Relaatiotietokannasta graafitietokantaan – Graafitietokannan edut tietojärjestelmän tietovarastona

Jussi Kallava

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Pro gradu -tutkielma
Ohjaaja: Marko Junkkari
Kesäkuu 2018

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Jussi Kallava: Relaatiotietokannasta graafitietokantaan – Graafitietokannan edut tietojärjestelmän tietovarastona
Pro gradu -tutkielma, 52 sivua
Kesäkuu 2018

Relaatiotietomalli on vakiintunut tapa organisoida ja käsitellä tietojärjestelmän tietoja. Sosiaalisen median kasvun, sekä uudenlaisen sisällön tuottamiseen keskittyneen internetkulttuurin myötä uudet, erilaiset tavat mallintaa tietoa ovat nousseet esiin. Vaatimusten kasvaessa tiedon nopean saatavuuden suhteen, ovat NoSQL-pohjaiset tietovarastot tulleet usein osaksi tietojärjestelmän kokonaisarkkitehtuuria. Tässä tutkielmassa esitellään tietomalleja, jotka ovat tietojärjestelmäkehityksessä relaatiotietomallin ohella keskeisiä tapoja mallintaa tietoa. Syvällisemmin keskitytään relaatiotietokannan ja graafitietokannan eroihin. Tutkielmassa etsitään myös tapauksia, joissa perinteisen relaatiopohjaisen tietokannan muuttaminen graafipohjaiseksi tuo esiin konkreettisia hyötyjä. Nämä hyödyt saattavat olla esimerkiksi tiedonmallinnukseen liittyviä, semanttisia tai arkkitehtuurisia hyötyjä. Hyödyt saattavat olla myös merkittäviä tiedonhaun kannalta, esimerkiksi yksinkertaisempien ja tehokkaampien hakukyselyjen muodossa. Toisaalta tutkielmassa huomioidaan tapaukset, joissa graafitietokantaan siirtymiselle ei löydy perusteltua syytä.

Avainsanat ja -sanonnat: tietomalli, relaatiotietokanta, graafitietokanta, DBMS, NoSQL, CAP, ACID, BASE, SQL, Cypher.

Sisällys

1. Johdanto

Sosiaalisen median, keskustelusivustojen, sekä yleisesti internetin kautta kulkevan tietomäärän merkittävän kasvun myötä, perinteisten relaatiotietokantojen rinnalle on kehitetty tietovarastoja, jotka tukevat paremmin suurten tietomäärien käsittelyä. Tällaisia tietovarastoja kutsutaan NoSQL-tietovarastoiksi. NoSQL-tietovarastojen keskeinen piirre on, että ne eivät tue relaatiotietomallia, SQL-kyselykieltä, eivätkä pääsääntöisesti myöskään tietokantatapahtumia. Usein NoSQL-tietovarastot toteuttavat hajautettua arkkitehtuuria, sekä relaatiotietomallia kevyempiä tietomalleja, joiden ansiosta tällaiset tietovarastot ovat monilta osin relaatiotietokantoja tehokkaampia. [Kuznetsov and Poskonin, 2014]

NoSQL-tietovarastot ovat kasvattaneet merkittävästi suosiotaan ensimmäisten NoSQL-tyyppisten tietokantaimplementaatioiden tultua markkinoille 2000-luvun alussa. [Mpinda et al., 2015] Tällä hetkellä merkittävimmiksi tuotteiksi NoSQL-tyyppisistä tietokantaratkaisuista ovat nousseet mm. Netflixin ja Instagramin käyttämä Apachen Cassandra, sekä dokumenttivarasto MongoDB. [DB-Engines, 2018]

Vaikka NoSQL on viime aikoina kasvattanut suosiotaan, niin graafitietokannat ovat jääneet hieman monien muiden NoSQL-ratkaisujen varjoon. Tähän lienee osasyynä se, että suurten yritysjättien käyttämät tietokantaratkaisut vievät huomiota pois muista NoSQL-tuotteista. Toisaalta taas osasyynä saattaa olla graafitietokantojen jollain tavalla kompleksinen vaikutelma, verrattuna esimerkiksi dokumentti- tai avain-arvo-pohjaiseen tietovarastoon. Graafitietomalliin pohjautuvan tietovaraston hyödyntäminen on kuitenkin joissain käyttötapauksissa erittäin varteenotettava ratkaisu. Erityisesti mikäli tiedonmallinnuksen kohde on graafipohjaisesti luonnollisesti mallinnettavissa.

Graafitietokannat, kuten muutkin NoSQL-tietokannat, saatetaan toisinaan mieltää helppona ratkaisuna tietojärjestelmän suorituskykyongelmiin. Nämä ongelmat tulevat useimmiten vastaan relaatiopohjaisten tietojärjestelmien kanssa, kun datamäärät kasvavat tarpeeksi suuriksi. Useimmiten ratkaisu ei kuitenkaan ole näin yksinkertainen. Tietokannan, kuten muidenkin tietojärjestelmän tietovarastoon liittyvien arkkitehtuurivalintojen, tulisi pohjautua puhtaasti tietojärjestelmän tarpeisiin. Nämä tarpeet voivat liittyä esimerkiksi tiedon käsittelyyn ja hakuihin, tai suuriin datamääriin.

Tässä tutkielmassa pyritään löytämään seikkoja, joiden perusteella siirtyminen relaatiotietokannasta johonkin toiseen tietovarastoon saattaisi olla perusteltua. Tutkielman tukena käytetään Elmasrin ja Navathen [2000] esittelemää tietomalli-kaaviota. Tarkastellaan tämän tietomallin pohjalta mallinnetun relaatiokaavion rakennetta ja pohditaan sen etuja, ja toisaalta mahdollisia ongelmia. Vastaavasti mallinnetaan esitellyn kaavion pohjalta graafimalli ja tutkitaan syntyneitä eroja relaatiomalliin verrattuna, rakenteen ja mahdollisten tietovarastoon kohdistuvien hakukyselyjen osalta.

Tutkielman alussa, luvussa 2 esitellään tutkielman kannalta olennaiset tietomallit. Tietomallien jälkeen, luvussa 3 tutustutaan relaatiotietokantoihin, sekä niihin olennaisesti liittyviin

käsitteisiin, kuten tapahtumiin ja ACID-malliin. Luvussa 4 esitellään NoSQL-tietovarastoja yleisesti, jonka jälkeen luvussa 5 perehdytään paremmin graafitietokantoihin. Luvussa 6 suoritetaan tietomallien vertailua, muodostamalla ER-kaavion pohjalta relaatiomalli, sekä graafimalli. Luvussa 6.3 puretaan näitä eroavaisuuksia ja tutkitaan eroja myös kyselyjen tasolla. Lopuksi luvussa 7 on johtopäätökset tietomallien vertailusta, sekä relaatio- ja graafitietokantojen eroista.

2. Tiedonmallinnus

Jotta tietojärjestelmää varten voidaan rakentaa toimiva ja järjestelmää hyvin palveleva tietovarasto, on tiedon rakenne ja sisältö mallinnettava asianmukaisesti. Yksi peruslähtökohta tietokannan rakentamisessa onkin tietovaraston abstrahoiminen. Abstrahoinnilla pyritään piilottamaan tietokannasta yksityiskohtia, joita suurin osa tietokannan käyttäjistä ei tarvitse. Tätä tarkoitusta varten on kehitetty erilaisia tietomalleja (data model). Tietomalli on kokoelma käsitteitä, joiden avulla voidaan kuvata tietokannan rakennetta, jotta abstraktio saadaan mallinnettua. Tietokannan rakenteella tarkoitetaan tietovaraston sisältämiä tietotyyppejä, tietotyyppien välisiä suhteita, sekä rajoituksia, joista eheä tietorakenne koostuu. [Elmasri and Navathe, 2000]

Tietomalli käsitteenä on kuitenkin määritelty eri lähteissä hieman eri tavoin. Kuten Codd [1980] mainitsee, niin lukuisissa lähteissä viitataan tietomalleihin ainoastaan kokoelmana tietorakennetyyppejä. Coddin tulkinnan mukaan tietomalleja tutkittaessa on kuitenkin erittäin olennaista huomioida myös operaattorit ja eheyssäännöt (integrity rules), jotta tietomallien vertailua voidaan pitää riittävänä.

Coddin tulkinta tietomalleista onkin lähellä Elmasrin ja Navathen [2000] määritelmää korkean tason ja matalan tason tietomallien väliin jäävästä loogisesta tietomallista (representational data model). Elmasri ja Navathe jakavat tietomallit kolmeen kategoriaan, korkean tason käsitteellisiin tietomalleihin (conceptual data model), matalan tason fyysisiin tietomalleihin (physical data models), sekä näiden väliin jääviin loogisiin tietomalleihin. Jatkossa tässä tutkielmassa tietomallilla tarkoitetaan Elmasrin ja Navathen esittämää loogista tietomallia, jollaisiksi tutkielmassa esiteltävät relaatio- ja graafitietomallikin voidaan kategorisoida. Myöhemmin esiteltävä ER-malli kuitenkin istuu määritelmällisesti paremmin korkean tason käsitteellisiin malleihin.

Tietomalleilla pyritään kuvaamaan reaalimaailman tilanteita, asioita ja niiden suhteita, jäsennellysti jatkokehitystä varten. Tiedon muoto ja rakenne voivat kuitenkin vaihdella tapauskohtaisesti hyvinkin paljon, jolloin yhdenlainen tietomalli ei sovi kuvaamaan kaikkia järjestelmiä. Tietomalli tulisi valita tukemaan käsiteltävän tiedon rakennetta ja ominaisuuksia. [Angles and Gutierrez, 2008]

Tietomalli koostuu kolmesta sille olennaisesta osa-alueesta. Ensiksi, se sisältää kokoelman tietovaraston tietotyypeistä, jotka mallintavat käsiteltävän tiedon eri entiteettejä. Toiseksi, se sisältää kokoelman operaattoreista tai säännöistä, joita voi soveltaa ensiksi mainitun listan valideihin instansseihin. Kolmanneksi, se sisältää säännöstön, joka ulkoisesti tai sisäisesti määrittelee tietomallin tilat tai tilan muutokset, kuten esimerkiksi lisäys-, päivitys- tai poistosäännöt. [Codd, 1980]

Tietovaraston mallinnuksen lisäksi on olemassa myös muita käyttökohteita tiedonmallinnuksen hyödyntämiseen. Tietomalleja voi käyttää apuna tietojärjestelmän kehittymisestä mahdollisesti seuraavien tietovaraston arkkitehtuuri- tai teknologiamuutoksien hallinnointiin. Hyvin toteutettujen tietomallien avulla voidaan tunnistaa tietojärjestelmän kannalta

sallitut tietotyypit ja näiden yhteydet. Tällainen käsiteltävän tiedon mallinnus auttaa merkittävästi tietovaraston muutosten hallinnassa. [Codd, 1980]

Vuoteen 1979 mennessä tietomalleja oli esitetty jo yli 40. Tosin suurin osa näistä oli melko keskeneräisiä hahmotelmia, jotka eivät ole jääneet elämään pidemmäksi aikaa. [Codd, 1980] Näistä alan tutkimuksissa esitetyistä malleista merkittävimmiksi on noussut kolme tietomallia. Nämä tietomallit ovat verkkotietomalli, relaatiotietomalli ja entiteettimalli. Näiden lisäksi vuonna 1976 Peter Chen [1976] esitti työvälineen korkean tason mallinnukseen, Entity-Relationship-mallin (ER-malli), joka hyödyntää parhaita puolia kaikkien kolmen edellä mainitun mallin ominaisuuksista.

Lisäksi Chen esitteli loogisen tietomallin määrittelyssä neljä tasoa, joiden kautta tietomallia voidaan visualisoida. Ensimmäisellä tasolla on entiteettejä ja suhteita koskeva tieto. Nämä voidaan ajatella ER-mallin perustana. Toisella tasolla tulee tiedon rakenne, joka yhdistää entiteetit ja suhteet. Kolmannella tasolla on ns. polkuriippumaton tietorakenne (access-path-independent data structure). Relaatiotietomallin voidaan ajatella yltävän tälle tasolle. Neljäs taso on vastaavasti polkuriippuvainen tietorakenne (access-path-dependent data structure), jollaisena esimerkiksi verkkotietomallia voidaan pitää. Palataan näihin Chenin esittämiin tietomallien tulkintoihin tutkielman myöhemmässä vaiheessa, kun rakennetaan graafimallia ER-mallin pohjalta. [Chen, 1976]

Tässä tutkielmassa esitellään ER-malli, relaatiotietomalli, verkkotietomalli, sekä graafitietomalli. Nämä ovat merkittävimmät tietomallit tutkielman kannalta, jossa johdetaan relaatio- ja graafimallit ER-mallin pohjalta.

2.1. Entity-Relationship-malli

Yksi käytetyimmistä korkean tason käsitteellisen mallintamisen työvälineistä on ER-kaavio (Entity-Relationship diagram). Vaikka ER-malli esiteltiin vasta kuusi vuotta relaatiotietomallin jälkeen, käsitellään se tässä tutkielmassa ennen relaatiotietomallia. Syynä tähän on se, että ER-malli on käsitteellisen mallintamisen työkalu, jota muun muassa relaatiopohjaisen tietojärjestelmän kehityksessä hyödynnetään ennen relaatiotietomallia. Toisin sanoen, ER-malli on apuväline monien muiden tietomallien luomiseen. Apenyo [1999] esitteli tämän kronologisesta aiheiden esittelystä aiheutuvan loogisen ristiriidan. Tämän vuoksi tässäkin tutkielmassa aiheet käsitellään edellä mainitussa järjestyksessä.

ER-kaavion ydinprimitiivi on entiteettityyppi (entity type), joka viittaa joukkoon samankaltaisia entiteettejä. Entiteettityypillä kuvataan jotakin todellisen maailman asiaa, joita tietomalli pitää sisällään. Entiteettityypillä on aina attribuutteja (attribute), jotka kuvaavat entiteetin ominaisuuksia tai tilaa. Entiteettityyppi voisi olla esimerkiksi auto, ja sillä voisi olla attribuuttina väri. Entiteettityypillä on lähes aina avainarvo, joka yksilöi entiteetin. On kuitenkin olemassa entiteettityyppejä, joilla tällaista avainarvoa ei ole muodostettavissa. Tällaisia entiteettejä kutsutaan heikoiksi entiteettityypeiksi (weak entity type). Heikoille entiteettityypeille tulee määritellä suhde jonkin vahvan entiteettityypin kanssa, jonka avulla heikko entiteettityyppi saa avainarvonsa.

Tällaista heikon entiteettityypin yhdistävää suhdetta kutsutaan määritteleväksi suhteeksi (identifying relationship). [Elmasri and Navathe, 2000]

ER-mallissa kuvatut attribuutit jakautuvat käyttötapaustensa perusteella eri tyyppisiksi attribuuteiksi. Attribuutti voi olla joko kooste tai yksinkertainen attribuutti (composite/simple). Koosteattribuutti haarautuu ER-mallissa aliattribuuteiksi, kuten esimerkiksi “Nimi” voisi haarautua attribuuteiksi “Etunimi” ja “Sukunimi”. Koosteattribuutit ovat hyödyllisiä tilanteissa, joissa koosteen muodostamaa arvoa hyödynnetään, mutta on myös tarve porautua tarkempaan aliattribuutin sisältämään tietoon. Lisäksi attribuutti voi olla yksi- tai moniarvoinen (single-valued/multivalued). Useimmiten attribuutit ovat yksiarvoisia, jolloin kyseisellä attribuutilla voi olla vain yhdeksi tietueeksi tulkittava arvo. Toisinaan on kuitenkin tarve ylläpitää mahdollisesti useampaa arvoa samassa attribuutissa, kuten esimerkiksi “Tutkinnot”-attribuutti, joka voi sisältää useammankin kuin yhden tutkinnon. Edellä mainittujen attribuuttityyppien lisäksi on olemassa vielä jako tallennettujen ja johdettujen attribuuttien välillä (stored/derived). Johdetulla attribuutilla tarkoitetaan sitä, että sen arvo on johdettavissa tallennetuista attribuuteista. Tällainen johdettu attribuutti voisi olla esimerkiksi “Ikä”, joka voitaisiin johtaa tallennetusta attribuutista “Syntymäaika”. [Elmasri and Navathe, 2000]

Entiteettityyppien ja näiden attribuuttien lisäksi ER-mallissa, nimensä mukaisesti, ydinkomponentti on suhde (relation). Suhde määrittelee kahden entiteetin välistä vuorovaikutusta tai tilaa. Voidaan ajatella esimerkiksi “isä-poika”-suhdetta kahden “henkilö”-entiteetin välillä. Lopulta kuitenkin entiteettityypin ja suhteen välinen ero saattaa olla tulkinnanvarainen. Esimerkiksi “avioliitto” saattaisi olla yhtä lailla entiteettityyppi tai suhde. Tällöin lopullisen linjauksen entiteettityypeistä ja suhteista tekee tietomallin määrittelijä. [Chen, 1976]

2.2. Relaatiotietomalli

Relaatiotietomalli kehitettiin vuonna 1969. [Codd, 1970] Se oli Coddin [1980] mukaan ensimmäinen tietomalli, joka kehitettiin jäsennellyn tiedon hallinnointiin ja kuvaamiseen. Coddin mukaan virheellisesti luullaan, että verkkotietomalli ja hierarkiatietomalli olisivat edeltäneet relaatiotietomallia. Virhekäsitys on todennäköisesti syntynyt siitä, että verkko- ja hierarkiatietomallipohjaisia *järjestelmiä* kehitettiin jo ennen 70-lukua, mutta kyseiset *tietomallit* esiteltiin vasta 1973. Aihe on sikäli tulkinnanvarainen, että verkkotietomallin implementaatio oli olemassa ennen relaatiotietomallia, mutta virallisesti tietomalli esiteltiin vasta myöhemmin vuonna 1969. Monien mielestä tämä lienee saivartelua, sillä esimerkiksi Harrington [2002] mainitsee ensimmäiseksi todelliseksi tietomalliksi hierarkiatietomallin, joka esiintyi kaupallisen tuotteen pohjana vuonna 1966. Verkkotietomallin historiaan ja määrittelyyn palataan kappaleessa 2.3. Voidaan kuitenkin todeta relaatiotietomallin olevan ensimmäisiä formalisoituja, tietokannan hallintajärjestelmien pohjaksi esiteltyjä tietomalleja. Coddin [1980] mukaan relaatiotietomalli eroaa verkko- ja hierarkiatietomalleista myös siinä, että relaatiopohjaiset järjestelmät tulivat vasta relaatiotietomallin esittelyn jälkeen, jolloin tietomallia pidettiin järjestelmäkehityksen pohjana.

Relaatiotietomalli oli merkittävä askel tietomallien kehityksessä. Matemaattiselta taustaltaan relaatiotietomalli perustuu joukko-oppista tuttuun binäärirelaatioon, joka on karteesisen tulon muodostaman määrittelyjoukkolistan osajoukko. [Ullman, 1982] Se nojaa vahvasti algebraan ja logiikkaan, ja näin ollen antoi tietomalleille matemaattisen pohjan. Relaatiotietomalli ja sen yhteydessä kehitetty, myöhemmin standardoitu, kyselykieli SQL, olivat myös merkittävä perusta koko tietokantakehitykselle ja -tutkimukselle. [Angles and Gutierrez, 2008]

Relaatiotietomalli koostuu relaatioista (relation), monikoista (tuple) ja attribuuteista (attribute). Relaatiot ovat monikkojoukkoja ja ne esitetään usein tauluina. Monikot kuvaavat entiteettityypin ilmentymiä, jotka yleensä vastaavat jotakin reaalimaailman ilmentymää. Attribuutit ovat entiteettityypin ominaisuuksia. Vastaavasti kuin relaatiot ovat tauluja, niin monikot ovat taulun rivejä ja attribuutit taulun sarakkeiden otsikoita. [Elmasri and Navathe, 2000] Relaatiot yhdistetään toisiinsa viiteavaimella, jolla viitataan yhdistettävän relaation pääavaimeen. Tämä muodostaa suhteen relaatioiden välille. Relaatiot voidaan myös ajatella relaatiotietomallin osakokonaisuuksina, jotka mallintavat kokonaisuuden tiettyä osaa.

Relaatiotietomalli nojaa vahvasti ennalta sovittuun malliin ja tiedon muotoon, jotka relaatiotietokannassa määritellään tietokannan kaaviossa (schema). Tämä luo vakaan pohjan tiedon varastoimiselle, kun tieto on aina sovitun mallin mukaan jäsenneltyä. Toisin sanoen relaatiotietomallin avulla voidaan saavuttaa korkean tason tiedon riippumattomuus. Toisaalta samaan aikaan relaatiotietomalli on suhteellisen kankea ja sen kyky ilmentää reaalimaailman käsitteellisiä suhteita on melko vajavainen. [Chen, 1976]

2.3. Verkkotietomalli

Ensimmäinen verkkotietomalliin pohjautuva järjestelmä esiteltiin vuonna 1967. Sen vastaanotto oli erittäin myönteinen, sillä se ratkaisi tiettyjä hierarkiatietomallissa havaittuja ongelmia ja rajoitteita. [Harrington, 2002] Verkkotietomallin juuret alkavat 1960-luvun alusta, jolloin Charles Bachman työskenteli General Electronicsilla ja suunnitteli tuotannonohjausjärjestelmää. Ratkaisuksi tähän järjestelmään Bachman kehitti integroidun tietovaraston (Integrated Data Store), joka oli ensimmäinen levypohjainen tietokannan hallintajärjestelmä, jolla oli proseduraalinen tiedon manipulointi- (DML) ja määrittelykieli (DDL). Myöhemmin 60-luvulla Conference on Data Systems Language (CODASYL) muodosti ryhmän, jonka tehtävänä oli kehittää standardoitu tietomalli tietokannan hallintajärjestelmä-ratkaisujen pohjalle. Tämä ryhmä oli nimeltään Data Base Task Group (DBTG). DBTG otti tietomallin kehittämisen pohjaksi Bachmanin kehittämän IDS:n ja formalisoivat sen. Vuonna 1971 DBTG julkaisi raportin, joka sisälsi tiedon manipulointi- ja määrittelykielet, joiden avulla verkkotietomalli standardoitiin. Tätä edelsi vuonna 1969 julkaistu alustava verkkotietomallin esittely. [Kruntorad, 2009]

Verkkotietomalli voidaan jakaa yksinkertaiseen (simple network data model) ja kompleksiseen (complex network data model) verkkotietomalliin. Yksinkertainen verkkotietomalli perustuu 1:n-suhteisiin, joissa emo-entiteetillä voi olla monta lapsi-entiteettiä. Verkkotietomallin etuina edeltäjäänsä hierarkiatietomallin verrattuna pidettiin erityisesti mahdollisuutta määritellä

entiteetille useampia kuin yksi emo-entieetti. Lisäksi verkkotietomalli erotti voimakkaammin loogisen ja fyysisen tason tietomallissa. [Harrington, 2002]

Kompleksisessa verkkotietomallissa sallitaan 1:n-suhteiden lisäksi myös n:m-suhteet, ilman ns. kooste-entiteetin (composite entity) luomista. Kompleksisen verkkotietomallin avulla pyrittiin luomaan verkkotietomalli, ilman yksinkertaisen verkkotietomallin rajoitteita. Tämän rajoitteen poistaminen kuitenkin johtaa tilanteeseen, jossa n:m-suhteille on hankala tallentaa tietoa, kun vastaavasti yksinkertaisessa verkkotietomallissa näiden välille luotiin kooste-entiteetti, joka mahdollistaa suhteeseen liittyvän tiedon tallentamisen. [Harrington, 2002]

2.4. Graafitietomalli

Graafitietomalli pohjautuu hyvin vahvasti verkkotietomalliin. Graafitietomalli on kuitenkin verkkotietomallia laajempi ja pitää sisällään abstraktiotason, joka verkkotietomallista puuttuu. Tämän ansiosta on helpompi eriyttää todellinen tietovaraston ilmentymä sen tietomallista. [Angles and Gutierrez, 2008] Graafi on lisäksi yksi keskeisimmistä tiedon abstraktioista tietojenkäsittelyssä. Graafi on myös hyvin luonnollinen tapa mallintaa maailmaa. Biologiastakin löytyy useita graafimaisia rakenteita, kuten aineenvaihdunnan verkosto tai geeniklusterit, joiden kuvaaminen graafilla on erityisen luonnollista. [Vicknair et al., 2010]

Graafitietomallipohjaiset tietokantaratkaisut sopivat parhaiten tilanteisiin, joissa tiedon yhteen liitettävyys ja topologia ovat tärkeämpiä tai yhtä tärkeitä, kuin tieto itsessään. Esimerkkeinä tällaisista sovellusaloista (domain) voisi olla esimerkiksi hyperteksti tai maantieteelliset sovellukset. Tällaisessa sovelluksissa yleensä tieto ja tiedon väliset yhteydet ovat samalla tasolla keskenään. Tämän tyyppisillä sovellusaloilla graafitietomalli tarjoaa lukuisia etuja tiedon mallinnukseen, joista tärkein on mahdollisuus kuvata tietoa luonnollisemmalla tavalla. Lisäksi kaikki tietuetta koskevat tiedot on mahdollista varastoida samaan solmuun (node/vertex), sekä esittää siihen liittyvät tiedot linkkien (arc/edge) ja niiden päästä löytyvien solmujen avulla. [Angles and Gutierrez, 2008]

Graafitietomallia on tieteellisissä julkaisuissa kuvattu monella eri tavalla. Useimmiten erot tavoissa kuvata graafitietomallia ovat hyvin pieniä, ja liittyvät lähinnä tapaan kuvata tiettyjä graafitietomallin ominaisuuksia. Tällaisia eroavaisuuksia kuvauksissa ovat esimerkiksi linkkien suunnan käyttö (directed graph) tai solmujen nimeäminen (labeled graph). Eroavaisuuksista riippumatta yksi olennainen ratkaistava ongelma yhdistää näitä kaikkia graafimallinnuksen tapoja, joka on loogisen ja fyysisen tietomallin erottaminen (schema vs. data). Useimmissa tapauksissa ero on selkeä. [Angles and Gutierrez, 2008]

Graafitietomalli mahdollistaa myös dataan kohdistuvien kyselyjen muodostamisen tuntematta tiedon rakennetta syvällisemmin. Kun esimerkiksi relaatiotietomallissa kyselyjä muodostettaessa on olennaista tuntea relaatiotietokannan taulujen suhde, joka on mahdollista tulkita tietomallin kaavioista (schema), voi graafitietomallissa solmu ja solmujen välinen suhde sisältää kaiken tarvittavan tiedon. Korkean tason abstrahoitujen kyselyjen muodostaminen on myös verrattain intuitiivista graafitietomallin eksplisiittisen rakenteen johdosta. [Angles and Gutierrez, 2008]

Verkkotietomalli ja graafitietomalli pohjautuvat graafiteoriaan, jonka esiasteen esitteli Leonhardt Euler vuonna 1736. Euler ratkaisi Königsbergin siltaongelman, joka on klassinen matemaattinen ongelma, jossa tavoitteena oli ylittää kaikki Königsbergin seitsemän siltaa täsmälleen kerran ja päätyä takaisin lähtöpisteeseen. Euler todisti tämän olevan mahdotonta, ja tällä todistuksella loi pohjan graafiteorialle. Eulerin tapa ratkaista ongelma oli se, josta graafiteorian alku syntyi. Euler piirsi ongelman graafiksi, ja totesi että siltojen ja maa-alueiden paikalla ei ollut merkitystä ongelman kannalta, kunhan siltojen lähtö ja päätepiste eivät muuttuneet. Näin Euler esitteli verkko- ja graafitietomallista tutut solmun ja linkin. Lisäksi Euler esitteli asteluku-käsitteen solmuille, joka tarkoittaa solmusta lähtevien linkkien lukumäärää. Ratkaisu itse ongelmaan on, että mikäli graafissa ei ole yhtään solmua, jonka asteluku on pariton, on ratkaisu mahdollinen. [Assad, 2007]

2.4.1. Graafitietomalli ja olio-ohjelmointi

Graafitietomallinnuksesta alettiin laajemmin keskustella kahdeksankymmentäluvulla olio-ohjelmoinnin myötä. Kuitenkin 90-luvulle siirryttäessä alaa valtasi muut tietomallit ja graafimallinnus jäi taka-alalle. Yhtenä tietomalliratkaisuna olio-ohjelmoinnin tueksi kehitettiin oliotietomalli (object-oriented db-model). Oliotietomalli sallii relaatiotietomalliin nähden huomattavasti moninaisemman tiedon rakenteen. Tämä antoi huomattavasti paremmat lähtökohdat tietovaraston suunnittelulle olio-pohjaisissa tietojärjestelmissä. Olennaisimpana erona oliotietomallissa graafitietomalliin on kuitenkin se, että oliomallin on relaatiotietomallin tapaan nojattava ennalta määriteltyyn tiedon muotoon (schema). Tämä vaikuttaa olennaisesti tiedon rakenteen muokattavuuteen. Lisäksi, vaikka oliotietomalli ja graafitietomalli molemmat hyödyntävät graafimaista rakennetta, ne mallintavat maailmaa eri tavoin. Oliotietomallissa maailmaa tarkastellaan kompleksisten olioiden kautta, joilla on kulloinkin voimassa oleva tila. Graafitietomalli sen sijaan näkee maailman yhteyksien verkostona, jossa painotetaan asioiden yhteyksiä ja näiden yhteyksien ominaisuuksia. [Angles and Gutierrez, 2008]

Yksi merkittävä asia tukee graafitietomallin tai oliotietomallin käyttöä olio-ohjelmointipohjaisissa tietojärjestelmissä. Tämä on tietokantatason entiteettityyppien yhdistäminen sovellustason luokkiin. Relaatiotietomalliin pohjautuvissa järjestelmissä on usein käytössä jonkinlainen luokka-relaatio-liitos. Tällaisia ovat esimerkiksi Javasta tuttu ORM (Object/Relational Mapping), jonka yksi tunnetuimmista ohjelmistokehyksistä on Hibernate. [Hibernate, 2018] Tällaisia liitoksia relaatiotietomallista mutkistaa se, että usein tietokantatason taulu ei vastaa sovellustason oliota, vaan se on kooste jostain taulujen liitoksista. Graafitietomallista tätä sovellustason ja tietokantatason välistä liitosta yksinkertaistaa huomattavasti se, että graafimalli on lähtökohtaisesti suoraan mallinnettavissa sovellustason luokiksi.

2.5. UML-mallinnus

Yksi yleisimmistä käsitteellisen mallintamisen notaatioista ER-mallin ohella on UML (Unified Modeling Language). UML on visuaalinen mallinnuskieli, jonka kehittivät Grady Booch, Jim

Rumbaugh ja Ivars Jacobson 90-luvun puolivälissä. Sen avulla voidaan mallintaa järjestelmän vaatimuksia, suunnitelmia ja toteutusta. UML-mallinnus nousi pian kehityksensä jälkeen käytännössä alalla vakioiduksi tavaksi jäsentää, kuvata, rakentaa ja dokumentoida tietojärjestelmiä. Tämän lisäksi Object Management Group (OMG) standardoi UML-mallinnuksen oliopohjaisen analyysin ja suunnittelun työvälineenä. UML-notaation avulla mallinnetaan käytännössä yhdeksää eri kaaviota. Nämä ovat luokkakaavio, käyttötapauskaavio, tilakaavio, aktiviteettikaavio, sekvenssikaavio, koostekaavio, oliokaavio, komponenttikaavio ja sijoittelukaavio. [Siau and Cao, 2001]

3. Relaatiotietokanta

Relaatiotietomallin esittelyn jälkeen 70-luvulla alettiin laajalti kehittämään erilaisia relaatiotietomalliin pohjautuvia tietokannan hallintajärjestelmiä. Ensimmäiset relaatiotietokannat julkaistiin 70-luvun lopulla, joita oli esimerkiksi Oracle ja IBM:n System R. Relaatiotietokantojen voittokulku johti myös siihen, että jotkut tahot alkoivat nimetä tietokantatuotteitaan käyttäen tuotenimessä sanaa "relaatio", vaikka tietokanta ei täyttäisikään relaatiotietokannan ominaisuuksia. Vuonna 1985 Codd määritteli 12 sääntöä, jotka tietokannan tulisi täyttää ollakseen aito relaatiotietokanta. [Codd, 1985] Näistä säännöistä voidaan kiteyttää, että tietokannan tulisi täyttää vähintään seuraavat kriteerit. Ensiksi, tietokannan tulee säilyttää kaikkea sisältämäänsä tietoa relaatioissa. Tarkoittaen, että kaikki tieto on tallennettuna taulun sarakkeissa ja ne ovat saatavilla sarakkeen nimen avulla. Toiseksi, kaikki tietokantaan kohdistuvat operaatiot, jotka ovat toteutettavissa käyttäjän ja järjestelmän tasolla ovat todellisia relaatio-operaatioita. Tällä tarkoitetaan, että kaikkien operaatioiden lähde on relaatio ja tulos on uusi relaatio. Kolmanneksi, järjestelmän tulee tukea vähintään yhtä muotoa JOIN-operaatiosta. [Elmasri and Navathe, 2000]

3.1. Tapahtumat ja ACID-malli

Relaatiotietomalliin pohjautuvat relaatiotietokannat useimmiten tukevat myös tapahtumia (transaction). Tapahtumat ovat tietokantaan kohdistuvia itsenäisiä operaatioketjuja, jotka suorittavat jonkin tietokantaan kohdistuvan toimenpiteen. Tapahtumat toimivat täysin erillään muista tapahtumista, joka mahdollistaa tiedon luotettavuuden virhetilanteiden sattuessa. Tapahtuman aikana myös tapahtumaan liittyvät taulujen rivit lukitaan tapahtuman ajaksi. Tällöin muut tapahtumat eivät pääse sotkemaan toista tapahtumaa. Tietokantatasolla tapahtuma päättyy tapahtuman hyväksymiseen (COMMIT), tai mikäli tapahtumassa tulee virheitä, hylkäämiseen (ROLLBACK). Käytännössä tietokannan hallintajärjestelmät pitävät lokia muutoksista, jotta palauttaminen (ROLLBACK) tapahtumaa edeltävään tilaan on mahdollista. [Hovi, 2011]

Oikeaoppinen tapahtuma täyttää tietyt ominaisuudet, joita "ACID"-johdonmukaisuusmalli kuvaa. ACID on akronyymi sanoista atomisuus (atomicity), eheyden säilyttäminen (consistency), eristävyys (isolation), pysyvyys (durability). [Hovi, 2011] Termillä viitataan tiedon johdonmukaiseen muokkaamiseen ja tietokannan tilan luotettavuuteen.

Atomisuus termillä tarkoitetaan tapahtuman sisältämien operaatioiden jakamattomuutta. Tämä on olennaista, jotta voidaan toteuttaa luotettavia tapahtumia, joiden sisältö perustuu useaan operaatioon. Esimerkiksi tilisiirrossa on olennaista, että toiselta tililtä rahaa poistuu ja toiselle tilille rahaa tulee lisää. Mikäli kumpi tahansa näistä operaatioista epäonnistuu, tulee tietokannan pystyä palautumaan tapahtuman perusteella oikein lähtötilanteeseen. [Msdn, 2017]

Eheyden säilyttämisellä viittaa tietokannan yhtenäisyyteen tapahtuman jälkeen. Edellä mainittua esimerkkiä hyödyntäen, tietokannan yhtenäisyys tarkoittaa, että kahden tilin välillä

tapahtuneen tilisiirron jälkeen tilien summa tulee olla sama kuin ennen siirtoa. Tällöin tietokanta on yhtenäisessä tilassa. [Msdn, 2017]

Eristävyys termillä tarkoitetaan tapahtumien eristäytyneisyyttä muista tapahtumista, sekä irrallisuutta tietokannan tilaan nähden. Joissain tilanteissa on tarpeellista, että tapahtuman sisältämien operaatioiden toteuttamat tilan muutokset tulevat näkyviin vasta tapahtuman suorituksen päätyttyä. [Msdn, 2017]

Pysyvyys viittaa tietokannan tilan kestävyyteen tapahtuman suorituksen jälkeen. [Msdn, 2017]

3.2. SQL-kyselykieli

SQL-kyselykieli on yksi suurimpia syitä relaatiotietokantojen suosioon. SQL-kielen kehittivät Donald Chamberlin ja Raymond Boyce [1974]. Kieli oli alun perin nimeltään SEQUEL, mutta nimettiin myöhemmin uudelleen SQL:ksi. Vuonna 1987 SQL-kielestä tuli standardoitu kieli [ISO/IEC 9075, 2018] relaatiotietokantojen käsittelyyn. Tämän jälkeen liikkuminen eri relaatiotietokantatuotteiden välillä oli huomattavasti aiempaa helpompaa. Standardoinnista huolimatta, eri tietokantatuotteiden toimittajat ovat lisänneet ominaisuuksia tuotteisiin, joita standardissa ei ole mukana. Näin ollen eroja tietokantatuotteiden välillä löytyy. Käytettäessä ainoastaan SQL:n standardoituja ominaisuuksia, on kuitenkin mahdollista tarvittaessa vaihtaa relaatiotietokanta toiseen ilman SQL-muutoksia. Lisäksi standardoinnin ansiosta SQL mahdollistaa usean eri relaatiotietokannan käytön samassa järjestelmässä, kun samat SQL-standardinmukaiset tietokantakyselyt on tulkittavissa eri tietokantatuotteilla. [Elmasri and Navathe, 2000]

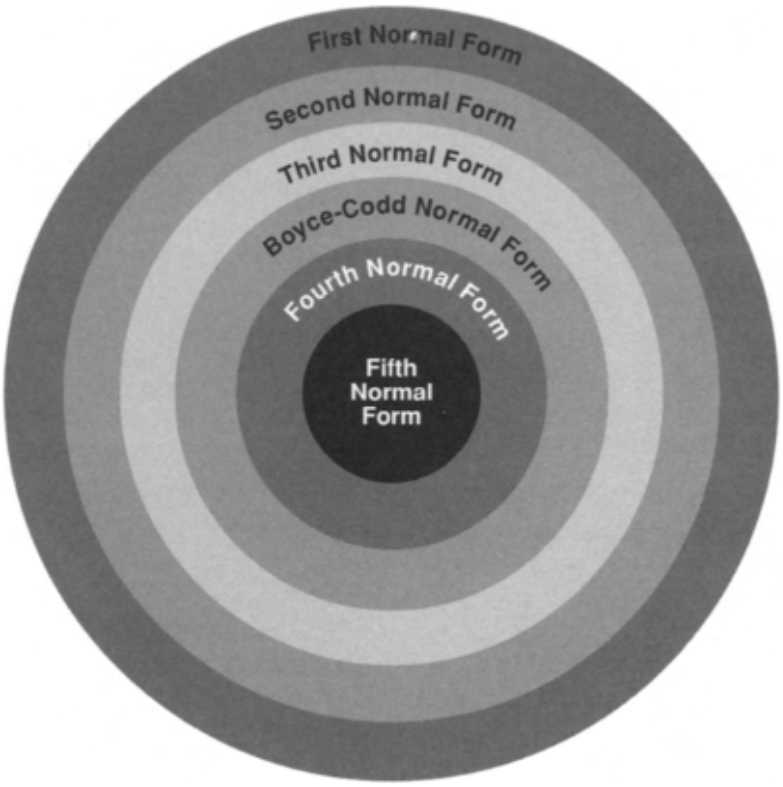
SQL on korkean tason kysely- ja manipulointikieli, jonka avulla on mahdollista sekä manipuloida, että määritellä relaatiotietokannan tietoja. Tästä syystä SQL on määritelmällisesti sekä tiedon määrittelykieli (data definition language, DDL), että tiedon manipulointikieli (data manipulation language, DML). [Elmasri and Navathe, 2000]

3.3. Tietokannan normalisointi ja normaalimuodot

Tietokannan normalisoinnissa tavoitteena on saada tietokanta mahdollisimman eheäksi, jotta vältytään ongelmilta joita huonosti suunniteltu relaatiotietokanta voi tuoda mukanaan. Erityisesti normalisoinnilla pyritään vähentämään tiedon toistuvuutta (redundancy), joka voi johtaa poikkeavuuksiin ja virhetilanteisiin myöhemmin tietokannan käytössä. [Eessaar, 2016] Normalisointia on mahdollista tehdä suunnittelun eri vaiheissa, mutta usein relaatiomalli luodaan hyvin suunnitellun ER-kaavion pohjalta, jolloin kaavio on korkeassa normaalimuodossa hyvin toteutetun ER-mallin ansiosta. On kuitenkin myös mahdollista suunnitella relaatiomalli suoraan ilman ER-kaaviota, jolloin on tärkeää seurata hyvän tietokantasuunnittelun malleja suunnittelun edetessä. [Harrington, 2002]

Relaatiotietokantojen suunnittelussa normalisoinnin teoreettisena apuvälineenä käytetään normaalimuotoja (normal forms). Kukin normaalimuoto sisältää tietyn joukon sääntöjä, joiden on täytyttävä, jotta tietokannan voidaan sanoa olevan kyseisessä normaalimuodossa. Normaalimuotoja

on määritelty olevan kuusi kappaletta. Ne ovat hierarkkisia, joten ylemmän tason normaalimuodon sääntöjen täyttyessä, myös alempien normaalimuotojen säännöt täyttyvät. (Kuva 1) Käytännössä useimpiin tapauksiin riittää, että tietokanta täyttää kolmannen normaalimuodon säännöt, jolloin vältytään suurimmalta osalta yleisimmin esiintyvistä huonon suunnittelun mukanaan tuomista tietokantaongelmista. Boyce-Codd ja neljäs normaalimuoto käsittelevät lähinnä erityistapauksia, eivätkä nämä normaalimuodon tasot näin ollen ole useimmissa tapauksissa tavoittelemisen arvoisia. [Harrington, 2002]



Kuva 1. Normaalimuotojen hierarkia [Harrington, 2002]

Ensimmäisen normaalimuodon (1NF) säännöt täyttyvät, kun tietokannan taulut ovat kaksiulotteisia ja eivät sisällä toistuvia ryhmiä (repeating groups). Toisin sanoen taulujen kaikki attribuutit ovat atomisia, eli yksiarvoisia. Mikäli tietokantamalli sisältää sarakkeita, joissa on mahdollista esiintyä esimerkiksi pilkkueroteltuna n-määrä arvoja, niin tulee tämä sarake siirtää omaan tauluunsa ja ylläpitää arvoja omina riveinään uudessa taulussa. Tällä tavoin saadaan normalisoitua tietokanta ensimmäiseen normaalimuotoon. [Harrington, 2002]

Toinen normaalimuoto (2NF) koostuu ensimmäisen normaalimuodon sääntöjen lisäksi seuraavasta säännöstä: ”Kaikki ei-avainattribuutit ovat funktionaalisesti riippuvaisia kokonaisesta avainattribuutista.” Funktionaalisesti riippuvaisella tarkoitetaan sitä, että millä tahansa hetkellä relaatiossa avainattribuutista johdettu tietty arvo viittaa vain yhteen tietystä sarakkeesta löytyvään arvoon. Esimerkiksi asiakasnumero viittaa vain yhteen tiettyyn etunimeen ja sukunimeen. Vaikkakin nämä saattavat muuttua, niin viittaus on silti olemassa vain yhteen tiettyyn sarakkeeseen. [Harrington, 2002]

Kolmas normaalimuoto (3NF) sisältää edellisten lisäksi säännön, relaatio ei sisällä transitiivisia riippuvuuksia (transitive dependencies). Transitiivisella riippuvuudella tarkoitetaan riippuvuutta, jossa A on riippuvainen B:stä ja B on riippuvainen C:stä, näin ollen A on riippuvainen C:stä. (A->B ja B->C, siis A->C) Tällainen tilanne ei siis ole sallittu, jotta tietokanta täyttää kolmannen normaalimuodon vaatimukset. Mikäli tällainen transitiivinen riippuvuus on havaittavissa, niin tulee relaatiota purkaa edelleen pienempiin tauluihin, jotta riippuvuus saadaan purettua. [Harrington, 2002]

Boyce-Codd normaalimuoto (BCNF) on käytännössä hieman tiukennettu kolmas normaalimuoto. Tällä on pyritty ratkaisemaan tiettyjä erikoistapauksia, joita ilmenee toisinaan. Useimmiten kuitenkin relaation ollessa kolmannessa normaalimuodossa, on valtaosa relaation eheyteen liittyvistä ongelmista ratkaistu ja voidaan olla tyytyväisiä suunnittelun tasoon. Boyce-Coddin normaalimuodon määritelmän mukaan, X -> A on triviaali funktionaalinen riippuvuus, tai X on superavain relaatiolle R. Superavaimella tarkoitetaan sellaista attribuuttia tai attribuuttijoukkoa, joka on relaatiossa arvoiltaan uniikki. Toisin sanoen tätä voidaan käyttää tunnistamaan relaatiotaulun rivi, eli se toimii kuin taulun avain. [Ullman, 1982]

Jotta relaatio olisi neljännessä normaalimuodossa (4NF), tulee relaation täyttää Boyce-Coddin normaalimuodon sääntöjen lisäksi sääntö, jonka mukaan relaatiossa ei ole moniarvoisia riippuvuuksia. Moniarvoinen riippuvuus on olemassa, kun jokaiselle attribuutin A arvolle on olemassa joukko attribuutin B arvoja, jotka liittyvät siihen, sekä joukko attribuutin C arvoja, jotka liittyvät siihen. Lisäksi B ja C ovat riippumattomia toisistaan. Tästä esimerkkinä käy elokuvainfo-niminen relaatio, joka sisältää attribuutit otsikko, näyttelijä, tuottaja. Otsikkoon (A) liittyy joukko näyttelijöitä (B), otsikkoon (A) liittyy joukko tuottajia (C), ja näyttelijät (B) ja tuottajat (C) eivät liity toisiinsa. [Harrington, 2002]

Viides normaalimuoto (5NF) jakaa relaation neljättä pienempiin osiin, mutta sitä ei pidetä yleisesti kovinkaan merkittävänä. Käytännössä suurin osa relaatioista on hyvin suunniteltuja jo ollessaan kolmannessa tai neljännessä normaalimuodossa.

3.4. Kritiikkiä relaatiotietokannoista

Riippumatta tietojärjestelmän luonteesta, relaatiotietokanta on jo vuosikymmenten ajan ollut monille tietojärjestelmäkehittäjille oletusvalinta tietovarastoksi. Kuitenkin tietojärjestelmien monimutkaistuessa ja tiedon määrän kasvaessa on skaalautuvuuden ja suorituskyvyn vuoksi tietokanta- ja tietojärjestelmäarkkitehtuuriratkaisuja mietittävä usein huomattavasti tarkemmin kuin ennen. Esimerkiksi Big Datasta puhuttaessa, on tietokantaratkaisut hyvin harvoin puhtaasti relaatiopohjaisia. Tiedon ja kyselyjen määrän kasvaessa turvaudutaan usein hajautettuihin tietovarastoihin, joissa tietoa jaetaan eri palvelimille. On kuitenkin erittäin vaikea saavuttaa korkeaa skaalautuvuutta uhraamatta tiedon eheyttä (consistency/durability). Hajautetussa tietokannassa on mahdollista, että tapahtuma suoritetaan loppuun, mutta tieto ei ole saavutettavissa kaikilla käyttäjillä samanaikaisesti. Vastaavasti ACID-pohjaisissa tietokannoissa on mahdoton taata täysi

saatavuus tietoverkkohäiriön sattuessa. Tämä johtuu siitä, että ACID-pohjaisia tietokantoja on vaikea hajauttaa, rikkomatta samalla ACID-periaatetta. [Wu et al., 2014]

Vicknair ja muut [2010] esittivät listan selkeästi tunnistettavissa olevia seikkoja, jotka viittaavat siihen, että olisi syytä pohtia relaatiotietokantaratkaisulle myös vaihtoehtoista toteutustapaa. Tällaisia seikkoja ovat esimerkiksi se, että tietokannasta löytyy paljon tauluja, joilla on runsaasti sarakkeita, joita ei suurimmalla osalla riveistä käytetä lainkaan. Tämä lisää tietokannan yleistä kuormittavuutta (overhead). Toinen tunnistettava seikka on attribuuttitaulut. Tällaisia ovat relaatiotietokantaan lisätyt taulut, joissa rivillä on vain viiteavain toisen taulun riviin, attribuutin nimi, sekä attribuutin arvo. Toisin sanoen taulut, joiden monikot rikastavat toisen taulun monikkojen tietoja. Mikäli tällaisia tauluja tarvitaan usein kyselyiden liitoksena JOIN-operaation avulla, kuormittavat nämä tietokantaa ja hidastavat kyselyitä. Relaatiotietokannassa saattaa esiintyä myös lukuisia monen-suhde-moneen taulujen suhteita, jotka vastaavasti kuormittavat tietokantaa erityisesti kyselypuolella liitosoperaatioiden takia. Tällaisia monen-suhde-moneen-suhteita puretaan usein luomalla suhteen väliin uusi taulu, jolloin luodusta taulusta on yhden-suhde-moneen-suhde molempiin alkuperäisiin tauluihin. Tällöin taulun tietoja koskevissa kyselyissä joudutaan tekemään useita liitoksia. Neljäs tunnistettava seikka on hierarkkiset rakenteet, joista tässä tutkielmassa on mainittu jo aiemminkin. Tämä on hyvin olennainen piirre, sillä hierarkiat mallintuvat huonosti relaatiossa ja ovat lisäksi erittäin raskaita kyselytasolla. Lopuksi yksi kehittäjän näkökulmasta helposti tunnistettava asia on usein toistuvat kaavion muutokset. Tämä lisää yleistä kuormittavuutta tietojärjestelmän kehittämisen näkökulmasta.

Relaatiotietokannoissa ongelmaksi saattaa muodostua myös vaatimus tiedon ennakoitavuudesta. Tällä tarkoitetaan sitä, että kun tietokantaan lisätään uutta tietoa, sen on oltava suunnitellun kaavion mukaista. Muutoin joudutaan muuttamaan kaaviota, eli käytännössä taulujen rakennetta, joka on raskas prosessi. Tämän lisäksi sama ennakoitavuus vaaditaan kyselyissä. Relaatiotietokantaa suunniteltaessa tulisi hyvissä ajoin tietää, mikäli tietokantaan aiotaan tehdä tietynlaisia vaativia kyselyjä, jotta kyselyihin voidaan varautua jo tietokannan rakenteessa.

4. NoSQL

Termiä NoSQL (Not Only SQL) on käytetty ensimmäisen kerran jo 1998, Carlo Strozzin kehittämässä pienessä tietokannanhallintajärjestelmässä, jonka hän nimesi “Strozzi NoSQL”:ksi. Strozzi NoSQL erosi relaatiotietokannoista siinä, että se ei käyttänyt SQL-kieltä tiedon hallinnointiin. [Kuznetsov and Poskonin, 2014] Myöhemmin termillä pyrittiin kuvaamaan uuden tyyppisiä tietovarastomalleja, jotka eivät seuranneet perinteistä relaatiotietomallia ja näin ollen eivät tukeneet perinteistä SQL-kyselykieltä. Määritelmän mukaisia varteenotettavia tietokantaimplementaatioita kuitenkin syntyi vasta 2000-luvulla, mistä lähtien NoSQL-tyyppisten tietovarastomallien, ja erityisesti uusien tietokantaimplementaatioiden kehitys on ollut nopeaa. Tästä johtuen termi NoSQL voidaan tänä päivänä mieltää hyvinkin laajasti ja se usein määrittelee enemmän, minkä tyyppinen jokin tietokantamalli ei ole, kuin millainen se on. [Mpinda et al., 2015]

NoSQL tietovarastot voidaan jakaa tietomalliensa mukaisiin pääkategorioihin. Nämä ovat avain-arvoparivarastot (key-value stores), dokumenttivarastot (document stores), laajennettava varastot/sarakeperhevarastot (extensible record stores/wide column stores) ja tuoreempana lisänä graafivarastot (graph stores). [Kuznetsov and Poskonin, 2014] Näiden lisäksi NoSQL tietovarastomalleiksi voidaan lukea harvemmin esillä olevat, monimallivarastot (multimodel stores), oliovarastot (object stores), moniulotteiset varastot (multidimensional stores), sekä XML-varastot (XML stores). [NOSQL databases, 2017]

4.1. BASE-malli

BASE-malli on NoSQL-tietovarastojen vastine relaatiotietokantojen ACID-mallille. BASE on akronyymi sanoista Basically Available, Soft state ja Eventual consistency. Käytännössä BASE-mallilla tarkoitetaan sitä, että NoSQL-tietovarastot tukevat näitä edellä mainittuja akronyymin mukaisia luonnehdintoja. [Chandra, 2015]

Basically Available tarkoittaa, että järjestelmä vaikuttaa toimivan. Tämä perustuu NoSQL-järjestelmissä yleisesti käytössä olevaan tiedon replikointiin, jolloin päästään siihen, että järjestelmä vastaa aina, vaikka osia sen arkkitehtuurista olisikin toimintakyvyttömänä. CAP-teoreeman (esitellään seuraavassa kappaleessa) pohjalta tällä tarkoitetaan sitä, että järjestelmä takaa saatavuuden (Availability). [Chandra, 2015]

Soft state viittaa järjestelmän vakiintumattomaan tilaan, eli tietoihin, jotka sillä hetkellä on saatavilla. Soft state-tilassa olevat tiedot saattavat muuttua, ellei niitä tarvittaessa päivitetä uudelleen. Tällä viitataan käytännössä esimerkiksi välimuistissa oleviin tietoihin, kuten session tiedot. Soft state-tilaa käyttämällä voidaan optimoida järjestelmää käyttämällä uudestaan tietoja, jotka ovat jo tiedossa. [Chandra, 2015]

Eventually consistent termillä viitataan tietojärjestelmän vääjäämättömään johdonmukaisuuteen. Tilanne tulee esiin erityisesti hajautetun järjestelmän tapauksessa, jolloin tietoja muutettaessa muutokset eivät välttämättä ole saavuttaneet kaikkia eri palvelimia. Tällöin on

tilanne, jossa tiedot kaikilla eri palvelimilla eivät vastaa toisiaan. Eventually consistent viittaa tämänkaltaiseen tilanteeseen, jolloin tiedot ovat lopulta johdonmukaisia, kun muutettava tieto on talletettu kaikille hajautetun järjestelmän eri palvelimille. [Chandra, 2015]

4.2. CAP-teoreema

Hajautetut tietokannat ovat nykyisin usein käytetty ratkaisu data-keskeisten ja korkean skaalautuvuuden vaativien tietojärjestelmien (Big Data) kehityksessä. Hajautetut järjestelmät tarjoavat mahdollisuuden korkeiden datamäärien käsittelyyn, pitäen samalla järjestelmän vasteajat lyhyinä. Tietovarastojen hajautus tuo kuitenkin mukanaan myös ongelmia. Näistä ongelmista varmasti keskeisin on tiedon replikoinnin ja hajautuksen mukanaan tuoma ongelma tiedon johdonmukaisuudesta eri palvelinten välillä, sekä näiden tietojen hallinnointi laite- tai verkkovirheiden sattuessa. Ongelma on hahmotettavissa hyvin CAP-teoreeman kautta. [Kleppmann, 2015]

2000-luvun alussa Fox ja Brewer esittelivät CAP-periaatteen (CAP Principle), josta myöhemmin, Gilbertin ja Lynchin formalisoinnin jälkeen tuli CAP-teoreema (CAP Theorem). Teoreeman mukaan data-keskeiset järjestelmät eivät kykene jatkuvasti säilyttämään kolmea keskeistä tietovaraston luonnehdintaa, jotka ovat tiedon johdonmukaisuus (**Consistency**), saatavuus (**Availability**) ja hajautuksen sieto (**Partition tolerance**). Toisin sanoen, järjestelmille tulee hetkiä, jolloin näistä kolmesta luonnehdinnasta vain kaksi pätee järjestelmän tilaa kuvailtaessa. Käytännössä tämä tarkoittaa sitä, että mikäli kyseessä ei ole hajautettu järjestelmä, niin hajautuksen sieto ei ole voimassa lainkaan. Tällöin CAP-teoreeman mukaan järjestelmä on johdonmukainen ja saatavuus riippuu järjestelmän tilasta. Tilanteessa verkko- tai laitehäiriöihin ei varsinaisesti ole varauduttu, ja järjestelmän saatavuus saattaa olla uhattuna. [Kleppmann, 2015]

Hajautetun järjestelmän yhteydessä CAP-teoreeman avulla kuvataan ongelmaa, jossa verkkohäiriön sattuessa on valittava tiedon johdonmukaisuuden ja saatavuuden väliltä. Tällä tarkoitetaan sitä, että verkon tai laitteen pettäessä kaikki hajautetun järjestelmän tiedot eivät ole saatavilla. Tällöin on valittava vastaako järjestelmä saatavilla olevan tiedon mukaan, vai odottaako se, että kaikki tieto on saatavilla ja vastaa vasta sitten varmasti oikealla tiedolla. Tällaisessa tilanteessa CAP-teoreeman mukaan P on aina voimassa, ja on valittava, tukeeko järjestelmä CP vai AP mukaisia luonnehdintoja. [Shi, 2014]

CAP-teoreemasta on esitetty myös jonkin verran kritiikkiä. Kleppmann [2015] paheksuu CAP-teoreeman kritiikissään erityisesti Gilbertin ja Lynchin esittämän tiedon saatavuuden (Availability) ylimalkaisuutta. Kleppmannin mukaan tulisi saatavuuden sijaan puhua latenssista. Tällöin määritelmä olisi intuitiivisempi ja tukisi paremmin käsitystä tiedon kulusta verkossa, jossa tietoa joutuu aina odottamaan, mutta kyse on koetusta odotuksen pituudesta. Kleppmann esittääkin CAP-teoreeman korvaajaksi hajautettujen tietojärjestelmien kritisoinnin työkaluna odotus-herkkyys-viitekehystä. (delay-sensitivity)

4.3. Kritiikkiä NoSQL:stä

NoSQL-tietovarastot on kehitetty ratkaisemaan monia relaatiotietokantojen ongelmia, mutta on myös paljon asioita, joissa relaatiotietokannat puolustavat paikkaansa. Yhtenä suurimmista relaatiotietokantojen eduista voidaan mainita standardoitu kyselykieli SQL. Vaikka monet NoSQL-tietovarastot tukevatkin SQL-kyselykieltä, niin ei näille ole olemassa vastaavaa standardoitua kieltä. Lisäksi jokainen NoSQL-tuote on itsenäinen ja ne toteuttavat asiat omilla tavoillaan, jolloin niiltä puuttuu relaatiotietokantojen kaltainen ennustettavuus ja ristiinsopivuus. Tämä on toki ymmärrettävää, ottaen huomioon, että relaatiotietokantoja on ollut yli 40 vuotta, kun NoSQL-trendi on vasta alle kymmenen vuoden ikäinen (tästä on lähteestä riippuen hieman eri tulkintoja). Tämä on siis toki nähtävä myös omalla tavallaan relaatiotietokantojen etuna. [Chandra, 2015]

5. Graafitietokanta

Kuten jo verkko- ja graafitietomallia käsittelevissä luvuissa on todettu, niin graafitietokannan yksi olennainen etu relaatiotietokantaan verrattuna on sen kyky sitoa tietoa entiteettien välisiin suhteisiin tehokkaasti. Tästä on johdettavissa monenlaisia käyttötapauksia, jotka voivat hyödyntää tätä tietokannan ominaisuutta. Esimerkiksi Neo4j markkinoi tuotettaan internetsivuillaan [Neo4j, 2018c] tähän pohjautuvien käyttötapausten avulla. Yksi näistä käyttötapauksista on väärinkäytösten tunnistus (Fraud detection). Tämä on hyvä esimerkki siitä, kuinka graafitietokannat ovat hyvä tuki myös tiedonlouhinnassa ja koneoppimisessa.

Vuosituhannen vaihteen NoSQL-trendin alusta alkaen NoSQL-tietokantoja on tullut markkinoille satoja. Samaan aikaan myös graafitietokantojen määrä on lisääntynyt. Tällä hetkellä ehkä merkittävimpiä graafitietokantatuotteita ovat Neo4j, DataStax Enterprise Graph (DSE Graph), OrientDB ja ArangoDB. Näistä Neo4j on ainoa puhtaasti graafitietokanta, sillä OrientDB ja ArangoDB ovat ns. monimallitietokantoja (multi-model database) ja DSE graph käyttää tietovarastonaan Cassandra-tietokantaa. [DataStax, 2018] Monimallitietokannoilla tarkoitetaan sitä, että ne tarjoavat tietokantaratkaisuja samaan aikaan monelle eri tietomallille. OrientDB tarjoaa tuen dokumentti, avain-arvo-pari ja oliopohjaisille tietomalleille. [OrientDB, 2018c] ArangoDB tukee graafin lisäksi avain-arvo-pari ja dokumenttipohjaisia tietomalleja. [ArangoDB, 2018] Neo4j käyttää tiedon tallennukseen ennalta määritellyn kokoisia listoja (array), joihin on tallennettu mm. solmua koskevat tiedot ja suhteet. [Bitnine, 2018]

Neo4j markkinoi tuotettaan myös hieman muiden NoSQL-tuotteiden kustannuksella todeten graafitietokantojen olevan ainoa ratkaisu tietotyyppien välisten suhteiden tehokkaaseen käsittelyyn. NoSQL-tietovarastot tallentavat usein joukkoja itsenäisiä dokumentteja tai avain-arvo-pareja. Lukunopeuksiltaan tällaiset tietovarastot ovat usein relaatiotietokantoja tehokkaampia. Näiltä kuitenkin puuttuu aito tietojen suhteiden käsittely, jossa saataisiin tietotyyppien välille eksplisiittisiä suhteita ja näille attribuutteja. Monet NoSQL-tietovarastojen päälle kehittävät tietojärjestelmäkehittäjät luovatkin keinotekoisia hierarkian ja suhteiden mallinnuksia, tallentamalla niin sanottuja koostetietoja tietueeseen. Graafitietokannat ratkaisevat suhteita ja hierarkioita koskevat suorituskykyhaasteet hyvin pitkälti sisäänrakennetusti, mikä on painava syy valita graafitietokanta hierarkioita käsitteleviin tietojärjestelmiin. [Neo4j, 2018d]

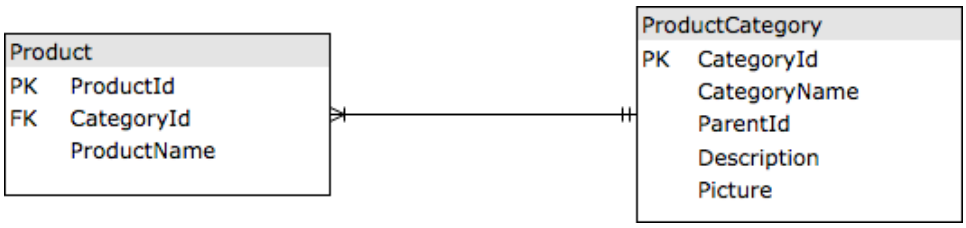
5.1. Graafikyselyt

Graafitietokannat eroavat olennaisesti relaatiotietokannoista muun muassa siinä, että graafitietokannoille ei ole yhtä standardoitua kyselykieltä, jota kaikki graafitietokannat tukisivat. Graafikyselykieliä on graafitietokantojen kehityksen mukana tullut lukuisia. Näistä suurimpina voidaan mainita Neo4j:n kehittäneen Neo Technologyn kehittämä Cypher, W3C:n kehittämä SPARQL, sekä Apachen Gremlin.

Graafikyselykielet eroavat toisistaan hieman niiden hakutoiminnallisuuksiensa painotuksien, sekä syntaksin suhteen. Graafikyselykielissä on tunnistettavissa kaksi ydinominaisuutta, joita ovat hahmonsovitus (pattern matching) ja navigaatio (navigation). Nämä kaksi kyselyn tapaa ovat graafikyselyiden ytimessä. Edellä mainituista graafikyselykielistä Cypher on vahvimmin hahmonsovitukseen nojaava kyselykieli, kun taas Gremlin on enemmän polkuorientoitunut kyselykieli. [Angles et al., 2016]

Graafikyselyissä on myös mahdollista toteuttaa merkityksellisiä kyselyjä tuntematta tietomallin rakennetta kokonaisvaltaisesti. Tällä tarkoitetaan esimerkiksi sitä, että graafikyselyssä voidaan toteuttaa aligraafi- tai polkukysely, vaikka tietomallin rakenne olisi tuntematon. Vastaavasti SQL-kysely relaatiotietokantaan vaatii relaatioiden yhteyden tuntemuksen, jotta kyselyä voidaan ylipäätään muodostaa. [Angles et al., 2016]

Neo4j esittelee [Neo4j, 2018g, 2018] esimerkin SQL-kyselyn ja graafikyselyn eroista, jossa tietynlaiseen kyselyyn joudutaan SQL-kyselyssä tekemään tietomallista riippuen yhdestä useampaan JOIN/UNION-operaatiota. Tämänkaltaiset taulujen liitokset lisäävät huomattavasti kuormaa tietokantapalvelimella ja hidastavat kyselyjen suorituskykyä. Neo4j:n mukaan tämänkaltainen selkeys ja intuitiivisuus Cypherilla tehdyissä graafikyselyissä jättävät kehittäjille aikaa keskittyä paremmin kyselyn lopputulokseen ja seuraaviin askeliin, sekä vähentää kyselyihin kohdistuvien virheiden etsintään käytettävää aikaa.



Kuva 2. Tuotekategorian relaatiokaavio

Kuvassa 2 on esitelty relaatiokaavio, johon koodiesimerkkien 1 ja 2 SQL- ja Cypher-kyselyt pohjautuvat. Relaatiokaaviossa on kaksi taulua, Product ja ProductCategory, joista ProductCategory on hierarkkinen. Käytännössä tämä tarkoittaa sitä, että tuotekategorialla voi olla alituotekategorioita. Esimerkiksi maitotuotteiden alikategoria voisi olla jugurttituotteet.

```
SELECT Product.ProductName
FROM Product
JOIN ProductCategory pc ON (Product.CategoryID = pc.CategoryID AND pc.CategoryName =
"Dairy Products")
JOIN ProductCategory pc1 ON (Product.CategoryID = pc1.CategoryID)
JOIN ProductCategory pc2 ON (pc1.ParentID = pc2.CategoryID AND pc2.CategoryName = "Dairy
Products")
JOIN ProductCategory pc3 ON (Product.CategoryID = pc3.CategoryID)
```

```
JOIN ProductCategory pc4 ON (pc3.ParentID = pc4.CategoryID)
JOIN ProductCategory pc5 ON (pc4.ParentID = pc5.CategoryID AND pc5.CategoryName = "Dairy
Products");
```

Koodiesimerkki 1. SQL-kysely tuotekategoriahierarkiasta [Neo4j, 2018b]

Koodiesimerkissä 1 Product-taulusta haetaan maitotuotteisiin (Dairy Products) kuuluvat tuotteet, kolmessa tasossa. Eli ensin etsitään suoraan maitotuotteet-kategoriaan kuuluvat tuotteet, jonka jälkeen yhdistetään kyselyyn myös tuotteet, joiden ylemmän tason kategoria on maitotuotteet. Näitä liitoksia tehdään kolmessa kategoriatasossa.

Erityinen vahvuus graafitietokannoissa ja niihin kohdistuvissa kyselyissä, sekä näiden suorituskyvyssä, löytyy niiden kyvystä käsitellä hierarkioita ja polkuja. Relaatiotietokantoja käytettäessä hierarkiat joudutaan useimmiten käsittelemään JOIN-operaattoreita käyttäen (Koodiesimerkki 1). Tällaiset kyselyt ovat suorituskyvyltään raskaita ja kyselyn rakenne saattaa olla vaikeasti hahmotettavissa. Cypherin tarjoamalla kyselysyntaksilla tämänkaltaiset kyselyt on mahdollista kirjoittaa SQL:ään verrattuna huomattavasti lyhyemmin (Koodiesimerkki 2). Lisäksi graafitietokanta suorittaa usein tämän kyselyn murto-osassa ajasta verrattuna relaatiotietokantaan. [Neo4j, 2018b]

```
MATCH (p:Product)-[:CATEGORY]->(l:ProductCategory)-[:PARENT*0..]-(:ProductCategory
{name:"Dairy Products"})
RETURN p.name
```

Koodiesimerkki 2. Cypher-kysely tuotekategoriahierarkiasta [Neo4j, 2018b]

Koodiesimerkin 2 Cypher-kyselyssä on mallinnettu koodiesimerkin 1 SQL-kyselyä vastaava kysely graafitietokantakyselynä. Käytännössä kyselyssä tehdään haku, jolla haetaan Product-solmut, jotka liittyvät ProductCategory-solmuun suhteen CATEGORY kautta. Tätä tarkennetaan vielä sillä, että ProductCategory-solmu voi liittyä toiseen ProductCategory-solmuun suhteen PARENT kautta ((p:Product)-[:CATEGORY]->(l:ProductCategory)). PARENT-suhteelle on kyselyssä annettu lisärajoite, että PARENT-suhteissa huomioidaan 0-n tasoiset isä-lapsi-suhteet ([:PARENT*0..]). Lopuksi näiden suhteiden kautta löytyvien ProductCategory-solmujen nimi on oltava ”Dairy Products” (:ProductCategory {name:"Dairy Products"}).

Kyselyissä selvästi löydettävissä jonkin verran eroavaisuuksia. Cypher-kysely on merkittävästi SQL-kyselyä lyhyempi. Tämä johtuu JOIN-liitosoperaatioista, joita relaatiotietokannassa on tällaisessa tapauksessa eksplisiittisesti kirjoitettava haluttujen hierakiatasojen määrän mukaan. Tähän ongelmaan on joillakin tietokantatuotteilla tarjolla syntaktista helpotusta (kuten Oraclen CONNECT BY), mutta sisäisesti ne toteuttavat kuitenkin edelleen saman liitoksen. Toisaalta SQL-kysely on relaatiotietomallin tunteville ja erityisesti SQL-

kyselyjä tehneille sisällöltään selkeä. On kuitenkin melko kiistatonta, että tämänkaltaisessa hierarkiakyselyssä graafitietokantaan kohdistuvat kyselyt tuntuvat SQL-kyselyjä luonnollisemmilta.

5.1.1. Suorituskyky

Relaatiotietokannan suorituskykyä on mahdollista kasvattaa monin tavoin. Yksi yleinen tapa tehostaa kyselyiden suorituskykyä on lisätä tauluille käänteisindeksejä (inverted index). Käänteisindeksit ovat tietorakenteita, jotka ylläpitävät tietoa dokumenteista tai riveistä, joista jokin tieto löytyy. Useimmiten käänteisindeksejä hyödynnetään luonnollisen kielen tekstihauissa, joissa käänteisindeksiin tallennetaan lista dokumenttien tunnisteista (id), joista tietty termi löytyy. [Rae et. al., 2014] Näiden avulla tietokantakyselyt tehostuvat, kun kyselynprosessoinnissa ei tarvitse käydä läpi koko taulua löytääkseen etsityt rivit. Kääntöpuolena indeksoinnissa on manipulointilauseiden (DML), kuten INSERT, UPDATE, DELETE hidastuminen, kun tietoa on kirjoitettava useampaan paikkaan. Usein indeksointia tehdään suoraan tietokantatasolla, jolloin on tärkeää mitoittaa indeksien määrä niin, että kyselyt tehostuvat, mutta manipulointilauseiden suoritus ei hidastu liikaa.

Indeksointia on mahdollista tehdä myös tietokannan ulkopuolelle, toiseen tietovarastoon. Tällöin kyse on useimmiten tiedon denormalisoinnista, jossa tavoitteena on keventää tiedon rakennetta, tiedon normalisoinnin kustannuksella. [Sanders and Shin, 2001] Yksi yleinen tapa erityisesti tekstihakuja käytettäessä on indeksoida tietokannasta tietoa esimerkiksi Apache Lucene hakukirjastoon. Tunnettuja Luceneen pohjautuvia hakumoottoreita ovat myös Apache Solr ja Elasticsearch, jotka ovat myös NoSQL-tietovarastoja. Näiden avulla on mahdollista tehostaa luonnollisen kielen hakujen suorituskykyä erittäin paljon. Kääntöpuolena tässä sen sijaan on saman tiedon ylläpito kahdessa paikassa. Tämä kasvattaa virheiden mahdollisuutta, eikä järjestelmä enää tue ACID-mallia.

Luonnollisten kielten haut saattavat olla relaatiotietokannoille melko raskaita, vaikka tietokantaan olisikin implementoitu indeksejä tehostamaan kyselyjä. Jopa kaupallisten relaatiotietotietokantojen kehittäjien on todettu olevan usein sitä mieltä, että tietokannan rinnalle olisi hyvä ottaa käyttöön erillinen tekstihakumoottori. [Rae et. al., 2014] Graafitietokannoista ainakin Neo4j käyttää sisäisesti hyväkseen Apache Lucene hakukirjastoa, joka mahdollistaa vasteajoiltaan suorituskykyiset tekstihaut suoraan tietokannasta. [Vicknair et al., 2010]

Relaatiotietokannoissa suurin ongelma suorituskyvyn kannalta on JOIN-operaatioiden tarve SQL-kyselyissä. Relaatiotietomallissa tieto jakaantuu eri relaatioihin pää- ja vierasavainten muodostamien liitosten avulla. Näiden liitosten läpikäyminen kyselyä tehtäessä on erittäin raskas operaatio. Operaatioita voidaan keventää luomalla relaatiotietokannan tauluille indeksejä, joista halutut rivit löytyvät nopeammin indeksien perusteella. Tietomäärän kasvaessa, tulee suorituskyvyssä raja vastaan myös indeksihakuja käytettäessä. Graafitietokannoissa, joissa suhteiden linkitykset on tehty fyysisillä linkityksillä, tietuetta ei tarvitse etsiä indeksihakuna, vaan linkitys osoittaa suoraan kohdepaikkaan. Tällöin relaatiotietokannan JOIN-operaation asymptoottisen suoritusajan ollessa O(log n), jossa n on rivien määrä, lähentelee aidossa graafitietokannassa tehdyn vastaavanlaisen linkitysoperaation suoritusaika O(1).

Graafitietokannassa linkityksen kohteen löytäminen ei vaadi listan läpikäymistä, jossa suoritusaika voisi tietomäärän lisääntyessä kasvaa. Tästä syystä suoritusajaksi merkitään O(1), joka tarkoittaa käytännössä sitä, että suoritusaika on vakio. Vastaavasti relaatiotietokannassa linkitetyn tiedon löytäminen vaatii indeksejäkin hyödynnettäessä iteraatiota. [OrientDB, 2018b]

5.2. Kritiikkiä graafitietokannoista

Yksi olennaisimpia heikkouksia graafitietokannoissa on eheysrajoitteiden (integrity constraints, IC) puute. Yksi hyvä tapa välttää tämän puutteen tuomia haittoja on laadukas käsitteellisen tason tietomallin suunnittelu, mutta tämän lisäksi tiedon eheyttä pitäisi valvoa myös fyysisellä tasolla. Kääntöpuolena tässä on se, että yksi graafitietokantojen eduista on riippumattomuus tiedon ennalta määritellystä muodosta (schema-less). Kuitenkin samaan aikaan, erityisesti tietokantoja hallinnoivilla tahoilla on monesti tarve varmistua tiedon eheydestä tietokantakaavion kaltaisella fyysisen tason työkalulla.

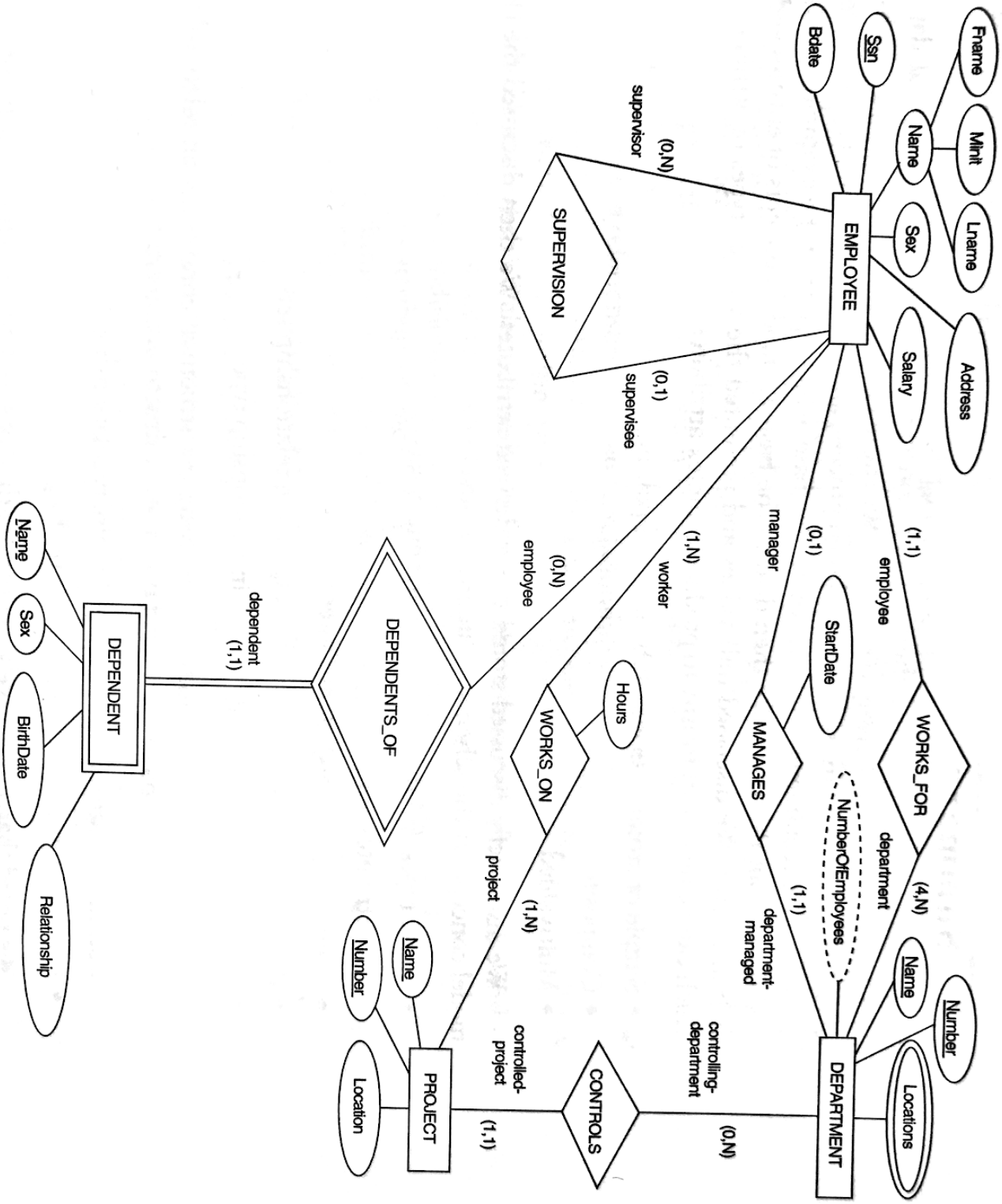
Tätä ristiriitaisuutta graafitietokantojen valmistajat ovat pyrkineet ratkaisemaan eri tavoin. Ainakin Neo4j, sekä OrientDB tarjoavat graafitietokantoja, joissa on tuki kaaviolle. OrientDB on jopa jakanut kaavion käytön valinnaisesti kolmelle tasolle, schema-full, schema-less ja schema-hybrid. Jotkut graafitietokantavalmistajat tarjoavat myös jonkinlaista IC-tukea jopa ilman kaaviota. [Pokorný et al. 2017] Neo4j:n graafitietokannssa on mahdollista kirjoittaa eksplisiittisiä rajoitteita, joilla voi esimerkiksi määritellä tietyn solmun sisältämän attribuutin uniikiksi, ja näin mallintaa relaatiotietokannan pääavainta. Lisäksi on mahdollista kirjoittaa muitakin relaatiotietokannasta tuttuja rajoitteita, kuten vaatimus tietystä entiteettityypin ilmentymästä ennen lisäystä. [Neo4j, 2018e]

Graafitietokantojen kaavion puute mahdollistaa myös tietokannan huonon toteutuksen. Neo4j on listannut tällaisia huonoja käytäntöjä, joita tulisi ehdottomasti välttää. Triviaalimpina asioina mainitaan tietokannan suunnittelun ohittaminen. Eli vaikka graafitietokantaan on mahdollista lisätä tietoa ennalta määrittelemättömässä muodossa, on ehdottoman tärkeää, että tietokannan sisältö suunnitellaan huolella, ja toteutuksessa noudatetaan joitain nimeyskäytäntöjä. Lisäksi erityishuomionarvoinen asia on graafitietokannan huono kyky varastoida binäärimuotoista, suurikokoista dataa. Toisin sanoen graafitietokanta ei ole dokumenttivarasto. Tämä saattaa olla huomionarvoista erityisesti relaatiotietokannasta graafitietokantaan siirryttäessä, sillä relaatiotietokannoissa saattaa olla käytetty esimerkiksi BLOB (Binary Large Object)-muotoisia taulun avaimia, joihin on tallennettu tiedostoja. [Neo4j, 2018f]

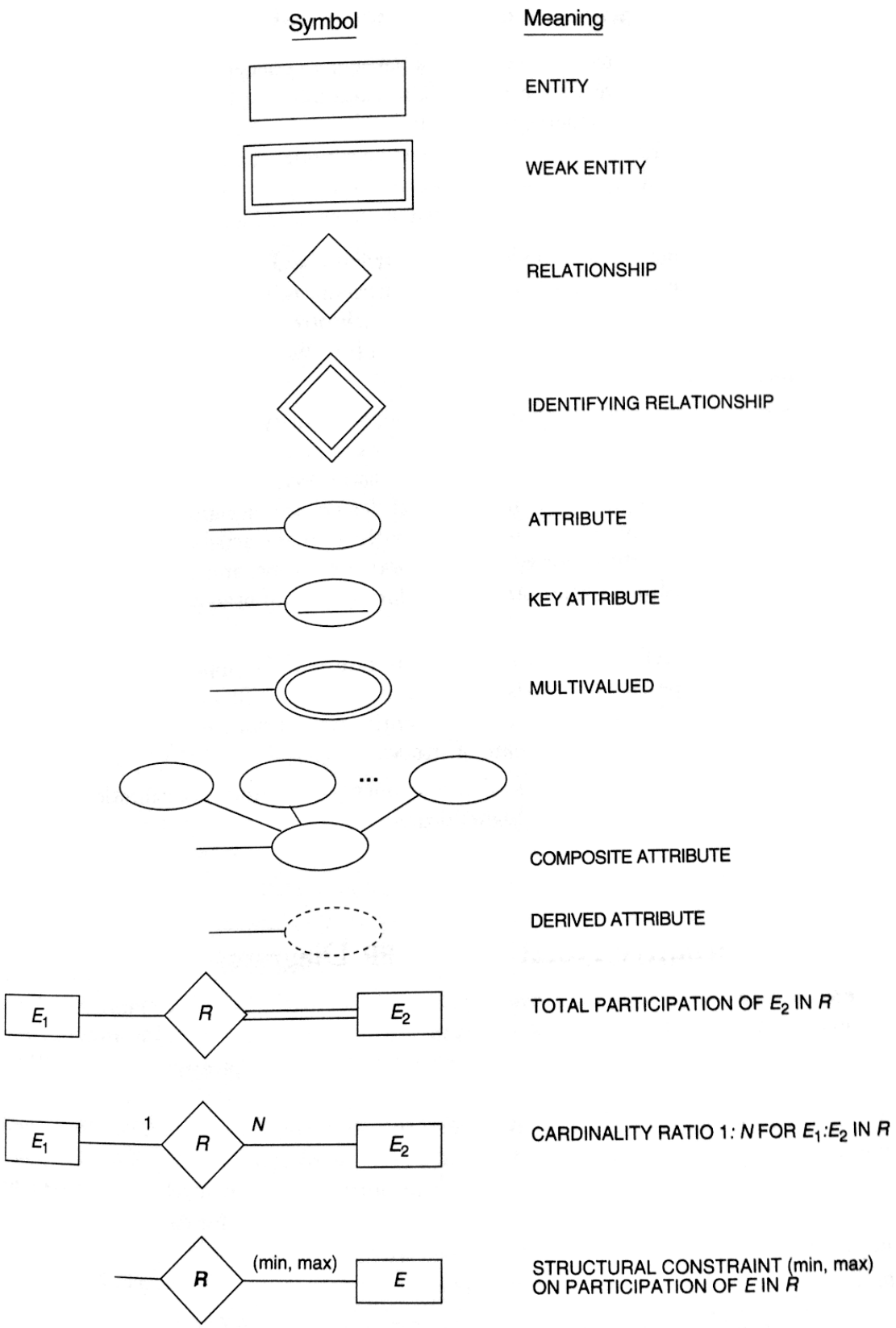
6. Tietomallien vertailu

Tähän mennessä on käyty läpi eri tietomalleja, niiden pohjalta kehitettyjä tietokannanhallintajärjestelmiä, sekä niihin liittyviä kyselykieliä. Tarkemmin on perehdytty relaatiotietokantoihin ja graafitietokantoihin. Graafitietokannat, kuten myös muut NoSQL-tietokannat, ovat viime vuosina alkaneet saada paremmin jalansijaa tietojärjestelmäkehityksessä, ja näin ollen relaatiotietokantojen monopoliasema saattaisi olla murtumassa. Jotta voidaan konkreettisesti havaita, kuinka graafitietokanta eroaa relaatiotietokannasta, vertaillaan, kuinka sama tietokanta on mahdollista ilmentää sekä relaatiossa, että graafina. Lisäksi vertaillaan, kuinka tietokantoihin kohdistetut kyselyt eroavat graafikyselykielillä ja SQL-kyselykielellä toteutettuna.

Tässä tutkielmassa on valittu vertailun lähtökohdaksi Elmasrin ja Navathen kirjassaan esittämä ER-kaavio yrityksen rakenteesta (Kuva 3). Tutkielmassa luodaan tämän ER-kaavion pohjalta relaatiomalli, sekä graafimalli kyseisestä tietovarastosta. Kuvassa 4 on esitetty selitteet ER-kaaviossa käytetyille symboleille ja notaatiolle. Yleensä ER-kaavio mallinnetaan käyttäen Chen-notaatiota, jonka Peter Chen esitteli vuonna 1976 julkaistussa artikkelissaan. Tässä tutkielmassa hyödynnetään kuitenkin Elmasrin ja Navthen käyttämää notaatiota, joka eroaa Chenin hieman esittelemästä notaatiosta. Elmasrin ja Navathen käyttämässä notaatiossa entiteetin kytkös suhteeseen esitetään minimi-maksimi-lukuarvoilla. Tämän (min,max)-notaation esitteli Jean-Raymond Abrial vuonna 1974. Tällä tarkoitetaan käytännössä sitä, että entiteettityypin E on osallistuttava suhteeseen R vähintään minimin ja korkeintaan maksimin verran. [Elmasri and Navathe, 2000]



Kuva 3. ER-kaavio yrityksen rakenteesta [Elmasri and Navathe, 2000]

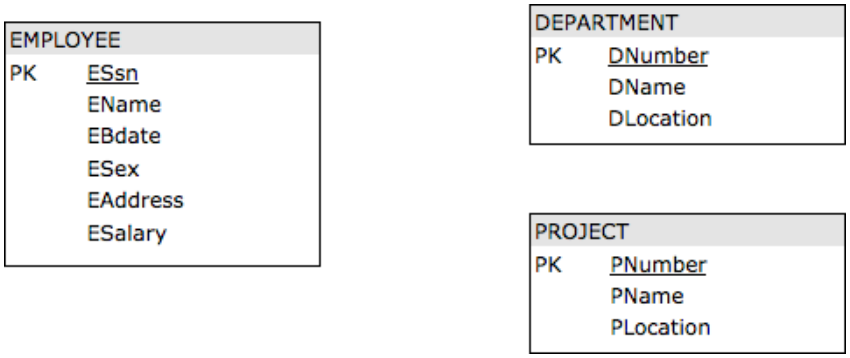


Kuva 4. ER-kaavion notaation selitteet [Elmasri and Navathe, 2000]

6.1. Relaatiokaavion muodostaminen ER-kaavion pohjalta

Relaatiokaavion muodostaminen ER-kaavion pohjalta on lähtökohtaisesti melko suoraviivaista. ER-kaavio sisältää jo nimensäkin mukaisesti olennaisimmat lähtökohdat relaatiokaavion muodostamiseen, eli entiteettityypit (entity) ja suhteet (relation). Tässä luvussa käydään läpi Elmasrin ja Navathen kirjassaan esittelemät vaiheet relaation luomiseen ER-kaavion pohjalta. Vaiheet muodostavat algoritmin, jonka perusteella relaation muodostamisen ER-kaaviosta tulisi olla suoraviivaista. Vastaavanlaisia vaiheittaisia tapoja ER-kaavion muuntamisesta relaatiomalliksi ovat toki esitelleet muutkin. Esimerkiksi Chen [1976] esitteli jo vuonna 1976 tapoja hyödyntää ER-kaaviota mm. relaatiomallin ja verkkomallin muodostamiseen. Myös Harrington [2002] on esitellyt vuonna 2002 hyvin toimivan ja yksinkertaistetun ohjenuoran, kuinka ER-kaaviosta muodostetaan relaatiokaavio.

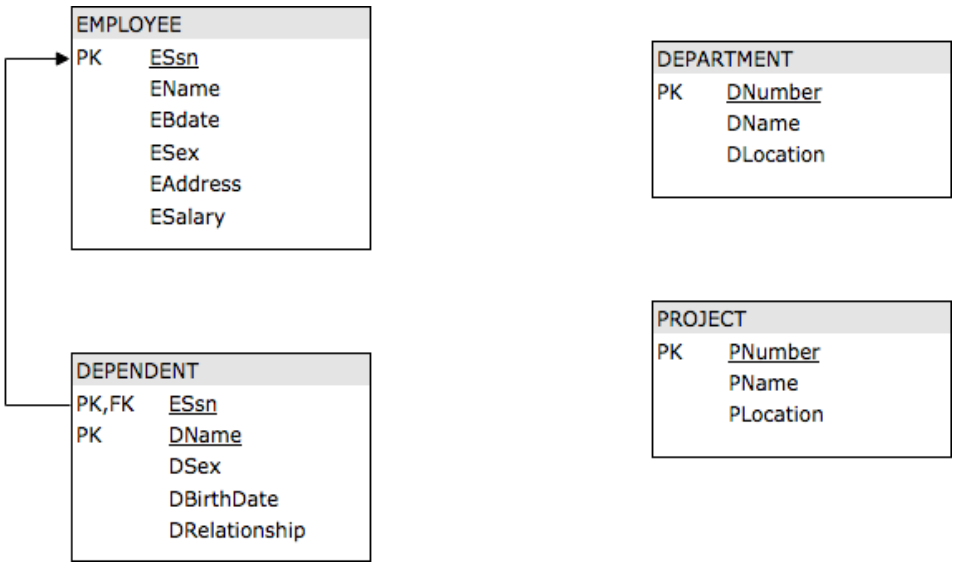
Ensimmäisessä vaiheessa Elmasrin ja Navathen algoritmia, tunnistetaan ER-kaaviosta niin sanotut vahvat, eli normaalit entiteettityypit, luodaan näille relaatiot kaavioon. Tämän jälkeen tunnistetaan kunkin entiteettityypin yksinkertaiset attribuutit ja lisätään ne relaatioihin. Lisäksi valitaan yksi avainattribuuteista relaation pääavaimeksi (primary key). Yrityksen rakenteesta (Kuva 3) löytyy neljä entiteettityyppiä, jotka ovat EMPLOYEE, DEPARTMENT, PROJECT ja DEPENDENT. DEPENDENT-entiteettityyppi on ns. heikko entiteettityyppi, joten jätetään se tässä vaiheessa lisäämättä kaavioon. Entiteettityyppiä kutsutaan heikoksi entiteettityypiksi, jos sillä on suhde, joka on luotu tunnistamaan entiteetti. [Chen, 1976] (vrt. kappale 1.1) Tässä tapauksessa suhde DEPENDENTS_OF määrittelee työntekijän huollettavat, jolloin DEPENDENT on heikko entiteettityyppi. Tässä vaiheessa relaatiokaavio näyttää kuvan 5 kaltaiselta.



Kuva 5. Relaatiokaavio 1. vaiheen jälkeen

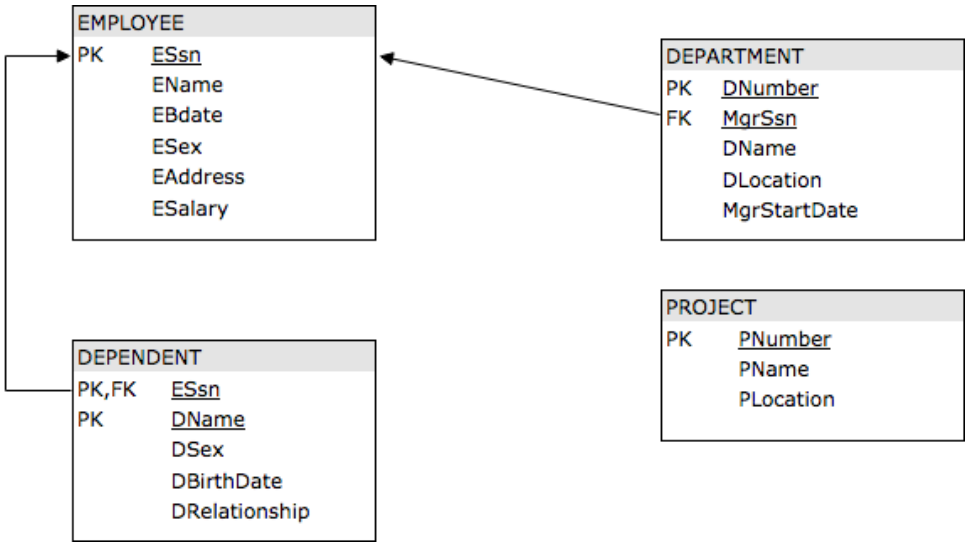
Seuraavassa vaiheessa tunnistetaan ER-kaaviosta heikot entiteettityypit. Edellisessä vaiheessa jo todettiinkin DEPENDENT-entiteettityypin olevan heikko entiteettityyppi. Luodaan tälle oma relaatio, lisätään yksinkertaiset attribuutit, sekä tunnistetaan heikon entiteettityypin omistajan pääavain ja lisätään se heikon entiteettityypin viiteavaimeksi (foreign key). Pääavaimeksi heikolle entiteettityypille tulee yhdistelmä, jossa on omistavan entiteettityypin pääavain ja mahdollisesti jokin heikon entiteettityypin attribuuteista. Heikon entiteettityypin pääavaimen valinnassa on monta vaihtoehtoa, mutta edetään tässä Elmasrin ja Navathen valitsemalla mallilla ja luodaan

DEPENDENT-relaation pääavaimeksi yhdistelmä omistavan entiteettityypin EMPLOYEE attribuutista ”ESsn”, sekä DEPENDENT-entiteettityypin attribuutista ”DName”. Tässä vaiheessa relaatiokaavio on kuvan 6 mukainen. [Elmasri and Navathe, 2000]



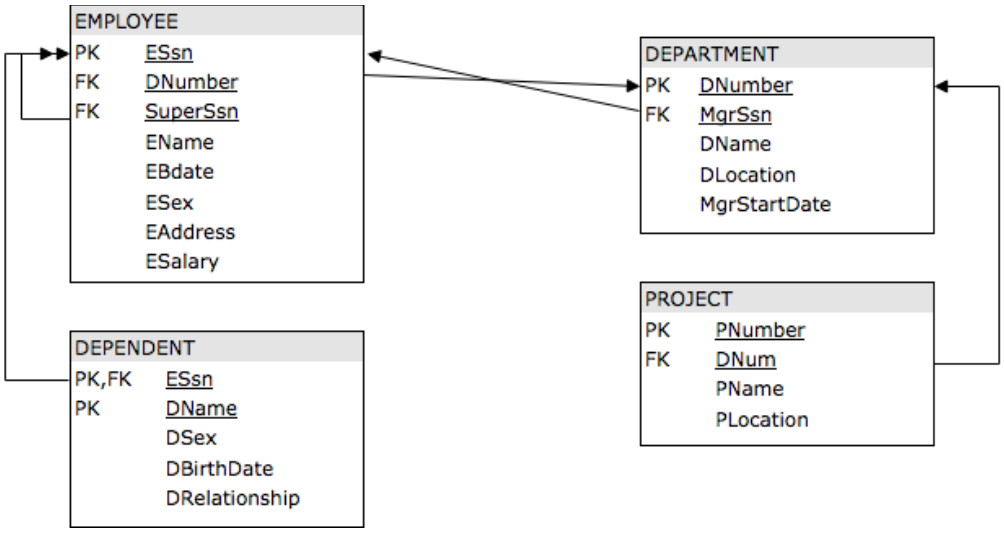
Kuva 6. Relaatiokaavio 2. vaiheen jälkeen

Vaiheessa 3 etsitään kaikki 1:1-suhteet ER-kaaviosta, sekä tunnistetaan siihen osallistuvat entiteettityypit. Määritellään toiselle suhteeseen kuuluvalle entiteettityypille, eli relaatiomallin relaatiolle, viiteavain, joka viittaa suhteen toisen entiteettityypin pääavaimeen. Viiteavaimen omistavaksi entiteettityypiksi on hyvä valita entiteettityyppi, jolla on “täysi osallistuminen” (total participation) suhteeseen. Tämän lisäksi, lisätään tästä entiteettityypistä muodostetulle relaatiolle kaikki yksinkertaiset attribuutit, jotka suhde sisältää. Yrityksen rakenteessa on tunnistettavissa yksi tämän kaltainen suhde, joka on MANAGES. Suhteeseen täysin osallistuva entiteettityyppi on DEPARTMENT, sillä jokaisella laitoksella on johtaja. Näin ollen, voidaan lisätä DEPARMENT-relaatioon viiteavaimen MGRSSN, jonka avulla tunnistetaan EMPLOYEE-relaatiosta laitoksen johtaja. Tämän jälkeen relaatiokaavio näyttää kuvan 7 kaltaiselta. [Elmasri and Navathe, 2000]



Kuva 7. Relaatiokaavio 3. vaiheen jälkeen

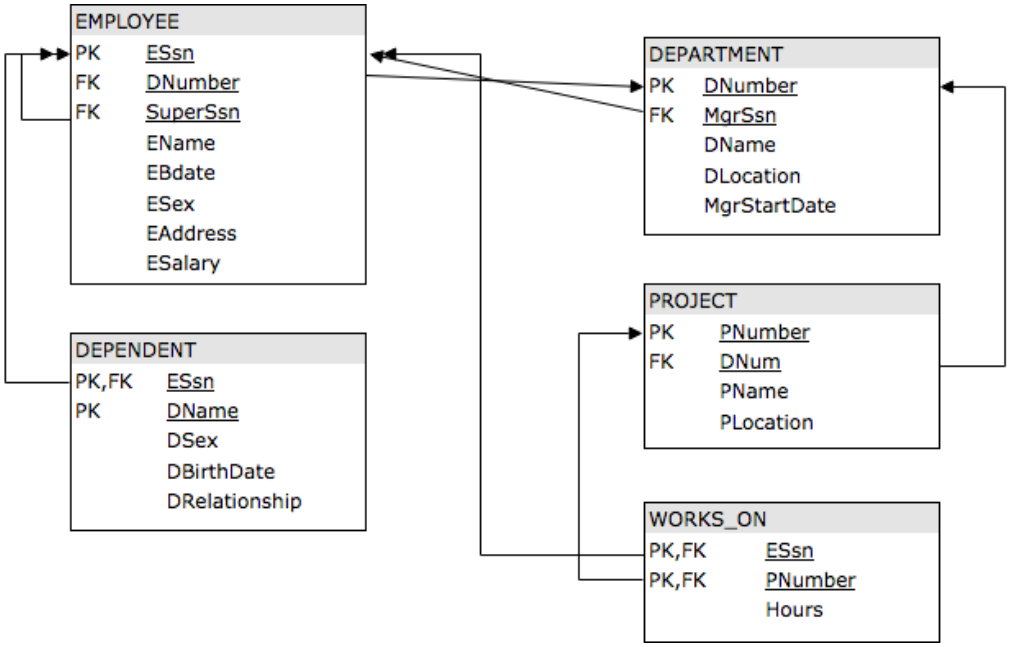
Seuraavaksi, 4. vaiheessa, etsitään kaikki 1:N-suhteet ja tunnistetaan näistä suhteista entiteettityyppi S, joka on suhteen N-puolella. Lisätään viiteavain relaatioon S, joka on suhteen toiselta puolelta löytyvän entiteettityypin T pääavain. Relaation viittaus toteutetaan näin päin, koska taulun S alkio on osallisena korkeintaan yhteen alkioon relaatiossa T. Tämän jälkeen lisätään, kuten edellisessäkin vaiheessa, kaikki suhteesta löytyvät yksinkertaiset attribuutit relaation S attribuuteiksi. Tässä vaiheessa relaatiokaavio näyttää kuvan 8 kaltaiselta. [Elmasri and Navathe, 2000]



Kuva 8. Relaatiokaavio 4. vaiheen jälkeen

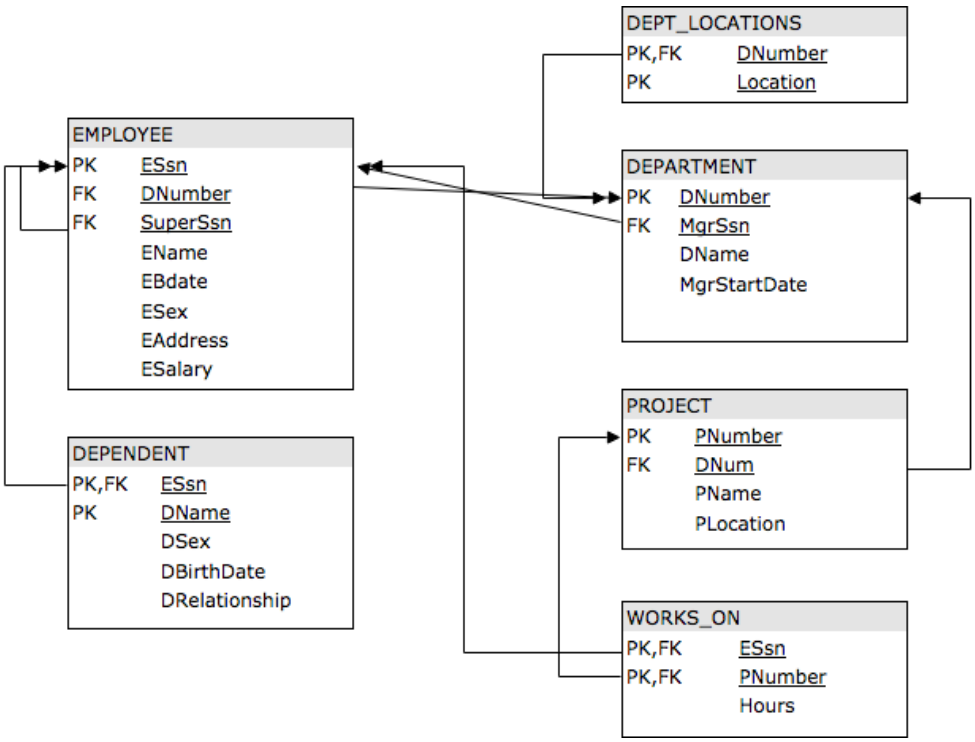
Elmasrin ja Navathen algoritmin 5. vaiheessa, tunnistetaan ER-kaaviosta kaikki n:m-suhteet ja luodaan näille vastaavat relaatiot kaavioon. Asetetaan luodun relaation viiteavaimiksi suhteeseen kuuluvien relaatioiden pääavaimet, ja tehdään näiden viiteavainten yhdistelmästä relaation

pääavain. Lisäksi, kuten aikaisemminkin, lisätään kaikki suhteen yksinkertaiset attribuutit relaation attribuuteiksi. Tässä tapauksessa luodaan uusi relaatio suhteesta WORKS_ON, lisätään sen viiteavaimiksi relaatioiden EMPLOYEE ja PROJECT pääavaimet, sekä luodaan näiden avainten yhdistelmästä relaation pääavain. Relaatiokaavio on tässä vaiheessa kuvan 9 mukainen.



Kuva 9. Relaatiokaavio 5. vaiheen jälkeen

Tässä vaiheessa algoritmia ER-kaaviosta on käyty läpi kaikki entiteettityypit ja suhteet. Seuraavaksi kaaviosta tunnistetaan kaikki moniarvoiset attribuutit, joita käsitellyssä esimerkissä on yksi. Kullekin moniarvoiselle attribuutille tulee luoda uusi relaatio kaavioon. Tälle relaatiolle luodaan attribuutti A, joka vastaa moniarvoisen attribuutin nimeä. Lisäksi relaatiolle luodaan viiteavain K, joka viittaa lähdetauluun. Lopuksi relaatiolle luodaan vielä pääavain, joka voi olla yhdistelmä luotua viiteavainta K ja toista attribuuttia A. Yrityksen rakenteessa moniarvoisia attribuutteja on yksi, ”Locations”. Luodaan tälle attribuutille oma relaationsa DEPT_LOCATIONS, lisätään sille attribuutti ”Location”, annetaan sille viiteavain ”DNumber” ja tehdään näiden yhdistelmästä relaation pääavain. Kun relaatio DEPT_LOCATIONS on luotu, voidaan luonnollisesti poistaa attribuutti ”Locations” relaatiosta Department. Tämän jälkeen relaatiokaavio on kuvan 10 mukainen.



Kuva 10. Relaatiokaavio 6. vaiheen jälkeen

Algoritmin viimeisessä vaiheessa tulee tunnistaa kaikki suhteet, joissa suhteeseen osallistuvia entiteettityyppejä on enemmän kuin kaksi (n-ary). Tällaiselle suhteelle tulee jälleen luoda uusi, sitä vastaava relaatio kaavioon. Relaatiolle asetetaan viiteavaimiksi kaikkien suhteeseen osallistuvien entiteettityyppien pääavaimet, sekä attribuuteiksi kaikki suhteen omat yksinkertaiset attribuutit. Käsitellyssä yrityksen rakennetta kuvaavassa ER-kaaviossa ei ole lainkaan tällaisia n-ary-suhteita.

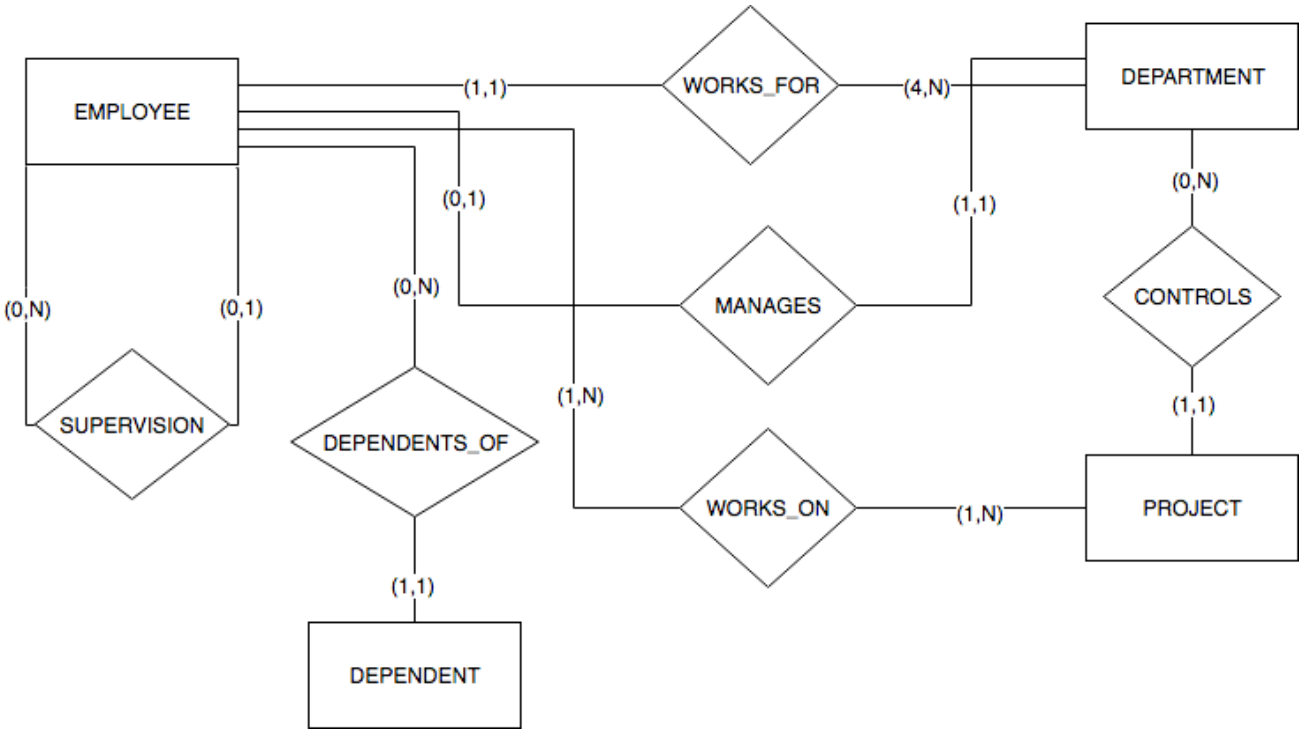
6.2. Graafimallin muodostaminen ER-kaavion pohjalta

Tietojärjestelmäkehityksessä on hyvin yleistä, että tietokantakuvaus aloitetaan ER-kaaviosta ja jatketaan sen pohjalta relaatiomalliin ja myöhemmin relaatiotietokantaan. Edellisessä kappaleessa käytiin läpi tätä prosessia, ja kuinka suoraviivaisesti ER-kaavio kääntyy relaatiokaavioksi. Graafimalli sen sijaan ei ole vastaavalla tavalla suoraviivaisesti muodostettavissa ER-mallin pohjalta. Tällaisesta muunnoksesta ei myöskään ole vastaavalla tavalla olemassa valmista tutkimusta ja kirjallisuutta. Esitellessään ER-mallin Chen [1976] kuitenkin esitteli myös tavat, kuinka ER-mallista voidaan johtaa verkkomalli, relaatiomalli, sekä entiteettimalli. Etsittäessä tapaa mallintaa graafimalli ER-mallin pohjalta, käytetään tässä pohjana Chenin esittelemää tapaa johtaa verkkomalli ER-mallista.

6.2.1. Muunnos verkkomalliksi

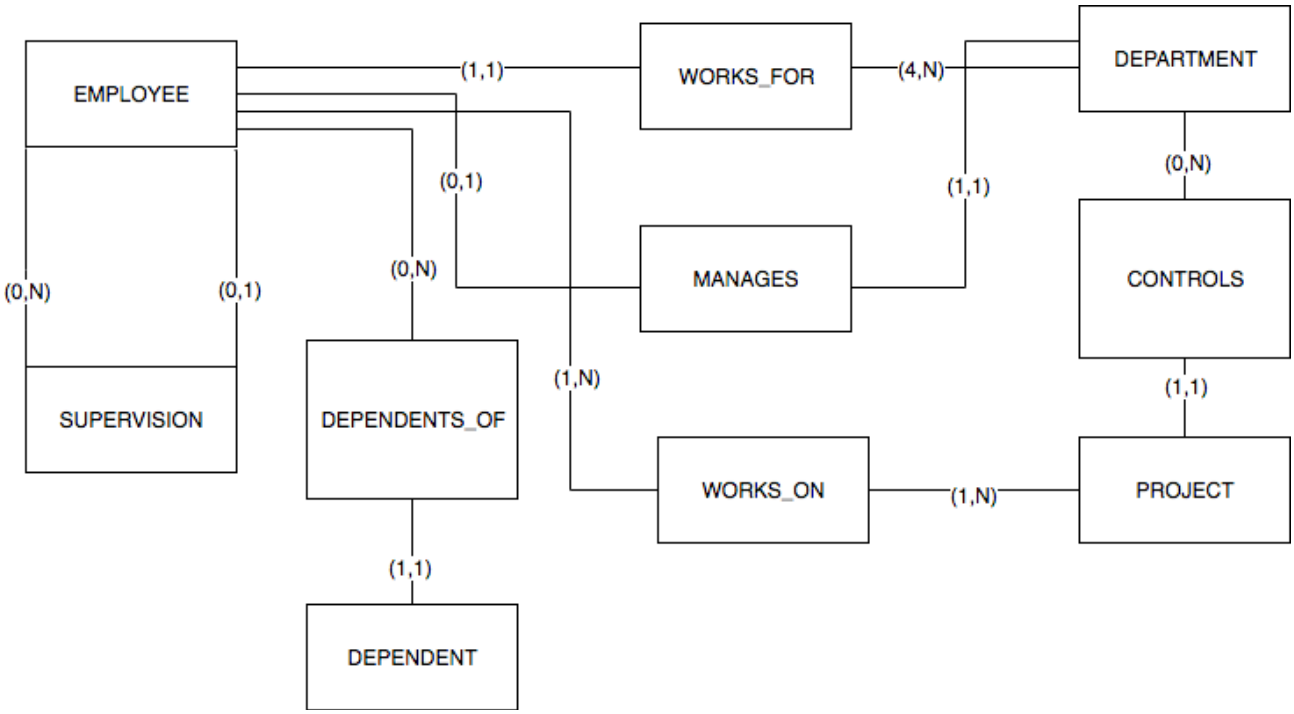
Chenin [1976] mukaan ER-mallia voidaan käyttää tiedonmallinnuksen työkaluna verkkotietomalliin pohjautuvissa järjestelmissä. ER-mallia esittelevässä artikkelissaan Chen esittää tavan muuttaa ER-malli verkkomalliksi kahdella eri tavalla. Ensimmäinen tapa on johtaa ER-mallista tietorakennekaavio (data structure diagram), jonka avulla verkkomallin johtaminen on hyvin suoraviivaista. Chen esittelee vaiheittaisen tavan, kuinka ER-mallista johdetaan tietorakennekaavio, joka vastaavasti on hyvin helposti muutettavissa verkkomalliksi. Ensin 1:n-suhteet muutetaan nuoliksi, jonka jälkeen n:m-suhteet muutetaan omiksi entiteettityypeikseen, josta lähtee nuolet suhteen alkuperäisiin entiteettityyppeihin.

Toinen tapa johtaa verkkomalli ER-mallista on suoraviivaisesti muuttaa kaikki ER-mallin suhteet omiksi entiteettityypeikseen. Eli jokainen suhde täytyy olla yhdistetty johonkin tietuetyyppiin. Toisin sanoen, vastaavasti kun relaatiomalliin siirryttäessä monen-suhde-moneen (n-ary) tyyppiset suhteet muutettiin omiksi tauluikseen, muutetaan verkkomalliin siirryttäessä kaikki suhteet omiksi tietuetyypeikseen (record type). Tarkastellaan seuraavaksi, kuinka Elmasrin ja Navathen esittelemästä yrityksen rakennetta kuvaavasta ER-kaaviosta tehdään tämä muutos. Jätetään kuitenkin tässä vaiheessa selkeyden vuoksi attribuutit huomioimatta ja keskitytään tietomallin entiteetteihin ja suhteisiin.



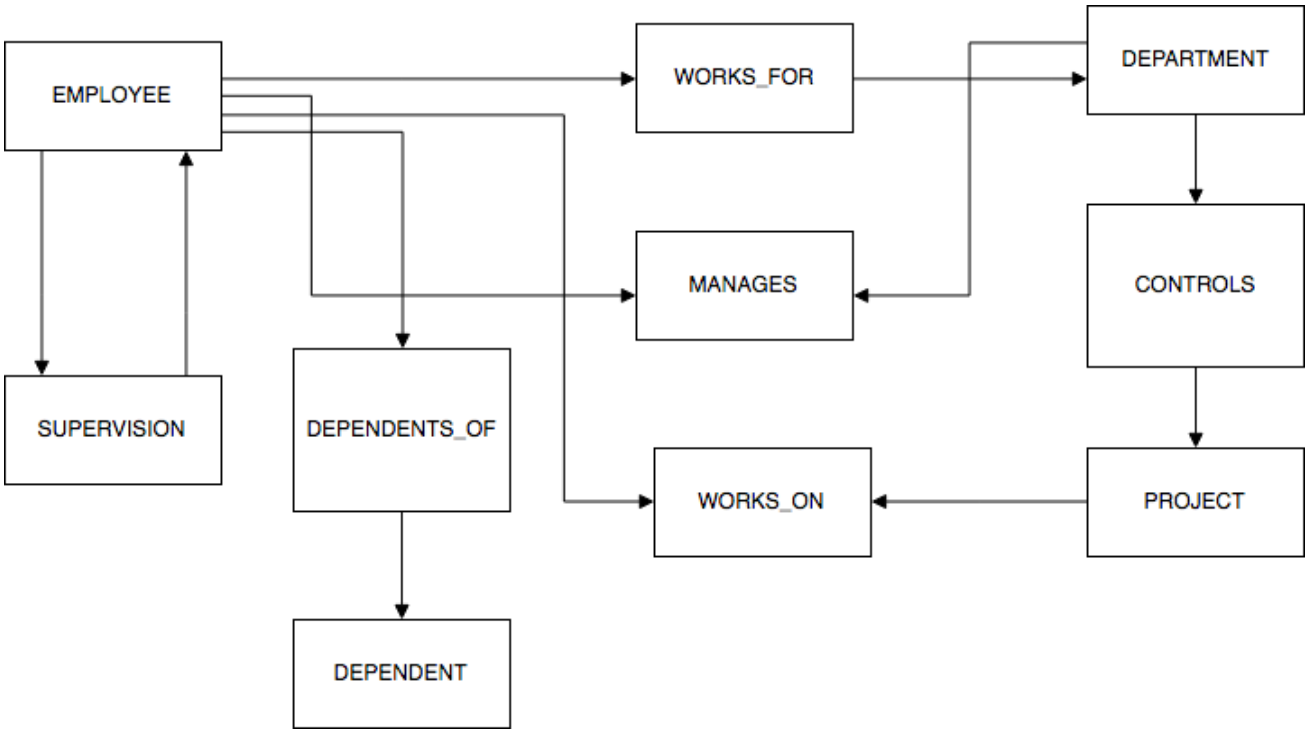
Kuva 11. ER-kaavio yrityksen rakenteesta

Kuvassa 11 on yksinkertaistettu esitys Elmasrin ja Navathen yrityksen rakennetta kuvaava ER-kaavio. Tästä on jalostettu kuvan 12 verkkomallin pohja, muuttamalla kaikki suhteet tietuetyypeiksi.



Kuva 12. ER-kaavion muuntaminen verkkomalliksi (Vaihe 1.)

Seuraavassa vaiheessa tietueiden väliset yhteydet tulisi tunnistaa, ja merkitä näille yhteyksille suunta. Suunnalla viitataan tietuetyyppien väliseen yhteyteen, jossa emo-tietuetyypistä (owner-record type) viitataan lapsi-tietuetyyppiin (member-record type). Kuvassa 13 on esitetty, kuinka yksinkertainen verkkomalli viimeistellään tietuetyyppien välisten suhteiden suuntien avulla.



Kuva 13. ER-kaavion muuntaminen verkkomalliksi (Vaihe 2.)

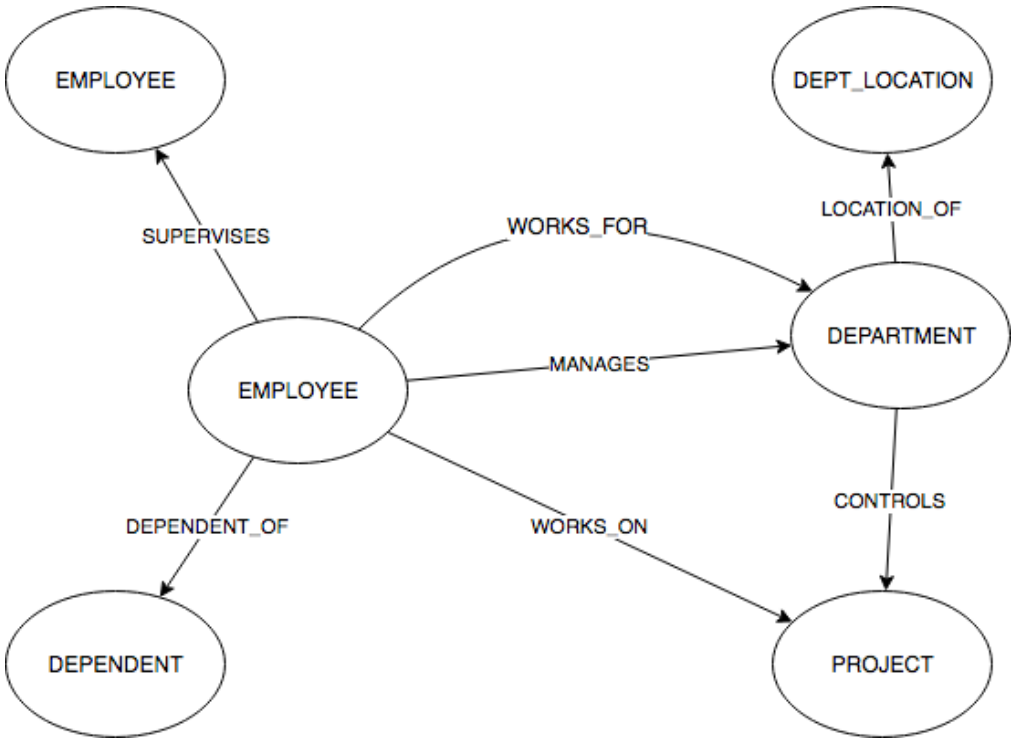
Kuten jo tästä muunnoksesta nähdään, niin ER-kaavion hyödyntäminen verkkotietomalliin pohjautuvien järjestelmien suunnittelussa on täysin perusteltua. Muunnoksen vaiheet ovat lopulta melko intuitiivisia ja vaiheita ei ole kovin montaa. Seuraavaksi tarkastellaan, kuinka näiden avulla päästään graafimalliin.

6.2.2. Muunnos graafimalliksi

Kuten Pokorný, Valenta ja Kovačič [2017] mainitsevat artikkelissaan, niin merkittävä askel minkä tahansa tietokannan käyttöönotossa on tarkka ja laadukas tietomallin kuvaus. Käytetään tässä tutkielmassa samoja graafitietomallin tarkentavia määrittelyjä, kuin Pokorný ja muut artikkelissaan. Luodaan edellisen kappaleen verkkomallimuotoon käännetty tietomalli, ominaisuussisältöiseksi graafitietomalliksi (property graph model), jossa on solmuja (node), ominaisuuksia (attribute), otsikoita (label) ja suhteita (edge). Kaikilla suhteilla on suunta, eli lähtösolmu ja päätesolmu. Tämä graafitietomallin muoto on yksi käytetyimmistä tavoista kuvat graafia, graafitietokantojen keskuudessa. Ainakin Neo4j, sekä Titan käytävät tätä tapaa kuvata graafia.

Graafimalli on mahdollista johtaa myös jo olemassa olevasta relaatiomallista. Tästä yhtenä esimerkkinä on DZonen yhdessä Neo4j:n kanssa julkaisema ohjeistus graafitietokannan mallintamiseen. Seuraavaksi tarkastellaan, saadaanko graafimalli johdettua suoraan kappaleessa 6.1 kuvatusta relaatiomallista. [DZone, 2018]

DZonen tarjoaman artikkelin mukaan muunnos relaatiomallista graafimalliin on hyvin yksinkertaista. Pääsääntönä muunnoksessa on pyrkiä eroon viiteavaimista kaikissa relaatioissa. Tämä on mahdollista nimettyjen suhteiden ansiosta, jotka korvaavat vierasavainten muodostamat relaatioiden väliset suhteet. Olennainen ero graafitietomallin ja relaatiotietomallin välillä onkin graafitietomallin kyky sitoa avain-arvo-pareja suhteisiin. Tämän ansiosta graafimallin kuvauksessa ei ole tarvetta kahden entiteettityypin välisille ns. liitos-relaatioille, joiden tarkoitus on toimia vain yhdisteenä kahden entiteettityypin välillä. Kuvassa 14 on johdettu kuvan 10 relaatiomalli graafimalliksi poistamalla viiteavaimet ja luomalla tarvittavat suhteet relaatioiden välillä.



Kuva 14. Relaatiomallin pohjalta muodostettu graafimalli-kaavio

Kuvassa 14 on käytetty notaatiota, joka on osittain suunnattu multigraafi (directed multigraph), mutta josta on poistettu silmukat, joissa linkki osoittaa takaisin samaan solmuun. Tällaiset “loopit” on poistettu lisäämällä kaavioon kopio tällaisesta solmusta ja luotu suhde-linkki näiden välille. [DZone, 2018]

Kuten kuvista 13 ja 14 voi huomata, niin verkkotietomalli ja graafitietomalli eroavat toisistaan jonkin verran. Keskeinen ero tietomalleissa on graafitietomallin kyky sitoa tietoa suhteisiin. Tämä tekee graafitietomallista erityisen ketterän tavan mallintaa tietoa. Graafitietomallissa on mahdollista luoda suhteita mielivaltaisesti eri solmujen välille, ja vastaavasti on mahdollistaa selvittää mitä suhteita solmuille kuuluu. Verkkotietomalli pohjautuu vielä paljolti hierarkkiatietomalliin, jossa suhteet muodostuvat relaatiotietomallin tapaan entiteettien kautta. Graafitietokannat pohjautuvat useimmiten myös ominaisuussisältöiseen graafitietomalliin, jossa solmuille ja suhteille on mahdollista tarvittaessa lisätä mielivaltaisia avain-arvo-pareja, määrittelemättä niitä etukäteen.

Muunnettaessa olemassa olevaa relaatiotietokantaa graafitietokannaksi, on olennaista käydä läpi relaatiomallissa olleet olennaiset eheystarkistelut ja toteuttaa ne myös graafitietokantaan. Tässä huomioitavaa on esimerkiksi, pääavainten rajoittaminen uniikeiksi. Näin myös graafitietokannassa säilyy osittain eräänlainen normaalimuoto. Lisäksi on hyvä lisätä indeksit usein haetuille attribuuteille, suorituskyvyn parantamiseksi. Graafitietokannasta voidaan myös poistaa tyhjät

attribuuttien arvot, joita ei enää tarvitse ylläpitää siirryttäessä pois relaatiotietokannasta. [Neo4j, 2018a]

6.3. Tietomallien eroavaisuudet

Tarkasteltaessa kuvia 9 ja 14, on tiedon kuvauksessa jonkin verran eroja. Merkittävimpänä asiana esiin nousee tiettyjen entiteettityyppien muuttuminen suhteeksi graafitietomallissa. Tämän voi kuitenkin ajatella vain eräänlaisena kosmeettisena muutoksena, sillä graafitietomallin suhteen voi yhtä lailla ajatella omana entiteettityyppinään. Toisaalta samaan aikaa tässä piilee ehkä olennaisin ero relaatiotietomallin ja graafitietomallin välillä. Tarkastellaan tätä esimerkin kautta. Mikäli malliin pitäisi lisätä uusi entiteettityyppi, harrastus (HOBBY), joka ilmentää mitä työntekijät harrastavat. Tässä tapauksessa relaatiokaavioon tulisi luoda uusi taulu ja määrittää sille pääavain. Tämän lisäksi tulisi luoda vielä toinen uusi taulu, HOBBY_OF, joka määrittelee kuka harrastaa mitäkin. Tälle taululle tulisi lisätä myös pääavain sekä viiteavaimet, jotka viittaavat EMPLOYEE- ja HOBBY-tauluihin. Tietokantatasolla nämä kaikki muutokset tulisi luonnollisesti tehdä tietokannan kaavioon.

Graafimallissa tällainen muutos on huomattavasti kevyempi toteuttaa. Käytännössä kaaviotasolla lisättäisiin uusi solmu, HOBBY. Lisäksi tähän solmuun kohdistettaisiin suhde, HOBBY_OF, solmusta EMPLOYEE. Tietokantatasolla muutoksia olemassa olevaan tietoon ei tarvitsisi tehdä, ellei tässä yhteydessä lisättäisi joitakin eheysrajoitteita, joiden takia vanhakin data pitäisi korjata. Tämän jälkeen graafitietokantaan voitaisiin yksinkertaisesti lisätä HOBBY_OF-tyyppisiä suhteita työntekijästä harrastukseen.

Kyseinen muutos toisi mukanaan myös potentiaalisen suorituskykyongelman relaatiotietokannassa. Tämä johtuu siitä, että käytännössä muutoksen mukana tulisi kaksi uutta tietokantataulua, joihin olisi tehtävä JOIN-operaatioita, kohdistettaessa kyselyjä työntekijöiden harrastustietoihin. Kuten luvussa 5.2 käytiin jo läpi, niin graafitietokannoissa vastaavaa pullonkaulaa ei ole. Kysely on myös graafikyselynä verrattain luettavampi, kun taululiitoksia ei tarvitse tehdä. Tarkastellaan seuraavassa kahta edellä lisättyyn harrastustietoon kohdistuvaa kyselyä. Koodiesimerkissä 3 on esimerkki SQL-kyselystä, jolla haetaan kaikki jalkapalloa harrastavat työntekijät. Koodiesimerkissä 4 on esimerkki vastaavasta kyselystä Cypher-graafikyselykielellä. Palataan kuitenkin näihin kyselykohtaisiin eroavaisuuksiin paremmin luvussa 6.3.1.

```
SELECT EMPLOYEE.Name
FROM EMPLOYEE
JOIN HOBBY_OF ON (EMPLOYEE.id = HOBBY_OF.employee_id)
JOIN HOBBY ON (HOBBY_OF.hobby_id = HOBBY.id)
WHERE HOBBY.Name = 'Football';
```

Koodiesimerkki 3. Jalkapalloa harrastavat SQL-kyselyllä

```
MATCH (e:EMPLOYEE)<-[:HOBBY_OF]-(h:HOBBY)
WHERE h.Name = 'Football'
RETURN e.Name
```

Koodiesimerkki 4. Jalkapalloa harrastavat Cypher-kyselyllä

Tutkielmassa käytetyn esimerkin laajuus on huomattavan pieni, jolloin relaatiotietomallin kankeus muuttuvan tiedon käsittelyssä ei tule niin kouriintuntuvasti esille, kuin jossakin todellisessa, laajan tietomallin tapauksessa saattaisi tulla. Syynä relaatiotietomallin muuttamisen kankeuteen on sen ennalta määritelty tiedon muoto. Tiedon muodon määrittelyssä ei sinänsä ole mitään vikaa tai väärää, päinvastoin. Mutta kankeuden tuo sen *ennalta* määriteltävyys. Tämä hankaloittaa uusien tietojen mallintamista olemassa olevaan kaavioon. Graafimallissa uudet asiat voidaan tuoda vanhojen rinnalle vaivattomasti lisänä olemassa olevaan, ilman riskiä vanhan mallin eheyden rikkoontumisesta.

On myös olemassa tietokantoja, jotka pyrkivät ottamaan parhaat puolet eri tietomalleista ja muodostamaan tällä tavoin yleiskäyttöisen tietokannan. Yksi esimerkki tällaisesta tietokannasta on OrientDB. Kuten jo aikaisemmin on todettu, niin OrientDB on monimallitietokanta, joka pyrkii yhdistelemään graafi-, dokumentti- ja relaatiotietovarastojen parhaita puolia. Yksi erityinen piirre OrientDB:ssä on, että se tukee SQL-kyselykieltä. Tällä tavoin OrientDB pyrkii minimoimaan uuden osaamisen kehittämisen tarvetta (learning curve), samalla hyödyntäen graafitietomallin ja graafikyselyiden etuja. Toisin sanoen OrientDB:n avulla on mahdollista kirjoittaa JOIN-operaatioita sisältäviä SQL-kyselyjä, jotka suoritetaan graafikyselyn omaisesti linkkejä hyödyntämällä. [OrientDB, 2018a]

Tarkastellaan seuraavaksi eri tietokantatuotteiden konkreettisia eroja, tutkielmassa esiteltyjä ominaisuuksia vertailemalla. Taulukossa 1. on vertailtu MySql-relaatiotietokantaa, Neo4j-graafitietokantaa, sekä OrientDB-monimallitietokantaa. Vertailussa on tarkasteltu, tukeeko kyseinen tietokanta tiettyä ominaisuutta vai ei, ja mahdollisesti lyhyesti selitetty tarkennuksia asiaan liittyen.

Ominaisuus	**MySql**	**Neo4j**	**OrientDB**
Lisenssi	Open Source GPL / Kaupallinen	Open Source GPL	Open Source Apache 2
Schema	Kyllä	Ehkä, (Muodostuu määritellyistä indekseistä ja eheysrajoitteista)	Kyllä, (Schema-less, schema-full, schema-hybrid)
Eheysrajoitukset (IC)	Kyllä	Kyllä	Kyllä
ACID-transaktiot	Kyllä	Kyllä	Kyllä
SQL	Kyllä	Ei	Kyllä
Cypher	Ei	Kyllä	Ei

Gremlin	Ei	Kyllä	Kyllä
Navigointi	Liitos	Linkki (fyysinen linkitys muistissa)	Linkki (fyysinen linkitys muistissa)
Olio-mallinnus	Kyllä (ORM)	Kyllä (OGM)	Kyllä (OGM)
Indeksit	Kyllä	Kyllä	Kyllä
Triggerit	Kyllä	Ehkä, (Ei löydy suoraa tukea, mutta on toteutettavissa)	Kyllä (Hook)
REST API	Ei	Kyllä	Kyllä
Tekstihaku	Kyllä	Kyllä (Lucene)	Kyllä (Lucene)

Taulukko 1.

6.3.1. Eroavaisuudet kyselyissä

Keskeinen ero relaatiotietokantojen ja graafitietokantojen välillä löytyy niihin kohdistetuissa kyselyissä. Tarkastellaan seuraavaksi eroavaisuuksia käytännössä, hyödyntäen edellä mallinnettua esimerkkiä. Tietovaraston sovellusalasta ja rakenteesta on mahdollista tunnistaa tarpeita, joita olemassa olevasta tietokantatoteutuksesta todennäköisesti haluttaisiin saada selville. Muodostetaan seuraavaksi, esimerkinomaisesti muutama kysely, joiden avulla voidaan vertailla kuinka kyselyt relaatiotietokantaan ja graafitietokantaan eroavat toisistaan.

Kyselyjä tehtäessä relaatiotietokannan etuna voidaan nähdä SQL-lauseen palauttaman tuloksen muodon ja tietotyyppien ennakoitavuus. Käytännössä tiedämme aina mitä sarakkeita kukin taulu sisältää, ja tiedämme niiden tietotyypit. Tämä on toki mahdollista toteuttaa myös graafitietokannassa, mutta tällaisten rajoitteiden määritteleminen syö pohjaa graafitietokannan keskeisimmältä ominaisuudelta, joka on sen kyky toimia muodoltaan määrittelemättömän (schema-less). Toisaalta gaafitietokanta on monissa kyselyissä tehokkaampi. Lisäksi monet kyselyt on mahdollista toteuttaa jollakin graafikyselykielellä luettavammin, kuin SQL:llä (vrt. kappale 5.2).

Tutkitaan ensimmäisessä vaiheessa kyselyjä, jotka sisältävät linkki- ja liitosoperaatioita. Tämän tyyppiset kyselyt sisältävät tietomallin rakenteen mukaan suurimmat erot relaatiotietomallin ja graafitietomallin välillä. Toisessa vaiheessa tutkitaan kyselyjä, joissa on hyödynnetty funktioita ja aggregointeja, jotka ovat yleisesti käytettyjä toimenpiteitä jalostettaessa tietoa erilaisiin näkymiin. Lopuksi tutkitaan hieman mahdollisuuksia tietojen päivitykseen kyselyjä hyödyntämällä. Kyselyjen muodostuksessa on käytetty apuna Hovin [2011] koostamia SQL-kyselyjä ja sovellettu joitain näistä kohdistumaan tutkielmassa käsiteltyyn tietovarastoon. Kaikki SQL-kyselyissä käytetyt operaatiot ovat ANSI-standardin tukemia, mutta tietokantatuotteiden välillä on tästä huolimatta joitakin syntaksieroja. Tässä luvussa käsitellyt SQL-kyselyt ovat kirjoitettu MySQL:n tukeman syntaksin mukaan.

6.3.1.1 Linkki- ja liitosoperaatiot

Ensimmäisessä esimerkissä tehdään haku, jolla haetaan kaikki henkilöt, jotka työskentelevät projektissa, jonka projektinumero on 100. Koodiesimerkissä 5 on muodostettu kysely SQL-lauseena ja koodiesimerkissä 6 vastaava kysely on muodostettu Cypher-kyselynä.

```
SELECT EMPLOYEE.EName
FROM EMPLOYEE
JOIN WORKS_ON ON (EMPLOYEE.ESsn = WORKS_ON.ESsn)
WHERE WORKS_ON.PNumber = 100;
```

Koodiesimerkki 5.

```
MATCH (e:EMPLOYEE)-[:WORKS_ON]->(p:PROJECT)
WHERE p.PNumber = '100'
RETURN e.EName
```

Koodiesimerkki 6.

Tämä esimerkki on hyvin yksinkertainen ja molemmat kyselyt ovat melko suoraviivaisia. Tästä voidaan kuitenkin havaita linkki- ja liitossuhteiden ero. SQL-kyselyssä liitossuhteet luodaan JOIN-operaattorin avulla, kun vastaavasti Cypherissä entiteettityyppien välinen linkitys muodostetaan linkitysten (`-[:WORKS_ON]->`) avulla, joita merkitään suhteen suunnan mukaisilla nuolilla. Neo4j-tietokannassa nuolen suunnalla on myös merkitys. Kyselyssä on mahdollista käyttää suhteita ilman suuntaa, jolloin kyselyyn merkittävästä nuolesta jätetään kärki pois (`-[:WORKS_ON]-`). Tällöin kuitenkin tietokannassa joudutaan käsittelemään kaikki solmua koskevat suhteet, suunnasta riippumatta, jolloin kyselyn suoritusaika pitenee. Suhteita luotaessa suunta on merkittävä, sillä Neo4j tallentaa aina suhteelle suunnan. Näin ollen myös solmu, johon suhteita liittyy tietää aina mitkä suhteet ovat lähteviä (outgoing) ja mitkä suhteet ovat saapuvia (incoming). [Graph grid, 2018]

Toinen tarpeellinen tieto voisi olla tunnistaa henkilöt, joilla on huollettavia ja työskentelevät projektissa, jonka laitos sijaitsee Espoossa.

```
SELECT EMPLOYEE.ESsn
FROM EMPLOYEE
JOIN WORKS_ON ON (EMPLOYEE.ESsn = WORKS_ON.ESsn)
JOIN PROJECT ON (WORKS_ON.PNumber = PROJECT.PNumber)
JOIN DEPARTMENT ON (PROJECT.DNumber = DEPARTMENT.DNumber)
JOIN DEPT_LOCATIONS ON (DEPARTMENT.DNumber = DEPT_LOCATIONS.DNumber)
WHERE (SELECT count(*) FROM DEPENDENT WHERE EMPLOYEE.ESsn = DEPENDENT.ESsn) > 0 AND
DEPT_LOCATIONS.Location = 'Espoo';
```

Koodiesimerkki 7.

```
MATCH (e:EMPLOYEE)-[:WORKS_ON]->(p:PROJECT)<-[:CONTROLS]-(d:DEPARTMENT)-[:LOCATION_OF]-
>(l:DEPT_LOCATION)
WHERE size((e)-[:DEPENDENT_OF]->(:DEPENDENT)) > 0 and l.Location = 'Espoo'
RETURN e.ESsn
```

Koodiesimerkki 8.

Tässä esimerkissä linkki- ja liitossuhteiden ero korostuu, kun kysely kattaa laajemmin eri entiteettityyppejä. Samaan aikaan mielestäni korostuu Cypher-kyselyjen luettavuus, kun liitoksia on paljon.

Kolmanneksi saatettaisiin tarvita tietoa, Espoossa sijaitsevan laitoksen esimiesten alaisuudessa työskentelevistä työntekijöistä.

```
SELECT e.EName
FROM EMPLOYEE AS e
JOIN EMPLOYEE AS e2 ON (e.SuperSsn = e2.ESsn)
JOIN DEPARTMENT AS d ON (e2.DNumber = d.DNumber)
JOIN DEPT_LOCATIONS AS dl ON (d.DNumber = dl.DNumber)
WHERE dl.Location = 'Espoo';
```

Koodiesimerkki 9.

```
MATCH (e:EMPLOYEE)<-[:SUPERVISES*1..]-(e2:EMPLOYEE)-[:WORKS_FOR]->(d:DEPARTMENT)-
[:LOCATION_OF]->(l:DEPT_LOCATION)
WHERE l.Location = 'Espoo'
RETURN e.EName
```

Koodiesimerkki 10.

Tässä esimerkissä on liitosten lisäksi hierarkkisuutta, joka tuo esiin Cypher-kyselyn hyvät puolet hierarkioiden käsittelyssä. Koodiesimerkissä 9 on luotu SQL-kysely, jossa on liitos työntekijöiden välillä heidän esimiessuhteensa avulla. Tässä kyselyssä on huomioitu vain yksi hierarkiataso. Mikäli tasoja haluttaisiin huomioida enemmän, olisi EMPLOYEE-tauluun tehtävä lisää JOIN-operaatioita tai hyödynnettävä jotakin tietokantatuotteen omaa hierarkiankäsittelyyn tarkoitettua lauseketta, kuten Oraclen CONNECT BY. Kyselyssä on myös annettu kyselyn käsittelemille tauluille aliaksia, käyttäen avainsanaa AS. Tämä on tehty osaltaan helpottamaan kyselyiden muodostamista ja luettavuutta. Taulut on tällöin mahdollista nimetä kyselyyn sopivammalla ja tarvittaessa lyhyemmällä nimellä. Koodiesimerkissä 10 on sama kysely Cypher-

kyselynä, mutta Cypher-kyselyssä on suoraan huomioituna kaikki hierarkiatasot. Eli käytännössä kyselyssä huomioidaan suoraan myös eri esimiesportaiden alaiset työntekijät.

6.3.1.2 Funktiot ja ryhmittely

Tarkastellaan seuraavaksi kyselyjä, joissa hyödynnetään erilaisia funktioita ja ryhmittelyjä. Yleinen virhetilanne tietovarastossa on duplikaattirivit. Tällaisia toistuvia, toisiaan vastaavia rivejä saattaa ilmaantua tietovarastoon esimerkiksi migraatioiden yhteydessä, jolloin tietoa siirretään toisesta tietovarastosta toiseen. Tarkastellaan seuraavaksi, kuinka relaatiotietokannassa on mahdollista paikallistaa taulukohtaiset moninkertaiset ilmentymät. Koodiesimerkissä 11 on SQL-kysely, joka etsii Project-taulusta projektin nimen, sijainnin ja laitoksen numeron perusteella toistuvia rivejä. Tässä tapauksessahan taululle on määritelty uniikki pääavain, joten rivit eivät ole aidosti duplikaatteja, mutta sisällöltään nämä saattavat olla toistuvia.

```
SELECT DNum, PName, PLocation
FROM PROJECT
GROUP BY DNum, PName, PLocation
HAVING COUNT(*) > 1;
```

Koodiesimerkki 11.

Kyselyn GROUP BY-lause määrittelee sarakkeet, joiden mukaan tulos ryhmitellään. Tämän jälkeen tuleva HAVING-lauseke on edellisen ryhmittelyn tarkenne, jolla rajataan ryhmittelyyn vain rivit, joita esiintyy useammin kuin kerran.

```
MATCH (n:Project)
WITH n.DNum as num, n.PName as name, n.PLocation as location, count(*) as count
WHERE count > 1
RETURN num, name, location;
```

Koodiesimerkki 12.

Koodiesimerkissä 12 on vastaava kysely Cypher-graafikyselynä. Cypher-kyselyssä ryhmittelyjoukon lukumäärät on mahdollista saada selville käyttämällä WITH-lauseketta. WITH-lauseketta käytettäessä on attribuuteille ja funktion tuloksille pakko antaa alias, jonka avulla kyselyn myöhemmissä osissa tuloksia voidaan käyttää. WITH-lausekkeen avulla on mahdollista manipuloida tulostetta, ennen kuin se välitetään sitä seuraaville kyselyn osille. WITH-lauseketta käytetään rajaamaan tulosjoukkoa, ennen kuin se välitetään toiselle MATCH-lausekkeelle. Vastaavasti kuin SQL-kyselyssä, sarakkeiden eli tässä tapauksessa ominaisuuksien (property) uudelleennimeäminen (alias) tapahtuu avainsanalla AS. Toinen käyttötapaus on ryhmittelyt, kuten tässäkin tapauksessa. WITH-lausekkeen jälkeen kyselyssä rajataan WHERE-ehdossa ryhmittelyn

tulokset vain niihin, joiden lukumäärä on yli yksi. Lopuksi palautetaan näiden toistuvien solmujen ominaisuuksia. [Neo4j, 2018h]

Tämänkaltaisesta kyselystä on vaikea löytää eroja eri SQL- ja Cypher kyselyjen väliltä. Kuitenkin olennainen ero, jota kyselyjen ulkomuodosta ei näy on tällaisessa tapauksessa suorituskyky. Graafitietokannat eivät ole erityisen tehokkaita vain listattaessa solmuja läpi. Graafitietkannat usein, kuten Neo4j:kin, tukevat indeksejä, mutta indeksit eivät ratkaise puhtaasti tämänkaltaista suorituskykyongelmaa. Kuten relaatiotietokantojen optimoinnistakin tiedetään, niin indeksihaut eivät ole tehokkaampia, mikäli rivimäärät kasvavat tarpeeksi suuriksi. Tämä johtuu siitä, että indeksihauissa joudutaan tekemään hajalevylukuja, jotka ovat hitaampia suorittaa kuin suorat taulurivien lukuoperaatiot. [Hovi, 2011]

Jalostettaessa tietoa järjestelmän käyttöön, tarvitaan tuloksen saamiseksi yleensä erinäisiä laskennallisia tai koostavia operaatioita. Näihin on apuna koostefunktiot, jotka ovat myös hyvin yleisesti käytettyjä SQL-kyselyissä. SQL tukee koostefunktioita, kuten AVG, SUM, MIN, MAX, COUNT, sekä laajaan kirjon muita matemaattisia funktioita. Näitä samoja funktioita on luonnollisesti toteutettu myös graafitietokantojen puolelle. Myös Cypher tukee kyselyissä edellä mainittuja koostefunktioita.

Muodostetaan seuraavaksi kysely, jossa haetaan kussakin eri laitoksessa työskentelevien työntekijöiden lukumäärä ja näiden henkilöiden palkkojen keskiarvo, sekä kyseisen laitoksen nimi.

```
SELECT EMPLOYEE.DNum, AVG(EMPLOYEE.ESalary), COUNT(*), DEPARTMENT.DName
FROM EMPLOYEE
JOIN DEPARTMENT ON (EMPLOYEE.DNumber = DEPARTMENT.DNumber)
GROUP BY EMPLOYEE.DNum, DEPARTMENT.DName;
```

Koodiesimerkki 13.

Esimerkin SQL-kyselyssä käytettiin JOIN-operaatiota tekemään liitos työntekijä- ja laitostaulujen välille, jotta saadaan laitoksen nimi kyselyn vastauksiin mukaan. Tämän jälkeen GROUP BY-lauseella ryhmiteltiin tulos laitosten mukaan. Kyselyn palauttamiin sarakkeisiin on määritelty palkkojen keskiarvo, käyttämällä AVG-funktiota, joka laskee keskiarvon ryhmittelyjoukon sisältä löytyvistä palkoista. Lisäksi SELECT-lauseen palauttamissa sarakkeissa on käytetty COUNT-funktiota, joka palauttaa ryhmittelyjoukon rivien lukumäärän.

```
MATCH (e:Employee)-[:WORKS_FOR]->(d:Department)
WITH d.DNumber as dnum, d.DName as dname, avg(e.ESalary) as avgSalary, count(*) as count
RETURN dnum, avgSalary, count, dname;
```

Koodiesimerkki 14.

Vastaava kysely Cypher-kyselynä on jälleen melko samankaltainen. Kyselyistä on löydettävissä eroja, jotka ovat jo tulleet aikaisemminkin vastaan. Näitä ovat JOIN-operaation korvautuminen graafitietokannan suhde-linkityksellä, sekä WITH-lausekkeen käyttö ryhmittelyssä. Näiden lisäksi kyselyssä palautetaan ominaisuuksia eri solmutyypeiltä. MATCH-lausekkeessa kullekin solmu- ja suhde-tyypille on mahdollista määritellä muuttuja, jonka avulla kyseiseen solmuun on mahdollista viitata myöhemmissä lausekkeissa. Koodiesimerkin 14 kyselyssä Employee-solmutyypille on määritelty muuttuja e, johon viitataan WITH-lausekkeessa. Tässä vaiheessa muuttujan avulla viitataan solmun tiettyyn ominaisuuteen ja annetaan sille alias, jonka avulla siihen voidaan viitata myöhemmin. Eli esimerkiksi `e.DNumber as dnum` määrittelee aliaksen dnum, joka viittaa työntekijän laitosnumeroon. Vastaava käsittely tapahtuu Department-solmun osalta.

Tarkastellaan seuraavaksi, kuinka monesta eri laitoksesta työntekijöitä on, mitkä nämä laitokset ovat, kuinka monta merkkiä laitoksen nimessä on, sekä mitkä ovat laitoksen nimen kolme ensimmäistä kirjainta isolla kirjoitettuna. Kyselyssä hyödynnetään nyt myös merkkijonofunktioita, jolloin päästään vertailemaan lisää, kuinka SQL:n ja Cypherin funktiot eroavat toisistaan.

```
SELECT COUNT(DISTINCT(EMPLOYEE.DNum)), DEPARTMENT.DName, LENGTH(DEPARTMENT.DName),
UPPER(SUBSTRING(DEPARTMENT.DName,1,3))
FROM EMPLOYEE
JOIN DEPARTMENT ON (EMPLOYEE.DNumber = DEPARTMENT.DNumber)
GROUP BY EMPLOYEE.DNum, DEPARTMENT.DName, LENGTH(DEPARTMENT.DName),
UPPER(SUBSTRING(DEPARTMENT.DName,1,3));
```

Koodiesimerkki 15.

Koodiesimerkin 15 kysely on rakenteeltaan hyvin samankaltainen, kuin edellisen koodiesimerkin 13 kysely. Esimerkissä 15 on käytetty laajemmin SQL:n tukemia funktioita SELECT-lauseen palauttamissa tiedoissa. Vastauksen ensimmäisessä sarakkeessa on sisäkkäin kaksi funktiota, joista DISTINCT poimii ensin uniikit laitosnumerot ja ulompi funktio, COUNT laskee näiden esiintymien lukumäärän. Kolmannessa sarakkeessa LENGTH-funktiolla palautetaan laitoksen nimen merkkijonon pituus. Lopuksi, viimeisessä sarakkeessa poimitaan SUBSTRING-funktiolla laitoksen nimen kolme ensimmäistä kirjainta ja UPPER-funktiolla muutetaan nämä isoiksi kirjaimiksi.

```
MATCH (e:Employee)-[:WORKS_FOR]->(d:Department)
WITH d.DName as dname
MATCH (e:Employee)-[:WORKS_FOR]->(d:Department)
RETURN count(distinct(d.DNumber)),dname, size(dname), toUpper(substring(dname, 0, 3));
```

Koodiesimerkki 16.

Tämä esimerkki osoittautui haastavimmaksi toteuttaa Cypher-kyselynä. Käytettäessä WITH-lauseketta, ei ollut löydettävissä yksinkertaista tapaa, jolla WITH-lausekkeen avulla ryhmittelyrivien määrän olisi saanut helposti selville. Tässä tapauksessa oli tehtävä kaksi sisäkkäistä MATCH-lauseketta. Ensimmäisessä ryhmiteltiin rivit laitoksen nimen perusteella, jonka jälkeen tehtiin uusi valinta työntekijöistä, jotka työskentelevät laitoksessa. Tämän pohjalta voitiin lopulta palauttaa lukumäärä uniikeista työntekijän laitosnumeroista. Muut RETURN-osassa käytetyt funktiot vastaavat pitkälti SQL-standardista tuttuja merkkijonofunktioita. Ainoa ero on Cypher-kyselyn size-funktio, joka vastaa SQL:n length-funktiota. Tämä onkin hyvin luonnollista, sillä graafikyselyssä length-sanalle on luonnollisempi käyttötapaus polkukyselyissä.

Muodostetaan vielä lopuksi yksi ryhmittelylauseketta GROUP BY hyödyntävä kysely, jossa etsitään PROJECT-taulusta sijainti, jossa on eniten projekteja.

```
SELECT PLocation, count(*) AS count
FROM PROJECT
GROUP BY PLocation
HAVING count(*) >= ALL
    (SELECT count(*)
    FROM PROJECT
    GROUP BY PLocation)
```

Koodiesimerkki 17.

Kuten koodiesimerkistä 17 nähdään, niin yksinkertaiselta vaikuttava kysely on yllättävän hankala muodostaa SQL-kyselynä. Kyselyssä ryhmitellään PROJECT-taulusta löytyvät rivit sijainnin mukaan. Näistä ryhmistä rajataan mukaan HAVING-lausekkeella vain sellaiset, joissa on rivejä enemmän tai yhtä paljon kuin missä tahansa ryhmässä. Näin saadaan sijainti tai sijainnit, joissa on eniten projekteja.

```
MATCH (p:Project)
WITH p.PLocation as location, count(*) as occurences
WITH MAX(occurences) as max
MATCH (p:Project)
WITH p.PLocation as location, count(*) as occurences, max
WHERE occurences = max
RETURN location, occurences
```

Koodiesimerkki 18.

Koodiesimerkin 18 Cypher-kyselyn muodostamisessa tulee vastaan sama ongelma, kuin edellisessä SQL-muotoisessa kyselyssä. Kyselyssä on ensin haettava paikkakuntien projektien

lukumäärät. Tämän jälkeen näistä lukumääristä otetaan talteen kyselyn jatkoa varten suurin lukumäärä. Tätä lukumäärää hyödynnetään seuraavassa MATCH-täsmäytyslausekkeessa, jonka ehtoihin asetetaan aiemmin talteen otettu suurin lukumäärä vastaamaan ilmentymien lukumäärää. Kysely on rakennettu hyvin samankaltaisesti sisäkkäisiä kyselyjä hyödyntämällä, kuin SQL-kyselykin.

6.3.1.3 Päivitys

Kyselyjen ohella tietokantaan kohdistetaan usein myös luonti- ja päivitysoperaatioita. Päivitys operaatioilla tehdään muutoksia olemassa oleviin tauluihin tai taulun riveihin. Päivitys voi siis kohdistua taulun rakenteeseen, joka vastaa kaaviomuutosta, tai taulun olemassa oleviin riveihin. Muodostetaan seuraavaksi päivitysoperaatioita tutkielmassa käsiteltyyn tietovarastoon.

Usein tietokannassa on tarpeellista muokata tai luoda uusia tietueita olemassa olevan tiedon pohjalta. Tällaisissa tapauksissa on mahdollista hyödyntää kyselyä päivityksen apuna. Päivitetään seuraavaksi kaikkien työntekijöiden palkat vastaamaan 90 prosenttia laitoksen johtajan palkasta ja pyöristetään tulos kahden desimaalin tarkkuuteen.

```
UPDATE EMPLOYEE e
SET ESalary = (
    SELECT CASE e2.ESsn
        WHEN e3.ESsn THEN e3.ESalary
        ELSE round(e3.ESalary*0.9,2)
        END
    FROM EMPLOYEE AS e2
    JOIN DEPARTMENT AS d ON (e.DNumber = d.DNumber)
    JOIN EMPLOYEE AS e3 ON (d.MgrSsn = e3.ESsn)
    WHERE e.ESsn = e2.ESsn
)
```

Koodiesimerkki 19.

Koodiesimerkissä 19 SET-lauseen alikysely hakee laitoksen johtajan palkan, josta lasketaan työntekijän uusi palkka ja pyöristetään. Alikyselyn SELECT-lausekkeessa on käytetty CASE-ehtoa, jolla varmistetaan, ettei johtajan itsensä palkka pienene. Eli koska johtaja työskentelee laitoksessa kuten työntekijätkin, pitää kyselyssä huomioida, että mikäli tarkastelemme parhaillaan johtajan palkkaa, pidetään se ennallaan. Tämän käsittelyn lopputulos sijoitetaan tauluun vanhan arvon tilalle.

```
MATCH (e:Employee)-[:WORKS_FOR]->(d:Department)<-[:MANAGES]-(e2:Employee)
SET e.ESalary = CASE e.ESsn
        WHEN e2.ESsn THEN e2.ESalary
```

```
        ELSE round(100 * (e2.ESalary*0.9)) / 100
        END
RETURN e.ESalary, e2.ESalary
```

Koodiesimerkki 20.

Tämä kysely on hieman suoraviivaisempi Cypher-kyselynä. Koodiesimerkissä 20 nähdään, kuinka SQL-kyselyn SET-lausekkeen alikyselyssä olevat JOIN-operaatiot voidaan Cypher-kyselyssä koostaa suoraan ensimmäisessä MATCH-lausekkeessa. Käytännössä graafikyselyssä haetaan suoraan työntekijän ja johtajan suhde laitokseen, jolloin myöhemmin kyselyn rakenteessa voidaan viitata suoraan näihin solmuihin ja suhteisiin. Cypher-kyselykieli tukee suoraan samaa SQL:n kaltaista syntaksia CASE-operaatiolle, joten tämä käsittely on täysin vastaava esimerkin 19 kanssa. Ongelmaksi Cypher-kyselyssä muodostuu round-funktio, joka ei tue parametrina annettavaa desimaalien lukumäärää, vaan pyöristää aina lähimpää kokonaislukuun. Tämän takia kyselyssä joudutaan muodostamaan desimaalit itse, tekemällä kerto- ja jako-operaatiot lopputulokselle.

Vertaillaan lopuksi tapoja, kuinka SQL- ja Cypher-kielillä tallennetaan tietokantaan uutta tietoa. Lisätään tietovarastoon uusi laitos, sekä tälle laitokselle kaksi uutta työntekijää, joista toinen on laitoksen johtaja.

```
INSERT INTO DEPARTMENT (DNumber,MgrSsn,DName)
VALUES (400,112233,'Markkinointi');
INSERT INTO EMPLOYEE (ESsn,DNumber,EName)
VALUES (112233,400,'Juhani');
INSERT INTO EMPLOYEE (ESsn,DNumber,EName)
VALUES (224455,400,'Esko');
```

Koodiesimerkki 21.

```
CREATE (:Employee{ ESsn:112233, EName: "Juhani" })-[:MANAGES]->(:Department { DName:
'Markkinointi', MgrSsn: 112233, DNumber: 400 })<-[:WORKS_FOR]-(:Employee{ ESsn:224455,
EName: "Esko" })
```

Koodiesimerkki 22.

Koodiesimerkissä 21 on tavanomaisia SQL-lauseita, joilla luodaan tauluihin uusia rivejä. Ensimmäisellä rivillä luodaan DEPARTMENT-tauluun uusi laitos, jonka nimi on ”Markkinointi”. Toisessa INSERT-lauseessa luodaan uusi työntekijä ”Juhani”, jolle asetetaan DNumber-sarakkeeseen edellä luodun laitoksen laitosnumero. Tällä tavoin luotiin viiteavaimen avulla liitos työntekijän ja laitoksen välillä. Lisäksi työtekijän ESsn-avain asetettiin uuden laitoksen MgrSsn-sarakkeen arvoksi, minkä avulla saadaan liitos laitoksesta laitoksen johtajaan. Vastaavien uusien

monikoiden ja näiden välisten liitosten luominen graafitietokantaan tapahtuu hyvin eri tyyppisesti. Koodiesimerkissä 22 tehdään vastaavat asiat graafimaisesti. Käytännössä Cypher-kielellä toteutettuna kyseisten tietojen vieminen tietokantaan onnistuu yhdellä CREATE-lauseella. Lause sisältää ensin uuden Employee-tyyppisen solmun luonnin. Tälle solmulle luodaan MANAGES-tyyppinen linkitys Department-solmuun, joka myös luodaan samalla kertaa. Tämän jälkeen uuden Department-tyyppisen solmun toiselle puolelle luodaan WORKS_FOR-tyyppisen suhteen kautta jälleen uusi Employee-solmu, jolla luodaan uusi työntekijä kyseiselle laitokselle. Lisäksi graafitietokannassa viiteavainsarakkeet voidaan jättää pois, jolloin tässä tapauksessa työntekijälle ei tarvitse erikseen tallentaa laitosnumeroa, vaan se ilmenee linkityksen kautta.

Edellä kuvatuissa luontilauseissa huomataan, kuinka eri tavoin relaatio- ja graafitietokanta tulkitsevat suhteita. Käytännössä SQL:n luontilauseissa on olennaista, että liitosten viiteavaimet menevät oikein, mutta suhteen määrittelystä ei tarvitsee sen enempää kantaa huolta. Cypher-kielellä vastaavasti on erityisen olennaista määritellä suhde, joka solmujen välille halutaan syntyvän, mutta tämän ansiosta ei tarvitse huolehtia viitetiedoista. Tällöin solmuun tarvitsee tallentaa vain sille itselleen kuuluvaa tietoa, eikä sen tarvitse huolehtia viitetiedoista muihin solmuihin.

6.3.1.4 Yhteenveto

Graafikyselyjä kirjoitettaessa on nopeasti havaittavissa, että Cypher-kielen syntaksi on melko tiivistetty ja kyselyiden muodostaminen on verrattain intuitiivista. Erityisesti eri entiteettityyppien välisiä suhteita käsittelevien kyselyjen muodostaminen tuntuu graafikyselynä luontevalta. Vaikka graafitietokannat ja graafikyselykielet olisivat ennestään tuntemattomia, niin Cypher-kyselyiden muodostaminen on nopeasti opittavissa. Uskoisin, että suhteita käsittelevien kyselyiden hahmottaminen SQL-kyselykielellä on verrattain haastavampaa. Toisaalta SQL on monille syntaksiltaan, rakenteeltaan, sekä tarjolla olevien funktioiden ja lausekkeiden puolesta tuttu. Lisäksi SQL on pitkän ajan saatossa jalostettu ja pitkälle kehittynyt ohjelmointikieli, joka on näin ollen luotettava ja jolle on saatavilla kattavasti dokumentaatiota.

Yksinkertainen tiedonhaku, sekä tiedonkäsittely erilaisten funktioiden avulla on SQL- ja Cypher-kielissä hyvin samankaltaista. Tietyn entiteettityypin attribuutin hakeminen, kuten merkkijonon kolmen ensimmäisen merkin poimiminen, toistuvat hyvin vastaavilla tavoilla molemmissa kielissä. Erot sen sijaan syntyvät parhaiten suhteiden määrittelyssä ja käsittelyssä. Mitä enemmän kyselyssä on liitoksia, missä tahansa kyselyn osassa, sitä paremmin Cypher-kysely on muodostettavissa SQL-kyselyyn verrattuna. Vastaava tilanne on tietojen luomisessa. Kun SQL-kyselyssä joudutaan toistamaan tietoa usean luontirivin ja viiteavainten muodossa, on Cypher-kyselyllä linkityksiä sisältävien rivien luonti suoraviivaista. Toisaalta toisteisuus relaatiotietokannassa ei aina ole huono asia. Viiteavainten avulla luotujen relaatiosuhteiden avulla on mahdollista toteuttaa kyselyjä ilman eksplisiittistä suhteen luomista. Mikäli graafitietokannassa ei ole suhdetta luotu, ei suhdetta ole olemassa, vaikka solmuilta löytyisikin toisiinsa viittaavia tietoja.

Cypher-kielen kankeus tuntui tuleva tulosten ryhmittelyssä ja ryhmien tietoihin liittyvissä käsittelyissä. SQL:ssä listausten ryhmittely ja ryhmän tai ryhmien tietojen käsittely ja aggregointi on melko selkeää. Sen sijaa Cypher-kielellä vastaavat asiat tuntuvat melko haastavilta. Tähän toki vaikuttaa myös kokemuksen laajuus. Kuitenkin, mielestäni raporttimainen tulosten käsittely, jossa rivejä ryhmitellään ja näille lasketaan erilaisia aggregointeja, on helpompi toteuttaa SQL-kyselynä.

Cypher-kyselykielessä, kuten muissakin graafikyselykielissä, etuna on polkuorientoituneiden kyselyjen kirjoittamisen intuitiivisuus. Junkkari ja muut [2016] tutkivat kuinka SQL:ään laajennettu polkukyselyominaisuus PathSQL vaikutti polkuorientoituneiden kyselyiden muodostamiseen. Tutkimuksessa havaittiin, että testihenkilöt kirjoittivat tällaisia kyselyitä huomattavasti nopeammin, kuin perinteisellä SQL:llä, sekä tekivät vähemmän virheitä. Vastaavasti kuin graafikyselykielet, PathSQL kätkee liitoskyselyn liitosehdot, joka tekee kyselyiden muodostamisesta huomattavasti nopeampaa ja kyselyistä luettavampia. Vastaava liitosehtojen piilotus tapahtuu graafikyselyissä väistämättä, kun liitoksia ei ole. Näin ollen, uskon että graafikyselyissä vastaava ilmiö voi hyvinkin toistua.

7. Johtopäätökset

Tutkielman tavoitteena oli tuoda esiin graafitietokantojen ja relaatiotietokantojen ominaisuuksia ja eroavaisuuksia. Lisäksi tavoitteena oli, että näiden esiteltyjen ominaisuuksien, sekä tutkielmassa tunnistettujen huomioiden avulla, olisi helpompi tunnistaa tilanteita, jolloin tietojärjestelmän tietokannan uudelleenvalinta voisi olla perusteltua. Tutkielmassa käytiin läpi alan tieteellisissä julkaisuissa esiteltyjä tietomalleja, sekä pohdittiin näiden ominaisuuksia ja eroavaisuuksia. Lisäksi tutkielmassa tutustuttiin NoSQL-tietovarastoihin, käytiin läpi niiden tapoja varastoida tietoa, sekä vertailtiin niiden eroavaisuuksia perinteisiin tietokantoihin. Erityinen huomio kiinnitettiin graafitietokantoihin, niiden ominaisuuksiin ja tapoihin käsitellä ja mallintaa tietoa. Lopuksi tehtiin konkreettinen muunnos eräästä ER-tietomallilla kuvatusta kaaviosta relaatiomalliin ja graafimalliin, sekä pohdittiin näiden eroavaisuuksia.

Tietokantavalintaa tehtäessä on erityisen tärkeää tuntea syvällisesti tietojärjestelmän sovellusala (domain), sekä ymmärtää tietojärjestelmän vaatimusten ja käyttötapausten kautta muodostuva ns. business-logiikka. Business-logiikalla tarkoitetaan tietojärjestelmään ohjelmoituja osia, jotka sisältävät todellisen logiikan tietojärjestelmän tiedon kulun kannalta. Lisäksi on hyvä tiedostaa tarve ja ongelmat, joita tietojärjestelmällä pyritään ratkaisemaan. Näiden tietojen avulla on mahdollista löytää toistuvia kaavoja ja erityisiä piirteitä järjestelmästä, jotka voivat antaa viitteitä siitä, millaisen tietomallin avulla järjestelmän tietovarasto olisi järkevintä mallintaa. Erityisesti tunnistettavia seikkoja ovat tiedon hierarkkisuus ja polkuorientoituneisuus, tai näiden tarve tiedon esittämisessä tai käsittelyssä. Tätä tunnistamista voi pyrkiä helpottamaan piirtämällä järjestelmän sovellusala kuvaksi. Tähän kuvaan tulee tunnistaa eri entiteettityypit ja suhteet, joita sovellusala sisältää. Lisäksi tulee tunnistaa yleisimmät kyselyt, joita järjestelmässä tarvitaan. Mikäli kaaviossa ilmenee runsaasti suhteita eri entiteettityyppien välillä ja kyselyissä tarvitaan usein tietoa näistä suhteista, saattaa graafitietokanta olla oikea valinta.

Valintaa tehtäessä voi olla järkevää pohtia myös käytännön työskentelyä valitun tietokannan kanssa. Yksi tähän vaikuttava tekijä on tietokantaan tehtävät kyselyt ja kuinka hyvin ne soveltuvat halutun tiedon hakemiseen. Tiedon ollessa polku- tai hierarkiaorientoitunutta on graafikyselyissä myös kyselyjen muodostamisen näkökulmasta etua. Kyselyt on tällöin mahdollista tiivistää lyhyempään ja ymmärrettävämpään muotoon, mikä vähentää myös virheiden mahdollisuutta.

Relaatiotietokantoja käyttävissä järjestelmissä on myös mahdollista tunnistaa tiettyjä asioita, jotka voisivat viitata siihen, että järjestelmän tietomalli sopisi mallinnettavaksi paremmin jollain muulla tietomallilla kuin relaatiotietomallilla. Kappaleessa 3.3 käytiin läpi lista, joka on hyvä käytännön tapa nopeasti selvittää, kannattaako relaatiopohjaisen järjestelmän tietokantaratkaisun vaihtoa alkaa pohtia laajemminkin.

Olennaisia seikkoja tiedon rakenteen lisäksi, joita tietokantavalintaa tehtäessä on otettava huomioon, ovat tiedon määrä ja vaatimukset tietojärjestelmän suorituskyvyn suhteen. Tietojärjestelmän erityisvaatimuksena saattaa olla esimerkiksi hyvin pienet vasteajat, jolloin on erityisesti huomioitava järjestelmän ennakoitu tietomäärä, sekä tämän tietomäärän tehokas käsittely

ja lukeminen. Tällaisessa tapauksessa on täysin selvää, että tietokantaratkaisun kokonaisuuteen on panostettava. Vastaavasti järjestelmä saattaa nojata erityisen suuriin datamääriin, jolloin on löydettävä tietokantaratkaisu, jolla on kapasiteettia suoriutua erityisen suurten datamäärien käsittelystä. Tähän saattaisi olla ratkaisuna esimerkiksi jonkinlainen hajautettu tietovarasto.

Yleinen ratkaisu relaatiotietokantoja käyttävien tietojärjestelmien tehokkuuden parantamiseksi, ja loppukäyttäjien kokemien järjestelmän vasteaikojen pienentämiseksi, on tiedon denormalisointi. Denormalisoinnilla tarkoitetaan normalisoidun tiedon monistamista tai ryhmittelyä, jonka avulla pyritään parantamaan lukuoperaatioiden suorituskykyä. Tämä on usein toimiva ratkaisu, mutta kuormittaa järjestelmää ylläpidon kannalta, erityisesti mikäli denormalisointia tehdään toiseen tietovarastoon. Tällöin tietoa joudutaan ylläpitämään monessa eri paikassa, ja näin ollen tehdään kompromissi tiedon eheyden näkökulmasta. Tässä mielessä graafitietokanta saattaisi olla perusteltu ratkaisu, sillä sen kyky käsitellä vaativia tietokantakyselyjä on todella hyvä. Graafitietokannassa ei ole tarvetta rakentaa erillistä denormalisoitua tietovarastoa tukemaan tavanomaisia liitoskyselyjä, jotka ovat relaatiotietokannassa todella raskaita. Toki myös relaatiotietokannat tukevat indeksejä, mutta tietokantojen indeksit eivät välttämättä ole yhtä tehokkaita, kuin monet NoSQL-ratkaisut. Lisäksi tietokantaindeksit kuormittavat tiedon manipulointia, eivätkä näin ollen välttämättä tuo toivottua tehokkuutta.

Denormalisointi on kuitenkin usein väistämätön ratkaisu. Erityisesti tilanteissa joissa tietojärjestelmältä vaaditaan kykyä suoriutua vaativasta kyselyistä, jollaisiin ei ole relaatiotietokantaa suunniteltaessa varauduttu. Relaatiotietokanta onkin usein väärä tietokantavalinta tilanteissa, joissa tiedon muotoon ja määrään on vaikea varautua ennalta. Tai tilanteissa, joissa sovellusalan vaatimukset tiedonhallinnan kannalta muuttuvat usein. Relaatiotietokannan kanssa tällainen ketterä tietomallin kehittäminen ei ole helppoa.

Tietokannasta on myös olennaista tunnistaa sarakkeita, joiden tietotyypit on määritelty käsittelemään suuria tietomääriä. Tällaisia voi olla esimerkiksi BLOB tai CLOB tyyppiset sarakkeet, joissa yleensä varastoidaan tiedostoja. Mikäli relaatiotietokanta sisältää paljon tämän tyyppisiä sarakkeita, ei graafitietokanta ole välttämättä oikea ratkaisu tietojärjestelmän tietovarastoksi. Graafitietokantaa ei ole suunniteltu varastoimaan tiedostoja, jolloin tiedostojen käsittelylle tulisi kehittää jokin rinnakkainen tietovarasto. Tämän tyyppisiin tarpeisiin saattaisi soveltua parhaiten jokin dokumenttivarasto, kuten MongoDB.

Mitä enemmän tarjolla oleviin tietokantoihin tutustuu, sitä vaikeammalta valinta saattaa tuntua. Kilpailu tietokantamarkkinoilla tuntuisi olevan melko kovaa, ja eri tarjoajat pyrkivät nopeasti kehittämään tuotteitaan vastaamaan ominaisuuksiltaan kilpailijoidensa tuotteita. Näin ollen kenttä elää jatkuvasti. Tietokantavalintaa tehtäessä on kuitenkin hyvä tutustua referensseihin, joissa tietokantaa on hyödynnetty. Lisäksi on hyvä tutustua standardeihin, joita tietokanta tukee, jotta mahdollinen tietokantatuotteen vaihto jatkossa olisi mahdollisimman sujuva.

Viiteluettelo

[Angles and Gutierrez, 2008] Angles R. and Gutierrez C., Survey of graph database models. *ACM Computing Surveys* **40**, 1 (Feb. 2008).

[Angles et al., 2016] Angles, R., Arenas M., Barcelo P., Hogan A., Reutter J., Vrgoč D., Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys* **50**, 68 (Sept. 2017).

[Apenyo, 1999] Apenyo K., Using the entity-relationship model to teach the relational model. *ACM SIGCSE Bulletin* **31**, 2 (Jun. 1999), 78-80.

[Assad, 2007] Assad A., Leonhard Euler: A brief appreciation. *Wiley InterScience - Networks* **49** (2007) 190–198.

[Chamberlin and Boyce, 1974] Chamberlin, D. D. and Boyce, R. F. SEQUEL: A Structured English Query Language. In: *Proc of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, (1974), 249–64.

[Chandra, 2015] Chandra D. G., BASE analysis of NoSQL database. *Future Generation Computer Systems* **52**, (2015), 13-21.

[Chen, 1976] Chen P. P., The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems (TODS)* **1**, 1 (Mar. 1976).

[Codd, 1970] Codd E. F., A relational model of data for large shared data banks. *Communication of ACM* **13**, 6 (Jun. 1970), 377-387.

[Codd, 1980] Codd E. F., Data models in database management. In: *Proc. of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling,* (1980), 112-114.

[Codd, 1985] Codd E. F., Is your DBMS really relational? *Computerworld* **19**, 41 (Oct. 1985), 1-2.

[DataStax, 2018] DataStax Enterprise Graph. https://www.datastax.com/products/datastax-enterprise-graph. Viitattu 24.2.2018.

[DB-Engines, 2018] DB-Engines Ranking. https://db-engines.com/en/ranking. Viitattu 12.5.2018.

[Eessaar, 2016] Eessaar E., The Database Normalization Theory and the Theory of Normalized Systems: Finding a Common Ground. *Baltic Journal of Modern Computing* **4**, 1 (2016), 5-33.

[Elmasri and Navathe, 2000] Elmasri R. and Navathe S. B., *Fundamentals of Database Systems.* 3rd edition, Benjamin/Cummings, 2000.

[DZone, 2018] From Relational to Graph: A Developer's Guide. https://dzone.com/refcardz/from-relational-to-graph-a-developers-guide/. Viitattu 21.1.2018.

[Harrington, 2002] Harrington J., *Relational Database Design Clearly Explained.* 2nd edition, Morgan Kaufmann, 2002.

[Hibernate, 2018] Hibernate ORM. http://hibernate.org/orm/. Viitattu 24.2.2018.

[Bitnine, 2018] History of Databases and Graph Database. http://bitnine.net/blog-graph-database/history-of-databases-and-graph-database/. Viitattu 24.2.2018.

[Hovi, 2011] Hovi A., *SQL-opas*. Codenco, 10th edition, 2011.

[ISO/IEC 9075, 2018] ISO/IEC 9075-1:2016 Information technology -- Database languages -- SQL -- Part 1: Framework (SQL/Framework). https://www.iso.org/standard/63555.html. Viitattu 13.5.2018.

[Junkkari et. al., 2016] Junkkari M., Vainio J.,Iltanen K., Arvola P., Kari H. and Kekäläinen J., Path Expressions in SQL: A User Study on Query Formulation. *Journal of Database Management* **27**, 3 (Sept. 2015).

[Kleppmann, 2015] Kleppmann M., A Critique of the CAP Theorem, *CoRR*, (Sept. 2015).

[Kuznetsov and Poskonin, 2014] Kuznetsov S. and Poskonin A., NoSQL Data Management Systems. *Programming and Computer Software* **40**, 6 (Nov. 2014), 323–332.

[Kruntorad, 2009] Kruntorad J. M., History of The CA IDMS Database Management System. *IEEE Annals of the History of Computing* 31, 4 (Oct-Dec 2009).

[Mpinda et al., 2015] Mpinda S., Ferreira L., Ribeiro M. and Santos M., Evaluation of Graph Databases Performance Through Indexing Techniques. *International Journal of Artificial Intelligence & Applications (IJAIA)* **6**, 5 (Sept. 2015).

[Neo4j, 2018a] From Relational to Neo4j. https://neo4j.com/developer/graph-db-vs-rdbms/. Viitattu 3.3.2018.

[Neo4j, 2018b] From SQL to Cypher. https://neo4j.com/developer/guide-sql-to-cypher/. Viitattu 24.2.2018.

[Neo4j, 2018c] Graph Database Use Cases. https://neo4j.com/use-cases/. Viitattu 25.2.2018.

[Neo4j, 2018d] Graph Databases for Beginners: Why Data Relationships Matter. https://neo4j.com/blog/why-graph-data-relationships-matter/. Viitattu 3.3.2018.

[Neo4j, 2018e] Neo4j Constraints. https://neo4j.com/docs/developer-manual/current/cypher/schema/constraints/. Viitattu 3.3.2018.

[Neo4j, 2018f] Welcome to the Dark Side: Neo4j Worst Practices (& How to Avoid Them). https://neo4j.com/blog/dark-side-neo4j-worst-practices/. Viitattu 3.3.2018.

[Neo4j, 2018g] What is a Graph Database. https://neo4j.com/developer/graph-database/. Viitattu 24.2.2018.

[Neo4j, 2018h] WITH. https://neo4j.com/docs/developer-manual/current/cypher/clauses/with/. Viitattu 25.5.2018

[NOSQL databases, 2017] NOSQL databases. http://nosql-database.org/. Viitattu 10.1.2017.

[OrientDB, 2018a] OrientDB vs MongoDB. https://orientdb.com/orientdb-vs-mongodb/. Viitattu 24.3.2018.

[Pokorný et al. 2017] Pokorný J., Valenta M. and Kovačič J., Integrity constraints in graph databases, *Procedia Computer Science* **109** (2017), 975-981.

[Rae et. al., 2014] Rae I., Halverson A. and Naughton J. F., In-RDBMS inverted indexes revisited. In: *IEEE 30th International Conference on Data Engineering*, (2014), 352-363.

[OrientDB, 2018b] Relationships. https://orientdb.com/docs/last/Tutorial-Relationships.html. Viitattu 14.4.2018.

[Graph grid, 2018] Relationship Direction in Cypher is Important. https://www.graphgrid.com/relationship-direction-in-cypher-is-important/. Viitattu 19.5.2018.

[Sanders and Shin, 2001], Sanders G. L. and Shin S., Denormalization effects on performance of RDBMS, In: *Proc. of the 34th Annual Hawaii International Conference on System Sciences*, (2001), 9

[Shi, 2014] Shi J. Y., Symposium: CAP-Plus for Big Data. In: *Proc. of the 2014 International Conference on Big Data Science and Computing*, (2014), 1-5

[Siau and Cao, 2001] Siau, K., and Cao, Q. Unified modeling language (UML) - a complexity analysis. *Journal of Database Management* **12,** 1 (Jan. 2001), 26-34.

[Ullman, 1982] Ullman J. D., *Priciples of Database Systems*. 2nd edition, Computer Science Press, 1982.

[Vicknair et al., 2010] Vicknair C., Macias M., Zhao Z., Nan X., Chen Y. and Wilkins D., A comparison of a graph database and a relational database: a data provenance perspective. In: *Proc. of the 48th Annual Southeast Regional Conference*, (2010), 1-6.

[Msdn, 2017] What is a Transaction. https://msdn.microsoft.com/en-us/library/aa366402(VS.85).aspx. Viitattu 1.4.2017.

[OrientDB, 2018c] Why a Multi-Model Database?. https://orientdb.com/multi-model-database/. Viitattu 24.2.2018.

[ArangoDB, 2018] Why ArangoDB?. https://www.arangodb.com/why-arangodb/. Viitattu 24.2.2018.

[Wu et al., 2014] Wu L., Yuan L. Y. and You J. H., BASIC: An alternative to BASE for large-scale data management system, In: *IEEE International Conference on Big Data (Big Data)*, (2014), 5-14.