Tampereen yliopisto

Katrin Dieter

GRAAFITIETOKANTOJEN KYSELYKIELET JA GQL-STANDARDI

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Huhtikuu 2023

Tiivistelmä

Katrin Dieter: Graafitietokantojen kyselykielet ja GQL-standardi
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Huhtikuu 2023

Graafitietokantojen kyselykielet ovat kieliä, joilla voidaan suorittaa kyselyitä graafitietokantoihin. Graafitietokantojen kyselykieliä on vertailtu toisiinsa aiemminkin, mutta vertailu on toteutettu tyypillisesti vain muutamalle eri kyselykielelle. Tässä tutkielmassa vertaillaan kuutta eri kyselykieltä. Tämän lisäksi kyselykieliä vertaillaan arvion mukaan vuonna 2024 julkaistavaan ominaisuusgraafien GQL-kyselykielistandardiin. Tavoitteena on saada selville, millaisia graafitietokantojen kyselykieliä on olemassa sekä mitä eroja ja yhtäläisyyksiä on eri kyselykielten välillä. Lisäksi halutaan selvittää, millainen tuleva kyselykielistandardi GQL on ja kuinka nykyiset kyselykielet eroavat siitä ominaisuuksiltaan ja syntaksiltaan.

Tutkielmassa vertailtavat kyselykielet ovat Cypher, PGQL, GSQL, Gremlin, SPARQL ja G-CORE. Tietoa kyselykielten ominaisuuksista ja syntaksista haettiin jokaisen kyselykielen virallisesta dokumentaatiosivustosta tai kyselykielestä kirjoitetuista tieteellisistä artikkeleista. Koska aineisto koostui pääosin dokumentaatioista, määritettiin tutkielman menetelmäksi dokumentaatioon perustuva katsaus. Tutkielmassa on yhteisiä piirteitä systemaattisen kirjallisuuskatsauksen kanssa.

Tutkielman tuloksena saatiin selville, että graafitietokantojen kyselykielten ominaisuuksissa ja syntakseissa on eroja, mutta samat ydinominaisuudet löytyvät niistä kaikista. Kyselykielistandardi GQL:n ominaisuuksissa ja syntaksissa on vaikutteita jo olemassa olevista kyselykielistä, mutta se tulee sisältämään myös uusia ja uniikkeja ominaisuuksia.

Avainsanat: graafi, graafitietokanta, kyselykieli, kyselykielistandardi, GQL

SISÄLLYS

1 JOHDANTO 1

2 TUTKIMUSMENETELMÄT 4

2.1 Dokumentaatioon perustuva katsaus 4
2.2 Tutkimusaineiston hankinta ja valintakriteerit 5
2.3 Hakutulokset 6
2.4 Aineiston analyysi 7

3 GRAAFI TIETORAKENTEENA 8

3.1 Graafi 8
3.2 RDF-graafi 9
3.3 Heterogeeninen graafi 10
3.4 Ominaisuusgraafi 10
3.5 Aligraafi 11

4 KYSELYT GRAAFITIETOKANNASSA 13

4.1 Kulkeminen graafissa 13
4.2 Graafimallit 13
4.3 Hahmonsovitus 14
4.4 Polkukyselyt 15
4.5 Paluuarvot 16
4.6 Graafialgoritmit 17

5 GRAAFITIETOKANTOJEN KYSELYKIELET 19

5.1 Cypher 19
5.2 PGQL 23
5.3 GSQL 26
5.4 Gremlin 31
5.5 SPARQL 34
5.6 G-CORE 38

6 GQL JA SQL/PGQ 42

6.1 Syntaksi ja ominaisuudet 42
6.2 Aikataulu 46

7 KYSELYKIELTEN JA GQL-STANDARDIN VERTAILU 47

7.1 Kyselykielten vertailu 49
7.2 Polkukyselyt eri kyselykielillä 50

7.3 GQL JA KYSELYKIELET 52
7.4 POLKUKYSELYT GQL:LLÄ 53
8 KESKUSTELU 55
9 YHTEENVETO 57
LÄHTEET 58

1 Johdanto

Graafien prosessointi on noussut keskeiseen rooliin yhteiskunnassamme. Tämä johtuu siitä, että linkitetyn datan määrä on kasvanut ennätyksellistä vauhtia. Monilla yhteiskunnallisesti kiinnostavilla alueilla voidaan jo nähdä suuria graafisia prosessointijärjestelmiä, jotka tukevat monia nousevia, mutta jo monimutkaisia ja monimuotoisia tiedonhallintaekosysteemejä. Tämän lisäksi tekoälyn ja koneoppimisen saralla graafiprosessointi on noussut valtavaan suosioon. [Sakr *et al.* 2021]

Graafitietokannat ovat tehokkaita vahvasti linkitetyn datan säilömisessä ja analysoinnissa, ja ne ovat nostaneet suosiotaan etenkin liikenneverkostojen, bioinformaation, kemian ja tähtitieteen järjestelmissä. Koska graafitietokannoilla ei vielä ole kyselykielistandardia, on eri järjestelmille olemassa omat kyselykielensä. [Sharma *et al.* 2021]

Kaksi suosituinta graafien esitystapaa ovat RDF-graafit ja ominaisuusgraafit. RDF-graafeissa data mallinnetaan suunnattuun graafiin, jossa kaarilla on nimikkeet. Ominaisuusgraafeissa data puolestaan mallinnetaan osittain suunnattuun multigraafiin, jossa sekä solmuilla että kaarilla voi nimikkeiden lisäksi olla ominaisuuksia. [Deutsch *et al.* 2021] Tässä tutkielmassa keskitytään enemmän ominaisuusgraafeihin, sillä toisin kuin RDF-graafeille, ominaisuusgraafeille ei ole vielä kyselykielistandardia. Tutkielmassa sivutaan kuitenkin myös RDF-graafeja, niiden kyselykielistandardi SPARQL:n osalta.

Ominaisuusgraafijärjestelmät sisältävät toisistaan eroavia muistimalleja ja kyselyominaisuuksia joko ohjelmointirajapinnan tai deklaratiivisen graafikyselykielen muodossa. Jo olemassa olevat graafitietokantojen kyselykielet vastaavat pääpiirteittäin toisiaan etenkin ydinominaisuuksien osalta. [Deutsch *et al.* 2021] Kyselykielten välillä on kuitenkin eroja niissä käytettävien semantiikkojen ja polkuilmaisujen välillä [Angles *et al.* 2018].

Koska graafitietokantojen ja ominaisuusgraafien käyttö on viime vuosina yleistynyt, nousi myös tarve kyselykielistandardille. Vuonna 2019 ISO/IEC tekninen komitea hyväksyi hankkeen, jossa luodaan kyselykielistandardi ominaisuusgraafeille. Tämän lisäksi vuodesta 2017 asti on ollut kehitteillä SQL:n laajennus SQL/PGQ, jonka avulla voidaan tehdä kyselyitä graafinäkymiin SQL-kyselyiden kautta. [Deutsch *et al.* 2021]

Graafitietokantojen kyselykieliä vertailevia tutkimuksia on julkaistu, mutta näissä vertaillaan tyypillisesti kahta tai kolmea kyselykieltä keskenään. Esimerkiksi Anglesin ja muiden artikkelissa vuodelta 2017 vertaillaan kolmea kyselykieltä (Cypher, Gremlin ja SPARQL), kuten myös Sharman ja muiden artikkelissa vuodelta 2021 (Cypher, PGQL ja SPARQL). Tässä tutkielmassa haluttiin ottaa vertailuun useampia kyselykieliä. Tämän lisäksi vertailuun on otettu mukaan myös yksi kyselykieliehdotelma, tuleva kyselykielistandardi ominaisuusgraafeille sekä RDF-graafien kyselykielistandardi.

Tässä tutkielmassa haetaan vastauksia seuraaviin kysymyksiin:

1. Millaisia graafitietokantojen kyselykieliä on olemassa?
2. Kuinka graafitietokantojen kyselykielet eroavat toisistaan?
3. Kuinka graafitietokantojen kyselykielet eroavat tämänhetkisen tiedon valossa tulevasta GQL-standardista?

Ensimmäiseen kysymykseen pyritään hakemaan vastauksia tarkastelemalla kyselykielten syntaktisia ilmauksia, selvittämällä kuinka graafissa navigoidaan ja mitä eri semantiikkoja kyselykieli noudattaa. Toiseen kysymykseen haetaan vastauksia vertailemalla kyselykielten eroja paluuarvojen, semantiikkojen sekä syntaksin osalta. Vertailun kohteena on lisäksi kahdensuuntaiset polkukyselyt, UNION-operaatio, CRUD-toiminnot sekä useamman graafin kyselyt. Kolmanteen kysymykseen pyritään vastaamaan vertailemalla GQL-standardia, joka on kehitteillä oleva kyselykielistandardi ominaisuusgraafeille, muihin kyselykieliin. GQL-standardia ja muita kyselykieliä vertaillaan toisiinsa samojen ominaisuuksien perusteella kuin toisen tutkimuskysymyksen kohdalla vertailtiin kyselykieliä. Kaikkiin kysymyksiin pyritään vastaamaan kyselykielten dokumentaatioihin perustuvan katsauksen avulla.

Tutkielmassa käytettäviä tutkimusmenetelmiä avataan tarkemmin luvussa 2, jossa kerrotaan myös tiedonhaun prosessista sekä siitä, kuinka tutkielmassa käytettävät aineistot on valittu. Luvussa 3 käydään läpi graafien teoriaa sekä kuvataan millaisia graafit ovat tietorakenteena. Luku 4 käsittelee graafitietokantoihin suoritettavien kyselyiden teoriaa ja sitä, minkä kaltaisia rajauksia kyselyissä ja niiden tulosten käsittelyissä voidaan käyttää. Tutkielmaan valitut kyselykielet ja niiden ominaisuudet esitellään luvussa 5, jossa pyritään myös vastaamaan ensimmäiseen tutkimuskysymykseen. Luvussa 6 kerrotaan tulevien GQL- ja SQL/PGQ-standardien jo

julkaistuista ominaisuuksista sekä tulevan julkaisun odotetusta ajankohdasta. Luvussa 7 vertaillaan luvussa 5 esiteltyjä kyselykieliä keskenään. Kyselykieliä vertaillaan myös tulevaan kyselykielistandardiin. Luvussa 7 vastataan tutkimuskysymyksiin 2 ja 3. Lopuksi luvussa 8 esitetään yhteenveto tutkielman tuloksista sekä esitetään ehdotuksia mahdollisille jatkotutkimuksille.

2 Tutkimusmenetelmät

2.1 Dokumentaatioon perustuva katsaus

Tämän tutkielman tutkimusmenetelmä on nimetty dokumentaatioon perustuvaksi katsaukseksi. Tämän menetelmän avulla pyritään kartoittamaan, analysoimaan sekä syntetisoimaan dokumentaation kohteena oleva asia tai ilmiö. Tässä tutkielmassa dokumentaatioon perustuvan katsauksen vaihejakoa sovelletaan systemaattisen kirjallisuuskatsauksen vaihejakoon.

Kirjallisuuskatsaus on tieteellisen kirjallisuuden systemaattista tutkimista tietystä aiheesta. Kirjallisuuskatsaus voi olla osa tutkimusta, mutta koko tutkimus voi myös koostua pelkästään kirjallisuuskatsaukseen. Jälkimmäisessä tapauksessa puhutaan systemaattisesta kirjallisuuskatsauksesta. Systemaattinen kirjallisuuskatsaus voi toimia pohjana tuleville tutkimuksille, mutta katsaukseen ei sisälly konkreettista tutkimusta. Kirjallisuuskatsauksessa ei tuoda ilmi omia ideoita, argumentteja tai oletuksia. Kirjallisuuskatsauksen tulisi kuvata tämänhetkistä tietoa uudesta ja luovasta näkökulmasta, ja näin johtaa uudenlaiseen ajatteluun sekä ymmärrykseen tutkittavasta aiheesta. [Efron ja Ravid 2019] Tämä sama tavoite voidaan liittää myös dokumentaatioon perustuvaan katsaukseen.

Xiaon ja Watsonin (2019) mukaan kirjallisuuskatsaus sisältää kahdeksan vaihetta. Ensimmäisessä vaiheessa (1) määritellään tutkimuskysymykset. Toisessa vaiheessa (2) tehdään alustava suunnitelma, jossa määritellään katsauksen suorittamisessa käytettävät menetelmät. Kolmas vaihe (3) koostuu kirjallisuuden etsimisestä ja neljännessä vaiheessa (4) seulotaan edellisessä vaiheessa löydetystä kirjallisuudesta tutkimukseen sopivimmat aineistot. Viidennessä vaiheessa (5) arvioidaan löydetyn tutkimusaineiston laatua, jonka jälkeen kuudennessa vaiheessa (6) puretaan aineistoista löytyvä data. Seitsemännessä vaiheessa (7) aineistoista löydettyä dataa analysoidaan tutkimuskysymysten mukaisesti. Lisäksi dataa vertaillaan esimerkiksi taulukoiden tai kaavioiden avulla. Viimeisessä eli kahdeksannessa vaiheessa (8) raportoidaan löydetyt tulokset.

Tässä tutkielmassa kirjallisuuskatsauksen vaiheet 1–5 käydään läpi luvussa 2. Kuudes vaihe, eli aineistojen purku on toteutettu luvuissa 5 ja 6, joissa kerrotaan valittujen kyselykielten ominaisuuksista. Seitsemäs vaihe, eli aineistoista löydetyn datan

analysointi tutkimuskysymysten mukaisesti, on toteutettu luvussa 7, jossa vertaillaan kyselykielien ominaisuuksia toisiinsa. Kirjallisuuskatsauksen viimeinen vaihe, eli tulosten raportointi, toteutuu luvussa 8.

Tämä tutkielma eroaa perinteisestä systemaattisesta kirjallisuuskatsauksesta, sillä tutkimusaineistona on käytetty useita ei-tieteellisiä lähteitä. Tutkielman kohteena ovat graafitietokantojen kyselykielet. Kun halutaan hakea tarkkaa tietoa tietyn kyselykielen ominaisuuksista ja syntaksista, on luotettavin lähde kyseisen kielen virallinen dokumentaatio. Nämä dokumentaatiot eivät ole löydettävissä tieteellisistä tietokannoista. Tutkielman lähteinä on käytetty myös tieteellisiä artikkeleita, mutta lähteiden pääpaino on kyselykielten virallisissa dokumentaatioissa. Tästä syystä tämän tutkielman tutkimusmenetelmän voidaan sanoa olevan dokumentaatioon perustuva katsaus. Tutkielma sisältää samat vaiheet, kuin edellä kuvattu systemaattinen kirjallisuuskatsaus, mutta haku on suppeampaa ja kohdistuu kyselykielten dokumentaatioiden etsimiseen.

2.2 Tutkimusaineiston hankinta ja valintakriteerit

Kirjallisuuskatsauksen laatu on vahvasti riippuvainen siihen kerätyn kirjallisuuden laadusta. Koska kirjallisuutta hakemalla löydetään tarvittava materiaali tutkimusta varten, pohjaa systemaattinen kirjallisuuskatsaus siihen, että kirjallisuutta haetaan systemaattisesti. [Xiao ja Watson 2019]

Xiao ja Watson (2019) esittelevät kolme tärkeintä lähdettä kirjallisuuden hakuun. Nämä kolme lähdettä ovat: (1) elektroniset tietokannat, (2) taaksepäin hakeminen ja (3) eteenpäin hakeminen. Koska mikään yksittäinen tietokanta ei sisällä kaikkea julkaistua materiaalia haetusta aiheesta, tulisi systemaattista hakua suorittaa useampiin eri tietokantoihin. Jos tutkimuksessa tarvitaan kirjallisuutta, joka on julkaistu aikaa ennen internettiä, tulisi hakuja kohdistaa myös kirjastojen arkistoihin. Taaksepäin hakemisella tarkoitetaan kirjallisuuden hakemista löydettyjen aineistojen lähdeluetteloiden avulla. Eteenpäin hakemisella pyritään löytämään julkaisuja, jotka ovat viitanneet löydettyyn aineistoon sen julkaisuajankohdan jälkeen.

Hakusanojen tulisi olla johdettu tutkimuskysymyksistä. Suoraan tutkimuskysymyksellä ei kuitenkaan kannata hakua suorittaa, vaan se tulisi leikellä hakutulosten optimoimista varten sopiviksi palasiksi. Jos hakusanoille löytyy synonyymejä tai vaihtoehtoisia kirjoitusasuja, kannattaa hakuja suorittaa myös niillä. Hakuja suorittaessa on suositeltava

käyttää Boolean-operaattoreita, kuten ”AND” ja ”OR”, mikäli mahdollista. [Xiao ja Watson 2019]

Sopivia hakusanoja valittaessa on hyvä muistaa pitää tasapaino siinä, että hakuehto on täsmällinen, mutta ei kuitenkaan liian yksityiskohtainen. Jos hakuehtoja ei ole rajattu tarpeeksi, voi hakutulos sisältää paljon epäolennaisia lähteitä. Jos taas hakuehto on liian tarkka, voi hakutuloksesta rajautua pois lähteitä, joista olisi löytynyt tutkimukseen sopivaa tietoa. Etenkin tutkimuksen alkuvaiheessa on parempi lähteä suorittamaan hakuja niin, että tärkeitä lähteitä ei rajautuisi hakutuloksista pois. [Xiao ja Watson 2019]

2.3 Hakutulokset

Hakusanoiksi valikoituivat valittujen kyselykielten nimet. Jokaisen kyselykielen kohdalla aineiston vaatimuksena oli, että sen täytyy olla haetun kyselykielen dokumentaatio. Tämän lisäksi kyselykielistä haettiin mahdollista lisätietoa tieteellisistä tietokannoista.

Kyselykieli	**Haku**	**Dokumentaatio**
Cypher	cypher AND graph	Neo4j. Docs. Getting Started 5. Introduction to Cypher.
PGQL	pgql	PGQL | Property Graph Query Language. PGQL 1.5 Specification.
GSQL	gsql	TigerGraph. Docs. GSQL Language Reference 3.8.
Gremlin	gremlin AND graph	The Apache Software Foundation. TinkerPop Documentation.
SPARQL	sparql	World Wide Web Consortium. W3C Recommendation. SPARQL Query Language for RDF.
G-CORE	g-core	Angles *et al.* 2018. G-CORE: A Core for Future Graph Query Languages.
GQL & SQL/PGQ	gql AND graph	Deutsch *et al.* 2021. Graph Pattern Matching in GQL and SQL/PGQ.

Taulukko 1. Kyselykielet ja valitut dokumentaatiot

Taulukossa 1 on esitelty haut, joiden avulla kunkin kyselykielen dokumentaatio on löydetty. Taulukossa on lisäksi esitetty ne tieteelliset artikkelit, jotka läpäisivät valintakriteerit ja jotka otettiin mukaan kyselykielten syntaksin kuvaamiseen. Tieteellisten artikkelien valintakriteerinä oli, että niiden täytyy sisältää kuvausta valitun kyselykielen syntaksista, ominaisuuksista tai semantiikoista. G-CORE ja GQL & SQL/PGQ eivät ole vielä käytössä olevia kyselykieliä, joten niiden dokumentaatiota ei ole vielä saatavilla. Näistä molemmista on kuitenkin kirjoitettu tieteellisiä artikkeleita. Näin ollen näiden kahden dokumentaatioina toimii Andor-tietokannasta löydetyt tieteelliset artikkelit. Taulukon 1 dokumentaatioiden tarkemmat tiedot löytyvät tutkielman lähteistä.

2.4 Aineiston analyysi

Kirjallisuuskatsausten laatustandardit eroavat erityyppisten katsausten välillä [Whittemore ja Knafl 2005]. Laadun arviointi ei ole välttämätöntä esimerkiksi tutkimuksissa, joissa selvitetään aiemman tutkitun tiedon laajuutta, mutta ei laatua. Tutkimuksissa, joissa pyritään jonkin asian tai ilmiön yleistämiseen, laadun arvioiminen on tärkeää [Xiao ja Watson 2019].

Tässä tutkielmassa ei vertailtu aiempaa tutkimustietoa tai tieteellisiä artikkeleita. Tutkielman aineistoina käytettiin pitkälti kyselykielten omia dokumentaatioita. Haettaessa tietoa kyselykielten teoriasta, on niiden omat dokumentaatiosivustot luotettavin lähde. Näitä dokumentaatiosivuja tai kyselykielten teoriaa ei juurikaan löydy tietokannoista, joista haetaan tieteellisiä julkaisuja.

G-CORE ja GQL & SQL/PGQ kyselykielien lähteinä käytettiin tieteellisiä artikkeleita. G-CORE-artikkelia voidaan pitää luotettavana, sillä sen ovat kirjoittaneet kyseisen kyselykielen luojat. Myös GQL & SQL/PGQ lähteeksi valittua artikkelia voidaan pitää luotettavana, sillä sen kirjoittajat ovat osa työryhmää, jotka työstävät näitä tulevia graafikyselykielistandardeja.

3 Graafi tietorakenteena

Graafijärjestelmät keskittyvät erilaisten suhteiden varastointiin ja hakuun. Näillä suhteilla voidaan kuvata esimerkiksi ihmisten välisiä suhteita, asuinpaikkoja, omistussuhteita ja niin edelleen. [Broecheler ja Gosnell 2020]

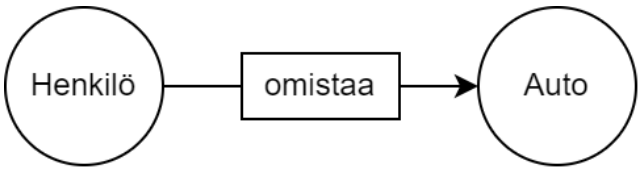
3.1 Graafi

Graafitietokannan kolme peruselementtiä ovat *graafi*, *solmu (vertex)* ja *särmä (edge)*. Graafissa tieto kuvataan kahden elementin avulla. Nämä kaksi elementtiä ovat solmu ja särmä. Solmu esittää konseptia tai kokonaisuutta ja särmä esittää suhdetta tai linkkiä kahden solmun välillä. Solmut voidaan linkittää toisiinsa särmien avulla. Nämä linkitykset kuvaavat suhteita eri tiedon palasten välillä. Graafissa särmä siis yhdistää kaksi solmua, mikä kuvastaa suhdetta näiden kahden yhdistetyn solmun välillä. Yhdessä kaikki solmut ja särmät muodostavat kokonaisuuden, jota kutsutaan graafiksi. [Broecheler ja Gosnell 2020] Graafi voidaan kuvata muodossa

$$G = (V, E),$$

jossa G on graafi, V on solmu ja E on särmä.

Graafitietokannassa data esitetään *solmu-* ja *särmänimikkeiden (labels)* avulla. Solmunimikkeillä kuvataan joukkoa objekteja, joilla on samanlaiset suhteet ja attribuutit. Särmänimikkeet nimeävät kahden solmun välillä olevan suhteen. Jokainen solmu luokitellaan solmunimikkeellä ja jokainen solmujen välillä oleva suhde luokitellaan särmänimikkeellä. Näin ollen solmunimikkeet kuvaavat datassa olevia kokonaisuuksia, jotka jakavat saman tyyppiset attribuutit ja samat suhteiden väliset nimikkeet. Särmänimikkeet kuvaavat suhteita solmunimikkeiden välillä. [Broecheler ja Gosnell 2020]

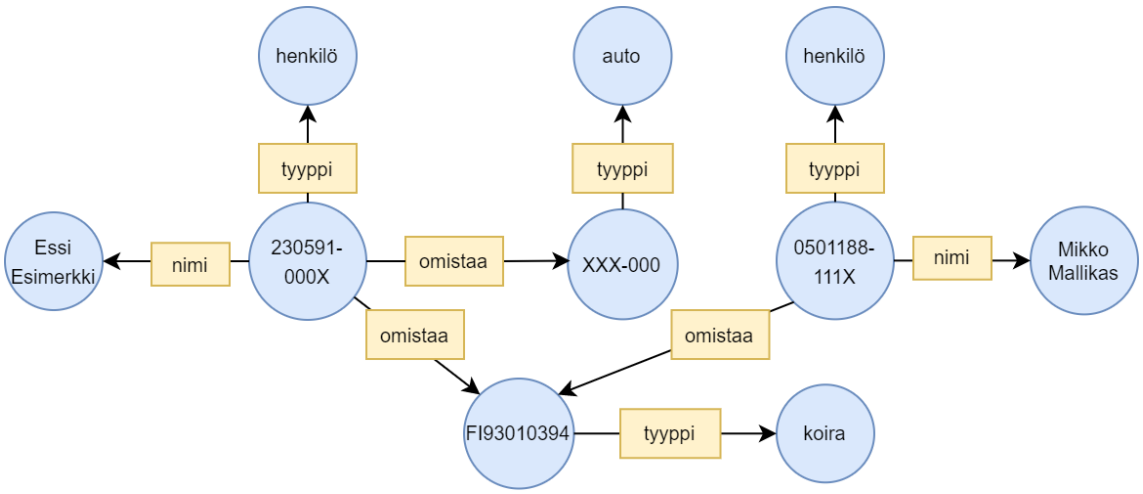


Kuva 1. Suunnattu graafi

Suhteita kuvaavilla särmillä on olemassa tietty suunta. Kun särmällä on suunta, sitä kutsutaan *kaareksi (arc)*. Kaaren suunta muodostuu samoin kuin datasta puhuttaessa.

Esimerkiksi lauseessa ”Henkilö omistaa auton” kaaren ’omistaa’ suunta on henkilöstä autoon. Kuva 1 havainnollistaa tätä esimerkkiä. Kaarien suunta voi olla *suunnattu (directed)* tai *kaksisuuntainen (bidirected)*. Suunnattu kaari osoittaa vain yhteen suuntaan: yhdestä solmusta toiseen solmuun. Kaksisuuntainen kaari puolestaan osoittaa molempiin suuntiin kahden solmun välillä. Kaksisuuntaisen kaaren *kaarinimikkeet* voivat erota toisistaan. [Broecheler ja Gosnell 2020] Kaarista voidaan käyttää myös nimitystä suhde.

Graafia, jossa kaksisuuntaiset kaaret ovat sallittuja, kutsutaan *multigraafiksi (multigraph)* tai särmämerkityksi graafiksi (edge-labelled graph) [Hogan *et al.* 2021]. Lisäksi multigraafissa sallitaan *silmukat (loops)*, joissa särmän molemmat päät osoittavan samaan solmuun [Marcus 2008]. Kuvassa 2 kuvataan multigraafia, jossa on kaksi henkilöä, yksi koira ja yksi auto.



Kuva 2. Multigraafi

Tässä tutkielmassa solmut esitetään ympyröinä, joiden sisällä on solmunimike. Kaaret ja niiden suunta kuvataan nuolella. Kaaria kuvaavien nuolien päällä on suorakaide, jonka sisällä on kaarinimike.

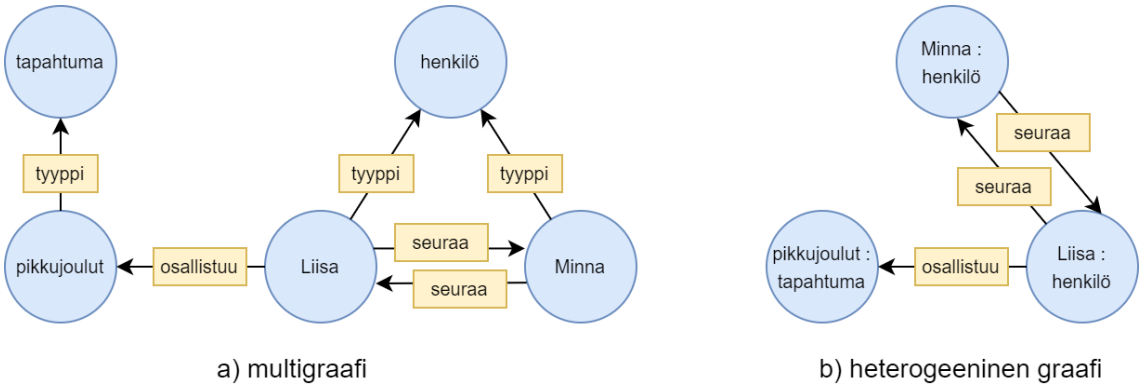
3.2 RDF-graafi

Multigraafien standardoidusta tietomallista käytetään lyhennettä *RDF*, joka tulee sanoista *Resource Description Framework* [Hogan *et al.* 2021]. RDF-graafit ovat subjekti-predikaatti-objekti-elementtien joukkoja, joissa solmu voi olla IRI (Internationalised Resource Identifiers), tyhjä solmu tai tyypitetty arvo. Subjekti- ja objektielementit kuvaavat solmuja ja predikaattielementti kaarta. Tyhjiä solmuja käytetään ilmaisemaan

entiteetin olemassaoloa. Tyypitetyt arvot voivat olla esimerkiksi merkkijonoja, numeroita tai päivämääriä [W3C 2014].

3.3 Heterogeeninen graafi

Heterogeenisessä graafissa (heterogeneous graph) jokaisella solmulla ja kaarella on tyyppi. Heterogeenisessä graafissa kaarien tyypit vastaavat multigraafin kaarinimikkeitä. Kun multigraafissa solmun tyyppi voidaan merkitä lisäämällä uusi kaari ja solmu, heterogeenisessä graafissa tämä kaikki onnistuu yhdessä solmussa. Kaarta kutsutaan *homogeeniseksi*, jos se yhdistää kaksi saman tyyppistä solmua. Jos kaari yhdistää kaksi eri tyyppiä olevaa solmua, on se heterogeeninen. Heterogeeniset graafit mahdollistavat solmujen jakamisen niiden tyyppien mukaan, mikä mahdollistaa niiden hyödyntämisen esimerkiksi koneoppimisessa. Toisin kun multigraafit, heterogeeniset graafit tyypillisesti sallivat yhdelle solmulle vain yhden siitä lähtevän ja yhden siihen osoittavan kaaren. [Hogan *et al.* 2021]



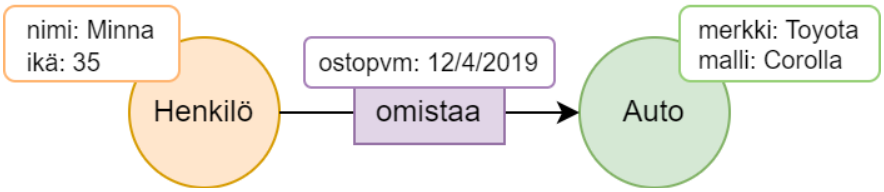
Kuva 3. Multigraafi (a) ja heterogeeninen graafi (b)

Kuvassa 3 on kuvattu multigraafin ja heterogeenisen graafin eroja. Heterogeenisessä graafissa solmut sisältävät tyypin, kun taas multigraafissa jokaisen solmun tyyppi täytyy ilmaista erillisellä solmulla ja tyyppi-kaarinimikkeellä. Kuten kuvasta ilmenee, ovat kaaret *osallistuu* ja *seuraa* molemmissa graafeissa samanlaiset.

3.4 Ominaisuusgraafi

Ominaisuusgraafi (property graph) on suosituin tapa mallintaa graafeja. Ominaisuusgraafi koostuu solmuista ja niiden välisistä suhteista. Kaikki suhteet eli kaaret on nimetty ja suunnattu. Lisäksi jokaisen kaaren molemmissa päissä on solmu. [Robinson *et al.* 2015]

Ominaisuusgraafissa tieto voidaan tallentaa *ominaisuuksiin* (*property*). Ominaisuudet vastaavat relaatiotietokantojen attribuutteja. Ominaisuuksissa kuvataan solmu- tai särmänimikkeen ominaisuuksia, kuten esimerkiksi nimiä ja päivämääriä. Myös solmun tai särmän identifioiva merkkisarja on ominaisuus [Broecheler ja Gosnell 2020]. Ominaisuudet ovat avain-arvopareja. Solmu tai kaari voi sisältää monta ominaisuutta [Robinson *et al.* 2015].



Kuva 4. Ominaisuusgraafi

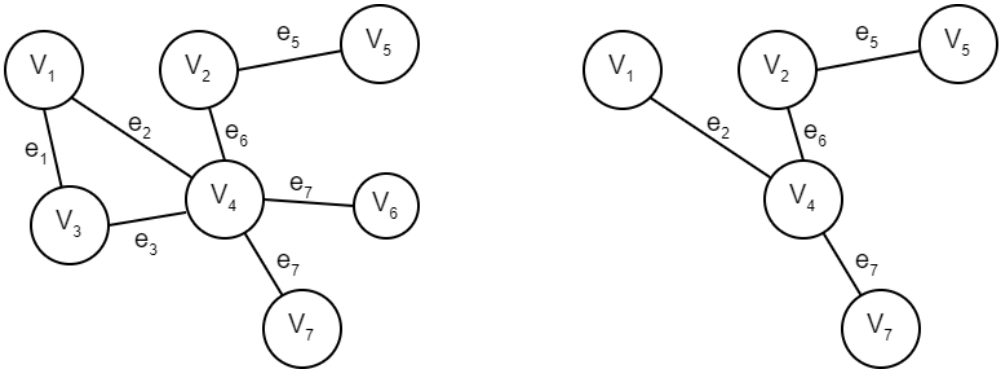
Kuvassa 3 on esimerkki ominaisuusgraafista. Graafissa on kaksi solmua, joiden solmunimikkeet ovat henkilö ja auto, sekä yksi kaari, jonka kaarinimike on omistaa. Sekä henkilösolmulla, että autosolmulla on kaksi attribuuttia. Henkilön attribuutit ovat nimi sekä ikä ja auton attribuutit merkki sekä malli. Kaarella ’omistaa’ on yksi attribuutti. Kaaren suunta on henkilöstä autoon.

3.5 Aligraafi

Graafin G aligraafi (subgraph) on mikä tahansa graafi H, jossa

$$V_H \subseteq V_G \text{ ja}$$
$$E_H \subseteq E_G, \text{ missä}$$

jokainen graafin H solmu on myös graafin G solmu, ja jokainen graafin H särmä on myös graafin G särmä. Näin ollen graafi G sisältää graafin H. [Gould 1988]



Kuva 5. Aligraafi

Kuvassa 4 on kuvattu vasemmalla graafi G ja oikealla sen aligraafi H. Kuten kuvasta käy ilmi, graafi G sisältää kaikki graafin H solmut ja särmät. Näin ollen graafi G sisältää graafin H, eli graafi H on graafin G aligraafi.

4 Kyselyt graafitietokannassa

4.1 Kulkeminen graafissa

Reitti (walk) graafissa on sarja, jossa on vuoron perään solmuja ja kaaria. Reitti solmusta toiseen on suljettu (*closed*), jos reitin ensimmäinen ja viimeinen solmu on sama. [Beineke ja Wilson 2013] Graafin sisällä voidaan kulkea erilaisia reittejä eli *polkuja (path)*. Polussa solmujen ja kaarien toistot ovat sallittuja. [Marcus 2008] Polku voi olla kuitenkin myös *yksinkertainen polku (trail)* tai *suora polku (simple path)*. Yksinkertaisessa polussa jokaisen kaaren kautta kuljetaan vain kerran. Suorassa polussa jokaisessa solmussa käydään vain kerran. [Beineke ja Wilson 2013] Polun pituus määräytyy polussa käytyjen kaarien lukumäärän mukaan. Tähän lasketaan mukaan myös mahdolliset toistot, jos kaaren kautta on kuljettu useamman kerran. [Marcus 2008] Jos polun pituus on 0, kyseessä on pelkkä solmu. [Balakrishnan ja Ranganathan 2012]

Reitti R graafissa G on vuorotteleva lauseke solmuja ja kaaria. Tämä voidaan ilmaista lausekkeella

$$W = v_0\, e_1\, v_1\, e_2\, v_2 \ldots e_p\, v_p,$$

jossa v_0 on reitin W lähtösolmu ja v_p päätössolmu.

Reitti alkaa aina solmulla ja päättyy solmuun. Reitti solmusta v_0 solmuun v_p voidaan esittää muodossa $v_0 – v_p$, sillä se yhdistää nämä kaksi solmua.

4.2 Graafimallit

Graafimallin yksinkertaisinta muotoa kutsutaan *perusgraafimalliksi (basic graph pattern)*. Graafimalli on graafi, joka voi sisältää muuttujia, kun taas graafi, johon kyselyt suoritetaan sisältää vakioita. Kuvassa 4 solmunimike *Henkilö* ja nimitunniste *Minna* ovat vakioita. Graafimallia P arvioidaan suhteessa graafiin G. Tämä tapahtuu muodostamalla kartoituksia siten, että graafimallin muuttujat korvataan graafin G vakioilla, kunnes kartoituksen alla oleva graafimalli sisältyy graafiin G. [Hogan *et al.* 2021]

Graafimallien arvioimisessa voidaan käyttää erilaisia semanttisia malleja. Näistä yleisimmät ovat *homomorfismiin perustuva semantiikka (homomorphism-based semantics)* sekä *isomorfismiin perustuva semantiikka (isomorphism-based semantics)*. Homomorfismiin perustuva semantiikassa sallitaan, että vakio *x* voidaan sitoa

graafimallissa samanaikaisesti useaan eri muuttujaan. Näin ollen vakio *x* voi olla sidottuna ”samalla rivillä”, eli yhdessä graafimallissa, useampaan eri muuttujaan. Isomorfismissa sen sijaan vaaditaan, että graafimallin solmujen ja/tai kaarien muuttujat tulee kartoittaa yksilöllisille termeille. [Hogan *et al.* 2021]

Perusgraafimallia voidaan laajentaa *kompleksisilla graafimalleilla (complex graph patterns)*. Nämä graafimallit mahdollistavat relaatioalgebraan käyttämisen graafikyselyissä. Relaatioalgebrata käytetään muun muassa SQL-kyselykielessä. Kompleksit graafimallit mahdollistavat muun muassa operaatiot: projektio (SELECT), valikointi (WHERE tai FILTER), yhdiste eli unioni (UNION), erotus (EXCEPT), luonnolliset liitokset (NATURAL JOIN) ja vasen ulkoliitos (LEFT OUTER JOIN). [Hogan *et al.* 2021]

Kompleksiset graafimallit voivat antaa tulokset duplikaatteina. Tämän vuoksi kyselykielissä voidaan käyttää erilaisia tapoja tulosten käsittelyyn. Näistä kaksi tyypillisintä semantiikkaa ovat *bag-semantiikka (bag semantics)* ja *set-semantiikka (set semantics)*. Bag-semantiikka säästää mahdolliset duplikaatit, kun taas set-semantiikka poistaa tuloksessa olevat toistot. Set-semantiikka toimii kuten DISTINCT-lauseke, eli se karsii tuloksesta pois kaikki duplikaatit. [Hogan *et al.* 2021] Valinta siitä, kumpaa tulostenkäsittely semantiikkaa kyselykielessä käytetään, riippuu siitä, minkä tyyppisiä paluuarvoja kyseinen kyselykieli palauttaa [Sharma *et al.* 2021].

4.3 Hahmonsovitus

Graafien *hahmonsovituksessa (pattern matching)* pyritään löytämään graafista G kaikki sellaiset esiintymät, jotka täsmäävät annettuun graafimalliin P. Hahmonsovitus määritellään tyypillisesti aligraafin isomorfismin suhteen. Tämä tarkoittaa, että haetaan kaikki graafin G aligraafit, jotka ovat isomorfisia graafimallin P suhteen. Mallin P esiintymä on graafin G aligraafi G′, jos on olemassa bijektio

$$f(\mathrm{v}) : \mathrm{P}(\mathrm{v}) \rightarrow \mathrm{G}'(\mathrm{v}),$$

jossa jokaisella aligraafin G′ solmulla v ja jokaisella funktion f(v) solmulla v on sama nimike. Lisäksi graafimallissa P on särmä (v, v′) jos ja vain jos aligraafissa G′ on olemassa särmä (f(v), f(v’)). [Fan *et al.* 2010]

4.4 Polkukyselyt

Polkukysely P voidaan määritellä lausekkeella

$$P = x \rightarrow^{\alpha} y, \text{ missä}$$

α määrittää haettavan polun ehdot. Ehdoissa α voidaan määritellä esimerkiksi polussa sallittujen kaarien nimike tai polun pituus. Muuttujat x ja y määrittävät polun päät, jotka voivat olla muuttujia, tietyt solmut tai näitä molempia. Muuttujat x ja y voivat myös osoittaa samaan solmuun. Merkin α sijasta voidaan käyttää myös merkkiä *. Käyttämällä tätä (*) merkkiä voidaan ilmaista, että solmujen x ja y välillä oleva polku voi olla mikä tahansa, eikä sille ole asetettu mitään ehtoja. Merkki α puolestaan ilmaisee, että polulle on asetettu jokin rajoitus. *Polkurajoituksia (path constraints)* on olemassa useampia, mutta niistä tunnetuin lienee *säännöllinen lauseke (regular expression)*. Säännölliset lausekkeet mahdollistavat polkujen ketjuttamisen käyttämällä niiden yhdistämistä tai erottamista ja käyttämällä polkua määrittelemättömän monta kertaa. Säännöllistä lauseketta käyttävää polkukyselyä kutsutaan *säännölliseksi polkukyselyksi* (*regular path query, RPQ*). [Angles *et al.* 2017]

Säännöllisiä polkukyselyitä voidaan laajentaa sallimalla niille lisälausekkeita. Säännölliset polkukyselyt sallivat polkujen kulkemisen vain yhteen suuntaan, eli kaaren määrään suuntaan (eteenpäin). Tietynlaisten kyselyjen kannalta olisi kuitenkin olennaista pystyä kulkemaan sekä eteen- että taaksepäin. Yksi säännöllisten polkukyselyiden lisälauseke on *kahdensuuntaiset säännölliset polkukyselyt (two-way regular path queries, 2RPQ)*, jotka mahdollistavat edellä kuvatun kulkemisen myös taaksepäin. [Calvanese *et al.* 2003] Kahdensuuntainen säännöllinen polkukysely P, voidaan määritellä lausekkeella

$$P = x \rightarrow^{\alpha-} y, \text{ missä}$$

α määrittää haettavan polun ehdot ja miinusmerkki (-) ilmaisee, että polkua saa kulkea molempiin suuntiin.

Säännöllisiä polkukyselyitä voidaan arvioida erilaisten semantiikkojen mukaan. Polkukyselyissä on mahdollista saada vastukseksi syklejä, joissa polkujen määrä on ääretön. Tämän vuoksi polkukyselyissä käytetään usein tuloksia rajoittavia semantiikkoja. [Hogan *et al.* 2021]

Lyhimmän polun semantiikkaa (shortest path semantics) noudattava kysely palauttaa nimensä mukaisesti annettujen ehtojen mukaisen lyhimmän polun. Tässä tapauksessa polkuun P(G) otetaan mukaan lyhimmän pituuden saava polku, joka täyttää polulle P asetetut vaatimukset. Tätä semantiikkaa käytetään, kun halutaan löytää linkki kahden solmun välillä ja niiden välinen lyhin mahdollinen polku. [Angles *et al.* 2017] Kysely voi noudattaa myös semantiikka, jossa yhdessä solmussa saa käydä vain kerran *(no-repeated-node semantics)* tai jossa yhden kaaren kautta saa kulkea vain kerran *(no-repeated-edge semantics)*. [Hogan *et al.* 2021] Kun käytetään semantiikka, jossa yhdessä solmussa saa käydä vain kerran, polku P(G) sisältää kaikki polut, jotka täyttävät annetut vaatimukset ja joissa jokainen polun varrella oleva solmu esiintyy vain kerran. Tämä semantiikka toimii hyvin esimerkiksi, kun halutaan etsiä reittiä paikasta A paikkaan B. Jos mitään rajoittavaa semantiikkaa ei käytetään, kyselyn semantiikka on sattumanvarainen *(arbitrary path semantics)*. Tämä tarkoittaa, että kysely palauttaa kaikki graafin G polut, jotka täyttävät niille annetut vaatimukset. Tätä semantiikkaa voidaan käyttää, kun halutaan tietää, onko kahden solmun välillä olemassa yhteyttä. [Angles *et al.* 2017] Yhdistämällä säännölliset polkukyselyt ja luvussa 4.3 esitellyt kompleksiset graafimallit syntyvät *kompleksiset navigointigraafimallit (complex navigational graph patterns)* [Hogan *et al.* 2021].

Jotta voitaisiin esittää kahden kyselyn semanttinen samanarvoisuus eri kyselykielten välillä, täytyy kyselykielten käyttää samaa tulosten käsittelysemantiikkaa (set tai bag) ja polkujen käsittelysemantiikkaa (lyhin polku, no-repeated-node, no-repeated-edge tai sattumanvarainen). Semantiikat ovat riippuvaisia kyselykielten käytännön toteutuksista. Tästä syystä semanttista samanarvoisuutta kahden kyselyn välillä ei voida todistaa ilman, että kyselyihin tehdään lauseopillisia muutoksia. [Sharma *et al.* 2021]

4.5 Paluuarvot

Graafikyselyiden paluuarvot riippuvat kyselyn tyypistä. Käyttäjä voi haluta tietää, sisältääkö graafi G jonkin tietyn polun tai mitkä solmut ovat yhteydessä toisiinsa tietyn polun kautta. Kyselyiden avulla voidaan myös selvittää, minkälainen polku yhdistää solmuja v_n ja v_m. Graafikyselyt voivat palauttaa neljä erityyppistä paluuarvoa.

- Totuusarvo: Graafin polkukysely voi palauttaa arvon tosi/epätosi (true/false). Esimerkiksi, onko graafissa G polkua, joka yhdistää kaksi tiettyä solmua.

- Solmu: Kysely voi palauttaa myös solmun tai solmuja. Kyselyssä määritellään tällöin tietty polku π graafissa G. Paluuarvona tällaisessa kyselyssä saadaan kaikki solmuparit, joita yhdistää polku π. Polun π solmupari u ja v ovat polun alku- ja loppupäät.

- Polku: Graafikyselyssä paluuarvona voi olla joku tai kaikki vaatimukset täyttävistä graafin G poluista. Esimerkiksi lyhimmän polun semantiikkaa käytettäessä voidaan palauttaa yksi tai useampi lyhimmistä poluista. Joissakin tapauksissa, kuten esimerkiksi reitinsuunnittelusovelluksissa, halutaan usein rajoittaa löydettyjä polkuja niin, että sovellus näyttää esimerkiksi kolme parasta reittivaihtoehtoa.

- Graafi: Kyselyiden avulla voidaan graafi G kuvata tiiviimmässä muodossa uuden graafin avulla. [Angles *et al.* 2017]

4.6 Graafialgoritmit

Graafialgoritmeillä pyritään ratkaisemaan graafiglobaaleja ongelmia. Kun graafista halutaan hakea tietyn tyyppisiä rakenteita, on kyseessä graafiglobaali ongelma. Näiden ongelmien ratkaisemisessa kyselyt eivät ole aina paras vaihtoehto. Graafiglobaali ongelma on yleensä merkki siitä, että ongelman ratkaisemiseen kannattaa harkita graafialgoritmin käyttämistä. [Barrasa *et al.* 2021]

Graafialgoritmeista yleisesti tunnetuimmat voidaan jakaa viiteen eri kategoriaan [Barrasa *et al.* 2021]:

- Keskeisyys (*centrality*)
- Yhteisöjen havainnointi (*community detection*)
- Polkujen etsiminen (*pathfinding*)
- Yhteyksien ennustaminen (*link prediction*)
- Samankaltaisuus (*similarity*)

Keskeisyyttä laskevia algoritmeja käytetään määrittämään verkoston erillisten solmujen tärkeyttä [Barrasa *et al.* 2021]. Näillä algoritmeilla pyritään erityisesti löytämään verkoston tärkeimmät solmut. Keskeisyyttä laskevia algoritmeja on useita erilaisia, ja ne pyrkivät mittaamaan hieman eri asioita. Algoritmeilla voidaan esimerkiksi arvioida

solmujen kykyä levittää informaatiota tai yhdistää toisistaan erillä olevia ryhmittymiä. [Needham ja Hodler 2019]

Yhteisöjen havainnointialgoritmeilla pyritään löytämään solmujen ryhmittymiä tai todennäköisiä jakautumia [Barrasa *et al.* 2021]. Yhtenäisyys on yksi graafiteorian ydinkonsepteista. Se mahdollistaa pitkälle kehitetyn verkostoanalyysin. Arvioimalla erityyppisiä solmujen ryhmittymiä, voidaan graafeista löytää rakenteita, kuten keskittymiä ja hierarkioita sekä ryhmittymien taipumuksia vetää puoleensa tai hylkiä solmuja. [Needham ja Hodler 2019]

Polkujen etsimisellä arvioidaan optimaalisia polkuja ja määritetään reittien laatua ja saavutettavuutta [Barrasa *et al.* 2021]. Polut ovat olennainen osa graafien analytiikkaa ja algoritmeja. Lyhimmän polun (*shortest path*) löytäminen on oletettavasti yleisin graafialgoritmeilla suoritettava tehtävä ja se toimiikin lähtökohtana monissa erityyppisissä analyyseissä. [Needham ja Hodler 2019]

Yhteyksien ennustamista käytetään arviomaan sitä, kuinka todennäköisesti solmut muodostavat toisiinsa yhteyden [Barrasa *et al.* 2021]. Se on yleinen koneoppimisessa käytettävä tehtävä graafeille. Koneoppimismallille annettavat syötteet ovat esimerkkejä solmupareista. Mallia koulutettaessa solmuparit merkitään joko vierekkäisiksi tai ei-vierekkäisiksi. [Neo4j 2023b]

Samankaltaisuusalgoritmit arvioivat sitä, kuinka samankaltaisia solmut ovat keskenään [Barrasa *et al.* 2021]. Kahden solmun samankaltaisuutta voidaan arvioida niiden ympäristön tai ominaisuuksien perusteella [Neo4j 2023b]. Graafien samankaltaisuuslaskennat voidaan jakaa karkeasti kahteen ryhmään. Ensimmäisen ryhmän laskelmien tulokset osoittavat, kuinka samankaltaisia kaksi graafia (G, G´) ovat kokonaisuudessaan. Toisen ryhmän metodit tuottavat hienojakoisemman kuvan graafien samankaltaisuudesta, vertaamalla jokaista solmua tai kaarta kahden graafin välillä. [Kollias *et al.* 2014]

5 Graafitietokantojen kyselykielet

Erilaiset graafimallit tarvitsevat omat kyselykielensä. Graafien kyselykielet eroavat toisistaan muun muassa lausekieleltään ja ominaisuuksiltaan. Seuraavissa aliluvuissa kuvataan tarkemmin graafikantojen kyselykieliä sekä niiden ominaisuuksia ja muodostussääntöjä. Luvuissa pyritään myös vastaamaan ensimmäiseen tutkimuskysymykseen: millaisia graafitietokantojen kyselykieliä on olemassa?

5.1 Cypher

Cypher on Neo4j-alustan kyselykieli, jolla voidaan hakea tietoa ominaisuusgraafeista. Cypherissa on vaikutteita SQL-kyselykielestä. Cypher-kyselykielestä tekee erityisen sen visuaaliset muodostussäännöt. [Neo4j 2023a] Cypher tarjoaa myös mahdollisuuden visualisoida graafeja [Sharma *et al.* 2021]. Cypherista on tehty myös yleinen, openCypher-kyselykieli, jota voidaan käyttää useilla eri graafitietokanta-alustoilla [Neo4j 2021].

Cypherissa solmut kuvataan niin, että solmunimike kirjoitetaan sulkeisiin. Solmunimike *auto* kuvattaisiin näin ollen seuraavasti: (auto). Sulkeiden sisään sijoitettu solmunimike on visuaalisesti samankaltainen kuin Cypherin datamallin esitys solmuista, joissa solmunimikkeet kirjoitetaan ympyröiden sisään. Jos tiettyyn solmuun halutaan viitata myöhemmin samassa kyselyssä, voidaan sille antaa muuttuja. Esimerkiksi auto-solmuun voitaisiin viitata lyhenteellä (a). Kuten ohjelmointikielissäkin, muuttujat voi nimetä oman mielensä mukaan. Jos solmua ei tarvitse palauttaa kyselyn tuloksissa, se voidaan myös esittää nimettömänä, käyttämällä tyhjiä sulkeita (). Edellä kuvatun kaltaiseen solmuun ei voi viitata myöhemmin kyselyssä. Jos kyselyssä ei eritellä haettavien solmujen solmunimikkeitä, kysely käy läpi kaikki tietokannassa olevat solmut. Tämä on hyvin vaivalloista etenkin, jos kysely suoritetaan todella suureen graafiin. Näin ollen kyselyissä on suositeltavaa käyttää solmunimikkeitä aina kun mahdollista. [Neo4j 2023a]

Alla on esitetty solmujen ja muuttujien kuvaaminen Cypherilla:

```
()                  // nimetön solmu (ei nimikettä tai muuttujaa)
(h:Henkilö)         // muuttuja h, nimike Henkilö
(:Auto)             // ei muuttujaa, nimike Auto
(juhla:Tapahtuma)   // muuttuja juhla, nimike Tapahtuma
```

Solmujen väliset suhteet kuvataan Cypherilla ASCII-nuolilla seuraavasti: --> tai <--. Myös tämä esitystapa on visuaalisesti samankaltainen kuin Cypherin datamallissa, joissa suhteet kuvataan nuolilla tai viivoilla.

Tarkemmat tiedot, kuten suhteiden nimikkeet ja niiden ominaisuudet, voidaan laittaa hakasulkeisiin nuolen kahden väliviivan väliin (-[x]->). Suhteen nimike kirjoitetaan isoilla kirjaimilla. Cypherilla voidaan tehdä myös kyselyitä, joissa ei välitetä suhteen suunnasta. Tällöin suhteet voidaan ilmaista kahdella väliviivalla (--). Cypher siis tukee kahdensuuntaisia säännöllisiä polkukyselyitä. Jos Cypher-kyselyssä ilmaisee suhteen suunnan väärin, ei kysely palauta tuloksia. Tämänkaltaisissa tapauksissa, joissa kyselyn tekijällä ei ole tietoa suhteen suunnasta, onkin kyselyssä järkevämpää käyttää ei-suunnattua suhdetta. Kuten solmulle, myös suhteelle voidaan antaa muuttuja, jolloin siihen voidaan viitata myöhemmin samassa kyselyssä. [Neo4j 2023a]

Alla on esitetty suhteiden ja muuttujien kuvaaminen Cypherilla:

```
-[suhde]->                  // muuttuja suhde, nimikettä ei määritetty
-[:OMISTAA]->               // suhteen nimike omistaa
-[omistaja:OMISTAA]->       // muuttuja omistaja ja omistaa-nimike
```

Solmujen ja suhteiden ominaisuudet ja niiden arvot kirjoitetaan Cypherilla aaltosulkeiden sisään. Aaltosulkeet puolestaan sijoitetaan joko solmun, eli sulkeiden sisään tai suhteen, eli hakasulkeiden sisään.

Solmu, jonka tyyppi on *henkilö* ja jolla on ominaisuus *nimi*, jonka arvo on *Tiina*, kuvataan Cypherilla seuraavasti:

```
(h:Henkilö {nimi: 'Tiina'})          // muuttuja h
```

Suhde, jonka tyyppi on *omistaa* ja jolla on ominaisuus *ostovuosi*, jonka arvo on *2021*, kuvataan Cypherilla seuraavasti:

```
-[om:OMISTAA {ostovuosi: 2021}]->    // muuttuja om
```

Cypherilla graafimallit voidaan esittää yhtäjaksoisina polkuina tai ne voidaan jakaa pienempiin osiin. Graafimalli saadaan luotua yhdistämällä solmujen ja suhteiden kuvaus. Lause *Tiina omistaa auton* kuvataan Cypherilla seuraavasti:

```
(h:Henkilö {nimi: 'Tiina'})-[om:OMISTAA]->(a:Auto {merkki: "Audi"})
```

Kahden solmun välisen polun pituus voidaan määritellä hakasulkeiden sisällä *-merkin avulla ja lisäämällä sen perään suhteiden lukumäärän. Pituus voidaan myös määritellä vaihteluvälinä.

```
(a)-[*2]->(b)            // polun pituus 2
(a)-[*3..5]->(b)         // polun pituus minimissään 3, maksimissaan 5
```

RETURN-lauseen avulla voidaan palauttaa solmuja, suhteita, solmujen ja suhteiden ominaisuuksia sekä polkuja. RETURN-lauseen käyttö on välttämätöntä silloin, kun tietokannasta halutaan lukea tietoa. Joskus kyselyt voivat palauttaa tuloksia, jotka ovat duplikaatteja. Jos tätä ei ole toivottua, voidaan RETURN-lauseen jälkeen käyttää avainsanaa DISTINCT, jolloin tuloksissa ei esiinny duplikaattituloksia. [Neo4j 2023a]

MATCH-lauseen avulla voidaan hakea malleja, jotka vastaavat kyselyssä esitettyä graafi- tai navigointimallia. MATCH-lauseessa toteutetaan myös mahdollinen graafimallien tai navigointimallien luonnollinen liitos (natural join). [Sharma *et al.* 2021] MATCH-lauseessa täytyy esitellä kaikki solmut, suhteet, ominaisuudet tai mallit, jotka halutaan kyselyn palauttavan. Haku, jossa haetaan *henkilöä*, jonka *nimi* on *Tiina* näyttää seuraavanlaiselta:

```
MATCH (h:Henkilö {nimi: 'Tiina'})
RETURN h
```

Hakutuloksia voidaan suodattaa WHERE-lauseella, jonka käyttö ei ole pakollista. Loogisen tai-operaation (OR) käyttö tapahtuu WHERE-lauseessa. [Sharma *et al.* 2021] Cypherilla voi suodattaa tuloksia käyttämällä säännöllisiä lausekkeita. Cypherin säännöllisten lausekkeiden syntaksi on peritty Javan Pattern-luokalta. Joissakin tapauksissa halutaan, että kysely palauttaa myös sellaisia malleja, jotka eivät täysin vastaa kyselyssä annettuja kriteereitä mallille. Näissä tapauksissa voidaan käyttää lausetta OPTIONAL MATCH, joka vastaa SQL-kyselykielen ulkoliitosta (*outer join*). [Neo4j 2023a]

Cypherissa on käytettävissä UNION-lauseke, joka yhdistää kahden tai useamman kyselyn tulokset yhdeksi tulokseksi. Tämä tulos sisältää jokaisen kyselyn kaikki rivit koottuna yhteen. UNION-lauseke yhdistää ensin kaikki saadut vastaukset, jonka jälkeen se poistaa tuloksesta mahdolliset duplikaatit. Jos halutaan, että kysely palauttaa myös duplikaattirivit, voidaan kyselyssä käyttää UNION ALL -lauseketta. [Neo4j 2023a] Kysely, joka sisältää unionin, esitetään Cypherilla seuraavasti:

```
MATCH (h:Henkilö)
RETURN h.nimi
UNION
MATCH (a:Auto)
RETURN a.merkki
```

Graafimallit luodaan Cypherilla CREATE-lauseilla, eli toisin sanoen tämän lauseen avulla lisätään tiedot graafitietokantaan. Yksi komento voi sisältää monta CREATE-lausetta. Onnistuneesti tietokantaan lisätyt tiedot voidaan palauttaa käyttämällä luontilauseen kanssa samassa kyselyssä RETURN-lausetta, joka on Cypher-kyselykielen projektio. RETURN-lauseen yhteydessä tulee antaa muuttuja, joka halutaan palauttaa. Esimerkiksi kysely:

```
CREATE (h:Henkilö {nimi: 'Timo', ikä: 38})
RETURN h,
```

palauttaa juuri luodun solmun *Henkilö*, jolla on ominaisuus *nimi*, jonka arvo on *Timo* ja ominaisuus *ikä*, jonka arvo on *38*.

Jos halutaan lisätä graafitietokantaan tietoja, jotka saattavat jo olla tietokannassa, kannattaa käyttää Cypherin MERGE-lausetta. Tämä lause on käytännössä MATCH- ja CREATE-lauseiden kombinaatio, sillä se etsii ensin, löytyykö tietokannasta entuudestaan kyseistä dataa. Jos dataa ei löydy, se lisätään tietokantaan. Käyttämällä MERGE-lausetta varmistutaan siis siitä, ettei tietokantaan lisätä vahingossa duplikaatti dataa tai rakenteita. Jo olemassa olevien solmujen ja suhteiden ominaisuuksia voidaan muokata SET-lauseella. [Neo4j 2023a]

Cypherin DELETE-lause toimii pitkälti samaan tapaan kuin SQL:ssä. Koska Neo4j-alusta noudattaa ACID-periaatteita, täytyy kyselyssä ottaa huomioon se, onko

poistettavalla solmulla suhteita. Solmua ei siis voi poistaa, jos siihen linkittyy yksi tai useampi suhde. Näin ollen mahdolliset solmuun linkittyvät suhteet tulee poistaa ensin ja vasta kun suhteet on poistettu, voidaan itse solmu poistaa. [Neo4j 2023a]

Cypher noudattaa isomorfismiin pohjaavaa semantiikkaa, jossa yhden kaaren kautta saa kulkea vain kerran (*no-repeated-edge semantics*) sekä bag-semantiikkaa, jossa tuloksissa esiintyvät duplikaatit säilyvät. DISTINCT-lauseketta käyttämällä voidaan käyttää myös set-semantiikkaa. [Neo4j 2023a] Lisäksi lyhimmän polun semantiikkaa on mahdollista käyttää. Homomorfismiin pohjaavaa semantiikkaa voidaan toteuttaa käyttämällä useampaa MATCH-lausetta. [Sharma *et al.* 2021]

5.2 PGQL

PGQL on SQL-pohjainen kyselykieli ominaisuusgraafeille ja se on yksi keskeisimmistä Oraclen graafitietokannan komponenteista. Koska PGQL on rakennettu SQL:n pohjalta, sisältää se kaikki SQL:n ominaisuudet. [PGQL 2023]

PGQL-kyselyt vastaavat syntaksiltaan hyvin paljon Cypheria, joten myös sen voidaan sanoa omaavaan visuaaliset muodostussäännöt. Solmut esitetään Cypherin tavoin sulkeiden sisällä ja solmun tyyppi on muotoa *:Tyyppi*. Alla on esitetty solmujen ja muuttujien kuvaaminen PGQL:llä:

```
()                      // nimetön solmu (ei nimikettä tai muuttujaa)
(h:Henkilö)             // muuttuja h, nimike Henkilö
(:Auto)                 // ei muuttujaa, nimike Auto
(juhla:Tapahtuma)       // muuttuja juhla, nimike Tapahtuma
(n:Henkilö|Tapahtuma)   // muuttuja n, nimike Henkilö tai Tapahtuma
```

Kuten edellä esitetyistä esimerkeistä voi huomata, vastaa PGQL:n esitystapa hyvin paljon Cypheria. PGQL:ssä on olemassa myös solmun esitystapa, jossa voidaan asettaa ehto solmulle. Yllä esitetty lause *(n:Henkilö|Tapahtuma)* tarkoittaa, että haettavan solmun *n* nimike voi olla joko *Henkilö* tai *Tapahtuma*.

Myös suhteiden kuvaaminen PGQL:lla vastaa hyvin pitkälti Cypherin mallia. Suhde esitetään muodossa -[:tyyppi]-> tai <-[:tyyppi]-, joissa *tyyppi* on suhteen nimike. Suhde voidaan asettaa talteen muuttujaan. PGQL tukee kahdensuuntaisia polkukyselyitä. Kahdensuuntaisuus ilmaistaan kahdella väliviivalla (--). Jos suhde halutaan tallentaa

muuttujaan, voidaan se ilmaista muodossa -[s:tyyppi]-, jossa *s* on muuttuja ja *tyyppi* suhteen nimike. [PGQL 2023]

Alla on esitetty suhteiden ja muuttujien kuvaaminen PGQL:lla:

```
-[suhde]->                  // muuttuja suhde, nimikettä ei määritetty
-[suhde]-                   // muuttuja suhde, kahdensuuntainen suhde
-[:omistaa]->               // suhteen nimike omistaa
-[om:omistaa]->             // muuttuja om ja omistaa-nimike
-[e:omistaa|seuraa]->       // muuttuja e, nimike omistaa tai seuraa
```

Toisin kuin Cypherissa, suhteiden nimikkeet ilmaistaan pienillä kirjaimilla. Myös suhteille voidaan asettaa disjunktio-ehto. Yllä esitetty lause *(e:omistaa|seuraa)* tarkoittaa, että haettavan suhteen *e* nimike voi olla joko *omistaa* tai *seuraa*. Jos suhteelle ei anneta muuttujaa tai nimikettä, määritellään se nuolella, jossa on vain yksi väliviiva (-> tai <-). [PGQL 2023]

Lause *Henkilö omistaa auton* kuvattaisiin PGQL:lla seuraavasti:

```
(h:Henkilö)-[:omistaa]->(a:Auto),
```

missä henkilösolmu tallennetaan muuttujaan *h* ja autosolmu tallennetaan muuttujaan *a*. Suhdetta *omistaa* ei tässä esimerkissä tallenneta muuttujaan.

Kahden solmun välisen polun pituus voidaan määritellä suhdetta kuvaavan nuolen jälkeen. Pituus voidaan määritellä vaihteluvälinä. Solmujen välinen etäisyys voidaan määritellä muun muassa seuraavilla tavoilla:

```
(a) -[e]->* (b)          // polun pituus 0 tai enemmän
(a) -[e]->{ 3, 5 } (b)   // polun pituus minimissään 3, maksimissaan 5
```

Tietokannasta haettavat graafimallit ja navigointimallit eli solmut ja suhteet esitetään FROM MATCH -lauseessa. Tässä lauseessa solmut ja suhteet voidaan tallentaa muuttujiin. [PGQL 2023] FROM MATCH -lauseessa toteutetaan myös mahdollinen graafimallien tai navigointimallien luonnollinen liitos (natural join) [Sharma *et al.* 2021].

Solmuja ja suhteita voidaan suodattaa niiden ominaisuuksien perusteella WHERE-lauseella, jossa voi käyttää säännöllisiä lausekkeita. WHERE-lauseen käyttö kyselyssä ei

ole pakollista. PGQL-kyselykielessä loogista tai-operaatiota (OR) käytetään WHERE-lauseessa. [Sharma *et al.* 2021] PGQL:n säännöllisten lausekkeiden syntaksi on peritty Javan Pattern-luokalta. Ne muuttujat ja ominaisuudet, jotka kyselystä halutaan palautuvan, kirjoitetaan SELECT-lauseeseen, joka toimii PGQL-kyselykielen projektiona. Jos halutaan, ettei kyselyn tulokset sisällä duplikaatteja, voidaan käyttää SELECT DISTINCT -lausetta. [PGQL 2023]

Esimerkki kysely PGQL:lla:

```
SELECT h.nimi, o.ostovuosi, a.merkki
FROM MATCH (h:Henkilö) -[o:omistaa]-> (a:Auto)
WHERE h.nimi = 'Tiina', jossa
```

haetaan kaikki Tiina-nimiset henkilöt, jotka omistavat auton. Kysely palauttaa henkilön nimen, joka on tässä tapauksessa aina Tiina, ostovuoden ja auton merkin. Kuten kyselystä voidaan huomata, muuttujan ominaisuuteen viitataan muodossa *muuttuja.ominaisuus*.

PGQL ei tue UNION-lausekkeen käyttöä ja se sallii vain rajoitetun konjunktiivisten kyselyiden liiton käytön. Tämä johtuu siitä, että WHERE-lausekkeessa voidaan käyttää loogista tai-operaatiota. Kyselyt, jotka sisältävät loogisen tai-operaation voidaan ilmaista konjunktiivisten kyselyiden liittona. [Sharma *et al.* 2021]

PGQL-kyselyiden paluuarvot voivat olla solmuja tai suhteita, solmujen tai suhteiden ominaisuuksia, totuusarvoja tai laskutoimitusten tuloksia. Kyselyt eivät palauta polkuja tai graafeja. Jos haluaa palauttaa solmun *n* kaikki ominaisuudet, on kysely muotoa:

```
SELECT n.*
FROM MATCH (n)
```

PGQL noudattaa homomorfismiin pohjaavaa semantiikkaa. Jos kuitenkin haluaa kyselyn noudattavan isomorfismia, voidaan se esittää seuraavalla tavalla:

```
SELECT x, y
WHERE (x) -> (y), x != y
```

PGQL käyttää polkukyselyissä sattumanvaraisten polkujen semantiikkaa. PGQL ei salli duplikaatti rivejä, jotta voidaan välttää sattumanvaraisten polkujen semantiikan käytöstä

johtuvia äärettömiä polkuja. Myös lyhimmän polun semantiikkaa on mahdollista käyttää. Koska PGQL ei salli duplikaatti rivejä käyttää se polkukyselyissä set-semantiikkaa. Graafien hahmonsovituksessa käytetään sen sijaan bag-semantiikkaa sekä homomorfismiin pohjaavaa semantiikkaa. Myös set-semantiikan käyttö hahmonsovituksessa on mahdollista. Kuten Cypherissa, duplikaatit voidaan poistaa käyttämällä kyselyssä avainsanaa DISTINCT. [Sharma *et al.* 2021]

Graafitietokantaan voidaan lisätä solmuja ja suhteita INSERT-lauseella. Yhdellä kyselyllä voidaan tehdä tietokantaan useita lisäyksiä. PGQL tukee vain suunnattujen suhteiden lisäämistä. Suhteita lisätessä myös sen suunta tulee siis ilmaista. Solmuja on mahdollista ilman niihin osoittavaa suhdetta. Solmujen ja suhteiden ominaisuuksia voidaan muuttaa UPDATE-lausekkeella. Myös tämä lause tukee useamman ominaisuuden lisäämistä samaan aikaan. Ominaisuuksia voi lisätä samassa kyselyssä sekä solmuille että suhteille. Solmuja ja suhteita voidaan poistaa käyttämällä DELETE-lauseketta. Saman entiteetin poisto useaan kertaan ei aiheuta konfliktia tai aiheuta poikkeustilannetta. Jos solmu poistetaan, myös kaikki siihen viittaavat suhteet poistetaan, joten tietokantaan ei jää suhteita, jotka eivät viittaisi mihinkään. [PGQL 2023]

Samassa kyselyssä voidaan tehdä samanaikaisesti useampi edellisessä kappaleessa esitellyistä muokkauslauseista, kuten päivittää solmua x ja samaan aikaan lisätä siihen viittaava suhde y. Kysely, jossa päivitetään ja poistetaan samaa entiteettiä, aiheuttaa poikkeuksen. Myös kysely, joka sisältää lisäys- ja poistolauseen voi aiheuttaa konflikteja, kuten tapauksessa, jossa lisättävä ominaisuus on riippuvainen ominaisuudesta, jota poistetaan. PGQL ei tue kyselyitä, joissa yksi kysely osoittaa useampaan eri graafiin. [PGQL 2023]

5.3 GSQL

GSQL on TigerGraph-alustan kyselykieli, jolla voidaan hakea tietoa ominaisuusgraafeista. GSQL sisältää vaikutteita SQL-kyselykielestä. GSQL on Turing-vahva ja se tukee sekä imperatiivista että proseduraalista ohjelmointia. [TigerGraph 2023]

GSQL-kyselyt vastaavat joiltakin osin PGQL:n kyselyiden syntaksia. Solmut ja muuttujat esitetään päinvastaisessa järjestyksessä kuin Cypher:ssa ja PGQL:ssä. Ensin esitetään solmun nimike ja sen jälkeen muuttuja, johon kyselystä palautuva solmu asetetaan. Alla on esitetty solmujen ja muuttujien kuvaaminen GSQL:llä:

```
Henkilö:h              // muuttuja h, nimike Henkilö
:s                     // muuttuja s
(henkilö|tapahtuma):n  // muuttuja n, nimike henkilö tai tapahtuma
```

Edellisissä esimerkeissä voidaan huomata eroja GSQL:n solmujen esitystavassa verrattuna jo esiteltyihin kyselykieliin. Solmun nimike asetetaan sulkeiden sisään vain, jos solmun nimikkeelle asetetaan ehto, kuten esimerkki *(henkilö|tapahtuma):n* osoittaa. Solmun nimikettä ei tarvitse esittää, mutta GSQL:n dokumentaatiossa suositellaan, että jokainen FROM-lausekkeessa esiteltävä solmu ja suhde asetettaisiin muuttujaan [TigerGraph 2023].

Myös suhteiden kuvaamisessa GSQL-kyselykielellä on eroja verrattuna kahteen jo esiteltyyn kyselykieleen. Suhde esitetään muodossa -(tyyppi>)- tai -(<tyyppi)-, jossa *tyyppi* on suhteen nimike. Suhde voidaan asettaa talteen muuttujaan, mikä on suositeltavaa. Tällöin suhde ilmaistaan muodossa -(:e)-, jossa *e* on muuttuja. GSQL tukee kahdensuuntaisia polkukyselyitä. Kahdensuuntaisuus ilmaistaan jättämällä pois suuntamerkki (< tai >). [TigerGraph 2023]

Alla on esitetty suhteiden ja muuttujien kuvaaminen GSQL:lla:

```
-(:e)-              // muuttuja e, kahdensuuntainen suhde
-(omistaa>:om)-     // muuttuja om, nimike omistaa
-(<Omistaa)-        // nimike Omistaa
-((omistaa|seuraa>):e)-  // nimike e, nimike omistaa tai seuraa
-(_)-               // kahdensuuntainen suhde, ei nimikettä tai muuttujaa
```

Suhteet esitetään sulkeiden sisällä. Suhteiden suunta ilmaistaan sulkeiden sisällä. Sulkeiden sisällä esitetään myös suhteen nimike ja muuttuja, joista nimike esitellään ensin. Suhteen suunta (< tai >) esitetään heti suhteen nimikkeen jälkeen, jonka jälkeen voidaan esitellä haluttu muuttuja. Myös suhteille voidaan asettaa ehtolause lauseen *-((omistaa|seuraa>):e)-* tapaan. Jos kyselyssä ei ole merkitystä suhteen nimikkeellä ja sitä ei haluta asettaa muuttujaan, esitetään sulkeiden sisällä alaviiva (_).

Lause *Henkilö omistaa auton* kuvattaisiin GSQL:lla seuraavasti:

```
Henkilö:h -(omistaa>)- Auto:a,
```

missä henkilösolmu tallennetaan muuttujaan *h* ja autosolmu tallennetaan muuttujaan *a*. Suhdetta *omistaa* ei tässä esimerkissä tallenneta muuttujaan.

Myös GSQL-kyselyissä voidaan esittää haluttu etäisyys kahden solmun välillä. Polun pituus voidaan määritellä myös vaihteluvälinä. Solmujen välinen etäisyys voidaan esittää muun muassa seuraavilla tavoilla:

```
:a - (e*) - :b          // polun pituus 0 tai enemmän
:a - (e>*3..5) - :b     // polun pituus minimissään 3, maksimissaan 5
```

Tietokannasta haettavat graafimallit ja navigointimallit esitetään FROM-lauseessa. Tässä lauseessa solmut ja suhteet voidaan tallentaa muuttujiin. Edellä kuvatut suhteiden ja polkujen esitykset tapahtuvat FROM-lauseessa. Myös mahdollinen polun pituuden määrittely voidaan tehdä tässä lauseessa.

GSQL-kyselykielen projektiona toimii SELECT-lause. Tässä lauseessa esitetään muuttuja, jonka kyselyn halutaan palauttavan. SELECT-lause voi sisältää vain yhden muuttujan, joka tulee ottaa FROM-lauseessa esitellyistä muuttujista. Duplikaattien poisto tuloksista onnistuu SELECT DISTINCT -lauseella. [TigerGraph 2023]

Solmujen ja suhteiden suodattaminen niiden ominaisuuksien perusteella toteutetaan WHERE-lauseessa. WHERE-lauseen käyttö kyselyssä ei ole pakollista. GSQL-kyselykielessä loogisen tai-operaation (OR) käyttö tapahtuu WHERE-lauseessa. Lauseessa voidaan käyttää aritmeettisia operaattoreita, vertailuoperaattoreita, loogisia operaattoreita, joukko-operaattoreita (*set operators*) sekä sulkeita. [TigerGraph 2023]

Esimerkki kysely GSQL:lla:

```
SELECT h
FROM Henkilö:h -(omistaa>)- Auto:a
WHERE h.nimi == ”Tiina”; , jossa
```

haetaan kaikki Tiina-nimiset henkilöt, jotka omistavat auton. Kysely palauttaa henkilösolmun *h* tiedot. GSQL-kyselyt eivät palauta yksittäisiä ominaisuuksia, joten jos kyselyn halutaan palauttavan solmun *h* ominaisuus *nimi*, tulee SELECT-lauseessa esittää solmu *h*. Kyselystä palautuu solmun *h* kaikki ominaisuudet. Solmun *h* ominaisuus *nimi* saadaan selville, hakemalla se kyselyn palauttamista tuloksista. Yllä olevasta kyselystä

voidaan huomata, että muuttujan ominaisuuteen viitataan muodossa *muuttuja.ominaisuus.*

GSQL-kyselyt toteutetaan CREATE QUERY -funktiossa, jossa kyselyn tulokset voidaan tallentaa muuttujaan. GSQL-kysely palauttaa aina joko joukon (*set*) solmuja tai taulukon. Kyselyt eivät palauta polkuja tai graafeja. Standardi GSQL-kyselyn paluuarvo on JSON-formaatissa. Kyselystä palautuvan solmu esitetään JSON-oliona, jonka avain-arvopareissa esitetään solmun ominaisuudet. Paluuarvo voi olla myös JSON-taulukko. [TigerGraph 2023]

Esimerkki GSQL kyselyfunktiosta:

```
CREATE QUERY hae_henkilo() FOR GRAPH Graafi {
    henkilo = SELECT h FROM Henkilö:h
            WHERE h.nimi == ”Tiina”;
    PRINT henkilo;
},
```

jossa kyselyfunktion nimi on *hae_henkilo* ja kysely kohdistuu graafiin nimeltä *Graafi*. Kyselyn tulos tallennetaan muuttujaan *henkilo*.

GSQL tukee joukko-operaatioissa sekä bag- että set-semantiikan käyttöä. SetAccum-lauseke tukee set-semantiikan käyttöä ja BacAccum bag-semantiikan käyttöä. Näiden lausekkeiden yhteydessä voidaan käyttää UNION-, INTERSECT- ja MINUS-operaatioita. Näitä operaatioita ei käytetä kyselyn sisällä, vaan operaationa kahden kyselyn tuloksille. Jos halutaan muodostaa esimerkiksi unioni joukolle *a*, joka on kyselyn k_1 tulos, ja joukolle *b*, joka on kyselyn k_2 tulos, muodostetaan unioni joukoista *a* ja *b*. Jos molemmat operoitavat joukot on saatu set-semantiikkaa käyttäen, noudattaa myös niistä muodostettava joukko sitä. Jos vähintään toinen operoitava joukko on saatu bag-semantiikkaa käyttäen, noudattaa tulosjoukko bag-semantiikkaa. Tässä tapauksessa set-semantiikkaa noudattavaa joukkoa käsitellään bag-semantiikkaa noudattaen. [TigerGraph 2023]

GSQL noudattaa kyselyissä lyhimmän polun semantiikkaa. Jos lyhimpiä polkuja on useampi, palautetaan niistä kaikki. [Deutsch *et al.* 2019] Kyselyissä on mahdollista käyttää sekä homomorfismiin, että isomorfismiin pohjaavia semantiikkoja. Tulokset

käsitellään bag-semantiikkaa noudattaen, mutta DISTINCT-avainsanalla voidaan toteuttaa myös kysely, joka noudattaa set-semantiikkaa. [TigerGraph 2023].

GSQL tukee solmujen ja suhteiden sekä näiden molempien ominaisuuksien muokkauslauseita, joissa käytetään lisäys-, poisto- tai päivityslausekkeita [Deutsch *et al.* 2019]. Lisäyslausekkeena toimivalla INSERT INTO -lauseella voidaan graafiin lisätä solmuja ja suhteita. Suhdetta lisätessä sen molemmissa päissä olevat solmut täytyy olla jo olemassa. Solmut voivat olla tietokannassa jo ennen kyselyä, jossa suhde lisätään, tai ne voidaan luoda samassa kyselyssä, mutta ennen suhteen lisäämistä. INSERT INTO -lausetta voidaan käyttää kyselystä erillisenä omana lauseenaan tai kyselyn sisällä olevana lauseena. [TigerGraph 2023].

UPDATE-lausekkeella voidaan päivittää solmujen ja suhteiden ominaisuuksia ja sitä voi käyttää vain kyselyn sisällä olevana lausekkeena. Solmut tai suhteet, joita halutaan muokata, esitellään FROM-lauseessa. FROM-lauseen jälkeen, esitetään valitun solmun tai suhteen ominaisuus ja sen uusi arvo SET-lausekkeessa. SET-lauseessa voidaan muokata vain niiden solmujen ja suhteiden ominaisuuksia, jotka on esitelty FROM-lauseessa. SET-lauseen jälkeen voidaan käyttää WHERE-lauseketta, jos muokattavia solmuja tai suhteita halutaan suodattaa. [TigerGraph 2023] UPDATE-lause, jossa käytetään WHERE-lauseketta, voidaan esittää seuraavalla tavalla:

```
UPDATE h FROM Henkilö:h
SET h.kotikunta= ”Tampere”
WHERE h.kotikunta == ”Turku”
      AND h.nimi == ”Tiina”;, jossa
```

päivitetään niiden *Henkilö*-solmujen *h* kotikunnaksi *Tampere*, joiden nimi on *Tiina* ja kotikunta *Turku*.

DELETE-lausekkeella voidaan poistaa valittuja solmuja ja suhteita. Lausetta voi käyttää vain kyselyn sisällä. GSQL:n DELETE-lausekkeella on kaskadivaikutus. Jos solmu poistetaan, myös kaikki siihen viittaavat suhteet poistetaan automaattisesti. Myös poistolausekkeen yhteydessä solmut tai suhteet, jotka halutaan poistaa, esitellään FROM-lauseessa. DELETE-lausekkeen yhteydessä FROM-lauseessa ei voida esittää kuin suhteita, joiden polun pituus on yksi. FROM-lauseen jälkeen voidaan käyttää WHERE-lauseketta. [TigerGraph 2023]

5.4 Gremlin

Gremlin on Apache TinkerPop -alustan kyselykieli ominaisuusgraafeille. Gremlin on funktionaalinen kieli, joka mahdollistaa monimutkaisten kyselyiden ja polkualgoritmien suorittamisen. Jokainen Gremlin kysely muodostuu sarjasta eri vaiheita. Jokainen vaihe suorittaa jonkin toiminnon tietovirralle. Jokaisella vaiheella voi muuntaa tietovirrassa olevia olioita, suodattaa niitä tai suorittaa laskennallisia toimenpiteitä. [The Apache Software Foundation 2023b] Gremlin on suunnattu enemmän graafeissa kulkemiseen, kuin graafien hahmonsovitukseen ja se on tässä tutkielmassa aiemmin esiteltyjä kyselykieliä funktionaalisempi [Angles *et al.* 2017]. Ominaisuusgraafien lisäksi Gremlin-kyselykielellä on mahdollista tehdä kyselyjä myös RDF-graafeihin [Thakkar *et al.* 2018] ja sillä voidaan suorittaa kyselyitä useisiin eri graafitietokantoihin [Rodriguez 2015].

Gremlin kysely alkaa tyypillisesti seuraavasti:

g.V(), joka

palauttaa listan kaikista graafin *g* solmuista. Muuttujan *g* tilalla voitaisiin käyttää mitä tahansa merkkiä tai merkkijonoa, mutta yleinen käytäntö on käyttää tätä merkkiä. Tämä muuttuja viitaa graafiin, johon kyselyä ollaan suorittamassa. Funktio *V()* hakee solmuja. Tämän tilalla voidaan käyttää myös funktiota *E()*, joka hakee kaikki graafin suhteet. Näiden tilalla voidaan käyttää myös konfiguraatiovaihtoehtoja, jotka toimivat kyselyn ohjeistuksena sille, kuinka graafissa kulkeminen toteutetaan. Nämä funktiot alkavat aina *with*-sanalla. Kaikki näistä kolmesta eri vaihtoehdosta aloittaa graafissa kulkemisen ja tietovirran syntymisen. [The Apache Software Foundation 2023b]

Graafissa voidaan kulkea solmuista kaariin muun muassa seuraavin tavoin:

```
g.V().outE(kaarinimike)   // siirrytään solmusta pois osoittavaan kaareen
g.V().inE(kaarinimike)    // siirrytään solmuun osoittavaan kaareen
g.V().both(kaarinimike)   // siirrytään sekä solmuun osoittavaan että siitä pois
                             osoittavaan kaareen
```

Edellä esitetyt kyselyn vaiheet ovat perustavanlaatuinen osa Gremliniä. Näiden vaiheiden avulla kyselyssä pystytään ”liikkumaan” graafissa. Gremlin tukee kahdensuuntaisia polkukyselyitä. Kaarista solmuihin voidaan liikkua muun muassa seuraavasti:

g.V().outE(”omistaa”).inV(), jossa

haetaan ne solmut, joihin *omistaa*-kaari osoittaa. Kyselyssä haetaan ensin graafin g kaikki solmut, jonka jälkeen siirrytään kaareen *omistaa*, jos se osoittaa solmusta poispäin. Tämän jälkeen haetaan ne solmut, joihin kaari *omistaa* osoittaa.

Kyselyn tulosta voidaan suodattaa *has()*-lauseella. Tällä lauseella voidaan suodattaa kaaria, solmuja ja solmujen ominaisuuksia. *Has()*-lauseessa määritellään, mitä ominaisuuksia kyselyyn kohdistuvien solmujen tai kaarien halutaan sisältävän. Ne solmut tai kaaret, jotka eivät sisällä vaadittuja ominaisuuksia, karsitaan pois [The Apache Software Foundation 2023b]. *Has()*-lauseesta on lukuisia eri variaatioita, joista esitellään seuraavaksi kolme esimerkkiä:

g.V().hasLabel('henkilö'), jossa

haetaan graafista *g* ne solmut, joiden solmunimike on *henkilö*.

g.V().hasLabel('henkilö','nimi','Tiina'), jossa

haetaan graafista *g* ne solmut, joiden solmunimike on *henkilö* ja joilla on ominaisuus *nimi*, jonka arvo on *tiina*.

g.V().has('henkilö', 'nimi', within('Tiina, 'Mikko')), jossa

haetaan graafista *g* ne solmut, joiden solmunimike on *henkilö* ja joilla on ominaisuus *nimi*, jonka arvo on *Tiina* tai *Mikko*. *Within()*-lauseke ei toimi samaan tapaan kuin *has()*-lause, sillä se ei suodata ominaisuuksia. *Within()*-lauseketta käytetään selkeyttämään kyselyä ja siinä käytettäviä ilmaisuja, jotta kyselystä saadaan helpommin luettavaa ja ylläpidettävää. [The Apache Software Foundation 2023b]

Aiemmin esiteltyjen kyselykielten esimerkeissä on haettu solmu, jonka nimike on *henkilö* sekä nimiominaisuus *Tiina* ja josta poispäin osoittaa nimikkeellä *omistaa* oleva kaari, joka osoittaa solmuun, jonka nimike on *auto*. Gremlinillä tämän haun voisi suorittaa useammallakin eri tavalla. Alla on esitetty yksi esimerkki, kuinka kyseinen haku voidaan suorittaa:

g.V().hasLabel(’henkilö’,’nimi’,’Tiina’).outE(’omistaa’).inV().
hasLabel(’auto’)

Seuraavassa esimerkissä esitellään tarkemmin graafissa kulkemista, joka voi tapahtua esimerkiksi seuraavalla tavalla:

g.V(1).repeat(out().simplePath()).until(hasId(5)).path().limit(1), jossa

graafissa *g* kulkeminen aloitetaan solmusta, jonka identifioiva tunniste on *1*. Kulkeminen tapahtuu solmuista ulospäin (*out()*) osoittavien kaarien kautta ja polkua jatketaan eteenpäin niin kauan, että vastaan tulee solmu, jonka identifioiva tunniste on *5*. *SimplePath()*-lausekkeella rajataan pois polkujen toistot ja *path()*-lauseke palauttaa kaikki ne solmut, joiden kautta polku on kulkenut. Käyttämällä *limit()*-lauseketta, kysely palauttaa vain yhden polun, joka on lyhin löydetty polku. [The Apache Software Foundation 2023b]

Duplikaattien poisto kyselyn tuloksista onnistuu *dedup()*-lauseen avulla. Jos oliot toistuvat useamman kerran datavirrassa, tämä lause suodattaa ne pois. Gremlinissä on käytettävissä union()-lauseke, joka toimii tulosjoukkojen yhdisteen muodostamisessa. Gremlinin SugarGremlinPlugin-luokka tarjoaa vertailuoperaattorit *ja* (*&*) sekä *tai* (|). Gremlinissä on käytettävissä myös lausekkeet *or()* ja *and()*, jotka toimivat vertailuoperaattoreiden tapaan. [The Apache Software Foundation 2023b] *Ja*-operaattoria ja *and()*-lauseketta voidaan käyttää esimerkiksi seuraavilla tavoilla:

```
g.V().where(outE('omistaa') & outE('seuraa')).name
g.V().where(outE('omistaa').and().outE('seuraa')).values('name')
```

Yllä esitetyt kyselyt käyttävät loogista ja-operaatiota kahdella eri tapaa. Kyselyt palauttavat saman vastauksen, eli ominaisuuden *nimi* arvot niiltä solmuilta, jotka täyttävät annetut vaatimukset niistä poispäin osoittaville kaarille.

Graafimallien arvioimiseen Gremlin käyttää homomorfismiin pohjaavaa semantiikkaa. Tulosten käsittelyyn käytetään bag-semantiikkaa, mutta *dedup()*-lauseen avulla on mahdollista käsitellä kyselyn tulokset myös set-semantiikkaa noudattaen. Polkukyselyt noudattavat sattumanvaraisten polkujen semantiikkaa. Myös polkukyselyiden tulokset käsitellään bag-semantiikkaa noudattaen tai *dedup()*-lauseen avulla set-semantiikan mukaisesti. Gremlin kyselyiden paluuarvot voivat olla solmuja tai polkuja. [Angles *et al.* 2017]

Graafitietokantaan voidaan lisätä solmuja, kaaria ja ominaisuuksia. Solmujen lisääminen tapahtuu lauseella *addV()*, kaarien lisääminen lauseella *addE()* ja ominaisuuksien lisääminen lauseella *property()*. Ominaisuuksia lisäävää *property()*-lausetta voidaan käyttää kaarien ja solmujen lisäyslauseiden yhteydessä. Ominaisuuksia voidaan lisätä myös jo olemassa oleviin solmuihin ja kaariin. *Property()*-lauseella voidaan myös päivittää jo olemassa olevia ominaisuuksia. [The Apache Software Foundation 2023b] Alla on esitetty esimerkkejä näiden kolmen lisäyslauseen käytöstä:

```
g.addV('henkilö').property('nimi','Tiina'), jossa
```

lisätään solmu, jonka nimike on *henkilö* ja jolla on ominaisuus *nimi*, jonka arvo on *Tiina*.

```
vTiina = g.V().has('nimi','Tiina').next()
vMikko = g.V().has('nimi','Mikko).next()
g.addE('seuraa').from(vTiina).to(vMikko), jossa
```

ensin haetaan ja tallennetaan solmut *Tiina* ja *Mikko* muuttujiin. Tämän jälkeen lisätään lauseella *addE()* kaari *seuraa* näiden kahden solmu välille niin, että kaari osoittaa solmusta *Tiina* solmuun *Mikko*.

Solmuja, kaaria ja niiden ominaisuuksia voidaan poistaa *drop()*-lauseella. Esimerkiksi kaikki solmut, joiden nimike on auto, voidaan poistaa seuraavasti:

```
g.V().hasLabel('auto').drop()
```

Drop()-lause ei palauta mitään. Kaaret voidaan poistaa kahdella eri tavalla. Jos kaaren alku- tai loppupäässä olevan solmun poistaa, poistetaan automaattisesti myös kaikki siihen viittaavat kaaret. Haluttu kaari voidaan myös poistaa ilman, että solmuja poistetaan. Tällöin määritellään poistettavan kaaren identifioimiseen tarvittavat määreet ja kun kyseinen kaari löydetään, se poistetaan tietokannasta. [Bechberger *et al.* 2020]

5.5 SPARQL

SPARQL on kyselykieli ja protokolla RDF-graafeille. Apache Jenan SPARQL dokumentaatiossa (2023) kuvataan kyseistä kyselykieltä datapainotteiseksi, sillä se suorittaa kyselyitä vain datamalleissa olevaan informaatioon. SPARQL ei tee muuta kuin hakee kyselyssä määritellyn informaation ja palauttaa sen joko joukkoina tai RDF-graafin muodossa. SPARQL-kyselykielen on suunnitellut W3C:n RDF Data Access -työryhmä.

Luvussa 3.2 kerrottiin RDF-graafeista ja siitä, kuinka ne voivat noudattaa IRI-standardia. Koska SPARQL on kyselykieli RDF-graafeille, täytyy kyselyissä määritellä nimiavaruus, jossa vakiot esiintyvät. SPARQL-kyselyssä nimiavaruus voidaan esittää PREFIX-lausekkeessa ennen varsinaista kyselyä. Tämä lyhentää kyselyä, sillä nimiavaruutta ei tarvitse toistaa kyselyn eri vaiheissa. [Angles *et al.* 2017] Tässä tutkielmassa PREFIX-lauseen määritteleminen ennen kyselyä ei ole olennaista, joten se on jätetty pois tässä kappaleessa esiteltävistä SPARQL esimerkkikyselyistä.

SPARQL-kyselyiden olennaisena rakennuspalikkana toimivat kolmiosaiset mallit, joissa esitetään RDF-graafin subjekti-predikaatti-objekti malli. SPARQL-kyselyissä käytetään tyypillisesti kysymysmerkkiä (?) muuttujien nimien edessä. Muuttujat, jotka kyselyn halutaan palauttavan, esitetään SELECT-lauseessa. SELECT-lause palauttaa solmujen tulosjoukon taulukossa [The Apache Software Foundation 2023a]. SELECT-lauseen yhteydessä voidaan käyttää DISTINCT-avainsanaa, joka poistaa duplikaatit tulosjoukosta [World Wide Web Consortium 2023]. Graafimalli, jota muuttujien halutaan noudattavan, esitetään WHERE-lauseessa [Angles *et al.* 2017]. WHERE-lauseessa voidaan käyttää String-arvojen testaamiseen säännöllisiin lausekkeisiin pohjautuvia operaatioita [The Apache Software Foundation 2023a].

```
SELECT ?h
WHERE {
    ?h :omistaa ?a . ?h :type :Henkilö .
    ?a :type :Auto . ?h :nimi "Tiina"
}, jossa
```

muuttujalla *h* on suhde *omistaa* muuttujaan *a*, muuttujan *h* tyyppi on *Henkilö*, muuttujan *a* tyyppi on *Auto* ja muuttujan *h* nimi on *Tiina*. Kysely sisältää neljä kolmiosaista mallia. Mallit erotetaan toisistaan pisteellä.

SPARQL tarjoaa laajan skaalan muun muassa suodatusilmailuja (FILTER), aritmeettisia operaatioita, ehtolauseita ja yhdistämislausekkeita. Nämä operaatiot ja lausekkeet toteutetaan WHERE-lauseen sisällä. SPARQL käyttää graafimallien arvioimiseen homomorfismiin perustuvaa semantiikkaa, mikä on hyvä ottaa huomioon tietyn tyyppisissä kyselyissä. Kahden solmun eriarvoisuus voidaan ilmaista WHERE-lauseen sisällä seuraavasti:

```
FILTER(?x != ?y)
```

SPARQL sisältää neljä eri kyselytyyppiä. Kaikki näistä kyselytyypeistä käyttävät kyselyn lopputuloksen muodostamiseen hahmonsovitusta. Lopputulos on kyselytyypin mukaan joko tulosjoukko, totuusarvo tai RDF-graafi. Näistä kyselytyypeistä ensimmäinen on SELECT, joka kuvattiin edellisessä kappaleessa. CONSTRUCT-kysely palauttaa RDF-graafin, joka on muodostettu ennalta määrätyn graafipohjan mukaisesti. ASK-kysely palauttaa boolean-totuusarvon. Tällä kyselyllä voidaan selvittää, täsmääkö kyselyssä esitetty malli graafitietokantaan. SPARQL palauttaa arvon ”*yes*” tai ”*no*”, riippuen siitä, täsmääkö haettu malli. Viimeisenä kyselytyyppinä on DESCRIBE-kysely, joka palauttaa RDF-graafin. Tämä graafi sisältää RDF-dataa kyselyssä annetusta lähteestä. Graafin sisältämää dataa ei määritetä kyselyssä, sillä DESCRIBE-kyselyssä sen määrittää SPARQL-kyselyprosessori. [World Wide Web Consortium 2023] SPARQL-kyselyillä on mahdollista tehdä samanaikaisesti kyselyitä useampaan eri graafiin. [Angles *et al.* 2017].

Kahden joukon unioni on mahdollista muodostaa WHERE-lausekkeessa UNION-avainsanan avulla. UNION-avainsanan käyttö tapahtuu seuraavasti:

```
SELECT DISTINCT ?e
WHERE {
    { :Tiina_Lymi :näyttelee ?e . }
    UNION
    { :Tiina_Lymi :ohjaa ?e . }
}, jossa
```

haetaan ne elokuvat *e*, joissa *Tiina_Lymi* näyttelee ja jotka hän on ohjannut. Molemmat kyselyssä esitettävät mallit haetaan toisistaan riippumattomasti ja niiden tulosjoukoista muodostetaan unioni [Angles *et al.* 2017]. Kyselyssä on käytetty myös DISTINCT-avainsanaa, jolla voidaan poistaa tulosjoukosta mahdolliset duplikaattiarvot.

Kahden argumentin erotuksena voidaan käyttää MINUS-avainsanaa. Erotus toimii niin, että sen molemmat mallit arvioidaan ensin erikseen, minkä jälkeen valitaan ne vasemmanpuoleisen mallin tulosjoukot, joita ei löydy oikeanpuoleisen mallin tulosjoukoista. Jos kyselyn halutaan palauttavan myös sellaisia tulosjoukkoja, jotka eivät kaikilta osin noudata graafimallille annettuja vaatimuksia, voidaan kyselyssä käyttää

OPTIONAL-avainsanaa. Jos OPTIONAL-lausekkeessa esitettävä ehto ei toteudu haettavalle objektille, palautetaan se silti, jos se täyttää kyselyn muut ehdot. [World Wide Web Consortium 2023]

SPARQL-kyselykielen yhteydessä käytetään termiä ominaisuuspolku (*property path*), jolla tarkoitetaan säännöllisiä polkukyselyitä. Ominaisuuspolku on mahdollinen reitti kahden solmun välillä. Ominaisuuspolut mahdollistavat osan SPARQL:n graafimallien ilmaisemisen tiiviimmässä muodossa. Tämän lisäksi ominaisuuspolut lisäävät mahdollisuuden sattumanvaraisten polkujen sovittamisen. SPARQL tukee kahdensuuntaisia polkukyselyitä. Lisäksi polkukyselyissä on mahdollista käyttää loogisia or-operaatiota (|). [World Wide Web Consortium 2023] Haettavan polun pituus voidaan määritellä muun muassa seuraavin tavoin:

```
?a :e* ?b                  // polun e pituus 0 tai enemmän
?a :e{3,5} ?b              // polun e pituus minimissään 3, maksimissaan 5
```

Käänteinen polku voidaan esittää SPARQL-kyselykielellä seuraavasti:

```
SELECT ?h1
WHERE {
        ?h1 ^:seuraa ?h2, jossa
}, jossa
```

haetaan ne *h1*, joita *h2 seuraa*.

SPARQL-kyselyt voivat palauttaa joko solmuja, totuusarvon tai graafin. Graafimallien arvioimisessa eli hahmonsovituksessa käytetään homomorfismiin perustuvaa semantiikkaa ja kyselyn tulokset käsitellään bag-semantiikkaa noudattaen. Tulokset on mahdollista käsitellä myös set-semantiikan noudattaen DISTINCT-avain sanan avulla. Polkukyselyissä käytetään sattumanvaraisten polkujen semantiikkaa ja tulokset käsitellään set-semantiikkaa noudattaen. [Angles *et al.* 2017]

Tietokantaan voidaan lisätä dataa INSERT-lauseella, joka toimii myös datan päivittämisessä. INSERT-lauseella voidaan lisätä tietokantaan kolmiosaisia graafimalleja, jotka määritellään WHERE-lauseessa. [World Wide Web Consortium 2023] Saman kaavan mukaan voidaan tehdä myös päivitysoperaatioita, kuten seuraava esimerkki osoittaa:

```
INSERT {?p :nimi 'Tiina' }
WHERE {
        ?p :nimi 'Maija'
 }, jossa
```

solmun *p*, jonka ominaisuuden *nimi* arvo on *Maija*, päivitetään arvoksi *Tiina*. SPARQL sisältää myös INSERT DATA -lauseen, jonka yhteydessä ei voida käyttää muuttujia eikä WHERE-lauseketta. [World Wide Web Consortium 2023]

Poisto-operaatiot suoritetaan DELETE-lauseella. DELETE-lauseella voidaan poistaa tietokannasta kolmiosaisia graafimalleja, jotka määritellään WHERE-lauseessa. SPARQL-sisältää myös poistolauseen DELETE DATA, jonka avulla voidaan myös poistaa kolmiosoisia graafimalleja. Erotuksena tällä lauseella on se, että DELETE-operaation yhteydessä on sallittua käyttää muuttujia sekä tyhjiä solmuja, kun taas DELETE DATA -operaation kanssa tämä ei ole mahdollista. Lisäksi DELETE DATA -operaation yhteydessä ei voi käyttää WHERE-lauseketta. [World Wide Web Consortium 2023]

5.6 G-CORE

G-CORE on suljettu kyselykieliehdotelma ominaisuusgraafeille. Kyselykieliehdotelmaa on ollut työstämässä LDBC Graph Query Language -työryhmä, johon kuuluu jäseniä sekä teollisuudesta että akateemisesta maailmasta. G-CORE:n ei ole tarkoitus toimia uutena standardina, vaan sen luomisen tavoitteena on ollut ohjata graafitietokantojen kyselykielien kehittymistä käytännöllisemmiksi, tehokkaammiksi sekä ilmaisuvoimaisemmiksi. G-CORE on luotu alan toiveita sekä tarpeita ajatellen ja sen tärkeimmät toiminnot on valittu teoriaan pohjaavan tutkimuksen sekä alan käytäntöjen mukaisesti. [Angles *et al.* 2018]

Jokainen G-CORE-kysely alkaa CONSTRUCT-lausekkeella. Tämä lauseke mahdollistaa sen, että jokaisen kyselyn tuloksena palautuu graafi. G-CORE-kyselyiden visuaaliset muodostussäännöt pohjautuvat Cypheriin. CONSTRUCT-lauseketta seuraa MATCH-lauseke, jossa solmut ja kaaret voidaan sitoa muuttujiin. CONSTRUCT-operaatio täyttää graafimallin jokaiselle MATCH-lauseen sitomalle muuttujalle. MATCH-lauseen jälkeen voidaan käyttää ON-lauseketta, jossa esitellään se graafi, josta dataa halutaan hakea. ON-lausekkeen käyttö ei ole pakollista, jos on määritelty oletusgraafi, josta tieto haetaan.

Solmuja ja kaaria voidaan suodattaa niiden ominaisuuksien perusteella WHERE-lausekkeessa. [Angles *et al.* 2018]

Esimerkkikysely G-CORE-kyselykielellä:

```
CONSTRUCT (h)
  MATCH (h:Henkilö)
    ON esimerkki_graafi
    WHERE h.nimi='Tiina', jossa
```

haetaan ne solmut *h*, joiden nimike on *Henkilö* ja joilla on ominaisuus *nimi*, jonka arvo on *Tiina*. CONSTRUCT-lauseke muodostaa löydetyistä solmuista tulosgraafin, joka tässä tapauksessa ei sisällä kaaria. Kysely palauttaa solmun *h* kaikki ominaisuudet, jotka sillä on graafissa *esimerkki_graafi*.

G-CORE-kyselykielessä suhteiden ja polkujen esittäminen eroavat toisistaan syntaksiltaan. Suhteiden kuvaaminen vastaa Cypherin visuaalisia muodostussääntöjä. [Angles *et al.* 2018] Suhde solmujen a ja b välillä voidaan esittää muun muassa seuraavilla tavoilla:

```
(a) -[:suhde]-> (b)        // suhteen nimike suhde
(a) <-[s:suhde]- (b)       // muuttuja s, suhteen nimike suhde
(a) -[]- (b)               // kaksisuuntainen suhde, suhteen nimike vapaa
```

Säännöllisten polkujen ilmaisemisen syntaksiin on haettu mallia PGQL-kyselykielestä. Polkuilmaukset kirjoitetaan kauttaviivojen -/ /- sisälle. Poluille voidaan asettaa nimikkeitä ja ominaisuuksia. [Angles *et al.* 2018] Polku solmujen h1 ja h2 välillä voidaan esittää seuraavalla tavalla:

```
(h1:Henkilö) -/s<:seuraa>/-> (h2:Henkilö), jossa
```

-/s<:seuraa>/-> on polkuilmaisu, joka on kirjoitettu merkkien -/ /-> väliin. Polun suunta voidaan ilmaista merkillä > tai <. Suhdenimike, jonka omaavia suhteita polun tulee seurata, määritellään merkkien < > väliin.

G-CORE-kyselykielessä polut ovat niin kutsuttuja ensimmäisen luokan kansalaisia (*first class citizens*). Tämä tarkoittaa, että poluille voidaan toteuttaa samoja toimintoja kuin

solmuille ja kaarille. Poluille voidaan asettaa erilaisia ehtoja, määreitä ja aggregointioperaatioita. Seuraavassa esimerkissä esitellään G-CORE:n polkukyselyissä käytettäviä toiminnallisuuksia.

```
CONSTRUCT (h1) -/@p:seurattavat{etäisyys:=c}/-> (h2)
MATCH (h1) -/3 SHORTEST p<:seuraa*> COST c/-> (h2)
WHERE (h1:Henkilö) AND (h2:Henkilö)
AND h1.nimi='Tiina' AND h2.kotikunta= 'Tampere'
```

Yllä olevassa kyselyssä *p<:seuraa*>* sitoo lyhimmän polun solmun *h1* (*Tiina*) ja jokaisen Henkilö-solmun *h2* välillä, jos henkilön *h2* kotikunta on *Tampere*. Avainsanan SHORTEST edessä oleva numero *3* tarkoittaa, että graafista voidaan hakea enintään kolme lyhintä polkua. Jos numeromääre jätetään kyselystä pois, palauttaa se vain yhden polun, joka on lyhin löydetty polku. MATCH-lauseessa käytettävä avainsana COST laskee polun pituuden, joka tallennetaan muuttujaan *c*. COST-avainsanan käyttö ei ole pakollista. CONSTRUCT-lauseessa käytettävällä @-merkillä ilmaistaan, että kysely palauttaa graafin, joka sisältää polkuja, jotka on sidottu muuttujaan *p*. Löydetyt polut saavat polkunimikkeen *seurattavat*, jolla on ominaisuus, jonka nimi on *etäisyys*. Ominaisuuteen *etäisyys* tallennetaan polun pituus *c*. Kuten kyselystä voidaan huomata, voidaan myös polkukyselyiden yhteydessä käyttää WHERE-lausetta. [Angles *et al.* 2018]

Polkukyselyissä on mahdollista käyttää myös ALL-avainsanaa, jolla voidaan hakea kaikki polut. Tätä avain sanaa ei voida käyttää, jos polku on sidottu muuttujaan ja muuttujaa käytetään kyselyssä. Tämä johtuu siitä, että tuloksia voi olla ääretön määrä, mikä vuoksi niiden käsittelystä tulisi haastavaa. Jos polkuun sidottua muuttujaa käytetään CONSTRUCT-lausekkeessa vain graafin palauttamiseen, on ALL-avainsanan käyttö sallittua. [Angles *et al.* 2018]

G-CORE mahdollistaa OPTIONAL-lausekkeen käytön. Jos OPTIONAL-lausekkeessa on useampia pilkuilla toisistaan erotettuja ehtoja, täytyy jokaisen niistä toteutua. Yhdessä kyselyssä voi olla useampi OPTIONAL-lauseke ja jokainen niistä voi sisältää oman WHERE-lausekkeen. OPTIONAL-lauseke vastaa SQL-kyselykielen vasenta ulkoliitosta, jossa MATCH-lauseessa esitetty graafimalli voidaan nähdä vasemmanpuoleisena taulukkona. [Angles *et al.* 2018]

Kyselyitä on mahdollista tehdä samanaikaisesti useampaan graafiin, mikä mahdollistaa datan integroimisen. Useamman graafin lisääminen kyselyyn toteutetaan MATCH-lauseen avulla seuraavasti:

```
CONSTRUCT (y) <-[:työntekijät]- (h)
MATCH (y:Yhtiö) ON yhtiö_graafi,
        (h:Henkilö) ON henkilö_graafi
WHERE y.nimi= h.työnantaja
UNION henkilö_graafi, josta
```

palautuu graafi, johon on integroitu henkilö- ja yhtiötietoja graafeista yhtiö_graafi ja henkilö_graafi. Kyselyssä haetaan yhtiölle *y* kaikki ne henkilöt *h*, joiden työnantaja yhtiö *y* on. Kyselystä voidaan huomata, että G-CORE-kyselyissä voidaan käyttää UNION-lausetta. Tämän lisäksi INTERSECT- (leikkaus) ja MINUS-lausekkeet (erotus) ovat käytettävissä. [Angles *et al.* 2018]

Graafimallien hahmonsovituksessa G-CORE käyttää homomorfismiin perustuvaa semantiikkaa ja tulostenkäsittelyssä set-semantiikkaa [Angles *et al.* 2017, G-CORE]. Polkukyselyissä käytetään lyhimmän polun semantiikkaa. [Angles *et al.* 2018] Koska G-CORE-polkukyselyiden tuloksena ALL-avainsanaa käytettäessä voi olla äärettömiä polkuja, käyttää se tulostenkäsittelyssä bag-semantiikkaa.

Poisto-operaationa toimii REMOVE-lauseke ja lisäys- sekä päivitysoperaationa SET-lauseke. Lisäys- ja päivitysoperaatiot suoritetaan ennen poisto-operaatioita. [Angles *et al.* 2017, G-CORE] G-CORE on vasta kyselykieliehdotelman tasolla ja vuoden 2018 konferenssijulkaisu on viimeisin LDBC Graph Query Language -työryhmän julkaisema kirjoitus aiheesta.

6 GQL ja SQL/PGQ

Ominaisuusgraafeille ollaan työstämässä kyselykielistandardia, joka on nimeltään GQL. GQL-projekti on nimetty toteutettavaksi ISO/IEC JTC1 SC32 WG3 Database Languages -työryhmälle, joka on kehittänyt myös SQL-kyselykielistandardin. ISO-komitean alkutavoitteena on luoda standardoitu kyselykieli graafitietokannoille. [Angles *et al.* 2021] GQL-projekti täydentää suunnitteilla olevaa SQL-laajennusta, jossa määritellään, kuinka graafinäkymät voidaan määrittää SQL-taulukkoskeemalla ja kuinka niihin voidaan tehdä kyselyitä. Tästä SQL-laajennuksesta käytetään nimeä SQL/PGQ, jota työstää sama ISO/IEC JTC1 SC32 WG3 Database Languages -työryhmä. Sekä GQL- että SQL/PGQ-standardia luodaan yhteistyössä LDBC-työryhmän, josta mainittiin luvussa 5.6, kanssa [Deutsch *et al.* 2021]

GQL tulee sisältämään kaikki CRUD-ominaisuudet, mikä tarkoittaa, että kyselykielen avulla voidaan luoda (*create*), lukea (*read*), päivittää (*update*) ja poistaa (*delete*) dataa tietokannasta. [Deutsch *et al.* 2021]

Tässä luvussa esitettävät GQL- ja SQL/PGQ-standardien ominaisuudet on esitelty WG3- ja LDBC-työryhmien jäsenien kirjoittamassa artikkelissa vuodelta 2021. GQL ja SQL/PGQ tulevat eroamaan toisistaan muun muassa kyselytulosten käsittelyn suhteen, mutta ne tulevat jakamaan paljon yhteisiä piirteitä. Tästä syystä artikkelissa molempien kyselykielien tulevia ominaisuuksia esitellään yhteisellä ”alikielellä” GPML. GPML-kyselykielen avulla on tarkoitus tuoda esille GQL- ja SQL/PGQ-kyselykielien tulevia ominaisuuksia ennen niiden standardien julkaisua.

6.1 Syntaksi ja ominaisuudet

Haettava graafimalli esitellään MATCH-lauseessa, jota voi seurata WHERE-lauseke. MATCH-lauseessa solmut esitetään sulkeiden sisällä. Haettavan solmun solmunimike ja sen sijoittaminen muuttujaan tapahtuu sulkeiden sisällä. WHERE-lauseessa voidaan asettaa haulle ehtoja, joita voidaan yhdistää loogisilla lausekkeilla AND, OR ja NOT. Esimerkiksi haku, jossa haetaan kaikki solmut *h*, joiden solmunimike on *Henkilö*, toteutetaan seuraavasti:

```
MATCH (h:Henkilö)
```

Solmu- ja kaarinimikkeille voidaan asettaa MATCH-lauseessa ehtoja konjunktio- (&), disjunktio- (|) ja negaatio-operaattoreilla (!). Loogisten operaatioiden lisäksi GPML sisältää niin kutsun ”villikortti” symbolin %, jolla voidaan hakea muun muassa solmuja, joilla ei ole solmunimikettä. Esimerkiksi haku, jossa haetaan kaikki solmut *x*, joiden solmunimike on *Henkilö* tai *Auto*, toteutetaan seuraavasti:

MATCH (x:Henkilö|Auto)

Kaaret esitetään MATCH-lauseessa hakasulkeiden sisällä. Graafimallissa solmuja ja kaaria voidaan ketjuttaa polkumalleiksi. Solmuista ja kaarista muodostettu polkumalli, voidaan esittää GPML-kyselykielellä seuraavasti:

MATCH (h1:Henkilö)−[:Seuraa]−>()<−[:Seuraa]−(h2:Henkilö), jossa

haetaan henkilösolmut *h1* ja *h2*, jotka seuraavat samaa solmua. Tässä esimerkissä kaaria ei sidota muuttujiin, mutta se voidaan tehdä samaan tapaan kuin solmujen muuttujiin asettaminen. Solmuja tai kaaria ei ole pakollista sijoittaa muuttujiin. Polut ovat GPML-kyselykielessä ensimmäisen luokan kansalaisia.

GPML tukee kahdensuuntaisia polkukyselyitä. Polun sallittu suunta tai suuntaamattomuus voidaan esittää seitsemällä eri tavalla, jotka on esitetty alla:

<−[suhde]−	vasemmalle suunnattu
−[suhde]−>	oikealle suunnattu
<~[suhde]~	vasemmalle suunnattu tai suuntaamaton
~[suhde]~>	oikealle suunnattu tai suuntaamaton
~[suhde]~	suuntaamaton
<−[suhde]−>	oikealle tai vasemmalle suunnattu
−[suhde]−	vasemmalle suunnattu, oikealle suunnattu tai suuntaamaton

Suuntaamattomat polut esitetään muodossa ~[*suhde*]~. Jos halutaan hakea polkuja, joiden suunta on suuntaamaton tai vasen, voidaan tämä esittää muodossa <~[*suhde*]~. Vastaavasti voidaan esittää polku, jonka suunta on suuntaamaton tai oikealle osoittava. Haettaessa polkuja, joiden suunnan halutaan olevan joko oikealle tai vasemmalle, kaari esitetään muodossa <−[*suhde*]−>. Jos kaaren suunta saa olla vasemmalle, oikealle tai suuntaamaton, esitetään se muodossa −[*suhde*]−.

Kuten solmuja, myös kaaria voidaan suodattaa niiden ominaisuuksien mukaan WHERE-lauseessa. Myös kaarinimikkeille voidaan asettaa ehtoja konjunktio-, disjunktio- ja negaatio-operaattoreilla MATCH-lauseessa.

WHERE-lauseke voidaan sijoittaa MATCH-lausekkeen jälkeen tai sen sisälle solmu- ja kaarimalleihin. Kysely voi sisältää yhtä aikaa useamman WHERE-lausekkeen, kuten seuraava esimerkki osoittaa:

```
MATCH k = (h:Henkilö WHERE h.nimi='Tiina')
        −[o:Omistaa WHERE o.ostovuosi>2020]−>(:Auto), jossa
```

haetaan ne henkilösolmut, joiden nimi on *Tiina* ja jotka omistavat auton, joka on ostettu aikaisintaan vuonna 2021. Kuten esimerkkikyselystä voidaan huomata, kysely sisältää kaksi WHERE-lauseketta, jotka molemmat ovat MATCH-lausekkeen sisällä. Esimerkkikysely sisältää myös mallin siitä, kuinka koko polku voidaan sitoa muuttujaan *k*. Tämä polkujen sitominen mahdollistaa monimutkaisten polkukyselyiden esittämisen.

Kuten monet muut graafitietokantojen kyselykielet, myös GPML mahdollistaa haettavien polkujen pituuden määrittelyn. Tämä voidaan toteuttaa MATCH-lauseessa, kaarimallin esittämisen jälkeen, muun muassa seuraavilla tavoilla:

```
−[:seuraa]−>{2,5}        // polun pituus minimissään 2, enintään 5
−[:seuraa]−>*            // polun pituus 0 tai enemmän
```

GPML sisältää kaksi erilaista unionioperaatiota. Ensimmäinen näistä on niin kutsuttu polkumalliunioni (*path pattern union*), jonka esitystapa on pystyviiva eli |-merkki. Polkumalliunioni muodostaa unionin joukkoista set-semantiikkaa käyttäen. Toinen unionioperaatio on nimeltään monisarjavuorottelu (*multiset alternation*), joka muodostaa joukkojen unionin multiset-semantiikkaa käyttäen. Monisarjavuorottelun esitystapa on muotoa: |+|. Nämä kaksi eri unionioperaatiota tuottavat tietyissä tilanteissa eri lopputulokset, sillä polkumalliunioni karsii duplikaatit pois tulosjoukosta.

Vaihtoehtoiset ominaisuudet voidaan esittää kysymysmerkkioperaattorilla seuraavasti:

```
MATCH (x) [−>(y)]?, jossa
```

Haetaan solmut *x*, joilla voi solmuun *y* osoittava suhde. Kysely palauttaa myös solmut *x*, joilla ei ole suhdetta solmuun *y*. Kysymysmerkkioperaattori vastaa monissa aiemmin tässä tutkielmassa esitellyissä kyselykielissä olevaa OPTIONAL-lauseketta.

GPML-kyselykielessä voidaan asettaa ehto sille, minkä semantiikan mukaan polkukyselyitä rajoitetaan. Näitä poluille annettavia ehtoja kutsutaan rajoittimiksi (*restrictors*) ja niiden avulla voidaan varmistaa, että polun pituus ei voi olla ääretön. GPML sisältää kolme eri polkurajoitinta. TRAIL-rajoitin noudattaa semantiikkaa, jossa yhden kaaren kautta saa kulkea vain kerran *(no-repeated-edge semantics)* ja ACYCLIC-rajoitin noudattaa semantiikkaa, jossa yhdessä solmussa saa käydä vain kerran *(no-repeated-node semantics)*. Kolmas rajoitin on nimeltään SIMPLE, joka noudattaa myös semantiikkaa, jossa yhdessä solmussa saa käydä vain kerran, sillä erotuksella, että polun ensimmäinen ja viimeinen solmu saa olla sama. Koska ilman rajoittimia polun pituus voi olla ääretön, käyttää GPML sattumanvaraisten polkujen semantiikkaa.

Rajoittimien lisäksi polkukyselyissä voidaan käyttää niin kutsuttuja valitsimia (*selectors*). Valitsin on algoritmi, jonka avulla voidaan hakea polku tai polkuja, jotka täyttävät valitsimen sisältämät vaatimukset haettavalle polulle. GPML sisältää kuusi valitsinalgoritmia. Esimerkiksi ANY SHORTEST-algoritmilla voidaan hakea yksi polku, jolla on lyhin pituus ja ALL SHORTEST hakee kaikki lyhimmät polut, joilla on sama pituus. GPML-kyselyille voidaan asettaa samanaikaisesti sekä rajoittimia että valitsimia.

Tulevan standardin tärkeimmät hahmonsovitusmallin vaiheet ovat normalisointi (*normalization*), laajennus (expansion), joustamaton hahmonsovitus (*rigid-pattern matching*) sekä pelkistäminen ja päällekkäisyyksien poistaminen (*reduction and deduplication*). Normalisoinnilla tarkoitetaan sitä, että GPML käyttää niin kutsuttua syntaktista sokeria (*syntactic sugar*), jonka tarkoituksena on tehdä mallien syntaksin kirjoittamisesta ja lukemisesta helpompaa. Syntaktista sokeria käytetään niin kysely- kuin ohjelmointikielissä. Eri kielien suunnittelijat käyttävät syntaktista sokeria myös supistamaan käyttäjälle näkyvää osaa ohjelmointi- tai kyselykielestä. [Pombrio *et al.* 2017] GPML syntaksin osalta syntaktisella sokerilla tarkoitetaan, että sillä muodostettavissa kyselyissä käytetään Cypher-tyylisiä ASCII-merkintöjä selkeyttämään esimerkiksi solmujen ja suhteiden esittämistä. Laajennuksella tarkoitetaan, että malli voidaan laajentaa joustamattomien mallien joukoiksi, joka ei ole unioni. Joustamaton hahmonsovitus tarkoittaa, että jokaiselle joustamattomalle mallille haetaan polkusidosten

joukko. Jokainen joustamattoman mallin peruselementti arvioidaan itsenäisesti ja lopuksi samoilla nimillä olevien muuttujien tulokset liitetään yhteen. Pelkistäminen ja päällekkäisyyksien poistaminen tarkoittaa, että joustamattomilla malleilla löydettyjä polkusidoksia pelkistetään, minkä jälkeen mahdolliset duplikaatit poistetaan, ennen hakutulosten palauttamista.

Kyselyiden paluuarvojen esitystapa riippuu siitä, tehdäänkö kysely SQL/PGQ- vai GQL-kyselykielellä. GQL-kyselyissä esitystavan on mahdollista olla monipuolisempi. Paluuarvot voidaan esittää esimerkiksi graafinäkymänä tai täysin uutena graafina. GQL-standardin ensimmäinen julkaisu tulee olemaan kuitenkin linjassa SQL/PGQ-standardin esitystapaan nähden. On silti ennakoitavissa, että tulevaisuudessa GQL-tulee sisältämään myös muita paluuarvojen esitystapoja. Tämän lisäksi GQL:än on suunnitteilla useampia ominaisuuksia, jotka eivät tule olemaan mukana vielä ensimmäisessä julkaistavassa versiossa. Ensimmäinen versio ei esimerkiksi mahdollista vielä useamman graafin kyselyä yhdellä kyselyllä. Tuleviin versioihin on myös suunnitteilla mahdollisuus graafimallien rajoittamiseen isomorfisilla sovitusmalleilla.

6.2 Aikataulu

Vuoden 2021 WG3- ja LDBC-työryhmien jäsenien kirjoittamassa artikkelissa on arvioitu, että SQL/PGQ-standardi julkaistaisiin maaliskuussa 2023 ja GQL-standardi syyskuussa 2023. GQL-standardin omalla sivustolla on arvioitu kesäkuussa 2022, että GQL-standardi tulisi voimaan huhtikuussa 2024. Tätä ennen GQL-standardin seuraavista versioista tullaan oletettavasti julkaisemaan yksi tai kaksi akateemista julkaisua. [JCC Consulting 2023] ISO:n verkkosivuilta voidaan nähdä, että SQL/PGQ-standardin käsittely (ISO/IEC DIS 9075-16) on tällä hetkellä hieman pidemmällä kuin GQL-standardin (ISO/IEC CD 39075.2) [International Organization for Standardization 2023a ja 2023b].

7 Kyselykielten ja GQL-standardin vertailu

Tässä luvussa vertaillaan luvussa 5 esiteltyjä kyselykieliä toisiinsa. Tämän lisäksi kyselykieliä vertaillaan luvussa 6 esiteltyyn GQL-kyselykielistandardiin. Kyselykielten vertailu toteutetaan taulukoinnin avulla.

Tutkielmassa vertailtavat kyselykielten ominaisuudet ovat seuraavat:

- ominaisuus- vai RDF-graafeille suunnattu kieli
- paluuarvot (totuusarvo, solmu, polku, graafi)
- graafimallien arvioimisessa käytettävä semantiikka (homomorfismi vai isomorfismi)
- tulostenkäsittelysemantiikka graafimalleille (bag vai set)
- polkukyselysemantiikka (sattumanvarainen, no-repeated-edge, no-repeated-node, lyhin polku)
- sallitaanko kahdensuuntaiset polkukyselyt?
- voiko yksi kysely kohdistua useaan eri graafiin?
- onko käytössä UNION-operaatio?
- sisältääkö kyselykieli CRUD-toiminnot (create, read, update, delete)?
- syntaksin tyyli (Cypher-tyylinen ”synteettisen sokerin” käyttö, SQL-tyylinen, funktionaalinen)

Yllä esitetyt ominaisuudet on sijoitettu taulukossa 2 pystyakselille. Kyselykielten nimet ovat taulukon vaaka-akselilla. Taulukossa eri semantiikkojen kohdalla esiintyvät tähtimerkit (*) tarkoittavat, että kyselykieli sisältää operaatiota, joiden avulla myös muita semantiikkoja on käytettävissä. Plus-merkki (+) GQL-kyselykielen ominaisuuksissa tarkoittaa, että kyselykielen ensimmäinen versio ei sisällä kyseistä ominaisuutta, mutta se on suunnitteilla myöhempiin versioihin. Tähti- ja plus-merkin yhdistelmä (+*) puolestaan tarkoittaa, että kyselykielen myöhemmät versiot tulevat sisältämään operaatioita, joiden avulla myös muita semantiikkoja on käytettävissä.

GQL-kyselykielen tulevista ominaisuksista kertova artikkeli ei antanut suoraa vastausta siihen, käytetäänkö tulostenkäsittelyssä set- vai bag-semantiikkaa. Artikkelissa ei ollut myöskään mainintaa mahdollisesta DISTINCT-operaatiosta. Tämän vuoksi taulukossa on kysymysmerkki kyseisen semantiikan kohdalla.

	Cypher	PGQL	GSQL	Gremlin	SPARQL	G-CORE	GQL
Graafityyppi	ominaisuusgraafi	ominaisuusgraafi	ominaisuusgraafi	ominaisuusgraafi	RDF-graafi	ominaisuusgraafi	ominaisuusgraafi
Paluuarvot	solmu, polku	totuusarvo, solmu	solmu	solmu, polku	solmu, totuus-arvo, graafi	graafi	solmu, polku
Semantiikka graafimalleille	isomorfismi*	homomorfismi*	homomorfismi*	homomorfismi	homomorfismi	homomorfismi	homomorfismi+*
Tulostenkäsittely-semanttiikka	bag-semanttikka*	bag-semantiikka*	bag-semantiikka*	bag-semantiikka*	bag-semantiikka*	set-semantiikka	?
Polkukysely-semanttiikka	no-repeated-edge*	sattumanvarainen*	lyhin polku	sattuman-varainen*	sattumanvarainen	lyhin polku	sattumanvarainen*
Kahdensuuntaiset polkukyselyt	kyllä	kyllä	kyllä	kyllä	kyllä	kyllä	kyllä
Usean graafin kyselyt	ei	ei	ei	ei	kyllä	kyllä	ei+
UNION-operaatio	kyllä	ei	kyllä	kyllä	kyllä	kyllä	kyllä
CRUD-toiminnot	kyllä	kyllä	kyllä	kyllä	kyllä	kyllä	kyllä
Syntaksin tyyli	SQL-vaikutteita, synteettinen sokeri	SQL-vaikutteita, synteettinen sokeri	SQL-vaikutteita	funktionaalinen	SQL-vaikutteita	SQL-vaikutteita, synteettinen sokeri	SQL-vaikutteita, synteettinen sokeri

Taulukko 2. Kyselykielten ominaisuudet

7.1 Kyselykielten vertailu

Taulukosta voidaan nähdä, että iso osa graafitietokannoille suunnatuista kyselykielistä on suunnattu ominaisuusgraafeille. Tämä johtunee siitä, että SPARQL on RDF-graafien kyselykielistandardi. Koska ominaisuusgraafeille ei ole vielä vastaava standardia, on eri ominaisuusgraafeja käyttävien järjestelmien täytynyt luoda oma kyselykieli.

Kyselykielten paluuarvojen välillä on suurta vaihtelua. Vain Cypher ja Gremlin sallivat polkujen palauttamisen. Jo käytössä olevista kyselykielistä vain SPARQL sallii graafin paluuarvona. Myös kyselykieliehdotelma G-CORE sallii graafin palauttamisen. Kaikki käytössä olevat kyselykielet sallivat kyselyiden palauttavan solmun. PGQL ja SPARQL sallivat lisäksi myös totuusarvojen palauttamisen. Paluuarvoja vertailtaessa SPARQL on monipuolisin, sillä se sallii kolme erityyppistä paluuarvoa. GSQL sen sijaan on paluuarvojen kannalta suppein, sillä se sallii ainoastaan solmujen palauttamisen.

Cypher on ainoa kyselykieli, joka käyttää graafimallien arvioimisessa lähtökohtaisesti isomorfismiin pohjaavaa semantiikkaa. Myös homomorfismiin pohjaavaa semantiikkaa on mahdollista käyttää lisäoperaatioilla. PGQL ja GSQL käyttävät lähtökohtaisesti homomorfismiin pohjaavaa semantiikkaa graafimallien arvioimisessa, mutta sallivat myös isomorfisen semantiikan käytön lisäoperaatioilla. Graafimallien tulostenkäsittelysemantiikka on kaikissa käytössä olevissa kyselykielissä bag-semantiikka. Kaikissa näissä on myös mahdollista noudattaa set-semantiikkaa käyttämällä DISTINCT-avainsanaa tai muuta vastaavaa toimintoa.

Kyselykielten eroavat toisistaan myös siinä, mitä polkukyselysemantiikkaa ne lähtökohtaisesti käyttävät. Valittujen kyselykielten osalta voidaan sanoa, että suosituin polkukyselysemantiikka on sattumanvaraisten polkujen semantiikka, jota käyttävät PGQL, Gremlin ja SPARQL. Lyhimmän polun semantiikka käyttävät GSQL sekä G-CORE. Cypher käyttää ainoana kyselykielenä no-repeated-edge-semantiikkaa. Moni kyselykieli sallii myös muiden polkusemantiikkojen käytön lisäoperaatioilla. Kaikki kyselykielet sallivat kahdensuuntaiset polkukyselyt.

Tällä hetkellä käytössä olevista kyselykielistä vain SPARQL sallii usean graafin kyselyt. Myös G-CORE kyselykieliehdotelma sallii kyselyt, jotka voivat kohdistua useampaan graafiin samanaikaisesti. PGQL:ä lukuun ottamatta kaikki kyselykielet mahdollistavat kahden joukon unionin muodostamisen. Kaikki kyselykielet sisältävät CRUD-toiminnot.

Gremlin eroaa syntaksiltaan selvästi muista kyselykielestä. Gremlin muistuttaa funktionaalisuudellaan jopa enemmän ohjelmointikieltä kuin kyselykieltä [Angles *et al.* 2017]. Cypher, PGQL ja G-CORE ovat saaneet vaikutteita SQL-kyselykielestä, minkä lisäksi niissä kaikissa on käytetty synteettistä sokeria, helpottamaan ja selkeyttämään mallien kirjoittamista ja lukemista. Myös GSQL ja SPARQL syntaksissa on selvästi vaikutteita SQL-kyselykielestä. Nämä kaksi kieltä eivät kuitenkaan sisällä juurikaan synteettistä sokeria.

Osa kyselykielistä on käytettävissä vain tietyissä graafitietokannoissa, kun taas osaa on mahdollista käyttää useilla eri alustoilla. PGQL toimii yhdessä Oraclen graafitietokannan kanssa ja GSQL TigerGraph-alustan kanssa. Cypher-kyselykielen käyttäminen on mahdollista Neo4j-alustalla, mutta Cypherista jatkokehitetty openCypher on käytettävissä useilla ei alustoilla. Myös Gremliniä ja SPARQL:ä voidaan käyttää useammalla eri alustalla. Koska G-CORE on kyselykieliehdotelma, ei sen käyttö ole tällä hetkellä mahdollista millään alustalla.

7.2 Polkukyselyt eri kyselykielillä

Vertaillaan seuraavaksi kyselykieliä kahden erilaisen polkukyselyn avulla. Kaikki esimerkkikyselyt palauttavat solmun, sillä se on ainoa paluuarvo, joka on käytettävissä kaikissa kyselykielissä.

Ensimmäisessä polkukyselyssä haetaan ensin kaikki solmut *h1*, joiden tyyppi on *Henkilö* ja joiden nimi on *Tiina*. Solmulla *h1* on *seuraa* tyyppinen suhde solmuun *h2*, jonka tyyppi on *Henkilö*. Solmulla *h2* on *seuraa* tyyppinen suhde solmuun *h3*, joka on tyypiltään *Henkilö* ja jolla on nimi *Pekka*. Molempien *seuraa* suhteiden suunta on vapaa.

Cypher

```
MATCH (h1:Henkilö {nimi: 'Tiina'})-[:SEURAA]-(h2:Henkilö)-[:SEURAA]-
    (h3:Henkilö {nimi: 'Pekka'})
RETURN h1
```

PGQL

```
SELECT h1
FROM MATCH (h1:Henkilö) -[:seuraa]- (h2:Henkilö) -[:seuraa]- (h3:Henkilö)
WHERE h1.nimi = 'Tiina' AND h3.nimi = 'Pekka'
```

GSQL

```
SELECT h1
FROM Henkilö:h1 -(seuraa)- Henkilö:h2 -(seuraa)- Henkilö:h3
WHERE h1.nimi == "Tiina" AND h3.nimi == "Pekka";
```

Gremlin

```
g.V().hasLabel('Henkilö', 'nimi', 'Tiina').bothE('seuraa').otherV().
hasLabel('Henkilö').bothE('seuraa').otherV().hasLabel('Henkilö', 'nimi', 'Pekka')
```

SPARQL

```
SELECT ?h1
WHERE {
    ?h1 (:seuraa/^:seuraa) ?h2 .
    ?h1 :type :Henkilö . ?h1 :nimi "Tiina" .
    ?h2 :type :Henkilö .
    ?h2 (:seuraa/^:seuraa) ?h3 .
    ?h3 :type :Henkilö . ?h3 :nimi "Pekka"
}
```

G-CORE

```
CONSTRUCT (h1)
MATCH (h1:Henkilö) -[:seuraa]- (h2:Henkilö) -[:seuraa]- (h3:Henkilö)
WHERE h1.nimi='Tiina' AND h3.nimi= 'Pekka'
```

Toisessa polkukyselyssä haetaan ne henkilöt, jotka asuvat samassa kaupungissa henkilön Tiina kanssa. Solmu *h1* on tyyppiä *Henkilö* ja sillä on ominaisuus nimi, jonka arvo on *Tiina*. Solmulla *h1* on suunnattu suhde *asuu* solmuun *k*, joka on tyyppiä *Kaupunki* ja jonka nimi on *Tampere*. Solmu *h2* on myös tyyppiä *Henkilö* ja sillä on suunnattu suhde, jonka tyyppi on *asuu*, solmuun *k*. Kysely palauttaa solmun *h2*.

Cypher

```
MATCH (h1:Henkilö {nimi: 'Tiina'})-[:ASUU]->
    (k:Kaupunki {nimi: 'Tampere'})<-[:ASUU]-(h2:Henkilö)
RETURN h2
```

PGQL

```
SELECT h2
FROM MATCH (h1:Henkilö) -[:asuu]-> (k:Kaupunki) <-[:asuu]- (h2:Henkilö)
WHERE h1.nimi = 'Tiina' AND k.nimi = 'Tampere'
```

GSQL

```
SELECT h2
FROM Henkilö:H1 -(asuu>)- Kaupunki:k -(<asuu)- Henkilö:h2
WHERE h1.nimi == "Tiina" AND k.nimi == "Tampere";
```

Gremlin

```
g.V().hasLabel('Henkilö').OutE('asuu').InV().
hasLabel('Kaupunki', 'nimi', 'Tampere').InE('asuu').OutV().
hasLabel('Henkilö', 'nimi', 'Tiina')
```

SPARQL

```
SELECT ?h2
WHERE {
    ?h1 :asuu ?k . ?h1 :type :Henkilö .
    ?h1 :nimi "Tiina" . ?k :type :Kaupunki .
    ?k :nimi "Tampere" . ?h2 :asuu ?k .
    ?h2 :type :Henkilö
}
```

G-CORE

```
CONSTRUCT (h2)
MATCH (h1:Henkilö) -[:asuu]-> (k:Kaupunki) <-[:asuu] -(h2:Henkilö)
WHERE h1.nimi='Tiina' AND k.nimi='Tampere'
```

7.3 GQL ja kyselykielet

Kuten aiemmin on tullut jo esille, tulee GQL-standardi olemaan kyselykielistandardi ominaisuusgraafeille. Kuten Cypherissä ja Gremlinissä, paluuarvoiksi sallitaan solmut ja polut. GQL:n myöhemmät versiot saattavat sisältää myös esimerkiksi graafin palauttamisen. Kuten suurin osa kyselykielistä, myös GQL noudattaa homomorfismiin pohjaavaa semantiikkaa graafimallien arvioimisessa. Myöhempiin versioihin on myös

suunnitteilla ominaisuus, joka mahdollistaa isomorfismin käytön kyselyissä. Kuten iso osa jo olemassa olevista kyselykielistä, myös GQL käyttää lyhimmän polun semantiikkaa polkukyselyissä. Tämän lisäksi GQL mahdollistaa muiden polkusemantiikkojen käytön lisäoperaatioilla. GQL sallii kahdensuuntaiset polkukyselyt. Ensimmäinen GQL-standardin versio ei tule sisältämään usean graafin kyselyitä, mutta tämä ominaisuus on suunnitteilla myöhempiin versioihin. GQL mahdollistaa UNION-operaation ja se sisältää CRUD-toiminnot. Syntaksi vastaa pitkälti Cypheria ja PGQL, eli GQL:ssä voidaan nähdä vaikutteita SQL:stä ja sen syntaksiin on lisätty synteettistä sokeria.

Kun lasketaan, millä kyselykielellä on eniten samoja ominaisuuksia kuin GQL:llä, saadaan vastaukseksi Gremlin, jolla on kahdeksan yhteistä ominaisuutta. Cypher ja PGQL omaavat seitsemän samaa ominaisuutta GQL:n kanssa. Kauimpana GQL:stä on SPARQL, jolla on vain viisi samaa ominaisuutta. Vaikka GQL:än on otettu vaikutteita Cypherista, on näissä silti merkittäviä eroja kyselyissä käytettävien semantiikkojen osalta. PGQL:n ja GQL:n eroavaisuudet löytyvät eroavaisuuksissa paluuarvojen ja UNION-operaation käytön suhteen. Vaikka Gremlinin syntaksi poikkeaa täysin GQL:n syntaksista, vastaa se tässä tutkielmassa tutkittujen ominaisuuksien perusteella eniten GQL:ää. Mikään tässä tutkielmassa vertailtu kyselykieli ei vastaa ominaisuuksiltaan täysin tulevaa ominaisuusgraafeille suunnattua kyselykielistandardia.

7.4 Polkukyselyt GQL:llä

Luvussa 7.2 vertailtiin kyselykieliä kahden polkukyselyn avulla. Esitetään seuraavaksi samat polkukyselyn GQL-kyselykielellä.

Kysely, jossa haetaan solmu *h1* nimeltään *Tiina*, jolla on suhde *seuraa Henkilö*-solmuun, jolla on suhde *seuraa* solmuun *h2*, esitetään GQL:llä seuraavasti:

```
MATCH (h1:Henkilö WHERE h1.nimi='Tiina')−[:Seuraa]−(:Henkilö)
    −[:Seuraa]−(h2:Henkilö WHERE h2.nimi='Pekka')
```

Koska GQL ei sisällä erillistä SELECT tai RETURN lausetta, palauttaa se kaikki kyselyssä esitellyt muuttujat. Kyselyssä halutaan määrittää solmun *h2* nimeksi *Pekka*, jolloin kyseinen solmu tulee sijoittaa muuttujaan. Näin ollen kysely palauttaa solmut *h1* ja *h2*. Tämä eroaa kaikista jo olemassa olevista kyselykielistä, sillä niissä on mahdollisuus jättää palauttamatta muuttujiin sidotut solmut.

Toisessa polkukyselyssä haetaan ne solmut *h2*, jotka asuvat samassa kaupungissa, jossa *Tiina* asuu. Kysely voidaan esittää seuraavasti:

```
MATCH (h1:Henkilö WHERE h1.nimi='Tiina')−[:Asuu]−>
    (k:Kaupunki WHERE k.nimi='Tampere')<−[:Asuu]−(h2:Henkilö)
```

Kyselyssä annetaan ehtoja sekä solmulle *h1* että solmulle *k*, jolloin nämä tulee sitoa muuttujiin. Näin ollen kysely palauttaa solmut *h1*, *k* ja *h2*.

8 Keskustelu

Luvussa 7 tehdyn vertailun perusteella voidaan todeta, että graafitietokantojen kyselykielet eroavat toisistaan semantiikkojen ja syntaksin osalta. Funktionaalinen Gremlin eroaa syntaksiltaan täysin muista kyselykielistä. Kaikki kielet sallivat kahdensuuntaiset polkukyselyt ja sisältävät täydet CRUD-toiminnot. Kaikkien kyselykielien paluuarvoksi sallitaan solmu, mutta muiden sallittujen paluuarvojen välillä on eroja. Esimerkiksi polkujen palauttaminen on mahdollista vain kahdella kyselykielellä. Suurin osa kyselykielistä noudattaa graafimallien arvioimisessa homomorfismiin pohjaavaa semantiikkaa ja tulosten palauttamisessa bag-semantiikkaa. Kyselykielien käyttämien polkusemantiikkojen välillä on vaihtelua. Moni kyselykieli mahdollistaa kuitenkin usean eri polkusemantiikan käytön.

Ominaisuusgraafien tulevassa GQL-kyselykielistandardissa voidaan nähdä samoja piirteitä, kuin jo olemassa olevissa graafitietokantojen kyselykielissä. Deutsch ja muut kertovat vuoden 2021 artikkelissaan, että GQL on graafitietokantojen kyselykielien pitkän kehityksen kulminaatio. Näiden kyselykielien kehitykseen ovat vaikuttaneet tutkimukset, standardit sekä käytännöt. GQL:n ensimmäinen versio tulee vastaamaan sallittujen paluuarvojen osalta Cypheria ja Gremliniä. Syntaksiltaan se muistuttaa hyvin paljon Cypheria ja PGQL:ää, sillä myös GQL-syntaksista on pyritty tekemään helpommin luettavaa ja kirjoitettavaa synteettisen sokerin avulla. GQL:ssä polkujen suunta tai suuntaamattomuus voidaan esittää seitsemällä eri tavalla. Osaan näistä esitystavoista on liitetty myös ehtolause. Esimerkiksi polku, joka esitetään muodossa −[suhde]−, sallitut suunnat ovat vasen, oikea tai suuntaamaton. Muissa kielissä polkujen suuntaa ei voida esittää vastaavalla tavalla. Tämä GQL-kyselykielen ominaisuus saattaa aiheuttaa sekaannusta, jos ei ole perehtynyt kunnolla GQL:n polkujen esitystapoihin.

GQL tulee sisältämään myös muita uusia ja uniikkeja ominaisuuksia, kuten rajoittimia ja valitsimia. Uutena ominaisuutena tulee myös se, että GQL-kyselykielessä polut ovat ensimmäisen luokan kansalaisia. Tämä ominaisuus on myös G-CORE-kyselykieliehdotelmassa, mutta ei vielä missään käytössä olevassa kyselykielessä. Ensimmäinen julkaistava versio GQL:stä ei tule vielä sisältämään kaikkia ominaisuksia, joita siihen on suunniteltu. Esimerkiksi kyselyn kohdistaminen useampaan graafiin ei ole vielä mahdollista ensimmäisessä versiossa.

GQL-kyselykielistandardin arvioitu julkaisuajankohta on tämänhetkisen tiedon valossa huhtikuussa 2024. Standardin julkaisun jälkeen mahdollisena jatkotutkimuksena voisi vertailla RDF-graafien SPARQL-kyselykielistandardia ja ominaisuusgraafien GQL-kyselykielistandardia.

9 Yhteenveto

Tutkielmassa vertailtiin graafitietokantojen kyselykieliä. Tutkielman tavoitteena oli selvittää, minkälaisia graafitietokantojen kyselykieliä on olemassa ja kuinka ne eroavat toisistaan. Lisäksi haluttiin selvittää, kuinka graafitietokantojen kyselykielet eroavat tulevasta GQL-kyselykielistandardista. Koska GQL-standardia ei ole vielä julkaistu, vertailtiin kyselykieliä niihin GQL:n ominaisuuksiin, jotka ovat jo julkaistu. Tutkielmaan vertailtaviin kyselykieliin valikoituivat Cypher, PGQL, GSQL, Gremlin, SPARQL ja G-CORE.

Vuoden 2017 artikkelissaan Angles ja muut vertailivat kolmea eri graafitietokantojen kyselykieltä: Cypheria, Gremliniä ja SPARQL:ää. Tuossa artikkelissa Angles ja muut totesivat, että nämä kyselykielet vastaavat ydinominaisuuksiltaan pitkälti toisiaan, mutta sisältävät kaikki myös omia uniikkeja piirteitä. Tämän tutkielman osalta voidaan päätyä samaan lopputulemaan. Kielten ominaisuuksissa, semantiikoissa ja syntakseissa on eroja, mutta samat ydinominaisuudet löytyvät kaikista kielistä.

Tuleva ominaisuusgraafien kyselykielistandardi GQL tulee sisältämään paljon vaikutteita jo olemassa olevista kyselykielistä ja syntaksiltaan se vastaa pitkälti Cypheria ja PGQL:ä. GQL tulee sisältämään myös uusia ja uniikkeja ominaisuuksia.

Graafitietokantojen kyselykielet sisältävät useita ominaisuuksia, joita tässä tutkielmassa ei vertailtu. Esimerkiksi Anglesin ja muiden vuoden 2017 artikkelissa sekä Sharman ja muiden vuoden 2021 artikkelissa kyselykieliä vertaillaan hieman laajemmin, mutta niissä molemmissa käsitellään vain kolmea eri kieltä. Näin ollen tässä tutkielmassa ei päästy vertailemaan graafitietokantojen kyselykielien kaikkia ominaisuuksia, mutta sen sijaan pyrittiin antamaan peruskuvaus useammasta eri kyselykielestä sekä esitellä niiden suurimpia eroja ja yhtäläisyyksiä.

Lähteet

Angles, R., Arenas, M., Barceló, P., Boncz, P., Fletcher, G. H. L., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J., Oskar van Rest, & Voigt, H. 2017. G-CORE: A Core for Future Graph Query Languages. arXiv.org.

Angles, R., Arenas, M., Barcelo, P., Boncz, P., Fletcher, G., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J., van Rest, O., & Voigt, H. 2018. G-CORE: A Core for Future Graph Query Languages. Proceedings of the 2018 International Conference on Management of Data, 1421–1432. https://doi.org/10.1145/3183713.3190654

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. 2017. Foundations of Modern Query Languages for Graph Databases. ACM Computing Surveys, 50(5), 1–40. https://doi.org/10.1145/3104031

Angles, R., Bonifati, A., Dumbrava, S., Fletcher, G., Hare, K. W., Hidders, J., Lee, V. E., Li, B., Libkin, L., Martens, W., Murlak, F., Perryman, J., Savković, O., Schmidt, M., Sequeda, J., Staworko, S., & Tomaszuk, D. 2021. PG-Keys: Keys for Property Graphs. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2423–2436. https://doi.org/10.1145/3448016.3457561

Balakrishnan, R., & Ranganathan, K. 2012. *A Textbook of Graph Theory* (2nd ed. 2012.). Springer New York. https://doi.org/10.1007/978-1-4614-4529-6

Barrasa, J., Hodler, A. E. & Webber, J. 2021. *Knowledge Graphs*. O'Reilly Media, Inc.

Bechberger, D., Perryman, J., & Wilmes, T. 2020. *Graph databases in action: examples in Gremlin*. Manning.

Beineke, L. W., & Wilson, R. J. 2013. *Topics in Structural Graph Theory*. Cambridge University Press.

Broecheler, M. & Gosnell, D. 2020. *The Practitioner's Guide to Graph Data*. O'Reilly Media, Inc.

Calvanese, D., De Giacomo, G., Lenzerini, M., & Vardi, M. Y. 2003. Reasoning on regular path queries. SIGMOD Record, 32(4), 83–92. https://doi.org/10.1145/959060.959076

Deutsch, A., Nadime Francis, Green, A., Hare, K., Li, B., Libkin, L., Lindaaker, T., Marsault, V., Martens, W., Michels, J., Murlak, F., Plantikow, S., Selmer, P., Voigt, H., Oskar van Rest, Vrgoč, D., Wu, M., & Zemke, F. 2021. Graph Pattern Matching in GQL and SQL/PGQ. arXiv.org.

Deutsch, A., Xu, Y., Wu, M., & Lee, V. 2019. TigerGraph: A Native MPP Graph Database. arXiv.org.

Efron, S. E., & Ravid, R. 2019. *Writing the literature review: a practical guide*. The Guilford Press.

Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., & Wu, Y. 2010. Graph pattern matching: from intractable to polynomial time. Proceedings of the VLDB Endowment, 3(1-2), 264–275. https://doi.org/10.14778/1920841.1920878

Gould, R. 1988. *Graph Theory*. Benjamin/Cummings.

Hogan, A., Blomqvist, E., Cochez, M., D'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. 2021. Knowledge graphs. ACM Computing Surveys, 54(4), 1–37. https://doi.org/10.1145/34477

International Organization for Standardization. ISO/IEC CD 39075.2 Information Technology — Database Languages — GQL. Viitattu 17.3.2023a. https://www.iso.org/standard/76120.html

International Organization for Standardization. ISO/IEC DIS 9075-16
Information technology — Database languages SQL — Part 16: Property Graph Queries (SQL/PGQ). Viitattu 17.3.2023b. https://www.iso.org/standard/79473.html

JCC Consulting, Inc., acting on behalf of an unincorporated association of ISO Graph Query Language Proponents, and licensed under the Apache License, Version 2.0. What

is in a Graph Query Language Standard? Viitattu 17.3.2023. https://www.gqlstandards.org/what-is-a-gql-standard

Kollias, G., Sathe, M., Schenk, O., & Grama, A. 2014. Fast parallel algorithms for graph similarity and matching. Journal of Parallel and Distributed Computing, 74(5), 2400–2410. https://doi.org/10.1016/j.jpdc.2013.12.010

Marcus, D. A. 2008. *Graph theory a problem oriented approach*. Mathematical Association of America.

Needham, M., & Hodler, A. E. 2019. *Graph algorithms: practical examples in Apache Spark and Neo4j* (First edition.). O'Reilly.

Neo4j, Inc. 2021. openCypher. Viitattu 5.4.2023. https://opencypher.org/

Neo4j. Docs. Getting Started 5. Introduction to Cypher. Viitattu 8.2.2023a. https://neo4j.com/docs/getting-started/current/cypher-intro/

Neo4j. Docs. Neo4j Graph data science 2.2. Link prediction pipelines. Viitattu 18.1.2023b. https://neo4j.com/docs/graph-data-science/current/machine-learning/linkprediction-pipelines/link-prediction/

PGQL | Property Graph Query Language. PGQL 1.5 Specification. Viitattu 17.2.2023. https://pgql-lang.org/spec/1.5/

Pombrio, J., Krishnamurthi, S., & Wand, M. 2017. Inferring scope through syntactic sugar. Proceedings of ACM on Programming Languages, 1(ICFP), 1–28. https://doi.org/10.1145/3110288

Robinson, I., Webber, J., & Eifrem, E. 2015. *Graph databases: new opportunities for connected data* (Second edition.). O'Reilly.

Rodriguez, M. A. 2015. The gremlin graph traversal machine and language (Invited Talk). DBPL 2015 - Proceedings of the 15th Symposium on Database Programming Languages, 1–10. https://doi.org/10.1145/2815072.2815073

Sakr, S., Bonifati, A., Voigt, H., Iosup, A., Ammar, K., Angles Rojas, R., Aref, W., Arenas, M., Besta, M., Boncz, P., Daudjee, K., Della Valle, E., Dumbrava, S., Hartig, O.,

Haslhofer, B., Hegeman, T., Hidders, J., Hose, K., Iamnitchi, A., … Yoneki, E. 2021. The future is big graphs! A community view on graph processing systems. Communications of the ACM, 64(9), 62–71. https://doi.org/10.1145/3434642

Sharma, C., Sinha, R., & Johnson, K. 2021. Practical and comprehensive formalisms for modelling contemporary graph query languages. Information Systems (Oxford), 102, 101816–. https://doi.org/10.1016/j.is.2021.101816

Thakkar, H., Punjani, D., Keswani, Y., Lehmann, J., & Auer, S. 2018. A Stitch in Time Saves Nine -- SPARQL querying of Property Graphs using Gremlin Traversals. arXiv.org.

The Apache Software Foundation. Apache Jena. SPARQL Tutorial. Viitattu 7.3.2023a. https://jena.apache.org/tutorials/sparql.html

The Apache Software Foundation. TinkerPop Documentation. Viitattu 3.3.2023b. https://tinkerpop.apache.org/docs/current/reference/#_tinkerpop_documentation

TigerGraph. Docs. GSQL Language Reference 3.8. Viitattu 27.2.2023. https://docs.tigergraph.com/gsql-ref/current/intro/

W3C. 2014. RDF 1.1 Concepts and Abstract Syntax: W3C Recommendation 25 February 2014. Viitattu 9.12.2022. https://www.w3.org/TR/rdf11-concepts/

Whittemore, R., & Knafl, K. 2005. The Integrative Review: Updated Methodology. Journal of Advanced Nursing 52 (5): 546–53.

World Wide Web Consortium. W3C Recommendation. SPARQL Query Language for RDF. Viitattu 8.3.2023. https://www.w3.org/TR/rdf-sparql-query/

Xiao, Y., & Watson, M. 2019. Guidance on Conducting a Systematic Literature Review. Journal of Planning Education and Research, 39(1), 93–112. https://doi.org/10.1177/0739456X17723971