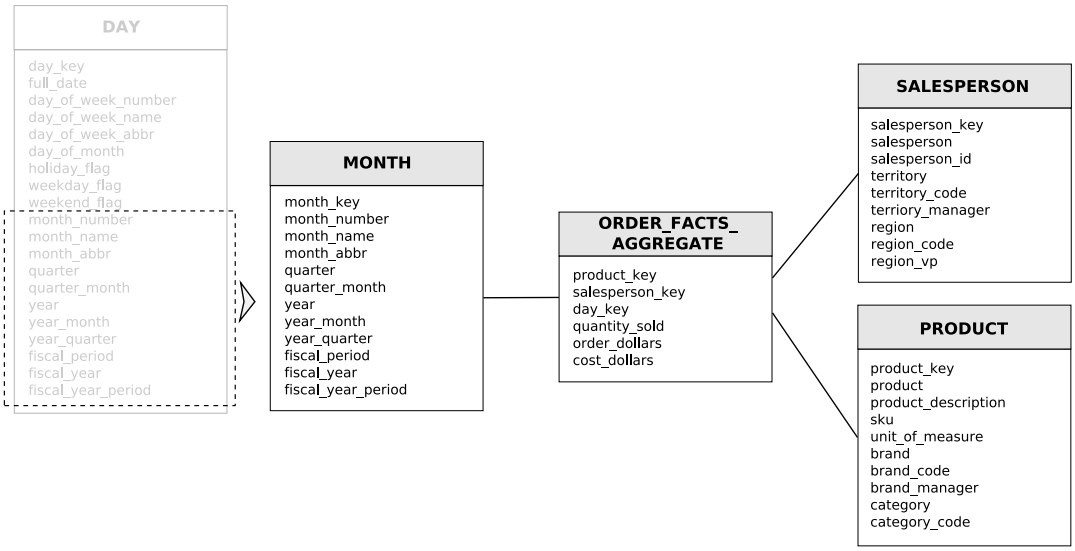
koostettaessa kuitenkin juuri siitä, että mitä karkeammalle tasolle tietoa koostetaan sitä harvempiin tiedonhakuihin se pystyy vastausta tarjoamaan.



Kuva 4: Koostettu faktataulu tilaus -prosessista (Adamson, 2006).

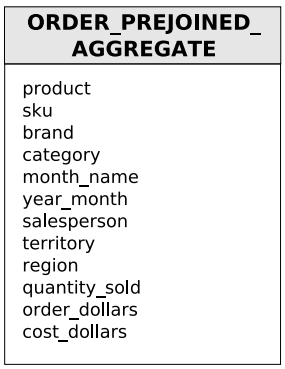
Kuvan 4 koosteessa oli yhteenvedetty vain faktataulun tietoja. Tällaisesta koosteskeemasta voidaan käyttää nimitystä *koostettu faktataulu* (*aggregate fact table*). Jos käsiteltäviä tietomassoja pitäisi edelleen tiivistää, olisi kuvan 4 koosteesta mahdollisuus muodostaa kuukausitason kooste lisäämällä skeemaan *koostettu dimensiotaulu* kuvan 5 osoittamalla tavalla. Sama kuukausitasoinen tieto olisi ollut laskettavissa myös kuvan 4 osoittamasta skeemasta, joten tässä tapauksessa menetetylle mahdollisuudelle koota päiväkohtaista koostetietoa tulisi olla vahvat perusteet.

Laajentaen edellisen koosteen muodostustapaa Adamson (2006) esittelee *esiliitetyn koosteen* (*pre-joined aggregates*). Esiliitettyjen koosteiden tarkoituksena on tiedonhaun käyttämän rivimäärän vähentämisen lisäksi vähentää myös tiedonhaussa tarvittavia taulujen välisiä liitoksia. Koosteen muodostamisen osalta tämä tarkoittaa sitä, että yhteenlaskettujen tietojen lisäksi muodostettavaan koostetauluun poimitaan dimensioistakin tiedonhaun tarvitsemat attribuutit kuvan 6 osoittamalla tavalla. Mahdollisesti saavutettuihin etuihin nähden esiliitetyillä koosteilla on myös haittapuolensa. Vaikka faktataulun tietoa koostettaisiin, pysyvät silti taulujen rivimäärät kohtuullisen suurina



Kuva 5: Koostettu faktataulu sekä koostettu dimensiotaulu tilaus -prosessista (Adamson, 2006).

ja dimensioiden attribuuttien toistaminen koostetaulun riveillä kasvattaa nopeasti taulun levytilan tarvetta. Esiliitettyä koostetta käytetään edelleen erityisenä toteutuskeinona sitä vaativissa tiedonhaun haastetilantessa. Esiliitettyjen koosteiden käyttöä on pidemmällä aikavälillä vähentänyt se, että useimmat käytössä olevat tietokannanhallintajärjestelmät ovat kehittäneet erityisiä optimointikeinoja tähtimalleissa tarvittavien taulujen välisten liitoksien käsittelemiseksi.



Kuva 6: Esiliitetty kooste (Adamson, 2006).

Tähän mennessä käsitellyt koosteet ovat yhteenlaskeneet faktataulun sisältämää tietoa halutun summatason saavuttamiseksi. Yhtä lailla tiedon koostamiseen viitataan, kun tietoa kootaan sitä muuttamatta perustason tähtimallista tiedonhaulle parempaan ase-

maan tai tiedon yhteenlaskemisen sijaan siitä koostetaan kokonaan uutta tietoa. Näiden koosteiden käyttötapojen yhteydessä Adamson (2006) käyttää koosteista nimityksiä *johdetut taulut* (*derived tables*) ja *uusia faktoja sisältävät taulut* (*tables with new facts*).

Johdetut taulut Adamson (2006) jaottelee *yhdistettyihin faktatauluihin* (*merged fact table*), *pivot -faktatauluihin* (*pivoted fact table*) ja *siivutettuihin faktatauluihin* (*sliced fact table*). Yhdistettyyn faktatauluun kootaan tietoa useammasta faktataulusta. Tätä tekniikkaa käytetään usein yhdistämään eri liiketoiminnoissa erillään kerättyä tietoa yli liiketoimintaprosessien tapahtuvan tiedonhaun mahdollisuuksien parantamiseksi. Vastakohtana edelliselle koosteen käyttötavalle esiintyy tietovarastojärjestelmissä joskus tarve erottaa osajoukko faktataulun tiedoista omaksi taulukseen. Adamsonin (2006) mukaan siivutetun faktataulun tekniikkaa käytetään kerätessä suuristä ja vaikeammin käytettävissä olevista tietomassoista tiedonhaun tarvitsema osajoukko fyysisesti paremmin tiedonhaun ulottuville.

Kahdessa edellä käsitellyssä koosteiden käyttötarkoituksessa faktataulujen tietoja ei koosteeseen millään tavalla työstetty toiseen tarkkuuteen. Enemmänkin kyse oli tietojen kopioinnista tiedon saattamiseksi paremmin tiedonhaun ulottuville. Viimeinenkin Adamsonin (2006) kuvaamasta johdettujen taulujen tyypeistä noudattaa samaa linjaa. Pivot -faktataulussa ideana on koostaa samat tiedot kuin faktataulussa, mutta tiedonhaun kannalta paremmassa muodossa. Käytännössä tähän tekniikkaan päädytään tilanteissa, joissa faktataulun rivi sisältää tiedon muodossa, jonka työstämiseksi tiedonhaussa jouduttaisiin käyttämään hitaita SQL -kyselykielen tai raportointivälineen tiedonmuuntamis -tekniikoita. Muuntamalla yhden faktataulun rivin sisältämä tieto useammalle riville tai päinvastoin, voidaan saada tieto paremmin vastaamaan tiedonhaun käyttämää esitysmuotoa.

Viimeisenä koosteiden käyttötarkoituksena käsitellään yllä mainittu uusia faktoja sisältävä taulu. Sitä käytetään muodostettaessa yhteenvetoja tiedoista, jotka eivät luonteeltaan ole perinteiseen tapaan eri tasoilla summattavissa. Adamson (2006) käyttää esimerkkinä faktataulua, joka pitäisi sisällään organisaation pankkitilien päiväkohtai-

set tiedot, kuten saldon. Pankkitilin saldo kuuluu osittain summattavissa oleviin tietoihin. Organisaatio olisi todennäköisesti hyvin kiinnostunut tietämään pankkitiliensä yhteenlasketun päivittäisen saldon, mutta kuukausitasolle päivittäisistä summista laskettu summatieto olisi käyttökelvoton. Sen sijaan kuukausitasolla organisaatio saattaisi olla kiinnostunut kuukausittaisesta päiväkohtaisten saldojen keskiarvosta, mikä on aivan uutta faktataulujen tiedoista johdettavaa tietoa. Adamson (2006) kärjistääkin määrittelyn uudesta faktasta seuraavasti. Aina kun tietoa koostetaan muussa tarkoituksessa kuin yhteenlaskemalla, muodostuu uusia faktoja.

Edellä käsiteltyjen koosteiden käyttötarkoituksien yhteydessä kerrattiin koosteiden tehtävä mahdollistaa paremmat vasteajat tiedonhauille. Koosteet käsittelevät faktataulujen sisältämiä tietoja tiedonhauille valmiimpaan muotoon ja usein samalla tietomäärältään pienemmäksi kokonaisuudeksi. Seuraavaksi käsitellään koosteisiin liitettyä näkymättömyyden käsitettä. Koosteiden näkymättömyydellä on merkittävä vaikutus siihen, kuinka tiedonhakujen käytettäväksi luodut koosteet päätyvät loppukäyttäjän tiedonhakujen vasteaikoja parantamaan.

3.3 Koosteiden näkymättömyys ja kyselyjen uudelleenkirjoitus

Koosteiden muodostamiselle on olemassa joukko suosituksia, joista osa on käytännön kautta muodostunut vaatimuksiksi asti. Yksi vaatimuksista vaikuttaa koosteiden käytettävyyteen ja koosteista aiheutuviin ylläpitokustannuksiin. Vaatimuksen mukaan jokaisen koostetaulun attribuutin surregaattiavainta lukuunottamatta tulisi olla osajoukko alkuperäisten atomitason fakta- ja dimensiotaulujen attribuuteista (Adamson, 2006).

Tämä attribuuttivaatimus ei ole koosteiden muodostamiselle kuitenkaan samanlainen ehdoton edellytys kuin myöhemmin käsiteltävä vaatimus tiedon eheydestä. Tarkasteltaessa edellisessä kohdassa käsitellyistä koosteiden käyttötarkoituksista esimerkiksi uusia faktoja sisältävää koostetta, voidaan havaita tilanteen olevan mainitun attribuuttivaatimuksen vastainen. Attribuuttivaatimus onkin tarkoitettu useimmiten käytettyjä ja useita tiedonhakuja samanaikaisesti palvelemaan tarkoitettuja koosteita varten, jot-

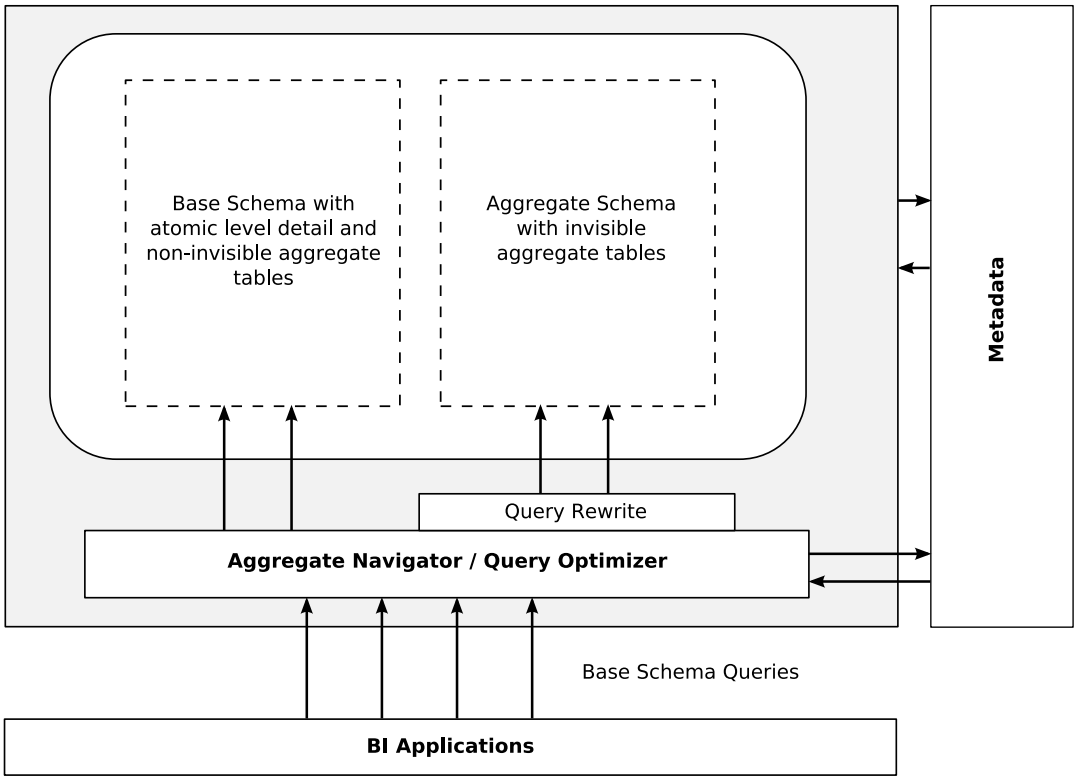
ka muodostetaan yhteiskäyttösten dimensioiden perusteella tietoja yhteenlaskemalla (Adamson, 2006; Kimball *& al.*, 2008).

Attribuuttivaatimuksella tavoitellaan koosteiden näkymättömyyttä ja näkymättömyyden mahdollistamaa koosteiden uudelleenkäytettävyyttä. Tavoitetila on, että loppukäyttäjille BI -välineiden kautta on näkyvissä mahdollisimman yksikäsitteinen edustus käytettävissä olevista tietovaraston sisältämistä tietojoukoista. Useimmiten näkyvissä on atomitason tiedot sisältävien tähtimallien faktataulut ja käytettävissä olevat tarkastelunäkökulmat dimensiotaulujen muodossa. Vaikka vasteaikojen parantamiseksi tietovarastoon olisi perustettu koosteita, ei loppukäyttäjän tulisi niiden olemassaoloa joutua tiedostamaan ja huomioimaan.

Adamsonin (2006) viittaaman attribuuttivaatimuksen tulisi edesauttaa tietokannanhallintajärjestelmän uudelleenkirjoittaa (*rewrite*) BI -välineen muodostamia atomitason tietoihin kohdistuvia tietokantakyselyjä käyttämään muodostettuja koosteita ja mahdollistaa siten koosteiden näkymättömyys. Tietokannanhallintajärjestelmän sisältämää tekniikkaa, joka mahdollistaa kyselyjen uudelleenkirjoituksen kutsutaan koostenavigaattoriksi (*aggregate navigator*).

Koostenavigaattorin toimintaideaa on havainnollistettu kuvassa 7. Samassa kuvassa palautetaan mieleen edellisessä kohdassa mainittu Adamsonin (2006) esittelemä tietomallien jakaminen osaksi perusskeemaa ja koosteskeemaa. Jakoa ei tehdä ainoastaan sen perusteella, onko kysymyksessä kooste vai ei. Sen sijaan ratkaisevammaksi tekijäksi mainittiin nimenomaan koosteiden näkymättömyys.

Adamson (2006) lajittelee osaksi perusskeemaa atomitason tiedot sisältävät tähtimallit sekä koosteet, joita koostenavigaattori ei pysty hyödyntämään kyselyjen uudelleenkirjoituksessa (*query rewrite*). Näiden koosteiden hyödyntämiseksi, tulee BI -sovelluksen kehittäjän määritellä koosteet loppukäyttäjien saataville perusskeeman tietomallien tavoin. Tämän kaltaisiksi näkymättömyyteen taipumattomiksi koosteiksi Adamson (2006) mainitsi edellisessä kohdassa mainituista koosteista johdetut faktataulut sekä uusia tietoja sisältävät faktataulut.



Kuva 7: Koosteiden jakaminen perusskeemaan ja koosteskeemaan.

Koosteiden jaottelu riippuu lopulta kuitenkin käytettävästä tietokannanhallintajärjestelmästä. Tarjolla olevilla tietokannanhallintajärjestelmillä on eriäviä ratkaisuja koosteiden hallinnan toteuttamiseksi. Ratkaisuista riippuen koosteiden toteuttamiselle asetetut rajoitukset näkymättömyyden mahdollistamiseksi vaihtelevat. Tietokannanhallintajärjestelmien tarjoajat myös kehittävät koosteiden hallinnan tekniikoitaan jatkuvasti.

Relaatiotietokannassa koostenavigaattori toteutetaan tietokannanhallintajärjestelmän tekniikoihin perustuvalla taustaratkaisulla (*back end*). Käytännössä tarjolla on myös välitason (*middle tier*) ratkaisuja sekä edustaratkaisuja (*front end*). Kimball *& al.* (2008) mainitsevat vaihtoehtoina koosteiden käsittelyn toteuttamisen OLAP -tietokantojen tai joidenkin loppukäyttäjälle suunnattujen BI -raportointi- ja analysointityökalujen sisäänrakennetuilla ratkaisuilla.

OLAP -tietokannoissa koostenavigaattorin idea on pääosin vastaava kuin relaatiotietokannanhallintajärjestelmienkin tarjoamissa ratkaisuissa. OLAP -tietokannanhallintajärjestelmissä koosteet varastoidaan kuitenkin relaatiotietokannan

sijaan erityisin OLAP -tietokannan käyttämin varastointitekniikoin. Usein OLAP -tietokannan menetelmiä käytettäessä tietokannan ylläpitäjällä on vähemmän vaikutusvaltaa siihen mitä koosteita muodostetaan ja miten koosteet fyysisesti varastoidaan tai indeksoidaan. Tekeekö OLAP -tietokannan käyttämä ratkaisu sitten paremmat valinnat kuin ylläpitäjä olisi tehnyt, riippuu niin käytettävistä tuotteista, käyttötilanteista sekä tietokannan ylläpitäjän koosteiden hallintaan liittyvistä teknisistä taidoista.

Esimerkiksi koosteiden valintaan (ks. luku 5) liittyen muodostui varhaisten OLAP -välineiden ongelmaksi erilaisten koosteiden muodostaminen liian herkästi eri käyttötilanteita varten. Tuloksena oli räjähdysmäinen tietokannan kokotarpeen kasvu ja hallinnoinnin vaikeudet. Monissa nykyisin käytössä olevissa OLAP -välineissä koosteiden valintaa on jo kehitetty havaittujen ongelmien seurauksena. Relaatiotietokannanhallintajärjestelmien ratkaisuissa koosteiden valinta pohjautuu loppukäyttäjän suorittamien tiedonhakujen analysointiin sekä manuaalisesti että erilaisten työkaulujen avulla. Tämä on jatkuva, haastava, aikaavievä ja teknistä ammattitaitoa vaativa tehtävä, joka tietovarastojärjestelmä -projektin lähtökohdista riippuen voi joskus puoltaa automatisoidumman OLAP -tietokannanhallintajärjestelmän tarjoamien ratkaisujen valintaan. Yleistä on myös käyttää erillistä OLAP -ratkaisua relaatiotietokannanhallintajärjestelmän rinnalla.

Loppukäyttäjän BI -välineiden toteuttama koosteiden hallinta voi tuntua helpolta vaihtoehdolta, ja tämänkin lähetymistavan kautta saavutetut vasteaikaedut voivat olla riittäviä. Haittapuoleksi hyötyjen rinnalle muodostuu koosteiden puutteet uudelleenkäytettävyydessä ja sitoutuminen valittuun raportointi- ja analysointityökaluun. Tietovarastoympäristöt muodostuvat usein hyvin suuriksi tietojärjestelmiksi ja loppukäyttäjien moninaisuudesta johtuen harvoin pystytään tarvittavia tietoja vain yhden loppukäyttäjälle suunnatun BI -työkalun kautta muodostamaan. Mitä lähemmäksi taustaratkaisua koostenavigaattori sijoitetaan, sitä useamman välitason ratkaisun ja edustaratkaisun käytettävissä samat koosteet ovat. Edellä mainitusta syystä Kimball *& al.* (2008) suosittelevat pitäytymään joko relaatiotietokannanhallintajärjestelmien tai OLAP -tietokantojen tarjoamissa vaihtoehdoissa. Jos esimerkiksi käytössä oleva relaatiotietokannanhallintajärjestelmä ei sisällä koostenavigaattoria, muodostuu toteutus perustelluksi myös lop-

pukäyttäjän BI -välineiden tarjoamilla ratkaisuilla.

Riippumatta koostenavigaattorin toteutustekniikasta on ajatus koostenavigaattorin tarkoituksesta yhtäläinen. Koosteet ovat fyysisiä erikseen varastoituja tietoja, jotka on luotu pääsääntöisesti tiedonhakujen vasteaikojen parantamiseksi johtamalla tietoja muista tiedoista. Karkealla tasolla koostenavigaattorin perimmäinen tarkoitus on automatisoida johdettujen tietojen tarvitsemia päivitysrutiineja sekä edesauttaa koosteiden hyödyntämistä eri loppukäyttäjän tiedonhauissa mahdollisimman läpinäkyvästi.

Adamson (2006) kiteyttää koostenavigaattorille asetettavat vaatimukset kahdeksaan eri vaatimukseen:

1. Koostenavigaattorin tulisi mahdollistaa loppukäyttäjien ja loppukäyttäjien käyttämien sovelluksien perustaa tiedonhakunsa koosteiden lähtötietoina oleviin perusskeeman tauluihin. Tämä vaatimuksen täyttymisen myötä minimoidaan BI -sovelluksiin kohdistuvaa ylläpitotyötä sekä poistetaan loppukäyttäjältä vaade itse ratkaista tiedonhakunsa reitti. Loppukäyttäjän tulisi pystyä määrittämään vain se, mitä tietoja hän haluaa analysoida tai käyttää raportoinnissa.

2. Perusskeeman tietokantatauluihin suunnattujen tiedonhakujen SQL -kyselyjen uudelleenkirjoituksen tulee tapahtua reaaliaikaisesti tiedonhaun suorituksen yhteydessä ja loppukäyttäjän kannalta näkymättömästi.

3. Koostenavigaattorin tulisi ylläpitää tietoa koosteiden tietojen ajantasaisuudesta lähtötietoihin nähden. Osa koosteista luodaan tietoisesti ylläpitämään koostetietoja esimerkiksi tietystä historiaan sijoittuvasta ajanjaksosta. Osa koosteista luodaan sen sijaan tarkoituksella ylläpitää yhtä tuoreita tietoa kuin koosteen perustana olevat atomitason faktataulut ylläpitävät. Jälkimmäisenä mainittujen koosteiden osalta koostenavigaattorin tulisi siirtää koosteet automaattisesti off line -tilaan siksi aikaa, kunnes atomitason faktatauluissa tapahtuneet muutokset on onnistuneesti koosteeseenkin heijastettu. Toisin sanoen koostenavigaattori ei saisi käyttää päivityksen tarpeessa olevaa koostetta kyselyjen uudelleenkirjoituksessa.

4. Koostenavigaattorin tulisi pystyä hallitsemaan uusien koosteiden lisääminen ja ole-

massa olevien koosteiden poistaminen aiheuttamatta muutostarpeita ja virhetilanteita olemassa oleviin raportoinnissa ja analysoinnissa käytettyihin tiedonhakuihin. Tämä vaatimus kuulostaa yksinkertaiselta, mutta Adamsonin (2006) mukaan vieläkin on olemassa koostenavigaattoreiden toteutustekniikoita, joissa koosteiden lisääminen ja poistaminen aiheuttaa huomattavia muutoksia ohjelmiin.

5. Koostenavigaattorin pitäisi pystyä tukemaan tähtimalleja, joissa faktataulu muodostuu koosteesta. Jos tietovarastojärjestelmässä aiotaan käyttää dimensioita ja faktatauluja yhdistävää koostetta, tulisi varmistaa koostenavigaattorin toteutustekniikan tätä tukevan. Jotkin koostenavigaattorit tukevat vain joko koostetun faktataulun käsittelyä tai dimensioita ja faktatauluja yhdistävän koosteen käsittelyä. Joihinkin tekniikoihin liittyy vaade lumihiutalemallin käyttämisestä, mikä Kimballin *& al.* (2008) mukaan tulisi olla ehdottoman harkittua ja rajattua vain muutamaan käyttötilanteeseen.

6. Koostenavigaattorin päätöksentekoon tarvitsemat tiedot tulisivat olla helposti määriteltävissä ja koostenavigaattorilla tehokkaasti saatavilla. Koostenavigaattoria varten ei pitäisi joutua suuriin manuaalisesti suoritettaviin tietojen vastaavuuksien tai hierarkioiden määrittelyyn. Ylläpidettävien tietojen osalta esimerkiksi tiedot koosteista, niiden riippuvuuksista perusskeeman tauluihin sekä tietokantataulujen koot tulisi olla nopeasti tietokannanhallintajärjestelmän ylläpitämistä metatiedoista saatavilla.

7. Koostenavigaattorin tulisi pystyä tarjoamaan kyselyjen uudelleenkirjoituksen mahdollisuus kaikille edustasovelluksille. Käytännössä tämä tarkoittaa, että koostenavigaattorin tulisi pystyä vastaanottamaan SQL -kyselyjä useista eri loppukäyttäjän BI -työkaluista ja uudelleenkirjoittaa kyselyjä BI -työkalusta riippumatta.

8. Koostenavigaattorin tulisi pystyä tukemaan eri tietovarastojärjestelmien arkkitehtuurisia vaihtoehtoja. Toisin sanoen useiden BI -työkalujen tukemisen lisäksi koosteiden käsittelijän tulisi pystyä tarjoamaan tukea useille taustaratkaisuille. Organisaation voimakas demograafinen hajautuminen voi johtaa niin faktataulujen kuin koosteidenkin hajauttamiseen eri tietovarastojärjestelmiin sekä maantieteellisesti että toiminnoittain. Nämä erilliset järjestelmät voivat olla lisäksi toteutettuja eri tietokannanhallintajärjestelmillä. Arkkitehtuuristen vaihtoehtojen hallinnoiminen asettaa haasteita niin kooste-

navigaattorin tarvitsemien metatietojen ylläpidolle kuin myös koostenavigaattorin kyvyille käsitellä eri tietokannanhallintajärjestelmien metatietoja.

Koostenavigaattorin toteuttamiseksi on siis olemassa eri tekniikoita taustaratkaisuista edustaratkaisuihin. Eri tekniikoihin liittyy erilaiset vaatimukset esimerkiksi kyselyjen uudelleenkirjoittamisen mahdollistamiseksi. Edellä mainittiin koostenavigaattoreiden varhaisille versioille tyypillinen kyselyjen uudelleenkirjoittamiseen liittynyt vaatimus dimensioiden ja faktataulujen attribuuttien samankaltaisuudesta. Adamsonin (2006) koostenavigaattorille asettamien vaatimuksien yhteydessä sivuttiin toista tärkeää vaatimusta tiedon eheyteen liittyen. Myös Kimball *& al.* (2008) painottavat tiedon yhtenäisyyden merkitystä ja sen koosteiden päivittämiselle asettamia haasteita. Ennen kuin siirrytään tarkastelemaan Oracle 10g -tietokannanhallintajärjestelmän ratkaisua koosteiden toteuttamiseksi, tarkastellaan vielä lyhyesti koosteiden suhdetta OLAP -raportointiin.

3.4 Koosteet OLAP -raportoinnissa

OLAP -raportoinnilla tarkoitetaan päätöksentekoa tukevaa raportointia, johon liitetään tyypillisesti vaatimukset moniulotteisesta suuria tietomääriä koskevasta tiedon tarkastelusta ja syvällisemmästä tietojen analysoinnista. Sen lisäksi, että OLAP -raportoinnissa kootaan perinteisen raportoinnin tapaan yhteenvetoja historiatietojen perusteella, toivotaan OLAP -raportoinnilta vastauksia myös haasteellisimpiin mitä jos -analyyseihin strategisen päätöksenteon tueksi. Moniulotteisuuden ja monimutkaisten tietojenkäsittelytarpeiden rinnalle nousee tiukka vaatimus toteuttaa OLAP -raportointi riittävillä vasteajoilla. (Connolly & Begg, 2005)

Kohdassa 2.1 viitattiin OLAP -raportoinnin ominaisuuksia koskevaan FASMI -testiin, jonka sisältämistä ominaisuuksista tarkemmin käsiteltiin raportoinnin vasteajoille asetettuja odotuksia. Nopeuteen liittyvää odotusta kuvasi FASMI -testin nimessä F -kirjain. Raporttien ja analyysien työstämisen nopeuden lisäksi FASMI -testin jäljelle jäävät neljä kirjainta viittaavat odotuksiin tietojen analysoitavuudesta (*analysis*), jae-

tun ympäristön vaatimista suojaustoimenpiteistä (*shared*), moniulotteisen tiedon käsittelystä (*multidimensional*) sekä tiedon saatavuudesta (*information*). (Pendse, 2008)

Koosteet muodostuvat varsinkin suuremmissa tietovarastojärjestelmissä olennaiseksi tekijäksi raportoinnin ja analysoinnin riittäviin vasteaikoihin pyrittäessä. OLAP -raportoinnille tyypillinen tietojen analysoiminen ja tiedon moniulotteinen tarkastelu lisää kuitenkin myös koosteiden hallinnan toteutustekniikalle asetettavia vaatimuksia. Olennainen osa tietojen analysointia on tietojen tarkastelu monesta eri tarkastelunäkökulmasta sekä porautuminen eri hierarkiatasoille halutun tarkastelutason saavuttamiseksi. Edellä mainittu luo tietokannanhallintajärjestelmille vaateen mahdollistaa tekniikat muun muassa dimensioiden ja hierarkioiden määrittelemiseksi. Kun tietoa tarkastellaan koosteiden kautta, nivoutuvat nämä OLAP -raportoinnin edellyttämät tekniikat osaksi myös koosteiden hallintaa.

4 Koosteiden hallinta Oracle 10g -tietokannanhallintajärjestelmässä

Edellisessä luvussa käsiteltiin koosteiden monimuotoisuutta sekä koosteiden hallintaan olennaisena osana kuuluvaa ajatusta koosteiden näkymättömyydestä. Lähtökohtana oli, että BI -sovelluksessa loppukäyttäjälle näkyvä kuvaus raportoinnin ja analysoinnin pohjalla olevista tiedoista olisi mahdollisimman selkeä ja ymmärrettävä. Tavoitetilanteessa loppukäyttäjä kohdistaisi tiedonhakunsa atomitasoiset tiedot sisältäviin tietomalleihin ja erilaisilla tekniikoilla toteutetut koostenavigaattorit uudelleenkirjoittaisivat tiedonhakujen käyttämät kyselyt tilanteeseen sopivia koosteita hyödyntäviksi.

Koostenavigaattorin toteutustekniikoiden kirjon mainittiin edellisessä luvussa olevan laaja ja eri ratkaisuja mainittiin kehitetyn niin BI -sovelluksiin, tietokannanhallintajärjestelmiin ja edellisten ääripäiden välimuodoiksi luettaviin tekniikoihin. Tässä luvussa keskitytään käsittelemään yhden kaupallisia tietokannanhallintajärjestelmiä toimittavan toimittajan ratkaisua koosteiden käsittelyn toteuttamiseksi. Käsittely kohdentuu tutkielman kirjoittamisajankohtana laajasti käytössä olevaan tietokannanhallintajärjestelmään Oracle 10g. Koosteiden luomisen, ylläpitämisen ja niiden näkymättömyyden tukemisen lisäksi tässä luvussa sivutaan perinteisten optimointikeinojen yhdistämistä koosteisiin.

Monet muutkin tietokannanhallintajärjestelmät tarjoavat omia tekniikoitaan koosteiden käsittelyn toteuttamiseksi. Tärkeätä tämän luvun avulla onkin muodostaa kokonaiskuva tekniikoihin liittyvistä osa-alueista. Koosteiden käsittelyä tukevien tietokannanhallintajärjestelmien välillä on tekniikoista eroteltavissa lähes vastaavat osa-alueet. Eri tekniikoiden termistöt ja menetelmät osa-alueisiin liittyen voivat kuitenkin vaihdella merkittävästi.

Tämä luku keskittyy kokoamaan yhteen Hobbsin *& al.* (2005), Powellin (2005) ja Oraclen (2005) mainitsemat keskeisimmät materialisoituihin näkymiin liittyvät tekniikat.

4.1 Materialisoidut näkymät koosteiden hallinnan ytimenä

Oracle 10g -tietokannanhallintajärjestelmässä koostenavigaattori muodostuu erilaisten ominaisuuksien yhdistelmästä, joka perustuu mahdollisuudelle luoda tietokantaan *materialisoituja näkymiä*. Muut ominaisuudet koostuvat tietokannanhallintajärjestelmän sisältämistä työkaluista käsitellä materialisoituja näkymiä ja analysoida niiden käyttöä ja käyttötarvetta. (Adamson, 2006; Hobbs *& al.*, 2005)

Operatiivisten järjestelmien kehitystyössä käytetään usein *näkymiä*, jotta tiedonhakujen tarvitsemien tietokantakyselyjen toteuttamiseksi olisi käytettävissä helppokäyttöisempiä ja selkeämpiä rajapintoja tietoihin. Näkymän käyttöön päädytään usein, kun tiedonhauissa toistuu useiden tietokantataulujen samanlainen liitostarve. Suorittaessaan näkymää käyttävää tietokantakyselyä tietokannanhallintajärjestelmä todellisuudessa suorittaa joka kerta myös näkymän määrityksen mukaiset taulujen väliset liittämiset ja tiedon rajaukset. Toisin sanoen näkymillä viitataan SQL -kyselykielellä muodostettuihin rajapintoina toimiviin tietokantaobjekteihin, jotka sovellukehittäjälle ja loppukäyttäjälle esittävät tiedon kolmanteen normaalimuotoon normalisoidun tietomallin sijaan muodossa, joka tukee paremmin toimintaympäristön mukaista loogista tietojen kokonaisuutta.

Tietojenkäsittelyn mahdollistaminen loogisempina kokonaisuuksina lisää järjestelmän käytettävyyttä esimerkiksi lisäämällä tietomallin tietojen ymmärrettävyyttä, vähentäen ohjelmistokehityksen virhetilanteiden mahdollisuutta keskittämällä usein käytettävien taulujen välisten liitosten määrää ja vähentäen tarvittavaa ylläpitotyötä taulurakenteiden muuttuessa. Tietovarastojärjestelmien osalta näkymät eivät kuitenkaan vastaa järjestelmän käytettävyyden keskeisempään haasteeseen tiedonhakujen vasteaikoja koskien. Itse asiassa näkymien käyttöä kehoitetaan tietovarastojärjestelmissä rajoittamaan ja edellä mainittu rajoitettavuus on kirjoitettu jopa ohjeeksi olla kokonaan käyttämättä perinteisiä näkymiä tietovarastojärjestelmissä (Kimball, 2008).

Ohjeistuksen olla käyttämättä tietovarastojärjestelmissä näkymiä ymmärtää edellisten lukujen perustelemasta tarpeesta laskea tiedonhakuja varten tietoja käyttötarkoitukseen

verraten valmiimmiksi koosteiksi. Vaikka perinteinen näkymä olisikin sisältänyt tietojen yhdistelemisen lisäksi tietojen yhteenkoostamista usein käytetylle käsittelytasolle, suorittaa tietokannanhallintajärjestelmä tarvittavat laskennat jokaisen näkymän käyttökerran kohdalla uudelleen. Materialisoidut näkymät vastaavat tietovarastojärjestelmien haasteeseen tallentaen muodostetun näkymän tulokset omaksi fyysiseksi tietokantatauluksi. Käytettäessä materialisoitua näkymää ei näkymän muodostamiseen kulu enää vastaavia laitteistoresursseja. Tiedot haetaan suoraan näkymän fyysisestä taulusta tai muodostetaan sinne valmiiksi laskettujen tietojen perusteella yleensä huomattavasti lähtötauluja pienemmän tietomäärän perusteella.

Materialisoidut näkymät ovat Oracle 10g -tietokannanhallintajärjestelmän vastaus edellisten lukujen valmiiksi laskettujen tietojen säilyttämiseksi tarkoitettujen koosteiden toteuttamiseksi. Termi materialisoitu näkymä pohjautuu kyseisten tietokantaobjektien luomiseksi tarvittavaan SQL -kyselykieleen luotuun syntaksiin, jota käsitellään tarkemmin seuraavissa kohdissa. Käsite materialisoitu näkymä on levinnut yleisesti muidenkin tietokannanhallintajärjestelmien käyttöön, vaikka monissa niissä on vastaavaa tarkoitusta varten olemassa olevia tietokantaobjekteja kutsuttu omilla termeillään. Esimerkiksi IBM:n DB2 -tietokannanhallintajärjestelmässä kyseisiä tietokantaobjekteja kutsutaan käsitteellä materialisoidut kyselytaulut (*Materialized Query Tables*) ja Microsoftin SQL Server -tietokannanhallintajärjestelmässä käsitteenä on esiintynyt indeksoidut näkymät.

Kuten käsitteen kooste kanssa, mielletään materialisoidut näkymät usein myös nimenomaan summatauluiksi, joissa tietoa on koostettu yhteenlaskemalla atomitasoista tietoa eri tekijöiden perusteella. On hyvä kuitenkin huomioida, että koosteita niin kuin myös materialisoituja näkymiä voidaan tuottaa muitakin käyttötarkoituksia varten perustuen käytännössä minkälaiseen SQL -kyselyyn tahansa. Esimerkiksi tiedon replikointitarkoituksessa saatetaan SQL -kyselyllä toistaa tietokantataulun sisältö sellaisenaan muokkaamatta ja yhdistelemättä sitä muiden taulujen kassa. Tavoiteltaessa materialisoitujen näkymien avulla muodostettujen koosteiden näkymättömyyttä, asetetaan SQL -kyselyille sen sijaan tiettyjä rajoituksia. Näitä rajoituksia käsitellään myöhemmin tässä luvussa.

Osana Oraclen koostenavigaattoria on muutama muukin tärkeä ominaisuus. Palauttaen mieleen tiedon eheysvaatimus lähdetiedot sisältävien faktataulujen ja koostetaulujen välillä, on Oracle 10g -tietokannanhallintajärjestelmässä sisäänrakennettuja tekniikoita materialisoitujen näkymien automaattiseksi päivittämiseksi ja tiedon eheyden varmistamiseksi. Lisäksi Oracle 10g -tietokannanhallintajärjestelmässä on sisäänrakennettu query rewrite -tekniikka kyselyjen uudelleenkirjoittamiseksi, dimensio -tietokantaobjekti johdettujen dimensioiden kuvaamiseksi sekä sovelluskehittäjien työvälineeksi tarkoitettu SQL Access Advisor auttamaan materialisoitujen näkymien luontitarpeen hahmottamisessa. Jokainen edellä mainittu ominaisuus käsitellään tarkemmin tämän ja seuraavan luvun aikana. Käsittely aloitetaan tarkentamalla seuraavassa kohdassa koosteiden luontia materialisoitua näkymää käyttäen.

4.2 Koosteen luominen materialisoidulla näkymällä

Edellisessä kohdassa tuotiin esille materialisoidun näkymän merkitys Oracle 10g -tietokannanhallintajärjestelmän koosteiden hallinnassa sekä selitettiin materialisoidun näkymän ero operatiivisissa järjestelmissä yleisemmin käytettyyn näkymään. Tässä kohdassa keskitytään tarkastelemaan luvussa 3 käsiteltyjen erilaisten koosteiden toteutusmahdollisuutta materialisoituja näkymiä käyttäen.

Kuvassa 8 on esitetty SQL-kyselykielen syntaksi materialisoidun näkymän luomiseksi. Syntaksista nähdään, että yksinkertaisimmillaan materialisoidun näkymän luomiseksi riittää näkymään poimittavan tiedon määrittävän SQL -kyselyn määrittäminen luontilauseeseen. Tavoiteltaessa koosteen perustamista, muodostuu SQL-kysely luonnollisesti koosteen tietosisällön määrittävästä SQL-kyselystä.

Kuvassa 9 on esitetty materialisoidun näkymän luontilause kuvassa 4 esitetyn koostetun faktataulun toteuttamiseksi. Samalla periaatteella olisi toteutettavissa niinikään kohdassa 3.2 esitetty esiliitetty kooste. Näkymään tiedot poimivan SQL -kyselyn lisäksi kuvan 9 luontilausekkeeseen on vaihtoehtoisista optioista liitetty mukaan optio BUILD DEFERRED. Edellä mainitun option tarkoituksena on kertoa tietokannnan-

```
CREATE MATERIALIZED VIEW [<schema>.]<materialized view>
     [ OF [<schema>.]<object type>]
     { ON PREBUILT TABLE [ WITH[ OUT ] REDUCED PRECISION ]
     | [ [ NO ] COMPRESS [ TABLESPACE <tablespace>] <segment stuff>
     | ORGANIZATION [ INDEX ... | EXTERNAL ... ] | CLUSTER ... ]
     [ column stuff ] [ partitioning stuff ] [ [ NO ] CACHE ] [ [ NO ] LOGGING ]
     [ [ NO ] PARALLEL [<n>] ] [ BUILD { IMMEDIATE | DEFERRED } ]
     }
     [ FOR UPDATE ]
     [ USING NO INDEX | USING INDEX [ TABLESPACE <tablespace> ] ]
     [ [ NEVER ] REFRESH {
               FAST | COMPLETE | FORCE
             | ON { DEMAND | COMMIT }
             | START WITH < date> | NEXT < date>
             | WITH { PRIMARY KEY | ROWID }
             | USING { ENFORCED | TRUSTED } CONSTRAINTS } ]
     [ { DISABLE | ENABLE } QUERY REWRITE ]
AS <subquery>;
```

Kuva 8: SQL -tiedonhallintakielen syntaksi materialisoidun näkymän luomiseksi (Powell, 2005).

hallintajärjestelmälle, että muodostettavan materialisoidun näkymän halutaan täydentyvän tiedoilla vasta seuraavan näkymän päivittymiseen johtavan tapahtuman seurauksena. Kyseisen option toisena vaihtoehtona on BUILD IMMEDIATE, joka on myös oletusoptiona silloin, kun kyseistä optiota ei erikseen luontilauseeseen liitetä. Oletuksena käytettävä vaihtoehto johtaa siihen, että kooste täydennetään tiedoilla heti materialisoidun näkymän määrittävän luontilauseen suorituksen yhteydessä. BUILD DEFERRED option yhteydessä mainittuihin näkymän päivittämiseen johtaviin tapahtumiin palataan seuraavassa kohdassa.

```
CREATE MATERIALIZED VIEW ORDER_FACTS_AGGREGATE_MW
     BUILD DEFERRED
AS SELECT
          product_key AS product_key,
          salesperson_key AS salesperson_key,
          day_key AS day_key,
          SUM(quantity_sold) AS quantity_sold,
          SUM(order_dollars) AS order_dollars,
          SUM(cost_dollars) AS cost_dollars
     FROM
          ORDER_FACTS
     GROUP BY
          product_key,
          salesperson_key,
          day_key;
```

Kuva 9: Kuvassa 4 esitetyn koostetun faktataulun toteuttavan materialisoidun näkymän luontilause.

Edellisessä luvussa esiteltiin myös koosteskeema, joka sisälsi koostetun faktataulun li-

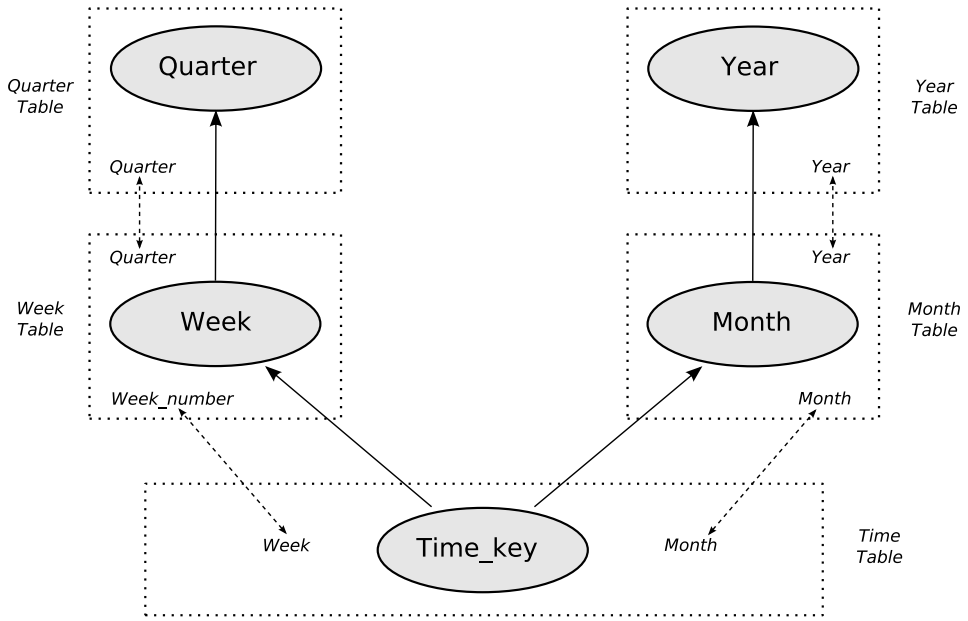
säksi koostetun dimensiotaulun. Dimensiotaulut toteutetaan faktataulujen tapaan tietokantatauluina, mistä voisi päätyä ajatukseen koostetun dimension toteuttamisesta koostettujen faktataulujen tapaan materialisoituja näkymiä käyttäen tai erottelemalla tarkastelutasot omiksi dimensiotauluikseen. Tähtimallin ajatus yhdestä dimensiotaulujen tasosta kärsisi ja jokaisesta tarkastelutasosta tulisi erikseen hallittava dimensiotaulu, mikä johtaisi lähemmäksi lumihiutalemallin ajattelutapaa. Edellä kuvatun tähtimallin kannalta väärän lähestymistavan sijaan Oracle 10g -tietokannanhallintajärjestelmän ratkaisu koostetun dimension tuottamiseksi on dimensio -objekti, jota käytetään myös lumiuhiutalemallissa taulujen välisten liitosten vähentämiseksi.

4.2.1 Dimensio -objekti

Dimensio -objektilla voidaan määrittää tietokantataulun tai useamman tietokantataulun sisältämien tietojen hierarkiatasoja ja niiden välisiä suhteita. Hierarkiarakenteiden lisäksi dimensio -objektilla voidaan määrittää lisäksi sarakkeiden välisiä riippuvuuksia. Dimensio -objektin määritysten perusteella tietokannanhallintajärjestelmä osaa käsitellä dimensiotaulua optimaalisimmissa kokonaisuuksissa käyttötilanteiden mukaan sekä hyödyntää tilannetta myös kyselyjen uudelleenkirjoituksessa ja tiedon analysoinnissa esiintyvien porautumistarpeiden yhteydessä. Käytännössä voidaan ajatella, että tietokannanhallintajärjestelmä pystyy käsittelemään denormalisoidun dimensiotaulun sisältämiä tietoja samaan tapaan kuin jos samat tiedot olisivat normalisoinnin seurauksena jaettu kokonaan fyysisesti eri tietokantatauluihin. Edellisten lukujen perusteella ymmärretään sitä vastoin tämän rajatumman tietojoukon käsittelyn mahdollistavan parempia vasteaikoja tiedonhauille.

Dimensio -objektien hallinnoimiseksi käytettäviä SQL -lauseita ei käydä tässä tutkielmassa läpi yhtä tarkalla tasolla kuin materialisoitujen näkymien osalta. Kuvassa 10 on kuvattu esimerkki dimensio -objektin luomisesta lumihiutalemallin mukaiselle dimensiolle. Esimerkistä käy ilmi dimensio -objektiin liittyvät keskeisimmät ominaisuudet.

Kuvasta 11 käy ilmi, kuinka LEVEL -määreellä kuvataan hierarkian tasot muodosta-



Kuva 10: Normalisoitu dimensio (Hobbs *& al.*, 2005) mukaillen.

```
CREATE DIMENSION time_dim
  LEVEL time_key IS time.time_key
  LEVEL month    IS month.month
  LEVEL quarter  IS quarter.quarter
  LEVEL year     IS year.year
  LEVEL week     IS week.week_number
  HIERARCHY calendar_rollup (
                    time_key CHILD OF
                    month    CHILD OF
                    year
                    JOIN KEY time.month REFERENCES month
                    JOIN KEY month.year REFERENCES year
            )
  HIERARCHY fiscal_rollup   (
                    time_key CHILD OF
                    week     CHILD OF
                    quarter
                    JOIN KEY time.week_number REFERENCES week
                    JOIN KEY week.quarter REFERENCES quarter
            )
  ATTRIBUTE time_key DETERMINES time.day_of_the_week
  ATTRIBUTE time_key DETERMINES month.month_name;
```

Kuva 11: Dimensio -objektin luontilause kuvan 10 normalisoidusta dimensiosta (Hobbs *& al.*, 2005) mukaillen.

vat sarakkeet. Tasojen määrittelemisen jälkeen voidaan yhteen dimensioon määrittää useampia hierarkioita tasojen välille. Useiden hierarkioiden määritteleminen on jopa välttämätöntä dimensioissa, joissa kaikkiin dimensio -tietoihin ei ole välttämättä liitettävissä arvoa kaikkiin tasoiksi määriteltyihin sarakkeisiin. Näin voi olla esimerkiksi organisaation toimipaikkatietoja ylläpitävän dimension kanssa, jossa kaupunkeihin liittyy toisissa maissa osavaltiotietoja ja toisissa ei. Hierarkian määrittävässä lohkossa esiinty myös määre JOIN KEY. Jos dimensio on mallinnettu denormalisoiduksi yhden tietokantataulun varaan rakentuvaksi dimensioksi, ei kyseistä määrettä tarvitse käyttää määrittämään tietokantataulujen välisiä liitoksia.

Hierarkioiden määrittämisen lisäksi mainittiin dimensio -objektilla pystyttävän määrittämään myös toisenlaisia tietojen välisiä riippuvuuksia. Kuvassa 11 tämä näkyy optiossa ATTRIBUTE. Esimerkin dimensio -objektin luomiseen käytettävässä lauseessa määritetään päivämäärätiedon määrittävän yksiselitteisen viikonpäivätiedon ja kuukausitiedon, jotka eivät ole määritetty hierarkiatasoja kuvaaviksi tiedoiksi.

Kuten moniin tietokantaobjekteihin liittyen, sisältää Oracle 10g -tietokannanhallintajärjestelmä erityisiä työkaluja erilaisten objektien hallintaa tukemaan. Dimensio -objektin hallintaa tukemaan löytyy tietokantapaketti DBMS_DIMENSION. Hobbs *& al.* (2005) mainitsevat paketista kaksi hyödyllistä proseduuria. Proseduurin DBMS_DIMENSION.DESCRIBE_DIMENSION avulla saadaan dimensio-objektin luontilauseen rakenteen kaltainen kuvaus tietokannan sisältämästä dimensio -objektista. Olettaen, että esimerkissä ollaan käsittelemässä skeemaa ”EASYDW”, SQL*Plus -työkalulla proseduuria kutsuttaisiin edellä kuvatun esimerkin dimensio -objektin tapauksessa seuraavasti:

```
set serveroutput on;
execute dbms_dimension.describe_dimension('EASYDW.time_dim');
```

Toisena hyödyllisenä proseduurina dimensio -objektiin liittyen mainitaan DBMS_DIMENSION.VALIDATE_DIMENSION. Tämän proseduurin käyttötarve korostuu dimensio -objektien ATTRIBUTE-määreitä käytettäessä. Toisin kuin

viite-eheyksien luomiseen tarkoitettujen CONSTRAINT -tietokantaobjektien tapauksessa, ATTRIBUTE -määreellä luodut riippuvuudet eivät ole samalla tavalla tietokannanhallintajärjestelmän automaattisen kontrollin alaisena. Esimerkiksi aikaisemminkin mainitun organisaatioiden toimipaikkatietoja ylläpitävän dimension tapauksessa, voisi ATTRIBUTE -määreellä olla kuvattu riippuvuus kaupungin ja postinumeron välillä. Riippuvuuden määrittämisestä huolimatta dimensioon pystytään lataamaan samalle kaupungille toisistaan eroavia postinumeroita. Proseduuria DBMS_DIMENSION.VALIDATE_DIMENSION voidaan käyttää dimensioon määritettyjen riippuvuuksien tarkastamiseen. Proseduuri kerää havaitsemansa epäkohdat tietokantatauluun DIMENSION_EXCEPTIONS, jonka olemassaolo tietokannan ylläpitäjän on varmistettava ennen proseduurin käyttöä. DIMENSION_EXCEPTIONS-taulun luomiseksi löytyy tietokannanhallintajärjestelmästä valmis skripti utldim.sql.

4.2.2 Koostetaulun muunnos materialisoiduksi näkymäksi

Luvun alussa mainittiin, että materialisoidut näkymät ovat Oracle 10g -tietokannanhallintajärjestelmän tekniikka, joka sisältää valmiita työkaluja koosteiden hallinnan automatisoimiseksi. Ennen materialisoitujen näkymien kehitystä ja yhä edelleen koostetietoja sisältäviä summatauluja on luotu myös tavallisiksi tietokantatauluiksi, jolloin automaattisten työkalujen sijaan tietojen ylläpitämiseksi tarvituista skripteistä on huolehdittu tietokannan ylläpitäjien toimesta. Suurempana puutteena tässä lähestymistavassa on se, että näin toteutetut koostetaulut eivät ole samaan tapaan koostenavigaattorilla kyselyjen uudelleen kirjoitukseen käytettävissä kuin materialisoidut näkymät. Mikäli näitä koostetauluja on haluttu loppukäyttäjän BI -työvälineen hyödyntävän, on koostetaulujen kuvaukset myös pitänyt määritellä loppukäyttäjän käsittelemiin tietokuvauksiin.

Kuvan 8 materialisoidun näkymän luontilauseen syntaksissa esiintyy vaihtoehtoinen optio ON PREBUILT TABLE. Kyseisen option avulla voidaan tavallisena tietokantatauluna toteutettu koostetaulu muuntaa materialisoiduksi näkymäksi, jonka jälkeen koostetaulu on tuotavissa myöhemmässä kohdassa käsitellyin menetelmin ja ehdoin

kyselyjen uudelleenkirjoituksen hyödynnettäväksi. Käytännössä optio ON PREBUILT TABLE ohjeistaa tietokannanhallintajärjestelmän muodostamaan olemassa olevalle taululle materialisoitujen näkymien tekniikoiden tarvitseman metadatan.

```
CREATE MATERIALIZED VIEW ORDER_FACTS_AGGREGATE
     ON PREBUILT TABLE WITH REDUCED PRECISION
AS SELECT
          product_key AS product_key,
          salesperson_key AS salesperson_key,
          day_key AS day_key,
          SUM(quantity_sold) AS quantity_sold,
          SUM(order_dollars) AS order_dollars,
          SUM(cost_dollars) AS cost_dollars
     FROM
          ORDER_FACTS_AGGREGATE
     GROUP BY
          product_key,
          salesperson_key,
          day_key;
```

Kuva 12: Tietokantataulun muuntaminen materialisoiduksi näkymäksi.

Kuvassa 12 on esitetty materialisoidun näkymän luontilause tilanteessa, jossa kuvan 5 koostettu faktataulu olisi ollut jo toteutettu tavallisena tietokantatauluna. Esimerkin lause ei poikkea huomattavasti kuvan 9 esittelemästä luontilauseesta. Kuvista on nähtävissä silti yksi merkitsevä seikka ON PREBUILT -option käyttöön liittyen. Vaikka Hobbs *& al* (2005) muistuttavatkin, että objektien nimennässä olisi hyvä tuoda esille viittaus objektin tyypistä, ei ON PREBUILT -option kanssa voi kuvan 9 osoittamaan tapaan lisätä nimen loppuun MV -määrettä. Muunnettaessa olemassaoleva tietokantataulu materialisoiduksi näkymäksi, on nimenä käytettävä tietokantataulun olemassaolevaa nimeä.

Toinen ON PREBUILT -option käyttöön liittyvä merkittävä asia ei edellä mainittuja luontilauseita vertailemalla yhtä lailla käy ilmi. Luontilauseiden sisältämät SQL -kyselyt materialisoidun näkymän sisällön määrittämiseksi eivät poikkea toisistaan. Muunnettaessa olemassa oleva tietokantataulu materialisoiduksi näkymäksi, ei tietokannanhallintajärjestelmä tarkasta, että tietokantataulun olemassaoleva sisältö vastaisi ON PREBUILT -optiota käyttävän luontilauseen SQL -kyselyn tulosjoukkoa. Tietokannanhallintajärjestelmä käyttää kyselyjen uudelleenkirjoituksessa luontilauseen SQL -kyselyä päätöksenteon perusteena, joten jos tietokannan ylläpitäjä ei ole tietokantataulun nykyisen sisällön täsmäävyyttä varmistanut, jää riski uudelleenkirjoituk-

sen mahdollisuudesta tuottaa harhaanjohtavia tuloksia kyseisen taulun kohdalla. Tämä asia on huomioitava myös kyselyjen uudelleenkirjoittamiseen liittyviä järjestelmäasetuksia määritettäessä, joita käsitellään tarkemmin jäljempänä.

ON PREBUILT-optioon liittyy tarkenteet WITH/WITHOUT REDUCED PRECISION. Oletuksena toimii tarkenne WITHOUT REDUCED PRECISION, joka asettaa tiukemmat ehdot SQL -kyselyn kuvaaman tietosisällön ja olemassaolevan tietokantataulun rakenteelliselle vastaavuudelle. Kyseistä tarkennetta käytettäessä ei sallita luontilauseen SQL -kyselyn kuvaamien sarakkeiden määrän tai tietotyypin poikkeavan olemassaolevaan tietokantatauluun verrattaessa. Huolimatta käytetystä tarkenteesta, tulee SQL -kyselyssä käytettyjen sarakkeiden alias -nimien täsmätä olemassaolevan tietokantataulun sarakkeiden nimien kanssa.

ON PREBUILT -option käyttöön liittyy myös tietokannan fyysistä suunnittelua koskevia rajoitteita. Muunnettaessa olemassaoleva tietokantataulu materialisoiduksi näkymäksi, tulee materialisoitu näkymä perustaa samaan skeemaan kuin olemassaoleva tietokantataulu. Lisäksi, jos luontilauseen SQL -kyselyssä ei ole mainittu kaikkia olemassaolevan taulun sarakkeita, ei näihin sarakkeisiin voi määritellä NOT NULL -rajoitetta ellei kyseisiin SQL -kyselyn ulkopuolisiin sarakkeisiin määritetä myös DEFAULT -optiota tietojen täydentämistä varten.

4.3 Materialisoitujen näkymien tietojen ylläpito

Edellisessä kohdassa mainittiin materialisoidun näkymän luontilauseen yhteydessä olevaa mahdollisuutta määrittää hetki, jolloin näkymä täydennetään luontilauseen sisältämän SQL -kyselyn tiedoilla. Toisena vaihtoehtona esiintyi tietojen täydentäminen seuraavan näkymän päivittämiseen johtavan tapahtuman yhteydessä. Tässä kohdassa käsittelemme materialisoidun näkymän luontilauseen optioita, joilla vaikutetaan materialisoidun näkymän päivitystapahtuman käynnistymiseen ja tapaan, jolla päivittäminen suoritetaan.

4.3.1 Virkistämistapa

Materialisoitujen näkymien päivittämiseen liittyen edellisessä luvussa tuotiin esille tiukka edellytys tiedon eheydelle koosteiden ja koosteiden lähtötiedot sisältävien taulujen välillä. Kun lähtötietoja sisältävien taulujen tiedot päivittyvät, ei koostenavigaattori saisi uudelleenkirjoittaa tiedonhakujen kyselyjä päivittämättömiä tietoja sisältäviä koosteita käyttäväksi. Rajoite ei aina ole näin suoraviivainen. Varsinkin kun koosteita luodaan parantamaan yksittäisiä raportointi- ja analysointitarpeita, määritetään koosteelle harkittu, kyseiseen käyttötarpeeseen sopiva päivitystiheys.

Materialisoidun näkymän luontilauseen REFRESH -lohkossa on mahdollisuus määritellä materialisoidulle näkymälle koosteen käyttötilanteeseen sopiva päivityskäytäntö. Päivityskäytännöt on jaettavissa neljään eri tyyppiin, joihin liittyen kuvan 8 luontilauseen syntaksista löytyy lausekkeet NEVER, COMPLETE, FAST ja FORCE.

Option NEVER REFRESH avulla voidaan toteuttaa staattinen materialisoitu näkymä. Staattisen materialisoidun näkymän tiedot pysyvät koskemattomina, vaikka niihin yritettäisiin kohdistaa Oraclen tietokantapakettien sisältämiä valmiita päivitysrutineeja. Tämän pävityskäytännön mukaisia materialisoituja näkymiä käytetään esimerkiksi tilanteissa, joissa koosteen halutaan ylläpitävän historiatietoja tietyltä ajanjaksolta. Ne ovat myös käyttökelpoisia silloin, kun materialisoidun näkymän päivittämisen halutaan sisäänrakennettujen päivitysrutiinien sijaan tapahtuvan omatekoisten päivitysskriptien välityksellä.

Muiden kuin staattisten näkymien osalta tilanne on toinen. Materialisoitujen näkymien halutaan reagoivan perusskeeman tauluissa tapahtuviin muutoksiin tietyin väliajoin. Lähestymistapoja materialisoitujen näkymien päivittämiseksi on kaksi. Toinen vaihtoehdoista on, että materialisoidun näkymän tiedot muodostetaan kokonaisuudessaan uudestaan suorittamalla luontilauseen SQL -kysely uudelleen ja korvaamalla näkymän sisältö uudella tulosjoukolla. Materialisoidun näkymän luontilauseessa tämä päivityskäytäntö on valittavissa optiolla REFRESH COMPLETE. Suurien tietokantataulujen sekä työläitä laskentoja ja monia liitoksia sisältävien koosteiden tapauksessa tämä päi-

vitysmuoto voi viedä kuitenkin huomattavan paljon aikaa. Varsinkin jos muutoksia perusskeeman tietokantatauluihin on tapahtunut vähän, on materialisoidun näkymän luominen tyhjästä liian raskas toimenpide päivityksen suorittamiseksi päivityksiin varatun aikaikkunan ollessa rajallinen.

Vaihtoehtona materialisoidun näkymän uudelleen muodostamiselle on *inkrementaalinen päivittäminen*, johon materialisoidun näkymän luontilauseessa viitataan optiolla REFRESH FAST. Inkrementaalisessa päivittämisessä materialisoituun näkymään heijastetaan vain perusskeeman tietokantatauluissa tapahtuneet muutokset. Kun inkrementaaliseen päivittämiseen lisätään ajatus sovituin väliajoin tapahtuvasta päivittämisestä, ymmärretään inkrementaalisen päivittämisen vaativan muitakin keinoja toteutuakseen. Tiedot perusskeeman tietokantatauluihin tapahtuneista muutoksista tulee pystyä säilyttämään koosteen päivittämiseen johtavaan tapahtumaan asti. Lisäksi inkrementaalinen päivittäminen asettaa rajoituksia materialisoiduissa näkymissä käytetyille SQL -kyselyille. Vaihtoehtoja muutoslokien ylläpitämiseksi ja inkrementaalisen päivittämisen asettamia rajoitteita käsitellään tarkemmin myöhemmin.

Neljästä materialisoidun näkymän luontilauseen tunnistamasta päivityskeinosta on käsittelemättä optio REFRESH FORCE. Tämä optio toimii oletusoptiona tapauksessa, että valinnaista REFRESH -lohkoa ei luontilauseeseen lainkaan määritetä. Määritettäessä materialisoidun näkymän päivityskeinoksi REFRESH FORCE, yrittää tietokannanhallintajärjestelmä suorittaa päivityksen ensisijaisesti inkrementaalisesti. Jos materialisoidun näkymän tapauksessa ei kuitenkaan kaikki inkrementaalisen päivityksen asettamat ehdot täyty, suoritetaan päivitys muodostamalla materialisoitu näkymä kokonaan uudelleen.

4.3.2 Virkistämisfrekvenssi

Edellä on käsitelty REFRESH -lohkon sisältämiä vaikutuskeinoja materialisoitujen näkymien päivityksessä käytettyyn tapaan liittyen. Seuraavaksi käsitellään REFRESH -lohkosta löytyviä materialisoitujen näkymien päivityshetkeen vaikuttavia optioita ON

DEMAND, ON COMMIT ja START WITH - NEXT.

Tietovarastojärjestelmissä tietojen ei useimmiten oleteta päivittyvän reaaliajassa tapahtumankäsittelyjärjestelmään nähden. Jos muutoksia atomitason tietoja sisältäviin faktatauluihin heijastetaan tietojen lähtöjärjestelmistä yksitellen, voidaan REFRESH -lohkossa valita päivityksen ajankohdan määrääväksi optioksi ON COMMIT. Kyseistä optiota käytettäessä materialisoitu näkymä päivitetään aina, kun atomitason faktatauluihin kohdistetut päivitystapahtumat vahvistetaan esimerkiksi osana tapahtumankäsittelyjärjestelmän tapahtumaa. Optiota ON COMMIT käytetään harvemmin tietovarastojärjestelmissä, koska taulujen suuren koon vuoksi käytäntö saattaa hidastaa merkittävästi tapahtumien käsittelyä olipa sitten kysymyksessä tapahtumankäsittelyjärjestelmän tapahtumaan kytketty faktataulun päivitys tai ON COMMIT -option käyttö faktatauluihin kohdistuvien päivitysajojen yhteydessä.

ON COMMIT -option sijaan tietovarastojärjestelmissä käytetään päivityshetken määrittämisesssä yleisemmin optiota ON DEMAND, joka mahdollistaa ylläpitäjälle suuremmat vaikutusmahdollisuudet materialisoitujen näkymien päivitysajankohtaan. Tietovarastojärjestelmille on tyypillistä atomitason faktataulujen päivittäminen yöllisen päivityksiin varatun aikaikkunnan puitteissa. Materialisoitujen näkymien päivittäminen sitä vastoin ajastetaan käynnistymään faktataulujen päivittymisen jälkeen. Oman haasteensa päivityksien hallintaan luovat 24/7 -järjestelmät sekä globaalit monien aikavyöhykkeiden piirissä toimivat järjestelmät.

ON DEMAND -optiota käytettäessä materialisoidun näkymän päivittäminen voidaan käynnistää manuaalisesti käyttämällä kantapaketin DBMS_MVIEW kolmea proseduuria.

```
DBMS_MVIEW.REFRESH
DBMS_MVIEW.REFRESH_DEPENDENT
DBMS_MVIEW.REFRESH_ALL_MVIEWS
```

Käytettävä proseduuri valitaan käyttötarkoituksen mukaisesti. Proseduurilla DBMS_MVIEW.REFRESH_ALL_MVIEWS voidaan kaikki materialisoidut nä-

kymät päivittää kerralla. Yleensä DBMS_MVIEW.REFRESH_ALL_MVIEWS -proseduuria käytetään vain suuria eräajoja sisältävien päivitysten yhteydessä. Toiset kaksi proseduuria mahdollistavat ylläpitäjälle enemmän vaikutusvaltaa esimerkiksi sopivia päivitysajankohtia materialisoiduille näkymille päätettäessä. DBMS_MVIEW.REFRESH_DEPENDENT -proseduurille ylläpitäjä voi antaa parametrina koosteiden lähdetauluna toimivan taulun nimen. Tämän päivitysmuodon yhteydessä koostenavigaattori huolehtii kaikkien niiden materialisoitujen näkymien päivittämisestä, joihin tietoa parametrina annetusta taulusta on johdettu. Jos käyttötarve vaatii vain tiettyjen materialisoitujen näkymien päivittämistä, viimeinen käsitellyistä proseduureista DBMS_MVIEW.REFRESH on oikea valinta. Kyseiselle proseduurille ylläpitäjä voi luetella niiden materialisoitujen näkymien nimet, jotka hän päivitettäväksi haluaa.

Erittäin yleinen tietovarastojärjestelmissä käytetty päivityshetken määrittävä optio on START WITH - NEXT. Kyseisen option avulla materialisoitu näkymä voidaan määrittää päivittyväksi säännöllisin väliajoin. Näin voitaisiin haluta toimittavan esimerkiksi kuukausikohtaisten raporttien tietoja ylläpitävien koosteiden kanssa.

4.3.3 Inkrementaalinen päivitys

Ennen päivityshetkeen vaikuttavien optioiden käsittelyä käsiteltiin materialisoidun näkymän luontilauseen REFRESH -lohkon optiot, jotka vaikuttivat materialisoidun näkymän päivitystapaan. Kyseisistä optioista REFRESH FAST määritteli materialisoidun näkymän päivittymään inkrementaalisesti. Edellytyksenä inkrementaaliselle päivittämiselle oli kuitenkin erityisen materialisoiduille näkymille suunnatun muutoslokin ylläpitäminen sekä tietyt rajoitukset koosteen luovassa SQL -kyselyssä.

Inkrementaalisen päivittämisen vaatiman lokin valintaan vaikuttaa lähtötiedot sisältävien taulujen päivitystapa. Mikäli näitä lähtötiedot sisältäviä fakta- tai koostetauluja päivitetään SQL -tiedonhallintakielen INSERT -, UPDATE - ja DELETE -lausein, voidaan muutoslokina käyttää erityistä materialisoiduille näkymille tarkoitet-

tua lokia. Jos lähtötiedot sisältäviä tauluja päivitetään esimerkiksi SQL*LOADER -tietojenlataustyövälineellä, myös materialisoitujen näkymien inkrementaalisen päivityksen tarvitsema loki muodostetaan työvälineen vaatimin keinoin (*direct loader log*). Tässä tutkielmassa ei käsitellä lataustyövälineen lokia. Sen sijaan seuraavaksi tarkastellaan tarkemmin ensin mainittua inkrementaalisen päivityksen vaatiman lokin toteutustapaa.

Materialisoiduille näkymille tarkoitetun lokin toimintaperiaate on pitkälti samanlainen kuin muillekin tietokantatauluille tietokannanhallintajärjestelmän ylläpitämän muutoslokin tapauksessa. Koosteen lähtötiedot sisältävissä tietokantatauluissa tapahtuneiden muutoksien tiedot tallennetaan erilliseen tiedostoon riittävän yksilöidysti. Lokin perusteella muutokset on määrättynä ajankohtana heijastettavissa koosteeseen muodostamalla tiedoston sisältämien tietojen perusteella tarvittavat SQL -tiedonhallintalauseet.

Kuvassa 13 on kuvattu materialisoidun näkymän käyttöön tarkoitetun lokin luontilause. Loki tulee luoda jokaiselle lähtötietoja sisältävälle tietokantataululle, josta on johdettu yksikin inkrementaalista päivitystä käyttävä kooste. Loki tulisi myös luoda ennen materialisoidun näkymän perustamista. Luotaessa loki materialisoidun näkymän perustamisen jälkeen, joudutaan materialisoidulle näkymälle kohdistamaan COMPLETE REFRESH ennen inkrementaalisen päivityksen käyttöönottoa. Lokin luomiseksi tulee ylläpitäjällä olla kyseiseen tietokantaskeemaan taulujen luontioikeus, triggereiden luontioikeus, taulujen kommentointioikeus sekä selailuoikeus kyseiseen koosteen lähtötiedot sisältävään tietokantatauluun.

```
CREATE MATERIALIZED VIEW LOG ON [<schema>.]<table>
     TABLESPACE <tablespace>
     [ [NO] LOGGING  ] [ [NO] CACHE  ] [ [NO] PARALLEL [<n>] ]
     [ partitioning stuff ]
     [ WITH
          {
               [ PRIMARY KEY ]
               [ , ROWID ]
               [ , SEQUENCE [(column [, column ...] ) ]
               [ , OBJECT ID ]
          }
          [  { INCLUDING NEW VALUES | EXCLUDING NEW VALUES } ]
     ];
```

Kuva 13: SQL -tiedonhallintakielen syntaksi materialisoiduille näkymille tarkoitetun lokin toteuttamiseksi (Powell, 2005).

Luontilauseen WITH -lohkoon sisältyy inkrementaalisen päivittämisen kannalta kolme tärkeää ominaisuutta. Ensinnäkin riippuen lähtötiedot sisältävän tietokantataulun sisältämien tietojen tyypistä, voidaan luontilauseen WITH -lohkossa vaikuttaa muutostietojen yksilöintitapaan. Yleisimmin muutoksien yksilöinti tapahtuu sitomalla muutos muutosta edustavan tietokantataulun rivin perusavaimeen tai tietokannanhallintajärjestelmän ylläpitämään rivin fyysiseen tunnukseen. Näitä vaihtoehtoja lokin luontilauseessa edustavat WITH -lohkon optiot PRIMARY KEY ja ROWID.

Ensisijaisena vaihtoehtona suositellaan perusavaimen käyttöä huomioiden kuitenkin myöhemmin käsiteltävät erilaisten koosteiden asettamat rajoitukset. Päivitettäessä lähtötietoja sisältäviin tauluihin suurempia tietomääriä kerralla, voidaan tietokantatauluihin joutua kohdistamaan DBMS_MVIEW -paketin sisältämiä tietokantataulujen fyysiseen uudelleenorganisointiin liittyviä toimenpiteitä sekä TRUNCATE -toiminnolla suoritettavia tiedon poistotoimenpiteitä. Edellä mainittujen toimenpiteiden yhteydessä tietokantataulujen rivien fyysiset tunnukset muuttuvat, joten ROWID -yksilöintiin perustuvassa koosteessa jouduttaisiin inkrementaalisen päivityksen sijaan suorittamaan tilanteen päivittämiseksi COMPLETE REFRESH.

Muutosten yksilöintiin liittyvää optiota OBJECT ID voidaan käyttää tilanteissa, joissa lähtötietoja ylläpitävän tietokantataulun tietueet muodostuvat käyttäjän TYPE -objektilla määrittämistä kokonaisuuksista. Yksilöinti voidaan tällöin sitoa tietokannanhallintajärjestelmän ylläpitämien kyseisten tietokantaobjektien tunnuksiin. Käsiteltyihin yksilöintivaihtoehtoihin liittyen on huomattavaa, että eri yksilöintivaihtoehtoja voidaan erilaisia koosteita varten yhdistää samassa lokissa. Toisaalta tämä muodostuu jopa välttämättömäksi, koska yhtä taulua kohden voi muodostaa vain yhden materialisoidulle näkymälle tarkoitetun lokin.

Toinen WITH -lohkon inkrementaalisen päivittämisen kannalta tärkeä osa on SEQUENCE -optio, jonka välityksellä tietokannanhallintajärjestelmälle tarkennetaan seurattavat sarakkeet.

Inkrementaaliseen päivitykseen liittyen kolmas WITH -lohkon olennainen asia on optio INCLUDING NEW VALUES. Tämän option avulla määritetään, että muutettuja

tietokantataulun rivejä koskien kirjoitetaan lokiin seurattavista sarakkeista sekä vanhat että muutetut arvot. Inkrementaalisen päivityksen kannalta tätä oletusoptiota on käytettävä vaihtoehtoisen EXCLUDING NEW VALUES sijaan.

Tähän mennessä on inkrementaaliseen päivitykseen liittyen käsitelty vaatimusta muodostaa erillinen loki muutoksien seuraamiseksi. Koosteiden muodostamisessa käytettyyn SQL -kyselykieleen kohdistuvia rajoitteita ja vaatimuksia on koottu liitteeseen 2 erilaisten koosteiden näkökulmasta. SQL -kyselyihin liittyvät vaatimukset muodostuvat pitkälti tietokannanhallintajärjestelmän laskennassa apuna tarvitsemista kentistä.

Inkrementaalisen päivityksen asettamat vaatimukset saattavat tuntua monimutkaisilta. Tarkoitus ei kuitenkaan ole tarkastaa vaatimuksia manuaaleista ja toimia muistin varassa. Oracle 10g -tietokannanhallintajärjestelmä sisältää sen sijaan työkaluja vaatimuksien toteutumisen analysoimiseksi.

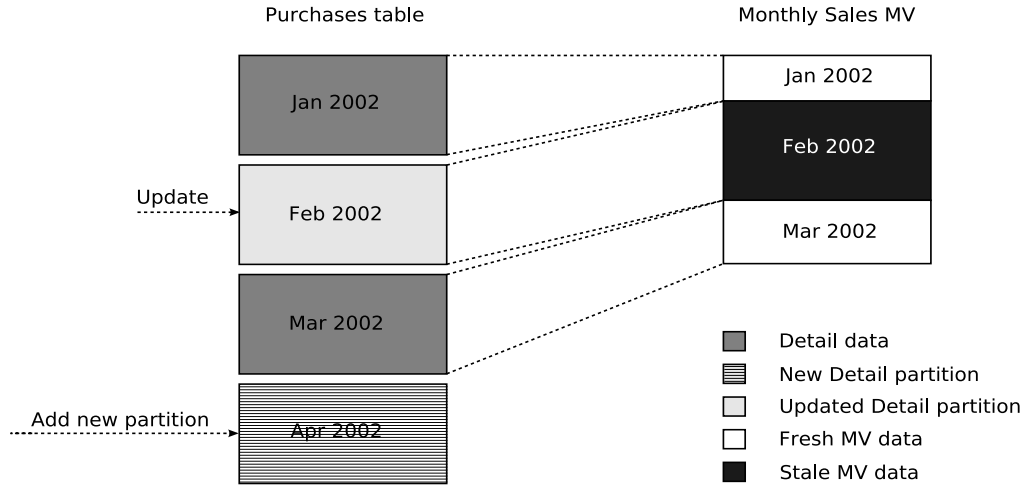
EXPLAIN_MVIEW -kantapaketti toimii työkaluna hahmotettaessa olemassa olevien materialisoitujen näkymien ominaisuuksia. EXPLAIN_MVIEW -kantapaketilla voi muodostaa esimerkiksi yhteenvetoja, joista voi tarkastaa tukeeko materialisoitu näkymä tällä hetkellä inkrementaalista päivittämistä tai kyselyjen uudelleenkirjoitusta. Työkalun kautta on tarvittaessa myös selvitettävissä syyt, miksi materialisoitu näkymä ei tällä hetkellä mainittuihin ominaisuuksiin yllä. TUNE_MVIEW -kantapaketti toimii työkaluna hahmotettaessa tarvittavia toimenpiteitä haluttujen ominaisuuksien saavuttamiseksi.

Liitteessä 2 käsiteltyjen vaatimuksien joukossa viitattiin rajoituksien vastaantullessa mahdollisuuteen toteuttaa inkrementaalinen päivittäminen koosteesta riippuen Partition Change Tracking -päivitystä käyttäen. Seuraavaksi tarkastellaan tarkemmin tätä päivityskeinoa.

4.3.4 PCT -päivitys

Tähän mennessä on käsitelty materialisoitujen näkymien sisältämien tietojen päivitysmenetelmiä perustuen vaihtoehtoihin, joihin on voinut ottaa kantaa jo materialisoidun näkymän luontilauseen yhteydessä. Oracle 10g - tietokannanhallintajärjestelmä sisältää myös päivityskeinon, joka tunnetaan nimellä Partition Change Tracking (PCT). Johtuen päivityskeinon ominaisuudesta ylläpitää muutoslokia ja heijastaa muutoksia materialisoituihin näkymiin muutoslokin tietoihin perustuen kohdennetusti, lukevat Hobbs *& al.* (2005) PCT -päivityksen yhdeksi inkrementaalisen päivittämisen keinoista.

PCT -päivitys liittyy tietovarastojärjestelmässä tyypillisesti vastaantulevaan tilanteeseen, jossa koosteita muodostetaan partitioitujen tietokantataulujen perusteella. Kuvassa 14 on havainnollistettu PCT -päivityksen toimintaideaa kuvaten kuukausitason myyntiyhteenvetotietoja ylläpitävän Monthly Sales -materialisoidun näkymän ja myynneistä atomitason tietoja ylläpitävän partitioidun Purchases -taulun suhdetta.



Kuva 14: Partition Change Tracking (Hobbs *& al.*, 2005).

PCT -päivitys perustuu ajatukseen, että tietokannanhallintajärjestelmä ylläpitää tietoja koosteiden tietolähteenä toimivien taulujen eri partitioissa tapahtuneista muutoksista. Vaikka materialisoitu näkymä itsessään ei ole partitioitu, pystyy tietokannanhallintajärjestelmä muutoksia kohdentaessaan käsittelemään materialisoitua näkymää ikään kuin se olisi partitoitu. Toisin sanoen tietokannanhallintajärjestelmä tunnistaa mate-

rialisoidusta näkymästä lähdetaulun partitioita vastaavia osioita ja päivitystapahtuman yhteydessä tietokannanhallintajärjestelmä osaa kohdentaa käsittelyn vain tarvittaviin materialisoidun näkymän osioihin.

Kuten aikaisemmin tässä kohdassa tuli ilmi voi PCT -päivitys joskus toimia ratkaisuna inkrementaalisen päivittämisen toteuttamiseksi tilanteissa, joissa materialisoiduille näkymille tarkoitettujen lokien toteuttamisen asettamat rajat on ylitettävä. Liitteessä 2 mainittujen tilanteiden ohella mainitsevat Hobss *& al.* (2005) PCT -päivityksen olevan joskus myös tehokkaampi vaihtoehto inkrementaalisen päivittämisen toteuttamiseksi materialisoiduille näkymille tarkoitettujen lokien sijaan, jos partitoituun lähdetiedot sisältävään tauluun on tapahtunut paljon muutoksia.

Kuten materialisoiduille näkymille tarkoitettujen inkrementaaliseen päivittämiseen tarvittavien muutoslokien tapauksessa, liittyy PCT -päivityksen mahdollistamiseen omat ehtonsa koskien sekä lähdetietoja ylläpitäviä tauluja ja muodostettavia materialisoitujan näkymiä. Partitioitujen lähdetaulujen osalta säännöt ovat yksinkertaiset ja rajautuvat kahteen seuraavaan sääntöön:

- Partitioitu lähdetaulu voi olla partitioitu käyttäen Range -, List -, Range-Hash - tai Range-List composite -partitiointimenetelmiä.
- Lähdetaulun partitiointiavaimen täytyy koostua vain yhdestä sarakkeesta.

Materialisoidulla näkymällä muodostettavan koosteen tulee sen sijaan olla toteutettu tukien yhtä seuraavista tekniikoista, jotka käydään seuraavaksi tarkemmin läpi:

- liitosriippuvuuden esiintyminen (*Join dependency expression*)
- partitiointiavaimen esiintyminen (*Partition key*)
- partitiointimerkitsijän käyttäminen (*Partition marker*)

Täyttääkseen liitosriippuvuuden materialisoidun näkymän muodostavan SQL -kyselyn täytyy sisältää tietokantataulujen liitos, jossa on mukana lähdetaulun partitioinnin perustana oleva partitiointiavain. Edellä mainitun liitoksen lisäksi koosteen muodostavan

SQL -kyselyn SELECT -lohkossa täytyy olla edustettuna sarake taulusta, jonka avaimeen lähdetaulun partititointiavain kuuluu. Liitosriippuvuuden toteutumista on havainnollistettu kuvissa 15 ja 16, joissa koosteita on muodostettu kuvassa 14 esitettyyn atomitasoiset myyntitiedot sisältävään Purchases -tauluun perustuen.

```
CREATE MATERIALIZED VIEW monthly_sales_mv
AS
SELECT t.month, t.year, p.product _id,
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales, COUNT (*)
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
      ps.product_id = p.product_id
GROUP BY t.month, t.year, p.product_id;
```

Kuva 15: Liitosriippuvuuden toteutuminen (Hobbs *& al.*, 2005).

Kuvan 15 mukaisessa koosteessa toteutuu liitosriippuvuuden esiintyminen perustuen siihen, että lähtötiedot sisältävä Purchases -taulu on partitoitu Time -taulun time_key -sarakkeen perusteella. Koosteen SQL -kysely sisältää liitoksen, jossa partitointiavain time_key on mukana. SQL -kyselyn SELECT -lohkossa täytetään sen sijaan toinen liitosriippuvuuden ehto sisällyttämällä koosteeseen Time -taulun sarakkeita.

```
CREATE MATERIALIZED VIEW regional_sales_mv
AS
SELECT c.region, SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales
FROM customer c, purchases ps
WHERE c.customer_id = ps.customer_id
GROUP BY c.region;
```

Kuva 16: Liitosriippuvuus ei toteudu (Hobbs *& al.*, 2005).

Sen sijaan kuvassa 16 havainnollistettu materialisoitu näkymä ei tulisi tukemaan PCT -päivitystä. Materialisoidun näkymän luova SQL -kysely ei sisällä liitosta partitiointiavaimeen kiinteästi liittyvään Time -tauluun, joten ehto liitosriippuvuuden esiintymisestä ei toteudu. Vaihtoehdoksi PCT -päivityksen mahdollistamiseksi kyseiseen koosteen kanssa jää täyttää joko ehto partitiointiavaimen esiintymiseen tai partitiointimerkitsijän käyttämiseen liittyen.

Partitiointiavaimen esiintymisen ehdon täyttäminen on kyseisen koosteen tapauksessa yksinkertaista. PCT -päivityksen mahdollistamiseksi riittää koosteen muodostavaan

SQL -kyselyyn lisätä mukaan Purchases -taulun partitiointiavaimen esiintyminen kuvan 17 osoittamalla tavalla. Jos koosteen muodostava SQL -kysely ei sisältäisi GROUP BY -lohkoa, riittäisi partitiointiavaimen lisääminen kelvollisen SQL -kyselyn muodostamiseksi vain SELECT -lohkoon.

```
CREATE MATERIALIZED VIEW regional_sales_partkey_mv
AS
SELECT c.region, ps.time_key,
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales
FROM customer c, purchases ps
WHERE c.customer_id = ps.customer_id
GROUP BY c.region, ps.time_key;
```

Kuva 17: Partitiointiavaimen esiintyminen (Hobbs *et al.*, 2005).

Valitsemalla partitiointiavaimen esiintymiseen liittyvän ehdon toteuttaminen PCT -päivityksen mahdollistamiseksi tulee huomioida tekniikan vaikutus koosteen rivimäärään. Kuvan 16 kuvaama kooste sisältäisi jokaista aluetta kohden vain yhden rivin. Lisäämällä koosteen muodostavaan SQL -kyselyyn mukaan partitiointiavain rivejä muodostuisi yhdelle alueelle useita partitiointiavaimen mukaisesti. Vaikka lähdetaulu olisi partitioitu kuvan 14 osoittamalla tavalla kuukausikohtaisiin partitioihin, muodostuisi koosteeseen yhdelle alueelle rivejä partition sisältämien erilaisten partitiointiavaimien lukumäärän mukaisesti. Esimerkiksi jos partitiointiavain noudattaisi päiväkohtaista tarkkuutta, voisi koosteeseen muodostua yhtä aluetta kohden esimerkiksi 30 eri yhteenvetoriviä kullekin kuukaudelle. Vuositasolla tämä tarkoittaisi vastaavasti satoja rivejä.

Toinen vaihtoehto korvata koosteen muodostavassa SQL -kyselyssä oleva liitosriippuvuuden esiintymisen puute on käyttää partitiointimerkitsijää. Partitiointimerkitsijä lisätään koosteen muodostavaan SQL -kyselyyn käyttämällä tietokantapaketin DBMS_MVIEW proseduuria PMARKER kuvan 18 havainnollistamlla tavalla. Huomattavaa partitiointimerkitsijän käytössä on, että partitiointiavaimen sijaan PMARKER -proseduuri käyttää erottelutekijänä lähdetiedot sisältävän taulun rowid -tunnuksia.

Rowid -tunnuksen sitomisesta erottelevaksi tekijäksi voisi helposti vetää johtopäätök-

sen, että koosteeseen muodostuisi jälleen enemmän rivejä yhtä aluetta kohden. Verrattaessa partitiointimerkitsijän käyttöä edellä käsiteltyyn partitiointiavaimen käyttöön PCT -päivityksen mahdollistajana, tilanne on kuitenkin päinvastainen. Partitiointimerkitsijää käytettäessä kombinaatiot rajoittuvat partitioiden lukumäärään. Jos partitiointiavaimen perusteella tiedot olisivat lähdetaulussa jaettu kuukausikohtaisiin partitioihin ja tietovarastojärjestelmä sisältäisi toistaiseksi vain yhden vuoden tiedot, muodostuisi kutakin aluetta kohden korkeintaan 12 riviä.

```
CREATE MATERIALIZED VIEW regional_sales_marker_mv
REFRESH FORCE
AS
SELECT c.region, DBMS_MVIEW.PMARKER(ps.rowid) as pmark,
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales
FROM customer c, purchases ps
WHERE c.customer_id = ps.customer_id
GROUP BY c.region, DBMS_MVIEW.PMARKER(ps.rowid);
```

Kuva 18: Partitiointimerkitsijän käyttäminen (Hobbs *et al.*, 2005).

Edellä on käsitelty lyhyesti materialisoidun näkymän PCT -päivityksen mahdollistamiseksi olevat tekniikat. Käytännön soveltamisessa voi tulla vastaan tilanne, että materialisoitu näkymä muodostetaan useamman partitioidun tietokantataulun perusteella. Tätä tilannetta varten on hyvä huomioida, että eri tekniikoita voidaan materialisoidun näkymän luontilauseessa yhdistää eri lähdetaulujen tarpeet huomioiden.

PCT -päivityksen hyöty näkyy usein inkrementaalisen päivityksen yhteydessä satunnaisesti tarvittavien COMPLETE REFRESH -päivitystarpeiden vähenemisenä. Toisaalta Oracle 10g -tietokannanhallintajärjestelmä sisältää erityisiä vasteaikoja parantavia optimointipiirteitä liittyen partitiointiavaimen ja liitosriippuvuuden avulla varmistettuihin PCT -päivityksiin.

Kuten aikaisemmin mainittiin, ei Oracle 10g -tietokannanhallintajärjestelmä mahdollista PCT -päivityksen määrittämistä koosteelle materialisoidun näkymän luontilauseen yhteydessä. PCT -päivityksen kohdistamiseksi materialisoituun näkymään tulee päivitys käynnistää käyttäen aiemmin mainittua DBMS_MVIEW.REFRESH proseduuria PCT -päivityksen valintaan liittyvällä parametrisoinnilla:

```
EXECUTE DBMS_MVIEW.REFRESH('monthly_sales_mv', 'P')
```

Tässä luvussa on tähän mennessä käsitelty materialisoitujen näkymien käsittelyä koosteiden päivitysmenetelmien kannalta. Seuraavaksi tarkastellaan materialisoitujen näkymien käsittelyä kyselyjen uudelleenkirjoittamisen näkökulmasta.

4.4 Materialisoidut näkymät ja kyselyjen uudelleenkirjoitus

Edellisessä luvussa käsiteltiin koosteiden roolia tietovarastojärjestelmään kohdistuvien tiedonhakujen vasteaikojen optimoijana sekä koosteiden näkymättömyyttä loppukäyttäjän kannalta. Tavoitetilanteessa loppukäyttäjälle näkyisi raportoinnissa ja analysoinnissa käytettävistä tiedoista mahdollisimman yksikäsitteinen näkymä ja eri tekniikoihin perustuvat koosteiden käsittelijät huolehtisivat atomitasoisiin tietoihin kohdennettujen tietokantakyselyjen uudelleenkirjoittamisesta käyttämään tilanteeseen sopivia koosteita.

Oracle 10g -tietokannanhallintajärjestelmän koosteiden hallinnan toteuttamiseksi tarkoitettu materialisoituihin näkymiin perustuva tekniikka tukee kyselyjen uudelleenkirjoitusta. Kuten inkrementaaliseen päivittämiseen liittyen, kuuluu kyselyjen uudelleenkirjoittamiseen joukko asetuksia ja rajoituksia, joihin perustuen tietokannanhallintajärjestelmän sisältämä optimoija (*optimizer*) tekee päätöksensä tiedonhakuja tulkitessaan. Kyselyjen uudelleenkirjoittamisen käsittely aloitetaan käsittelemällä neljää asetuksille asetettua vaatimusta.

4.4.1 Kyselyjen uudelleenkirjoituksen mahdollistaminen

Edellä on käsitelty huomattava joukko kuvassa 8 esitettyjä materialisoidun näkymän luontilauseessa esiintyviä optioita. Tähän mennessä käsittelemättömien optioiden joukosta löytyy myös kyselyjen uudelleenkirjoitukseen liittyvä optio DISABLE_QUERY_REWRITE/ENABLE_QUERY_REWRITE. Oletuksena luotu materialisoitu näkymä ei ole kyselyjen uudelleenkirjoituksen käytettävissä. Jos materiali-

soitua näkymää halutaan hyödyntää kyselyjen uudelleenkirjoituksessa, tulee se varustaa optiolla ENABLE_QUERY_REWRITE.

Luontilauseen option lisäksi kyselyjen uudelleenkirjoittamiseen liittyy kolme systeemiparametria (Hobbs *& al.*, 2005; Oracle, 2005). Yksi systeemiparametreista on parametri QUERY_REWRITE_ENABLED, jolle voidaan määrittää arvot FALSE, TRUE tai FORCE seuraavasti:

```
ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE;
```

Käytettäessä arvoa FALSE optimoija ei kyselyjen kääntämisen yhteydessä analysoi lainkaan kyselyn uudelleenkirjoittamisen kannattavuutta, kuten arvon TRUE yhteydessä. Arvolla FORCE uudelleenkirjoittamisen hyötyjen analysoinnista sen sijaan luovutaan ohjaamalla optimoija aina valitsemaan uudelleenkirjoitus, kun se on tiedonhakuun liittyen mahdollista. Arvoa FORCE suositellaan käytettäväksi vain, jos tietokannan ylläpitäjät ovat varmoja kyselyjen hyötyvän koosteiden käyttämisestä ja kyselyjen kääntämiseen kuluvaa aikaa halutaan vähentää. QUERY_REWRITE_ENABLED systeemiparametrin oletusarvo riippuu systeemiparametrin OPTIMIZER_FEATURES_ENABLES asetuksista. Jos edellä mainitun systeemiparametrin arvo on 10.0.0 tai suurempi, oletusarvo QUERY_REWRITE_ENABLED parametrille on TRUE, muutoin FALSE.

Toinen kyselyjen uudelleenkirjoittamiseen liittyvistä systeemiparametreista on parametri QUERY_REWRITE_INTEGRITY, jonka avulla säädellään kyselyjen uudelleenkirjoituksessa käytettäviä sääntöjä ja arvioinnissa käytettävän metadatan laajuutta. Tämän parametri on avainasemassa, kun tavoitteena on vaikuttaa aikaisemmin käsiteltyyn lähdetaulujen ja koostetaulujen tietojen eheyssäännön toteutumiseen koosteen käyttötarpeiden näkökulmasta.

Parametrille QUERY_REWRITE_INTEGRITY voi määrittää arvot ENFORCED, TRUSTED tai STALE_TOLERATED, joista arvo ENFORCED toimii oletusarvona. Arvolla ENFORCED voidaan toteuttaa mainittu lähdetaulujen ja koos-

teen tietojen eheysvaatimus tiukimmassa mahdollisessa muodossaan. Oracle 10g -tietokannanhallintajärjestelmä pitää huolen siitä, että huolimatta kyselyjen kohdistamisesta lähdetauluihin tai koostetauluihin tulos on aina sama. Eheyssäännön toteutumisen varmistamiseksi tietokannanhallintajärjestelmä joutuu arvoa ENFORCED käytettäessä validoimaan kaikki koosteiden ja lähtötaulujen väliset liitokset, mikä tietovarastojärjestelmissä voi kuluttaa hyötyihin nähden kohtuuttomia aikakustannuksia. Tiukan eheyslinjan varmistamiseksi ENFORCED arvoon liittyy myös tiukimmat käyttötilanteisiin liittyvät rajoitteet. Kyseisen parametrin käyttö ei tue dimensio -objektien käyttöä osana koostetta, mikä on huomattava rajoittava tekijä tähtimallinnuksella toteutettujen tietovarastojärjestelmien koosteiden hallinnassa. Rajoitteen kiertäminen voisi johtaa useiden turhien esiliitettyjen koosteiden toteuttamiseen ylläpitokustannuksia tarpeettomasti kasvattaen. Toinen rajoittava tekijä liittyy tilanteisiin, joissa olemassa olevasta tietokantataulusta muodostetaan materialisoitu näkymä PREBUILT TABLE -optiota materialisoidun näkymän luontilauseessa käyttäen. Näin luotu taulu ei ole käytettävissä uudelleenkirjoitukseen, ellei siihen kohdisteta materialisoidun näkymän kokonaan uudelleen luovaa päivitystoimintoa.

Lähdetaulujen ja koostetaulujen välinen eheyssääntö todettiin kohdassa 4.3.1 olevan loppujen lopuksi riippuvainen loppukäyttäjän asettamista toiminnallisista määrityksistä. Esimerkkinä mainittiin tietyltä aikaväliltä tietoja ylläpitävät koosteet, joiden päivitystiheys voi olla huomattavasti lähdetaulujen päivitystiheyttä harvempi. Näitä käyttötarkoituksia varten QUERY_REWRITE_INTEGRITY -systeemiparametrille on annettava arvo STALE_TOLERATED. Arvon nimi kuvaa tämän arvon toimintatapaa. Optimoijan sallitaan käyttävän kyselyjen uudelleenkirjoituksessa sellaisia koosteita, jotka eivät koosteen lähdetaulujen tietoihin nähden ole ajantasalla.

Kahden edellä mainitun systeemiparametriin QUERY_REWRITE_INTEGRITY liittyvän arvon välimuotoa edustaa arvo TRUSTED, joka johtaa optimoijan tulkitsemaan koosteiden ja lähdetaulujen välisiä liitoksia validoimatta niitä. TRUSTED -arvon käyttö edellyttää tarvittavien taulujen välisiin suhteiden määrittelyä RELY -optiolla. Arvoa TRUSTED käytettäessä tukee optimoija myös dimensio -rakenteita sekä PREBUILT TABLE -optiolla muodostettuja materialisoituja näkymiä. TRUSTED arvo jättää kui-

tenkin enemmän vastuuta tietojen oikeellisuudesta tietokannan ylläpitäjän varaan. Ennen arvon käyttöönottoa tulisikin ylläpitäjän varmistua muiden keinojan avulla siitä, että taulujen väliset viite-eheydet ovat luotettavia ja PREBUILT TABLE -optiolla muodostetun materialisoidun näkymän sisältö todella vastaa luontilauseessa esitetyn SQL -kyselyn tulosta.

Tietovarastojärjestelmissä usein käytettyjä QUERY_REWRITE_INTEGRITY -systeemiparametrin arvoja ovat TRUSTED ja STALE_TOLERATED. Jälkimmäistä arvoista vihjataan käyttämään myös testatessa kyselyjen uudelleenkirjoitettavuutta perustettujen materialisoitujen näkymien osalta. Jos kysely ei uudelleenkirjoitu STALE_TOLERATED -arvoa käytettäessä, ei se tule uudelleenkirjoittumaan kahta muutakaan mainittua arvoa käytettäessä.

Kolmas kyselyjen uudelleenkirjoittamiseen vaikuttavista systeemiparametreistä on Oraclen (2005) esittelemä parametri OPTIMIZER_MODE. Kyselyjen uudelleenkirjoittamisen mahdollistamiseksi parametrille tulisi asettaa jokin arvoista ALL_ROWS, FIRST_ROWS tai FIRST_ROWS_n. Arvossa FIRST_ROWS_n kirjaimella n kuvataan tulosjoukosta palautettavien rivien lukumäärää, joka voi olla 1, 10, 100 tai 1000.

OPTIMIZER_MODE -systeemiparametrilla vaikutetaan optimoijan uudelleenkirjoituksessa käyttämiin päättelysääntöihin. Arvolla FIRST_ROWS optimoija arvioi valintaansa sekä kyselyjen kustannuksia laskemalla että soveltamalla tunnettuja heuristiikkoja sopivan suorituspolun saavuttamiseksi. Arvolla FIRST_ROWS_n optimoija tukeutuu vain kustannuslaskentaan arvioiden suoritusta myös palautettavien rivien lukumäärän perusteella.

4.4.2 Kyselyjen uudelleenkirjoittumisen tarkkailu

Uudelleenkirjoitus perustuu optimoijan tiedonhakujen yhteydessä suorittamiin analyyseihin materialisoitujen näkymien sopivuudesta tiedonhakuun tunnettuihin heuristiikkoihin perustuen sekä kustannusarvioihin uudelleenkirjoittamisella saavutetuista vasteaikahyödyistä. Ohjelmistoille tyypilliseen tapaan optimoijan uudelleenkirjoittamisen

yhteydessä tekemä analysointi perustuu ennaltamäärättyyn päättelylogiikkaan, jonka toteutuksen laajuus asettaa rajat tuettavien tiedonhakujen ominaisuuksille.

Uudelleenkirjoituksen logiikka perustuu merkkijonovertailuun. Optimoija vertaa suoritettavan tiedonhaun sisältämää SQL -kyselyä materialisoitujen näkymän perustana olevaan SQL -kyselyyn merkkijonovertailuja käyttäen. Jos SQL -kyselyt eivät vastaa toisiaan, lieventää optimoija vaatimuksia täydellisestä vastaavuudesta esimerkiksi jättämällä SELECT -lohkon vertailun ulkopuolelle. Sarakkeiden sijaan optimoija voi keskittyä tarkastelemaan tietokantataulujen välisten liitoksien yhdenmukaisuutta tai GROUP BY -lohkon sisältöä. Merkkijonovertailun ohella optimoija käyttää analysoinnissa hyödykseen lähdetauluista ja koosteista löytyviä metatietoja sekä tilastotietoja, minkä vuoksi ylläpitäjän olisi tärkeää huolehtia tilastotietojen säännöllisestä päivittämisestä DBMS_STATS -tietokantapaketin työkaluja käyttäen. (Hobbs *& al.*, 2005; Oracle, 2005)

Hobbs *& al.* (2005) esittelevät muutamia tyypillisiä uudelleenkirjoituksen tukemia tilanteita. Suoraviivaisinta uudelleenkirjoitus optimoijalle on, jos tiedonhaun ja koosteen muodostavan luontilauseen SQL -kyselyt vastaavat toisiaan (*SQL Text Match*). Edellä mainittua vastaavuutta ei kuitenkaan löydy sarakkeiden ollessa eri järjestyksessä tai SQL -kyselyssä käytettyjen sarakkeiden poiketessa toisistaan.

Tulkitessaan uudelleenkirjoituksen mahdollisuutta SQL -kyselyn FROM -, WHERE - ja GROUP BY -lohkojen perusteella, optimoija löytää muunlaisia mahdollisuuksia. Esimerkiksi koosteen sisältäessä yhteenlaskettuja tietoja kuukausitasolla ja tiedonhaun hakiessa tietoja vuositasolla, kykenee optimoija ilman dimensioon kytkettyjä hierarkiamäärityksiäkin laskemaan tiedot materialisoitua näkymää käyttäen (*Aggregate Rollup*). Edellytyksenä kuitenkin olisi, että kuukausitason tietoja ylläpitävillä riveillä olisi myös vuosi-tietoa kuvaava sarake käytössä. Toisaalta tarvittavat asetukset ja rajoitukset huomioiden pystyy optimoija selviämään hierarkiatasot ylittävistä uudelleenkirjoitustarpeista nimenomaan dimensioiden hierarkiamäärityksiin perustuen (*Rewrite using dimensions and constraints*).

Tiedonhauissa tulee useasti kuitenkin vastaan tilanne, että sarakkeiden lukumäärä poik-

keaa optimoijan analysoimissa SQL -kyselyissä toisistaan. Optimoija voi hyödyntää uudelleenkirjoitusta joskus näissäkin tilanteissa. Esimerkiksi jos tiedonhaku sisältäisi yhden sarakkeen enemmän johtuen tarpeesta tuoda dimensio-taulun attribuutti koosteen ulkopuolelta, voi optimoija kyetä hyödyntämään koostetta tiedonhaussa yhdistämällä attribuutin mukaan saamiseksi koosteen ja dimensio -taulun (*Join-back*). Koosteen pienentäessä tietokantataulujen väliseen liitokseen osallistuvia rivejä, on tulos yleensä tämänkin uudelleenkirjoituksen seurauksena vasteajaltaan parempi kuin lähdettäessä attribuuttia tavoittelemaan lähdetaulujen ja dimensio -taulujen välisellä tiedonhaulla.

Oracle 10g -tietokannanhallintajärjestelmä sisältää muitakin käytännön yhteydessä vastaantuleviin tilanteisiin ratkaisun muodostavia tekniikoita. Näistä tunnettuna ja erittäin suositeltunakin tekniikkana mainitaan mahdollisuus määrittää optimoijalle erityisiä vihjeitä (*hints*). Vihjeet perustuvat mahdollisuuteen liittää SQL -kyselyn SELECT -lohkoon yksi määreistä REWRITE(<materialisoidun näkymän nimi>), NO_REWRITE tai REWRITE_OR_ERROR. Kyseisten määreiden avulla voidaan optimoijaa pyytää ohjaamaa kysely tiettyä materialisoitua näkymää käyttäväksi, pyytää optimoijaa olla lainkaan yrittämättä uudelleenkirjoitusta kyseisen kyselyn kanssa tai pyytää optimoijaa päättämään käsittely virhetilanteeseen uudelleenkirjoituksen ollessa mahdotonta. (Hobbs *& al.*, 2005; Oracle, 2005)

Aikaisemmin mainittiin, että myös kyselyjen uudelleenkirjoittamiseen liittyy asetuksien ohella muitakin rajoitteita. Rajoitteisiin kuuluu joukko SQL -kyselyjen sisältöön liittyviä rajoitteita ja vaatimuksia, joista osa on samankaltaisia kuin liitteessä 2 mainitut inkrementaalista päivitystä koskevat vaatimukset. Tutkielmassa ei lähdetä näitä vaatimuksia yksityiskohtaisesti läpikäymään, sillä sääntöjen täsmällisen hallitsemisen sijaan materialisoitujen näkymiä käytettäessä on tarkoituksenmukaisempaa oppia tuntemaan ja hyödyntämään käyttötilanteiden analysoimiseksi tarkoitettuja tietokannanhallintajärjestelmän tarjoamia työkaluja. Monet vaatimuksista muu muassa koskevat optimoijan laskennassa apuna tarvitsemien sarakkeiden mukaanottoa SQL -kyselyjen SELECT -lohkoon. Näillä vaatimuksilla ei ole tarkoitus rasittaa ylläpitäjän muistikapasiteettiä, vaan sen sijaan ylläpitäjä voi eri työkaluilla selvittää optimoijan kyselyyn

kohdistamat vaateet.

Kyselyjen uudelleenkirjoittamisen seurannan hallinnoimiseksi tarjoaa Oracle 10g -tietokannanhallintajärjestelmä useita työkaluja. Yksi työkaluista on EXPLAIN PLAN -työkalu, jolla voidaan seurata SQL -kyselyn käsittelyn kulkua tietokannanhallintajärjestelmän kyselyä suorittaessa. Tämän työkalun avulla koosteiden ylläpitäjä voi etsiä vastausta kysymykseen, osaako optimoija tietynlaisen kyselyn tapauksessa hyödyntää muodostettuja materialisoituja näkymiä. Mikäli työkalun palauttamassa suorituksen etenemisen kuvauksessa esiintyy maininta MAT_VIEW REWRITE ACCESS FULL, on optimoija ohjannut kyselyn materialisoitua näkymää käyttämään.

Selvitettäessä uudelleenkirjoituksen tulokseen vaikuttavia seikkoja, muodostuvat kaksi keskeisintä työkalua DBMS_MVIEW -tietokantapaketin proseduureista EXPLAIN_MVIEW ja EXPLAIN_REWRITE. Kuten aikaisemminkin mainittiin EXPLAIN_MVIEW -proseduuri palauttaa näkymiin liittyen yhteenvetotietoja esimerkiksi materialisoitujen näkymien sen hetkisistä mahdollisuuksista tukea inkrementaalista päivitystä tai kyselyjen uudelleenkirjoitusta. EXPLAIN_REWRITE -proseduurilla sen sijaan voidaan etsiä tarkempia syitä, miksi materialisoitu näkymä ei esimerkiksi kyseisenä hetkenä uudelleenkirjoitusta tue.

4.5 Materialisoitujen näkymien tiedonkäsittelyn optimointi

Luvussa 2 mainittiin koosteiden olevan jo sinällään tievarastojärjestelmissä käytetty vasteaikojen parantamiseen tähtäävä optimointikeino. Koska koosteet ovat muiden tietokantataulujen kaltaisia kokonaisuuksia, voidaan niihin kohdentaa myös vastaavia optimointikeinoja kuin muihinkin tietokantatauluihin. Tutkielman kohdistuessa Oracle 10g -tietokannanhallintajärjestelmän koosteiden hallintaan, tarkastellaan tässä kohdassa vain lyhyesti materialisoitujen näkymien indeksointia ja partitiointia.

4.5.1 Materialisoitujen näkymien indeksointi

Materialisoitujen näkymien indeksoinnin osalta tulee tilanne analysoida samalla tavalla kuin päätettäessä koosteen päivittämiskäytäntöä. Jokaista erillistä muutostietoa päivittäessään tietokannanhallintajärjestelmä päivittää myös indeksitietoja. Jos näkymään heijastettavia muutostietoja on paljon tai tauluun liittyy paljon eri indeksejä, muodostuu jossain vaiheessa tehokkaammaksi ottaa ennen päivitystä indeksointi pois käytöstä ja muodostaa se kokonaan uudelleen päivityksen jälkeen indeksien päivittämiseksi tarvittavien laskentojen minimoimiseksi. (Kimball *& al.*, 2008)

Materialisoiduille näkymille on käytössä vastaavat indeksointivaihtoehdot kuin muillekin tietokantatauluille. Tietovarastojärjestelmissä käytetään usein kuitenkin operatiivisissa järjestelmissä usein käytetyn *B-puuindeksin* lisäksi harvemmin käytettyjä indeksityyppejä kuten *bittikarttaindeksejä*, *klusteri-indeksejä* tai *IOT-indeksejä (index organized table)*. Tässä yhteydessä ei syvennytä tarkastelemaan edellä mainittuja indeksityyppejä tarkemmin.

Johtuen materialisoitujen näkymien tyypillisestä tehtävästä muodostaa yhteenvetotietoja sisältäviä koosteita, muodostetaan usein B-puuindeksejä yhdistelytekijöinä toimivien sarakkeiden perusteella. Edellä mainitun käytännön mukaisesti muodostaa Oracle 10g-tietokannanhallintajärjestelmä oletusarvoisesti indeksin materialisoidulle näkymälle sen perustamisen yhteydessä. Jos näin ei haluta käyvän, tulee luontilauseessa käyttää kuvassa 8 esitettyä optiota NO INDEX.

Yksi tärkeä materialisoitujen näkymien indeksointiin vaikuttava seikka on koosteen toimiminen tähtimallisessa koostekaaviossa fakta-tauluna. Jotta Oracle 10g-tietokannanhallintajärjestelmä pystyisi hyödyntämään optimointikeinojaan tähtimalliseen kaavioon kohdistuviin kyselyihin, tulee faktataulun viiteavaimien olla toteutettu bittikarttaindeksiä käyttäen. Koska Oracle 10g -tietokannanhallintajärjestelmä ei automaattisesti indeksejä viiteavaimille perusta, jää edellinen tietokannan ylläpitäjän varaan.

Huomioiden edellä olevat ohjeistukset koosteiden indeksointiin liityen, voitaisiin ku-

vassa 9 esitetylle koostetulle faktataululle muodostaa indeksit kuvan 19 osoittamalla tavalla. Tämän tutkielman viimeisessä luvussa käydään lyhyesti läpi työkaluja materialisoitujen näkymien toteuttamistarpeen kartoittamiseksi. Käsiteltävistä työvälineistä SQL Access Advisor auttaa tietokannan ylläpitäjiä myös indeksoinnin suunnittelussa.

```
CREATE INDEX ORDER_FACTS_AGGREGATE_GROUP_INDEX
ON ORDER_FACTS_AGGREGATE_MV (product_key, salesperson_key, day_key)
PCTFREE 5
TABLESPACE INDX
STORAGE (initial 64k next 64k pctincrease 0) ;

CREATE BITMAP INDEX ORDER_FACTS_AGGREGATE_PRODUCT_INDEX
ON ORDER_FACTS_AGGREGATE_MV (product_key);

CREATE BITMAP INDEX ORDER_FACTS_AGGREGATE_SALESPERSON_INDEX
ON ORDER_FACTS_AGGREGATE_MV (salesperson_key);

CREATE BITMAP INDEX ORDER_FACTS_AGGREGATE_DAY_INDEX
ON ORDER_FACTS_AGGREGATE_MV (day_key);
```

Kuva 19: Esimerkki kuvassa 9 esitetyn koostetun faktataulun indeksointimahdollisuudesta.

4.5.2 Materialisoitujen näkymien partitiointi

Indeksoinnin lisäksi tietovarastojärjestelmissä suorituksen optimointiin yleisesti Hobbs *& al.* (2005) mukaan käytetty keino on partitiointi. Materialisoitujen näkymien partitiointitapa valitaan usein mukailemaan lähdetiedot ylläpitävien faktataulujen partitioinnissa käytettyä tapaa. Edelliseen muodostaa poikkeuksen aikaisemmin kohdassa 4.3.4 käsitelty PCT -päivitys, jonka avulla materialisoitua näkymää itsessään ei tarvitse partitioida tavoiteltaessa materialisoidun näkymän käsittelyä partitioidun taulun kaltaisesti.

Partitioinnilla tavoitelluista eduista kaksi merkittävintä ovat rinnakkainsuorituksen mahdollistaminen materialisoitujen näkymien päivittämisen yhteydessä sekä tiedonhakuja ja päivityksiä nopeuttava partitiointikäsittely (*partition pruning*). Paritiointikäsittelyn ideaa käsiteltiin jo PCT -päivityksen käsittelyn yhteydessä havainnollistaen asiaa kuvalla 14. Optimoija pystyy suoritettavan SQL -lauseen perusteella päättelemään tarvitsemansa partitiot samalla vähentäen kyselyn suorittamisessa tarvittavien I/O -aikaa vaativien lohkojen käsittelymäärää. Lohkojen käsittelymäärän vaikutusta

I/O -vasteaikaan on havainnollistettu liitteessä 1. Erona kuvan 14 tilanteeseen on, että partitioitaessa materialisoitu näkymä, sijaitsevat osiot fyysisestikin erillään toisistaan. Yleensä partitiointi ei merkittävästi tehosta tiedonhakujen vasteaikoja. Suurempi hyöty saavutetaan päivityksien yhteydessä heijastettiinpa niitä inkrementaalisesti tai enemmän tietomassaa sisältävien päivitysajojen kautta.

Materialisoidun näkymän partitioimiseksi sisältää materialisoidun näkymän luontilause option PARTITION BY, joka kuvassa 8 sisältyy tarkemmin aukikirjoittamattomaan lohkoon ”partition stuff”. Kuvassa 20 on havainnollistettu partitioidun materialisoidun näkymän luomista.

```
CREATE MATERIALIZED VIEW ORDER_FACTS_AGGREGATE_MW
PARTITION by RANGE(day_key) (
     partition orders_jan2003
          values less than (TO_DATE('01022003'','DDMMYYYY'))
          tablespace orders_jan2003,
     partition orders_feb2003
          values less than (TO_DATE('01032003'','DDMMYYYY'))
          tablespace orders_feb2003,
     partition orders_mar2003
          values less than (TO_DATE('01042003'','DDMMYYYY'))
          tablespace orders_mar2003 )
     BUILD IMMEDIATE
AS SELECT
          product_key AS product_key,
          salesperson_key AS salesperson_key,
          day_key AS day_key,
          SUM(quantity_sold) AS quantity_sold,
          SUM(order_dollars) AS order_dollars,
          SUM(cost_dollars) AS cost_dollars
     FROM
          ORDER_FACTS
     GROUP BY
          product_key,
          salesperson_key,
          day_key;
```

Kuva 20: Esimerkki kuvassa 9 esitetyn koostetun faktataulun partitiointimahdollisuudesta.

Tutkielmassa ei ryhdytä tarkemmin tarkastelemaan valintaa eri partitiointikeinojen välillä, sillä kysymys asetetaan yleensä jo faktataulujen fyysisen suunnittelun aikana.

4.5.3 Periaatteita materialisoitujen näkymien tiedonkäsittelyn optimoimiseksi

Kolmannessa luvussa on käsitelty erilaisia materialisoitujen näkymien käsittelyyn liittyviä ominaisuuksia ja niiden tarjoamia mahdollisuuksia eri käyttötilanteisiin sovellet-

tavaksi. Powell (2005) on koonnut yhteen muutamia tärkeimpiä eri tekniikoiden käyttämiseen liittyviä ohjenuoria, joilla on merkittäviä vaikutuksia materialisoitujen näkymien tiedonkäsittelyn vasteaikoihin:

- Hyödynnä ehdottomasti kyselyjen uudelleenkirjoitusta, jos vain tietokannanhallintaympäristö ja luotujen koosteiden ominaisuudet sitä tukevat.
- Inkrementaalinen päivitys varsinkin ON COMMIT -optiota käytettäessä heikentää materialisoitujen näkymien päivityksen vasteaikoja. Materialisoiduille näkymille tarkoitettuja lokeja ei tulisi luoda varmuuden vuoksi, vaan ainoastaan inkrementaalista päivitystä tarvittaessa todelliseen tarpeeseen perustuen.
- Määritä tarkkaan koosteiden todelliset päivitystiheystarpeet loppukäyttäjän asettamiin toiminnallisiin määrityksiin nähden. Staattisille koosteille optio NEVER REFRESH on oikea vaihtoehto. Jos kooste ei vaadi inkrementaalista päivitystä, optio USING NO INDEX on usein tarkoituksenmukainen päivitettäessä koosteisiin paljon muutoksia kerralla.
- Käytä optioita NOLOGGING, COMPRESS, PARALLEL ja TABLESPACE<skeema> niihin kuuluvissa tilanteissa.

4.6 Materialisoitujen näkymien hallinta

Tässä luvussa on tähän mennessä keskitytty käsittelemään materialisoitujen näkymien ominaisuuksia tarkastelemalla ominaisuuksiin vaikuttavia optioita materialisoidun näkymän luontilauseessa. Palaamalla tarkastelussa materialisoitujen näkymien ajatteluun yhtenä Oracle 10g -tietokannanhallintajärjestelmän sisältämistä tietokantaobjekteista, muistutetaan tässä kohden objektien hallinnan muista piirteistä.

Ensinnäkin muiden tietokantataulujen tavoin materialisoitujen näkymien ominaisuuksien muuttamiseksi objektin luomisen jälkeen on olemassa oma SQL -tiedonhallintakielen lause. Materialisoiduille näkymille lause on ALTER MATERIA-

LIZED VIEW. Muutoslauseen avulla materialisoidulle näkymälle voidaan suorittaa kuusi toimenpidettä (Oracle, 2005):

- päivitystavan vaihtaminen (FAST, FORCE, COMPLETE, NEVER)
- päivityshetken vaihtaminen (ON COMMIT, ON DEMAND)
- materialisoidun näkymän uudelleen kääntäminen (COMPILE)
- kyselyjen uudelleenkirjoituksen hyödyntämisen määrittäminen
- materialisoidun näkymän tilan asettaminen päivitetyksi, vaikka materialisoituun näkymään ei vielä olisi kaikkia lähdetaulun muutoksia heijastettu
- partitioinnin ylläpito-operaatioiden suorittaminen.

Materialisoidun näkymän kääntäminen muunnoslauseen sisältämällä COMPILE -optiolla on tarkoitettu tilanteisiin, joissa materialisoitu näkymä on joutunut invalid -tilaan. Materialisoidun näkymän uudelleen kääntäminen on kohtuullisen nopea toimenpide tietomäärästä riippumatta. Jos materialisoitu näkymä ennen invalid -tilaan joutumistaan oli määritetty hyödynnettäväksi uudelleenkirjoituksessa, palaa se uudelleenkääntämisen jälkeen optimoijan käyttöön.

Kaikki muut materialisoituihin näkymiin kohdistuvat muutokset suoritetaan poistamalla olemassa oleva materialisoitu näkymä ja luomalla se uudestaan luontilausetta käyttäen.

Materialisoitujen näkymien poistamiseksi SQL -tiedonhallintakielestä löytyy muiden objektien tavoin DROP -lause. Perinteiseen muiden tietokantaobjektien yhteydessä tapahtuvaan DROP -lauseen toimintaan verrattaessa, sisältyy lauseen toimintaan erityispiirre materialisoitujen näkymien osalta. Jos DROP -lauseella yritetään hävittää ON PREBUILT -optiolla luotua materialisoitua näkymää, ei materialisoidun näkymän fyysinen tietokantataulu poistu, vaan näkymä palaa perinteiseksi tietokantatauluksi. Kyseisen toiminnan ymmärtää, kun palautetaan mieleen aiemmin käsitelty ON PREBUILT -option toiminta. Luotaessa materialisoitu näkymä olemassa olleesta tietokan-

tataulusta, luontilause määritteli vain materialisoidulle näkymälle tarvittavat metatiedot.

Edellä käsiteltyjen materialisoitujen näkymien hallintaan liittyvien SQL -tiedonhallintakielen lauseiden lisäksi tulee muistaa toinen tietokantaobjektien hallinnoimiseen vaikuttava asia. SQL -tiedonhallintakielen lauseiden käyttämiseksi tulee ylläpitäjälle olla ensiksi myönnetty GRANT -lauseella sopivat oikeudet. Sopivien oikeuksien määrittelyyn vaikuttaa muun muassa se, ovatko koosteen lähdetaulut ylläpitäjän oman tietokantakaavion osana vai ei. Vastaavasti myönnettäviin oikeuksiin vaikuttaa koosteiden käyttötarkoitus. Monien materialisoitujen näkymien luontilauseen sisältämien optioiden käyttämiseksi tulee ylläpitäjälle olla myönnetty muistakin oikeuksia, kuten QUERY REWRITE -oikeus.

4.7 Materialisoidut näkymät OLAP -raportoinnissa

Materialisoidut näkymät ovat koosteiden toteutuskeinona mahdollistamassa riittäviä vasteaikoja raportoinnissa ja analysoinnissa käytetyille tiedonhauille varastoimalla valmiiksi laskettuja tietoja tai tuomalla ne fyysisesti paremmin loppukäyttäjän ulottuville. Yhdessä tietokannanhallintajärjestelmän dimensioiden ja hierarkioiden määrittelyyn tarkoitettujen ratkaisujen kanssa materialisoidut näkymät antavat tuen OLAP -raportoinnin moniulotteiselle tiedon tarkastelulle ja eri tarkastelutasojen huomioimiselle. Näiden OLAP -raportointia tukevien objektien lisäksi Oracle 10g -tietokannanhallintajärjestelmä sisältää SQL -kyselykielen laajennuksia eirtyisesti OLAP -tiedonhakuja tukemaan. Näitä laajennuksia ovat esimerkiksi GROUB BY -lohkon laajennukset optioilla CUBE, ROLLUP ja GROUPING SETS. Optioilla pystytään vaikuttamaan SQL -kyselyjen yhteydessä kombinaatioihin, joiden perusteella kyselyn palauttamaan tulosjoukkoon muodostetaan myös summatiedot halutuista tarkastelunäkökulmista. Suurinta osaa laajennuksista voi käyttää myös osana materialisoidun näkymän SQL -lausetta, mutta tässä tapauksessa koosteen uudelleenkäytettävyys eri BI -sovellusten kannalta usein kärsii. (Hobbs *& al.*, 2005; Kimball *& al.*, 2008)

Sen lisäksi, että materialisoidut näkymät muiden tekniikoiden kanssa tukevat jo omalta osaltaan OLAP -raportointia, ovat ne koosteina tärkeässä roolissa toimiessaan lähtötietoina erillisille OLAP -raportointia tukeville loppukäyttäjän työkaluille. Nämä työkalut voivat olla relaatiomallia käyttäviä BI -työkaluja tai erillisiä OLAP -välineitä, joilla on käytössä omia relaatiomallista poikkeavia tiedon mallintamistekniikoita. Useimmat välineet tukevat tiedonhakuja relaatiotietokannoista ja käyttötarkoituksesta riippuen ei välttämättä atomitasoista tietoa lähdetä toistamaan OLAP -raportointivälineen omiin varastoihin vaan tiedot haetaan esimerkiksi Oracle 10g tietokannanhallintajärjestelmän tapauksessa materialisoiduilla näkymillä toteutetuista koosteista.

5 Materialisoitavien näkymien valinta

Tutkielmassa on tähän mennessä käsitelty koosteiden käyttötapoja tiedonhakujen asettamiin vasteaikahaasteisiin vastattaessa. Koosteiden teknisempää toteutusta tarkasteltiin Oracle 10g -tietokannanhallintajärjestelmän materialisoitujen näkymien toteutusta läpikäymällä. Tässä luvussa pohditaan lopuksi lyhyesti kysymystä, miten valita muodostettavat koosteet. Ensiksi tarkastellaan valintaan vaikuttavia tekijöitä, jonka jälkeen tarkastellaan siirtymistä manuaalisesta algoritmipohjaisesta analysoinnista tietokannanhallintajärjestelmien työvälineiden avulla tapahtuvaan analysointiin. Esimerkkinä käytetään jälleen Oracle 10g -tietokannanhallintajärjestelmää.

5.1 Valintaan vaikuttavat osa-alueet

Kuten suunniteltaessa tietokantatauluissa käytettäviä indeksejä, tasapainoillaan koosteidenkin suunnittelussa saavutettujen hyötyjen ja niiden aikaansaamiseksi vaadittavien kustannusten välillä. Kohdassa 2 alustettiin valintakysymystä viittaamalla mix-ajattelumalliin, jossa tasapainoittelun osatekijöinä toimivat tietomix, kustannusmix ja saatavuusmix. Käytännössä termit purkautuvat edelleen yksityiskohtaisemmille tarkastelutasoille. Ciferri & Souza (2001) nostavat esille saatavuuden osalta vasteaikatavoitteet ja kustannuksien osalta ylläpitokustannukset. Käsiteltävään tietoon liittyen tasapainottelutekijäksi muodostuu skaalautuvuus ja koostenavigaattoreiden mahdollisesti ylläpidolle asettamat rajoitteet. Koostenavigaattorin rajoituksista he nostavat esille loppukäyttäjien raportointi- ja analysointivälineinä käyttämien web -sovelluksien luoman tarpeen tukea maantieteellisesti hajautettujen replikoituja koosteita sisältävien järjestelmien koosteiden hallintaa.

Kimball *& al.* (2008) asettaa reunaehdot kuvaamalla koosteiden valintaa taiteenlajiksi, jossa taidetta on olla muodostamatta koosteita liikaa ja toisaalta liian vähän. Kustannuskysymyksiin liittyen Kimball *& al.* (2008) paneutuvat ylläpitokustannuksien lisäksi myös laitteistokustannuksien tasolle. Kuten liitteessä 1 on kerrottu, vaikuttaa esimerkiksi laitteiston käsittelemä lohkon suuruus (*block size*) vasteaikoihin. Samalla he

muistuttavat suorittimien määrän valinnan vaikutuksesta rinnakkaissuorituksen mahdollistamiseen ja partitioinnin antamaan tukeen. Jos muut optimointikeinot ovat tehokkaammin käytössä, vähenee koosteiden tarve. Kimball *& al.* (2008) pohtivat kattavasti myös eri BI -välineiden vaikutusta valintakysymykseen. Vaikka he suosittelevatkin koostenavigaattorin toteuttamista nimenomaan tietokannanhallintajärjestelmän puolella, voi käytännön elämässä tulla vastaan tilanne, että BI -välineen koosteiden käsittelyyn antama tuki riittää riittävien vasteaikojen ylläpitämiseen. Kustannustekijöihin luettakoon vielä koosteiden usein merkittäväksi muodostuva levytilan tarve.

Lightstone *& al.* (2007) muistuttavat tosielämän realiteetistä, joka valintakysymyksen osalta tarkoittaa pyrkimystä olla tavoittelematta optimaalista täydellisyyttä. Heidän mukaansa useimmissa tapauksissa on päämäärän voinut tyytyväisin mielin katsoa saavutetuksi päästessä 85% optimaalisesta tehokkuudesta.

5.2 Algoritmeista valintapäätöstä tukeviin työkaluihin

1900 -luvun puolella ei tietokannanhallintajärjestelmät vielä tukeneet koosteiden hallintaa samalla tavoin kuin nykyään. Oraclessa SNAPSHOT -tietokantaobjekti edelsi materialisoitujen näkymiä, mutta tiedon päivitykset ja tietojen kuranttiuden seuranta jäi ylläpitäjän skriptien varaan. Idea materialisoiduista näkymistä oli kuitenkin jo syntynyt alan kirjallisuudessa.

Ennen tietokannanhallintajärjestelmien koosteiden hallinnalle tarjoamaa tukea valintakysymyksessä käytettiin valintapäätöksiä helpottamaan tunnettuja algoritmeja. Yksi tunnetuimmista valinta-algoritmeista on *ahne (greedyn) algoritmi*, joka lähtee suorituskyvyn ja materialisoitujen näkymien tilatarpeen tasapainoittamisesta tilatarpeen toimiessa kriittisenä päätöstä ohjaavana tekijänä. Jos päätöksenteossa halutaan arvioida mukaan myös ylläpitokustannuksia, löytyy käyttöön *Baraliksen algoritmi*. (Ciferri & Souza, 2001; Gupta & Mumic, 1998; Chan *& al.*, 1999)

Nykyään tilanne on toinen. Yhä useammat tietokannanhallintajärjestelmät tukevat jollakin asteella koosteiden hallintaa ja yhtenä siihen kuuluvana osana koosteiden valin-

taa. Oracle 10g -tietokannanhallintajärjestelmässä työkaluja on monia. Kokonaisuus ei kuitenkaan lähde liikkeelle työkaluista, vaan tietovarastojärjestelmien muuttuvista raportointi- ja analysointitarpeista. Koska ad hoc -raportointi on olennainen osa tietovarastojärjestelmän tarjoamia palveluita, ei järjestelmän voi olettaakaan olevan varautunut kaikkiin loppukäyttäjien muodostamiin tiedonhakuihin automaattisesti. Tärkeämmäksi muodostuukin tiedonhakujen kontrollointi. Kimball *& al.* (2008) korostavat tärkeinä tilannetiedon lähteinä käyttäjäpalautteiden keräämistä sekä virhe- ja seurantalokien seuraamista. Erityisen ongelman muodostavatkin kyselyt, jotka eivät seurattaville lokitiedostoille päädy. Samasta lähtökohdasta tarkastelevat asiaa myös Hobbs *& al.* (2005). Järjestelmän ylläpitäjän kuuluisi tarkkailla jatkuvasti järjestelmässä suoritettujen SQL -kyselyjen vasteaikojen tilannetta.

Hobbs *& al.* (2005) suosittelevat käyttämään graafista tietokannanhallintatyökalua Oracle Database 10g Enterprise Manageria suoritettujen SQL -tietokantakyselyiden seurantaan. Välineellä voi tunnistaa pitkään suorittuneet tai keskeytyneet kyselyt, jotka voi ottaa välineen avulla tarkempaan käsittelyyn. Ylläpitäjän tulisi pohtia näiden kyselyjen osalta esimerkiksi sitä, toistuuko kyselyissä samankaltaisia uusia tiedonhakuja.

Varsinainen materialisoitujen näkymien valintaan liittyvä työkalu Oracle 10g -tietokannanhallintajärjestelmässä on SQL Access Advisor. Ylläpitäjä voi poimia ongelmakyselyt tämän työkalun suorittamaan tarkempaan analysointiin. SQL Access Advisor tutkii kyselyn sisällön ja ehdottaa kyselyn tehokkaamman suorittamisen vaatimat indeksit ja materialisoidut näkymät. Tähänkin työkaluun liittyen Hobbs *& al.* (2005) suosittelevat käyttämään graafista versiota, mutta komentoriviltä käytettäväksi työkalun saa DBMS_ADVISOR -tietokantapaketin kautta. Työkalujen antamien tuloksien jälkeen ylläpitäjän vastuulle kuitenkin lopulta jää arvioida saavutettavien hyötyjen ja kustannusten suhde. Aina ei myöskään ole mahdollista muodostaa sopivaa koostetta tiedonhaun tukemiseksi. Näissä tapauksissa vaihtoehdoksi muodostuu työväline SQL Tuning Advisor, joka analysoi mahdolliset muutostarpeet itse SQL -kyselyssä. Edellä mainittujen materialisoitujen näkymien valintaa tukevien työkalujen lisäksi Oracle 10g -tietokannanhallintajärjestelmä sisältää omia työvälineitä esimerkiksi partitioin-

nin, muistinkäytön ja kyselyjen uudelleenkirjoittumisen seurantaan. Niitä ei tässä tutkielmassa läpikäydä tarkemmin.

Vaikka materialisoitujen näkymien valintaan onkin tueksi työkaluja, tulee ohjelmistokehittäjän itse pystyä analysoimaan todellinen tilanne. Itse asiassa koko valintakysymys on suurempi kokonaisuus, joka alkaa jo tietovarastojärjestelmän määrittely- ja suunnitteluvaiheen aikana. Tietojärjestelmätoimittajalla on haastava tehtävä pystyä hahmottamaan asetettujen toiminnallisten vaatimuksien perusteella koosteiden tarpeen määrittäminen. Jos toiminnallisissa määrityksissä on mukana erittäin haasteellisia vasteaikatavoitteita, on toimittajalle uhkarohkeaa lähteä järjestelmää koosteiden hallinnan kustannuksia huomioimatta tarjoamaan. Toisaalta, vähintään yhtä haasteellinen on asiakaspuolen tehtävä valita tarjouksista sopiva. Jos toiminnallisissa vaatimuksissa on vasteaikakriittisiä tiedonhakuja, olisi asiakkaalta yhtä uhkarohkeaa lähteä valitsemaan tarjousta, johon ei koosteiden hallinnalle ole kustannuksia laskettu. Materialisotujen näkymien valinta on laaja kysymys.

6 Yhteenveto

Ohjelmistokehittäjältä vaadittava askel tapahtumankäsittelyjärjestelmien kehitystyöstä tietovarastojärjestelmien kehitystyöhön voi muodostua tyhjän päälle polkaisuksi, jos ohjelmistokehittäjä ei siihen mennessä ole valmentautunut tulevaan huolehtimalla dimensionaalisen tiedonmallinnustavan omaksumisesta, jatkuvan suorituskyvyn osatekijöiden tarkkailuvaateen sisäistämisestä ja koosteiden merkityksen tiedostamisesta raportoinnille ja analysoinnille asetettujen vasteaikatavoitteiden saavuttamisessa.

Porautumisen ja muuttuvien analysointi- ja raportointitarpeiden mahdollistamiseksi tietovarastojärjestelmään tulisi tieto kerätä mahdollisimman hienojakoisella tasolla. Lähestymistapa johtaa väistämättä siihen, että varastoitavan tiedon määrä kasvaa tietovarastojärjestelmissä erittäin suureksi. Teratavujen tietomääriä käsiteltäessä eivät perinteiset tapahtumankäsittelyjärjestelmissäkin käytetyt vasteaikojen optimointikeinot enää yksin riitä, vaan mukaan tarvitaan I/O -aikakustannuksia voimakkaammin vähentävä koosteiden muodostamiseen perustuva tekniikka.

Koosteiden muodostaminen perustuu ajatukselle johtaa perusskeemaan kuuluvista hienorakeisimmat tiedot sisältävistä tietokantatauluista omia tietokokonaisuuksia fyysisesti erillisiin tietokantatauluihin. Tietojen koostamistapa riippuu käyttötarkoituksesta. Usein tietoa koostetaan fakta- ja dimensiotauluista tarkoituksena muodostaa analysointi- ja raportointitasoja paremmin vastaavia yhteenvetotietoja I/O -vasteaikojen vähentämiseksi tiedonhakujen yhteydessä. Toisaalta koosteet voivat olla esimerkiksi suoria kopioitakin lähdetaulujen tiedoista. Tällöin voi olla kysymys tiedon replikoinnista maantieteellisesti lähemmäksi loppukäyttäjien käyttämiä BI -sovelluksia.

Koosteiden avulla saavutetuilla vasteaikahyödyillä on omat kustannuksensa. Koosteet vievät olennaisesti levytilaa ja koosteiden ylläpito tietokannanhallintajärjestelmästä riippumatta vaatii ylläpitotyötä. Kuten indeksejä käytettäessä, tulee koosteiden perustamispäätöksen tapahtua harkitusti. Tavoitetilanteessa onnistutaan muodostamaan monia analysointi- ja raportointitarpeita tukevia koosteita, joita tietokannanhallintajärjestel-

män sisältämä koostenavigaattori pystyy loppukäyttäjälle näkymättömästi hyödyntämään uudelleenkirjoittaen tiedonhakujen SQL -kyselyt käyttämään perusskeeman taulujen sijasta tilanteeseen sopivia koosteita.

Tarkasteltaessa Oracle 10g -tietokannanhallintajärjestelmän materialisoituihin näkymiin perustuvaa ratkaisua koosteiden toteuttamisesta, tuli esille kokonaiskäsitys koosteiden hallinnan toteutustekniikan muodostamista osa-alueista. Käyttötarkoituksesta riippuen ylläpitäjällä on mahdollisuus ottaa kantaa muun muassa koosteiden päivitystapaan, päivityshetkeen, koosteiden tiedonkäsittelyn optimointikeinoihin ja koosteiden käytettävyyteen kyselyjen uudelleenkirjoituksessa. Tietokannanhallintajärjestelmät ovat viime vuosina kehittäneet koosteiden hallinnan tekniikoita esimerkiksi automatisoimalla ennen käsitöinä suoritettuja päivitysrutiineja. Automatisointiin liittyy kuitenkin paljon koosteille asetettuja vaatimuksia, joiden analysointi ja toteuttaminen vaatii ylläpitäjältä oman aikansa. Rajoitteita ei kuitenkaan ole tarkoitus muistaa ulkoa tai selata manuaaleista, vaan merkityksellisempää on omaksua tietokannanhallintajärjestelmän tarjoamat työkalut koosteiden tilanteiden ja muutostarpeiden analysoimiseksi.

On väitetty, ettei raportoinnin ja analysoinnin vasteajoilta riittävää tietovarastojärjestelmää pystytä ilman koosteita saavuttamaan. Suurten teratavuja sisältävien tietovarastojärjestelmien osalta tämä on helppo lukea totuudeksi. Koosteiden käyttö tulisi huomioida kuitenkin esimerkiksi levytilatarpeen osalta jo tarjouskilpailuvaiheessa, josta herää kysymys asian erottelusta järjestelmän hintalapusta ja asiakkaan mahdollisuuksista tunnistaa tästä aiheutuneen hintaeron vasteaikojen palauttama todellinen arvo. Yksi asia on kuitenkin varma. Muutos tietovarastojärjestelmien analysointi- ja raportointitarpeissa on jatkuvaa. Tähän liittyen olennaiseksi osaksi tietovarastojärjestelmän ylläpitotyötä muodostuu tiedonhakujen seuranta ja vasteaikojen optimointitarpeiden tunnistaminen. Koosteita ei pidä perustaa hetken mielijohteesta. Ennen koosteen luomista täytyy olla varmuus siitä, ettei vasteaikaongelma piile ohjelmistokehittäjän tuottamassa SQL -kyselyssä. Sen vuoksi väittäisin, että ohjelmistokehittäjän, jolle SQL -kyselyn analysoimiseksi tarvittavat työkalut ovat omassa kehitysympäristössään vieraita, ei askelta tietovarastojärjestelmien kehitystyöhön kannata harkita.

Viitteet

1keydata (2008) *Data Warehousing and Business Intelligence*. WWW-sivusto, `http://www.1keydata.com/datawarehousing/datawarehouse.html` (23.3.2008).

Adamson, C. (2006) *Mastering Data Warehouse Aggregates*. Wiley Publishing, Indianapolis.

Borden, N. H. (1964) The Concept of the Marketing Mix. *Journal of Advertising Research* **4**(6), 2-7.

Chan, G., Qing, L., Feng, L. (1999) Design and Selection of Materialized Views in a Data Warehousing Environment: A Case Study. *DOLAP '99, ACM Second International Workshop on Data Warehousing* (The ACM Digital Library), ACM, New York, 42-47.

Ciferri, C., Souza, F. (2001) Materialized Views in Data Warehousing Environments. *XXI International Conference of the Chilean Computer Science Society* (Computer Science Society), SCCC, Punta Arenas, 3-12.

Connolly, T. M., Begg, C. E. (2005) *Database Systems: A Practical Approach to Design, Implementation and Management*. Pearson Education Limited, Essex.

Gupta, A., Mumick I. (1999) *Materialized Views: Techniques, Implementations, and Applications*. The MIT Press, England.

Hobbs, L., Hillson, S., Lawande, S., Smith, P. (2005) *Oracle Database 10g Data Warehousing*. Elsevier Digital Press, Burlington.

Inmon, W. H. (1996) *Building the Data Warehouse*. John Wiley & Sons, New York.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., Becker, B. (2008) *The Data Warehouse Lifecycle Toolkit, Second Edition*. Wiley Publishing, Indianapolis.

Lightstone, S., Teorey, T., Nadeau, T. (2007) *Physical Database Design*. Morgan Kaufmann publications, San Francisco.

O'Neil, P., O'Neil, E. (2001) *Database: Principles, Programming and Performance, Second Edition*. Morgan Kaufmann publications, San Francisco.

Oracle (2005) *Oracle Database SQL Reference, 10g Release 2(10.2)*. Oracle Press, USA.

Pendse, N. (2008) *What is OLAP? The OLAP Report*. WWW-sivusto, `http://www.olapreport.com/fasmi.htm` (26.2.2008).

Roussopoulos, N. (1982) View Indexing in Relatonal Databases. *ACM Trans. Database systems* **7**(2), 258-290.

Shneiderman, B. (1984) Response Time and Display Rate in Human Performance with computers. *ACM Computing Surveys (CSUR)* **16**(3), 265-285.

Silberschatz, A., Korth, H. F., Sudarshan, S. (2006) *Database System Concepts, Fifth Edition*. McGraw-Hill, New York.

Stackowiak, R., Rayman, J., Greenwald, R. (2007) *Oracle Data Warehousing*. Wiley Publishing, Indianapolis.

Liite 1: Esimerkki I/O -vasteaikojen arvioinnista

Havainnollistaakseen heuristiikkojen hyötyjen kannattavuutta LightStone *& al.* (2007) esittelevät esimerkin tietokantataulujen välisten liitoksien ja rajaustekijöiden järjestyksen vaikutuksesta I/O -aikakustannuksiin SQL -kyselyjen suorituksessa. Kuvassa 21 on esitetty otos esimerkissä käytettyjen tietokantataulujen sisällöstä. Part -taulu ylläpitää tietoja tavaranvalmistuksessa käytetyistä osista. Supplier -taulu ylläpitää tietoja tavarantoimittajista, ja Shipment -taulussa on kuvattu tavarantoimittajien toimituksien sisältämät osat.

Part (P)			**Supplier (S)**				**Shipment (SH)**			
pnum	**pname**	**wt**	**snum**	**sname**	**city**	**status**	**snum**	**pnum**	**qty**	**shipdate**
p1	bolt	3	s1	brown	NY	3	s1	p2	50	1-4-90
p2	nail	6	s2	garcia	LA	2	s1	p3	45	2-17-90
p3	nut	2	s3	kinsey	NY	3	s2	p1	100	11-5-89
							s2	p3	60	6-30-91
							s3	p3	50	8-12-91

Kuva 21: Kuvaus esimerkissä käsiteltävistä tietokantatauluista (LightStone *& al.*), 2007.

Esimerkin laskennassa käytettävät tietokantataulujen kenttien koot tavuina ilmaistuna sekä muut laskentaan vaikuttavat oletukset:

- Supplier: snum(5), sname(20), city(10), status(2) -> yhden tietokantataulun rivin muodostaman tietueen koko 37 tavua
- Part: pnum(8), pname(10), wt(5) -> yhden tietokantataulun rivin muodostaman tietueen koko 23 tavua
- Shipment: snum(5), pnum(8), qty(5), shipdate(8) -> yhden tietokantataulun rivin muodostaman tietueen koko 26 tavua
- Käytettävä lohkon koko (block size) = 15 000 tavua
- Laskettaessa tietokantataulun järjestämiseen kuluvaa aikaa, käytetään arvioinnissa kaavaa $2 * nb * log_M nb$, jossa nb kuvastaa lajiteltavan taulun tietomäärän

lohkojen määrää (O'Neil & O'Neil, 2001; Silberschatz *et al.*, 2006). LightStone *& al.* (2007) käyttävät M -arvona lukua 3.

- LightStonen *& al.* (2007) esimerkistä poiketen lohkojen käsittelymäärän lisäksi arvioidaan läpikäytävässä esimerkissä vaadittua I/O -aikaa olettaen prosessorin yhden lohkon vaatimaksi käsittelyajaksi 5,8 ms erään jaetun ympäristön laitteisto- ja ympäristötekijöiden mukaisesti.

I/O -vasteaikojen pohtimiseksi muodostetaan SQL -kysely tarkoituksena vastata kysymykseen, minkä nimisiä osia ovat New Yorkin alueen toimittajat toimittaneet. SQL -kysely näyttäisi seuraavalta:

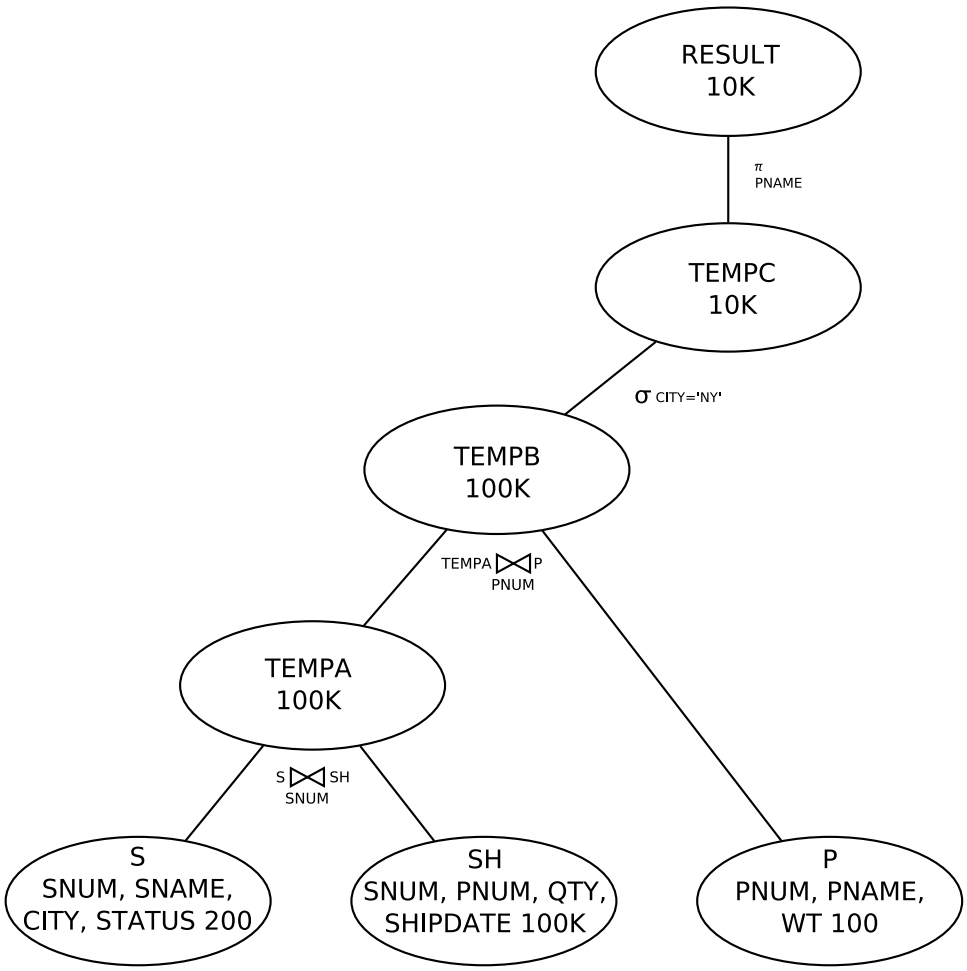
```
SELECT p.pname
FROM Part P, Shipment SH, Supplier S
WHERE SH.snum = S.snum
    AND SH.pnum = P.pnum
    AND  S.city = 'NY';
```

SQL -kyselyn perusteella ryhdytään tarkastelemaan kyselyn käsittelemiseksi tarvittavien lohkojen käsittelykertojen lukumäärää kahdelta eri näkökantilta. Ensiksi tarkastellaan vaihtoehtoa, jossa ensiksi suoritetaan tietokantataulujen väliset liitokset ja sitten vasta tulosjoukon rajaaminen kaupunkikoodin perusteella. Toisessa vaihtoehdossa rajaukset suoritetaan ensin ja vasta sitten tietokantataulujen väliset liitokset.

Vaihtoehto 1 - tietokantataulujen väliset liitokset ensin

Kuvassa 22 on esitetty kyselyn kyselysuunnitelma (*execution plan*) ensimmäisen vaihtoehdon osalta Roussopouloksen (1982) mallinnustapaa käyttäen. Arvioitaessa kunkin suoritusvaiheen tarvitsemaa I/O -aikakustannusta, hahmotellaan kyselyssä mukana olevien taulujen sekä suorituksessa tarvittavien väliaikaisten aputaulujen luku- ja kirjoitusoperaatioissa käsiteltävien lohkojen määrää sekä taulujen järjestelytarvetta (*sort*). Yhdessä tietokantataulun luku- ja kirjoitusoperaatiossa käytetty lohkojen määrä saa-

daan laskemalla ensiksi jaksotuskerroin, joka kuvaa yhteen lohkoon mahtuvien tietokantataulun rivien lukumäärän (*blocking factor, BF*). Jaksotuskertoimen laskemisen jälkeen jaetaan taulun sisältämien rivien lukumäärä lasketulla jaksotuskertoimella. Kuvan 23 taulukossa esitetään vaihtoehtoon 1 laskettuja jaksotuskertoimia.



Kuva 22: Vaihtoehdon 1 suoritusketju (LightStone *& al.*, 2007).

Table	Row Size	No. Rows	BF	Scan Table (no. Blocks)
supplier (S)	37 bytes	200	405	1
part (P)	23 bytes	100	652	1
shipment (SH)	26 bytes	100K	576	174
TEMPA (S join SH)	58 bytes	100K	258	388
TEMPB (TEMPA join P)	73 bytes	100K	205	488
TEMPC (select TEMPB)	73 bytes	10K	205	49

BF is the blocking factor or rows per block (estimated with average row size).

Kuva 23: Vaihtoehto 1 - liitokset ensin (LightStone *& al.*), 2007.

Ensimmäisessä vaihtoehdossa tietokannanhallintajärjestelmä aloittaa kyselyn suorittamisen yhdistämällä tietokantataulujen Supplier ja Shipment tietosisällön ja muodostamalla tulosjoukosta väliaikaisen aputaulun TEMPA. Koska Supplier -taulu on kooltaan pieni mahtuessaan kokonaan yhteen lohkoon, ei tämän liitoksen suorittamiseksi lasketa mukaan järjestelykustannuksia. Toimenpiteestä muodostettaisiin seuraavanlainen laskelma lohkojen käsittelymäärän laskemiseksi:

```
Käsiteltävien lohkojen määrä (toimenpide 1)
 = lue S + lue SH + kirjoita TEMPA
 = ceiling(200/405) + ceiling(100 000/576) + ceiling (100 000/258)
 = 1 + 174 + 388
 = 563
```

Seuraavassa vaiheessa muodostettu aputaulu TEMPA ja tietokantataulu Part liitetään toisiinsa. TEMPA:n ollessa kookas ja indeksoimaton taulu, täytyy laskennassa laskea taululle TEMPA järjestelykustannuksia. Toimenpiteestä 2 saataisiin seuraavanlainen laskelma:

```
Käsiteltävien lohkojen määrä (toimenpide 2)
 = lue P + järjestele TEMPA + lue TEMPA + kirjoita TEMPB
 = 1 + 2 * 388 * log(3)388 + 388 + 488
 = 1 + 4214 + 388 + 488
 = 5 091
```

Kolmannessa vaiheessa päästään suorittamaan rajaaminen taulun TEMPB sisältämien tietojen pohjalta ja tulosjoukosta muodostetaan aputaulu TEMPC. Lopuksi kyselyn tulos luettaisiin taulusta TEPC. Kyseisiin vaiheisiin liittyvät lohkojen käsittelymäärien laskennat olisivat seuraavat:

```
Käsiteltävien lohkojen määrä (toimenpide 3)
 = lue TEMPB + kirjoita TEMPC
```

```
 = 488 + 49
 = 537

Käsiteltävien lohkojen määrä (toimenpide 4)
 = lue TEMPC
 = 49
```

Tietokantataulujen välisten liitosten suorittaminen ensiksi johtaisi edellä olevien laskentojen perusteella lopputulokseen:

```
Käsiteltävät lohkot (kaikki toimenpiteet yhteensä)
  = 563 + 5 091 + 537 + 49
  = 6240
```

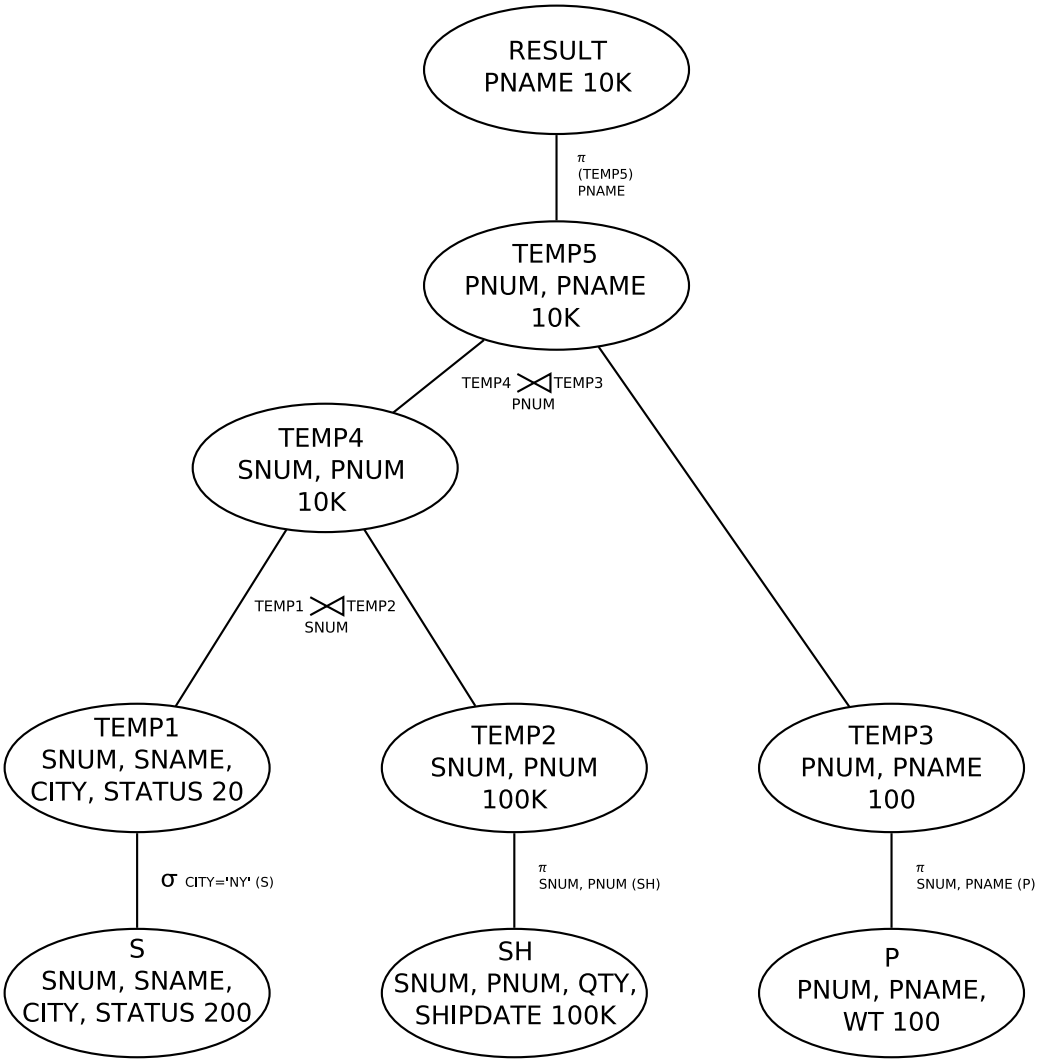
Käytettäessä prosessorille laskettua yhden lohkon käsittelyajaksi annettua keskimääräistä käsittelyaikaa 5,8 ms / lohko, muodostuisi kyselyn I/O -kustannukseksi noin 36 sekunttia.

Vaihtoehto 2 - rajaukset ensin

Vaihtoehtoisessa käsittelytavassa arvioidaan kyselyn I/O -kustannuksia tilanteessa, että SQL -kyselyyn sisältyvä rajaus suoritetaan ensin ja tietokantataulujen väliset liitokset vasta rajauksen jälkeen. Kuvissa 24 ja 25 on esitetty toiseen vaihtoehtoon liittyvä suorituksen kulku ja tauluille lasketut jaksotuskertoimet.

Toisessa vaihtoehdossa ensinmäinen toimenpide muodostuu taulun Supplier tietojen raajamisesta ja aputaulun TEMP1 muodostamisesta. Huomioitavaa tässä vaiheessa on, että TEMP1 muodostuu rajatumman tietojoukon perusteella. Lohkojen käsittelymääräksi laskettaisiin:

```
Käsiteltävien lohkojen määrä (toimenpide 1)
 = lue S + kirjoita 10 prosenttia TEMP1
```



Kuva 24: Vaihtoehdon 2 suoritusketju (LightStone *& al.*, 2007).

Table	Row Size	No. Rows	BF	Blocks to Scan Table
supplier (S)	37 B	200	405	1
part (P)	23 B	100	652	1
shipment (SH)	26 B	100K	576	174
TEMP1 (select from S)	37 B	20	405	1
TEMP2 (project over SH)	13 B	100K	1,153	87
TEMP3 (project over P)	18 B	100	833	1
TEMP4 (TEMP1 semi-join TEMP2)	13 B	10K	1,153	9
TEMP5 (TEMP4 semi-join TEMP3)	18 B	10K	833	13

Kuva 25: Vaihtoehto 2 - rajaukset ensin (LightStone *& al.*, 2007).

```
 = 1 + 1
 = 2
```

Seuraavassa vaiheessa tauluista Shipment ja Part rajataan käsittelyyn vain tarvittavat sarakkeet, jotta seuraavaksi muodostettavissa tietokantataulujen välisissä liitoksissa olisi jokaisesta taulusta mukana vain tarvittavat kentät. Turhaan mukana kuljetettavat sarakkeet suurentavat välitaulujen kokoa ja lisäävät näin käsiteltävien lohkojen määrää. Tietokantataulujen Shipment ja Part projektioista saadaan laskelmat:

```
Käsiteltävien lohkojen määrä (toimenpide 2)
 = lue SH + kirjoita TEMP2
 = 174 + 87
 = 261

Käsiteltävien lohkojen määrä (toimenpide 3)
 = lue P + kirjoita TEMP3
 = 1 + 1
 = 2
```

Kyselyn suoritusketju jatkuu taulujen Supplier ja Shipment liitoksen toteuttamisella. Toisin kuin ensimmäisen vaihtoehdon tapauksessa on alkuperäisten tietokantataulujen sijasta molemmista tietokantatauluista liitoksen jäseninä huomattavasti pienemmät taulut TEMP1 ja TEMP2. Tässä toimenpiteessä käsiteltävien lohkojen määräksi saadaan:

```
Käsiteltävien lohkojen määrä (toimenpide 4)
 = lue TEMP1 + lue TEMP2 + kirjoita TEMP4
 = 1 + 87 + 9
 = 97
```

Tähän mennessä suoritettujen toimenpiteiden seurauksen on luotu lopullisen kyselyn tuloksen muodostamiseksi tarvittavat välitaulut. Viimeisinä kyselyn suorituksen toimenpiteinä muodostetaan välitaulujen TEMP3 ja TEMP4 liitos sekä luetaan kyseisen

liitoksen tuottamasta välitaulusta kyselyn tulos. Viimeisissä toimenpiteissä käsiteltävien lohkojen määräksi saadaan:

```
Käsiteltävien lohkojen määrä (toimenpide 5)
 = lue TEMP3 + lue TEMP4 + kirjoita TEMP5
 = 1 + 9 + 13
 = 23

Käsiteltävien lohkojen määrä (toimenpide 6)
 = lue TEMP5
 = 13
```

Rajauksien suorittaminen ensiksi johtaisi edellä olevien laskentojen perusteella lopputulokseen:

```
Käsiteltävät lohkot (kaikki toimenpiteet yhteensä)
  = 2 + 261 + 2 + 97 + 23 + 13
  = 398
```

Käytettäessä prosessorille laskettua yhden lohkon käsittelyajaksi annettua keskimääräistä käsittelyaikaa 5,8 ms / lohko, muodostuisi kyselyn I/O -kustannuksesi muodostuisi noin 2,3 sekunttia.

Lopputulos

SQL -kyselyjen optimointiin liittyvistä heuristiikoista tässä liitteessä käsitellyllä aikaisia rajauksia painottavalla toimintatavalla on merkittävä vaikutus kyselyjen I/O -aikakustannuksiin. Kustannukset vähenevät, koska kyselyn suoritusketjun aikaisissa vaiheissa vähennetään heti laskennassa mukana kuljetettavaa tietomäärää saavuttaen lohkojen käsittelymäärän väheneminen. Toisaalta esimerkin kautta on ymmärrettävissä myös lohkon koon kasvattamisen tuomat vasteaikahyödyt. Jos tietomäärä pidetään vakiona, voidaan kyselyn suorittamisessa tarvittavaa lohkojen käsittelymäärää vähentää

kasvattamalla lohkon kokoa. Vaikutus ei kuitenkaan ole yhtä suoraviivainen käsiteltävän tietomäärän rajaamiseen nähden, koska suurempien lohkokokojen kanssa yleensä myös prosessorin lohkon käsittelemiseksi tarvitsema aika jonkin verran kasvaa.

Käsiteltyä esimerkkiä voi ajatella myös koosteisiin liitettävän vasteaikojen parantamiseen tähtäävän tarkoituksen näkökantilta. Esimerkin tapauksessa tietomäärät olivat tietovarastojärjestelmien atomitason tietoja ylläpitäviin tauluihin nähden vähäiset. Kun taulujen koot kasvavat gigatavuihin, toimivat koosteet edeltävänä toimenpiteenä jo ennen varsinaisten kyselyjen suorittamisen sisältämiä toimenpiteitä. Koosteiden avulla tietokantataulujen käsittelytaso yritetään saada kyselyjen kannalta I/O -kustannustenkin kannalta paremmin käsiteltävälle tasolle.

Liite 2: Inkrementaaliseen päivitykseen liittyviä vaatimuksia

Taulujen välisiä liitoksia sisältävät koosteet (Hobbs *& al.*, 2005; Oracle, 2005; Powell, 2005):

- Lähtötauluihin luotavissa materialisoiduillen näkymille tarkoitetuissa lokeissa on muutosten yksilöintiin vaikuttavaksi optioksi valittava ROWID.
- Koosteen muodostavassa SQL-kyselyssä tulee SELECT -lohkoon liittää jokaiselle liitoksessa mukana olevalle taululle rowid -sarake mukaan. Esimerkiksi yhdistettäessä asiakkaan ja tilauksen tietoja, tulisi SELECT -lohkon sisältää sarakkeet asiakas.rowid ja tilaus.rowid.
- Jos kysymyksessä on liitos, jonka kardinaliteetti noudattaa joko yhden suhdetta moneen tai monen suhdetta moneen, on liitoksessa mukana olevista sarakkeista ilmoitettava lähtötaulujen lokeissa sarakkeet, jotka eivät ole osana perusavainta. Esimerkiksi koosteessa, jossa yhdistetään asiakkaan ja asiakkaan tilausten tietoja, tulisi tilaus-taululle luotavaan lokiin sisällyttää taulujen välisen liitoksen luonnissa käytetty asiakkaan tunnusta kuvaava sarake.
- Lähtötietoja ylläpitävälle taululle luotavaan materialisoidulle näkymälle tarkoitettuun lokiin on sisällytettävä SEQUENCE -lohkoon koosteen SQL -kyselystä tietojen suodattajana toimivat sarakkeet, jotka eivät ole osana liitoksia eikä perusavainta. Esimerkiki rajattaessa koosteeseen tilaus-taulusta rivejä tilauksen arvon mukaan, tulisi tilauksen arvoa ylläpitävä sarake liittää SEQUENCE-lohkoon.
- Jokainen funktio, kuten SUM(<sarake>), tarvitsee SELECT -lohkoon liitettäväksi myös COUNT(<sarake>) - sekä COUNT(*) -funktiot.
- Käytettäessä STDDEV ja VARIANCE -funktioita, tarvitaan SELECT -lohkoon myös COUNT(<sarake>) - sekä SUM(<sarake>) - funktiot.

- Kaikki GROUP BY -lohkossa ryhmittelytekijöiksi määrätyt sarakkeet on oltava mukana SELECT -lohkossa.
- Lisäksi jos yhteenvetotiedot muodostetaan useamman taulun perusteella, liitetään vaatimuksiin edellä käsittelyt vaatimukset liitoksia sisältäviin koosteisiin liittyen.

Yhteenvetotietoja sisältävät koosteet (Hobbs *& al.*, 2005; Oracle, 2005; Powell, 2005):

- Sallittuja laskennassa käytettäviä funktioita ovat COUNT, SUM, AVG, VARIANCE, STDDEV, MIN ja MAX.
- Jokainen funktio, kuten SUM(<sarake>), tarvitsee SELECT -lohkoon liitettäväksi myös COUNT(<sarake>) - sekä COUNT(*) -funktiot.
- Käytettäessä STDDEV ja VARIANCE -funktioita, tarvitaan SELECT -lohkoon myös COUNT(<sarake>) -, SUM(<sarake>) - ja SUM(<sarake>*<sarake>) -funktiot.
- Kaikki GROUP BY -lohkossa ryhmittelytekijöiksi määrätyt sarakkeet on oltava mukana SELECT -lohkossa.
- Koosteen lähtötietoja ylläpitäville tauluille luotaviin lokeihin on muutosten yksilöinnin määrittäväksi optioksi valittava ROWID sekä sisällytettävä mukaan optiot SEQUENCE ja INCLUDE NEW VALUES.
- Lisäksi jos yhteenvetotiedot muodostetaan useamman taulun perusteella, liitetään vaatimuksiin edellä käsittelyt vaatimukset liitoksia sisältäviin koosteisiin liittyen.

Muita huomioitavia tilanteita (Hobbs *& al.*, 2005; Oracle, 2005; Powell, 2005):

- Jos koosteen lähtötietoja ylläpitäville tauluille on suoritettu tietojen päivittämisen yhteydessä partitiointiin liittyviä toimenpiteitä, ei inkrementaalinen päivitys

ole materialisoiduille näkymille tarkoitettujan lokeja käyttäen mahdollista ennen kuin kooste on kertaalleen päivitetty optiolla COMPLETE REFRESH. Näin käy esimerkiksi tilanteissa, että päivityksen yhteydessä lisätään uusi partitio tai poistetaan vanha. Ratkaisuna ongelmaan voi koosteesta riippuen olla Partition Change Tracking -päivitys.

- Jos koosteessa käytetään materialisoiduille näkymille tarkoitettujen lokien kannalta kiellettyjä analysointiin tarkoitettuja funktioita, kuten RANK -funktiota, voidaan materialisoitua näkymää koosteesta riippuen päivittää inkrementaalisesti Partition Change Tracking -päivityksen avulla.