Tampereen yliopisto

Chi-Hao Lay

QUERY-BY-EXAMPLE GRAAFITIETOKANNASSA

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Toukokuu 2022

TIIVISTELMÄ

Chi-Hao Lay: Query-by-example graafitietokannassa
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2022

Verkkomaisen tiedon käsittely graafitietokannassa on vaihtoehto relaatiotietokannoille. Taulukkomuotoiset tietokannat ovat kehittäjille helppo ymmärtää, mutta graafitietokannat voivat vaatia osaamista graafiteoriasta. Erilaisia kieliä on olemassa ja ne ovat hyviä automaattiseen tiedonkäsittelyyn, mutta eivät mahdollista manuaalista tiedonhakua yleiseen käyttöön. Ongelmaa on relaatiotietokannoissa ratkaistu QBE-kyselykielellä, jota tässä tutkielmassa sovelletaan graafitietokantaan.

Tutkielmassa määritellään QBE-kyselykieli formaalisti graafitietokannalle, johdetaan kyselykielen syntaksi ja toteutetaan se taulukkomuotoisena alkuperäisen QBE-kyselykielen perusteella. QBE-kyselykieltä voidaan pitää graafisena käyttöliittymänä tietokannalle, jossa käyttäjä määrittää taulun kyselylle, jonka pohjalta muodostetaan tietokannan tiedoista vastaava taulu. Prototyyppi on kehitetty Neo4j-graafitietokannan hallintajärjestelmän päälle, painopiste on ominaisuusgraafeissa ja tekstimuotoisissa kyselyissä. Formalismin pohjalta voidaan toteuttaa erilaisia käyttöliittymiä, joissa kysely- ja tulosgraafit esitetään visuaalisesti. Prototyypin taulukkomuotoisesta kyselykielessä on myös potentiaalia hakuehtojen määrittelemiseen verkkosivutoteutuksena.

Tutkielma on konstruktiivinen ja keskeisiä havaintoja ovat toteuttamisen monimutkaisuus graafi-tietorakenteen vuoksi. Lisäksi helppokäyttöisyys osaltaan vähentää ilmaisuvoimaa, jolloin kyselyt ovat rajoittuneempia kuin muut kyselykielet. Graafitietokannan QBE-kyselyjen ei tarvitse myöskään merkittävästi erota alkuperäisestä QBE-kyselykielestä, erot korostuvat lähinnä kaarien ilmaisemisessa.

Avainsanat: graafi, graafitietokanta, kyselykieli, query-by-example, QBE

Sisällys

1 Johdanto

Tässä tutkielmassa käsitellään *Query-by-Example (QBE)* -kyselykieltä graafitietokannan kyselykielenä. QBE-kyselykieli on alkujaan kehitetty käytettäväksi relaatiotietokannoissa [Zloof, 1975a]. Kyselykielessä käyttäjä määrittää rakenteen taululle ja täyttää siihen esimerkkiarvot, jonka jälkeen ohjelmisto tulkitsee kyselyn ja palauttaa tuloksen samalla rakenteella [Zloof, 1975a]. Tarkastelun kohteena on graafitietokannat, jossa graafin kyselyrakenteen perusteella palautetaan sitä vastaava graafi.

Teoreettinen tausta graafikyselykielille saa alkunsa Cruz ja muiden [1987] määrittelemästä visuaalisesta graafikyselykielestä, jolla haetaan tietoa graafista. Tässä lähestymistavassa kyselygraafi määritellään aligraafijoukon avulla, jotka täsmätään graafiin. Polut esitetään säännöllisillä lausekkeilla, joita myöhemmissä tutkimuksissa kutsutaan *säännöllisiksi polkukyselyiksi (regular path queries)*. Monet kyselykielet hyödyntävät formalismin pohjana säännöllisiä polkukyselyitä [Libkin *et al.*, 2013]. Kyselykielet näyttävät kuitenkin erilaisilta ja toimivat käyttäjän näkökulmasta hieman eri tavoin. Säännöllisiin polkukyselyihin perustuu muun muassa Libkin ja muiden [2013] ehdottama XPath. Svensk [2021] tutkii XPathin kääntämistä Cypherille ja kertoo kääntäjän toteuttamisesta.

Angles ja muut [2017] määrittävät teoreettisen pohjan graafikyselykielille ja vertailevat sitä vakiintuneisiin kieliin Cypher, Gremlin sekä SPARQL. Graafikyselykielillä ei ole kuitenkaan ollut olemassa SQL-kyselykieltä vastaavaa standardia, jonka vuoksi uudesta GQL-kyselykielestä ollaan muodostamassa uusi standardikieli [GQL Standard Proponents, 2022]. GQL-kyselykieli ottaa vaikutteita muun muassa Cypher, PGQL, G-Core ja SQL -kyselykielistä [GQL Standard Proponents, 2022]. Nämä kyselykielet hyödyntävät avainsanoja ja voivat sisältää monimutkaisia ilmaisuja, joita voi olla hankala oppia. Tämän vuoksi voi olla iso kynnys ottaa graafitietoja käyttöön ilman graafista käyttöliittymää, jonka lisäksi myös osaaminen graafiteoriasta saattaa olla tarpeen.

Zloof [1975b] pyrkii ratkaisemaan relaatiokantojen käytettävyysongelmaa QBE-kyselykielen avulla. QBE-kyselykielestä on olemassa laajennos *Office Procedures by Example (OBE)*, joka automatisoi tietokannan käytön muun muassa sähköpostien ja raporttien laatimisessa [Zloof, 1981]. Zloof [1975a] kuvailee kyselyjen lisäksi myös taulujen luonnit ja tietojen päivitysoperaatiot. Alkuperäinen QBE-kyselykieli soveltuu kuitenkin vain relaatiotietokannoille, joten Jabocs ja Walczak [1983] määrittävät *Generalized Query-by-Example (GQBE)* -kyselykielen, joka tukee myös hierarkkisia tietokantoja. Taulut voivat GQBE-kielessä olla *sisäkkäisiä (nested)*, jolloin niillä voi-

daan esittää puu-rakenteita [Jabocs ja Walczak, 1983]. GQBE-kieli tukee myös *verkkomalliin (network model)* pohjautuvia tietokantoja [Jabocs ja Walczak, 1983]. Verkkomalli muistuttaa joiltakin osin graafeja, mutta eivät kuitenkaan vastaa moderneja graafitietokantoja. Lorentzos ja Dondis [2006] käsittelevät *QBEN*-kyselykielessä sisäkkäisiä tauluja, sillä SQL-kielen laajennokset sisäkkäisille taulukoille on koettu hankalaksi käyttää.

Braga ja muut [2005] soveltavat QBE-kielen periaatteita *XQuery by Example (XQBE)* -kyselykielen määrittelyssä. XQBE on visuaalinen QBE-kyselykieli XML-puulle [Braga *et al.*, 2005]. Shelly ja muut [2012] määrittävät QBE-kielelle sovelluksen terveydenhuoltodatan hakemiseen graafisella käyttöliittymällä. Shellyn ja muiden [2012] mukaan XQBE vaatii osaamista puu-tietorakenteista verrattuna tavalliseen QBE-tauluun. Ratkaisuksi Shelly ja muut [2012] ehdottavat *Archetype Query Language (AQL)* -kyselykielen päälle rakennettua *AQBE*-kyselykieltä, joka on taulukkomuotoinen.

Graafitietokantojen kanssa ongelmat korostuvat, sillä taustalla ovat graafit, jolloin mahdollinen graafiteorian osaamisen vaatimus voi muodostua esteeksi. Taulukot voidaan kokea helpommaksi käyttää, sillä taulukkolaskentaohjelmat ovat yleisessä käytössä. QBE-kyselykieli taulukkomuotoisena voisi olla ratkaisu tähän ongelmaan ja sen perusteella voidaan myös kehittää käyttöliittymiä graafitietokannan käyttämiseen. Thomas ja Gould [1975] tutkivat QBE-kyselykielen käyttöä relaatiotietokannoissa, opettelun nopeus havaittiin nopeammaksi ja virheitä tapahtui vähemmän kuin muissa vertailun kyselykielissä. Thomas ja Gould [1975] arvelevat, että QBE-kyselykielen merkittävimpiä etuja ovat valmiit kyselytaulupohjat sekä sanattomuus eli avainsanoja on vähän. SQL ja QBE -kyselykielten inhimillisten seikkojen vertailussa huomataan myös, että SQL-kyselykielen käyttö on tehokkaampaa jos opettelee ensin QBE-kyselykielen [Yen ja Scamell, 1993]. Lisäksi Yen ja Scamell [1993] havaitsivat, että SQL-kyselyjen muotoilussa esiintyi enemmän virheitä kuin QBE-kyselyissä, sillä taulukkomainen rakenne rajaa mahdollisuuksia tehdä virheitä.

GraphQL -rajapintojen yleistyessä, sitä on sovellettu myös suoraan relaatiotietokantoihin esim. *Graphile*[1]. *Dgraph*[2] on eräs GraphQL-tietokanta. GraphQL muistuttaa QBE-periaatetta JSON-rakenteilla, mutta tulokset rajataan *suodatustoiminnoilla (filter)* [GraphQL, 2022]. Esimerkkiarvoja ei myöskään sovelleta suodattamisessa. Graafeihin liittyviä QBE-periaatteita seuraavia tutkimuksia vaikuttaisi olevan niukasti ja lähestymiskulma on usein visuaalinen. Esimerkiksi Jin ja muut [2010] käsittelevät visuaalista graafikyselyä ja siihen liittyviä hakumenetelmiä graafitieto-

[1] `https://www.graphile.org/`
[2] `https://dgraph.io/graphdb/`

kannasta. Jayaram ja muut [2014] soveltavat QBE-kyselykieltä tietograafeihin eli taustalla on idea muotoilla kyselyt monikkoina ilman, että käyttäjälle näkyy graafeja. Jayaram ja muiden [2014] esimerkissä käyttäjä syöttää avainsanoja pilkulla erotettuna ja tulokset näytetään taulukkona.

Tämä pro gradu -tutkielma on konstruktiivinen ja sisältää prototyypin toteutuksen kyselykielen QBE-taulukkomuotoisena graafitietokannalle. Lähdekoodi on saatavilla verkossa avoimen lähdekoodin projektina: `https://github.com/cschlay/qbe-gradu`. Taulukkomuotoinen kysely perustuu relaatiotietokannoissa käytettyyn QBE-kyselykieleen [Zloof, 1975b], jota muokataan graafitietokantaan sopivaksi. Uudempi toteutus tästä kyselykielestä (IBM QBE[3]) on myös vaikuttanut merkintöihin. Lisäksi tietojen päivittämistä käsitellään lyhyesti, sillä ne voidaan esittää samankaltaisilla rakenteilla, joita Zloof [1975a] esittää. Tässä tutkielmassa selvitetään millainen QBE-kyselykieli sopisi kyselyjen suorittamiseksi graafitietokantaan, ja kuinka sellainen voidaan toteuttaa. Prototyypin toteuttamiseen käytetään Java 11 -ohjelmointikieltä, sillä prototyypin toteuttamisen aikaan Neo4j versio 4.4 ei tukenut uudempia Java versioita. Ohjelmointityyli noudattaa *olio-ohjelmoinnin (object-oriented programming)* periaatteita, jotka oletetaan olevan lukijalle tuttuja.

Tutkielmassa edetään abstraktiotasoittain luvusta 3, joka käsittelee graafiteoriaa. Tämän jälkeen luku 4 käsittelee tarvittavia tietorakenteita ja luvussa 5 lähestytään graafitietokantaa. Luku 6 formalisoi kyselygraafin ja tulosgraafin. Luvussa 7 esitetään varsinainen taulukkokyselyn syntaksi ja luvussa 8 käsitellään taulukkomuotoisen kyselyn jäsennystä ja tuloksen tulostamista. Luvussa 9 käsitellään lisäominaisuuksia, joita ei sisällytetä prototyyppiin. Lopuksi luvussa 10 vertaillaan lyhyesti muita QBE-kyselykieliä määriteltyyn graafikyselykieleen.

2 Merkinnät

Tässä luvussa esitellään tässä tutkielmassa käytettävät merkinnät sekä määritellään säännölliset ja loogiset lausekkeet. Matematiikan joukko-opin ja logiikan perusmerkinnät oletetaan tutuksi. Symbolilla $\mathbb{B} = \{true, false\}$ merkitään totuusarvojen joukkoa. Merkintä _ monikon alkion tilalla tarkoittaa sitä, että alkion arvosta ei olla kiinnostuneita ja se voi olla mitä tahansa.

[3] `https://www.ibm.com/docs/en/qmf/11.1?topic=cics-query-by-example`

2.1 Merkkijonot

Merkkijonot määritellään formaalin kieliteorian mukaisesti joukosta aakkosia Σ [Meduna, 2014, s. 13], joita ovat kaikki UTF-8 määritellyt merkit. Kaikkien merkkijonojen joukkoa merkitään Σ^* [Meduna, 2014, s. 13]. Jokainen aakkonen yksinään on merkkijono eli jos $s \in \Sigma$ niin $s \in \Sigma^*$. Tyhjä merkkijono ϵ on myös merkkijono. Kahden merkkijonon s_1 ja s_2 *konkatenaatio (concatenation)* muodostaa uuden merkkijonon $s_1 s_2 \in \Sigma^*$. [Meduna, 2014, s. 13–14.] Merkkijonojen joukkoa merkitään jatkossa symbolilla $\mathbb{S}$. Merkkijonot merkitään pseudokoodissa lainausmerkkien väliin "`teksti`". Merkintää käytetään myös matemaattisessa tekstissä erottamaan ne muista arvoista. Pseudokoodit voivat sisältää lainausmerkkejä `"` merkkijonoissa kuten "`"teksti"`", jolloin ne ovat osa merkkijonoa.

2.2 Säännölliset lausekkeet

Säännölliset lausekkeet ovat lyhyitä merkkijonoja, jotka ilmaisevat kieleen hyväksyttäviä sanoja konkatenaatio-, yhdiste- ja sulkeumaoperaatioiden avulla [Meduna, 2014, s. 45]. Olkoon R säännöllisten lausekkeiden joukko. Säännöllisten lausekkeiden määritelmä perustuu aakkostoon Σ, joka on joukko merkkejä kuten suomen kielen aakkoset. Kaikki aakkoset $s \in \Sigma$ ovat säännöllisiä lausekkeita. Konkatenaatio on kahden säännöllisen lausekkeen muodostama säännöllinen lauseke $s = (s_1 s_2) \in R$, missä $s_1, s_2 \in R$. [Meduna, 2014, s. 45.] Perusoperaatioihin kuuluu myös sulkeuma eli toisto-operaatio [Meduna, 2014, s. 45], joka toistaa säännöllistä lauseketta n kertaa, merkitään s^n. Erityistapauksena on ääretön toisto, jota merkitään s^*. [Meduna, 2014, s. 14.]

2.3 Loogiset lausekkeet

Loogiset lausekkeet ilmaisevat totuusarvoja, käyttö on sama kuin ohjelmointikielissä yleensä. Kyselyissä näitä tarvitaan esimerkiksi numeeristen arvovälien ilmaisemiseen. Esimerkiksi arvoväli $x \in]2, 8[$ ilmaistaan muodossa $x > 2 \land x < 8$. Tarkempi merkkijonoesitys käsitellään myöhemmin loogisten lausekkeiden tulkinnan yhteydessä. Kyselyissä loogiset lausekkeet sisältävät muuttujan x kuten esimerkiksi funktio $f(x) = x > 2 \land x < 8$ esittää loogista lauseketta.

Vertailuoperaattoreiden avulla määritellään *vertailulausekkeita*, joita käytetään loogisten lausekkeiden määrittelyn lähtökohtana. Nämä ovat aina binäärioperaatioita eli muotoa $x \circ b$, missä $\circ \in \{=, <, \leq, >, \geq\}$, $b \in \mathbb{R}$ ja x on muuttuja. Loogiset lausekkeet määritellään symbolien $\land$, $\lor$ ja $\neg$ avulla niin, että jos c ja d ovat vertailulausekkeita tai loogisia lausekkeita, niin $(c \land d)$, $(c \lor d)$ ja $(\neg c)$ ovat loogisia

lauseita (vrt. [Ebbinghaus *et al.*, 1994, s. 15–16]). Tällöin muodostetuista lausekkeista muodostetaan rekursiivisesti uusia lausekkeita. Määritelmä on suppeampi kuin Ebbinghaus ja muiden [1994] S-kaava määritelmä, sillä esimerkiksi implikaatio $\rightarrow$ ja ekvivalenssi $\leftrightarrow$ jätetään pois käytöstä, jonka lisäksi atomisten S-termien tilalla käytetään vertailulausekkeita ja reaalilukuja. Aritmeettisia laskuoperaattoreita ei käsitellä tässä tutkielmassa.

3 Graafiteoria

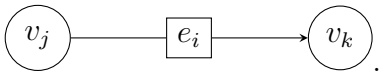
Verkkomaisesta tiedosta eli kaikesta, mistä on olemassa yhteyksiä objektien välillä, voidaan mallintaa graafiksi, jolloin graafiteoriassa tutkittuja ominaisuuksia voidaan soveltaa. Graafin prosessointi on klassinen ongelma ja algoritmeja on kehitetty niin kauan kuin graafiteoriaa on tutkittu. Tutkielman kannalta keskeisiä graafiteorian tutkimusaiheita ovat *graafin läpikäynti (graph traversal)*, *graafin täsmäys (graph matching)* sekä *graafin merkitseminen (graph labeling)*. Myös *graafin isomorfismi (graph isomorphism)* ja *homeomorfismi (homeomorphism)* esiintyvät graafikyselyjä käsiteltävissä teksteissä.

3.1 Graafi

Tavallisessa graafissa $G = (V, E)$ on määritelty joukko *solmuja (vertices)* V ja joukko *särmiä (edges)* E. Solmuilla tarkoitetaan objekteja, ja särmillä yhteyttä näiden välillä. Särmä voidaan merkitä joko joukkona $e_i = \{v_j, v_k\}$ tai parina $e_i = (v_j, v_k)$, missä $v_j, v_k \in V$. Jos särmä määritellään parina, niin särmällä on suunta ja sitä kutsutaan *kaareksi (arc)*. Kun graafin kaikki särmät ovat kaaria, graafi on *suunnattu graafi (directed graph)*. [Foulds, 1992, s. 9–11.]

Kaari määritellään graafikyselyjä varten hieman eri tavalla, sillä graafikyselyissä kahden solmun välillä voi olla useampi kaari [Cruz *et al.*, 1987]. Tällaiset graafit ovat *multigraafeja (multigraph)* [Foulds, 1992, s. 11]. Tässä tutkielmassa määritellään kaarelle jokin erottava tunniste kaarien erottamiseksi kuten id-tunniste $id \in \mathbb{N}$ eli kaari on $e_{id} = (id, v_j, v_k)$, missä $v_j, v_k \in V$. Eri kaarilla ei voi olla samaa tunnistetta.

Esimerkki 1. Suunnattu graafi visualisoidaan niin, että solmut piirretään ympyröinä ja kaaret viivoina. Kaari $e_i = (i, v_j, v_k)$ on neliö kahden solmun välissä



.

Vakiintuneessa merkinnässä ei käytetä neliöitä, vaan kaari on pelkkä nuoli. Fouldsin [1992] mukaan graafimerkinnät nuolilla ja palloilla saivat alkunsa kemiallisten yh-

disteiden piirtämisessä. Tässä tutkielmassa neliöillä selkeytetään tietojen liittäminen kaareen.

3.2 Ominaisuusgraafi

Ominaisuusgraafi (property graph) määritellään Anglesia ja muita [2017] mukaillen suunnatun graafin laajennoksena

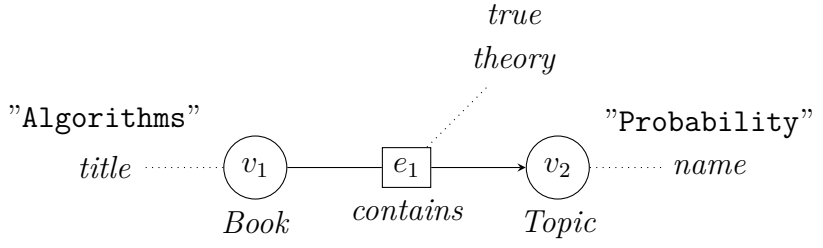
$$G = (V, E, \lambda, \sigma),$$

missä *kuvaus (function)* λ asettaa *nimikkeen (label)* nimien joukosta L kaarille ja solmuille eli $\lambda : V \cup E \to L$. Kuvaus σ antaa kaaren tai solmun ominaisuudelle arvon $\sigma(v, p) = x$ tai $\sigma(e, p) = x$, missä $x \in P_{value}$ on ominaisuuden arvo ja $p \in P_{name}$ sen nimi [Angles *et al.*, 2017]. Joukko P_{name} sisältää ominaisuuksien nimiä, ja P_{value} on joukko totuusarvoja, merkkijonoja ja reaalilukuja. Nimikejoukko L sisältää merkkijonoja, joita käytetään *särmämerkittyjen graafien (edge-labelled graph)* määrittelemiseen, johon ominaisuusgraafi perustuu [Angles *et al.*, 2017]. Cruz ja muut [1987] käyttävät myös kuvauksia nimikkeiden ja arvojen liittämisessä solmuille ja kaarille. Tässä tutkielmassa nimikkeet ja ominaisuudet sulautetaan myöhemmin graafitietokantojen yhteydessä solmu- ja kaarijoukkojen alkioihin.

Esimerkki 2. Ominaisuusgraafi visualisoidaan laajentamalla esimerkin 1 graafin esitystä niin, että ominaisuudet esitetään tekstinä ja nimet merkitään kaaren tai solmun alapuolelle. Havainnollistetaan tätä graafilla, joka ilmaisee: *kirja Algorithms sisältää teoriaa aiheesta todennäköisyyslaskenta.* Olkoon $G = (V, E, \lambda, \sigma)$ ominaisuusgraafi, missä $V = \{v_1, v_2\}$ ja $E = \{e_1\} = \{(1, v_1, v_2)\}$. Lisäksi olkoon ominaisuuksien arvojoukko

$$P_{value} = \{\text{"}\texttt{Algorithms}\text{"}, \text{"}\texttt{Probability}\text{"}, true\},$$

nimikejoukko $L = \{Book, Topic, contains\}$ ja $P_{name} = \{name, title, theory\}$. Tällöin graafin G visuaalinen esitys on



Kuvaus λ antaa siis solmuille nimet $\lambda(v_1) = Book$ ja $\lambda(v_2) = Topic$ kuten myös kaarelle $\lambda(e_1) = contains$. Ominaisuuden arvo määräytyy kuvauksen σ avulla $\sigma(title, v_1) =$ "`Algorithms`", $\sigma(theory, e_1) = true$ ja $\sigma(name, v_2) =$ "`Probability`". Huomionarvosta on, että ominaisuusgraafin visualisointi tällä tavalla mahdollistaa sen käsittelyn myös tavallisena graafina muuttamalla kaaret ja ominaisuudet solmuiksi.

3.3 Polku

Polkuja (path) käytetään graafikyselyissä määrittämään millaisia kaaria kahden solmun välillä voi olla [Angles *et al.*, 2017]. Määritellään polku T jonona

$$T = (v_1, e_1, v_2, \ldots, e_n, v_n),$$

missä solmuja v ja kaaria e luetellaan vuorotellen [Foulds, 1992, s. 17]. Foulds [1992] määrittelee polut niin, että alkiot eivät voi toistua. Määritelmä on hieman liian tiukka, sillä *luuppeja (loop)* ja *silmukoita (cycle)* voi esiintyä graafissa, kun niitä käytetään hakuehtojen rajaamiseen. Luupit ovat kaaria, joissa lähtö- ja maalisolmu on samat, merkitään $(v, v) \in E$, missä v on jokin solmu [Foulds, 1992, s. 11]. Silmukat ovat polkujen erityistapaus $(v, \ldots, v)$, jossa lähtö- ja maalisolmut ovat samoja [Foulds, 1992, s. 18]. Polun määritelmää löysennetään niin, että päätesolmut v_1 ja v_n voivat olla samoja, mutta näiden välillä esiintyvät alkiot eivät. Kun viimeinen solmu voi olla sama kuin ensimmäinen solmu, voidaan esittää esimerkiksi suhde "henkilö antaa itselleen lahjan".

Poluilla on graafikyselyihin liittyen erillinen tutkimusaihe säännölliset polkukyselyt, joissa koko polku merkitään tulkattavana merkkijonona [Angles *et al.*, 2017]. Tässä tutkielmassa ei kuitenkaan hyödynnetä näitä, vaikka kuuluvatkin graafikyselyjen perusprimitiiveihin. Määritellään *polkusärmäksi* sellaiset kaaret, jotka esittävät polkua samalla merkinnällä kuin kaaret. Nämä kaaret esittävät käytännössä transitiivisia relaatioita esimerkiksi polkusärmä $(\_, v_1, v_3)$ ilmaisee polun $(v_1, e_1, v_2, e_2, v_3)$, joka voi koostua kaarista $e_1 = (1, v_1, v_2)$ ja $e_2 = (2, v_2, v_3)$.

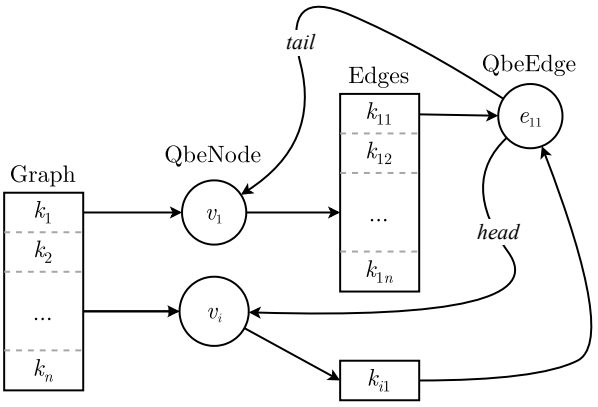
Polkuja hyödynnetään tässä tutkielmassa graafin läpikäymisessä, jotta voidaan tarkistaa solmut ja kaaret, jotka on käyty läpi. Tällä tavalla vältetään tilanne, jossa samoja solmuja tai kaaria käytetään uudelleen. Tämä toistuvuusrajoite kuuluu yleisesti polkukyselyn semantiikkaan [Angles *et al.*, 2017].

4 Graafeihin liittyvät tietorakenteet

Tässä tutkielmassa käsitellään graafeja ohjelmallisesti, joten *vieruslistaa (adjacency list)* muistuttavaa rakennetta käytetään graafin esittämiseksi. Cormen ja muut [2009, s. 590] käyttävät vieruslistan esittämiseksi linkitettyä listaa, jonka alkiot ovat lista solmuja. Tällä tavalla ilmaistaan, että listan muilla solmuilla on olemassa särmä ensimmäisen alkion kanssa. Ongelma tässä tavassa on se, että solmujen ja särmien etsiminen vaatii aina graafin läpikäymisen. Lisäksi esitystavan tarkoitus on esittää särmät solmujen välillä [Cormen *et al.*, 2009, s. 590], joten niihin ei voi tallentaa tietoa kuten ominaisuusgraafi vaatii. Sama ongelma on myös matriisiesityksessä. Tästä syystä käytetään *hajautustaulua (hash table)*, jotta solmut voidaan hakea suoraan jollakin tunnisteella. Hajautustaulu on käytännössä *taulukko (array)*, jossa alkioiden paikka lasketaan merkkijonon perusteella [Cormen *et al.*, 2009, s. 254].

4.1 Graafi hajautustauluna

Hajautustaulu-tietorakenteille on yleensä olemassa valmis toteutus eri kielten standardikirjastoissa esimerkiksi `HashMap<K, V>`[4]. Avaimina käytetään merkkijonoja ja arvoina solmuja eli `HashMap<String, QbeNode>`. Jokaisella solmulla on kokoelma kaaria, joista on olemassa *viitteet (reference)* lähtö- ja maalisolmuille. Kaaret säilötään myös hajautustaulussa `HashMap<String, QbeEdge>`. Kuva 1 havainnollistaa tietorakennetta solmulla v_1, jolla on tunnus $k_1 \in K$ ja kaarella e_{11} on tunnus $k_{11} \in K$. Avaintunnusten joukko $K \subset \mathbb{S}$ sisältää siis merkkijonoja. Avainten indeksointi kuvassa 1 perustuu solmuihin, joilla on tunnus k_i ja siihen liittyvillä kaarilla k_{ij}.



Kuva 1: Graafin esitys hajautustauluna.

[4] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html`

Määritelmä 1 (Solmujen lisäys ja poisto)**.** Olkoon avaintunnusten joukko $K \subset \mathbb{S}$. Määritellään graafille $G = (V, E)$ keskeisimmät funktiot hajautustaulussa

- $\textsc{Node}(G,\ k) \rightarrow v_k$ palauttaa graafista G solmun $v_k \in V$, jolla on tunnus $k \in K$,
- $\textsc{AddNode}(G,\ k,\ v_k) \rightarrow G'$ lisää solmun v_k graafiin G tunnisteella $k \in K$,
- $\textsc{RemoveNode}(G,\ k) \rightarrow G'$ poistaa graafista G solmun, jolla on tunniste $k \in K$ eli $v_k \in V$, kaikki siihen liittyvät särmät poistetaan myös.

Koska graafi määritellään solmu- ja särmäjoukkoina, niin lisääminen ja poistaminen tapahtuu joukko-operaatioiden avulla. Esimerkiksi määritelmän 1 solmun poistaminen $G - v_k$ määritellään solmujoukon erotuksen $V \setminus \{v\}$ avulla niin, että myös kaikki solmuun v liittyvät särmät poistetaan. Vastaavasti solmun lisääminen merkitään $G + v_k$. [Diestel, 2016, s. 4.]

Määritelmä 2 (Särmien lisäys ja poisto)**.** Olkoon avaintunnusten joukko $K \subset \mathbb{S}$. Käytetyn hajautustaulu-tietorakenteen vuoksi särmien käsittely on solmuilla, mutta apufunktiot särmäoperaatioille määritellään graafille $G = (V, E)$

- $\textsc{Edge}(G, k) \rightarrow e_k$ palauttaa särmän $e_k \in E$, jolla tunniste $k \in K$ graafista G,
- $\textsc{AddEdge}(G, k, e_k, v_1, v_2) \rightarrow G'$ lisää graafiin G särmän e_k solmujen v_1 ja v_2 välille tunnuksella $k \in K$, myös päätesolmut lisätään, jos niitä ei löydy graafista G,
- $\textsc{RemoveEdge}(G, k) \rightarrow G'$ poistaa särmän $e_k \in E$ graafista G, jolla on tunnus $k \in K$.

Määritelmässä 2 särmän lisäys ei edellytä, että solmut olisivat valmiiksi graafissa. Solmu- ja särmäoperaatioiden lisäksi tarvitaan graafitason operaatioita. Keskeisin on graafin *unioni (union)*, jossa kaksi graafia yhdistetään yhdeksi graafiksi. Koska graafi määritellään joukkojen avulla, graafien $G_1 = (V_1, E_1)$ ja $G_2 = (V_2, E_2)$ unioni on $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ [Diestel, 2016, s. 3].

4.2 Ominaisuusgraafi tietorakenteena

Myös ominaisuusgraafi esitetään hajautustauluna, mutta graafioperaatioiden lisäksi tarvitaan operaatioita solmuille ja kaarille. Koska ominaisuusgraafia käsitellään graafin laajennoksena, niin myös kohdassa 4.1 määritellyt operaatiot ovat voimassa. Ero ominaisuusgraafin määritelmään on, että nimikkeet käsitellään solmujen ja kaarien niminä. Solmulla ja kaarella voi olla vain yksi nimi.

Määritelmä 3 (Solmu ominaisuusgraafissa)**.** Olkoon P_{name} joukko ominaisuuksien nimiä ja P_{value} mahdollisia arvoja ominaisuuksille. Määritellään kaikille ominaisuusgraafin $G = (V, E, \lambda, \sigma)$ solmuille $v \in V$ perusoperaatiot

- $\textsc{Name}(v) \to l$ palauttaa solmuun $v \in V$ liitetyn nimikkeen $l = \lambda(v)$,
- $\textsc{Property}(v, p) \to x$ palauttaa solmun $v \in V$ ominaisuuden $p \in P_{name}$ arvon $x \in P_{value}$ eli $\sigma(v, p) = x$,
- $\textsc{SetProperty}(v, p, x) \to v'$ liittää solmulle $v \in V$ ominaisuuden $p \in P_{name}$ arvolla $x \in P_{value}$,
- $\textsc{RemoveProperty}(v, p) \to v'$ poistaa solmulta $v \in V$ ominaisuuden $p \in P_{name}$.

Määritelmä 4 (Kaari ominaisuusgraafissa)**.** Kaari $e_i = (i, v_j, v_k)$ ominaisuusgraafissa sisältää samat funktiot kuin solmukin, jonka lisäksi

- $\textsc{tail}(e_i) \to v_i$ palauttaa kaaren e_i lähtösolmun v_j,
- $\textsc{head}(e_i) \to v_j$ palauttaa kaaren e_i maalisolmun v_k.

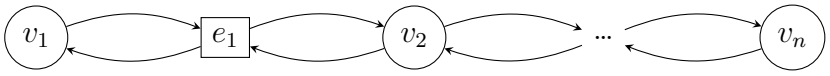
Määritelmässä 3 nimi NAME on funktio, mutta myöhemmin pseudokoodissa sitä käytetään muodossa $v.name$. Voidaan siis merkitä näin

$$\textsc{Name}(v) = v.\textsc{Name}() = v.name.$$

Vastaavasti merkitään määritelmän 4 päätesolmut $e.tail$ ja $e.head$.

4.3 Polku tietorakenteena

Polku voidaan esittää *kaksisuuntaisesti linkittyvänä listana (doubly linked list)*. Linkitetyssä listassa alkioilla on viitteet seuraavaan alkioon [Cormen *et al.*, 2009, s. 236], kuva 2 havainnollistaa polun rakennetta. Graafin $G = (V, E)$ poluille määritellään



Kuva 2: Polkutietorakenteen visualisointi.

operaatiot

- $\textsc{Add}(T, e) \to T'$ lisää kaaren $e_i = (i, v_j, v_k) \in E$ ja maalisolmun $v_j \in V$ polun T perään. Jos polussa ei ole alkioita, lisätään myös lähtösolmu $v_k \in V$,

- VISITED$(T, y) \to \mathbb{B}$ palauttaa totuusarvon siitä, onko solmu tai kaari $y \in V \cup E$ polun T alkio.

Toteutuksessa voidaan käyttää myös muita listarakenteita, jotka mahdollistavat listan kapasiteetin kasvattamisen ja pienentämisen.

5 Graafitietokanta

Graafitietokannat hyödyntävät graafeja tietojen tallentamisessa, jolloin monet graafiteoriasta tutut käsitteet ja algoritmit ovat käytettävissä. Tietomalleja on kuitenkin monia ja toteutukset tukevat eri tavoin eri graafeja. Angles [2012] vertailee eri graafitietokantoja ja esille nousevat erot toteutuksessa, tuesta eri graafirakenteille ja graafialgoritmeille. Angles [2012] sisällyttää vertailuun myös *hypergraafitietokannat (hypergraph)* ja *sisäkkäiset graafit (nested graph)*. Tässä luvussa esitellään tässä tutkielmassa käytettävää tietomallia, joka pohjautuu ominaisuusgraafiin ja sitä hyödyntävään *graafitietokannan hallintajärjestelmään (graph database management system)* Neo4j [2022].

Tarkasteltavat QBE-kyselykielen periaatteet soveltuvat Neo4j:n lisäksi muillekin ominaisuusgraafitietokannoille kuten *Nebula Graph*[5] ja *JanusGraph*[6]. Neo4j käsittelee solmuja (vertex) nimellä *solmu (node)* ja kaaria *suhteina (relationship)*. Tietokannoilla on tyypillisesti myös käytössä jokin kyselykieli kuten Cypher tai Nebula Query Language (nQL), mutta tässä tutkielmassa ei tarkastella näitä. Neo4j:n ominaisuusgraafi-tietomallia hyödynnetään tutkielmassa suppeammin kuin mahdollista, jotta periaatteet olisivat mahdollisimman yleiskäyttöisiä. Neo4j valitaan tutkielmaan sen tarjoaman ohjelmointirajapinnan vuoksi, joka mahdollistaa solmujen ja kaarien hakemisen suoraan ilman erillistä kyselykieltä.

Graafitietokanta määritellään ominaisuusgraafin variaationa $G = (V, E)$, missä solmu on muotoa $(id_v, l_v, P_v) \in V$ ja kaari $(id_e, l_e, id_{v_1}, id_{v_2}, P_e) \in E$. Arvo $id \in \mathbb{N}$ on id-tunniste, l_v solmun nimi, l_e kaaren nimi, P_v on solmun ominaisuusjoukko ja P_e kaaren ominaisuusjoukko. Kaaren päätesolmut ilmaistaan solmujen id-tunnisteilla. Aiemmin määritetyllä ominaisuusgraafin kuvauksella σ ilmaistuna solmun $v \in V$ ominaisuusjoukko on

$$P_v = \{(p, \sigma(p, v)) \mid p \in P_{name}\},$$

vastaavasti kaaren $e \in E$ ominaisuusjoukko olisi

$$P_e = \{(p, \sigma(p, e)) \mid p \in P_{name}\}.$$

[5] `https://nebula-graph.io/`
[6] `https://janusgraph.org/`

5.1 Tietomallin rajoitteet

Neo4j:n tukemista *tietotyypeistä (data type)* (ks. [Neo4j, 2022, s. 15]) hyödynnetään kokonaisluvut $\mathbb{Z}$, liukuluvut $\mathbb{F}$, merkkijonot $\mathbb{S}$ ja totuusarvot $\mathbb{B}$. Liukulukuja käsitellään kuin ne olisivat reaalilukujen joukko $\mathbb{R}$. Ominaisuusgraafin määritelmän ominaisuuksien arvojen joukko P_{value} ja nimikejoukko L määritellään rajoittumaan näihin tietotyyppeihin eli $L \subset \mathbb{S}$ ja ominaisuuden arvojoukko $P_{value} \subset \mathbb{R} \cup \mathbb{S} \cup \mathbb{B}$.

Neo4j-graafitietokannan hallintajärjestelmässä yhdellä solmulla voi olla useampi nimi [Neo4j, 2022, s. 140]. Tässä tutkielmassa käsitellään tietomallia niin, että solmulla voi olla vain yksi nimi. Hakua useammalla nimellä ei siis oteta huomioon.

Solmujen nimeämiskäytäntönä on `PascalCase` ja kaarille `camelCase`. Tämä poikkeaa suositellusta käytännöstä, jossa kaaret nimetään isoilla kirjaimilla [Neo4j, 2022, s. 18]. Nimellä on merkitys taulukon otsikoiden jäsentämisessä, jossa solmut tunnistetaan kaarista alkukirjaimen perusteella.

5.2 Esimerkkigraafi

Merkintöjen ja kyselykielen syntaksin havainnollistamiseksi käytetään esimerkkejä, jotka suorittavat kyselyjä kuvan 3 graafitietokantaan. Esimerkkigraafi on yksinkertainen graafi, joka mallintaa kurssien suhteita kirjoihin ja niiden aihepiireihin. Tällainen graafi pyrkii esittämään mahdollisimman kattavasti eri tilanteita, joita graafin prosessoinnissa voi tulla vastaan. Kuvan 3 graafi käyttää solmujen nimiä

$$L_V = \{Course, Book, Student, Topic\},$$

ja kaarien nimiä

$$L_E = \{recommends, uses, written\ for, contains\}.$$

Ominaisuuksien nimien joukko on

$$P_{name} = \{name, primary, difficulty, level, theory, \\ period, year, price, supplemental, title\}$$

ja arvojen joukko

$$P_{value} = \{\text{"}\texttt{Introduction to Graph Theory}\text{"}, \text{"}\texttt{Graph Theory}\text{"}, \\ \text{"}\texttt{Mathematics for Computer Science}\text{"}, \\ \text{"}\texttt{Randomized Algorithms}\text{"}, \text{"}\texttt{Probability}\text{"}, \text{"}\texttt{easy}\text{"}, \\ \text{"}\texttt{Winter II}\text{"}, true, false, 60.00, 5.00, 2010, 1989, 2, 4\}.$$

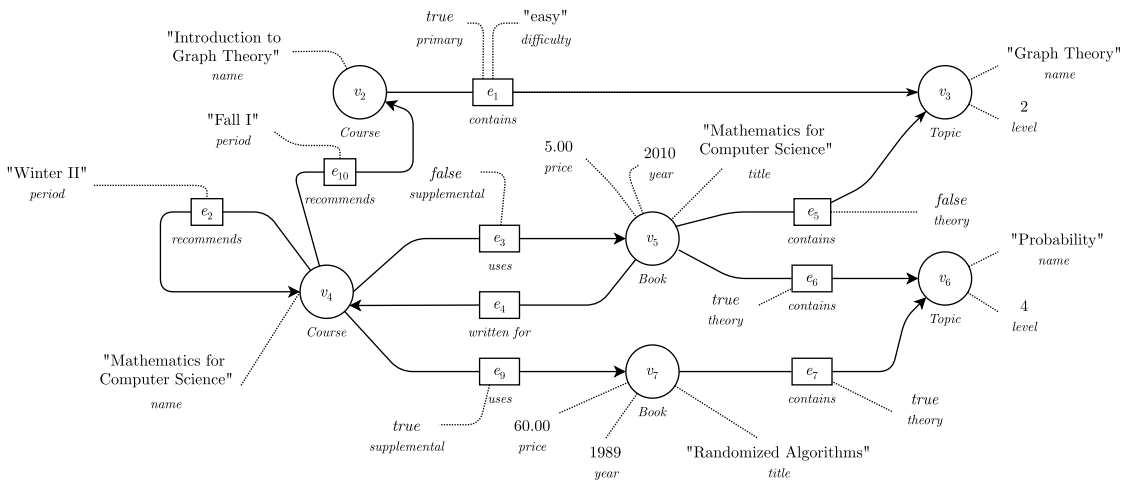
Näiden joukkojen pohjalta muodostetaan solmujen joukko

$$
\begin{aligned}
V = \{&(2, Course, \{(name, \text{"}\texttt{Introduction to Graph Theory}\text{"})\}),\\
&(3, Topic, \{(name, \text{"}\texttt{Graph Theory}\text{"}), (level, 2)\})\\
&(4, Course, \{(name, \text{"}\texttt{Mathematics for Computer Science}\text{"})\})\\
&(5, Book, \{(price, 5.00), (year, 2010)\})\\
&(6, Topic, \{(name, \text{"}\texttt{Probability}\text{"}), (level, 4)\})\\
&(7, Book, \{(price, 60.00), (year, 1989), (title, \text{"}\texttt{Randomized Algorithms}\text{"})\})\}
\end{aligned}
$$

ja kaarien joukko

$$
\begin{aligned}
E = \{&(1, contains, 2, 3, \{(primary, true), (difficulty, \text{"}\texttt{easy}\text{"})\}),\\
&(2, recommends, 4, 4, \{(period, \text{"}\texttt{Winter II}\text{"})\}),\\
&(3, uses, 4, 5, \{(supplemental, false)\}),\\
&(4, written\ for, 5, 4, \emptyset),\\
&(5, contains, 5, 3, \{(theory, false)\}),\\
&(6, contains, 5, 6, \{(theory, true)\}),\\
&(7, contains, 7, 6, \{(theory, true)\}),\\
&(9, uses, 4, 7, \{(supplemental, true)\}),\\
&(10, recommends, 4, 2, \{(period, \text{"}\texttt{Fall I}\text{"})\})\}.
\end{aligned}
$$

Kaarien joukossa solmut merkitään lyhyesti muodossa v_{id}, jossa id vastaa solmujen ensimmäistä alkiota. Kuvassa 3 solmujen ja kaarien id näkyy muodossa v_{id} ja e_{id}.



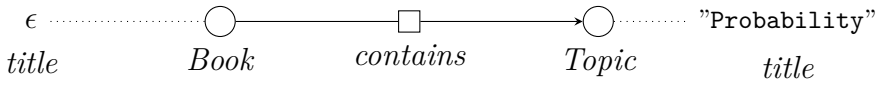
Kuva 3: Esimerkkigraafi visualisoituna.

Prototyyppi sisältää ohjeet kuvan 3 esimerkkigraafin käyttöön. Myöhemmin esiteltävät taulukkomuotoiset kyselyt esitetään tähän esimerkkigraafiin perustuen.

6 Kysely- ja tulosgraafit

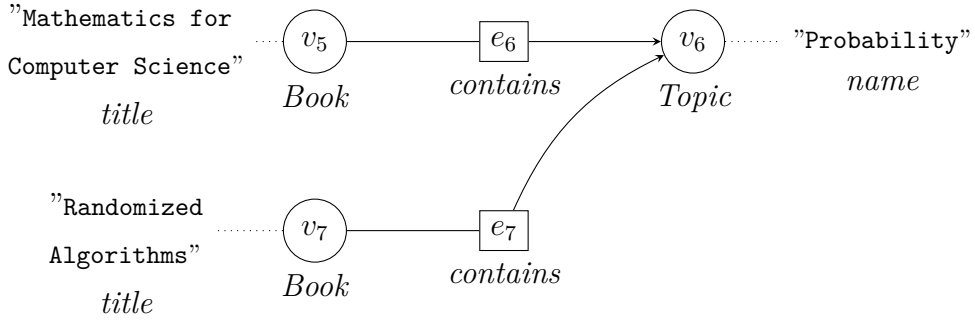
Kyselykielen toteutuksen ytimenä ovat graafit, joita käydään läpi. Kyselyn lopputulosta kutsutaan *tulosgraafiksi (result graph)* ja sen rakennetta kuvaavaa graafia *kyselygraafiksi (query graph)*. Kyselygraafi on siis malli, jota sovelletaan graafitietokantaan, jolloin kaikki mallin rajoitteet täyttävät graafitietokannan solmut ja kaaret muodostavat tulosgraafin. Käsiteltäviä graafeja on siis kolme kappaletta: graafitietokanta G, kyselygraafi Q ja tulosgraafi R. Tavoitteena on, että kaikki QBE-muotoiset kyselyt, jotka voidaan kääntää kyselygraafiksi, pystytään suorittamaan.

Esimerkki 3. Kyselygraafi visualisoidaan merkitsemällä solmujen ja kaarien alapuolelle näiden nimet. Esimerkiksi kysely ”aihetta todennäköisyyslaskentaa käsittelevät kirjat” haetaan graafilla



.

Vastaavassa tulosgraafissa on kaksi kirjaa, sillä molemmilla kirjoilla on kaari aiheeseen `Probability`. Kyselygraafissa esiintyvät solmut ja kaaret on löydyttävä graafitietokannasta. Tyhjä merkkijono määrittää sen, että kirjan nimi sisällytetään tulosgraafiin.



.

Seuraavat kohdat tässä luvussa perustuvat tutkielman ohjaajan Marko Junkkarin luonnostelmaan graafikyselystä. Samankaltaista määritelmää käytetään myös AQTA-kielessä XML-puukyselyille [Hua ja Hein, 2021]. Tämä määritelmä poikkeaa Angles ja muiden [2017] ominaisuusgraafin määritelmästä niin, että ominaisuudet esitetään pareina ja nimet merkitään suoraan solmuun tai kaareen. Taulukko

1 on tiivistelmä merkinnöistä, joita tässä luvussa määritellään. Taulukossa 1 omi-

Piirre	**Graafi** G	**Kysely** Q	**Tulos** R
Ominaisuus P	(p, x)	(p, c, s)	(p, x, s)
Solmu V	(id_v, l_v, P_v)	(l_v, P_q)	(id_v, l_v, P_v)
Kaari E	$(id_e, l_e, id_{v_1}, id_{v_2}, P_e)$	$(l_e, l_{v_1}, l_{v_2}, P_q, t)$	$(id_e, l_e, id_{v_1}, id_{v_2}, P_e, t)$
Graafi	$G = (V, E)$	$Q = (V_Q, E_Q)$	$R = (V_R, E_R)$

Taulukko 1: Kooste kyselyn graafimerkinnöistä.

naisuudet ovat siis pareja (p, x), missä p on ominaisuuden nimi ja x sen arvo. Arvo $c : P_{value} \to \mathbb{B}$ on funktio, jolla hyväksytään ominaisuuden arvo x. Merkinnässä (p, c, s) alkio $s \in \mathbb{B}$ kertoo palautetaanko ominaisuus tulosgraafiin. Id-tunnisteet id_v ja id_e määritetään graafitietokannassa, l_v viittaa solmun nimeen ja l_e kaaren nimeen. Ominaisuusjoukot P_v ja P_e sisältävät siis pareja (p, x). Joukot P_q sisältävät kolmikkoja (p, m, s). Merkintä $t \in \mathbb{B}$ kertoo, onko kyseessä polkusärmä.

6.1 Kyselygraafi

Tässä kohdassa formalisoidaan ominaisuusgraafi $G = (V, E, \lambda, \sigma)$ kyselygraafiksi. Kyselygraafi on graafimalli $Q = (V_Q, E_Q)$, jonka solmuja kutsutaan *kyselysolmuksi* ja kaaria *kyselykaariksi*. Vastaavasti nimitykset tulosgraafissa ovat *tulossolmu* ja *tuloskaari*. Eroja kyselysolmuilla ja graafitietokannan solmuilla on, että kyselysolmuista puuttuu id-tunniste. Kaarilla on id-tunnisteen lisäksi eroa päätesolmuilla niin, että kyselykaarilla on solmujen tilalla vain nimet. Lisäksi kyselykaarilla on transitiivisuus alkio.

Määritellään kyselygraafin piirteet ominaisuusgraafiin perustuen

- *nimikejoukko* $L_V \subset \mathbb{S}$ kyselysolmuille ja kyselykaarille $L_E \subset \mathbb{S}$,
- ominaisuuksiin liittyy joukko nimiä $P_{name} \subset \mathbb{S}$ ja joukko C ominaisuuksien arvojen tarkistusfunktioita $c : P_{value} \to \mathbb{B}$, missä P_{value} on ominaisuuksien arvojen joukko. Ominaisuuksien arvoina voivat olla merkkijonot, reaaliluvut ja totuusarvot. Kaikkien ominaisuuksien joukko on siis

  $$P_Q = \{(p, c, s) \mid p \in P_{name} \wedge c \in C \wedge s \in \mathbb{B}\},$$

  arvo s ilmaisee näkyvyyttä tulosgraafissa, ominaisuus näytetään tulosgraafissa, kun $s = true$,
- kyselysolmu on $(l_v, P_q) \in V_Q$, missä $l_v \in L_V$ on solmun nimi ja $P_q \subset P_Q$ on siihen liittyvä ominaisuusjoukko,

- kyselykaari on $(l_e, l_{v_1}, l_{v_2}, P_q, t) \in E$, missä $l_e \in L_E$ on kaaren nimi, $l_{v_1} \in L_V$ on lähtösolmun nimi ja $l_{v_2} \in L_V$ maalisolmun nimi, $P_q \subset P_Q$ on ominaisuusjoukko, $t \in \mathbb{B}$ ilmaisee transitiivisuutta eli sitä esittääkö kyselykaari mitä tahansa polkua päätesolmujen nimiä l_{v_1} ja l_{v_2} vastaavien solmujen välillä.

Erityistapauksissa kyselysolmun tai -kaaren nimi on tyhjä merkkijono ϵ, joka ilmaisee tuntemattomia solmuja ja kaaria. Tällöin näillä on erityinen merkitys, että rajaus tapahtuu ainoastaan ominaisuuksien perusteella. Ominaisuus ei voi olla tuntematon, mutta se voi olla arvoltaan *null*, jolloin haetaan puuttuvat ominaisuudet. Sovelluskehityksessä on usein tilanne, jossa tietoa haetaan identiteetin perusteella, jolloin käytetään id-tunnisteita. Näitä ei ole formalisoitu, sillä ne toteutetaan kuin olisivat solmun tai kaaren ominaisuuksia.

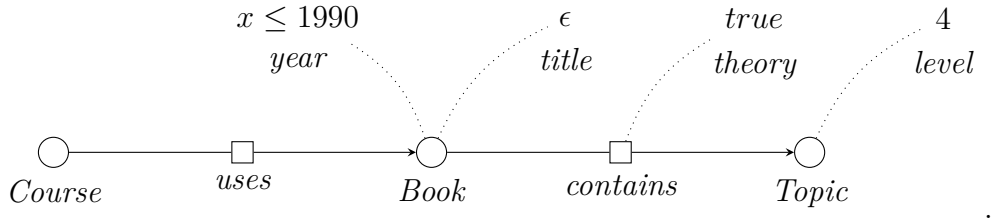
Esimerkki 4. Tarkastellaan kyselygraafia $Q = (V_Q, E_Q)$, jolla haetaan kurssikirjat, jotka on julkaistu ennen 1990 ja aiheen vaativuustaso on 4. Tällöin kyselysolmujen joukko on

$$\begin{aligned} V_Q = \{&(Course, \emptyset), \\ &(Book, \{(year, x \leq 1990, true), (title, x = \epsilon, true)\}), \\ &(Topic, \{(level, x = 4, false)\})\} \end{aligned}$$

ja kyselykaarien joukko

$$\begin{aligned} E_Q = \{&(uses, Course, Book, \emptyset, false), \\ &(contains, Book, Topic, \{(theory, x = true, false)\}, false)\}. \end{aligned}$$

Kyselygraafin Q visuaalinen esitys (yhtäsuuruudet kuten $x = 4$ jätetään merkitsemättä) on



.

Esimerkin 4 kyselygraafin perusteella graafitietokannasta on siis löydyttävä kaaret $(Course, uses, Book)$ sekä $(Book, contains, Topic)$.

6.2 Tulosgraafi

Tulosgraafi $R = (V_R, E_R)$ määritellään graafitietokannan $G = (V, E)$ johdoksena eli kyseessä ei ole aligraafi. Oleellisin ero on, että ominaisuusjoukko sisältää vain osan ominaisuuksista ja alkio sisältää myös kyselysolmun näkyvyys alkion $s \in \mathbb{B}$. Lisäksi kyselykaaren säilyy transitiivisuusalkio $t \in \mathbb{B}$ säilyy tulosgraafissa. Eroa kyselygraafiin on, että id-tunnisteita käytetään nimien tilalla.

Olkoon $G = (V, E)$ graafitietokanta ja $Q = (V_Q, E_Q)$ kyselygraafi. Määritellään tulosgraafi seuraavissa tapauksissa:

1. Jos $E_Q = \emptyset$, niin tulosgraafi on $R = (V_R, \emptyset)$. Tällöin määritellään ainoastaan tulossolmujen joukko V_R. Graafitietokannan solmusta $(id_v, l_v, P_v) \in V$ johdettu tulossolmu (id_v, l_v, P_{R_v}) kuuluu tulossolmujen joukkoon V_R, jos on olemassa kyselysolmu $(l_q, P_q) \in V_Q$ niin, että $l_q = l_v$ tai $l_q = \epsilon$, ja kaikilla kyselysolmun ominaisuuksilla $(p, c, s) \in P_Q$ pätee $\exists (p, x_v) \in P_v$ niin, että $c(x_v) = true$. Ominaisuuksien joukko P_{R_v} koostuu monikoista (p, x_v, s).

2. Jos $E_Q \neq \emptyset$, niin tulosgraafi on $R = (V_R, E_R)$. Olkoon $(id_e, l_e, id_{v_1}, id_{v_2}, P_e) \in E$, $(id_{v_1}, l_{v_1}, P_{v_1}) \in V$ ja $(id_{v_2}, l_{v_2}, P_{v_2}) \in V$. Kaari $(id_e, l_e, id_{v_1}, id_{v_2}, P_{R_e}, t)$ kuuluu tuloskaarien joukkoon E_Q, jos on olemassa kyselykaari $(l_e, l_{v_1}, l_{v_2}, P_q, t) \in E_Q$ niin, että kaikille kyselykaaren ominaisuuksille $(p, c, s) \in P_e$ pätee $\exists (p, x_e) \in P_e$ ehto $c(x_e) = true$. Tuloskaaren ominaisuuksien joukko P_{R_e} koostuu monikoista (p, x_e, s). Tulossolmut $(id_{v_1}, l_{v_1}, P_{R_{v_1}})$ ja $(id_{v_2}, l_{v_2}, P_{R_{v_2}})$ kuuluvat tulossolmujen joukkoon V_R, jos ne voidaan johtaa kohdan 1 perusteella.

Jos kohdan 2 määritelmässä olisikin kyse transitiivisesta kyselykaaresta eli $t = true$, niin kaaren sijasta pitäisi olla polku, jossa jokin kaari täyttää ominaisuusrajoitteet, muulloin määritelmät säilyvät samoina. Transitiivisuutta ei kuitenkaan käsitellä tässä tutkielmassa tämän enempää.

Esimerkki 5. Olkoon $G = (V, E)$ kohdan 5.2 graafitietokanta ja $Q = (V_Q, \emptyset)$ kyselygraafi, missä

$$V_Q = \{(Book, \{(year, x \leq 1990, true), (title, x = \epsilon, true)\})\}.$$

Kun kyselygraafia Q sovitetaan graafitietokantaan G saadaan tulosgraafi $R = (V_R, \emptyset)$, jossa esiintyy vastaavat tulossolmut

$V_R = \{$

$(5, Book, \{(year, 2010, true), (title, $"`Mathematics for Computer Science`"$, true)\}),$

$(7, Book, \{(year, 1989, true), (title, $"`Randomized Algorithms`"$, true)\})$

$\}$.

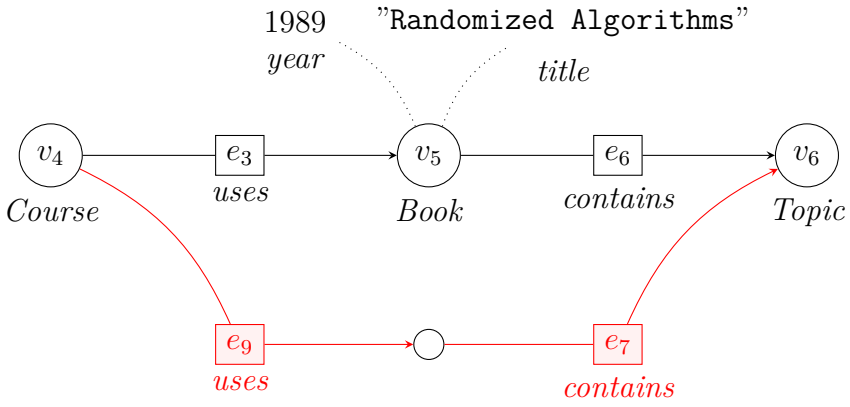
Esimerkki 6. Tarkastellaan esimerkin 4 kyselygraafia vastaavaa tulosgraafia $R = (V_R, E_R)$, missä

$$V_R = \{(4, Course, \emptyset),$$
$$(7, Book, \{(year, 1989, true), (title, \text{"}\texttt{Randomized Algorithms}\text{"}, true)\})\}$$

ja

$$E_R = \{(3, uses, 4, 5, \emptyset, false),$$
$$(6, contains, 5, 6, \{(theory, false)\}, false),$$
$$(7, contains, 7, 6, \{(theory, false)\}, false),$$
$$(9, uses, 4, 7, \{(level, 4, false)\}, false)\}.$$

Tässä tapauksessa toinen solmuista eli `Book`, jolla olisi id-tunniste 6 puuttuu tulossolmujen joukosta V_R, sillä se ei täytä ominaisuusrajoitteita. Merkitään tämä puuttuva solmu tyhjällä solmulla, punaisella värillä merkityt kaaret ovat *viallisia*.



Esimerkki 6 sisältää viallisia kaaria, joten käyttäjälle lopputulosta tulostettaessa voidaan jättää tulostamatta polut, jotka sisältävät näitä kaaria. Lisäksi myös solmut, joilla ei ole yhtään kaarta polussa, joka ei ole viallinen voidaan jättää tulostamatta. Käyttäjälle näytettävä graafi olisi siis hieman erilainen kuin tulosgraafi.

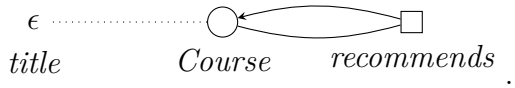
6.3 Erityisiä kyselygraafeja

Graafiteoriassa on olemassa erityistapauksia, jotka voidaan esittää graafitietokannassa. Tässä luvussa pohditaan lyhyesti kyselyissä mahdollisesti esiintyviä luuppeja ja silmukoita. Tällaisiin rakenteiden käsittelyyn liittyvät ikuiset silmukat, joiden kohdalla graafin läpikäynnissä voidaan jäädä jumiin, kun seuraava solmu palaa aina takaisin alkuun. Tavallisia polkugraafeja voidaan pitää myös erityistapauksena, joka

voi vaikuttaa graafin läpikäyntiin, sillä yksi solmu voi kuulua useampaan polkuun ja kaikki solmuun liittyvien kyselygraafin poluille on löydyttävä vastine.

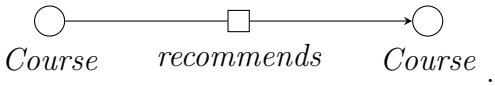
Eräs ongelma on päätesolmun validointi, kun silmukassa lähtösolmun oltava *validi* eli kuuluttava tulosgraafiin, jotta maalisolmu voisi olla validi. Angles ja muut [2017] määrittelevät näiden seikkojen ratkaisemiseksi erilaisia semantiikkoja, joilla voidaan katkaista polun käsittely, jos esimerkiksi sama solmu tai kaari esiintyy. Tässä tutkielmassa lopetetaan läpikäynti, jos aiemmin esiintynyt solmu esiintyy uudelleen. Jos päätesolmut ovat samoja, niin maalisolmu oletetaan validiksi, jolloin lähtösolmu on validi.

Esimerkki 7. Luuppeja voi esiintyä graafitietokannassa ja näitä halutaan löytää. Esimerkkigraafissa on olemassa kaari `recommends` eli kurssi suosittelee itseään ja vastaava kyselygraafi on



.

Myös eri solmut samalla nimellä voivat sisältyä kyselyyn

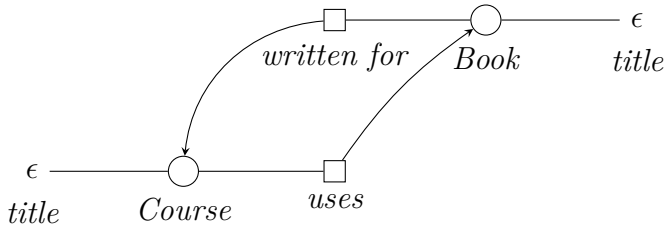


.

Esimerkki 8. Eräs silmukka esimerkkigraafissa

$$(Course, uses, Book, written\ for, Course)$$

etsii kurssit, jotka käyttävät materiaalina kurssia varten kirjoitettuja kirjoja. Vastaava kyselygraafi on



.

6.4 Kyselygraafin läpikäynti

Kyselygraafin läpikäynti hakee graafitietokannasta tiedot solmu kerrallaan ominaisuuksineen ja kaikki niihin liittyvät kaaret päätesolmuineen. Funktio 1 on rekursiivinen tapa käydä graafi läpi kaikkia löydettyjä polkuja pitkin. Toteutus tarkistaa

rekursiivisesti kyselykaaren päätesolmuja. Nämä tarkistukset voidaan optimoida tarkistamalla ensin solmujen tai kaarien olemassaolo tulosgraafista tai hyödyntää välimuistiin tallentamista, jos solmu tai kaari tiedetään varmasti kuuluvan tulosgraafiin. Merkitään ekvivalenssimerkinnällä $v \equiv v_q$ sitä, että solmu v täyttää kyselysolmun v_q ehdot, vastaavasti $e \equiv e_q$. Seuraavat funktiot ovat karkeita ja havainnollistavat toteutuksen ideaa, mutta eivät vastaa sitä täysin. Selkeyden vuoksi tulosgraafia käsitellään graafitietokannan aligraafina. Ne myös rakentavat tulosgraafin hieman eri tavalla kuin kohdan 6.2 tulosgraafi, sillä prototyypissä pyritään karsimaan virheelliset kaaret jo läpikäynnin aikana.

Funktio 1 Kyselygraafin läpikäynti ja tulosgraafin rakentaminen

Syöte: Kyselygraafi $Q = (V_Q, E_Q)$ ja graafitietokanta $G = (V, E)$
Tuloste: Tulosgraafi $R = (V_R, E_R)$

```
1 function TraverseQueryGraph(Q, G)
2     R ← (∅, ∅)                                  ▷ Merkitään R = (V_R, E_R)
3     for all (v, v_q) ∈ V × V_Q do               ▷ Q = (V_Q, E_Q), G = (V, E)
4         (V_r, E_r) ← TraverseNode(Q, G, v, v_q)   ▷ ks. funktio 2
5         if V_r ≠ ∅ then
6             R ← (V_R ∪ V_r, E_R ∪ E_r)
7     return R
```

Funktiossa 2 käydään läpi kaikki solmuun liittyvät kaaret ja sen päätesolmut. Kyselykaaren päätesolmut käsiteltiin kohdassa 6.1 niiden niminä, joten funktiossa 2 kaaren attribuutit *tail* ja *head* esittävät myös päätesolmujen nimiä. Algoritmissa käytetään globaalia tilamuuttujaa $V_{pending}$, joka pitää kirjaa rekursiossa aiemmin käsiteltyjä solmuja ikuisten silmukoiden välttämiseksi. Funktio 3 liittyy läheisesti funktioon 2, sillä ne kutsuvat toisiaan.

Funktio 2 Kyselysolmun ja sen kaarien tarkistus

Syöte: Kyselygraafi Q, graafitietokanta G, kyselysolmu v_q ja graafin solmu v

Tuloste: Graafitietokannan G aligraafi G_r sisältää solmun v sekä siihen liittyvät kaaret päätesolmuineen, jos se täyttää kysesolmun v_q ehdot

```
1  V_pending ← ∅                                          ▷ Rekursiossa olevat solmut
2  function TraverseNode(Q, G, v, v_q)
3      if v ≢ v_q then return (∅, ∅)
4      if v ∈ V_pending then return ({v}, ∅)
5      V_pending ← V_pending ∪ {v}
6      G_r ← ({v}, ∅)                                     ▷ Merkitään G_r = (V_r, E_r)
7      for all e_q ∈ E_Q do                               ▷ Q = (V_Q, E_Q)
8          if e_q.tail = v_q.name ∨ e_q.head = v_q.name then
9              found ← false
10             for all e ∈ E do                           ▷ G = (V, E)
11                 (V'_r, E'_r) ← TraverseEdge(Q, G, e, e_q)   ▷ ks. funktio 3
12                 if E'_r ≠ ∅ then
13                     G_r ← (V_r ∪ V'_r, E_r ∪ E'_r)
14                     found ← true
15             if found = false then return (∅, ∅)
16     V_pending ← V_pending \ {v}
17     return G_r
```

Funktio 3 Kyselysolmun ja sen kaarien tarkistus

Syöte: Kyselygraafi Q, graafi G, kyselykaari e_q ja graafin kaari e

Tuloste: Graafin G aligraafi, jossa on kaaret ja niiden päätesolmut

```
1  function TraverseEdge(Q, G, e, e_q)
2      if e ≡ e_q then
3          v_1 ← Node(G, e.tail)
4          v_q1 ← Node(Q, e_q.tail)
5          v_2 ← Node(G, e.head)
6          v_q2 ← Node(Q, e_q.head)
7          (V_r1, E_r1) ← TraverseNode(Q, G, v_1, v_q1)
8          (V_r2, E_r2) ← TraverseNode(Q, G, v_2, v_q2)
9          if V_r1 ≠ ∅ ∧ V_r2 ≠ ∅ then
10             e_r = (e.id, e.name, v_1.id, v_2.id, e.properties)
11             return (V_r1 ∪ V_r2, E_r1 ∪ E_r2 ∪ {e_r})
12     return (∅, ∅)
```

6.5 Ominaisuuksien arvojen tarkistaminen

Tässä kohdassa käsitellään tarkemmin täsmäyksen toteuttamisesta. Poiketaan kohdan 6.1 formalismista, sillä tämä kohta käsittelee ohjelmallista tarkistusta ominaisuuspareille (p, x) ja (p, x_q). Mahdollisia tilanteita ovat siis

1. samat arvot $x_q = x$,
2. ominaisuuden arvo x voi olla mitä tahansa $x_q = \epsilon$,
3. ominaisuuden arvo puuttuu $x_q = null$ ja ominaisuutta (p, x) ei ole määritelty,
4. x on merkkijono ja se tarkistetaan säännöllisellä lausekkeella,
5. x on numeerinen arvo ja se tarkistetaan loogisella lausekkeella.

Täsmällinen haku on helpointa toteuttaa, sillä ohjelmointikielet sisältävät sille valmiit vertailuoperaattorit. Oletetaan, että vertailuoperaattoreita on tarkoitus soveltaa arvoille, joilla on sama tyyppi. Funktio 4 näyttää järjestyksen, jossa arvoja vertaillaan. Arvovälit merkitään loogisina lausekkeina.

Säännöllisten lausekkeiden tulkinta kuuluu yleensä ohjelmointikielen standardikirjastoihin. Esimerkiksi Java toteuttaa `Pattern`[7]-luokan, jolla voi täsmätä merkki-

[7] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/regex/Pattern.html`

Funktio 4 Ominaisuuden arvon tarkistaminen

Syöte: Kyselyn ominaisuuden arvo x_q ja tietokannasta haettu arvo x

Tuloste: Totuusarvo $\mathbb{B}$ siitä hyväksytäänkö ominaisuus

```
1 function checkProperty(x_q, x)
2     if x_q = ϵ ∨ (x_q = null ∧ x = null) then
3         return true
4     if x_q ∈ 𝕊 then                          ▷ 𝕊 on merkkijonojen joukko
5         return CheckRegularExpression(x_q, x)
6     if x_q ∈ ℒ then                          ▷ ℒ on joukko loogisia lausekkeita
7         return CheckLogicalExpression(x_q, x)
8     return x_q = x
```

jonoja. Tuki on siis olemassa kaikille merkinnöille, joita `Pattern`-luokka tukee.

6.6 Loogisten lausekkeiden tulkinta

Tässä tutkielmassa loogisten lausekkeiden syntaksi poikkeaa tavallisesta logiikan merkinnöistä niin, että merkitään operaattorit funktioina $\text{AND}(z_1, z_2)$, $\text{OR}(z_1, z_2)$ ja $\text{NOT}(z_1)$, missä z_i on jokin luku tai looginen lause. Tätä merkintää käytetään, sillä se muistutaa *puolalaista notaatiota (polish notation)*, jota on helppo käsitellä *pino-tietorakenteen* avulla. Perinteinen menetelmä on muuntaa kaikki merkinnät muodosta $a \circ b$ (infix) puolalaiseen merkintätapaan ja käsitellä se puu-tietorakenteena [Krtolica ja Stanimirović, 2004].

Avainsanalla NOT on kaksi tarkoitusta: (1) se tulkitaan erisuuruutena $\neq$, kun argumentti on numero ja (2) logiikan negaationa $\neg$, kun argumentti on totuusarvo.

Esimerkki 9. Eräs tapa välin $[a, b]$ merkitsemiseksi on $\text{AND}(\geq a, \leq b)$. Jos lisäksi rajataan arvo c pois, merkitään $\text{AND}(\geq a, \text{AND}(\leq b, \text{NOT}(c)))$.

Loogisten lausekkeiden tulkitseminen tapahtuu niin, että kyselygraafiin säilötään lauseke sellaisenaan ominaisuuden arvoksi ja se tulkitaan jokaisen solmun kohdalla uudelleen. Vaihtoehtoisesti tämän voisi toteuttaa niin, että lausekkeesta muodostetaan puu.

Funktio 5 kuvailee tapaa tulkita loogiset lausekkeet pinon avulla. Tässä algoritmissa looginen lauseke L pilkotaan merkkijonosta taulukoksi (array), taulukoilla on attribuutti *length*, joka kertoo sen pituuden. Sulkumerkinnällä kuten $T[1]$ valitaan alkio indeksistä 1. Tässä tutkielmassa kaikkien taulukoiden indeksointi aloitetaan

luvusta 1. Apufunktio COMPARE suorittaa vertailuoperaation kahden luvun välillä ja EVALUATE loogiset operaatiot, esimerkiksi merkinnällä $\textsc{Compare}(\circ, a, b)$ tarkoitetaan $a \circ b$.

Funktio 5 Loogisen lausekkeen tulkinta pinon avulla (vrt. [Rosetta Code, 2022])

Syöte: Looginen lauseke L merkkijonona ja vertailuarvo x_0

Tuloste: Totuusarvo $b \in \mathbb{B}$

```
1  function CheckLogicalExpression(L, x_0)
2      T ← Tokenize(L)                          ▷ Erotellaan tunnistettavat sanat
3      S ← emptyStack
4      for i ← T.length to 1 do                 ▷ Käsitellään käänteisessä järjestyksessä
5          t ← T[i]
6          if t ∈ {=, >, ≥, <, ≤} then
7              x_1 ← Pop(S)                     ▷ Pinon päältä arvo x_1 ∈ L
8              r ← Compare(t, x_0, x_1)         ▷ x_0 ∘ x_1, missä ∘ = t
9          else if t ∈ {AND, OR, NOT} then
10             x_1 ← Pop(S)                     ▷ Pinon päältä arvo x_1 ∈ 𝔹
11             if t = NOT ∧ x_1 ∈ 𝔹 then
12                 S ← Push(S, ¬x_1)
13             else
14                 x_2 ← Pop(S)                 ▷ x_2 ∈ 𝔹
15                 r ← Evaluate(t, x_1, x_2)    ▷ x_1 ∘ x_2, missä ∘ = t
16                 S ← Push(S, r)
17         else
18             S ← Push(S, t)
19     r ← Pop(S)
20     if r ∈ 𝔹 then return r
21     return r = x_0
```

Taulukko 2 havainnollistaa lausekkeen `AND(>= 1, AND(<= 6, NOT(5)))` tulkintaa pinon S avulla arvolle 2. Taulukon viimeinen sarake esittää pinon lisäämisen jälkeistä operaatiota.

Lauseke	**Pino** S	**Tulkinta**
`AND >= 1 AND <= 6 NOT 5`	$()$	
`AND >= 1 AND <= 6 NOT`	(5)	$2 \neq 5$
`AND >= 1 AND <= 6`	$(true)$	
`AND >= 1 AND <=`	$(6, true)$	$2 \leq 6$
`AND >= 1 AND`	$(true, true)$	$true \wedge true$
`AND >= 1`	$(true)$	
`AND >=`	$(1, true)$	$2 \geq 1$
`AND`	$(true, true)$	$true \wedge true$

Taulukko 2: Loogisen lausekkeen tulkinta.

6.7 Graafien yhdistäminen

Graafien yhdistäminen on tarpeen tulosgraafeilla, kun useammasta tulosgraafista muodostetaan uusi graafi. Tätä operaatiota hyödynnetään myöhemmin taulukkomuotoisessa kyselyssä yhdistämään useamman rivin kyselyissä tulosgraafit yhdeksi graafiksi. Periaate muistuttaa Cruzin ja muiden [1987] tapaa muotoilla kysely useamman graafin avulla. Funktio 6 kuvaa graafien G_1 ja G_2 yhdistämistä. Merkintä $P_1 \cup_\times P_2$ tarkoittaa yhdistettä $\cup$, mutta samannimisillä ominaisuuksilla on eri arvot, jolloin arvo valitaan joukosta P_2 eli

$$P_1 \cup_\times P_2 = P_2 \cup \{(p, x_1) \mid (p, x_1) \in P_1 \wedge (p, x_2) \notin P_2\}.$$

Funktio 6 Kahden graafin yhdistäminen.

Syöte: Graafit $G_1 = (V_1, E_1)$ ja $G_2 = (V_2, E_2)$

Tuloste: Graafi $G = G_1 \cup G_2$

1 **function** GraphUnion(G_1, G_2)
2 $G \leftarrow G_1$ ▷ Merkitään $G = (V, E)$
3 **for all** $(id_v, l_v, P_2) \in V_2$ **do** ▷ Iteroidaan solmut
4 $(\_, \_, P_1) \leftarrow$ Node(G, id_v) ▷ $P_1 = \emptyset$, jos ei ole olemassa
5 $v \leftarrow (id_v, l_v, P_1 \cup_\times P_2)$
6 $G \leftarrow (V \cup \{v\}, E)$
7 **for all** $(id_e, l_e, id_{v_1}, id_{v_2}, P_2) \in E_2$ **do**
8 $(\_, \_, \_, \_, P_1) \leftarrow$ Edge(G, id_e) ▷ $P_1 = \emptyset$, jos ei ole olemassa
9 $e \leftarrow (id_e, l_e, id_{v_1}, id_{v_2}, P_1 \cup_\times P_2)$
10 $G \leftarrow (V, E \cup \{e\})$
11 **return** G

7 Taulukkomuotoinen QBE-kyselykieli

Taulukkomuotoisen kyselykielen perustana käytetään relaatiotietokantojen QBE-kyselykieltä (ks. [Zloof, 1975a,b]). Relaatiotietokannat ovat taulukkomuotoisia, ja niille on määritelty relaatioalgebran perusoperaatiot *projektio (projection)*, *liitos (join)* ja *joukko-operaatiot* [Codd, 1970]. Tässä luvussa määritellään vastineet graafitietokannan QBE-taulukoille niin, että kysely- ja tulosgraafit rakentuvat näiden pohjalta. Tämä luku käsittelee vain yksinkertaisia taulukoita, joissa on yksi rivi ja painotus on tietojen hakemisessa. Tietojen muokkaaminen esitellään lisäominaisuusluvussa 9.

7.1 Taulukot

Solmut ja kaaret esitetään taulukkomuodossa eli ominaisuudet merkitään sarakkeina. Ensimmäinen sarake QBE-kielessä on taulun nimi [Zloof, 1975a], joten sitä vastaa solmun tai kaaren nimi. Tätä saraketta kutsutaan tässä tutkielmassa *nimisarakkeeksi*. Tämän sarakkeen arvoiksi kelpaavat avainsanat `INSERT`, `DELETE`, ja `UPDATE` [Zloof, 1975a]. Näiden lisäksi käytetään uutta avainsanaa `QUERY`, joka asettaa rivin lukumuotoon. Muut sarakkeet eli *ominaisuussarakkeet* esittävät solmuille tai kaarille liitettyjen ominaisuuksien arvoja. Näille annetaan esimerkkiarvot, joita käytetään suodattamaan tuloksia. Ominaisuudet, jotka halutaan tulosgraafiin, on aina kirjoitettava taulukkoon, vaikka kenttä olisi tyhjä.

Aiemmin mainittiin nimeämiskäytännöstä, että isolla kirjaimella alkavat taulukon nimet tunnistetaan solmuiksi ja pienellä kaareksi. Kaarilla ja ominaisuuksilla voi kuitenkin olla sama nimi, joten tyyppi tunnistetaan nimisarakkeen arvon avulla. Tämä ei kuitenkaan toimi eri nimikäytäntöjen kanssa.

7.1.1 Ominaisuussarakkeet

Tässä kohdassa määritellään ominaisuussarakkeiden arvot. Ominaisuussarakkeet eivät tarvitse erillistä `QUERY`-määritettä. Muita *lisämääritteitä (modifier)* käytetään kuitenkin tässä tutkielmassa, joten ne merkitään ennen esimerkkiarvoa

`[`***määrite***`] ` ***arvo***.

Tarkempi syntaksi esitellään käyttötapauksen mukaan, joten se voi sisältää myös arvon jälkeen muita avainsanoja. Ominaisuussarakkeita voidaan käyttää hakuehtoina rajaamaan tuloksia ja määrittämään sarakkeiden näkyvyys tuloksessa. Jos sarake palautetaan tulostaulussa, niin sarakkeen otsikon viimeiseksi merkiksi lisätään `*`.

Merkkijonot merkitään lainausmerkein `"teksti"` ja tyhjät merkkijonot `""`. Säännölliset lausekkeet ovat myös merkkijonoja, mutta lainausmerkkien tilalla on merkki `/` kuten `/Introduction to .*/`. Loogiset lausekkeet merkitään muodossa `>= 1` ja `AND(> -1, < 1)` eli käytettävät merkit ovat `>`, `>=`, `=`, `<`, `<=`, `(`, `)` sekä avainsanat `AND`, `OR` ja `NOT`.

Kokonaisluvut merkitään sellaisenaan `2022` ilman välimerkkejä ja reaaliluvut `2.022` pisteellä erotettuna. Totuusarvot vastaavat avainsanoja `true` ja `false`. Jos sarake on tyhjä, niin mikä tahansa arvo tulkitaan kelpaavaksi. Määrittelemättömät ominaisuudet haetaan avainsanalla `NULL`.

7.1.2 Solmut taulukkona

Solmujen merkintä taulukkona muistuttaa relaatiotietokannan QBE-kyselyä. Yleinen rakenne on

```
| Solmu | ominaisuus       |
|-------+------------------|,
| QUERY | esimerkkiarvo    |
```

missä ominaisuussarakkeita voi olla useampia.

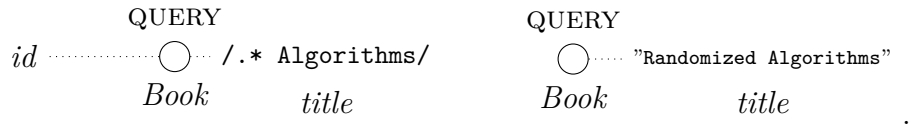
Esimerkki 10. Tarkastellaan solmua `Book`, jolla on ominaisuudet `id` ja `title`. Kyselytaulukko

```
| Book  | id* | title*           |
|-------+-----+------------------|
| QUERY |     | /.* Algorithms/ |
```

palauttaa tulostaulun

```
| id | title                     |
|----+---------------------------|·
| 7  | "Randomized Algorithms" |
```

Vastaavat kysely- ja tulosgraafit ovat



.

7.1.3 Kaaret taulukkona

Kaaret eroavat solmuista niin, että nimisarakkeeseen merkitään lähtö- ja maalisolmujen nimet muodossa

QUERY ***Lähtösolmu.Maalisolmu***.

Kyselygraafi muotoutuu myös eri tavalla, sillä siihen lisätään epäsuorasti myös lähtö- ja maalisolmut, vaikka niitä ei olisi erikseen määritelty. Kaarilla on olemassa ääritapaus, jossa se on luuppi eli päätesolmut ovat samat. Tällöin merkitään

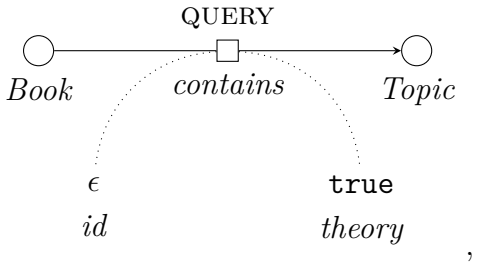
QUERY ***Solmu.Solmu***

eli solmujen nimet ovat samoja. Tämän lisäksi voi olla tapaus, jossa tulokseen halutaan kaaret muihin samannimisiin solmuihin, mutta ei luuppeja. Tämän erottaminen on kuitenkin ongelmallinen, joten idea kuvaillaan lisäominaisuutena.

Esimerkki 11. Tarkastellaan kaarta `contains`, joka liittää kirjan johonkin aihepiiriin

contains	id*	theory*
QUERY Book.Topic		true

Tämä muodostaa kaaren `contains` solmujen `Book` ja `Topic` välille. Kyselygraafi on seuraavanlainen



sillä kaari ei voi esiintyä yksinään ilman solmuja. Tulostaulu näyttää

id	theory
6	true
7	true

7.2 Liitokset

Liitosoperaatio (join) poikkeaa QBE-kyselykielestä [Zloof, 1975a,b] niin, että käytetään vain yhtä taulukkoa. Alkuperäisessä QBE-kielessä käytetään kahta taulukkoa, jossa molemmilla on sama sarake ja esimerkkiarvo [Zloof, 1975a].

Kun merkitään samaan taulukkoon sekä solmu että kaari, voi olla tarpeen erottaa ominaisuudet merkitsemällä niiden kaaren tai solmun nimi ominaisuuden eteen

nimisarake.ominaisuus.

Tämä muistuttaa SQL-kielen tapaa erotella samannimisiä sarakkeita (ks. [PostgreSQL 14, 2021, s. 11–12]). Jos ominaisuuden edessä ei ole nimeä, niin se oletetaan lähimmäksi nimisarakkeeksi vasemmalla. Lisäksi usean solmun tapauksessa on aina määriteltävä kaari näiden välille.

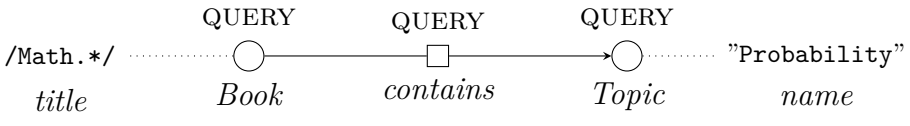
Esimerkki 12. Seuraava taulukko liittää kaksi solmua, missä haetaan todennäköisyyttä käsitteleviä kirjoja

```
| Book  | Topic | contains          | Book.title* | Topic.name*   |
|-------+-------+-------------------+-------------+---------------|.
| QUERY | QUERY | QUERY Book.Topic | /Math.*/    | "Probability" |
```

Sama taulukko voidaan merkitä myös niin, että ominaisuuksille voidaan päätellä solmu ja kaari

```
| Book  | title*    | Topic | name*          | contains          |
|-------+-----------+-------+----------------+-------------------|.
| QUERY | /Math.*/  | QUERY | "Probability"  | QUERY Book.Topic |
```

Molemmat taulukot jäsentyvät samanlaiseksi kyselygraafiksi



.

Tulostaulu näyttää seuraavanlaiselta

```
| Book.title                          | Topic.name    |
|-------------------------------------+---------------|.
| "Mathematics for Computer Science" | "Probability" |
```

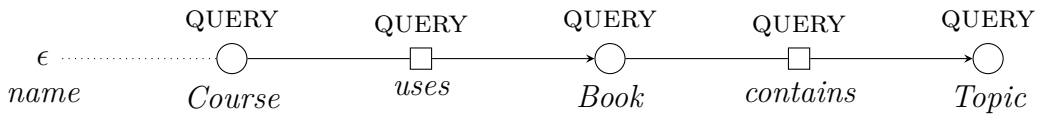
7.3 Polut

Liitoksiin liittyy läheisesti myös polut, kyselygraafissa voi olla silmukoita ja useita polkuja. Polut merkitään niin, että kaikki solmut ja kaaret ovat samassa taulukossa.

Esimerkki 13. Suoraa polkua $(Course, uses, Book, contains, Topic)$ esitetään taulukolla

```
|  uses              | contains          | Course | name* |
|--------------------+-------------------+--------+-------|.
| QUERY Course.Book | QUERY Book.Topic | QUERY  |       |
```

Tämä muodostaa kyselygraafin, joka on polkugraafi

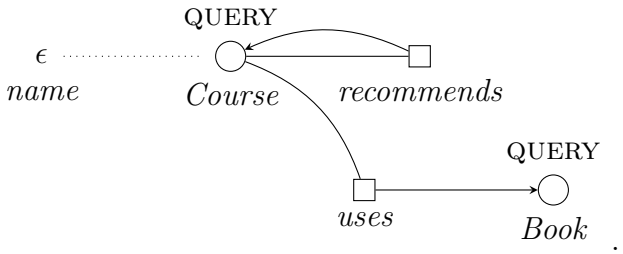


Yleisesti kaikki graafit, joissa kaikkien solmujen välillä on olemassa jokin polku, voidaan merkitä taulukkona. Graafitietokannassa poluilla on suunta, mutta sillä ei ole merkitystä graafin läpikäymisen kannalta.

Esimerkki 14. Kahden polun $(Course, recommends, Course)$ ja $(Course, uses, Book)$ merkintä tarkoittaa, että tietokannassa on oltava molemmat polut

```
| Course  | id* | name* | recommends           | uses               |
|---------+-----+-------+----------------------+--------------------|·
| QUERY   |     |       | QUERY Course.Course | QUERY Course.Book |
```

Kyselygraafi näyttää haarautuvalta



Kaarilla voi olla sama nimi, kun nimisarakkeen arvo sisältää eri solmuja. Esimerkiksi taululla

```
| contains           | contains          | Topic | name* |
|--------------------+-------------------+-------+-------|
| QUERY Course.Topic | QUERY Book.Topic | QUERY |       |
```

löytyy yksi aihe

```
| name           |
|----------------|·
| "Graph Theory" |
```

7.4 Projektio

Edellä mainitut operaatiot rajaavat rivejä tuloksesta, mutta ei sarakkeita. Kaikkia sarakkeita ei kuitenkaan aina tarvita tuloksessa, mutta tarvitaan hakuehdoissa. Sarakkeiden rajaamiseksi hyödynnetään projektiota, joka rajaa pois sarakkeita [Codd,

1970]. QBE-kyselykieli käyttää merkintää `P.` (print) sarakkeen arvon edessä, jolloin se sisällyttää sarakkeen tulokseen [Zloof, 1975a]. Tästä poiketaan ja käytetään merkkiä `*`, joka lisätään sarakkeen viimeiseksi merkiksi. Jos `*`-merkki löytyy, niin sarake näytetään tuloksessa. Sarakkeet voidaan tämä lisäksi nimetä uudelleen avainsanalla `AS` eli *aliaksena*, joka tuttu mm. SQL-kielestä (vrt. [PostgreSQL 14, 2021, s. 116]). Tämä on tarpeen, kun sarakkeiden nimi on kokonaisuudessaan esillä ja halutaan lyhyt nimi tulokseen tai samannimisiä sarakkeita on useita. Ominaisuussarakkeen otsakkeelle saadaan yleinen sääntö

```
[ nimi. ]ominaisuus [ AS alias ][*].
```

Esimerkki 15. Havainnollistetaan kyselyä ja sen tulosta yksinkertaisella esimerkillä

```
| Book  | title AS Title* | contains          | level |
|-------+-----------------+-------------------+-------|.
| QUERY | /Math.*/        | QUERY Book.Topic | >= 3  |
```

Tulostauluksi muodostuu

```
| Title                                |
|--------------------------------------|.
| "Mathematics for Computer Science" |
```

7.5 Rivien koostaminen

Yleisiä tietokantaoperaatioita ovat myös rivien *koostaminen (aggregation)*. Esimerkkejä SQL-kielessä ovat `COUNT`, `SUM`, `AVG` sekä `MIN` ja `MAX` [PostgreSQL 14, 2021, s. 13–14]. Cypher toteuttaa näiden lisäksi myös normaalijakauman laskufunktioita ja prosenttilaskuja [Neo4j, 2022, s. 234–247]. Tässä tutkielmassa tarkastellaan erikseen funktioita `COUNT` ja `SUM`, sillä muiden funktioiden toteutukset muistuttaisivat funktiota `SUM`. Eroja näissä kahdessa ryhmässä on, että funktiota `COUNT` käytetään nimisarakkeessa ja muut ominaisuussarakkeessa. Koontiin liittyy myös *ryhmittely (grouping)* -operaatiot (ks. [PostgreSQL 14, 2021, s. 121–123]), mutta näitä ei erityisemmin käsitellä tässä tutkielmassa. Koontioperaatiot vaativat kuitenkin epäsuorasti ryhmittelyä, sillä esimerkiksi kaaren laskeminen solmujen suhteen tarkoittaa ryhmittelyä solmuille.

Zloof [1975b] määrittelee merkinnän alleviivauksella sen sarakkeen esimerkkiarvon kohdalle, jonka suhteen kootaan rivit. XQBE-kyselykieli toteuttaa koontioperaatiot kyselypuussa lisäämällä erillisiä *koontisolmuja (aggregation node)* [Braga *et al.*, 2005]. Tässä tutkielmassa käytetään avainsanoja, joilla laajennetaan kyselygraafin solmuja. Tulosgraafi voi sisältää *koottuja solmuja*, jotka eivät vastaa mitään graafitietokannan solmua, mutta toimivat esim. laskurina.

Tässä tutkielmassa ei oteta huomioon sisäkkäisten koontifunktioiden käyttöä eikä sen käyttämistä hakuehtona. Vain yksinkertaiset tapaukset, joissa on yksi koontifunktio, käsitellään. Prototyypissä koontifunktioita kutsutaan kohdassa 6.4 määriteltyjen funktioiden TraverseNode ja TraverseEdge lopussa ennen arvojen palauttamista.

7.5.1 Koontifunktio `COUNT`

Koontifunktiota `COUNT` käytetään laskemaan solmujen tai kaarien lukumääriä. Koontifunktio `COUNT` merkitään solmun nimisarakkeeseen muodossa

`COUNT [ AS` ***`alias`*** `]`.

Nimi `alias` määrittää lopputuloksen sarakkeen nimen, oletuksena käytetään nimeä `Solmu.count`. Jos funktiota `COUNT` sovelletaan kaareen, niin myös päätesolmut merkitään

`COUNT` ***`Lähtösolmu.Maalisolmu nimisarake`*** `[ AS` ***`alias`*** `]`.

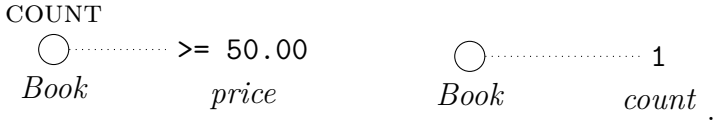
Tämä tapa ei ole kuitenkaan riittävän ilmaisuvoimainen, jos yhteen ryhmään kuuluu vain osa sarakkeista. Tässä tapauksessa mukaan tulee kaikki määritellyt sarakkeet.

Prototyypissä funktiolla `COUNT` on rajoite, että solmuja voidaan laskea vain, jos ne esiintyvät yksinään ilman kaarta tai se toimii lähtösolmuna muihin solmuihin. Jos polkuun liittyvien solmujen määriä halutaan laskea, niin kaarien laskemisella saadaan sama tulos.

Esimerkki 16 (Yksinkertainen laskenta)**.** Yksinkertaisimillaan lasketaan yhden tulostaulun rivit ilman ryhmittelyä

```
| Book  | price    |            | Book.count |
|-------+----------|            |------------|.
| COUNT | >= 50.00 |            | 1          |
```

Kysely (vasemmalla) laskee yli 50,00 rahayksikköä maksavien kirjojen lukumäärän. Kyselygraafi (vasemmalla) `Book` sisältää koontisolmun ja tulosgraafi (oikealla) sisältää vastaavan ominaisuuden `count`



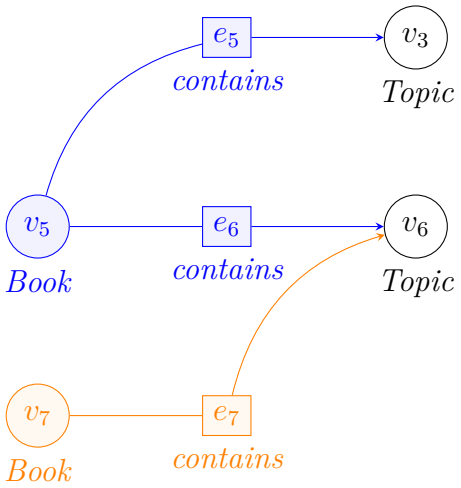
Esimerkki 17 (Kaarien laskeminen solmujen suhteen)**.** Tarkastellaan tapausta, jossa ryhmitellään kirjat ja lasketaan käsiteltävien aiheiden lukumäärä

```
| Book   | title* | contains                                |
|--------+--------+-----------------------------------------|.
| QUERY  |        | COUNT Book.Topic Book AS topics         |
```

Kysely ryhmittelee ensin kirjat ominaisuuden `id` suhteen ja tämän jälkeen laskee kaarien `contains` lukumäärät (ks. kuva 4). Tulostauluksi saadaan tällöin

```
| title                                | topics |
|--------------------------------------+--------|
| "Mathematics for Computer Science" | 2      |
| "Randomized Algorithms"            | 1      |
```

Kuva 4 havainnollistaa ryhmittelyä esimerkkigraafista, jossa on kaksi ryhmää eli kirjaa. Samalla värillä merkityt elementit kuuluvat samaan ryhmään. Tässä ilmenee klassinen *graafin väritys (graph coloring)* -ongelma, sillä ryhmittely voidaan toteuttaa tulosgraafiin värittämällä ensin ryhmään kuuluvat solmut ja kaaret, jonka jälkeen kootaan ne värien mukaan. Solmut `Topic` eivät kuulu ryhmittelyyn.



Kuva 4: Havainnollistus esimerkkigraafin ryhmittelystä.

7.5.2 Funktion `COUNT` toteutus

Koontifunktio `COUNT` kutsutaan aina polkujen validoinnin päätteeksi, jolloin laskuri kasvattaa koontisolmun ominaisuuden `_count` arvoa yhdellä. Jos ominaisuutta ei ole vielä määritelty, asetetaan alkuarvoksi 1. Koontisolmuja voi olla useampia, jos koottavia ryhmiä enemmän kuin yksi. Koontisolmu voi olla jokin toinen solmu, jonka suhteen lasketaan tai se on lisätty vain laskuria varten.

Funktio 7 on osa laajempaa kokonaisuutta graafin läpikäymisessä. Tässä havainnollistetaan vain laskurin osuutta, jota kutsutaan jokaisen solmun kohdalla. Apufunktio FindAggregationNode etsii ensin tulosgraafista koontisolmun ja jos sel-

laista ei löydy, niin se etsitään polun w solmuista. On mahdollista, että koontisolmua ei ole olemassa, tällaiset tilanteet ratkaistaan niin, että funktio FindAggregationNode luo sellaisen tarvittaessa. Tulossolmuille ja -kaarille määritellään attribuutti *aggregated*, joka kertoo sen, onko solmu- tai kaari jo laskettu.

Funktio 7 Koontifunktio `COUNT`

Syöte: Polun kulku w, tulosgraafi R, kyselysolmu tai -kaari q, tulossolmu tai -kaari r, solmun nimi g, jonka suhteen kootaan

Tuloste: Tulosgraafi R sisälten kootut tiedot

```
1  function AggregateCount(R, q, r, g, w)
2      v ← FindAggregationNode(R, q, g, w)
3      if v ≠ null ∧ r.aggregated = false then
4          x ← Property(v, _count)
5          if x = null then
6              v ← SetProperty(v, _count, 1)
7          else
8              v ← SetProperty(v, _count, x + 1)
9          r.aggregated = true
10     return (V_R ∪ {v}, E_R)                    ▷ R = (V_R, E_R)
```

7.5.3 Koontifunktio `SUM`

Koontifunktiota `SUM` käytetään laskemaan numeeristen sarakkeiden summia eli graafitietokannassa ominaisuuksien arvojen summa. Zloof [1975a] käyttää `SUM`-funktiota esimerkkinä rivien hakuehdoille muodossa `SUM ALL = arvo`. Prototyypissä ei oteta tällaisia hakuehtoja huomioon. Ryhmiteltävä solmu tai kaari merkitään aina, ja koonnissa polun alkiot välissä jätetään huomiotta. Yleinen syntaksi on

`SUM` ***Solmu***,

missä hakuehto on mikä tahansa esimerkkiarvo tai looginen lauseke ominaisuudelle. Alaviivaa `_` voidaan käyttää silloin, kun ominaisuus on samalle solmulle. Summa täytyy aina koota solmun suhteen, sillä kaarella voi olla vain yksi lähtö- ja maalisolmu, jolloin koonti summaisi vain yhden rivin. Merkintää voidaan laajentaa myös rajaamaan tuloksia ennen summaamista

`SUM` ***Solmu*** `[` ***hakuehto*** `]`,

mutta tätä ei käsitellä tässä tutkielmassa.

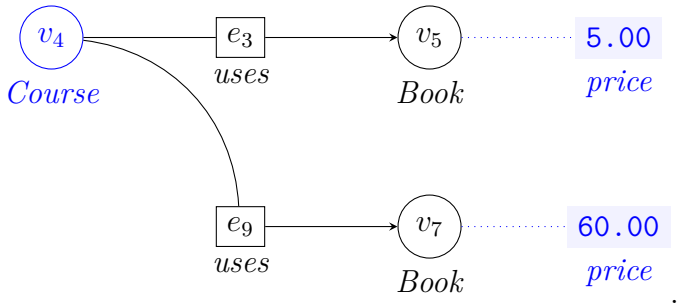
Esimerkki 18 (Numeeristen arvojen laskuoperaatiot)**.** Lasketaan kurssin oppikirjakustannukset

```
| Course | name* | uses               | Book   | price*      |
|--------+-------+--------------------+--------+-------------|.
| QUERY  |       | QUERY Course.Book  | QUERY  | SUM Course  |
```

Tällöin saadaan tulostaulu

```
| name                                  | price |
|---------------------------------------+-------|.
| "Mathematics for Computer Science"    | 65.00 |
```

Tämä löydetään alla näkyvästä graafitietokannan aligraafista



7.5.4 Funktion `SUM` toteutus

Lukujen summan määrittäminen muistuttaa hieman lukumäärän laskemista. Ero on siinä, kuinka laskuoperaatio toteutetaan. Apufunktio FindAggregationNode on sama kuin kohdassa 7.5.3 `COUNT`-funktion yhteydessä. Tulossolmujen ja -kaarien attribuutti *aggregated* on myös sama. Funktio 8 on myös funktion 7 tavoin osa isompaa kokonaisuutta, josta havainnollistetaan summaoperaatiota.

Funktio 8 Koontifunktio `SUM`

Syöte: Polun kulku w, tulosgraafi R, kyselysolmu tai -kaari q, tulossolmu tai -kaari r, summattavan ominaisuuden nimi $p \in P_{name}$, solmu nimi g, jonka suhteen kootaan

Tuloste: Tulosgraafi R sisältäen kootut tiedot

```
1  function AggregateSum(R, q, r, g, p, w)
2      v ← FindAggregationNode(R, q, g, w)
3      if v ≠ null ∧ r.aggregated = false then
4          x_1 ← Property(v, p)
5          x_2 ← Property(r, p)
6          v ← SetProperty(v, p, x_1 + x_2)
7      else
8          x ← Property(v, p)                    ▷ x = 0, jos ei ole määritelty.
9          v ← SetProperty(v, p, x)
10     return (V_R ∪ {v}, E_R)                   ▷ R = (V_R, E_R)
```

Jos tarkoituksena on laskea keskiarvoja `AVG`, niin ne voidaan laskea tulosgraafin muodostamisen jälkeen. Funktiosta 8 huomataan, että funktio `MAX` voidaan toteuttaa suoraviivaisesti korvaamalla ominaisuuden yhteenlasku asettamalla suurempi arvo ominaisuuden arvoksi. Vastaavasti myös `MIN`.

7.6 Useamman rivin kyselyt

Useamman rivin kyselyitä tarvitaan erilaisten hakujen toteuttamiseksi silloin, kun erilaisia ehtoja sovelletaan samoille ominaisuuksille. Tällaisten kyselyjen tulkinta toteutetaan niin, että jokaisesta rivistä rakennetaan oma kyselygraafi ja vastaava tulosgraafi. Lopullinen tulosgraafi on tulosgraafien unioni. Kyselygraafit käydään läpi siinä järjestyksessä kuin ne riveinä esiintyvät. Jos kahdella tulosgraafilla samoilla solmuilla tai kaarilla on samannimisiä ominaisuuksia eri arvoilla, valitaan arvo viimeisimmästä tulosgraafista.

Esimerkki 19. Tarkastellaan taulukkoa, joka hakee kaksi eri joukkoa kirjoja eri ehdoilla

```
| Book  | title* | year    | price |
|-------+--------+---------+-------|
| QUERY |        | < 1990  |       |
| QUERY |        |         | < 10  |
```

Kyselygraafeja olisi tällöin kaksi kappaletta rivinumeroita vastaavasti Q_1 ja Q_2. Tulosgraafi R on näitä vastaavien tulosgraafien $R_1 = (V_{R_1}, E_{R_1})$ ja $R_2 = (V_{R_2}, E_{R_2})$ unioni

$$R = R_1 \cup R_2 = (V_{R_1} \cup V_{R_2}, E_{R_1} \cup E_{R_2}).$$

8 QBE-taulukoiden jäsennys ja tulostus

Kyselykielen käsittelyssä ensimmäinen vaihe on *jäsentää (parse)* kysely merkkijonosta graafiksi. Jäsennysprosessi jaetaan kahteen osaan (1) sarakkeiden otsikot ja (2) rivit. Taulukoissa on säännönmukaisuus, joten rivit löydetään pilkkomalla merkkijonot rivinvaihtomerkkien `\n` ja sarakkeiden jakajan `|` perusteella. Sarakkeiden erottimien kanssa on huomioitava niiden käyttö säännöllisissä lausekkeissa ja merkkijonoissa.

8.1 Otsakkeiden jäsentäminen

Otsikot sisältävät sarakkeen nimen ja valinta siitä näytetäänkö sarake tuloksessa. Funktio 9 näyttää otsakkeiden jäsennyksen otsakkeeksi h', jolla on attribuutit

- *select* totuusarvo siitä näytetäänkö ominaisuus lopputuloksessa,
- *entity* solmun tai kaaren nimi,
- *name* ominaisuuden nimi,
- *display* ominaisuuden nimi tulostaulussa.

Kuvauksesta jätetään pois ylimääräisten välimerkkien poistot. Funktio 9 käsittelee nimen pilkkomista, missä kaikki otsakkeen jäsenet on merkitty esiin. Toteutuksessa eri tapaukset tulee ottaa huomioon lisäämällä `if`-lauseita eri tilanteisiin. Merkintä $h[1 : h.length - 1]$ tarkoittaa, että merkkijonosta valitaan osa eli viimeinen merkki jää pois. Apufunktio Split pilkkoo merkkijonot valitun merkkijonon kohdalta taulukoksi (array), pilkkomiskohdan merkkijonoa ei sisällytetä paluuarvoon.

Funktio 9 Solmun ja kaaren otsakkeiden jäsentäminen

Syöte: Sarakkeen otsake $h \in \mathbb{S}$ ja tulos h'

Tuloste: Käsitelty otsake h'

```
1 function ParseHeader(h, h′)
2     h′.select ← h[h.length] = "*"                          ▷ Projektio
3     T ← Split(h[1 : h.length − 1], ".")      ▷ [entiteetti, nimi AS alias]
4     h′.entity ← T[1]
5     T′ ← Split(T[2], "AS")                                 ▷ [nimi, alias]
6     h′.name ← T′[1]
7     h′.display ← T′[2] or T′[1]                ▷ Nimi on alias tai nimi
8     return h′
```

8.2 Rivit

Ominaisuussarakkeiden esimerkkiarvojen jäsennys on lähes samanlainen kaarille ja solmuille. Keskeinen osa on ominaisuuden arvon muunnos Java-tietotyypiksi ja tapa jäsentää rivit otsakkeiden kanssa. Funktio 10 käsittelee yksittäisen rivin sarake kerrallaan ja samalla luetaan otsakkeet, jolloin voidaan päätellä, onko sarake solmu vai kaari. Virheellinen syntaksi huomataan muun muassa silloin, kun solmuja tai kaaria ei löydy ominaisuussarakkeita jäsentäessä.

Funktiossa 10 otsakkeet H sisältävät alkioita, jotka on määritelty funktion 9 perusteella. Nämä kaikki otsakkeet iteroidaan, joten h_{prev} pitää kirjaa viimeisimmästä nimisarakkeen otsakkeesta. Apufunktiota IsNode arvioi otsakkeen ja sarakkeen arvon perusteella onko kyseessä solmu, samoin tekee myös apufunktio IsEdge. Apufunktiot ParseNode ja ParseEdge muodostavat otsakkeesta h sekä sarakkeen arvosta s solmun tai kaaren kyselygraafiin. Erityinen apufunktio FindNodeOrEdge palauttaa kyselygraafista parin $(v_q, null)$ tai $(null, e_q)$ riippuen siitä kumpaa tyyppiä nimi esittää.

Funktio 10 Rivin muuttaminen merkkijonosta graafin solmuksi tai kaareksi

Syöte: Kyselygraafi $Q = (V_Q, E_Q)$, taulukko otsakkeita H, rivi L taulukkona
Tuloste: Kyselygraafi Q

```
1  function ParseRow(Q, H, L)
2      h_prev ← null                          ▷ Edellisen nimisarakkeen otsake
3      for i ← 1 to H.length do
4          (h, s) ← (H[i], L[i])              ▷ Sarakkeen otsake h ja sen arvo s
5          if IsNode(h, s) then
6              h_prev ← h
7              v_q ← ParseNode(h, s)
8              Q ← (V_Q ∪ {v_q}, E_Q)
9          else if IsEdge(h, s) then
10             h_prev ← h
11             e_q ← ParseEdge(h, s)
12             Q ← (V_Q, E_Q ∪ {e_q})
13         else
14             (v_q, e_q) ← FindNodeOrEdge(Q, h_prev.entity)
15             (p, x) ← ParseProperty(Q, h, s)
16             if v_q ≠ null then
17                 v_q ← SetProperty(v_q, p, x)
18                 Q ← (V_Q ∪ {v_q}, E_Q)
19             else if e_q ≠ null then
20                 e_q ← SetProperty(e_q, p, x)
21                 Q ← (V_Q, E_Q ∪ {e_q})
22     return Q
```

8.3 Tuloksen palauttaminen

Tulosgraafi muunnetaan kyselyn lopuksi taulukoksi, joka voidaan edelleen tulostaa merkkijonona. Tarkastellaan ensin taulukoksi muuntamista, jossa lasketaan samalla sarakkeiden leveydet tulostusta varten. Tulosgraafin läpikäynti muistuttaa graafin syvyyshakua.

Funktio 11 esittää ratkaisua, jossa tulosgraafi muunnetaan solmu kerrallaan riviksi. Funktiossa 11 käytetään apufunktiota Max, joka valitsee kahdesta argumentista suuremman. Otsaketaulukko H sisältää aiemmin määriteltyjä otsakkeita, josta attribuutti *display* eli tulostaulussa näkyvä nimi on käytössä. Yhdellä solmulla voi

olla kaarien vuoksi useampi rivi, jos se esiintyy useamman kaaren päätesolmuna. Funktiot 12 ja 13 toimivat rekursiivisesti niin, että ne täyttävät rivien sarakkeet. Tulossolmuille v_r ja kaarille e_r määritellään attribuutti *visited*, jotta samoja solmuja ei tulosteta tarpeettomasti.

Funktio 11 Tulosgraafin R muuntaminen taulukoksi T

Syöte: Tulosgraafi $R = (V_R, E_R)$ ja sarakkeiden otsakkeet H taulukkona
Tuloste: Otsikkorivi H', tulosrivien joukko L ja sarakkeiden leveydet W

```
1  function ToTable(R, H)
2      W ← (0, ..., 0)                          ▷ Sarakkeiden leveydet
3      H' ← (ϵ, ..., ϵ)                        ▷ Otsikkorivi tyhjillä merkkijonoilla ϵ
4      for i ← 1 to H.length do
5          H'[i] ← H[i].display
6          W[i] ← Max(W[i], H[i].length)
7      L ← ∅                                    ▷ Tulostaulun rivien joukko
8      for all v_r ∈ V_R do                     ▷ R = (V_R, E_R)
9          if v_r.visited = false then
10             L' ← FindNodeRows(R, v_r, (ϵ, ..., ϵ), H, W)    ▷ ks. funktio 12
11             L ← L ∪ L'
12     return (H', T, W)
```

8.3.1 Rivin tietojen täyttö

Funktiot 12 ja 13 muistuttavat toisiaan ja funktionaalisen ohjelmoinnin funktio argumenteilla voidaan yleistää rivin täyttö. Ratkaisu on rekursiivinen, sillä molemmat funktiot kutsuvat toisiaan. Apufunktio FillProperties täyttää sarakkeiden tiedot rivipohjaan T. Otsakkeiden tietojen pohjalta täytetään sarakkeet jokaisen solmun tai kaaren kohdalla. Apufunktio Edges etsii kaikki kaaret, joissa solmu v_r on päätesolmuna.

Funktio 12 Solmun v muuntaminen riviksi

Syöte: Tulosgraafi $R = (V_R, E_R)$, tulossolmu v_r, rivipohja T, otsakkeet H, sarakkeiden leveydet W

Tuloste: Rivien joukko L

```
1  function FindNodeRows(R, v_r, T, H, W)
2      v_r.visited ← true
3      L ← ∅
4      T' ← FillProperties(v_r, T, H, W)        ▷ Täyttää rivipohjan sarakkeita
5      for all e_r ∈ Edges(R, v_r) do
6          L ← L ∪ FindEdgeRows(e_r, T')        ▷ ks. funktio 13
7      if L = ∅ then return {T'}
8      return L
```

Funktio 13 Kaaren e muuntaminen riviksi

Syöte: Tulosgraafi $R = (V_R, E_R)$, kaari e_r, rivipohja T, otsakkeet H, sarakkeiden leveydet W

Tuloste: Rivien joukko L

```
1  function FindEdgeRows(R, e_r, T, H, W)
2      e_r.visited ← true
3      L ← ∅
4      T' ← FillProperties(e_r, T, H, W)
5      v_1 ← Node(R, e_r.tail)
6      v_2 ← Node(R, e_r.head)
7      if v_1.visited = false then
8          L ← L ∪ FindNodeRows(v_1, T')        ▷ ks. funktio 12
9      if v_2.visited = false then
10         L ← L ∪ FindNodeRows(v_2, T')        ▷ ks. funktio 12
11     if L = ∅ then return {T'}
12     return L
```

8.3.2 Merkkijonoesitys

Kun taulun rakenne ja sarakkeiden leveydet ovat selvillä, niin se voidaan tulostaa merkkijonoksi. Prosessi on suoraviivainen ja vain erityistapaukset `null`-arvot ja merkkijonot on huomioitava. Numerot ja totuusarvot voidaan tulostaa sellaisenaan.

Funktio 14 näyttää yksittäisen rivin tulostuksen. Otsakkeet määriteltiin aikaisemmin kohdassa 8.1, joten attribuuttia *select* hyödynnetään määrittämään sarakkeen näkyvyys. Apufunktio IsString kertoo, onko muuttuja merkkijono. Kaikilla merkkijonoilla on samat operaatiot ja attribuutit kuin taulukoilla (array). Lisäksi merkkijonoille määritellään +-operaatio, joka yhdistää kaksi merkkijonoa yhdeksi. Max valitsee kahdesta arvosta suuremman ja Repeat muodostaa merkkijonon, jossa toistetaan samaa merkkiä n kertaa. Merkkijonolla `\n` ilmaistaan rivinvaihtoa.

Funktio 14 Rivin tulostaminen merkkijonona

Syöte: Tulosrivi L, sarakkeiden leveydet W, otsakkeet H

Tuloste: Merkkijono s

```
1  function RowToString(L, W, H)
2      s ← ϵ                                  ▷ Alustetaan tyhjällä merkkijonolla ϵ
3      for i ← 1 to L.length do
4          if H[i].select = true then
5              x ← L[i]                        ▷ Luetaan sarakkeen arvo
6              if IsString(x) then
7                  x ← "" + s + ""             ▷ Lisätään lainausmerkit
8              n ← Max(1, W[i] − x.length + 1) ▷ Tyhjien välien lukumäärä
9              s ← "|  " + s + Repeat(" ", n)
10     return s + "|\n"
```

9 Lisäominaisuudet

Tässä luvussa tarkastellaan kyselykieleen laajennettavia ominaisuuksia. Näitä ei ole otettu mukaan lopulliseen versioon joko monimutkaisuuden vuoksi tai ne eivät ole suoria tiedonhakuoperaatioita. Nämä ovat syntaksiltaan kuitenkin helposti laajennettavissa, joten ideat kootaan tähän lukuun.

9.1 Nimettömät solmut ja kaaret

Esimerkeissä 10 ja 11 solmuilla ja kaarilla on nimet, mutta ne voivat myös puuttua tai solmuja haetaan vain ominaisuuksien perusteella. Tällaisista kyselyistä on etua, jos esimerkiksi kaikilla solmuilla tai kaarilla on olemassa samoja ominaisuuksia kuten luontiaika. Tulosten hyödyntäminen voi olla näillä tapauksissa hankala käyttää, sillä käyttäjä ei tiedä mihin solmuun tai kaareen ne liittyvät. Näistä on kuitenkin hyötyä

rajauksessa, voidaan esimerkiksi hakea kaikki viimeisellä myyntipäivällä merkityt solmut.

Esimerkki 20. Nimettömät solmut merkitään taulukkoon merkillä `_`. Seuraava taulukko hakisi kaikki solmut, joilla on ominaisuuden `price` arvo on yli 50

_	id*	price*
QUERY		>= 50.00

Äärimmäisissä tapauksissa kaarilla ei ole tietoa nimistä

_	id*	timestamp
QUERY _._		>= 3600

9.2 Indeksoidut nimet

Luuppien ja silmukoiden kanssa tulee vastaan ongelma, jossa polku sisältää kaksi samannimistä solmua, mutta näillä on eri rajoitteet. Tällaisissa tilanteissa tarvitsee erottaa kaksi solmua. Tämä osoittautui liian hankalaksi toteuttaa valittuun graafitietorakenteeseen, joten ideaa esitellään jatkotutkimuksia varten.

Havaittuja ongelmia ovat muun muassa se kuinka tulosgraafiin merkitään indeksit, sillä sama solmu voi täyttää useamman solmun ehdot. Graafin tietorakenne on tässä tutkielmassa määritelty yksilöimään solmut ja kaaret id-tunnisteella, mutta nämä pitäisi esittää useampaan kertaa. Jos luuppia

$$(Course_1, recommends, Course_1)$$

haetaan, niin tulosgraafissa voi olla jokin toinen solmu, jolla on sama indeksi 1 ja tällöin tulos voi sisältää ylimääräisiä kaaria.

Jos taulukossa esiintyy useampi samanniminen solmu, niin ne voidaan merkitä indeksien kanssa. Esimerkiksi kaari $(Course_1, recommends, Course_2)$, jossa lähtösolmu- ja maalisolmu ovat eri merkitään taulukkoon muodossa

Course:1	Course:2	recommends
QUERY	QUERY	QUERY Course:1.Course:2

Samalla tavalla merkitään myös useammat samannimiset kaaret, eli nimisarakkeen otsakkeen perässä on ***`:indeksi`***. Ominaisuussarakkeiden otsakkeiden alkuun merkitään myös indeksit, jos niitä ei voida päätellä. Indeksointi jätetään merkitsemättä, jos päätesolmut ovat samat. Samanlaista indeksimerkintää käytetään myös GQBE-kielessä [Jabocs ja Walczak, 1983], kun samannimisiä taulukoita on sisäkkäin.

9.3 Rivien järjestäminen

Taulukoissa voi olla tarpeen järjestellä rivejä, jotta ne voidaan näyttää käyttäjälle selkeämmin. Tämä on kuitenkin taulukko-operaatio, joten tässä tutkielmassa ei ote sitä huomioon. Graafia ei voida järjestää, sillä se ei ole suoraan listattavissa. Rivien järjestämisessä voidaan käyttää SQL-kielen avainsanoja `ASC` ja `DESC` [PostgreSQL 14, 2021, s. 130]

```
| Book  | title* ASC |
|-------+------------|.
| QUERY |            |
```

9.4 Transitiiviset kaaret

Transitiivisilla kaarilla on olemassa tuntematon (mikä tahansa) polku päätesolmujen välillä. Tällöin voidaan käyttää merkintää

```
| kaari                           | ominaisuus |
|---------------------------------+------------|.
| QUERY Lähtösolmu._.Maalisolmu |            |
```

Ongelma on kuitenkin siinä, että ei tiedetä mille polun kaarelle määritellyt ominaisuudet kuuluvat. Ei voida myöskään määritellä millaisia solmuja kuuluu polkuun. Nämä voidaan kuitenkin rajata tarkemmin kahdella kaarella

```
| e1             | e2             |
|----------------+----------------|.
| QUERY v1._.v2 | QUERY v2._.v3 |
```

9.5 Datan manipulointi

Edelliset luvut käsittelevät tietojen hakemista graafitietokannasta, aihetta sivuten esitellään lyhyesti syntaksi tietojen lisäämiselle, päivittämiselle ja poistamiselle. Nämä operaatiot ovat myös taulukkomuotoista ja laajennettu kyselygraafiin, jolloin sama graafi voi sisältää myös rajauksia.

Usean rivin päivitys- ja lisäysoperaatioissa tulee huomioida tietokantajärjestelmien peruspiirre *atomisuus (atomicity)*, joka varmistaa sen, että kaikki muutokset tallentuvat tai mitään niistä ei tallenneta [Haerder ja Reuter, 1983]. Erillisten kyselygraafien prosessoinnissa on siis huomioitava lopullinen tallennus vasta kun kaikki on käsitelty. Tämä voidaan ratkaista *transaktioiden (transaction)* avulla, jossa perusoperaatiot mahdollistavat muutosten viimeistelyn tai niiden perumisen [Haerder ja Reuter, 1983].

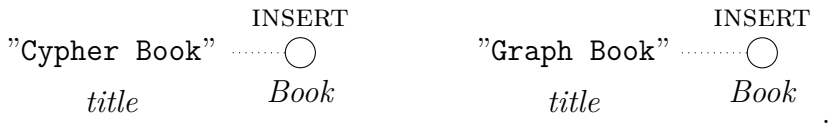
9.5.1 Lisäys

Tietojen lisääminen määritetään avainsanalla `INSERT` [Zloof, 1975a]. Se luo aina uuden solmun tai kaaren. Kaarien lisäämisessä solmut on oltava ensin lisätty tietokantaan.

Esimerkki 21 (Solmun lisääminen)**.** Solmu `Book` lisätään tietokantaan taulukolla

```
| Book   | title          |
|--------+----------------|
| INSERT | "Cypher Book"  |
| INSERT | "Graph Book"   |
```

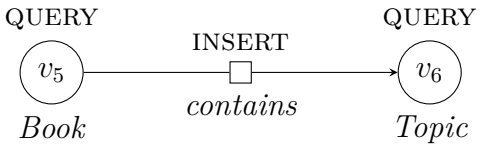
Kyselygraafeja muodostuu kaksi kappaletta



Esimerkki 22 (Kaaren lisääminen)**.** Oletetaan, että solmut `Book` ja `Topic` on lisätty tietokantaan. Kaaren lisääminen näiden välille on

```
| contains           | Book.id | Topic.id |
|--------------------+---------+----------|
| INSERT Book.Topic  | 5       | 6        |
```

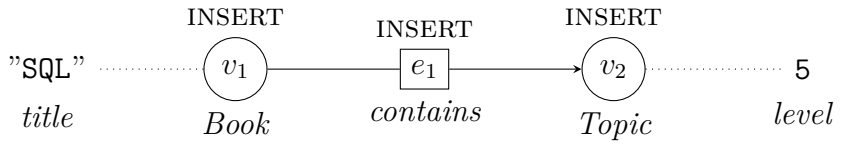
Kyselygraafi on vastaavasti



Esimerkki 23 (Yhdistetty lisäys)**.** Solmut `Book` ja `Topic` lisätään samaan aikaan kaaren `contains` kanssa

```
| Book   | title | contains           | Topic  | level |
|--------+-------+--------------------+--------+-------|
| INSERT | "SQL" | INSERT Book.Topic  | INSERT | 5     |
```

Useampiakin solmuja ja kaaria voidaan lisätä tällä tavalla, mutta taulukon pituus voi vaikeuttaa sarakkeiden seuraamista.



Esimerkistä 23 huomataan, että näin voidaan merkitä kaaren lisäys kaikille solmuille, jotka löytyvät hakuehdoilla

```
| Book  | title    | contains           | Topic | level |
|-------+----------+--------------------+-------+-------|.
| QUERY | /Math.*/ | INSERT Book.Topic | QUERY | 5     |
```

9.5.2 Päivitys

Olemassa olevien tietojen päivittäminen on tarpeen tiedon muuttuessa. Tarkastellaan kahdentyyppistä tiedon päivittämistä: (1) tiedon haku päivitettävän sarakkeen arvolla ja (2) päivitys id-tunnisteen avulla. Yksittäisten kenttien päivittäminen tapahtuu niin, että ominaisuussarakkeeseen kirjoitetaan avainsana

`UPDATE` ***arvo***.

Jos saraketta käytetään myös hakuehtona, niin se voidaan merkitä

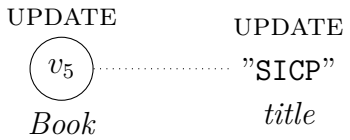
`UPDATE` ***vanha*** `[ TO` ***uusi*** `]`.

Nimisarakkeeseen kirjoitetaan myös sana `UPDATE` [Zloof, 1975a]. Alkuperäisessä QBE -kielessä käytetään kaksi riviä, joista toinen toimii hakuehtona Zloof [1975a].

Esimerkki 24 (Yksinkertainen päivitys)**.** Päivitysoperaatioita voidaan soveltaa myös useammalle solmulle, kuten myös poistaminen vastaavalla tavalla.

```
| Book   | id | title             |
|--------+----+-------------------|
| UPDATE | 5  | UPDATE "SICP"     |
```

Kysely hakee tietokannasta id-tunnisteella solmun `Book` ja päivittää sille nimikkeen.



Esimerkki 25 (Tietojen massapäivitys)**.** Joskus hakuehtojen yhdistäminen ja päivittäminen on tarpeen. Tällöin kentät, joilla ei ole avainsanaa `UPDATE` käytetään hakuehtoina

```
| Book   | year   | price         |
|--------+--------+---------------|.
| UPDATE | < 1990 | UPDATE 10.00 |
```

Kysely laittaa kaikki kirjat, jotka on julkaistu ennen vuotta 1990 alennusmyyntiin kiinteään hintaan

9.5.3 Poisto

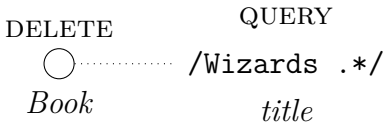
Tietoja poistetaan avainsanalla `DELETE`. Poistaminen muistuttaa alkuperäistä QBE -kieltä [Zloof, 1975a]. Solmujen poistamisessa poistetaan myös kaikki siihen liittyvät kaaret. Yksittäisiä ominaisuuksia voisi myös poistaa samalla avainsanalla merkitsemällä sarakkeen kohdalle, jota poistetaan. Voi kuitenkin olla, että `UPDATE NULL` olisi parempi estämään vahingossa poistamista.

Lisävarmistus kaarille voi olla tarpeen, jos kaari halutaan poistettavan ennen solmun poistoa. Tällaisia tilanteita voi olla, jos ominaisuuden käyttö on integroitu ulkoiseen palveluun.

Esimerkki 26 (Poistaminen)**.** Kokonaiset solmut tai kaaret voidaan poistaa merkitsemällä nimisarakkeen kohdalle `DELETE`. Kuten päivitysoperaatioissakin, myös tässä voidaan poistaa id-tunnisteella tai massapoistona

```
| Book   | title          |
|--------+----------------|.
| DELETE | /Wizards .*/ |
```

Kysely poistaa kaikki solmut, joiden nimi alkaa sanalla `Wizards`



10 Vertailu muihin QBE-kyselykieliin

Tässä luvussa vertaillaan määriteltyä taulukkomuotoista graafikyselykieltä alkuperäiseen QBE-kieleen ja läheiseen XQBE-kieleen. Vertailtavia piirteitä ovat tuki kyselykielen ominaisuuksille ja graafin esittäminen kyseisillä kielillä. Monet kyselykieliin liittyvät tutkimukset kuten [Braga *et al.*, 2005; Shelly *et al.*, 2012] taulukoivat kyselykielten ominaisuuksia ja vertailevat niitä. Tässä luvussa vertaillaan samalla tavalla.

Huomionarvoista on kuitenkin, että muut kielet eivät ole suunniteltu graafitietokantoja varten. Tässä luvussa tutkielman kyselykieleen viitataan nimellä *TGQBE (Tabular Graph QBE)*.

Taulukko 3 vertailee tuettuja ominaisuuksia alkuperäiseen [Zloof, 1975b] QBE -kieleen ja visuaaliseen [Braga *et al.*, 2005] XQBE-kieleen. Tähdellä * merkitään lisäominaisuudet, joita ei ole toteutettu, mutta toteutettavissa määritetyillä merkinnöillä.

Ominaisuus	**TGQBE**	**QBE**	**XQBE**
Kieli taustalla		SQL	XQuery
Projektio	×	×	×
Aliakset	×		×
Liitokset	×	×	×
Transitiiviset suhteet	×*		×
Vertailuoperaattorit arvoille	×	×	×
Loogiset operaattorit arvoille	×	IBM QBE	×
Koontioperaatiot	×	×	×
Kootut arvot hakuehtoina		×	×
Aritmeettiset operaatiot			×
Lasketut attribuutit			×
Tulosten järjestäminen	×*	IBM QBE	×
Alkioiden lisäys tulokseen			×
Nimettömät entiteetit	×*		
Tietojen päivittäminen	×*	×	

Taulukko 3: Vertailu QBE-kielten kyselyominaisuuksista.

XQBE-kyselykielessä ei ole määritelty tietojen päivitysoperaatioita [Braga *et al.*, 2005], mutta QBE-kielessä on [Zloof, 1975a]. Edistyneitä kyselyominaisuuksia kuten käyttäjän määrittämiä funktioita ei tue mikään näistä. Braga ja muut [2005] toteavat, että käyttäjät, jotka niitä tarvitsisivat, pystyisivät hyödyntämään XQuerykieltä suoraan. Jabocs ja Walczak [1983] käsittelee *konjunktiivisia (conjunctive)* ja *disjunktiivisia (disjunctive)* kyselyitä GQBE-kielessä käyttäen useita tauluja. Konjunktiivinen tarkoittaa, että kyselyn kaikki ehdot täyttyvät. Disjunktiivisessa kyselyssä riittää, että täyttää jotkin ehdoista. QBE käyttää usean rivin kyselyjä, joissa projektion avainasanan `P.` käyttö kertoo, kumpi operaatio on kyseessä [Zloof, 1981]. Jos vain ensimmäisellä rivillä on `P.`, niin kysely on konjunktiivinen ja muulloin disjunktiivinen [Zloof, 1981]. TGQBE-kielessä vain disjunktiivinen kysely huomioidaan

usean rivin kyselyissä, sillä konjunktiiviset ehdot on sulautettu loogisiin ja säännöllisiin lausekkeisiin. XQBE ei mahdollista tällaisia kyselyitä XML-puille, mutta Braga ja muut [2005] kertovat sen olevan tarkoitus, sillä niiden monimutkaisuus hankaloittaisi käyttöä.

Selkeimmät erot XQBE-kieleen ovat alkioiden luonti tulokseen eli XML-puun elementtejä, jotka eivät esiintyisi muulloin. QBE-kielessä tätä ei ole määritelty eli tulokseen ei voi määritellä ylimääräisiä sarakkeita. Tähän piirteeseen liittyy myös lasketut arvot ja koontioperaatiot. XQBE eroaa merkittävästi myös sen visuaalisen luonteen vuoksi.

Käyttötarkoituksensa vuoksi XQBE on loistava puukyselyihin, mutta graafin kanssa on rajoitteita muun muassa silmukoiden ja luuppien kanssa. XQBE-kieltä voidaan kuitenkin hyödyntää puiden etsimiseksi graafista. QBE-kieltä ei ole suunniteltu graafeille tai puille, mutta sen yleistys GQBE pystyy esittämään puita sisäkkäisten taulukoiden avulla [Jabocs ja Walczak, 1983] kuten myös QBEN [Lorentzos ja Dondis, 2006], jolloin sillä voidaan myös hakea puita graafitietokannoista. Alkuperäinen QBE-kieli pystyisi ilmaisemaan korkeintaan yhden suoran polun. XQBE-kieli on siis vertailun lopputuloksena ilmaisuvoimaisin, jos graafikyselyjä ei oteta huomioon. Tilanne voisi olla kuitenkin erilainen, jos graafikyselykieli olisi toteutettu XQBE:n tavoin visuaalisesti. Taulukko 4 vertailee lopuksi mahdollisesta tuesta graafikyselyn esittämiseksi eli nämä ovat vain pohdintaa.

Ominaisuus	**TGQBE**	**QBE**	**XQBE**
Solmut	×	×	×
Kaaret	×	×	×
Luupit	×	×	
Polut	×	×	×
Silmukat	×	×	
Puut	×	GQBE ja QBEN	×
Yhtenäiset graafit	×		
Epäyhtenäiset graafit	×		

Taulukko 4: Pohdinta tuesta graafeille.

11 Yhteenveto

Tutkielmassa tarkasteltiin QBE-taulukkomuotoista kyselyä ominaisuusgraafitietokantaan. Tavoitteena oli selvittää QBE-kyselykielen soveltuvuus graafitietokantoihin ja aihetta lähestyttiin kyselygraafien kautta. Kyselygraafin määritteleminen

mahdollistaa eri kyselykielten syntaksit samalle kyselygraafille. Ongelmana on kuitenkin se, että kyselykieli ei voi olla ilmaisuvoimaisempi kuin kyselygraafi, sillä se ei mahdollista sitä edistyneempiä tapauksia. Tutkielmassa formalisoitiin kyselygraafi ja tulosgraafi sekä toteutettiin prototyyppi kyselyjen suorittamiseksi. Toteutus kysely- ja tulosgraafiin perustuen mahdollistaa muun muassa visuaalisen kyselykielen toteutuksen, joka voisi muistuttaa Cruzin ja muiden [1987] määrittelemää visuaalista graafikyselykieltä, mutta sovellettuna ominaisuusgraafeihin. Verkkosivutoteutukset taulukkomuotoisesta QBE-kyselykielestä hakutoimintona on myös potentiaalinen tutkimusaihe käyttöliittymien osalta, sillä vastaavaa on aiemmin kokeiltu AQBE-kielellä terveydenhuolto-ohjelmistoihin [Shelly *et al.*, 2012].

Työn tuloksista voidaan tehdä johtopäätös, että QBE-kielestä laajennettu taulukkoesitys kykenee esittämään graafin. Yleisesti voidaan huomata, että tuettavat toiminnot ovat rajatummat kuin avainsanoihin pohjautuvat kyselykielet. Tämä voi olla kuitenkin hyväksyttävä puute, sillä QBE-kyselykielellä on selkeitä etuja rakenteen rajaamisessa, joka helpottaa oppimista kuten SQL ja QBE-kielten kanssa on todettu [Yen ja Scamell, 1993]. Muita lähestymistapoja voisi myös vertailla, esimerkiksi se onko helpompi käyttää erillisiä tauluja liitoksissa vai yhtä graafien esittämiseksi. Erillisiä tauluja voidaan myös tutkia epäyhtenäisen graafin muodostamiseen. Lisää selvitystä vaativat myös vertailut visuaalisiin graafikyselykieliin kuten [Cruz *et al.*, 1987].

Määritellystä kyselykielen syntaksista huomataan myös, että sen ei tarvitse merkittävästi erota alkuperäisestä QBE-kielestä. Suurimmat erot liittyvät kaarien merkitsemiseen. Esille nousivat muutkin QBE-kielet kuten XQBE, AQBE, QBEN ja GQBE, näitä kieliä voisi myös tutkia tarkemmin graafitietokannan kyselykielinä. Esimerkiksi GQBE ja QBEN -kielten osalta voitaisiin selvittää sisäkkäisten taulujen hyödyntämistä, tässä tutkielmassa käsiteltiin vain tavallista taulukkoa.

Tehokkuuden arviointi ja algoritmien optimointi puuttuu, mikä voisi olla jatkotutkimuksen aihe. Esimerkiksi graafin läpikäynti on toteutettu rekursiivisesti, mikä voi olla ongelmallinen isoilla graafitietokannoilla muistirajoitteiden vuoksi. Prototyyppi toteutettiin Neo4j-tietokannalle, mutta myös muillekin voisi kehittää kuten NebulaGraph ja JanusGraph. Edistyneempiä graafioperaatioita kuten *graafin erotusta* ja *leikkausta* tai algoritmien tuloksilla rajaamista ei tarkasteltu. Yksinkertaisempiakaan rajauksia kuten *solmun asteella* eli kaarien lukumäärällä rajaaminen ei huomioitu, joten nämä ovat mahdollisia jatkotutkimuksen aiheita.

Prototyyppi on toteutukseltaan hieman puutteellinen, sillä se ei pysty ilmaisemaan hakuja, joissa ainoastaan luupit sisällytetään. Idea esitettiin kohdassa 9.2, mutta se ei kuitenkaan ole helposti toteutettavissa ja vaatisi lisää tutkimusta siitä

kuinka määriteltyä rakennetta voitaisiin muuttaa tämän tukemiseksi. Transitiiviset kaaret sekä nimettömät solmut jäivät myös tarkastelematta. Prototyyppi ei myöskään karsi riittävän tehokkaasti solmuja tulosgraafista. Tämä korostuu koontioperaatioissa, jossa ylimääräisiä rivejä saattaa esiintyä lopputuloksessa.

Tutkielmaa tehdessä tuli esille myös GraphML-syntaksi, joka on XML-muotoinen tapa esittää graafi. Tämä voisi olla potentiaalinen QBE-sovellus eli kyselygraafi esitettäisiinkin XML-muodossa ja tulos palautetaan HTML-muodossa. Tällöin voisi olla mahdollista kehittää visuaalinen sisällönhallintajärjestelmä graafitietokannan päälle.

12 Viitteet

Angles, Renzo. 2012. A comparison of current graph database models. In: *IEEE* 28^{th} *International Conference on Data Engineering Workshops*, 171–177.

Angles, Renzo, Marcelo Arenas, Pablo Barceló, Aidan Hogan and Domagoj Vroč. 2017. Foundations of modern query languages for graph databases. *ACM Computing Surveys* 50, 5, 1–40.

Braga, Daniele, Alessandro Campi and Stefano Ceri. 2005. XQBE (XQuery by Example): A visual interface to the standard XML query language. *ACM Transactions on Database Systems* 30, 2, 398–443.

Codd, Edgar F. 1970. A relational model of data for large shared data banks. *Communications of the ACM* 13, 6, 377–387.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. 2009. *Introduction to Algorithms* 3^{rd} *Edition*. The MIT Press.

Cruz, Isabel F., Alberto O. Mendelzon and Peter T. Wood. 1987. A graphical query language supporting recursion. *ACM SIGMOD Record* 16, 3, 323–330.

Diestel, Reinhard. 2016. *Graph Theory* 5^{th} *Electronic Edition*. Springer.

Ebbinghaus, Heinz-Dieter, Jörg Flum and Wolfgang Thomas. 1994. *Mathematical Logic* 2^{nd} *Edition*. Springer New York.

Foulds, L. R. 1992. *Graph Theory Applications*. Springer New York.

GraphQL. 2022. `https://graphql.org/learn/queries/`. Viitattu 10.1.2022.

ISO Graph Query Language Proponents. 2022. Existing Languages. `https://www.gqlstandards.org/existing-languages`. Viitattu 19.3.2022.

Haerder, Theo and Andreas Reuter. 1983. Principles of transaction-oriented database recovery. *ACM Computing Surveys* 15, 4, 287–317.

Hua, Yingbing and Bjorn Hein. 2021. AQT - A query template for AutomationML. *IEEE Transactions on Industrial Informatics* 17, 2, 1018–1028.

Jacobs, Barry E. and Cynthia A. Walczak. 1983. A generalized query-by-example data manipulation language based on database logic. *IEEE Transactions on Software Engineering SE*-9, 1, 40–57.

Jayaram, Nandish, Mahesh Gupta, Arijit Khan, Chengkai Li, Xifeng Yan and Ramez Elmasri. 2014. GQBE: Querying knowledge graphs by example entity tuples. In: *IEEE 30th International Conference on Data Engineering*, 1250–1253.

Jin, Changjiu, Sourav S. Bhowmick, Xiaokui Xiao, James Cheng and Byron Choi. 2010. GBLENDER: Towards blending visual query formulation and query processing in graph databases. In: *Proceedings of the* 2010 *ACM SIGMOD International Conference on Management of Data*, 111–122.

Krtolica, Predrag V. and Predrag S. Stanimirović. 2004. Reverse polish notation method. *International Journal of Computer Mathematics 81*, 3, 273–284.

Libkin, Leonid, Wim Martens and Domagoj Vrgoč. 2013. Querying graph databases with XPath. In: *Proceedings of the* 16^{th} *International Conference on Database Theory*, 129–140.

Lorentzos, Nikos A., and Konstantinos A. Dondis. 2006. Query by example for nested tables. *Database and Expert Systems Applications*, 1460, 716–725.

Meduna, Alexander. 2014. *Formal Languages and Computation.* Auerbach Publications.

Neo4j. 2022. The Neo4j Cypher Manual v4.4. Saatavilla: `https://neo4j.com/docs/pdf/neo4j-cypher-manual-4.4.pdf`.

The PostgreSQL Global Development Group. 2021. *PostgreSQL* 14.1 *Documentation.* Saatavilla: `https://www.postgresql.org/files/documentation/pdf/14/postgresql-14-US.pdf`.

Rosetta Code. 2022. Parsing/RPN calculator algorithm. `https://rosettacode.org/wiki/Parsing/RPN_calculator_algorithm#Java_2`. Viitattu 25.1.2022.

Sachdeva, Shelly, Daigo Yaginuma, Wanming Chu and Subhash Bhalla. 2012. AQBE - QBE style queries for archetyped data. *IEICE Transactions on Information and Systems E95.D*, 3, 861–871.

Svensk, Sami-Santeri. 2021. *XPathin semantiikka graafitietokannoissa Cypher -kyselykielelle käännettynä.* Pro gradu -tutkielma. Informaatioteknologian ja viestinnän tiedekunta, Tampereen yliopisto.

Thomas, John C. and John D. Gould. 1975. A psychological study of query by example. In: *Proceedings of the May 19-22, National Computer Conference and Exposition (AFIPS '75)*, 439–445.

Yen, Minnie Y.M. and Richard W. Scamell. 1993. A human factors experimental comparison of SQL and QBE. *IEEE Transactions on Software Engineering* 19, 390–409.

Zloof, M. Moshé. 1975a. Query-by-example: the invocation and definition of tables and forms. In: *Proceedings of the 1st International Conference on Very Large Data Bases*, 1–24.

Zloof, M. Moshé. 1975b. Query by example. In: *Proceedings of the May* 19–22*, National Computer Conference and Exposition (AFIPS '75)*, 431–438.

Zloof, M. Moshé. 1981. QBE/OBE: A language for office and business automation. *Computer* 14, 13–22.