Ville Pirttimäki

Relaatiotietokantojen indeksointi

Tietotekniikan
kandidaatintutkielma
7. joulukuuta 2010

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Ville Pirttimäki
Yhteystiedot: ville.pirttimaki@jyu.fi
Työn nimi: Relaatiotietokantojen indeksointi
Title in English: Indexes in Relational Databases
Työ: Tietotekniikan kandidaatintutkielma
Sivumäärä: 24
Tiivistelmä: Datan määrän kasvaessa monet tietokannat ja näiden taulut ovat kasvaneet rivimäärältään eksponentiaalisesti. Datan haku suuria tauluja läpikäymällä on kaikki rivit lukevaa perinteistä menetelmää käyttäen tehotonta ja aikaa vievää. Tietokantaindeksit pyrkivät ratkaisemaan kyseisen ongelman tallentamalla hakujen kannalta oleellista dataa. Tutkielma tarkastelee indeksoinnin perusmenetelmiä ja sopivan indeksointitavan valintaa.
English abstract: As the amount of data increases, the size of tables has increased exponentially in many databases. Searching for data from the tables using the traditional method of going through all the lines one by one is ineffective and time consuming. Database indexes aim to solve the problem by using alternative methods of storing data that is essential for the searches. The thesis presents the basic methods of indexing and advices on creating suitable indexes.
Avainsanat: B-puu, bittikarttaindeksi, indeksi, indeksointi, hajasipaisu, ryvästävä indeksi, QUBE, relaatiotietokanta, sarjasipaisu, sipaisu, yksikäsitteinen indeksi.
Keywords: B-tree, bitmap index, clustering index, index, indexing, QUBE, random touch, relational database, sequential touch, touch, unique index.

Sisältö

1 Johdanto

Hakuaika on yksi runsaasti dataa sisältävien tietokantojen monista haasteista. Triviaali tapa toteuttaa haku on lukea käsiteltävän taulun tai tauluyhdistelmän jokainen rivi ja hyväksyä tai hylätä kyseisiä rivejä haun mukaan. Tämä saattaa kuitenkin olla liian aikaa vievä tapa. Esimerkiksi 600 Mt levytilaa vievässä taulussa jo pelkkä levyltälukuaika on 15 sekuntia, jos oletetaan keskimääräisen levylukunopeuden olevan lähteen [Lahdenmäki, s. 49] mainitsema 40 Mt/s. Tämä saattaa ylittää haulta toivotun vasteajan moninkertaisesti. Yksi keino hakuajan lyhentämiseksi ovat indeksit.

Indeksit ovat tietorakenteita, jotka kopioivat tietokannan taulujen sisältämän datan muotoon, josta tiettyjen hakujen on nopeampaa löytää tarvittava tieto. Esimerkkinä tällaisesta tietorakenteesta on hakupuu, josta tietyn arvon löytäminen onnistuu huomattavasti lyhyemmässä ajassa kuin järjestämättömästä listasta, johon indeksoimaton tietokantataulu on verrattavissa. Haittana indeksien käytössä on lisääntynyt tallennustilan tarve. Koska indeksit eivät yleensä korvaa indeksoitua tietokantataulua, vaan tallentuvat alkuperäisen tietokantataulun rinnalle, moninkertaistavat indeksit kunkin indeksoidun taulun vaatiman tallennustilan. Myös tietoja muokkaavien ja lisäävien toimintojen suoritusajat kasvavat, koska yksittäinen lisäys tai muutos vaatii tietokannanhallintajärjestelmältä toimenpiteitä eri indeksejä varten.

Tutkielma on suunnattu tietokantojen perusperiaatteet tunteville ihmisille, kuten tietokantojen suunnittelijoille, jotka haluavat saada yleiskuvan indeksoinnista. Luku 2 käsittelee indeksin luomiseen ja käyttämiseen liittyviä perusperiaatteita. Luku 3 tarkastelee yleisimmin käytössä olevia indeksoinnin tekniikoita ja tietorakenteita. Luvussa 4 esitellään keinoja tunnistaa indeksointia vaativia tauluja. Luku 5 esittelee käsitteitä ja menetelmiä, joita hyödyntäen voidaan arvioida haun kestoa ja täten myös indeksin tehokkuutta. Luku 6 esittelee menetelmiä hakuun sopivan B-puuindeksin valintaan luvussa 5 esiteltyjä käsitteitä apuna käyttäen. Lopuksi luku 7 käy läpi syitä, miksi indeksien turhaa käyttöä tulisi välttää.

2 Indeksointiprosessi

Lähteen [Connolly, s. 284-285] mukaan tietokannan kehitysvaiheissa (eng. *database system development lifecycle*) on yksinkertaistettuna nähtävissä seuraavat vaiheet: esivalmistelu, vaatimussuunnittelu, tietokannan suunnittelu, toteutus, testaus ja ylläpito. Ne käydään läpi syklisesti tietokannan kehittyessä ja muuttuessa. Luvussa kuvataan indeksoinnin huomioimista eri kehitysvaiheissa, indeksien perustoimintaperiaatetta ja indeksin luomisen perusteita.

2.1 Indeksit tietokantojen kehitysvaiheissa

Mikäli indeksien käyttöä halutaan **suunnitella etukäteen**, luontevin ajakohta tälle suunnittelulle on lähteen [Connolly, s. 284 ja 508] mukaan tietokannan suunnitteluvaihe. Se on jaettavissa kolmeen erilliseen osaan: konseptuaaliseen, loogiseen ja fyysiseen suunnitteluun. Indeksointi on osana fyysistä suunnittelua, jossa päätetään aiemmissa vaiheissa suunniteltujen taulujen lopullinen toteutustapa. Tutkielmassa tätä kutsutaan **proaktiiviseksi** suunnitteluksi.

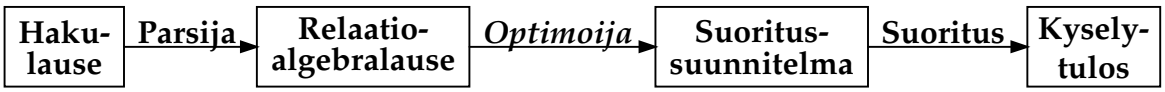
Fyysinen suunnitteluvaihe on lisäksi ensimmäinen tietokannan kehitysprosessin osa, jossa käytettävän tietokannanhallintajärjestelmän kiinnittäminen on pakollista. Indeksien suunnittelua ei ole mielekästä tehdä ennen tätä vaihetta, koska ei ole välttämättä tiedossa, mitä indeksointitapoja käytettävä tietokannanhallintajärjestelmä tulee tukemaan.

Indeksien käyttöä ei aina suunnitella ennalta. Kuten lähteessä [Pratt] kuvataan, indeksiä ei ole pakko luoda yhtä aikaa siihen liittyvän taulun kanssa, vaan indeksejä voidaan tarvittaessa luoda lisää **tietokannan käyttöönotton jälkeen**. Näin indekseistä tulee myös osa testaus- ja ylläpitovaiheita, kun indeksejä luodaan korjaamaan yllättävän suuria tai ajan myötä kasvaneita hakuaikoja. Tähän viitataan luvuissa 2.1 ja 5 **reaktiivisena** indeksien suunnitteluna.

2.2 Indeksit hakujen optimoinnissa

Indeksien luonnista huolehtii yleensä tietokannan ylläpitäjä, mutta varsinaisesta käytöstä vastaa tietokannanhallintajärjestelmän **optimointiohjelma** (engl. *query optimizer*) lähteen [Kifer, luku 11.1] kuvaaman kuvan 1 prosessin mukaisesti. Kun tietokannanhallintajärjestelmälle syötetään SQL-muotoinen kysely, se parsitaan ensin relaatioalgebramuotoiseksi lauseeksi, joka syötetään optimointiohjelmalle. Tämä taas luo kyselyn mahdollisia **suoritussuunnitelmia** (engl. *query execution plan*) käytettävissä olevien resurssien perusteella ja valitsee näistä sen, jonka suoritusa-

jan se arvioi lyhyimmäksi. Päättelyssä yksi käytettävistä resursseista ovat olemassa olevat indeksit.



Kuva 1: Optimoijan sijainti haun suorituksessa.

Jotkin tietokannanhallintajärjestelmät luovat lähteen [Hovi, s. 158] mukaan automaattisesti yksikäsitteisen indeksin perusavaimelle[1], mikäli tämä on määrätty. Muutoin indeksien luonti on tähän oikeutettujen tietokannan ylläpitäjien hallinnassa.

2.3 Indeksien luominen

Kuten [Connolly, luku 6.3.5] mainitsee, indeksien luontiin ja hallintaan ei ole komentoja SQL-standardissa, vaan jokainen indeksien käyttöä tukeva tietokannanhallintajärjestelmä on kehittänyt omat lisäyksensä SQL-standardiin. Lisätty syntaksi vaihtelee sen mukaan, mitä indeksointitekniikoita kukin tietokannanhallintajärjestelmä tarjoaa. Yleensä **indeksin luova SQL-komento** on lähteen [Date, luku 5.3] mukaan syntaksiltaan seuraavan kaltainen:

```
CREATE [UNIQUE] INDEX index_name ON base_table
(column[order][, column [order]...]) [CLUSTERED]
```

Komennossa `base_table`-tauluun määritellään `index_name`-niminen indeksi luettelemalla indeksoitavat sarakkeet, sekä sarakkeiden järjestäminen nousevaan tai laskevaan järjestykseen. `CLUSTERED`-avainsana luo ryvästävän indeksin, kun taas `UNIQUE`-avainsanaa käytetään yksikäsitteisen indeksin luomiseen (tarkemmat selitykset termeille luvussa 3.4). Käskyn suorittamiseksi käyttäjällä on oltava tarvittavat oikeudet tietokannanhallintajärjestelmään.

Esimerkiksi nimiä ja osoitteita sisältävän tietokantataulun indeksointi tapahtuu seuraavan kaltaisella SQL-komennolla

```
CREATE INDEX indeksi1 ON
yhteystiedot(sukunimi, etunimi, kaupunki) CLUSTERED
```

Komento luo `indeksi1`-nimisen ryvästävän indeksin, joka on järjestetty sukunimen, etunimen ja kaupungin mukaan kyseisessä järjestyksessä.

[1]Lähteen [Date, s. 725] mukaan perusavaimet indeksoivaa indeksiä kutsutaan kirjallisuudessa usein nimellä *primary index*, kun taas muita indeksejä kutsutaan nimellä *secondary index*.

3 Indeksointitekniikoita

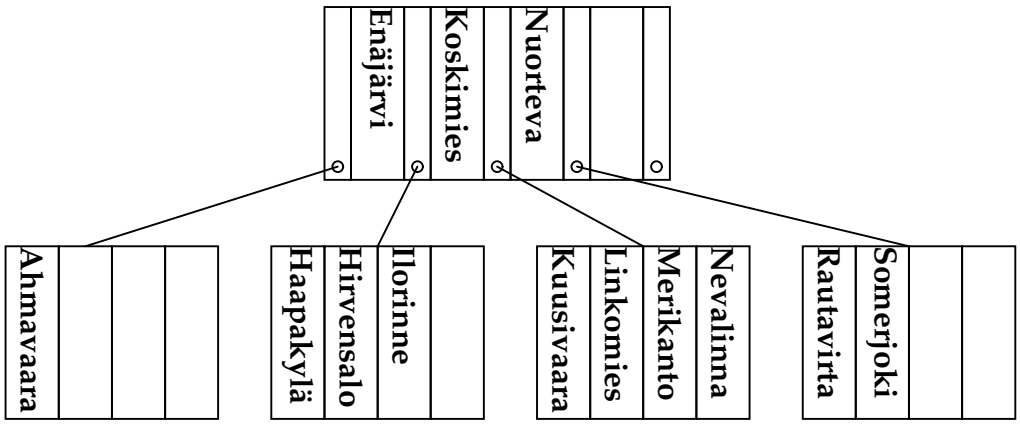
Indeksi voidaan toteuttaa tietokannanhallintajärjestelmässä käyttäen useita erilaisia tietorakenteita. Käytössä olevat tekniikat vaihtelevat tietokannanhallintajärjestelmän mukaan. SQL Serverin [Microsoft], MySQL:n [Sun], Oraclen [Oracle] ja PostgreSQL:n [PostgreSQL 1] tarjoamia tekniikoita tarkasteltaessa huomataan B-puuindeksoinnin olevan laajimmalti käytössä oleva tekniikka, jota kaikki edellä mainitut tukevat ja oletusarvoisesti käyttävät. Oracle tukee myös bittikarttaindeksointia sekä PostgreSQL ja MySQL tukevat hajautusindeksiä.

Näiden lisäksi on olemassa tekniikoita erityisten tietotyyppien indeksointiin. PostgreSQL pystyy indeksoimaan lähteen [PostgreSQL 2] mukaan **tekstin täyshakua** (engl. *full text search*) GiST- ja GIN-tekniikoita käyttäen. SQL Server [Microsoft] pystyy indeksoimaan XML-dataa sekä SQL Server ja MySQL [Sun] tukevat spatiaalisen (eli kuvan sijaintia kaksiulotteisessa avaruudessa käsittelevän) datan indeksointia.

3.1 B-puuindeksit

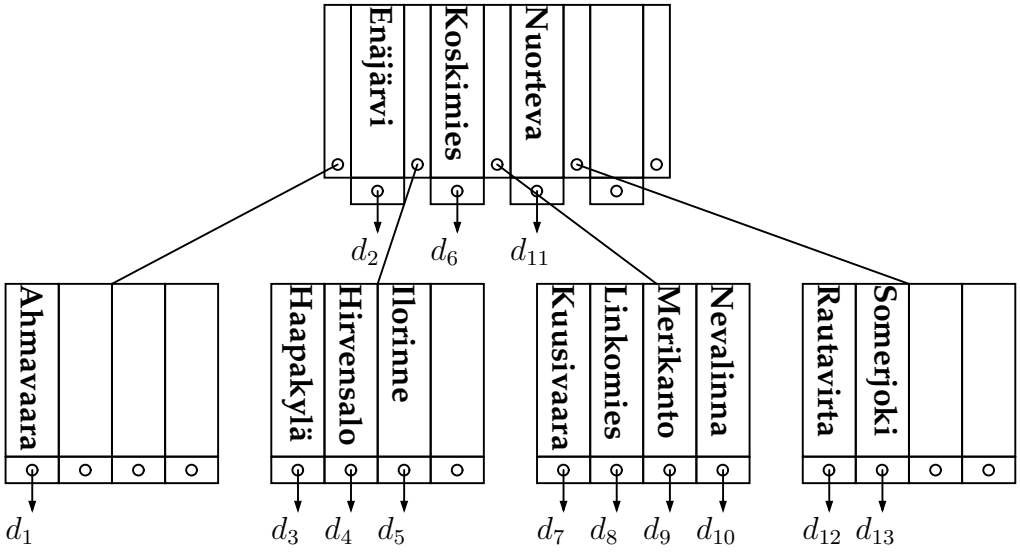
Elmasrin [Elmasri, luku 6.3.1] mukaan **hakupuu** (engl. *search tree*) on puumuotoinen tietorakenne, joka on suunniteltu erityisesti yksittäisen alkion hakemiseen suuresta määrästä alkioita. Yksinkertaisesti määriteltynä hakupuu koostuu kuvassa 2 esitetyn mukaisista solmuista, joista jokainen sisältää enintään n kappaletta viitteitä alemman tason solmuihin. Jokaisessa solmussa on enintään $n-1$ kappaletta hakuarvoja sijoiteltuna siten, että jokainen viitteiden väli sisältää enintään yhden hakuarvon. Poikkeuksena lehtisolmut sisältävät pelkkiä hakuarvoja. Hakuarvot ja viitteet on järjestetty siten, että jokaisessa solmussa hakuarvot ovat järjestyksessä. Lisäksi kahden hakuarvon välissä oleva viite osoittaa siihen alipuuhun, joka sisältää kaikki näiden arvojen välissä olevat hakuarvot.

Näin rakennetusta puusta tietyn alkion (tai tästä seuraavan alkion) hakeminen onnistuu vertaamalla haetun alkion arvoa juurisolmun kuhunkin hakuarvoon. Mikäli haettua arvoa ei löydetä tästä solmusta, voidaan hakuarvovälien perusteella helposti päätellä, mistä solmun alipuusta haettu arvo tulisi löytyä. Prosessi toistetaan tämän alipuun juurisolmuun, kunnes alkio löydetään tai solmulla ei enää ole alipuita. Koko hakuoperaatio on suoritettavissa logaritmisessa ajassa, mikäli puun solmut ovat tasapainossa, eli kullakin puun tasolla alipuut ovat suunnilleen saman kokoisia.



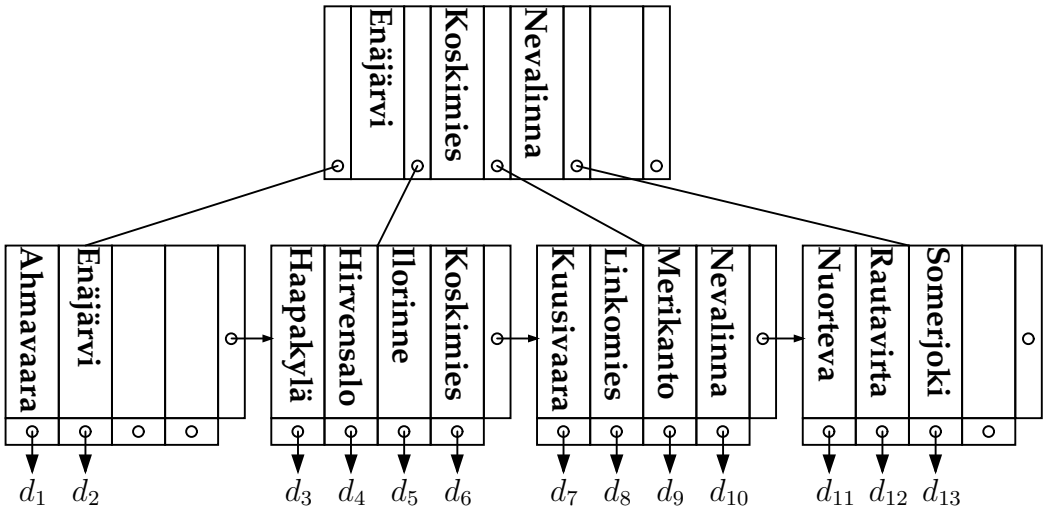
Kuva 2: Esimerkki sukunimillä täytetystä hakupuusta.

B-puu (engl. *B-tree*) on eräs hakupuun variaatio, ja siitä esimerkki on esitelty kuvassa 3. Tavallinen hakupuu ei takaa puun alkioiden olevan tasapainossa, kun B-puuhun liittyvät alkioiden lisäys- ja poistoalgoritmit takaavat puun jokaisen lehtisolmun olevan aina samalla tasolla. Lisäksi jokaisella solmulla, lukuunottamatta juuri- ja lehtisolmuja, on vähintään $\lceil n/2 \rceil$ lasta. Tämä tieto tai itse järjestysalgoritmit eivät kuitenkaan ole indeksoinnin kannalta mielenkiintoisia. Mielenkiintoisin ero tavalliseen hakupuuhun B-puussa on se, että B-puun solmut eivät sisällä pelkkiä hakuarvoja, vaan hakuarvojen (indeksien tapauksessa indeksirivit) ja dataviitteiden (indeksien tapauksessa viitteet indeksiriviä vastaavaan tietokantataulun riviin) muodostamia pareja. Huomaa ero lehtisolmujen ja tavallisten solmujen välillä. [Elmasri, luku 6.3.1]



Kuva 3: Kuvan 2 hakupuu B-puun muodossa.

Tarkasteltaessa kirjallisuuden kuvauksia tietokannanhallintajärjestelmien B-puiden rakenteista, huomataan B-puurakenteen variaatioiden vastaavan suurelta osin **B+-puuta** (engl. *B+-tree*). B+-puu on B-puurakenteen variaatio, jossa lehtisolmuihin kuulumattomat solmut sisältävät vain hakuarvoja ja viitteitä alemman tason solmuihin (katso kuva 4). Dataviitteet löytyvät vasta lehtisolmuista asiaankuuluvien hakuarvojen ohesta. Toisin kuin B-puussa tai hakupuussa, kyseisten solmujen hakuarvot toistetaan myös lehtisolmuissa siten, että hakuarvo sisällytetään alipuuhun, johon tätä edeltävä viite viittaa. Lisäksi, vaikka B+-puun määritelmä ei sitä vaadi, linkitetään monissa B+-puuvariaatioissa jokainen lehtisolmu aina seuraavaan, jotta hakuarvojen ja näiden dataviitteiden sarjoja olisi helpompi käydä läpi. [Elmasri, luku 6.3.2]



Kuva 4: Kuvan 2 hakupuu B+-puun muodossa.

3.2 Hajautustauluindeksi

Hajautustauluindeksi (engl. *hash map index*) tai lähteen [Date, s. 733] mukaan vaihtoehtoisesti pelkkä hajautus (engl. *hashing*) muistuttaa jokseenkin puurakenteista indeksiä, mutta on suunniteltu huomattavasti rajoitetumpia käyttötapauksia varten. Kirja [Kifer, luku 9.6] esittelee kyseisen tietorakenteen eri tyyppeineen varsin tarkasti.

Lyhyesti sanottuna hajautustauluindeksi perustuu **hajautusfunktion** (engl. *hash function*) käyttöön, jonka avulla annettu avainarvo voidaan yksiselitteisesti muuttaa numeeriseksi arvoksi tietyltä väliltä (jonka pituus on samalla myös hajautustaulun paikkojen määrä). Kun indeksoitavaan tietokantatauluun lisätään rivi, avainarvosta lasketaan hajautusfunktion avulla arvo, joka määrää indeksoitavan rivin paikan

hajautustaulusta. Vastaavasti, kun indeksistä haetaan riviä indeksoidun sarakkeen avulla, saadaan sen paikka hajautustaulusta antamalla hajautusfunktiolle parametriksi hakuarvo.

Hajautustauluindeksiin tehdyn haun täysin lineaarisen suoritusajan (olettaen hajautusfunktion suoritusajan olevan lineaarinen) estävät ainoastaan **törmäykset**. Hajautusfunktio voi nimittäin tuottaa saman tuloksen usealle avainarvolle, jolloin samaan hajautustaulun paikkaan joudutaan sijoittamaan useampi arvo. Tällöin myös hajautustauluindeksiä luettaessa on käytävä läpi kaikki saman paikan arvot.

Hajautusfunktion toteutukseen taikka hajautustaulun dynaamiseen koon muuttamiseen ei tutkielmassa puututa, sillä sitä on kuvattu hyvin lähteessä [Kifer]. Mainittakoon, ettei periaatteessa mikään estä funktion toteutusta rajaamattomalle määrälle parametrejä, eli indeksoinnin tapauksessa sarakkeita. Käytännössä kuitenkin hajautustauluindeksointi on toteutettavissa vain yhdelle sarakkeelle kerralla, kuten esimerkiksi PostgreSQL:n osalta korostetaan lähteessä [PostgreSQL 3].

3.3 Bittikarttaindeksi

Mikäli taulun jonkin sarakkeen kardinaliteetti on alhainen (ts. sarake voi saada vain rajoitetun joukon arvoja), voidaan saraketta koskevia kyselyitä varten luoda bittikarttaindeksi (engl. *bitmap index*). Lähteen [Hovi, s. 154-155] mukaan se on **kaksiulotteinen taulukko bittejä** (katso taulukko 1), jonka jokainen rivi vastaa yhtä taulun sarakkeen mahdollisista arvoista ja jokainen sarake vastaa yhtä tietokantataulun riviä (tai päinvastoin). Tämän indeksin avulla saadaan nopeasti binäärioperaatioita käyttäen selville jokainen tietokantataulun rivi, jonka indeksoidun sarakkeen arvo joko täyttää tietyn ehdon tai on tietyn rivin indeksoidun sarakkeen arvo. Lähteen [Elmasri, s. 849] mukaan bittikarttaindeksin etuna on se, että suurenkin rivimäärän indeksointi bittikarttaindeksoinnilla vie vain murto-osan itse tietokantataulun viemästä tilasta.

	ID					
Etunimi	**1**	**2**	**3**	**4**	**5**	**6**
Onni	1	0	0	0	0	1
Eetu	0	1	0	0	1	0
Veeti	0	0	1	1	0	0
Elias	0	0	0	0	0	0

Taulukko 1: Esimerkki bittikarttaindeksistä.

Lähteen [Lahdenmäki, s. 25-26] mukaan bittikarttaindeksit soveltuvat hakujen **suorituskyvyn parantamiseen** vain tietyissä käyttötapauksia, kuten esimerkiksi tietyt ehdot täyttäviä rivejä laskiessa. Yleesä bittikarttaindeksillä ei pystytä parantamaan suorituskykyä johtuen bittikarttaindeksin rajoitetusta tietosisällöstä. Mikäli haku tarvitsee dataa varsinaisesta tietokantataulusta (esimerkiksi jonkin sarakkeen arvon), joudutaan suorittamaan rivin haku levyltä, joka myöhemmin luvussa 5.1 todetaan hyvin aikaa vieväksi operaatioksi.

Myös **useamman sarakkeen sisällyttäminen** samaan bittikarttaindeksiin on mahdollista. Tämä voidaan toteuttaa esimerkiksi listaamalla ensimmäisen sarakkeen mahdolliset arvot bittikarttaindeksin ensimmäisiin riveihin, toisen sarakkeen arvot tämän jälkeisiin riveihin ja niin edespäin, pitäen samalla kirjaa, minkä sarakkeen mahdollista arvoa kukin rivi kuvaa. [Hovi, s. 154-155]

3.4 Ryvästävä indeksi ja yksilöivä indeksi

Luvuissa 3.1-3.3 kuvatut tekniikat ovat erilaisia indeksejä toteuttavia tietorakenteita. Sen sijaan ryvästävä (engl. *clustering*) ja yksilöivä (engl. *unique*) indeksi eivät niinkään määrää indeksiin tallennetun tiedon rakennetta. Kyseiset indeksityypit siis eroavat toiminnaltaan perusindekseistä ja ovat näinollen ennemminkin luvuissa 3.1-3.2 kuvattujen indeksityyppien erikoistapauksia. Ryvästävä ja yksilöivä indeksi vaativat kuitenkin tietorakenteen pitävän yllä jonkinlaista viittausten yksikäsitteistä järjestystä, joten esimerkiksi luvussa 3.3 kuvattua bittikarttaindeksiä ei voi määrätä ryvästäväksi taikka yksilöiväksi.

Jos tietokannan taululle on määrätty **ryvästävä indeksi**, taulun rivit järjestyvät levyllä indeksin määräämään järjestykseen. Yleensä tämä toteutetaan (etenkin B-puuindeksin tapauksessa) korvaamalla tietokantataulun normaali tietorakenne indeksin tietorakenteella. Koska ryvästävä indeksi vaikuttaa tietokannan fyysiseen rakenteeseen, voi kullakin taululla olla vain yksi ryvästävä indeksi kerrallaan. Mikäli ryvästävää indeksiä ei olla erikseen määritelty, luovat useimmat tietokannanhallintajärjestelmät taululle ryvästävän indeksin taulun avainsarakkeista. Käytännössä ryvästävä indeksi vaikuttaa vain hakuaikoihin ja tilankäyttöön. [Elmasri, luku 6.1.2]

Lähteen [Hovi, s. 158] mukaan **yksilöivä indeksi** varmistaa, ettei indeksin riveihin lisätä samat arvot sisältävää duplikaattia. Samalla estetään myös vastaavan rivin lisääminen kohdetauluun. Tämän tyyppinen indeksi auttaa varmistamaan tietokannan eheyden siten, että tietyn sarakeyhdistelmän sisältämät tiedot eivät toistu missään taulun riveistä. Lisäksi indeksin yksilöivyys toimii lisäinformaationa luvussa 2.2 kuvatulle optimoijalle sen päättäessä haun suoritustavasta. Otetaan esimerkiksi

taulu, jolla on on yksilöivä indeksi sarakkeista `nimi` ja `ika`. Jos taulussa on jo rivi, jossa `nimi=Nuorteva`, `ika=27` ja `kaupunki=Helsinki`, tauluun ei pysty lisäämään riviä, jossa `nimi=Nuorteva`, `ika=27` ja `kaupunki=Jyväskylä`.

4 Indeksoitavan taulun valinta

Kuten luvussa 2.2 todettiin, indeksejä hyödyntää tietokannanhallintajärjestelmän optimoija etsiessään mahdollisimman nopeaa tapaa suorittaa sille annettu haku. Tästä johtuen indeksin hyödyllisyys riippuu sitä käyttävistä hauista. Silti indeksiä ei määritellä haulle, vaan yksittäisille tauluille.[2] Indeksoitava taulu on tunnistettava, ennen kuin indeksi voidaan luoda.

Luvussa tarkastellaan yksinkertaisia tapoja indeksoitavan taulun tunnistamiseksi. Teoreettiseen haun analysointiin liittyviä käsitteitä ja menetelmiä kuvataan tarkemmin luvussa 5. Itse indeksin valintaan palataan luvussa 6.

4.1 Haun analysointi

Triviaalissa tapauksessa kysely käsittelee vain yhtä taulua, jolloin indeksi on määritettävä tälle taululle olettaen, että indeksoinnin tarve on todettu. Jos kysely kohdistuu useampaan tauluun, tulee indeksoitava tietokantataulu tai -taulut tunnistaa haun käsittelemistä tauluista. Tätä varten on analysoitava haun eteneminen tietokannanhallintajärjestelmässä ja arvioitava kuhunkin tauluun liittyvien toimenpiteiden pituutta.

Mikäli indeksointia suoritetaan **reaktiivisesti** (katso luku 2.1) olemassa olevaan tietokantaan, voidaan haun suoritustapaa arvioida `EXPLAIN`-komennon avulla. `EXPLAIN` on SQL-standardiin kuulumaton komento, jonka monet tietokannanhallintajärjestelmät kuitenkin tavalla tai toisella toteuttavat. Komento kirjoitetaan analysoitavan haun eteen. Optimointiohjelma komennon saatuaan listaa sen, kuinka tietokannanhallintajärjestelmä kyselyn suorittaisi. Komennon avulla voidaan arvioida kyselyn toteuttamiseen käytettäviä olemassaolevia indeksejä ja järjestysoperaatioita sekä haku- ja liitosoperaatioiden järjestyksiä. [Date, s. 770]

4.2 Suuret taulut

Luvussa 4.1 käytiin läpi tapausta, jossa tunnettiin vaadittua hitaammin toimiva haku ja määriteltiin sen avulla mahdollisesti indeksoitavat tietokantataulut. Tämän lisäksi on mahdollista löytää tauluja, joiden nykyinen tai mahdollisesti toteutuva rivimäärä voi johtaa kyseisiin tauluihin liittyvien hakujen hitaaseen suoritukseen.

Mikäli tietokantataulun haun suorittamiseen ei ole olemassa sopivaa indeksiä, on mahdollista, että tietokannanhallintajärjestelmä joutuu käymään läpi kaikki tie-

[2] Jotkin tietokannanhallintajärjestelmät sallivat haun tuloksen indeksoinnin näkymän muodossa, mutta tätäkin käsitellään indeksin kannalta kuin tavallista taulua.

tokannan rivit suorittaessaan haun. Luvussa 5.1 tullaan toteamaan, että perättäisten taulurivien läpikäyminen tulee viemään noin 0.01 ms kutakin riviä kohden. **Keskimääräistä rivin hakuaikaa** hyödyntäen voidaan helposti tunnistaa etenkin indeksoimattomat taulut, joihin kohdistuvat haut voivat ylittää hakuajalle annetun rajan. Oletetaan esimerkiksi, että jokaisen tietokantaan suoritetun haun halutaan vievän alle sekunnin. Käyttäen edellä mainittua rivikohtaista käsittelyaikaa, voidaan helposti laskea yli 100 000 riviä sisältävien ja indeksoimattomien taulujen vaativan yli sekunnin kestävän läpiluvun.

Pelkän tietokantataulun tunnistaminen ei kuitenkaan riitä indeksien luomiseen, ellei kyseessä ole yksi- tai kaksisarakkeinen taulu. **Kaikkien mahdollisten indeksien luominen taululle ei ole mielekästä**, sillä mahdollisten indeksien määrä kasvaa eksponentiaalisesti suhteessa taulun sarakkeiden määrään. Tämän takia tulee ensin tunnistaa tauluun kohdistuvat haut, ja johtaa näistä tarvittavat indeksit esimerkiksi luvussa 6 käsiteltävillä menetelmillä.

Lisäksi on otettava huomioon, että annettu käsittelyaika on vain arvio. Esimerkiksi lähteen [Hovi, s. 158] mukaan jotkin tietokannanhallintajärjestelmät luovat automaattisesti indeksoimattomille tauluille taulun perusavaimista koostetun indeksin, mikä voi ehkäistä pessimistisen arvion toteutumisen käytännössä.

5 Haun keston arviointi

Luvussa 4.1 käsiteltiin reaktiivista indeksointia, jossa indeksoinnilla pyrittiin ratkaisemaan tietokannan käytössä esiintyvää hitautta. Sen teho on helppo todeta yksinkertaisesti tarkistamalla, toimiiko indeksointia vaativa haku indeksoinnin jälkeen halutussa ajassa.

Tämän etenemistavan vastakohtana on luvussa 2.1 mainittu proaktiivinen indeksointi, jonka avulla indeksejä pyritään luomaan ennen kuin ongelmia ilmenee, mahdollisesti jo tietokannan suunnitteluvaiheessa. Tällöin ei pahimmassa tapauksessa indeksoitavaa tietokantaa ole vielä edes luotu, joten indeksoinnin teho täytyy pystyä todentamaan jollain muulla tavalla. Tätä varten hakuun kuluvaa aikaa on pystyttävä arvioimaan hakua suorittamatta.

5.1 Sipaisut ja hajaluku

Sipaisu (engl. *touch*) määritellään lähteessä [Hovi, s. 345] abstraktiksi tapahtumaksi, jossa tietokannanhallintajärjestelmä tutkii tietokannasta yhden rivin. Sipaisuiden käsitteen avulla voidaan abstrahoida todellisuudessa haun suoritukseen liittyvien operaatioiden joukkoa, ja näin pystytään helpommin arvioimaan tietyn hakureitin aiheuttamien levylukuoperaatioiden kestoa. **Hajasipaisu** (engl. *random touch*) vastaa yksittäisen tietokantarivin hakua, kun rivin sijainti levyllä tunnetaan. Esimerkiksi, kun indeksi ei sisällä kaikkia haun sisältämiä taulusarakkeita, aiheuttaa minkä tahansa rivin haku hajasipaisun tietokannanhallintajärjestelmän joutuessa lukemaan indeksin lisäksi varsinaisen tietokantataulun rivin. **Sarjasipaisu** (engl. *sequential touch*) vastaa keskimääräistä aikaa yhden rivin hakuun luettaessa useita perättäisiä rivejä samalla kertaa olettaen, että sarjan ensimmäinen rivi on jo noudettu hajahipaisulla.

Kun tietokannasta haetaan yksittäistä riviä, hakee tietokannanhallintajärjestelmä fyysiseltä levyltä koko rivin sisältävän tietokantasivun[3]. Yhden tietokantasivun lukua yhden rivin hakemiseksi kutsutaan lähteen [Hovi, s. 337] mukaan **hajaluvuksi** (engl. *random read*). Se on pitkäkestoisin operaatio luettaessa tietoa levyltä ja soveltuu näin hajasipaisun pessimistiseksi aika-arvioksi.

Arvioitaessa **hajalukuun kuluvaa aikaa**, tulee ottaa huomioon mahdollisuus, että levy on jo käytössä ja kyseinen levyluku joutuu odottamaan vuoroaan. Oletetaan lähteen [Lahdenmäki, s. 14] perusteella, että levyyn kohdistuu 50 lukua sekunnissa

[3]Tietokantasivu on pienin yksikkö, jonka tietokannanhallintajärjestelmä lukee levyltä kerralla. Tyypillinen koko 4 tai 8 Kt [Hovi, s. 147].

ja että yhteen lukuun kuluu 6ms. Keskimääräinen jonotusaika Q saadaan kaavalla

$$Q = \frac{uS}{1-u},$$

jossa S on keskimääräinen lukuaika ja u on osuus ajasta, jolloin levy ei ole vapaa. Esimerkiksi $u = 50\ lukua/s * 0.006\ s/luku = 0.3$ antaa $Q = \frac{0.3*6\ ms}{1-0.3} \approx 3\ ms$. Kun tämä lisätään edellä mainittuun 6ms:n levyltälukuaikaan ja lisätään tähän summaan vielä 1ms siirtoaika fyysiseltä levyltä tietokannanhallintajärjestelmään saadaan yhteisajaksi 10ms. Tätä tullaan tutkielmasta tästä lähtien käyttämään hajalukuun kuluvan ajan arviona.

Lähteen [Lahdenmäki, s. 15] mukaan monissa nykyisissä kovalevyjärjestelmissä on oma sisäänrakennettu välimuisti, johon haettava tietokantasivu on saattanut tallentua jossain aiemmassa luvussa. Mikäli voidaan olettaa haettavan sivun tallentuneen välimuistiin esimerkiksi silloin, kun samaa sivua on haettu vasta äskettäin, voidaan kovalevylukuoperaation pituudeksi olettaa noin 1ms.

Arvioitaessa yhteen **sarjasipaisuun kuluvaa aikaa**, levylukuun kuluvan ajan arviointi ei ole enää mielekästä. Koska tietokantarivejä luetaan kovalevyiltä tietokantasivu kerrallaan, yksittäisen rivin lukemiseen kuluvan ajan arviointi ei ole kovinkaan yksiselitteistä. Olettaen, että tietokantasivun koko on 4Kt ja yhden sivun lukuaika on edellä laskettu 10ms, saadaan keskimääräiseksi perättäisen datan lukunopeudeksi 400Kt/s. Mutta johtuen tietokantarivin tauluittain vaihtelevasta koosta, keskimääräisestä lukunopeudesta ei pystytä päättelemään rivikohtaista lukuaikaa. Lisäksi perättäisiä tietokantasivuja luettaessa voidaan sekä limittää levylukua ja prosessointia että lukea useampia sivuja kerralla, joka voi lähteen [Lahdenmäki, s. 16] mukaan nostaa lukunopeuden jopa 40Mt/s. Niinpä on mielekkäämpää arvioida sarjasipaisua yksittäisen rivin hyväksymiseen kuluvan prosessointiajan avulla, joka voidaan arvioida lähteen [Lahdenmäki, s. 70] mukaan noin 0.01ms:n pituiseksi.

5.2 Läpäisykerroin ja indeksisiivu

Predikaatiksi määritellään hakuehto, jolla on kiinteästi määrätty kohdesarake, mutta haettava arvo voi olla hausta riippuen joko kiinteä arvo (esim. `kaupunki = 'Helsinki'`) tai muuttuja (esim. `kaupunki = :kaupunki`). Lähteen [Lahdenmäki, s. 37] mukaan predikaatin **läpäisykerroin** (engl. *filter factor*) kertoo, kuinka suuri osa tietokantataulun riveistä täyttävät predikaatin hakuehdon, ja se ilmaistaan reaalilukuna arvojen 0 (ei hakuehdon täyttäviä rivejä) ja 1 (kaikki rivit täyttävät hakuehdon) väliltä tai prosenttina. Läpäisykertoimen arvot ovat yleen-

sä arvioita, sillä tarkkojen läpäisykertoimien seuraaminen yhtä tai useampaa arvoa kohden alati muuttuvassa tietokannassa ei ole mielekästä.

Keskimääräinen läpäisykerroin on yksi mahdollinen tapa ilmaista läpäisykerroin muuttujan sisältävälle predikaatille. Mielekkäin se on yhtäsuuruuspredikaatin tapauksessa, jolloin läpäisykerroin on predikaatin kohdesarakkeen kardinaliteetin käänteisluku. Lisäksi predikaatille on olemassa arvokohtaisia läpäisykertoimia, joista pessimistisessä suoritusaikojen arvioinneissa käytetään yleensä suurinta.

Läpäisykerroin lasketaan **useamman predikaatin yhdistelmälle** täsmälleen samalla tavalla kuin yksittäiselle predikaatille, mutta läpäisykertoimen laskentaan vaikuttavien rivien tulee täyttää koko predikaattiryhmän totuusehto. Yhden tai useamman predikaatin täyttäminen riippuu siitä, onko predikaatit liitetty toisiinsa `AND`, `OR` vai jollakin muulla konjuktiolla. Lisäksi predikaattien välinen korrelaatio vaikuttaa näiden yhdistelmän suuruuteen. [Lahdenmäki, s. 37-39]

Olkoon esimerkiksi taulussa miljoona riviä sekä sarakkeet `sukunimi`, `kaupunki` ja `laani`. Mikäli predikaatin `sukunimi = 'Kuusivaara'` läpäisykerroin on 1%, rajaa se taulusta $1000000 * 0,01 = 10000$ riviä tulokseen. Jos predikaatin `kaupunki = 'Helsinki'` läpäisykerroin on 5%, yhdistetyn predikaatin `sukunimi = 'Kuusivaara' AND kaupunki = 'Helsinki'` läpäisykerroin on $1\% * 5\% = 0,02\%$, joka rajaa täten taulusta $1000000 * 0,0002 = 200$ riviä. Tämä johtuu näiden kahden predikaatin välisen korrelaation puutteesta. Mikäli taas yhdistetään vahvan korrelaation omaavat predikaatit `kaupunki = 'Helsinki'` ja `laani = 'Uusimaa'` (jälkimmäisen läpäisykertoimen ollessa 10%) predikaatin `kaupunki = 'Helsinki' AND laani = 'Uusimaa'` läpäisykerroin on 10%, eikä $5\% * 10\% = 0,5\%$.

Läpäisykerroin on tärkeä työväline arvioitaessa hakuun kuuluvan **indeksisiivun** (engl. *index slice*) pituutta (ts. rivien määrää). Indeksisiivu kuvaa hakua varten indeksistä luettavia rivejä, ja sen pituus saadaan kertomalla seuraavaksi kuvattavan predikaattiyhdistelmän läpäisykerroin koko taulun rivien määrällä. Parhaimmassa tapauksessa indeksisiivun pituus on sama kuin haun kaikkien predikaattien yhdistelmän rajaama rivien määrä. Tämä vaatii kuitenkin sen, että kaikki predikaattiyhdistelmän toteuttavat rivit ovat indeksissä peräkkäin (ja olettaen, etteivät nämä predikaatit ole luvussa 6.3 käsiteltäviä hankalasti indeksoitavia predikaatteja). Mikäli tämä ei toteudu, tulee hausta tunnistaa predikaatti tai predikaattiyhdistelmä, joka rajaa taulusta tai sopivasta indeksistä yhtenäisen alueen peräkkäisiä rivejä. [Lahdenmäki, s. 39-40]

5.3 QUBE-menetelmä

Lahdenmäen ja Leachin kirja [Lahdenmäki, s. 65-75] esittelee QUBE-metodin (Quick Upper-Bound Estimate), jonka avulla voidaan luvuissa 5.1 ja 5.2 määriteltyjä käsitteitä apuna käyttäen arvioida hakuun kuluvaa aikaa tunnetun tai arvioidun hakupolun mukaan. Arvion laskemisen yksinkertaistamiseksi QUBE ei huomioi mm. suorituslukkoihin tai levylukuihin liityviä jonotusaikoja, vaan keskittyy sipaisuihin ja `FETCH`-käskyihin kuluvaan suoritusaikaan.

QUBE perustuu **kaavaan**

$$V = H * 10 + S * 0.01 + T * 0.1,$$

jossa V on vasteaika sekuntteina, H on hajasipaisujen määrä, S on sarjasipaisujen määrä ja T on tulosrivien määrä. Kaavassa on käytetty luvussa 5.1 kuvattuja **suoritusajan arvioita** sipaisuille sekä 0.1ms:n arviota kunkin tulosrivin käsittelyyn kuluvaksi ajaksi. Mikäli käytetyt arviot eivät tietyssä tietokannassa ole paikkaansapitäviä, ovat arviot helposti vaihdettavissa sopivampiin, mikäli ne ovat tiedossa.

Hakuun kuuluvien sipaisujen määrä ja tyyppi on laskettavissa käytettyjen indeksisiivujen pituudesta sekä siitä, onko indeksi paksu indeksi (määritellään luvussa 6.1) tai ryvästävä (katso luku 3.4). Mikäli haun suorituksessa ei ole käytetty indeksiä, on oletettavissa, että n riviä sisältävä taulu aiheuttaa yhden hajasipaisun sekä $n - 1$ sarjasipaisua, kun taulun kaikki rivit käydään läpi. Mikäli haussa on käytetty indeksiä, aiheuttaa jokainen indeksisiivu yhden hajasipaisun sekä $(n * l) - 1$ sarjasipaisua, jossa n on indeksoidun taulun rivimäärä ja l indeksin läpäisykerroin. Mikäli indeksi ei ole joko paksu tai ryvästävä, aiheuttaa indeksiin kuulumattomien sarakkeiden luku lisäksi $n * l$ hajasipaisua.

Arvio voidaan tehdä myös **suhteellisena**, jolloin eo. QUBE-kaavassa tulosrivit jätetään huomioimatta. Näin voidaan suoraviivaistaa prosessia verrattaessa saman haun eri hakupolkuja. Tulosten rivien, ja siten myös rivimäärien, ollessa samoja, niiden käsittelyyn kuluva aika ei vaihtele eri hakupolkujen välillä, ja voidaan näin jättää vertailusta pois.

6 B-puuindeksin valinta

Kun indeksoitava tietokantataulu on tunnistettu, on seuraavaksi valittava sopiva indeksi. Luvussa 3 todetun mukaisesti B-puuindeksit ovat yleisimmin käytössä relaatiotietokannanhallintajärjestelmissä, joten luvussa keskitytään lähinnä B-puuindeksin sarakkeiden valintaan. Koska indeksoinnin tarve ilmenee toivottua hitaammin suoriutuvana hakuna, on indeksin suunnittelu hakulähtöistä ja hakukohtaista.

Yksittäinen indeksi ei kuitenkaan ole varattu vain ja ainoastaan tiettyä hakua varten. Luvuissa 6.1-6.3 keskitytään lähinnä optimaalisen, tai mahdollisimman paljon hakua nopeuttavan indeksin suunnitteluun. Yleensä kuitenkin riittää, että jo olemassa olevien indeksien joukosta löytyy jokin, jonka avulla haku saadaan tarpeeksi nopeaksi.

6.1 Kolmen tähden indeksi

Tapio Lahdenmäen kirjoissaan [Hovi, s. 186-187] ja [Lahdenmäki, s. 49-52] esittelemä kolmen tähden indeksi (engl. *three star index*) on ideaalinen indeksi yksittäisen `SELECT`-lauseen suhteen. Se koostuu kolmesta kriteeristä eli tähdestä, joista jokainen auttaa omalta osaltaan `SELECT`-lauseen mahdollisimman nopeaan suorittamiseen. Tähdet ovat Lahdenmäen määrittämässä järjestyksessä ja, vaikka niihin viitataankin järjestysnumeroilla, on kukin tähti edellisestä riippumaton itsenäinen kokonaisuutensa. Jokainen kyseisistä kriteereistä kykenee itsenäisesti parantamaan `SELECT`-lauseen suorituskykyä, joskin useimmissa tilanteissa kaikkien kolmen toteutuminen takaa parhaan mahdollisen suoritusajan.

Ensimmäisen tähden kriteerissä indeksi järjestää kaikki `SELECT`-lauseen tarvitsemat rivit siten, että ne löytyvät indeksistä peräkkäin, tai ainakin niin läheltä toisiaan kuin mahdollista. Tällä tavoin pyritään välttämään hajasipaisuja ja tarpeettomien rivien lukuja indeksiä luettaessa. Vähittäisvaatimus tähän on, että kaikki `SELECT`-lauseen yhtäsuuruuspredikaattien käyttämät sarakkeet ovat indeksin alussa missä tahansa järjestyksessä. Arvovälin määrittävän predikaatin vaikutusta käsitellään luvussa 6.2.

Toisen tähden kriteerin täyttääkseen indeksi järjestää hakemansa rivit valmiiksi `SELECT`-lauseen vaatimaan järjestykseen. Tämän toteuttamiseksi indeksin tulee sisältää kaikki `SELECT`-lauseen `ORDER BY` -osan sisältämät sarakkeet samassa järjestyksessä ja samoja järjestyssääntöjä (`ASC`, `DESC`) noudattaen. Näin pystytään takaamaan, että indeksiä käytettäessä rivit saadaan luettua suoraan halutussa järjestyksessä, eikä aikaa kulu näin rivien järjestämiseen erikseen.

Kolmannen tähden kriteeri on kriteereistä yleensä tärkein etenkin, jos indeksi ei ole ryvästävä. Tämä kriteeri vaatii, että indeksi sisältää kaikki `SELECT`-lauseessa esiintyvät sarakkeet missä tahansa järjestyksessä. Tämän ehdon täyttävää indeksiä kutsutaan **paksuksi indeksiksi** (engl. *fat index*), ja se mahdollistaa **täyden indeksiluvun** (engl. *full index scan*) eli haun toteuttamisen vain ja ainoastaan indeksin avulla. Kaiken tarvittavan datan löytyessä valmiiksi indeksistä, ei varsinaista tietokantataulua tarvitse käydä lukemassa erikseen. Tällöin vältytään kaikilta hajasipaisuilta tietokantatauluun, vaikka kyseessä ei olisikaan ryvästävä indeksi. Jos indeksi täyttää samalla myös ensimmäisen tähden kriteerin, voidaan kaikki data lukea yhdellä hajasipaisulla ja sarjasipaisuilla.

Esimerkiksi kolmen tähden indeksi voidaan johtaa helposti hausta

```
SELECT nimi, ika, kaupunki, laani FROM esimerkkitaulu
WHERE laani = 'Uusimaa' SORT BY nimi
```

Ensin otetaan ensimmäisen tähden vaatimusten mukaisesti indeksiin mukaan sarake `laani`, koska tämä määrätään yhtäsuuruuspredikaatilla. Seuraavaksi lisätään indeksiin sarake `nimi`, jotta toisen tähden ehto järjestyksestä täytetään. Kolmannen tähden vaatimus kaikkien sarakkeiden mukanaolosta täytetään lisäämällä jäljelle jääneet sarakkeet `ika` ja `kaupunki`. Näin ollen `SELECT`-lauseen kolmen tähden indeksi voidaan luoda lauseella

```
CREATE INDEX tahdet ON esimerkkitaulu
(laani, nimi, ika, kaupunki)
```

6.2 Kahden kandidaatin menetelmä

Luvun 6.1 näkökulmasta ideaalisen indeksin toteuttaminen ei aina ole mahdollista. Esimerkiksi tietyn välin rajaava predikaatti voi estää kahden ensimmäisen tähden ehtojen yhtäaikaisen toteutumisen.

Otetaan **esimerkiksi** haku

```
SELECT nimi, ika, laani FROM esimerkkitaulu WHERE
laani = 'Uusimaa' AND ika > 50 ORDER BY nimi
```

Jotta indeksisiivu olisi mahdollisimman lyhyt ja yhtenäinen, tulisi indeksin ensimmäisiksi sarakkeiksi asettaa `laani` ja `ika`. Toisaalta, jotta indeksiluvun tulos olisi valmiiksi järjestyksessä, tulisi sarakkeen `nimi` olla ennen saraketta `ika`. Tämä järjestys taas rikkoisi indeksisiivun useampaan osaan, eikä näin ole mielekäs valinta hajalukujen määrän lisääntyessä.

Mikäli haulle ei pystytä toteuttamaan luvun 6.1 mukaisesti ideaalista indeksiä, voidaan sille suunnitella **paras mahdollinen indeksi kahden indeksikandidaatin avulla.** [Lahdenmäki, s. 54-55] Ensimmäinen kandidaatti (kandidaatti A) muodostetaan noudattaen seuraavia askeleita:

1. Valitaan hausta sarakkeet, jotka liittyvät yhtäsuuruuspredikaatteihin. Nämä muodostavat indeksin ensimmäiset sarakkeet missä tahansa järjestyksessä.

2. Valitaan seuraavaksi sarakkeeksi se, joka liittyy pahimmassa tapauksessa pienimmän läpäisykertoimen omaavaan predikaattiin. Indeksoinnille hankalat predikaatit (kuvataan tarkemmin luvussa 6.3) kannattaa jättää huomioimatta tässä vaiheessa.

3. Lisätään aiemmin lisättyjen perään `ORDER BY` -sarakkeet samassa järjestyksessä kuin nämä ovat haussa.

4. Lisätään indeksin loppuun ne sarakkeet, joihin haku vielä edellisten lisäksi viittaa. Haun tehokkuuden kannalta sarakkeiden järjestyksellä ei ole väliä, mutta datan osalta muuttuvimmat sarakkeet suositellaan jätettäväksi viimeisiksi.

Mikäli tulosrivejä ei saada kandidaatti A:n avulla halutussa järjestyksessä, vaan indeksin lukemisen jälkeen rivit joudutaan järjestämään erikseen, luodaan toinen kandidaatti. Kandidaatti B voidaan luoda käyttäen täsmälleen samoja ohjeita kuin kandidaatti A:nkin osalta lukuunottamatta vaihetta 2, jota ei suoriteta lainkaan. Molemmat kandidaatit täyttävät tällöin kolmen tähden indeksin kolmannen tähden vaatimuksen. Kandidaatti A painottaa kuitenkin ensimmäisen tähden vaatimuksia, kun kandidaatti B painottaa enemmän toisen tähden vaatimuksia.

Valinta kandidaattien välillä tulee suorittaa **käyttötarkoituksen mukaan**. Mikäli mahdollista, molempia kandidaatteja voidaan testata varsinaisessa tietokannassa. Tällöin valitaan yksinkertaisesti paremmin suorituva kandidaatti.

Mikäli valinta tehdään ilman käytännön testausta, joudutaan sopivamman kandidaatin valitsemiseksi arvioimaan eri kandidaattien suoriutumisaikaa esimerkiksi luvussa 5.3 esitettyä QUBE-menetelmää käyttäen. Arvioinnissa on otettava huomioon kandidaatin A vaatima tulosrivien järjestäminen. Nykyisillä tietokannanhallintajärjestelmillä järjestäminen on suhteellisen nopea operaatio. Järjestämisoperaation suoritusajaksi voidaan arvioida kuluvan keskimäärin 10μs järjestettävää riviä kohden, joka on tarpeeksi pieni luku sisällytettäväksi QUBE-menetelmässä tulosrivin käsittelyyn liittyvään aika-arvioon. Tämän vuoksi **kandidaatin A tarjoamasta**

useimmiten nopeammasta levyluvusta on enemmän hyötyä kuin kandidaatin B tarjoamasta järjestämisen tarpeen välttämisestä. Mutta kandidaatti B on kuitenkin huomioimisen arvoinen vaihtoehto etenkin, jos haun on tarkoitus palauttaa vain tietty määrä ensimmäisiä tulosrivejä. Tällöin nimittäin kandidaattia A käyttäen joudutaan lukemaan kaikki tulosrivit rajoituksesta huolimatta, kun taas kandidaattia B käyttäen levyluvut voidaan rajoittaa haluttuun määrään rivejä. [Lahdenmäki, s. 55]

Luvun alun esimerkille indeksikandidaatti A sisältäisi sarakkeet järjestyksessä `laani`, `ika` ja `nimi`, kun taas kandidaatti B sisältäisi sarakkeet järjestyksessä `laani`, `nimi` ja `ika`. Käyttäen luvun 5.3 QUBE-menetelmää ja luvussa 5.2 esiteltyjä läpäisykertoimia näitä kahta kandidaattia voidaan vertailla keskenään. Tarkastellaan ensimmäiseksi kandidaattia A. Oletetaan, että predikaatin `ika > 50` läpäisykerroin on 20%. Tästä saadaan indeksisiivun pituudeksi (katso luku 5.2) $1000000 * 0,2 * 0,01 = 2000$ riviä, joiden lukuajaksi voidaan laskea (katso luku 5.3) $10ms + 1999 * 0,01ms + 2000 * 0,1ms \approx 230ms$. Kandidaatti B:n määrittämä indeksisiivu taas on pituudeltaan $1000000 * 0,01 = 10000$ riviä (olettaen, että hakua varten luetaan yksi iso indeksisiivu, eikä useaa lyhyempää siivua), josta voidaan taas laskea hakuajaksi $10ms + 9999 * 0,01ms + 10000 * 0,1ms \approx 1100ms$. On selvästi nähtävissä, että kandidaatti B on käytännöllinen vain, jos haussa pyydetään vain rajoitettua määrää tuloksia, esimerkiksi 50 ensimmäistä tulosta. Tällöin kandidaatti A:ta käyttäen hakuaika pysyy edelleen samana, mutta koska kandidaatti B:ssä rivit ovat valmiiksi järjestyksessä, vie haku vain $10ms + 49 * 0,01ms + 50 * 0,1ms \approx 15ms$.

6.3 Indeksoinnille hankalat predikaatit

On olemassa predikaatteja, joiden hakuajan arvioimisessa tietokannanhallintajärjestelmä ei voi käyttää apuna indeksejä. Tyypillisesti nämä ovat esimerkiksi funktioita taikka negaatioita sisältäviä hakuehtoja. Indeksoinnin kannalta **hankalat predikaatit riippuvat siitä, kuinka indeksointi ja indeksien käyttö on toteutettu**. Näin ollen kaiken kattavaa listaa hankalista predikaateista ei voida antaa. `LIKE` ja `OR` ovat esimerkkejä haastavista predikaateista, joita käytettäessä tulee yleensä noudattaa varovaisuutta ja selvittää tietokannanhallintajärjestelmän kyvykkyys käsitellä niitä.

`LIKE`-predikaatilla voidaan hakea kenttiä, jotka sisältävät annetun maskin mukaisen merkkijonon. Maski voi sisältää erinäisiä korvausmerkkejä, joista esimerkiksi %-merkki voi korvata minkätahansa mittaisen merkkijonon. Lähteen [Lahdenmäki, s. 91] mukaan jotkin tietokannanhallintajärjestelmät eivät pysty käyttämään indeksejä lainkaan predikaatin seulomiseen. Toiset taas pystyvät käyttämään indeksejä vain rajoitetusti `LIKE`-predikaatin yhteydessä. Esimerkiksi DB2-tietokannanhallinta-

järjestelmä pystyy rajaamaan haun rivit tiettyyn indeksisiivuun, mikäli maski ei ala millään korvausmerkillä.

Lähteen [Lahdenmäki, s. 92] mukaan myös **`OR`-liitokset** ovat hankalia käsitellä. Vaikka kaksi `OR`illa toisiinsa liitettyä predikaattia voivatkin olla helposti löydettävissä indeksien avulla, ei tietokannanhallintajärjestelmä enää pysty hylkäämään rivejä vain yhden sarakkeen avulla, vaan joutuu tarkistamaan myös toisen sarakkeen arvon. Pahimmassa tapauksessa tietokanta joutuu käymään taulun kaikki rivit läpi. Teoriassa on mahdollista lukea molempien liitettyjen predikaattien mukaiset indeksisiivut erikseen ja liittää nämä tulokset yhteen, mutta ei pidä olettaa optimoijan pystyvän tähän operaatioon itsenäisesti.

7 Indeksoinnin negatiiviset sivuvaikutukset

Indeksointi aiheuttaa aina epätoivottuja sivuvaikutuksia. Lähteen [Lahdenmäki, s. 58] mukaan indeksoinnin käyttöä tulee harkita tarkkaan etenkin, jos tietokantaan on tarkoitus tehdä huomattava määrä lisäys-, päivitys- ja poisto-operaatioita. **Kukin indeksi hidastaa ko. muokkausoperaatioita**, koska vastaavat operaatiot on toistettava myös indekseille. Kyseisten operaatioiden viemät ajat ovat verrattavissa luvussa 5.1 kuvattuihin haja- ja sarjasipaisuihin kuluviin aikoihin. Esimerkiksi, jos taululle on määrätty kymmenen indeksiä, yhden rivin lisääminen tauluun kestää 10ms:n sijaan indeksien vuoksi 0.1s. Toisaalta, mikäli samaan tauluun lisätään 20 riviä, jotka järjestyvät yhdessä indeksissä peräkkäin, indeksien päivitykseen kuluva aika on luvun 5.3 QUBE-kaavan mukaisesti noin $(9 * 10ms + 0.1ms) * 20 \approx 1.8s$.

Kuten lähde [Pratt, s. 163] toteaa, indeksi lisäksi **vie aina levytilaa**, koska tauluihin tehtävät muutokset on tehtävä myös indekseihin. **Poikkeuksena** ovat luvussa 3.4 käsitellyt **ryvästävät indeksit**, koska nämä eivät luo uutta tietorakennetta alkuperäisen taulun rinnalle. B+-puuindeksien osalta indeksien vaatima levytila on lähteen [Lahdenmäki, s. 61] ohjeiden mukaan helposti laskettavissa, jos vain indeksoitavan taulun rakenne ja joko todellinen tai arvioitu rivimäärä on tiedossa. Yksittäisen indeksin vaatima levytila voidaan laskea kertomalla indeksoitavien sarakkeiden pituus rivimäärällä ja kertoimella 1.5, joka kuvaa puurakenteeseen liittyviä lisävaateita levytilan suhteen, kuten osoittimia ja solmujen tyhjiä tiloja. Kuten luvussa 3.1 B-puita käsiteltäessä todettiin, voi puun solmuissa olla jopa puolet tyhjää tilaa, joka on kuitenkin varattava solmuja varten.

Lisäksi **indeksit lisäävät yleistä kovalevykuormitusta** hidastaen näin koko tietokannan toimintaa. Levyoperaatiot suoritetaankin tietokannanhallintajärjestelmissä yleensä asynkronisesti. Nämä siis jättävät muutokset odottamaan väliaikaiseen, tilaan kunnes ne voidaan kirjoittaa levylle häiritsemättä hakujen suoritusta. Muutokset on silti suoritettava jossain vaiheessa, mikä vaikuttaa keskimääräiseen kovalevykuormitukseen ja tietokannan yleiseen suorituskykyyn.

Lähteen [Lahdenmäki, s. 60] mukaan **suurimpana ongelmana ovat rivien lisäys- ja poisto-operaatiot**, jotka voivat aiheuttaa indeksien lukumäärän verran tietokantasivun luku- ja kirjoitusoperaatioita jokaista lisättyä tai poistettua riviä kohden. Poikkeuksena ovat ryvästävät indeksit ja indeksit, joissa operaatiot kohdistuvat samaan tietokantasivuun. Päivitysoperaatioiden vaikutus levykuormitukseen on yleensä pienempi, sillä nämä aiheuttavat luku- ja kirjoitusoperaation vain niille indekseille, joiden sarakkeita on muutettu.

8 Yhteenveto

Indeksit ovat varsin tehokas keino tietokantahakujen nopeuttamiseen. Ne soveltuvat erityisesti tietokantoihin, joissa datan muutokset ovat tarpeeksi harvoja, jolloin niihin kuluvien operaatioiden suoritusajan kasvu ei haittaa tietokantasovelluksen toimintaa.

Koska indeksejä voidaan lisätä tietokantaan lähes milloin tahansa, soveltuvat ne hyvin reaktiiviseen ongelmanratkaisuun. Myös indeksien käytön suunnittelu ennen tietokannan olemassaoloa onnistuu, mikäli pystytään arvioimaan tiettyjä ominaisuuksia tietokannan datasta. Tällöin voidaan läpäisykertoimia ja arvioituja sipaisuaikoja käyttäen arvioida ennalta indeksoinnin tehokkuutta.

Tutkielma käsitteli vain indeksoinnin perusperiaatteita. Huomioimatta jäivät esimerkiksi taululiitoksien indeksoinnit sekä useita tauluja käsittelevissä hauissa taulujen läpikäyntijärjestyksen vaikutukset indeksien tehokkuuteen ja valintaan. Valitettavasti indeksien käyttöä ei tietokantoja yleisesti käsittelevissä kirjoissa käsitellä kovinkaan syvällisesti. Suurin osa tutkielmasta tukeutuu yhteen indeksointispesifiin teokseen [Lahdenmäki], joka lähteistä ainoana käsittelee indeksointia peruskäsitteitä ja -tekniikoita syvällisemmin.

Lähteet

[Connolly] Thomas M. Connolly, *Database Systems: A Practical Approach to Design, Implementation and Management*, Addison-Wesley, 2005.

[Date] C. J. Date, *An Introduction to Database Systems*, 5.ed, Addison-Wesley, Reading (MA), 1991.

[Elmasri] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, 3.ed., Addison-Wesley, Reading (MA), 2000.

[Hovi] Ari Hovi, Jouni Huotari ja Tapio Lahdenmäki, *Tietokantojen suunnittelu & indeksointi*, Docendo, Jyväskylä, 2005.

[Kifer] Michale Kifer, Arthur Bernstein and Philip M. Lewis, *Databases and Transaction Processing: An Application-Oriented Approach*, Addison-Wesley, Boston (MA), 2002.

[Lahdenmäki] Tapio Lahdenmäki and Michael Leach, *Relational Database Index Design and the Optimizers*, Wiley-Interscience, 2005.

[Microsoft] Microsoft Corporation, *SQL Server 2008 Books Online (October 2008), CREATE INDEX (Transact-SQL)*, <URL: `http://msdn.microsoft.com/en-us/library/ms188783.aspx`>, viitattu 3.11.2008.

[Oracle] Oracle, *Oracle® Database Administrator's Guide 11g Release 1 (11.1), CREATE INDEX*, <URL: `http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/statements_5011.htm`>, viitattu 3.11.2008.

[PostgreSQL 1] PostgreSQL Global Development Group, *PostgreSQL 8.3.5 Documentation, CREATE INDEX*, <URL: `http://www.postgresql.org/docs/current/static/sql-createindex.html`>, viitattu 3.11.2008.

[PostgreSQL 2] PostgreSQL Global Development Group, *PostgreSQL 8.4devel Documentation, GiST and GIN Index Types*, <URL: `http://developer.postgresql.org/pgdocs/postgres/textsearch-indexes.html`>, viitattu 9.11.2008.

[PostgreSQL 3] PostgreSQL Global Development Group, *PostgreSQL 8.3.5 Documentation, Index Types*, <URL: `http://www.postgresql.org/docs/8.1/static/indexes-types.html`>, viitattu 4.1.2009.

[Pratt] Philip J. Pratt and Joseph J. Adamsk, *Database Systems: Management and Design*, luku 5.2, Boyd & Fraser, Boston (MA), 1987.

[Sun] Sun Microsystems, *MySQL 5.0 Reference Manual, 7.4.5. How MySQL Uses Indexes*, <URL: `http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html`>, viitattu 3.11.2008.