Tampereen yliopisto

Saija Seppä

VISUAALINEN KYSELYMALLI GRAAFITIETOKANTAAN

Kieliopillinen määrittely ja toteutus

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Toukokuu 2023

TIIVISTELMÄ

Saija Seppä: Visuaalinen kyselymalli graafitietokantaan: Kieliopillinen määrittely ja toteutus
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2023

Tämän tutkielman teoreettisissa osissa kuvataan graafitietokantojen periaatteita, graafitietokanta Neo4j:n toimintaa sekä sen kyselykieltä Cypheriä yleisesti. Lisäksi tutkitaan hakutuloksen visualisointia graafitietokannasta ja käydään läpi Neo4j-tietokannalle suunniteltuja visualisoinnin teknologioita.

Graafitietokannassa tieto on visuaalista ja verkkomaista, mutta luonnollisena esteenä laajemmalle käytölle on kyselykielien haastavuus loppukäyttäjälle. Tutkielman konstruktiivinen osuus kuvaa oman kieliopin ja verkkosovelluksen kehittämistä niin, että graafitietokannan hyödyt ja tulosten visuaalisuus ovat myös loppukäyttäjän saatavilla. Luotu kielioppi on s-attribuoitu ja siinä laadittiin säännöt, joiden perusteella voidaan muuntaa tekstimuotoinen hakulauseke sovelluksessa Cypher-kyselykielelle.

Sovellus kehitettiin JavaScriptin React-kirjastolla ja tietokantana oli Neo4j-graafitietokanta. Sovellukseen toteutettiin kaksi erilaista hakutapaa, joista ensimmäinen sisältää ennalta määriteltyjä parametrisoituja hakuja. Se mahdollistaa sovelluksen käytön ilman tietokannan rakenteen tarkempaa tuntemusta. Toisena hakutapana kehitettiin malli, joka hyödyntää luotua kielioppia. Tässä hakutavassa käyttäjä voi määritellä kyselypolun askel kerrallaan ilman Cypher-kyselykieltä.

Tutkielman taustalla oli Euroopan komission osarahoittama tutkimushanke EurOMo sekä hankkeen ensimmäistä vaihetta varten kokoama tietokanta. Tietokanta sisältää tietoja 15 Euroopan maan uutismedioista ja niiden yritys- sekä henkilöomistajista. Sen tavoitteena on lisätä medioiden omistussuhteiden läpinäkyvyyttä ja tietoa taustalla vaikuttavista omistajista.

Avainsanat: graafitietokanta, graph database, Neo4j, Cypher, attribuuttikielioppi, attribute grammar

SISÄLLYS

1 Johdanto 2

2 Graafimalli 4

2.1 Graafiteoria 4

2.2 Ominaisuusgraafi 5

3 Graafitietokannat 7

3.1 Graafitietokantoja ja niiden piirteitä 7

3.2 Graafitietokantojen hyödyt 9

3.3 Neo4j 9

4 Graafitietokantojen kyselykielet 11

4.1 Säännölliset polkukyselyt 11

4.2 Cypher 12

5 Visualisointi graafitietokannoissa 16

5.1 Graafitietokantojen visualisoinnin yleisiä piirteitä 16

5.2 Visualisoinnin työkalut Neo4j-graafitietokannan kanssa 18

6 Sovelluksen rakenne ja tietokanta 20

6.1 Sovelluksen taustahanke 20

6.2 Tietokannan sisältämä data 22

6.3 Sovelluksen vaatimusmäärittely ja teknologiat 23

7 Perushaku 26

7.1 Taustaa ja perusperiaatteet 26

7.2 Käyttötapaukset 26

8 Edistynyt haku 33

8.1 Edistyneen haun toteutus sovelluksessa 33

8.2 Kontekstivapaa kielioppi ja attribuuttikielioppi 35

8.3 Kyselyiden evaluointi 41

9 Pohdinta 51

10 Johtopäätökset 53

Viiteluettelo 54

1 Johdanto

Ensimmäiset graafiteoriat ja mallinnukset graafeista esitettiin jo 1800-luvulla [Foulds, 1995, s. 3–4]. Niiden pohjalta kehitettiin graafitietokantoja 1980- ja 1990-luvuilla, jolloin ne olivat suosiossa muun muassa hypertekstin antamien mahdollisuuksien ansiosta. Varhaiset graafitietokannat jäivät kuitenkin pian taka-alalle ja helppokäyttöisemmät relaatiotietokannat nousivat suosioon erityisesti liiketoiminnan sovelluksissa [Angles and Gutierrez, 2008]. Relaatiotietokannat ovat edelleenkin suosituimpia tietokantoja [DB-Engines, 2023].

Ohjelmistojen vaatimusten laajentumisen ja tietojen sekä sovellusten monipuolistumisen myötä myös muiden kuin relaatiotietokantojen käyttö on kasvanut [Angles, 2012]. Näitä muita tietokantoja kutsutaan NoSQL-tietokannoiksi (*not only SQL*) ja ne käsittävät useita erilaisia dokumentti-, avain-arvo- ja graafitietokantoja.

Erilaisista NoSQL-tietokannoista graafitietokannat ovat kasvattaneet suosiotaan sovelluskehityksessä [Angles, 2012]. Tähän on vaikuttanut sovellusalueiden monipuolistuminen ja siten relaatiotietokantojen asettamat rajoitteet tiedon hallinnassa. Voimakkaasti toisiinsa linkittyneen tiedon käsittelyssä relaatiotietokannat vaativat pitkiä ja monimutkaisia liitosoperaatioita. Graafitietokannat taas ovat parhaimmillaan silloin, kun tieto on verkkomaista ja toisiinsa linkittynyttä, mikä on omiaan lisäämään graafitietokantojen suosiota. Sosiaalisen median sovellusten sisältämä tieto on jopa intuitiivista esittää graafimuodossa, jolloin erityisesti tietojen välisillä suhteilla on merkittävä rooli. Muita tärkeitä sovellusalueita graafitietokannoille ovat kemian ja biologian sekä karttapalveluiden ja liikenneverkkojen alat.

Tässä tutkielmassa kuvataan yleisesti graafitietokantoja ja niiden taustalla olevaa graafiteoriaa. Tarkemmin kuvattavaksi graafitietokannaksi valittiin Neo4j, joka on tällä hetkellä eniten käytetty graafitietokanta [DB-Engines, 2023]. Neo4j:n lisäksi kuvataan siihen kehitettyä kyselykieltä Cypheriä. Cypher on deklaratiivinen kyselykieli, joka käytännössä ilmaisee, millaista lopputulosta tietokantakyselyllä haetaan. Graafitietokantoihin kehitettyjä kyselykieliä on olemassa useita, mutta puutteena niiden kehityksessä voidaan nähdä yhteisten yleisten standardien puuttuminen [Angles, 2012].

Graafitietokannasta tieto on mahdollista saada visuaalisessa helposti hahmotettavassa muodossa. Graafitietokannoilla on paljon annettavaa tietojen analysoinnille, tulosten algoritmiselle operoinnille sekä haluttavien ominaisuuksien korostamiselle tuloksissa. Haasteena graafitietokantojen käyttäjäkunnan laajentamiselle myös muille kuin ohjelmoijille voidaan kuitenkin pitää vaatimusta kyselykielien osaamisesta.

Tässä tutkielmassa tutkimuskysymyksenä olikin juuri hakumallin luominen ja graafitietokannan hakutuloksen visuaalinen esitys loppukäyttäjälle. Tavoitteena oli luoda kielioppi, jonka avulla voidaan muuntaa tekstimuotoisia hakulauseita Neo4j graafitietokan-

nan Cypher-kyselykielelle. Kielioppia sovellettiin käytäntöön kehittämällä verkkosovellus, jonka avulla myös kyselykieltä osaamaton loppukäyttäjä voi tehdä virallisen kyselykielen omaisia hakuja graafitietokantaan.

Sovellus kehitettiin JavaScriptin React-kirjastolla ja tulosten visualisointiin käytettiin Neovis.js-kirjastoa. Toteutustavaksi valittiin kahden eri kyselytavan kehittäminen: perushaun sekä edistyneen haun. Perushaun avulla käyttäjän on mahdollista tehdä graafitietokantaan ennalta määriteltyjä hakuja niin, että käyttäjä antaa haettavien entiteettien ja suhteiden nimiä sekä näiden ominaisuuksia. Käyttäjä voi myös valita kuinka laajasti hän on kiinnostunut entiteettiin liittyvistä suhteista. Sovellus muodostaa annettujen tietojen perusteella Cypher-kyselylauseen graafitietokantaan ja näyttää hakutuloksen käyttäjälle visualisoituna.

Edistynyttä hakua varten kehitettiin oma kielioppinsa, jota käyttäen sovellus muodostaa Cypher-lauseet käyttäjän antamien tietojen mukaisesti. Kieliopin perustana käytettiin attribuuttikielioppia, joka on Donald Knuthin [1968] julkaisema kontekstivapaan kieliopin laajennos. Se on ollut julkaisustaan lähtien vankka perusta ohjelmointikielien ja niiden kääntäjien teknisten toteutusten pohjana. Tutkielmassa luotu kielioppi luokitellaan kontekstivapaata kielioppia laajentavaa attribuuttikielioppia toteuttavaksi.

Tutkielman taustalla on Euroopan komission osarahoittama tutkimushanke EurOMo (*Euromedia Ownership Monitor*) ja sen kokoama tietokanta eurooppalaisten uutismedioiden omistussuhteista [Euromo, 2023]. Tietokanta sisältää tietoja eurooppalaisista mediayrityksistä, niiden omistajista ja yritysten keskinäisistä omistussuhteista. EurOMo-hanke pyrkii lisäämään uutismedioiden omistussuhteiden läpinäkyvyyttä kaikilla tasoilla ja tukemaan siten tiedotusvälineiden moniarvoisuutta.

Tutkielman sisältö etenee lukujen 2 ja 3 graafiteorian ja graafitietokantojen kautta luvun 4 teoriaan graafitietokantojen kyselykielistä. Kaikissa luvuissa käsitellään teoriaa yleisellä tasolla, mutta myös kuvataan tarkemmin valittuja aiheita. Näitä ovat graafitietokanta Neo4j sekä sen kyselykieli Cypher. Luvussa 5 käydään läpi erilaisia visualisointitapoja graafitietokannoille sekä tarkemmin muutamaa visualisointityökalua. Luku 6 kattaa kehitetyn sovelluksen esittelyn taustasta tarkempaan toteutukseen. Sovelluksen kahta erilaista hakutapaa on käsitelty luvuissa 7 ja 8. Perushakua ennalta annettujen hakumahdollisuuksien kanssa käydään läpi luvussa 7 ja edistyneen haun hakulausekkeen muodostumista ja kielioppia sen taustalla käydään läpi luvussa 8. Tutkielman yhteenvetona toimivat luvut 9 ja 10, joissa pohditaan tutkielmasta pois rajautuneita alueita sekä tulevaisuuden kehityskohteita ja kootaan yhteen tutkielmaa.

2 Graafimalli

Graafi on paljon tutkittu matemaattinen tapa esittää tietoa myös visuaalisesti. Graafeja on käytetty pitkään ja kuuluisia mallinnuksia graafiteoriasta on esitetty 1700- ja 1800-luvuilla, esimerkkinä Königsbergin siltapulma vuodelta 1736 [Foulds, 1995, s. 3–4]. Siinä tarkoituksena oli ylittää kaupungin halkaisevan joen sillat niin, että jokainen silta kuljetaan vain kerran ja aloitus sekä lopetus olisivat samassa pisteessä. Sveitsiläinen matemaatikko L. Euler todisti pulman olevan mahdoton ratkaista ja samalla hän tuli luoneeksi graafiteorialle pohjan.

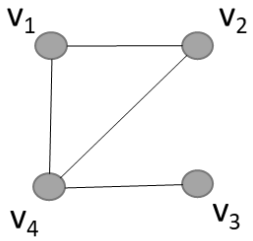
Itse termi graafi on peräisin 1800-luvun lopun kemistien kehittämästä tavasta esittää yhdisteiden kemiallisia rakenteita [Foulds, 1995, s. 7–8]. Atomien liittymistä toisiinsa kutsuttiin termillä *graafinen merkintätapa* (*graphic notation*), mistä nimi graafi on johdettu.

Tässä luvussa esitellään graafiteorian yleisiä piirteitä luvussa 2.1. Graafitietokantoihin liittyviä ominaisuusgraafeja käydään läpi luvussa 2.2.

2.1 Graafiteoria

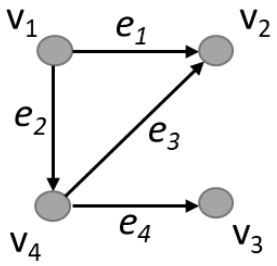
Graafi voidaan määritellä järjestetyksi pariksi G = (V, E), missä V on äärellinen joukko *solmuja* (*vertices*) ja E on joukko kaaria [Foulds, 1995, s. 9–14]. Silloin, kun solmuja p ja q yhdistää kaari e, näitä solmuja kutsutaan *vierussolmuiksi* (*adjacent*) [Koivisto ja Niemistö, 2018]. Kaari e taas kulkee solmujen p ja q kautta (*incident with*).

Erilaisia graafeja ovat esimerkiksi yksinkertainen graafi sekä multigraafi ja nämä voivat olla suunnattuja tai suuntaamattomia [Koivisto ja Niemistö, 2018]. Yksinkertaisessa graafissa ei voi olla kaarta itseensä ja kaaret voidaan esittää järjestämättöminä solmupareina. Kuvassa 1 on esitetty yksinkertaisen graafin kuvamallinnus. Solmut on identifioitu numeroin ja kaaret esitetään lähtö- ja päätesolmujen kaksipaikkaisina joukkoina.



Kuva 1. Yksinkertainen graafi G = ({v1, v2, v3, v4}, {{v1,v2}, {v1, v3}, {v2, v3}, {v3, v4}}).

Tämän tutkielman kannalta oleellisin graafi on kuitenkin suunnattu multigraafi, mikä tarkoittaa, että kahden solmun välillä voi olla usea suunnattu kaari. Kuvassa 2 on esimerkki suunnatusta graafista. Solmut on nimetty numeroin samoin kuin kuvassa 1, mutta nyt suunnatussa graafissa kaaret on myös nimetty numeroin.



Kuva 2. Suunnattu graafi (V, E) = ({v1, v2, v3, v4}, {e1, e2, e3, e4}).

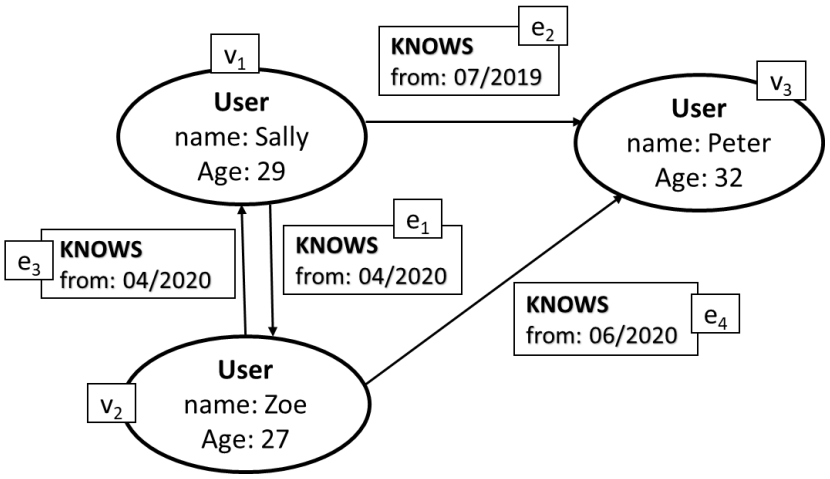
2.2 Ominaisuusgraafi

Kaarinimikkeinen graafi (*edge-labelled graph*) on graafi, jossa jokaiseen kaareen on liitetty nimiketieto. Ominaisuusgraafi (*property graph*) on kaarinimikkeisen graafin mukaelma ja laajennos, jossa solmuilla ja kaarilla voi olla määriteltynä ominaisuuksia [Angles *et al.*, 2017]. Ominaisuusgraafi voidaan määritellä formaalisti monikkona G = (V, E, ρ, λ, σ), jossa

1. V merkitsee äärellistä solmujen joukkoa.
2. E merkitsee äärellistä kaarien joukkoa.
3. ρ: E → (V × V) on funktio, joka osoittaa jokaisen kahden solmun (∈ V) välisen kaaren kuuluvan joukkoon E.
4. λ: (V ∪ E) → L on funktio, joka osoittaa solmulle tai kaarelle nimikkeen joukosta L.
5. σ: (V ∪ E) × P → PV on funktio, joka osoittaa solmun tai kaaren ominaisuuden (∈ P) arvon (∈ PV).

Angles ja muut [2017] määrittelevät ominaisuusgraafin niin, että solmut ja kaaret sisältävät yhden nimikkeen ja ominaisuuksilla on korkeintaan yksi arvo. Kuitenkin joissain sovelluksissa solmujen ja kaarien ominaisuuksilla voi olla useita arvoja. Tällöin voidaan puhua *moniarvoisista ominaisuusgraafeista* (*multi-valued property graphs*). Useimmiten graafitietokannoissa käytetty ominaisuusgraafin malli sisältää mahdollisuuden asettaa solmuille useita eri ominaisuuksia, mutta kaarille vain yhden, useimmiten yksilöivän ominaisuuden.

Kuvassa 3 on kuvattuna esimerkki sosiaalisen median sovelluksen käyttäjistä ja heidän välisistänsä suhteista. Tässä ominaisuusgraafissa solmuina ovat käyttäjät ja kaarina (suhteina) tieto tuntemisesta. Kaarien suunnasta näkee, kuka tuntee kenetkin ja tästä saakin hyvän kuvan graafin tarjoamasta visuaalisesta tiedosta. Solmuilla on ominaisuuksina käyttäjän nimi sekä ikä ja kaarilla ominaisuuksina on tieto, mistä asti käyttäjä on toisen tuntenut.



Kuva 3. Ominaisuusgraafi sosiaalisen median esimerkkisuhteista.

Kuvan 3 graafi voidaan esittää formaalisti aiemmin kuvattua määritelmää käyttäen seuraavanlaisesti:

- $V = \{v_1, v_2, v_3\}$
- $E = \{e_1, e_2, e_3, e_4\}$
- $P = \{\text{name, age, from}\}$
- $L = \{\text{User, KNOWS}\}$
- $\lambda = \{(v_1, \text{User}), (v_2, \text{User}), (v_3, \text{User}), (e_1, \text{KNOWS}), (e_2, \text{KNOWS}), (e_3, \text{KNOWS}), (e_4, \text{KNOWS})\}$
- $\rho = \{[e_1, (v_1, v_2)], [e_2, (v_1, v_3)], [e_3, (v_2, v_1)], [e_4, (v_2, v_3)]\}$
- $\sigma = \{[(v_1, \text{name}), \text{Sally}], [(v_1, \text{age}), 29], [(v_2, \text{name}), \text{Zoe}], [(v_2, \text{age}), 27], [(v_3, \text{name}), \text{Peter}], [(v_3, \text{age}), 32], [(e_1, \text{from}), 04/2020], [(e_2, \text{from}), 07/2019], [(e_3, \text{from}), 04/2020], [(e_4, \text{from}), 06/2020]\}$

3 Graafitietokannat

Graafitietokannat ovat saavuttaneet yhä suurempaa suosiota viime vuosina. Graafimuotoisten tietokantojen sovellusalueet ovat laajentuneet muun muassa sosiaalisten medioiden, biologian ja oliosuuntautuneen datan alalla [Libkin *et al*., 2016]. Kun tieto on verkottunutta ja tietojen väliset suhteet merkittävässä roolissa, kuten sosiaalisen median sovelluksissa, graafitietokannat ovat osoittaneet hyötynsä.

Luvussa 3.1 käydään läpi muutamia graafitietokantoja sekä niiden piirteitä ja eroja. Graafitietokantojen tehokkuutta ja hyötyä käsitelleitä tutkimuksia käydään läpi luvussa 3.2. Viimeisenä luvussa 3.3 kuvataan tutkielmassa kehitetyn sovelluksen tietokannaksi valittua Neo4j:n toimintaa tarkemmin.

3.1 Graafitietokantoja ja niiden piirteitä

Graafitietokannoille ei ole muodostettu yhteisiä määritelmiä, vaikkakin yhteisiä piirteitä tietokannoilla onkin. De Virgilio ja muut [2013] listaavat yhteisiä piirteitä ja yksi näistä yhteisistä piirteistä on indeksivapaus, mikä tarkoittaa, että jokaisella solmulla on tieto vain omista vierussolmuistaan. Tämä kuitenkin on vanhentunutta tietoa, sillä esimerkiksi Neo4j mahdollistaa laajan indeksoinnin ja suositteleekin sitä hakujen tehokkuuden parantamiseksi [Neo4j, 2023].

Toisena yhteisenä piirteenä on, että graafitietokantoja voidaan pääsääntöisesti pitää ominaisuusgraafeina [de Virgilio *et al.*, 2013]. Luvussa 2.2 esiteltiin ominaisuusgraafien peruspiirteet. Graafitietokannat perustuvat luvussa 2 esitettyyn graafiteoriaan, joten graafitietokannoilla on vahva teoreettinen pohja.

DB-Engines-sivuston listauksen perusteella suosituimpia graafitietokantoja ovat tällä hetkellä Neo4j, Microsoft Azure Cosmos DB, Virtuoso sekä Arango DB [DB-Engines, 2023]. Listauksen mukaan näiden neljän tietokannan suosio ja järjestys listalla ovat pysyneet samana jo ainakin vuoden.

DB-Engines-sivuston listauksen pohjana on kuuden mittarin tapa laskea tietokantojen suosiota [DB-Engines, 2023]. Näitä mittareita ovat mainintojen määrät Googlen ja Bingin hakukoneissa, yleinen kiinnostus eli esiintymistiheys Google Trendsissä, teknisten kysymysten määrä Stack Overflow- tai DBA Stack Exchange -sivustoilla. Lisäksi mittareina on työtarjousten määrä Indeed- ja Simply Hired-sivustoilla, tietokannan maininnat LinkedIn-profiileissa sekä maininnan tietokannasta sisältävien twiittien määrä Twitterissä.

Mainituista suosituimmista tietokannoista vain Neo4j on täysin puhtaasti graafitietokanta, sillä muut ovat *monimalli-tietokantoja (multi-model database*) [DB-Engines, 2023]. Monimalli-tietokanta tarkoittaa tietokantaa, joka voi sisältää erilaisia tietokantaratkaisuja. Microsoft Azure Cosmos DB tarjoaa mahdollisuuden dokumentti-, graafi, avain-arvo- sekä sarakeperhetietokannoille. Myös Virtuoso ja Arango DB mahdollistavat

tietokannan rakenteen olevan avain-arvo-, dokumentti- tai graafitietokanta. Virtuoso tukee lisäksi myös relaatiotietokantoja.

Taulukossa 1 on esitetty mainittujen tietokantojen oleellisimpia ominaisuuksia perustuen DB-Engines-sivuston antamiin tietoihin ja listauksiin. Huomattavaa on, että kolme neljästä on avoimen lähdekoodin tietokantoja ja käyttö on näin mahdollista useammalle käyttäjälle. Virtuoso on taas ainoa, joka vaatii pakollisena skeeman määrittelyn tietokantaan.

	Tietorakenne	**Julkaisu**	**Lisenssi**	**Skeema**	**Transaktio-konsepti**
Neo4j	Graafi	2007	Avoin lähdekoodi	Valinnainen	ACID
Microsoft Azure Cosmos DB	Monimalli	2014	Kaupallinen	Ei	Osin ACID
Virtuoso	Monimalli	1998	Avoin lähdekoodi	Kyllä	ACID
Arango DB	Monimalli	2012	Avoin lähdekoodi	Ei	ACID

Taulukko 1. Graafitietokantojen vertailu.

Taulukon 1 graafitietokannoista kaikki toteuttavat vähintään osittain ACID-periaatetta [Baton and van Bruggen, 2017]. ACID-periaate on yleinen tietokantojen toimintaa kuvaava listaus ominaisuuksista, joita hyvällä tietokannalla halutaan olevan. ACID-lyhenne tulee sanoista *Atomicity, Consistency, Isolation* ja *Durability*.

- **Atomicity:** Atomisuus. Muutokset tietokannassa noudattavat kaikki tai ei mitään -periaatetta. Jos jokin osa transaktioista ei onnistu, kaikki palautuu tietokannassa lähtötilanteeseen.
- **Consistency:** Eheys. Neo4j kohdalla eheys merkitsee sitä, että kaikilla suhteilla on alku- ja loppusolmu. Tiukan skeeman omaavissa tietokannoissa eheys merkitsee myös skeeman noudattamista, mutta koska Neo4j skeema on valinnainen, sitä ei vaadita.
- **Isolation:** Eristyneisyys. Rinnakkain samaan aikaan samaan tietokantaan toteutetut transaktiot eivät saa vaikuttaa toisiinsa. Päällekkäin toteutetut transaktiot eivät tiedä toistensa väliaikaisia tiloja.
- **Durability:** Pysyvyys. Transaktion suorittamisen jälkeen ne eivät voi enää kadota. Lokikirjanpito transaktioista takaa tiedonhallinnan laadun.

3.2 Graafitietokantojen hyödyt

Laajasti käytetyillä relaatiotietokannoilla on vahva asema tietojen tallettamisessa ja monissa tapauksissa niiden käyttö onkin perusteltua [Robinson *et al.*, 2015]. Graafitietokannoilla on kuitenkin hyötynsä.

Graafitietokannat ovat painottuneita tiedon välisiin suhteisiin, suhteet (kaaret) solmujen välillä luovat perustan tietokannalle [Robinson *et al.*, 2015]. Tämä painotus tuo eron verrattuna esimerkiksi relaatiotietokantojen erilaisiin liitoksiin ja siten tiedon rakenteeseen. Kun suhteet ovat merkittävässä roolissa tiedontallennuksessa, graafitietokannan etuna on suorituskyky. Tietokannan koon kasvaessa kyselyiden suoritusajat pysyvät graafitietokannoissa hyvin maltillisina.

Joustavuus nousee graafitietokantojen hyödyksi tilanteissa, joissa talletettavan tiedon muoto ja laatu voivat muuttua [Robinson *et al*., 2015]. Graafitietokannat eivät vaadi useinkaan erikseen määriteltyjä skeemoja tai tiettyjä rakenteita, joten tiedon mallinnus on vapaampaa kuin esimerkiksi relaatiotietokannoissa. Tämä tuo joustavuutta ja mahdollisuuden tallettaa uudenlaisia tietoja tietokantaan ja laajentaa sitä.

Monissa tutkimuksissa on osoitettu, että graafitietokannat ovat tehokkaampia kuin relaatiotietokannat. Khan ja muut [2019] vertailivat Oracle 11 g ja Neo4j 3.03 Community -versiota datan ollessa potilastietoja, lääkityksiä ja hoitohenkilökuntaa. Tässä vertailussa erilaisten count(*)-kyselyiden lopputulemana Neo4j oli tehokkaampi.

Holzschuher ja Peinl [2013] testasivat Neo4j versiota 1.8 sekä MySQL-tietokantaa erilaisten backend-ratkaisujen kanssa. Tuloksena oli, että tietokannan koon kasvaessa Neo4j:n tehokkuus pysyi lähes vakaana, kun taas MySQL:n tehokkuus putosi.

Näitä tutkimuksia yhdistää suhteellisen yksinkertaisten hakukyselyiden muodostaminen ja lopputulema graafitietokannan paremmasta tehokkuudesta verrattuna perinteisempiin relaatiotietokantoihin.

Kotiranta ja muut [2022] kuitenkin vertailivat tietokantojen tehokkuutta erityisesti tilanteissa, joissa hakukyselyt olivat monimutkaisia. Mukana vertailussa oli mukana relaatiotietokannoista MariaDB sekä kaksi MySQL-versiota. Graafitietokannoista oli valittu Neo4j. Tässä tutkimuksessa Neo4j oli tehokkaampi yksinkertaisten kyselyiden kohdalla ollen vähintään kolme kertaa nopeampi kuin relaatiotietokannat. Kuitenkin kaikista monimutkaisimpien kyselyiden kohdalla relaatiotietokannat voittivat Neo4j:n johtuen relaatiotietokantojen indeksoinnin antamasta hyödystä tehokkuuteen.

3.3 Neo4j

Neo4j on vuonna 2007 ensimmäisen kerran julkaistu avoimen lähdekoodin graafitietokanta [Neo4j, 2023]. Se on kehitetty Java-ohjelmointikielellä ja Neo4j onkin tällä hetkellä käytetyin graafitietokantaohjelmisto [DB-Engines, 2023]. DB-Engines listaa tietokantoja

perustuen niiden suosioon, joka on mitattu kuuden mittarin avulla, jotka esiteltiin aiemmin luvussa 3.1. Toistaiseksi uusin versio 5.6 Neo4j-tietokannasta on julkaistu maaliskuussa 2023, mikä toi edelleen parannuksia muun muassa tehokkuuteen ja skaalautuvuuteen [Neo4j, 2023].

Neo4j toteuttaa tietokantojen yleistä *ACID*-periaatetta, joka esiteltiin aiemmin luvussa 3.1 [Baton and van Bruggen, 2017]. ACID-periaatteen toteuttaminen voidaan katsoa erittäin suureksi hyödyksi, sillä näin Neo4j -tietokantojen luotettavuutta voidaan pitää perinteisten relaatiotietokantojen tasolla.

Tieto on jäsennetty Neo4j-tietokantaan graafiteorian mukaisesti solmuina ja kaarina. Kaarien sijaan käytetään kuitenkin termiä suhteet. Solmuilla ja suhteilla voi olla ominaisuuksia, joten tieto varastoidaankin siten ominaisuusgraafeina [Neo4j, 2023]. Solmuilla on aina *nimiketieto* (*label*) sekä lisäksi *ominaisuuksia* (*properties*) voi olla useita. Suhteet ovat yhteyksiä solmujen välillä ja ne vastaavat suunnattuja kaaria. Suhteella on tyyppi eli nimi sekä alku- ja loppusolmu. Lisäksi yhdellä solmulla voi olla useita eri suhteita muihin solmuihin ilman vaikutusta suorituskykyyn.

Tietojen eheyden ylläpitämiseen Neo4j tarjoaa mahdollisuuden skeemojen käyttöön tietojen hallinnassa [Neo4j, 2023]. Lisäksi erilaiset rajoitteet lisäävät tiedon eheyttä. Solmujen ominaisuuksille voidaan asettaa rajoitus, joka määrää solmujen tietyn ominaisuuden arvon uniikiksi. Toisin sanoen eri solmuissa ei tällöin voi olla ominaisuuksilla samoja arvoja, mikä on erityisen hyödyllistä esimerkiksi nimien tai identifioivien numeroiden kanssa. Neo4j tarjoaa monipuolisempia rajoituksia maksullisessa yritysversiossaan. Esimerkiksi solmuille voidaan nimetä ominaisuuksia, jotka niille on aina pakollista määritellä.

Neo4j tarjoaa yhteensopivuutta useiden eri ohjelmointikielien kanssa ja se on mahdollista yhdistää monin tavoin kehitettäviin sovelluksiin [Neo4j, 2023]. Neo4j tukee ajureita Java-, JavaScript-, Python-, Go- sekä .Net- ohjelmointikielille. Yhteys sovelluksen ja tietokannan välillä toteutuu erillisen binäärisen Bolt-protokollan kautta. Tietokanta voidaan luoda Neo4j:n tarjoamaan työpöytäsovellukseen, johon otetaan kehitettävästä sovelluksesta yhteys. Toinen vaihtoehto on luoda tietokanta Neo4j:n AuraDB-nimiseen pilvipalveluun.

4 Graafitietokantojen kyselykielet

Yhteisen standardin puuttumista graafien kyselykieliltä voidaan pitää isoimpana heikkona puolena graafitietokantoja ajatellen [Angles, 2012]. Kuitenkin kyselykieliä on kehitetty useita erilaisia.

Graafien kyselykielet voidaan jakaa karkeasti kahteen erilaiseen tyyppiin [de Virgilio *et al.*, 2013]. Näissä kyselyn eteneminen perustuu kyselyn ja tietokannan väliseen graafien vertailuun. Toisen tyypin kyselykielet pohjautuvat tietokannan polkujen osoittamiseen. Tässä kappaleessa käydään läpi ensimmäisen tyypin kyselykieliä, sillä *graafien kaavioiden täsmäys* (*graph pattern matching*) on useamman paljon käytettyjen graafien kyselykielen taustalla [Angles *et al.*, 2017].

Ominaisuusgraafeihin kehitettyjä kyselykieliä ovat muun muassa Cypher ja Gremlin. Näissä on paljon eroja toteutuksissa ja ilmaisuvoimissa, mutta niillä on myös yhteisiä tekijöitä [Angles *et al.*, 2017]. Näiden kyselykielien ydinkonsepti on sama: graafitietokantojen hakujen kaksi perusoperaatiota graafien kaavioiden täsmäys ja graafien navigointi sisältyvät niihin.

Luvussa 4.1 käydään ensin läpi säännöllisten polkukyselyiden teoriaa, mikä on formaali pohja kyselykielille. Tutkielman kannalta tärkein kyselykieli on luvussa 4.2 esiteltävä Cypher, sillä se on Neo4j-graafitietokantaan kehitetty kyselykieli. Cypheristä ja sen syntaksista esitellään tärkeimpiä perusasioita.

4.1 Säännölliset polkukyselyt

Graafitietokantojen navigoinnin ja hakukyselyjen mekanismien perustana on s*äännölliset polkukyselyt* (*regular path queries, RPQ*) sekä sen laajennokset [Calvanese *et al*., 2003]. Säännölliset polkukyselyt antavat joustavia tapoja navigoida graafissa, esimerkiksi polkujen hakeminen on mahdollista RPQ:n avulla, vaikka polkujen pituuksia ei ennalta tiedettäisi.

Säännöllinen polkukysely määritellään formaalisti triplettinä P = (x, L, y) [Barceló *et al*., 2014]. Tässä P kuvaa polkua, jossa x on polun alkusolmu ja y on polun päätesolmu. L on säännöllinen lauseke ja $L \subseteq \Sigma^*$, eli säännöllisen lausekkeen merkit kuuluvat aakkostoon Σ. Toisin selitettynä säännölliset polkukyselyt hakevat graafitietokannasta kaksi solmua, joita yhdistää polku. Polun nimiketiedot kuuluvat annettuun säännölliseen lausekkeeseen.

Mainittu säännöllinen lauseke polkukyselyssä muodostetaan käyttäen yleisiä säännöllisten lausekkeiden sääntöjä ja merkintöjä [Barceló *et al*., 2014]. Näitä ovat esimerkiksi merkit ?, * ja +. Kysymysmerkki säännöllisessä lausekkeessa määrää sitä edeltävän merkin esiintymään nolla tai yksi kertaa. Siten lauseke *colou?r* täsmää molempiin ”color” ja ”colour”. Tähtimerkki määrää sitä edeltävän merkin esiintymään nolla tai useamman kerran. Tästä esimerkkinä lauseke *ab*c* täsmää muun muassa lauseisiin ”ac”, ”abc”,

”abbc”, joissa merkin b esiintymismäärät vaihtelevat. Kolmas paljon käytetty merkki + määrää sitä edeltävän merkin esiintymään yhden tai useamman kerran. Täten lauseke *ab+c* täsmää esimerkiksi lauseisiin ”abc” ja ”abbc”, mutta ei lauseeseen ”ac”.

Tiedon hakeminen graafitietokannasta on yksinkertaisimmillaan luodun kaavan vertaamista graafiin ja niiden vastaavuuden tarkastelua [Barceló *et al*., 2014]. Monimutkaisimpia tilanteita varten on kehitetty laajennoksia yksinkertaisimmille säännöllisille polkukyselyille. *Konjunktiiviset säännölliset polkukyselyt* (*conjunctive regular path queries*) ovat yksi kehitetty laajennos, joka vastaa monimutkaisimpien kyselyiden tarpeeseen sallimalla polkuihin myös välisolmut.

Barceló ja muut [2014] kävivät läpi useita tutkimuksia, joissa konjunktiivisten polkukyselyiden käytännöllisyyttä ja hyödyllisyyttä oli tutkittu. Kun konjunktiivisen polkukyselyn ydin on luotu säännöllinen polkukysely, se todettiin käytännössä paremmaksi vaihtoehdoksi kuin pelkkä yksinkertainen polkukysely. Konjunktiivisissa polkukyselyissä voidaan esimerkiksi yhdistää muuttujien avulla useampia polkukyselyitä, mikä tekee siitä kattavamman tavan hakea tietoa.

Toinen yleinen laajennos on *kaksisuuntaiset polkukyselyt* (*two-way regular path queries, 2RPQ*) [Calvanese *et al.*, 2003]. Niissä on mahdollisuus kulkea solmuja yhdistäviä kaaria molempiin suuntiin. Tämä merkitään symbolilla –. Polkukyselyn käyttämä aakkosto merkitään täten formaalisti $\Sigma \pm = \Sigma \cup \{r^- \mid r \in \Sigma\}$, missä r^- vastaa relaation r inversiota eli käänteisrelaatioita. Käytetty aakkosto on siis yhdiste tavallisesta aakkostosta sekä niiden inversioista. Tämä kuvaa polkujen kulkemista molempiin suuntiin.

4.2 Cypher

Cypher on Neo4j-graafitietokantaan kehitetty kyselykieli [Baton and van Bruggen, 2017]. Se on deklaratiivinen kyselykieli, mikä tarkoittaa, että kyselyssä määritellään millaista lopputulosta haetaan. Cypher toteuttaa myös graafin kaavioiden täsmäystä. Kaaviossa 1 esitetään, miten sen polkusyntaksi toteuttaa aina tiettyä kaavaa, jossa kuvataan solmujen ja kaarien yhteyksiä toisiinsa [Angles *et al.*, 2017].

```
(NODE1) - [:RELATIONSHIP] -> (NODE2)
```

Kaavio 1. Cypher-kyselyn polkusyntaksin perusrakenne.

Kyselyiden perussyntaksin selkeys ja helppous tuovat mahdollisuuden hahmottaa myös visuaalisesti kyselyn tarkoitusta ja tavoitetta [Baton and van Bruggen, 2017]. Kyselyiden ilmaisukyky onkin Cypherin isoimpia vahvuuksia ja se tekee perusasioiden oppimisesta uudellekin käyttäjälle selkeämpää.

Syntaksin tarkempaa läpikäymistä varten kaaviossa 2 on esitettynä eräs Cypher-kysely mallitietokantaan, joka sisältää tietoa elokuvista ja niiden näyttelijöistä. Kyselyllä

halutaan löytää henkilöt (Person), jotka ovat näytelleet samassa elokuvassa nimeltään ”Unforgiven”. Kyselylausekkeen perusrakenne koostuu kahdesta lohkosta, ensimmäinen MATCH-lohko sisältää itse kyselyn ja RETURN-lohko sisältää tiedon palautettavista muuttujista. MATCH-lohkon sisällä esitellään haettavat solmut ja suhteet. RETURN-lohko määrittelee kyselystä palautettavat muuttujat. Esimerkkikaaviossa on kiinnostuttu henkilöistä, joihin kysely täsmää.

```
MATCH (x1:Person) –[:ACTS]-> (:Movie{title:'Unforgiven'}) <-[:ACTS]– (x2:Person)
RETURN x1, x2
```

Kaavio 2. Cypher-kyselyn esimerkki.

Syntaksissa solmu esitetään siten, että ensin solmulle annetaan muuttuja ja kaksoispisteen jälkeen esitetään vielä solmun nimiketieto (esimerkiksi x1:Person) [Neo4j, 2023]. On kuitenkin myös mahdollista muodostaa kysely niin, että solmun muuttuja tai nimiketieto jätetään pois (esimerkiksi :Person tai x). Jos solmulle halutaan antaa jokin tietty ehto, kuten esimerkkikaaviossa 2 elokuvan nimi on määritelty olemaan ”Unforgiven”, ehto annetaan solmun nimiketiedon jälkeen aaltosulkeissa. Ensimmäisenä annetaan solmun ominaisuuden nimi ja sille annetaan haluttu arvo. Suhde esitetään syntaksissa vastaavalla tavalla niin, että muuttuja ja suhteen nimiketieto erotetaan kaksoispisteellä. Useimmiten suhde ei kuitenkaan tarvitse muuttujaa, vaan suhde palautetaan tuloksessa joka tapauksessa. Luvussa 5 esiteltävän visualisointityökalu Neovis.js:n kanssa suhde kuitenkin vaatii aina muuttujan myös suhteelle. Viivat ja nuolet syntaksissa kuvaavat suhteen suuntaa: etsitään henkilöä, joka on näytellyt kyseisessä elokuvassa.

Cypher toteuttaa *ei-toistuvan-kaaren semantiikkaa* (*no-repeated-edge semantics*), mikä kaavion 2 esimerkkikyselyssä tarkoittaa sitä, että tuloksista jätetään pois osumat, joissa solmujen x1 ja x2 arvoiksi tulisi sama henkilö [Angles *et al.*, 2017]. Näin yhtäkään kaarta ei käydä läpi kahdesti eikä täten sama solmu esiinny molemmissa muuttujissa samaan aikaan.

Cypher tukee osin luvussa 4.1 kuvattuja säännöllisiä polkulausekkeita [Angles *et al.*, 2017]. Säännöllisten lausekkeiden merkeissä Cypher sallii tähden lisättäväksi suhteen ominaisuuden yhteyteen, vaikkakin se on rajoitettu vain yhteen nimiketietoon. Kaaviossa 3 on esimerkkilauseke, joka hakee henkilöiden välisiä polkuja. Koska tähtimerkki tarkoitti lausekkeen esiintymistä nolla, yksi tai monta kertaa, esimerkkilauseke hakee annettujen henkilöiden välille kaikki minkä tahansa pituiset polut, joissa suhteen nimiketieto on ”knows”.

```
MATCH (x1:Person) –[:KNOWS*]-> (x2:Person)
RETURN x1, x2
```

Kaavio 3. Cypher-kyselyn malli käyttäen suhteessa tähtimerkkiä.

Tähtimerkkiä voidaan käyttää Cypher-kyselyssä myös ilman suhteen määrittelevää nimiketietoa [Angles *et al.*, 2017]. Kaavio 4 esittää tästä esimerkin, joka hakee kaikki mahdolliset polut annettujen henkilöiden välille. Silloin nimiketietoa ei ole määritelty kyselyyn tarkemmin, vaan suhdetta kuvaa vain tähtimerkki. Kaavion 4 mukaisissa kyselyissä myös polun pituus saa olla mikä tahansa.

```
MATCH (x1:Person) –[*]-> (x2:Person)
RETURN x1, x2
```

Kaavio 4. Cypher-kyselyn malli minkä tahansa polun löytämiseksi.

Kaikkien mahdollisten polkujen lisäksi Cypher mahdollistaa myös yksittäisen lyhyimmän polun löytämisen [Angles *et al.*, 2017]. Kaaviossa 5 esitetään esimerkkikysely lyhyimmän polun löytämiseksi kahden henkilön välille. Kysely antaa vastaukseksi lyhyimmän polun, joka Julie-nimisen henkilön ja hänen kavereiden kavereiden välillä on.

```
MATCH (julie:Person{firstname:'Julie'}),
p = shortestPath((julie) –[:KNOWS*]-> (x:Person))
RETURN p
```

Kaavio 5. Cypher-kysely lyhyimmän polun löytämiseksi.

Cypher sallii myös solmujen välisen polun pituuden tarkan määrittelyn hakulausekkeeseen [Angles *et al.*, 2017]. Tämä on selkeä tapa rajoittaa kyselyiden tuloksia ja tarkentaa kyselyn ytimekkyyttä. Kaaviossa 6 esitetään hakukysely, jossa henkilöiden välisiä suhteita haetaan poluista, joiden pituus on vähintään kahden ja enintään seitsemän kaaren mittainen.

```
MATCH (x1:Person) –[:KNOWS*2..7]-> (x2:Person)
RETURN x1, x2
```

Kaavio 6. Cypher-kysely, jossa polun pituus on määritelty.

Aiemmin esitelty tapa esittää ominaisuuden arvolle ehto voi olla käytössä vain silloin, kun kyse on yhtäsuuruudesta [Neo4j, 2023]. Toisin sanoen silloin, kun ehdoksi asetetaan muu kuin yhtäsuuruus, tulee käyttää Cypherin WHERE-lohkoa. Kaaviossa 7 kysely hakee henkilöitä, joiden ikä on suurempi kuin 40. WHERE-lohkossa määritellään muuttujan

ominaisuuden ikä olevan suurempi kuin 40. Tämän kyselyesimerkin RETURN-lohkossa on myös esitetty tapa muokata palautettavien muuttujien tietojen laajuutta. Kysely palauttaa henkilön kaikkien tietojen sijaan henkilön nimen ja iän.

```
MATCH (x:Person)
WHERE x.age > 40
RETURN x.name, x.age
```

Kaavio 7. Cypher-kyselyn esimerkki.

Tässä esitellyt esimerkit Cypherin syntaksista käsittävät hyvin pienen osan kaikista mahdollisuuksista. Nämä yksinkertaiset esimerkit ovat kuitenkin esimerkkejä, joilla voi hahmottaa syntaksin rakenteen keskeisimpiä asioita. On siis hyvä huomioida, että Cypher on esiteltyä paljon laajempi ja ilmaisuvoimaisempi kieli. Esimerkeissä käsiteltiin vain kyselyrakenteita, jotka ovat tutkielman kannalta oleellisia.

5 Visualisointi graafitietokannoissa

Koska graafitietokannoissa tiedon rakenne on lähtökohtaisesti helppo hahmottaa visuaalisesti, tiedon esittämiseen loppukäyttäjille on monia vaihtoehtoja ja tapoja. Luvussa 5.1 käydään läpi visualisoinnin perusperiaatteita ja lisäksi luvussa 5.2 kuvataan tarkemmin muutamaa konkreettista työkalua, jotka on tarkoitettu Neo4j-graafitietokannan kanssa käytettäväksi. Tutkielmassa keskitytään syvimmin Neo4j-graafitietokantaan ja siksi valitut työkalut ovat siihen kehitettyjä ja yhdessä toimimaan suunniteltuja konkreettisia visualisointityökaluja.

5.1 Graafitietokantojen visualisoinnin yleisiä piirteitä

Yleisesti voidaan ajatella, että tiedon visualisointi loppukäyttäjälle on tapa auttaa käyttäjiä ymmärtämään ja analysoimaan tietoa sekä kehittämään ajatuksia tiedoista ja niiden suhteista [Huang *et al*., 2018]. Oikealla tavalla visualisoitu tieto voi auttaa käyttäjää esimerkiksi suorittamaan yksinkertaisia prosenttilaskuja, jotka muuten vaatisivat erityistä osaamista tiedon tulkintaan. Tiedon hyvä visualisointi auttaa myös paljastamaan tiettyä kaavaa toistavaa tietoa tai muuten piilossa olevaa tiedon muotoa. Lisäksi visualisesti esitetty tieto tarjoaa tavallaan avuksi ulkoisen muistin ihmisen rajallisen muistikapasiteetin tueksi.

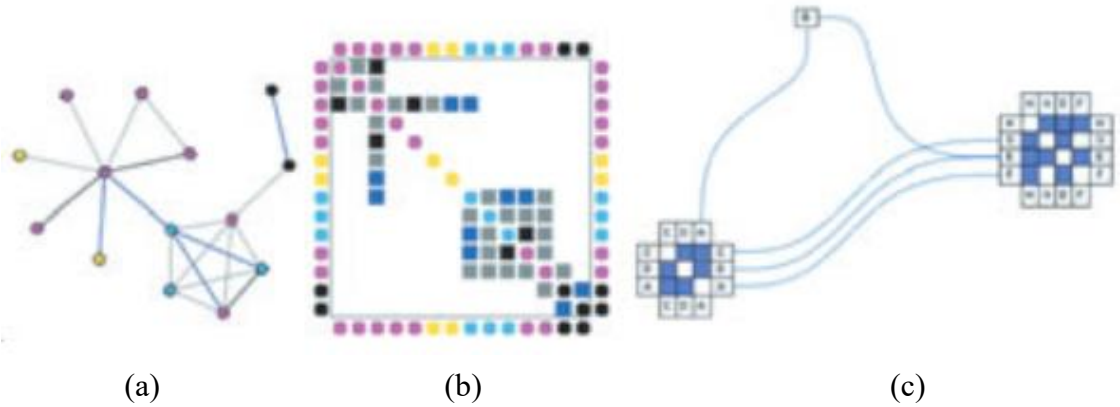
Tiedon visualisointiin on tutkimuksissa kehitetty useita työkaluja ja tapoja, mutta vain osa niistä on päässyt laajempaan käyttöön käytännössä [Huang *et al*., 2018]. Ulkonäkö ja nopeus ovat tärkeitä ominaisuuksia visualisoinnissa, mutta jos käyttäjä ei ymmärrä tietoa oikein, on lopputulos huono. Huang ja muut [2018] ehdottavatkin ohjeeksi visualisoinnin suunnitteluun katsomaan prosessia käyttäjän näkökulmasta sen lisäksi, että huolehditaan visualisoinnin tehokkuudesta ja tulkittavuudesta.

Visualisoinnissa voidaan ajatella olevan kolme näkökulmaa, jotka yhdessä tuovat graafien visuaalisen analysoinnin mahdolliseksi ja antavat parhaan kokonaiskuvan [von Landesberger *et al*., 2011]. Visuaalinen esitys, käyttäjän vuorovaikutus ja algoritmiset analyysit muodostavat perustan tehokkaalle graafin visualisoinnille. Ne myös liittyvät toisiinsa ja osittain toimivat lomittain. Esimerkiksi ennen tiedon visualisointia näkyväksi voidaan tietoa prosessoida ennalta ja suorittaa algoritmisia laskutoimituksia, jolloin tieto visualisoituna näyttää erilaista näkökulmaa kuin pelkkä raakatieto.

Tiedon käsittelystä ennen sen visualisointia käytetään termiä algoritminen esikäsittely ja se voidaan toteuttaa eri tavoilla riippuen tiedon laadusta sekä halutusta korostustavasta visuaalisuuteen [von Landesberger *et al*., 2011]. Yksinkertaisimmillaan algoritmisen esikäsittelyn voidaan ajatella tuovan yksinkertaistusta tietoon ja sen visualisointiin. Tästä seuraa näytetyn graafin pienempi koko, joka auttaa tiedon hahmottamisessa. Samalla kuitenkin graafin rakenne säilyy. Tiedon algoritmista esikäsittelyä voidaan käyttää

korostamaan kiinnostavimpia osia graafista tai muokkaamaan kaarien ja solmujen sijoittelua visualisoinnissa.

Suunnattujen graafien visualisoinnissa voidaan käyttää erilaisia tekniikoita, joilla jokaisella on hyötynsä ja haasteensa visuaalisuuteen [von Landesberger *et al*., 2011]. Nämä tekniikat voidaan jakaa kolmeen ryhmään: solmu-linkki-kaaviot, matriisikaavio ja näistä kahdesta aiemmasta muodostettu yhdistelmä hybridikaavio. Kuvassa 4 voi hahmottaa hyvin eri tekniikoiden erot esittää tieto visuaalisesti.



(a) (b) (c)

Kuva 4. Solmu-linkki- (a), matriisi (b) sekä hybridikaaviot (c) [von Landesberger *et al*., 2011].

Solmu-linkki-kaavio on hyvin intuitiivinen tapa esittää tieto graafista käyttäjälle [von Landesberger *et al*., 2011]. Tämä solmu-linkki-kaavio sisältää graafia vastaavasti solmuja ja solmujen välisiä suhteita kuvaavat taas viivat niiden välillä. Solmu-linkki-kaavio toimii tiedon esitystapana erityisesti silloin, kun graafi on kooltaan pieni tai pienehkö [von Landesberger *et al*., 2011; Ma and Muelder, 2013]. Liian isoissa ja monimutkaisissa graafeissa solmu-linkki-kaavion esitys saattaa helposti muuttua sekavaksi. Haasteeksi voi muodostua solmujen sijoittuminen kuvassa päällekkäin, jolloin luettavuus ja selkeys kärsivät. Vaatimuksina solmu-linkki-kaaviolle voidaankin pitää seuraavia ominaisuuksia: solmut eivät asetu päällekkäin, kaarien risteäminen toistensa kanssa on minimoitu, kaarien pituus on yhtenäinen sekä graafin rakenteet on helppo hahmottaa [von Landesberger *et al*., 2011].

Matriisikaavio esittää tiedon hyvin eri tavalla kuin solmu-linkki-kaavio ja se sopiikin erityisesti isoille graafeille sekä graafeille, joissa tieto asettuisi solmu-linkki-kaavioon hyvin tiheästi [von Landesberger *et al*., 2011]. Matriisikaavion etuna on sen selkeys sekä se, että siitä puuttuvat kokonaan solmujen ja kaarien päällekkäisyydet. Näin kaavio pysyy luettavana, vaikka koko kasvaisi suureksikin. Huonona puolena matriisikaavioissa tulee esille vaikeus seurata polkuja ja hahmottaa nopeasti tiedon rakennetta. Graafeista visualisoitujen kuvien loppukäyttäjät eivät myöskään ole välttämättä tutustuneet matriisikaavioiden periaatteisiin, joten niiden tulkinta voi olla heille vaikeaa.

Solmu-linkki- sekä matriisikaavioiden yhdistäminen visuaalisessa esityksessä on perusteltua silloin, kun graafin tiedossa esiintyy tiheämpiä esiintymiä, mutta myös selkeitä tiedon ryhmittymiä [von Landesberger *et al*., 2011]. Graafin tiheämmät osuudet voidaan esittää matriisikaavioina ja lisäksi kaarilla yhdistää näitä matriiseja toisiinsa kuvan 4c esimerkin mukaisesti. Tämä on keino käyttää hyväksi niin matriisi- kuin solmu-linkkikaavion hyvät puolet visuaalisessa esityksessä.

Vuorovaikutus visualisoidun graafin kanssa on monesti eri tavoin mahdollista [von Landesberger *et al*., 2011]. Eri visualisoinnin työkaluissa mahdollisuudet ovat erilaisia, mutta yleisimpiä mahdollisuuksia ovat zoomaus, solmujen valinta klikkaamalla, solmujen yhteyksien korostaminen. Myös solmujen etsintä niiden tiettyjen ominaisuuksien arvojen perusteella on eräs mahdollisuus. Zoomaus mahdollistaa kuvan tutkimisen tarkemmin pienemmältä alueelta ja voi selventää graafin rakenteen hahmottamista. Zoomaus voi olla myös automatisoitu niin, että klikkaamalla jotain graafin osaa kuva tarkentuu lähemmäs valittua kohtaa.

Visualisointi on tämänhetkisillä teknologioilla jo suurimmilta osin tehokasta ja nopeaa, mutta tulevaisuuden vaatimukset asettavat tarvetta edelleen parantaa näitä ominaisuuksia [von Landesberger *et al*., 2011]. Edelleen graafien koot kasvavat ja erityisesti niiden tietojen esittämisen muokkaaminen algoritmisesti on silloin tärkeässä roolissa. Tulevaisuuden haasteena voidaan myös nähdä tiedon tyyppien lisääntyminen ja toisaalta visualisoinnin työkalujen tulisi sopeutua laajasti erityyppisiin tietoihin ja niiden esittämiseen.

5.2 Visualisoinnin työkalut Neo4j-graafitietokannan kanssa

Työkaluissa on erilaisia ominaisuuksia ja painotuksia ja ne voidaankin jakaa neljään eri kategoriaan: kehitystyökaluihin, tutkimistyökaluihin, analyysityökaluihin sekä raportointityökaluihin [Visualization, 2023]. Jotkin työkaluista voidaan luokitella kuuluvaksi useampaankin kategoriaan monipuolisten ominaisuuksiensa vuoksi. Osa Neo4j:n kanssa toimivista työkaluista on ilmaisia, mutta osa on maksullisia yrityksille suunnattuja työkaluja ja niitä ei tässä tutkielmassa käsitellä tarkemmin.

Kehitystyökaluihin luokiteltavia ilmaisia työkaluja ovat Neovis.js sekä Popoto.js [Visualization, 2023]. Näitä molempia voi käyttää sovellusten kehittämisessä tiedon visualisointiin ja ne ovat molemmat avoimen lähdekoodin JavaScript-pohjaisia sovelluksia. Neovis.js ja Popoto.js ovat samantyylisesti toimivia JavaScript-kirjastoja ja niillä on yhteisiä piirteitä. Esimerkiksi molempien visuaalista tyyliä pystyy muokkaamaan halutunlaiseksi.

Niin Neovis.js- kuin Popoto.js-työkaluissa on helposti mahdollista yhdistää Neo4j-tietokannan Cypher-kyselykielellä tehdyt kyselylauseet visuaalisen kuvan muodostami-

seen [Visualization, 2023]. Neovis.js-työkalun parhaimpina ominaisuuksina voidaan pitää mahdollisuutta muokata solmujen kokoja ja suhteiden paksuutta perustuen erikseen määriteltäviin ominaisuuksien arvoihin [Neovis, 2023]. Lisäksi solmuja voi luokitella kuuluvaksi eri ryhmiin esimerkiksi niiden nimiketietojen perusteella. Neovis.js toimii parhaiten silloin, kun hakutulos on pienehkö ja solmuja on siinä korkeintaan parikymmentä. Näin solmujen ja suhteiden tietoja voi nähdä tarkemmin ja kokonaisuuden pystyy hahmottamaan. Liian suurella tuloksella kuva on liian tiheä mikä hankaloittaa hakutuloksen lukemista.

Kaksi muuta ilmaista visualisointityökalua ovat Neo4j:n omat tuotteet Neo4j Browser sekä Neo4j Bloom [Visualization, 2023]. Nämä toimivat suoraan joko Neo4j työpöytäsovelluksesta tai verkkosivuilla AuraDB:n kautta. Neo4j Browser on enemmän kehitystyökalu, mutta sitä ei voi yhdistää suoraan sovelluskehitykseen. Sen avulla voi testata ja muodostaa Cypher-kyselylauseita tietokantaan ja ylipäätään tutkia sekä ylläpitää tietokannan sisältämää tietoa. Neo4j Bloom voidaan luokitella tutkimus- ja analyysityökalujen kategorioihin. Sillä voidaan tehdä hakuja tietokantaan ja analysoida tuloksia sekä muokata tulosten esittämisen tapoja. Bloom-työkalu tunnistaa myös luonnollisella kielellä tehtyjä hakuja, joten sitä on mahdollista käyttää myös ilman Cypher-kyselykielen käyttöä. Lisäksi Bloom antaa käyttäjälle ehdotuksia haettavista solmuista ja suhteista. Käyttäjältä vaaditaan kuitenkin tietoa solmujen ja suhteiden ominaisuuksien nimistä, jotta niitä voidaan hyödyntää.

Visualisointityökalujen monipuolinen tarjonta tuo runsaasti mahdollisuuksia sovelluskehitykseen ja graafitietokantojen tutkimiseen sekä analysointiin. Ongelmaksi edellä esitetyissä Neovis.js- ja Popoto.js-visualisointityökaluissa nousee dokumentaation ja ohjeiden puute. Koska Neo4j Browser sekä Bloom ovat Neo4j:n ylläpitämiä työkaluja, niihin löytyy ohjeita Neo4j:n verkkosivuilta ja siten niiden ominaisuudet on helpompi hahmottaa ja ottaa käyttöön.

6 Sovelluksen rakenne ja tietokanta

Osana tutkielmaa toteutin web-sovelluksen, joka käyttää graafitietokanta Neo4j:tä sekä visualisoi sieltä tietoa käyttäjälle. Tässä luvussa esitellään toteutettu sovellus tarkemmin sekä käydään läpi tietokannan sisältämää tietoa. Sovelluksen lähtökohtana olleesta hankkeesta esitetään taustoja luvussa 6.1. Tietokantaa ja sen sisältämää dataa käydään läpi luvussa 6.2 ja vaatimusmäärittelyä sekä teknologioita sovelluksen taustalla luvussa 6.3.

6.1 Sovelluksen taustahanke

Sovelluksen lähtökohtana oli Euroopan komission osarahoittama pilottihanke *EurOMo* (*Euromedia Ownership Monitor*), joka on osa Euroopan demokratiaa tukevaa toimintasuunnitelmaa [Euromo, 2023]. Hankkeen ensimmäisessä vaiheessa on mukana 15 Euroopan maata ja näistä maista mukana on yliopistoja ja tutkimusyksiköitä toteuttamassa sitä. Hankkeen koordinaattorina toimii itävaltalainen Salzburgin yliopisto.

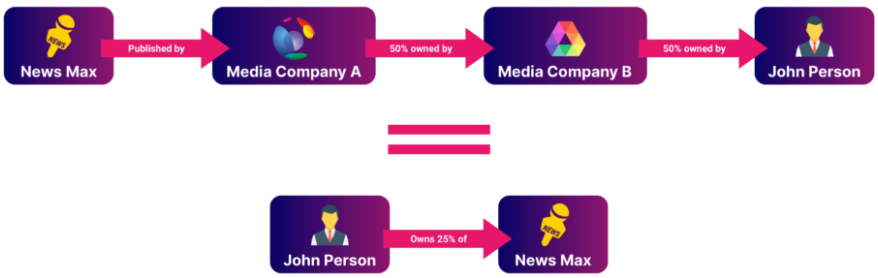
Uutismediat ovat demokraattisessa Euroopassa tärkeässä roolissa oikean tiedon tarjoamisessa kansalaisille ajankohtaisista asioista [Euromo, 2023]. Myös väärinkäytösten paljastamisissa ja erityisesti vallan väärinkäytössä median rooli on suuri. Väärinkäytösten estämisen ja oikean tiedon esittämisen sekä ylipäänsä medioiden luotettavuuden ylläpitämisen yhtenä edellytyksenä on omistussuhteiden läpinäkyvyys ja julkisuus. EurOMo-hanke vastaa tähän tarpeeseen julkaisemalla kootusti omistustietoja painetuista lehdistä, verkkomedioista sekä radio- ja televisiolähetyksiä tekevistä yrityksistä.

EurOMo-hankkeen kantavana ajatuksena on, että medioiden omistuksen ja hallinnan läpinäkyvyyttä tulisi edistää erityisesti tiedotusvälineissä, jotka tuottavat yleiseen mielipiteeseen vaikuttavia julkaisuja [Euromo2, 2023]. Tärkeimpänä laajasti keskustelua herättävinä medioina voidaan pitää uutismedioita, mutta valvonnan tulee yltää myös sisällön jakeluun osallistuviin tahoihin ja siihen liittyviin prosesseihin. Painetussa mediassa sekä radio- ja televisiolähetyksissä nämä medioiden jakeluun ja sen prosesseihin liittyvät tahot ovat rajoitetummin toimivia, mutta digitaalisissa verkkomedioissa niiden toiminta on laajempaa ja hankalammin hahmotettavaa. Verkkomediat muun muassa suodattavat ja priorisoivat sisältöjä eri alustoilla monin eri tavoin, mikä aiheuttaa käyttäjäkohtaisia eroja siihen, mitä uutisia he näkevät.

Hankkeen osallistujamaissa maatiimit ovat keränneet tietoa kotimaidensa medioiden omistussuhteista vuosina 2021–2022 [Euromo2, 2023]. Nyt pilottivaiheessa uutismedioista on valittu rajattu otos perustuen niiden arvioituun merkitykseen vaikuttajana yleisiin mielipiteisiin. Tämän takia kerätty tieto ei ole kaikenkattavaa, mutta sisältää silti laajasti niin kansallisia medioita kuin myös paikallisiakin toimijoita. Lisäksi jokaisen maan otokset sisältävät perinteisiä painettuja medioita, verkkomedioita sekä radio- ja televisiomedioita. Kaikki tieto on kerätty julkisesti saatavilla olevista lähteistä, joten tietokanta on kokonaisuutena yhä julkista tietoa. Tiedonkeruussa on hyödynnetty yritysten ja medioiden

verkkosivuja, yritysrekistereitä ja -raportteja, kansallisten viranomaisten raportteja sekä julkisesti saatavilla olevia tutkimuksia. Tieto voidaan määritellä julkiseksi, kun se on kenen tahansa henkilön saatavilla ilmaiseksi tai maksullisessa palvelussa.

Omistussuhteet ovat hankkeen tietojenkeruun ytimessä ja hyvä esimerkki tiedosta, mikä voidaan kerättyjen omistussuhteiden perusteella havainnollistaa, on epäsuorat omistuspolut [Euromo2, 2023]. Kuvassa 5 on mallinnettu omistussuhteiden ketjuja, joissa uutismedian ja henkilön välillä on todellinen polku. Kuvassa yhtiö B omistaa uutismediaa julkaisevasta yhtiö A:sta 50 % ja henkilö omistaa yhtiöstä B 50 %. Näin ollen henkilön epäsuora omistussuhde suoraan uutismediaan voidaan katsoa olevan 25 %. Yritysten omistajasuhteiden läpinäkyvyys on tärkeää, sillä pitkänkin omistuspolun takaa voi löytyä henkilöitä, jotka vaikuttavat uutismediaan ja sen julkaisuihin.



Kuva 5. Epäsuoran omistuspolun esittäminen henkilöstä mediaan [Euromo2].

Kerätyistä tiedoista EurOMo-hankkeen verkkosivuilla on julkaistu relaatiotietokanta ja hakumahdollisuus tähän tietokantaan [Euromo, 2023]. Tämä hakutoiminto relaatiotietokantaan antaa tulokset taulukkomuodossa. Sittemmin tietokannan tiedot on hankkeessa kopioitu myös luvussa 3.2 esiteltyyn graafitietokanta Neo4j:hin, joka on käytössä myös tässä tutkielmassa kehittämässäni sovelluksessa.

EurOMo-hankkeen sisältämä data käsittää suurimpia uutismedioita läpi Euroopan sekä niiden henkilö- ja yritysomistajien tietoja. Koska omistussuhteet ovat moninaisia, omistuskuvioiden kokonaisuutta on vaikea hahmottaa relaatiotietokannan tarjoamasta taulukkomahdollisuudesta. Esimerkiksi tilanteessa, jossa yritys omistaa useampaa eri yritystä, jotka taas omistavat toisia yrityksiä, on käyttäjän haastavaa löytää kokonaiskuvaa omistuksista, kun tiedot ovat vain taulukkomuodossa tekstinä.

Kokonaisuutta tai pidempiä omistusketjuja on siis vaikea löytää pelkästään taulukon ja relaatiotietokannan perusteella. Tähän ongelmaan graafitietokanta ja sen visualisointi tuovat helpomman tavan perehtyä tutkimuksen kaltaiseen tietoon, jossa suhteet ovat merkittävässä roolissa. Kuva 6 näyttää tulokset graafitietokantaan tehdystä hausta ja tuloksen visualisoinnista kehittämässäni sovelluksessa. Nämä tulokset ovat vastaus hakuun, jossa on haluttu selvittää yritysomistaja PunaMusta Media Oyj:n sekä uutismedia Aamulehden

väliset polut. Esimerkistä hahmottaa, miten visuaalinen kuva on merkityksellinen tulosten tulkinnassa.

Kuva 6. Haun tulos visualisoituna kuvana.

6.2 Tietokannan sisältämä data

Tutkielman tietokannan tarkoitus on selvittää eri medioiden omistussuhteita: kuka omistaa mitäkin ja kuinka paljon sekä mistä maasta omistus on lähtöisin. Graafitietokanta sisältääkin neljää eri tyyppiä olevaa solmua: uutismediat (outlets), yritysomistajat (legal owners), henkilöomistajat (persons) sekä maat (country). Suhdetyypit näiden välillä ovat ”omistaa” (owns) sekä ”mistä kotoisin” (from). Jokaisella solmutyypillä on omat ominaisuutensa. Uutismedioiden ominaisuuksia on esitelty taulukossa 2 tarkemmin. Tietokannan tarkoituksen näkökulmasta oleellisia tietoja ovat muun muassa vaikuttamiset päätoimittajan (editor-in-chief) valintaan sekä liittyminen puolueisiin, kirkkoon tai eturyhmään.

	Aamulehti	**Dagens Nyheter**	**La Repubblica**
Country	FI	SE	IT
Staff	79	Information not available	No information
CEO(s)	Pia Kalsta	Anders Eriksson	Corrado Corradi
Editor(s)-in-chief	Jussi Tuulensuu	Peter Wolodarski	Maurizio Molinari
Newsroom participation in the choice of editor-in-chief	No newsroom participation	Not applicable	Some newsroom participation / veto
Affiliation to party, church or interest group	No evidence of affiliation	No evidence of affiliation	FIAT, Exor NV, ECA

Taulukko 2. Tietokannan esimerkkidataa uutismedioista.

Taulukossa 3 on esimerkkejä yritysomistajien ominaisuuksista tietokannassa. Näistä oleellisimpia ovat yrityksen kokoluokka, julkisen rahoituksen määrät sekä liittyminen puolueisiin, kirkkoon tai etujärjestöihin. Henkilöiden ominaisuuksia tietokannassa ovat etu- ja sukunimi. Maiden ominaisuuksia ovat maan nimi sekä maakoodi eli lyhenne maan nimestä.

	Sanoma Oyj	**Radio Subasio S.r.l.**	**Aftonbladet Hierta AB**
VAT	FI15243611	IT00419950548	SE556100112301
Legal type	3 Limited liability companies	3 Limited liability companies	3 Limited liability companies
Revenue 2020	Information not available	0,02	146,7
Hired stuff	198	No information	279
Public funding	No information	0,290	0,4
Affiliation to party, church or interest group	No evidence of affiliation	No evidence of affiliation	No evidence of affiliation

Taulukko 3. Tietokannan esimerkkidataa yritysomistajista.

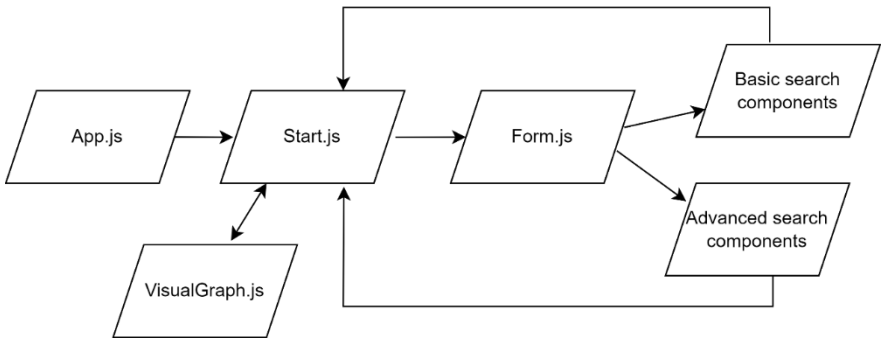
6.3 Sovelluksen vaatimusmäärittely ja teknologiat

Tutkielman tiedonhakuun ja sen visualisointiin keskittyvä sovellus on toteutettu käyttäen JavaScriptin React-kirjastoa käyttäen SPA-logiikkaa (*single page app*). Tietokantana toimii graafitietokanta Neo4j ja yhteyden muodostamisessa ja tiedon visualisoinnissa toimii luvussa 5 tarkemmin esitelty Neovis.js. Sovellus on toteutettu englanniksi kansainvälisen tutkimustaustan vuoksi. Lisäksi sovellus on tulevaisuudessa mahdollisesti tulossa jossain muodossa käyttöön EurOMo-hankkeen verkkosivustolle.

Sovellus on jaettu kahdenlaiseen hakutoimintoon, joita käyttäjä voi käyttää tiedonhaussa. ”Basic search” (jatkossa perushaku) sekä ”Advanced search” (jatkossa edistynyt haku) on toteutettu vastaamaan erilaisiin tiedonhaun tarpeisiin ottaen huomioon tietokannan tiedon luonne. Käyttäjän tutustuessa tietokannan tietoihin ja hakumahdollisuuksiin sekä muodostuviin tuloskaavioihin perushaku antaa mahdollisuuden käyttää ennalta määriteltyjä hakuja. Niiden avulla saa kattavan kuvan tietojen välisistä yhteyksistä. Edistynyt haku antaa käyttäjälle mahdollisuuden määritellä itse haettavat tiedot ja niiden väliset yhteydet. Edistynyt haku sisältää kuitenkin oletuksen, että käyttäjä tuntee tietokannan tiedon solmu-suhde-tyyppisen luonteen.

Sovelluksen rakenne on suunniteltu mukailemaan React-kirjaston omaa suositusta tiedostojen ja kansioiden sijoittelusta [React, 2023]. Suositus on kuitenkin hyvin vapaa,

joten rakenteeseen on vaikuttanut sovelluksen tärkeimmän tehtävän eli tiedon visualisoinnin tuoma näkökulma. Tiedon kulkua sovelluksessa on kuvattu kaaviona kuvassa 7. Sovellus käynnistyy alkaen App.js-tiedostosta ja rakentuu siitä alkaen hakukomponentteihin asti ketjuna. Yksinkertaistettuna tiedon kulku lähtee käyttäjän tekemästä hausta, kun hän valitsee haettavat tiedot valitsemassaan hakukomponenteissa. Sieltä käyttäjän antamat tiedot palaavat Start.js-tiedostoon, jossa annettujen tietojen perusteella muodostetaan Cypher-hakulauseke. Start.js lähettää hakulausekkeen VisualGraph-komponentille, joka vastaa visualisoinnista ja palauttaa visualisoinnin takaisin Start.js-tiedostoon. Start.js vastaa komponenttien renderöinnistä käyttäjän nähtäville.



Kuva 7. Tiedon kulku sovelluksen komponenttien välillä.

Ennen sovelluksen kehityksen aloittamista toteutin tutkielman ohjaajan kanssa vaatimusmäärittelyn sovellukselle. Tässä vaiheessa asetetut vaatimukset kohdistuivat lähinnä hakutoimintoihin ja niiden antamiin mahdollisuuksiin. Käyttöliittymän ulkonäkö jätettiin vapaasti toteutettavaksi ja yksinkertaiseksi, sillä EurOMo-hankkeen asettamista vaatimuksista sovelluksen ulkonäölle ei ollut vielä tietoa.

Vaatimukset on koottu yhteen taulukkoon 4. Ne on luokiteltu niiden tarpeellisuuden perusteella välttämättömiin, suositeltuihin sekä lisäominaisuuksiin. Välttämättömät vaatimukset on sovelluksen toteutettava ja suositeltavat vaatimukset olisivat tärkeitä toteuttaa. Lisäominaisuudet jäävät ominaisuuksiksi, jotka voi toteuttaa myöhemmin käytettävissä olevan ajan puitteissa. Vaatimukset on lisäksi luokiteltu tyypeittäin toiminnallisiin ja ei-toiminnallisiin sekä reunaehtoihin ja rajoitteisiin. Toiminnalliset vaatimukset kuvaavat sovelluksen toimintoja ja kertovat mitä sillä voi tehdä. Ei-toiminnalliset vaatimukset taas kuvaavat esimerkiksi sovelluksen laatua ja turvallisuutta. Näitä ei-toiminnallisia vaatimuksia on tässä vaatimusmäärittelyssä listattu lähinnä suositeltavana ominaisuutena, sillä kehitystyöhön käytetty aika on ollut rajallista ja osaan tulee vaikuttamaan EurOMo-hankkeen sisältä myöhemmin tulevat vaatimukset. Reunaehdot ja rajoitteet antavat tekniset raamit sovellukselle, kun käytetyt teknologiat ja käyttöympäristöt määritellään.

Nro	Kuvaus	Tarpeellisuus	Tyyppi
1	Sovellus pyörii yliopiston omalla palvelimella	välttämätön	reunaehd. ja rajoit.
2	Sovellus käyttää yliopiston palvelimen Neo4j-tietokantaa	välttämätön	reunaehd. ja rajoit.
3	Käytetyt teknologiat ovat React-kirjasto, Neo4j-tietokanta, Neovis.js-visualisointityökalu	välttämätön	reunaehd. ja rajoit.
4	Sovellus on selaimella käytettävä	välttämätön	reunaehd. ja rajoit.
5	Sovelluksen kieli on englanti	välttämätön	ei-toiminnallinen
6	Sovellus on saavutettava	suositeltava	ei-toiminnallinen
7	Visualisoinnin tuloskuvassa solmujen värit kertovat solmun tyypin	välttämätön	ei-toiminnallinen
8	Tulossolmujen ja -suhteiden nimet ovat näkyvissä	välttämätön	ei-toiminnallinen
9	Käyttäjä voi tehdä haun sanan osalla	suositeltava	toiminnallinen
10	Käyttäjä voi hakea uutismedian ja sen suhteet halutulla laajuudella	välttämätön	toiminnallinen
11	Käyttäjä voi hakea henkilön ja hänen suhteensa halutulla laajuudella	välttämätön	toiminnallinen
12	Käyttäjä voi hakea yritysomistajan ja sen suhteet halutulla laajuudella	välttämätön	toiminnallinen
13	Käyttäjä voi hakea kahden maan väliset suhteet	välttämätön	toiminnallinen
14	Käyttäjä voi hakea suhteet henkilön ja uutismedian välillä	välttämätön	toiminnallinen
15	Käyttäjä voi hakea suhteet uutismedian ja yritysomistajan välillä	välttämätön	toiminnallinen
16	Käyttäjä voi muodostaa yksityiskohtaisen haun edistyneen haun avulla	välttämätön	toiminnallinen
17	Käyttäjä voi tutkia hakutuloksen solmujen ominaisuuksia	välttämätön	toiminnallinen
18	Klikkaamalla tuloksen yksittäistä solmua käyttäjä voi laajentaa kyseisen solmun näkyviä suhteita	lisäominaisuus	toiminnallinen
19	Käyttäjä voi määritellä hakuun solmujen ja suhteiden ominaisuuksia erisuuruusvertailulla	lisäominaisuus	toiminnallinen

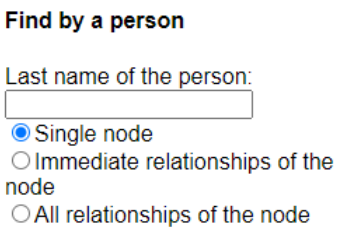
Taulukko 4. Sovelluksen vaatimukset.

7 Perushaku

Perushaku sisältää seitsemän erilaista ennalta annettua hakumahdollisuutta, joissa käyttäjä voi määritellä haettavien tietojen nimet. Tässä luvussa esitellään näitä hakumahdollisuuksia tarkemmin. Luvussa 7.1 taustoitetaan perushakuun tehtyjä valintoja yleisellä tasolla ja luvussa 7.2 käydään tarkemmin läpi hakuvaihtoehtoja käyttötapausten kautta.

7.1 Taustaa ja perusperiaatteet

Ennalta annetut hakumahdollisuudet mahdollistavat haut, joiden toiminta ja sopivuus tietokantaan on varmistettu. Etuna on, ettei käyttäjän tarvitse tietää täysin tietokannan tiedon luonnetta etukäteen, vaan voi hakuja tekemällä tutkia löydöksiä. Kuvassa 8 on osa käyttöliittymää perushaussa.



Kuva 8. Käyttöliittymä henkilön suhteiden hakemiseen perushaussa.

Perushakuun valitut hakumahdollisuudet on valittu pohtien tietokannan tietoja ja EurOMo-hankkeen alkuperäistä tiedon tallettamisen tarkoitusta. Käyttäjän on mahdollista tehdä haku tietokannan kolmesta eri solmutyypistä: uutismediasta, henkilöstä tai yritysomistajasta. Näistä kaikissa kolmessa on mahdollisuus hakea yksittäinen solmu, solmun lähimmät välittömät suhteet tai solmun kaikki suhteet. Solmuista on mahdollista tehdä myös osasanahaku, joka kattaa kaikki solmutyypit. Haku antaa tulokseksi kaikki solmut, joiden nimessä on annettu sanan osa.

Kolme viimeistä hakumahdollisuutta ovat eri solmujen välisten suhteiden hakemiseen ja kartoittamiseen sopivia. Haku kahden maan välisistä suhteista antaa annettujen maiden ja yritysomistajien suhteet. Tulokseen tulevat yritysomistajat, joilla on omistussuhteita toisiinsa ja jotka ovat toisesta annetusta maasta kotoisin. Vastaavasti haut henkilön ja uutismedian välisistä suhteista sekä uutismedian ja yritysomistajan välisistä suhteista antavat omistussuhteet annettujen solmujen välillä.

7.2 Käyttötapaukset

Tässä luvussa kuvataan erilaisia käyttötapauksia kirjallisesti kuvaamalla käyttäjän ja sovelluksen toimintaa. Ensin käydään läpi yleisiä asioita liittyen käyttötapauksiin ja sen jälkeen esitellään seitsemän käyttötapausta, jotka kattavat sovelluksen hakumahdollisuudet

hyvin. Käyttötapauksien yhteydessä annetaan kuva käyttöliittymästä liittyen hakutilanteeseen sekä kuva hakutuloksesta.

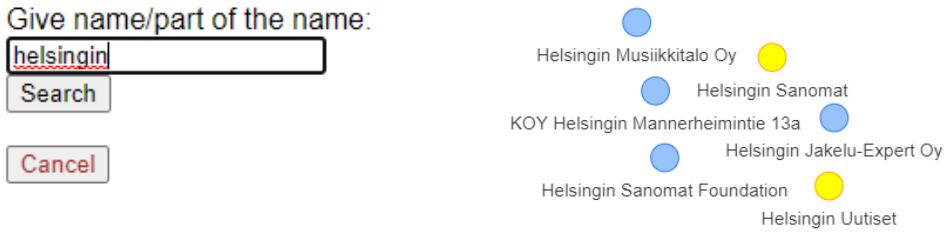
Kaikissa käyttötapauksissa toimijana on sovelluksen käyttäjä eikä sitä enää erikseen mainita myöhemmin. Lisäksi jokaisen käyttötapauksen esiehtona pidetään käyttäjän perustietämystä sovelluksen käyttötarkoituksesta ja sen sisältämästä tiedosta. Tätäkään ei uudelleen erikseen mainita. Käyttötapaukset sisältävät sanallisen kuvauksen käyttäjän toimista sekä yleisiä perusteluja haun tarpeellisuudesta ja hyödyistä käyttäjälle. Lisäksi käyttötapauksen yhteydessä annetaan käyttäjän antaman syötteen perusteella muodostettu Cypher-kysely. Näihin Cypher-kyselyihin on muokattu muuttujaa niin, että käyttäjän antama syöte on suoraan osa kyselyä. Sovelluksen koodissa näiden tilalla on luonnollisesti muuttuja, johon asetetaan aina suorituksen yhteydessä käyttäjän antamat syötteet.

Kaikkiin käyttötapauksiin pätee myös samanlainen poikkeuksien käsittely. Jos käyttäjä kirjoittaa haettavan käsitteen väärin, sovellus ei anna tuloskuvaa ollenkaan. Vielä sovellus ei myöskään anna näissä tilanteissa virheilmoituksia, joten tämä on tulevaisuuden kehityskohde sovelluksessa.

Käyttötapaukset uutismedioiden, henkilöiden ja yritysomistajien hakumahdollisuuksista on toteutettu niin, että kunkin solmutyypin kohdalla toteutetaan yksi kolmesta hakutavasta. Ensimmäiseksi haetaan yksittäinen solmu, seuraavaksi solmun lähimmät suhteet ja viimeisen solmutyypin kohdalla solmun kaikki suhteet.

Hakeminen sanan osalla

Yleistä: Haku sanan osalla on tarpeellinen hakutapa silloin, kun käyttäjä ei ole varma hakemansa asian tarkasta kirjoitusasusta. Hakemalla sanan osalla ”helsingin” saadaan kuvan 9 mukainen hakutulos, mistä käyttäjä voi kirjoitusasun tarkastaa. Koska sovellus ja tietokanta vaativat, että hakusanan kirjoitusasu on täysin oikein, huomioiden myös isot alkukirjaimet, on osasanahaku hyödyllinen käyttäjälle juuri kirjoitusasun tarkastamiseen.
Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ”Search with only part of the word”. Aukeavaan tekstikenttään käyttäjä kirjoittaa haluamansa hakusanan ja klikkaa hakupainiketta ”Search”. Avattu tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.
Cypher: `MATCH (x) WHERE x.Name = ~'(?i).*helsingin.*' RETURN x`
Lopputulos: Käyttäjä on valinnut hakusanaksi sanan, jolla löytyy useampi tulos tietokannasta ja kaikki tulokset näkyvät ruudulla. Näistä käyttäjä saa selville haluamansa solmun täyden koko nimen mahdollisia jatkohakuja varten viemällä hiiren halutun solmun kohdalle.



(a) (b)

Kuva 9. Haku sanan osalla, käyttöliittymän kuva (a) ja tuloskuva (b).

Haku uutismedian nimellä

Yleistä: Yksittäisen solmun haku on mahdollinen kaikille kolmelle solmutyypille. Se on hyödyllinen, kun käyttäjä haluaa lisätietoa tietystä uutismediasta, henkilöstä tai yritysomistajasta. Solmusta saatavat lisätiedot aukeavat kuvan 10 mukaisesti, kun hiiren osoittimen vie solmun kohdalle

Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ”Find by outlet”. Aukeavaan tekstikenttään käyttäjä kirjoittaa haluamansa uutismedian nimen ja valitsee haettavaksi ”Single node” eli yksittäisen solmun, jolla on käyttäjän kirjoittama nimi. Käyttäjä klikkaa hakupainiketta ”Search” ja tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.

Cypher: `MATCH (x:Outlets{Name:'Aamulehti'}) RETURN x`

Lopputulos: Ruutuun visualisoituu hakutulos, joka vastaa käyttäjän antamaa hakuehtoa. Yksittäisen solmun haku näyttäytyy yhtenä solmuna, jonka tarkempia tietoja voi tarkastella viemällä hiiren solmun kohdalle.

(a) (b)

Kuva 10. Yksittäisen uutismediasolmun haku, käyttöliittymän kuva (a) ja tuloskuva (b).

Haku henkilön sukunimellä

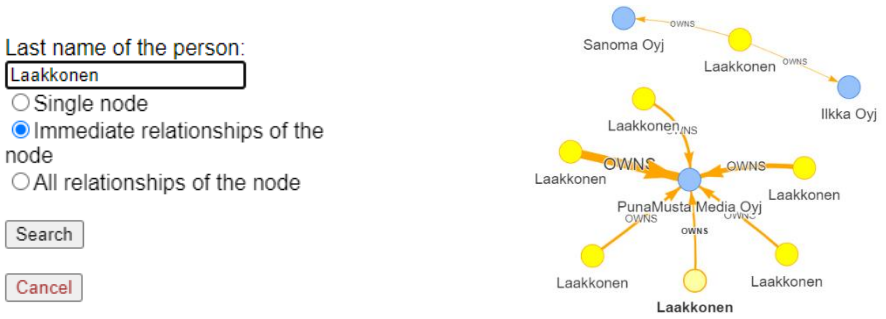
Yleistä: Välittömien suhteiden hakeminen onnistuu kaikilla kolmella eri solmutyypillä. Se on hyödyllinen näyttäessään halutun solmun läheiset suhteet. Käyttäjä saa näin hyvän kuvan solmuun liittyvistä muista solmuista ja suhteista niiden välillä. Kuvassa 11 on haettu välittömät suhteet sukunimellä ”Laakkonen”. Henkilöitä löytyy useita samannimisiä ja kuva kertoo heidän omistuksistaan. Läheisten suhteiden haussa hakutulos pysyy usein

maltillisen kokoisena ja solmuja ja niiden välisiä suhteita on helppo tarkastella tarkemmin.

Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ”Find by the person”. Aukeavaan tekstikenttään käyttäjä kirjoittaa haluamansa henkilön sukunimen ja valitsee haettavaksi ”Immediate relationships of the node” eli välittömät suhteet solmusta, jolla on käyttäjän kirjoittama nimi. Käyttäjä klikkaa hakupainiketta ”Search” ja tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.

Cypher:
```
MATCH (x:Persons{Name:'Laakkonen'}) -[i:OWNS]-> (y)
RETURN x, i, y
```

Lopputulos: Ruutuun visualisoituu hakutulos, joka vastaa käyttäjän antamaa hakuehtoa. Valitun solmun välittömät suhteet sisältävät solmut, joilla on suora yhteys haettuun henkilöön.



(a) (b)

Kuva 11. Välittömien suhteiden hakeminen henkilön sukunimellä, käyttöliittymän kuva (a) ja tuloskuva (b).

Haku yritysomistajan nimellä

Yleistä: Kaikkien suhteiden haku onnistuu kaikilla kolmella solmutyypillä. Se on hyödyllinen hakutapa, sillä kokonaiskuvan omistuksista saa muodostettua hyvin. Kuvassa 12 on haettuna kaikki suhteet yritysomistajalle Keskisuomalainen Oyj. Kuten kuvasta näkee, tarkempaan tarkasteluun tämä hakutapa ei aina sovellu, sillä hakutuloksia voi olla erittäin paljon. Tällöin yksittäisen solmun tietoja ja suhteita on vaikea tarkastella kuvan ollessa näin laaja.

Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ”Find by the legal owner”. Aukeavaan tekstikenttään käyttäjä kirjoittaa haluamansa yritysomistajan nimen ja valitsee haettavaksi ”All relationships of the node” eli kaikki suhteet liittyen solmun, jolla on käyttäjän kirjoittama nimi. Käyttäjä klikkaa hakupainiketta ”Search” ja tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.

Cypher:
```
MATCH (x:Legal_owners{Name:'Keskisuomalainen Oyj'})
-[i:OWNS*]-> (y)
MATCH (y) -[j:OWNS|FROM*]-> (z)
RETURN x, y, z, i, j
```

Lopputulos: Ruutuun visualisoituu hakutulos, joka vastaa käyttäjän antamaa hakuehtoa. Tulos sisältää valitun solmun kaikki suhteet, joten siinä on kaukaisiakin yhteyksiä solmujen välillä.

(a) (b)

Kuva 12. Kaikkien suhteiden hakeminen yritysomistajan nimellä, käyttöliittymän kuva (a) ja tuloskuva (b).

Haku kahden maan välisistä suhteista

Yleistä: Haku kahden maan välisistä suhteista on erinomainen tapa helpottaa omistuskoukeroiden selvittämistä eri maiden välillä. Kuvassa 13 on tehty haku Suomen ja Ruotsin välisistä suhteista. Hakutulos on kiinnostava ja antaa nopeasti hyvän yleiskuvan yrityksistä, joissa omistukset ovat liittyneenä molempiin maihin. Esimerkiksi Suomessa televisiotarjontaa tuottava MTV Oy on ruotsalaisen TV4 Media AB:n omistama. On perusteltua, että ylikansalliset omistukset ovat läpinäkyviä ja tiedossa. Suomessa tämä ei tällä hetkellä ole suuri ongelma, mutta Keski-Euroopassa on useita mediatoimijoita, joilla on merkittävää julkaisutoimintaa ja jotka ovat täysin naapurimaan omistuksessa. Median riippumattomuus vaatii omistussuhteiden läpinäkyvyyttä.

Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ”Find relationships between two countries”. Hakuun aukeaa kaksi tekstikenttää, joihin käyttäjä kirjoittaa kahden haluamansa maan nimet englanniksi. Käyttäjä klikkaa hakupainiketta ”Search” ja tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.

Cypher:
```
MATCH (x:Country{Name:'Finland'}) <-[r:FROM]- (z:Legal_owners)
  -[o:OWNS]- (w:Legal_owners) -[e:FROM]->
  (y:Country{Name:'Sweden'})
RETURN x, y, z, w, r, o, e
```

Lopputulos: Ruutuun visualisoituu hakutulos, joka vastaa käyttäjän antamaa hakuehtoa. Kahden maan väliset suhteet sisältävät solmut ja suhteet, jotka ovat välittömässä yhteydessä käyttäjän antamiin molempiin maihin.

(a) (b)

Kuva 13. Kahden maan välisten suhteiden hakeminen, käyttöliittymän kuva (a) ja tuloskuva (b).

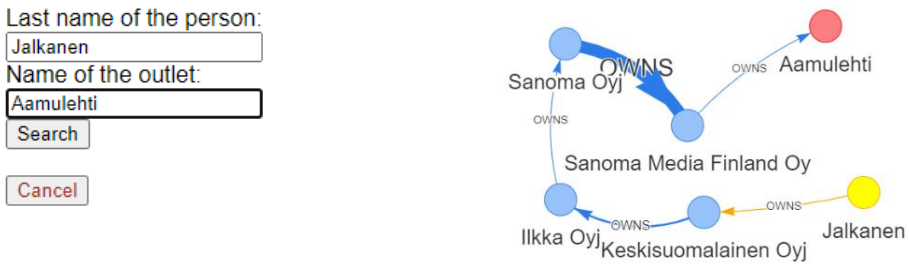
Haku henkilön ja uutismedian välisistä suhteista

Yleistä: Henkilöiden ja uutismedioiden välisten suhteiden näyttäminen voi paljastaa pitkänkin polun niiden väliltä. Kuvassa 14 on hakutulos henkilön nimeltä Jalkanen ja uutismedia Aamulehden välisistä suhteista. Haku paljastaa epäsuoran omistussuhteen henkilön ja uutismedian väliltä. Nämä voivat olla merkittäviä ja niiden läpinäkyvyys on tärkeää. Henkilölle itselleen vain suora omistussuhde voi olla merkittävä esimerkiksi rahallisesti, mutta hakua vastaavat epäsuorat omistussuhteet tuovat ilmi polut, joissa vaikutusmahdollisuudet medioiden välillä ilmenevät.

Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ”Find relationships between person and outlet”. Hakuun aukeaa kaksi tekstikenttää, joista toiseen käyttäjä kirjoittaa henkilön sukunimen ja toiseen uutismedian nimen. Käyttäjä klikkaa hakupainiketta ”Search” ja tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.

Cypher:
```
MATCH (x:Outlets{Name:'Aamulehti'}) <-[i:OWNS*]-
(y:Legal_owners) <-[j:OWNS*]- (z:Persons{Name:'Jalkanen'})
RETURN x, y, z, i, j
```

Lopputulos: Ruutuun visualisoituu hakutulos, joka vastaa käyttäjän antamaa hakuehtoa.



(a) (b)

Kuva 14. Henkilön ja uutismedian välisten suhteiden hakeminen, käyttöliittymän kuva (a) ja tuloskuva (b).

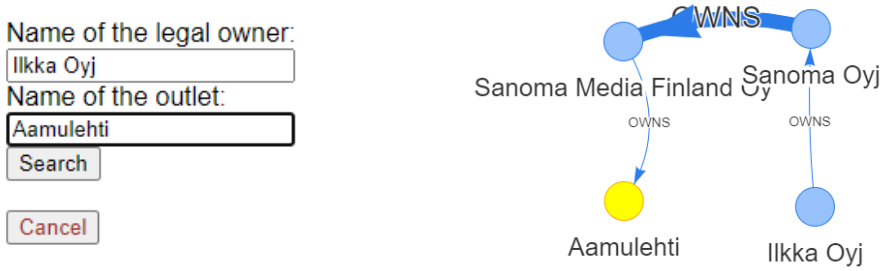
Haku uutismedian ja yritysomistajan välisistä suhteista

Yleistä: Uutismedian ja yritysomistajien väliset suhteet ovat vastaavassa roolissa kuin aiemmin esitellyt henkilöiden ja uutismedioiden väliset suhteet. Kuvassa 15 on hakutulos uutismedia Aamulehden ja yritysomistaja Ilkka Oyj:n välillä olevista suhteista. Myös yritysten välisten omistusten selventäminen tuo läpinäkyvyyttä omistussuhteisiin ja yritysten välisiin vaikutusmahdollisuuksiin.

Kuvaus: Käyttäjä klikkaa ensin ”Basic search” ja sitten ” Find relationships between legal owner and outlet”. Hakuun aukeaa kaksi tekstikenttää, joista toiseen käyttäjä kirjoittaa yritysomistajan nimen ja toiseen uutismedian nimen. Käyttäjä klikkaa hakupainiketta ”Search” ja tekstikenttä sulkeutuu. Hakutulos näkyy visualisoituna kuvana keskellä ruutua.

Cypher:
```
MATCH (x:Outlets{Name:'Aamulehti'}) <-[i:OWNS*]-
(y:Legal_owners) <-[j:OWNS*]- (z:Legal_owners{Name:'Ilkka Oyj'})
RETURN x, y, z, i, j
```

Lopputulos: Ruutuun visualisoituu hakutulos, joka vastaa käyttäjän antamaa hakuehtoa.



(a) (b)

Kuva 15. Uutismedian ja yritysomistajan välisten suhteiden hakeminen, käyttöliittymäkuva (a) ja tuloskuva (b).

8 Edistynyt haku

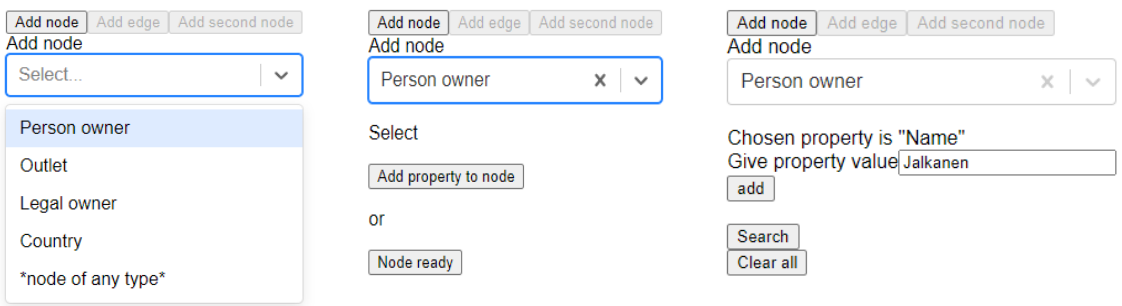
Toinen hakuvaihtoehto on edistynyt haku. Se antaa käyttäjälle mahdollisuuden muodostaa *ad hoc* -kyselylause osa kerrallaan ilman Cypher-kyselykielen osaamista. Edistynyt haku vastaa erityisesti käyttäjän tarpeeseen etsiä ja löytää tietoa, mitä perushaun ennalta määritellyissä vaihtoehdoissa ei ole annettu.

Tässä luvussa käydään tarkemmin teoriaa tämän edistyneen haun taustalla sekä avataan luotua kielioppia tarkemmin. Lisäksi annetaan esimerkkejä hauista, joista muodostetaan Cypher-kyselylauseke kielioppia käyttäen. Edistyneen käyttöliittymä ja tekstimuotoisen hakulausekkeen muodostaminen esitellään luvussa 8.1, sovelluksen taustalla oleva kielioppi käydään läpi luvussa 8.2 ja esimerkkikyselyitä evaluoidaan kieliopin avulla luvussa 8.3.

8.1 Edistyneen haun toteutus sovelluksessa

Edistynyt haku on kehitetty muodostamaan käyttäjän antamista parametreista hakulausekkeita Cypherillä alla esitellyn kieliopin mukaisesti. Käyttäjän oletetaan tietävän solmu-suhde-rakenteen ja ymmärtävän tietokannan sisältämän tiedon merkityksen.

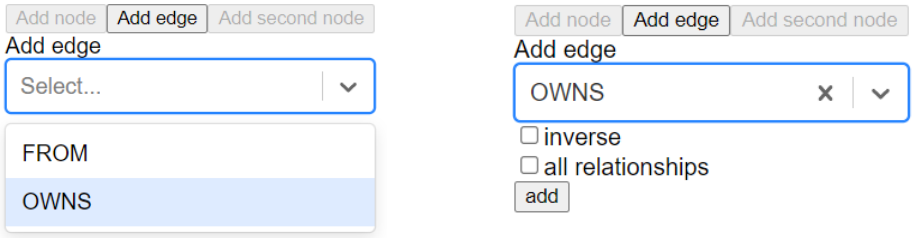
Käyttäjä päättää itse haetaanko vain yhtä solmua vai useampaa solmua ja kaarta niiden välillä. Jokaisesta lisättävästä solmusta tai kaaresta muodostetaan oma lohkonsa hakulausekkeesta. Ensimmäisenä valitaan hakulausekkeen ensimmäinen solmu. Kuvassa 16 a-osa näyttää käyttöliittymän solmun tyypin valinnan. Ensimmäisen valinnan jälkeen käyttäjä voi valita b-osan mukaisesti määritelläänkö solmulle vielä ominaisuus vai jätetäänkö hakuun vain solmutyyppi ilman ominaisuuksia. Kuvan 16 c-osa näyttää solmun ominaisuuden arvon määrittelyn. Ominaisuudet on sovelluksessa rajattu kaikille solmuille vain niiden nimeen ja muiden ominaisuuksien hyödyntäminen haussa jää tulevaisuuden kehityskohteeksi.



(a) (b) (c)

Kuva 16. Solmun tyypin valinta (a), valinta määritelläänkö solmulle ominaisuus vai ei (b) ja ominaisuuden arvon antaminen (c).

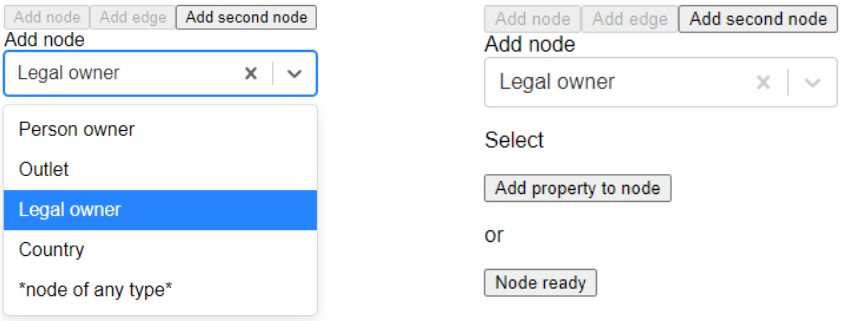
Kun solmu on lisätty, tekstimuotoinen hakulauseke koostuu yhdestä lohkosta, jossa on valittuna solmu: Persons (Name = Jalkanen). Tässä vaiheessa käyttäjä voi päättää hakea vain tätä yhtä solmua tai hän voi jatkaa hakulauseketta lisäämällä kaaren ja toisen solmun. Kaaren osalta on mahdollista valita sen tyyppi ja suunta. Kuvan 17 a-osassa kaareksi valitaan ”OWNS”, minkä jälkeen tulee mahdolliseksi kääntää kaaren suunta vastakkaiseksi. Kuvan 17 b-osassa näytetään mahdollisuus valita kaarelle lisävalintana kaaren suunnan muutos tai kaaren laajuuden muutos kaikkiin solmujen välisiin suhteisiin. Oletuksena kaaren suunta on vasemmalta oikealle ja inverse kääntää suunnan toisin päin. Jos kaaren laajuudeksi valitaan kaikki suhteet, haku sisältää suhteet laajemmin.



(a) (b)

Kuva 17. Kaaren valinta käyttöliittymässä (a) ja kaaren lisävalinnat (b).

Kaaren jälkeen valitaan sitä seuraava solmu vastaavasti kuin ensimmäinen solmu aiemmin. Solmun tyyppi valitaan ensimmäiseksi kuten kuvan 18 a-osassa ja sitten joko jatketaan solmun ominaisuuden määrittelyyn tai päätetään hakea ilman tarkempaa ominaisuutta. Kun ominaisuutta ei määritellä, haetaan kaikkia solmuja, jotka ovat annettua tyyppiä. Valittaessa jälkimmäinen vaihtoehto muodostuva lohko tekstimuotoiseen hakulausekkeeseen on ”Legal owners”.



(a) (b)

Kuva 18. Toisen solmun lisääminen, tyypin valinta (a) ja valintamahdollisuus jättää ominaisuus määrittelemättä tältä solmulta (b).

Kokonaisuudessaan esimerkein muodostettu tekstimuotoinen hakulauseke on ”Persons (Name = Jalkanen) OWNS Legal owners”. Tämän sovellus muuntaa Cypher-kyselykielelle kaavion 8 mukaisesti ja suorittaa haun tietokantaan.

```
MATCH (a:Persons{Name:'Jalkanen'}) -[b:OWNS]->
(c:Legal_owners)
RETURN a, b, c
```

Kaavio 8. Muodostettu Cypher-kysely.

Hakutulos esitetään visuaalisesti kuvassa 19 samoin kuin perushauissa luvussa 7.2. Hakutulos on siis vastaus kyselyyn, jolla haluttiin selvittää kaikki yritysomistajat, joita henkilö nimeltä Jalkanen omistaa. Tässä esimerkissä esitettiin vain yhden kaaren pituinen hakulause. Hakulauseen pituudelle ei ole kuitenkaan asetettu rajoituksia eli hakua olisi voitu jatkaa antamalla uusia kaaria ja solmuja sekä niiden mahdollisia ominaisuusrajoitteita.

Kuva 19. Hakutulos muodostetusta kyselystä.

8.2 Kontekstivapaa kielioppi ja attribuuttikielioppi

Ohjelmointikielien ja niiden kääntäjien taustalla on aina jokin kielioppi eli säännöt miten kieltä käytetään [Karol, 2006]. Donald Knuth [1968] kehitti kontekstivapaan kieliopin pohjalta attribuuttikieliopin, mikä on yhä nykyäänkin pätevä pohja ohjelmointikielien ja kääntäjien määrittelylle. Tässä tutkielmassa käännettävä kieli on ylläkuvattu tekstimuotoinen kysely, joka vastaa graafisesti määriteltyä polkua.

Formaalisti kontekstivapaa kielioppi määritellään monikkona G = (N, T, P, Q) [Karol, 2006], missä

- N merkitsee nonterminaalien joukkoa
- T merkitsee terminaalien joukkoa
- P merkitsee kieliopin sääntöjä
- Q merkitsee aloitussymbolia.

Lisäksi pätee $N \cap T = \emptyset$ eli nonterminaalien ja terminaalien leikkaus on tyhjä joukko. Toisin sanoen nonterminaalit ja terminaalit eivät sisällä yhteisiä tekijöitä. Toinen lisämäärittely on $Q \in N$ eli aloitussymboli kuuluu nonterminaalien joukkoon.

Säännöt P ovat muotoa A → B, missä A on nonterminaali ja B on sekvenssi, joka voi muodostua vapaasti terminaaleista ja nonterminaaleista. Lyhimmillään A on yksi terminaali tai nonterminaali. Jos nonterminaali A esiintyy myös sekvenssissä B, se numeroidaan attribuuttikieliopin yhteydessä. Kun kysely jäsennetään kieliopilla, siitä muodostuu jäsennyspuu, mikä ilmaisee missä järjestyksessä sääntöjä on sovellettu. Jäsennyspuut esitetään kuvina myöhemmin luvun 8.3 kyselyiden kieliopin mukaisen läpikäymisen yhteydessä.

Tämän tutkielman sovelluksessa nonterminaalit ja terminaalit koostuvat erikseen alla esitellyistä joukoista:

- N = {Q, M, Node, Node_label, Property, Edge}
- T = values ∪ property_names ∪ node_labels ∪ edge_labels ∪ {=, *, inverse}

Näistä terminaaleista äärelliset joukot node_labels sekä edge_labels sisältävät sovelluksen solmujen ja kaarien nimikkeet. Esimerkkisovelluksessa node_labels = {Persons, Outlets, Legal_owners, Country} ja edge_labels = {OWNS, FROM}. Property_names on ominaisuuksien nimien joukko, joka on tässä tutkielmassa rajattu sisältämään vain ominaisuuden ”Name”. Todellisuudessa solmuilla on kuitenkin lisäksi muitakin ominaisuuksia. Values taas on joukko, joka sisältää mitkä tahansa merkkijonot, jotka on annettu hauntekoa varten. Muodostetut kieliopin säännöt on esitelty taulukossa 5.

1	Q → M
2	M → Node
3	M1 → Node Edge M2
4	Node → Node_label
5	Node → Node_label Property where Node_label ∈ node_labels
6	Property → P_name = Value where P_name ∈ property_names, Value ∈ values
7	Edge → *
8	Edge → *inverse
9	Edge → Edge_name where Edge_name ∈ edge_labels
10	Edge → Edge_name inverse where Edge_name ∈ edge_labels
11	Edge → Edge_name*
12	Edge → Edge_name* inverse
13	Edge_name ∈ edge_labels

Taulukko 5. Kehitetyn attribuuttikieliopin säännöt.

Ensimmäinen sääntö sisältää aloitussymbolin Q ja siten koko aloitusfraasin, joka aiotaan muuntaa Cypheriksi. Sitä sovelletaan aina. Toista sääntöä sovelletaan, jos jäsennettävä hakulauseke sisältää vain yhden solmun. Kolmatta sääntöä sovelletaan, kun hakupolku sisältää kaaren. M1 alkaa siis solmulla (Node), jonka jälkeen on kaari (Edge) ja jota seuraa polun loppuosa (M2). Tätä sääntöä voidaan soveltaa rekursiivisesti, kunnes M2 koostuu enää yhdestä solmusta, jolloin sääntöä 2 käytetään rekursion päätösehtona. Kolmannen säännön soveltaminen tapahtuu siis silloin, kun haettavassa lauseessa on vähintään kaksi solmua ja yksi suhde. Tämän säännön M2:een sovelletaan sääntöä 2 tai sääntöä 3 uudestaan.

Neljäs, viides ja kuudes sääntö liittyvät solmuun ja sen ominaisuuksien nimeämiseen. Jos solmulle ei ole annettu ominaisuuden nimeä, sovelletaan neljättä sääntöä. Viidettä ja kuudetta sääntöä taas sovelletaan, jos solmulle on annettu tarkempi ominaisuus. Loput säännöt antavat ohjeet suhteiden muuntamiseen eri tilanteissa. Tähti (*) tarkoittaa tilannetta, jolloin sovelletaan transitiivista suhdetta. Suhteisiin liittyvät säännöt ovat pareittain, joissa toisella säännöllä on muuten identtisen säännön lisäterminä inverse. Tämä tarkoittaa suhteen käänteistä suuntaa. Oletussuunta suhteella on vasemmalta oikealle, mutta inversen avulla suunta käännetään oikealta vasemmalle.

Attribuuttikielioppi on käytännössä kontekstivapaan kieliopin laajennos. Kun kontekstivapaaseen kielioppiin lisätään attribuutteja eli merkityksiä, muodostuu attribuuttikieliopin perusta [Knuth, 1968]. Attribuutit voidaan ymmärtää myös funktioina, joilla siirretään tietoa kieliopin jäsentämisen yhteydessä. Attribuuttikieliopissa kontekstivapaata kielioppia laajennetaan mahdollisuudella kahden eri tyypin attribuutteihin. Näitä ovat perityt attribuutit ja synteettiset attribuutit. Perittyjä attribuutteja käytetään kuvaamaan tiedonkulkua syntaksin jäsennyspuurakenteen solmuissa ylhäältä alaspäin. Tämä tarkoittaa, että jos säännön A → B nonterminaaliin A on sidottu jokin ominaisuus, se voidaan siirtää perityllä attribuutilla säännön oikealle puolelle oleviin nonterminaaleihin. Synteettiset attribuutit taas kuvaavat tiedon kulkua alhaalta ylöspäin eli ne siirtävät tietoa säännön oikealla puolella olevista symboleista säännön vasemmalla puolella oleviin nonterminaaleihin. Attribuutit ovat aina joko synteettisiä tai perittyjä.

S-attribuoitu kielioppi on yksi attribuuttikieliopin versioista ja siinä käytetään ainoastaan synteettisiä attribuutteja [Karol, 2006]. Tässä tutkielmassa määritelty kielioppi on S-attribuoitu, joten kaikki esitellyt attribuutit ovat synteettisiä.

Attribuuttikieliopin käyttö on toteutettu niin, että alun tekstimuotoinen hakulauseke esitetään rakenteella `solmu[-suhde-solmu]*`, mikä tarkoittaa, että lauseessa on vähintään yksi solmu ja lisäksi mielivaltainen määrä suhde-solmupareja muodostamaan lauseen. Esimerkki tällaisesta tekstimuotoisesta hakulauseesta olisi: Persons (Name = Jalka-

nen) OWNS Legal_owners (Name = Keskisuomalainen Oyj). Tämä hakulauseke merkitsee, että etsitään henkilön, jonka nimi on Jalkanen, omistussuhde yritysomistajaan nimeltä Keskisuomalainen Oyj.

Esiteltyä kontekstivapaata kielioppia laajennetaan siis attribuuteilla, jotka liitetään aina johonkin sääntöön. Seuraavaksi käydään läpi sääntöjen attribuutit. Säännöt koostuvat nuolen vasemmalla puolella olevasta nonterminaalista ja nuolen oikealla puolella olevista nonterminaaleista tai terminaaleista. Jokainen sääntö sisältää synteettisen *query*-attribuutin, joka sisältää MATCH-lohkon tai sen osan. Lisäksi joidenkin sääntöjen yhteydessä on myös synteettinen *return*-attribuutti, joka määrittelee Cypher-kyselyyn liittyvää osaa palautettavien muuttujien suhteen. Attribuuteissa on käytetty merkkijonojen konkatenaatio-operaatiota, mitä merkitään symbolilla ⊕. Tällä kuvataan lauseen eri osien, kiinteiden sekä arvoilla täydennettyjen, yhdistämistä toisiinsa. *Query*- ja *return*-attribuuteissa on käytetty myös punaista väriä erottamaan Cypher-kyselyn kiinteät osat.

Ensimmäinen sääntö sisältää aloitussymbolin, mihin asetetaan käyttäjän antamista hakuvalinnoista saadut parametrit. *Query*-attribuutti sisältää valmiin Cypher-lauseen sisältäen sen molemmat osat MATCH sekä RETURN. Komentosana MATCH yhdistetään siis ensin attribuutin *query*(M) palauttamaan merkkijonoon operaatiolla ⊕, minkä jälkeen tulee komentosana RETURN ja lopuksi attribuutin *return*(M) palauttama merkkijono.

1. Q → M

query(Q) = **MATCH** ⊕ `query(M)` ⊕ **RETURN** ⊕ `return(M)`

Toiseen ja kolmanteen sääntöön liittyy kaksi attribuuttia: *query* ja *return*. Symboliin M voidaan soveltaa kumpaa sääntöä tahansa. Jos haettavana on vain yksi solmu, sovelletaan sääntöä 2. Useamman solmun ja suhteen haussa sovelletaan kolmatta sääntöä. Kolmatta sääntöä käytettäessä toisen ja seuraavien solmujen yhteydessä sovelletaan toista sääntöä, jolloin attribuuttien sisältö siirtyy sellaisenaan nonterminaalille M. Säännössä 3 *query*-attribuutti muodostuu solmun Node, kaaren Edge ja loppupolun M2 *query*-attribuuttien yhdistelmästä. Attribuutti *return*(M1) muodostuu vastaavasti, mutta nyt arvot erotellaan pilkuilla toisistaan.

2. M → Node

query(M) = *query*(Node)

return(M) = *return*(Node)

3. M1 → Node Edge M2

query(M1) = *query*(Node) ⊕ *query*(Edge) ⊕ *query*(M2)

return(M1) = *return*(Node) ⊕ **,** ⊕ *return*(Edge) ⊕ **,** ⊕ *return*(M2)

Säännöt 4–6 sisältävät semanttiset ohjeet solmujen määrittelemiseen. Neljättä sääntöä sovelletaan silloin, kun solmulle ei ole annettu ominaisuutta. Silloin *query*-attribuutin mukaisesti luodaan uusi muuttuja funktiolla new(). Attribuutissa *query*(Node) muuttujaan sidotaan nimiketieto käyttäen Cypherin mukaista kaksoispistemerkintää sulkujen sisällä. Attribuutti *return*(Node) saa arvokseen muuttujan sellaisenaan. Jos solmulla on nimiketiedon lisäksi määriteltynä ominaisuus, sovelletaan solmun määrittelyssä sääntöjä 5 ja 6. Viidennessä säännössä asetetaan myös nimiketieto ja luodaan uusi muuttuja, joka esitetään Cypherin tapaan aaltosulkeiden sisällä nimikkeen jälkeen. Säännössä 6 ominaisuus ja sen arvo esitetään Cypherin mukaisesti. Tämä siirtyy sääntöön 5.

4. Node → Node_label
query(Node) = `(` ⊕ `x:` ⊕ `Node_label` ⊕ `)`
return(Node) = `x`
where x = new()

5. Node → Node_label Property
query(Node) = `(`⊕ `x:` ⊕ `Node_label` ⊕ `{` ⊕ *query*`(Property)` ⊕ `})`
return(Node) = `x`
where x = new(), Node_label ∈ node_labels

6. Property → P_name = Value
query(Property) = `P_name` ⊕ `:'` ⊕ `Value` ⊕ `'`
where P_name ∈ property_names, Value ∈ values

Seitsemäs ja kahdeksas sääntö muodostavat suhteen osuuden Cypher-kyselyyn silloin, kun haetaan tarkemmin määrittelemättä mitä tahansa suhdetta. Suhteen nimiketieto korvataan vain tähtimerkillä attribuuttiin *query*(Edge). Lisäksi määritellään hakasulkeet ja nuolet Cypher-kyselyn mukaisesti. Attribuuttiin *return*(Edge) ei säännöissä 7 ja 8 sidota mitään. Näitä sääntöjä ei kuitenkaan voi viedä toteutukseen kehitetyn sovelluksen yhteydessä, sillä sovelluksen käyttämä visualisointiteknologia Neovis.js ei mahdollista suhteiden hakemista ilman niiden nimiketietojen määrittelyä.

7. Edge → *
query(Edge) = `-[ * ]->`
return(Edge) = NULL

8. Edge → * inverse
query(Edge) = `<-[ * ]-`
return(Edge) = NULL

Yhdeksättä ja kymmenettä sääntöä sovelletaan, kun suhteella on nimiketieto ja halutaan hakea yhden pituinen polku solmujen välillä. Attribuutti *query*(Edge) sitoo suhteen nimiketiedon Cypherin mukaisesti hakasulkeiden ja nuolen avulla. Näiden sääntöjen jälkeen sovelletaan viimeistä sääntöä, sääntöä 13, joka asettaa suhteen nimen attribuuttiin uuden muuttujan kanssa.

9. Edge→ Edge_name
query(Edge) = `-[ ⊕ query(Edge_name) ⊕ ]->`
return(Edge) = *return*(Edge_name)

10. Edge→ Edge_name inverse
query(Edge) = `<-[ ⊕ query(Edge_name) ⊕ ]-`
return(Edge) = *return*(Edge_name)

Yhdestoista ja kahdestoista sääntö vastaavat aiempia suhteita määritteleviä sääntöjä, mutta tässä lisänä on suhteen pituus. Nämä säännöt muodostavat Cypher-kyselylauseeseen suhteen osuuden niin, että suhde muodostuu nyt kahdesta suhteesta ja solmusta niiden välillä. Jotta kysely kattaa kaikki suhteet kahden solmun välillä, tarvitsee kysely tätä laajempaa suhdemäärittelyä, koska muuten Cypher-ilmaisu ei palauta kaikkia suhteita solmujen välillä. Tämä myös tarkoittaa, että suhteen nimi on käsiteltävä kahdesti. Säännöissä tähän erotteluun on käytetty numerointia ja kannattaa huomata, että Edge_name1 = Edge_name2 = Edge_name. Haettavien suhteiden pituus määritellään molempiin suhteisiin tähtimerkillä ja ”0..”-merkinnällä, mikä merkitsee suhteen pituutta, jonka sallitaan olevan 0 tai suurempi. Aiempien sääntöjen tapaan attribuutteihin *query(*Edge) tulee myös Cypher-kyselyn mukaisesti hakasulkeet ja nuolet. Molemmat *query*(Edge)-attribuutit määritellään samalla tavalla ja molempiin sovelletaan sääntöä 13, jolla muodostetaan suhteiden nimiketiedot ja uudet muuttujat mukaan kyselylauseisiin. Suhteiden väliin funktiolla new() muodostettava muuttuja *x* sitoo haettavat suhteet ja solmut yhteen, jotta koko polku voidaan palauttaa.

11. Edge → Edge_name*

query(Edge) = `-[` ⊕ *query*(Edge_name1) ⊕ `*0..` ⊕ `]->` ⊕
`(`⊕ x ⊕`)` ⊕ `-[` ⊕ *query*(Edge_name2) ⊕ `*0..` ⊕ `]->`

return(Edge) = *return*(Edge_name1) ⊕ `,` ⊕ x ⊕ `,` ⊕ *return*(Edge_name2)

where x = new()

12. Edge → Edge_name* inverse

query(Edge) = `<-[` ⊕ *query*(Edge_name1) ⊕ `*0..` ⊕ `]-` ⊕ `(`⊕ x ⊕`)`
⊕ `<-[` ⊕ *query*(Edge_name2) ⊕ `*0..` ⊕ `]-`

return(Edge) = *return*(Edge_name1) ⊕ `,` ⊕ x ⊕ `,` ⊕ *return*(Edge_name2)

where x = new()

Kolmastoista ja viimeinen sääntö on jo aiemmin sivuttu suhteen nimiketiedon asettava sääntö. Sitä sovelletaan aina, kun suhteen nimi on tiedossa. *Query*-attribuuttiin asetetaan uusi muuttuja funktiolla new(). Muuttuja ja nimiketieto sidotaan attribuuttiin *query*(Edge_name) lisäämällä muuttujan jälkeen kaksoispiste, minkä jälkeen lisätään nimiketieto. Muuttuja sidotaan sellaisenaan attribuuttiin *return*(Edge_name).

13. Edge_name ∈ edge_labels

query(Edge_name) = x ⊕ `:` ⊕ Edge_name

return(Edge_name) = x

where x = new(), Edge_name ∈ edge_labels

8.3 Kyselyiden evaluointi

Tässä luvussa käydään läpi esimerkkejä tehdyistä hauista ja niiden perusteella evaluoidaan semanttisten sääntöjen mukaisesti Cypher-kyselyitä. Lisäksi kyselyiden evaluoinneista esitetään kuvallisesti jäsennyspuut. Evaluoitaviksi esimerkeiksi on valittu laajuudeltaan erilaisia hakuja, jotta sääntöjen toimintaa kyselyiden muodostamisessa pystyy hahmottamaan monipuolisesti.

Yhden solmun kysely

Ensimmäisenä evaluoidaan yksinkertaisin kysely eli yhden solmun etsivä haku. Käytetään esimerkkinä henkilöä nimeltä Jalkanen. Näin ollen muodostunut tekstimuotoinen hakulauseke on muotoa: Persons (Name = Jalkanen). Tämän lauseen muuttaminen Cypher-kyselykielelle attribuuttikielioppisääntöjä käyttäen vaatii neljän säännön soveltamista. Lähtötilanteessa sovellamme siis ensimmäistä sääntöä, jossa aloitussymboli Q merkitsee mainittua tekstimuotoista hakulauseketta.

1. Q → M

Q = Persons (Name = Jalkanen)

query(Q) = `MATCH ⊕ query(M) ⊕ RETURN ⊕ return(M)`

query(Q) =
```
MATCH (x1:Persons{Name:'Jalkanen'})
RETURN x1
```

Seuraavana sovelletaan sääntöä 2, missä alustetaan solmun määritteleminen. Niin M- kuin Node-nonterminaali saavat arvokseen saman tekstimuotoisen hakulauseen kokonaisuudessaan, sillä haussa on vain yksi solmu.

2. M → Node

M = Persons (Name = Jalkanen)

query(M) = *query*(Node) = `(x1:Persons{Name:'Jalkanen'})`

return(M) = *return*(Node) = `x1`

Seuraavana sovelletaan sääntöä 5, missä solmun nimiketieto asetetaan Cypher-kyselylauseeseen uuden luodun muuttujan kanssa. Ominaisuuden tarkempi määrittely siirtyy vielä seuraavaan sääntöön. Solmun nimiketieto on Persons ja ominaisuus tekstimuotoisessa ilmauksessa on “Name = Jalkanen”. Uusi muuttuja luodaan funktiolla new() ja se siirtyy sellaisenaan attribuuttiin *return*(Node).

5. Node → Node_label Property

Node = Persons (Name = Jalkanen)

query(Node) = `( ⊕ x: ⊕ Node_label ⊕ { ⊕ query(Property) ⊕})`

query(Node) = `(x1:Persons{Name:'Jalkanen'})`

return(Node) = `x1`

Viimeisenä sovelletaan sääntöä 6, missä ominaisuuden nimi ja arvo asetetaan Cypher-kyselylauseeseen. Ominaisuuden nimi P_name on ”Name” ja sen arvo Jalkanen.

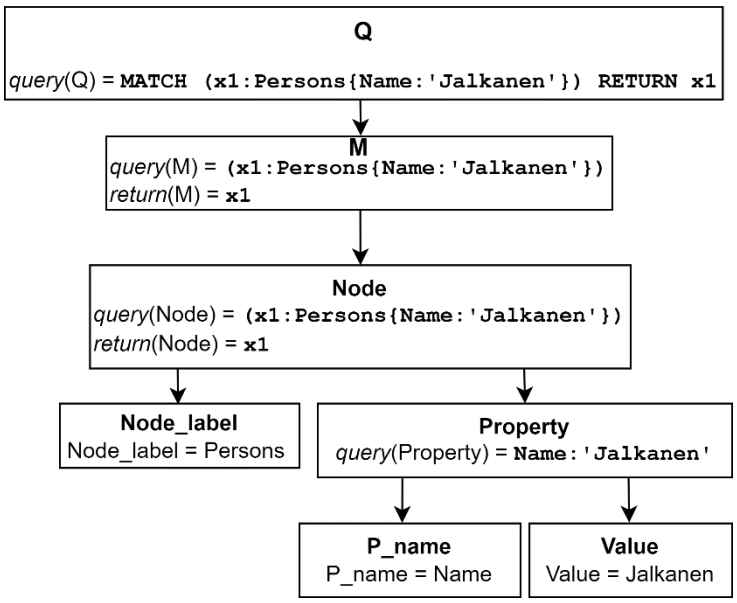
6. Property → P_name = Value

Property = Name = Jalkanen

query(Property) = `P_name ⊕ :' ⊕ Value ⊕ '`

query(Property) = `Name:'Jalkanen'`

Kun kaikki säännöt on käyty läpi, muodostuu Cypher-kyselykielelle muutettu hakulauseke: `MATCH (x1:Persons{Name:'Jalkanen'}) RETURN x1.` Kuvassa 20 on esitettynä sama evaluaatio jäsennyspuuna, missä kysely muodostuu synteettisten attribuuttien avulla, kun kyselypuu käydään läpi alhaalta ylöspäin.



Kuva 20. Persons (Name = Jalkanen) haun jäsennyspuu.

Solmu-suhde-solmu-kysely

Toinen evaluoitava hakutilanne on lause, joka sisältää kaksi solmua ja yhden suhteen niiden välillä. Otetaan tästä esimerkiksi tekstimuotoinen lauseke ”Persons (Name = Jalkanen) OWNS Legal_owners (Name = Keskisuomalainen Oyj)”. Tämä edellistä asteen monimutkaisempi haku vaatii seitsemän eri säännön soveltamista. Muutamaa näistä seitsemästä sovelletaan kahteen kertaan, sillä haku sisältää kaksi eri solmua. Nytkin aloitetaan soveltamalla ensimmäistä sääntöä, johon asetetaan koko tekstimuotoinen hakulauseke.

1. **Q → M**

Q = Persons (Name = Jalkanen) OWNS Legal_owners (Name = Keskisuomalainen Oyj)

query(Q) = **MATCH** ⊕ *query*(M) ⊕ **RETURN** ⊕ *return*(M)

```
query(Q) = MATCH (x1:Persons{Name:'Jalkanen'}) -[e1:OWNS]->
           (x2:Legal_owners{Name:'Keskisuomalainen Oyj'})
           RETURN x1, e1, x2
```

Seuraava sovellettava sääntö on kieliopin kolmas sääntö, jota sovelletaan tilanteissa, joissa solmuja on useampia. Ensimmäiselle solmulle ja suhteelle sovelletaan tämän jälkeen suoraan sääntöjä niiden muodostamiseksi, mutta toinen solmu kuvautuu tässä vaiheessa symbolille M2, mikä jäsennetään myöhemmin säännöllä 2.

3. M1 → Node Edge M2

Q = Persons (Name = Jalkanen) OWNS Legal_owners (Name = Keskisuomalainen Oyj)

query(M1) = *query*(Node) ⊕ *query*(Edge) ⊕ *query*(M2)

query(M1) = `(x1:Persons{Name:'Jalkanen'}) -[e1:OWNS]->`
`(x2:Legal_owners{Name:'Keskisuomalainen Oyj'})`

return(M1) = *return*(Node) ⊕ `,` ⊕ *return*(Edge) ⊕ `,` ⊕ *return*(M2)

return(M1) = `x1, e1, x2`

Ensimmäiselle solmulle (Node = Persons (Name = Jalkanen)) sovelletaan viidettä sääntöä nimiketiedon ja ominaisuuden nimeämiseen vastaavasti kuin aiemmin esitellyssä kyselyn evaluoinnissa.

5. Node → Node_label Property

Node = Persons (Name = Jalkanen)

query(Node) = `(⊕ x: ⊕ Node_label ⊕ { ⊕` *query*`(Property) ⊕})`

query(Node) = `(x1:Persons{Name:'Jalkanen'})`

return(Node) = `x1`

Viimeisenä ensimmäiseen solmuun sovelletaan sääntöä 6, missä ominaisuuden nimi ja arvo asetetaan Cypher-kyselylauseeseen. Ominaisuuden nimi P_name on ”Name” ja sen arvo Jalkanen.

6. Property → P_name = Value

Property = Name = Jalkanen

query(Property) = `P_name ⊕ :' ⊕ Value ⊕ '`

query(Property) = `Name:'Jalkanen'`

Suhteelle sovelletaan ensimmäisenä yhdeksättä sääntöä. Siinä muodostetaan Cypher-kyselyn suhteen ympärille tulevat hakasulut ja nuoli. Lisäksi sääntöä 13 sovelletaan suhteen nimiketiedon lisäämiseen ja uuden muuttujan luomiseen.

9. Edge→ Edge_name

Edge = OWNS

query(Edge) = `-[ ⊕ query(Edge_name) ⊕ ]->`

query(Edge) = `-[e1:OWNS]->`

return(Edge) = *return*(Edge_name)

return(Edge) = e1

13. Edge_name ∈ edge_labels

Edge_name = OWNS

query(Edge_name) = `x ⊕ : ⊕ Edge_name`

query(Edge_name) = `e1:OWNS`

return(Edge_name) = `e1`

Viimeiset kolme sovellettavaa sääntöä ovat toinen, viides ja kuudes sääntö. Toinen sääntö on aiemmin mainittu sääntö tilanteisiin, joissa kyselyssä tulee olemaan vähintään kaksi eri solmua. Vastaavasti kuin edellisessä solmussa, käytetään sääntöjä viisi ja kuusi solmun tarkempaan määrittelyyn. Viidennessä säännössä asetetaan nimiketieto solmulle ja ominaisuuden nimi ja arvo asetetaan kuudennessa säännössä. Kuvassa 21 on esitettynä evaluoitua hakua vastaava jäsennyspuu.

2. M → Node

M = Legal_owners (Name = Keskisuomalainen Oyj)

query(M) = *query*(Node)

query(M) = `(x2:Legal_owners{Name:'Keskisuomalainen Oyj'})`

return(M) = *return*(Node)

return(M) = `x2`

5. Node → Node_label Property

Node = Legal_owners (Name = Keskisuomalainen Oyj)

query(Node) = `(⊕ x: ⊕ Node_label ⊕ { ⊕ query(Property) ⊕})`

query(Node) = `(x2:Legal_owners{Name:'Keskisuomalainen Oyj'})`

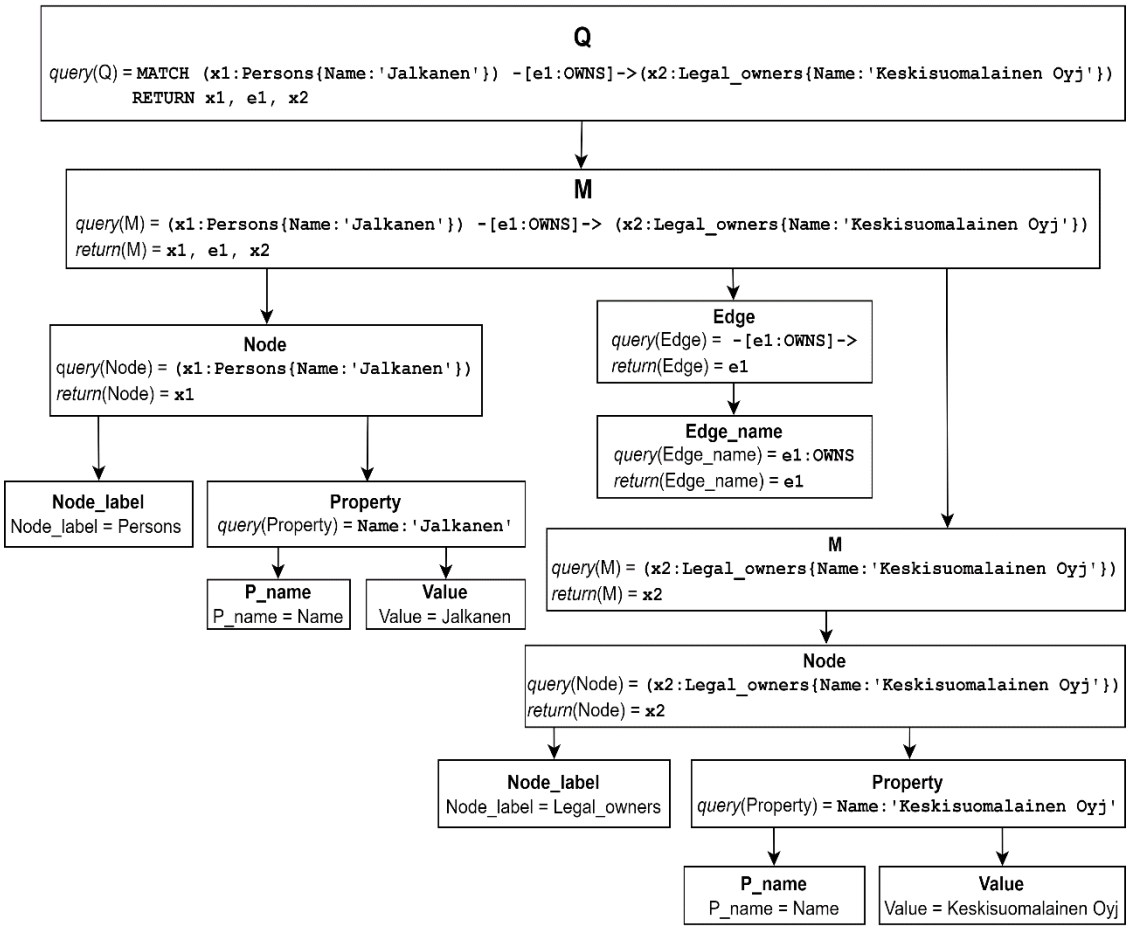
return(Node) = `x2`

6. Property → P_name = Value

Property = P_name = Keskisuomalainen Oyj

query(Property) = `P_name ⊕ :' ⊕ Value ⊕ '`

query(Property) = `Name:'Keskisuomalainen Oyj'`



Kuva 21. Persons (Name = Jalkanen) OWNS Legal_owners (Name = Keskisuomalainen Oyj) haun jäsennyspuu.

Tähtikysely nimiketiedolla

Tähtikysely nimiketiedolla on Cypher-kysely, jossa suhteen nimiketiedon lisäksi on merkittynä tähti (*). Tähtimerkinnällä tarkoitetaan tekstimuotoisessa hakulausekkeessa kaikkia polkuja annettujen solmujen välillä eli kyseessä on transitiivinen sulkeuma. Tässä evaluoinnissa käydään läpi tekstimuotoinen hakulauseke Persons (Name = Jalkanen) OWNS* Outlets (Name = Aamulehti). Kieliopin sääntöjä sovelletaan siis niin, että halutaan selvittää kaikki omistussuhteet henkilön Jalkanen ja uutismedia Aamulehden välillä.

Tämä sama kysely on esimerkkinä luvun 7.2 perushaun käyttötapauksissa ja kyselyn hakutulos näytetään kyseisessä luvussa kuvassa 14. Tässä esimerkissä käyttäjä kuitenkin rakentaa kyselyn itse, jolloin haku tai tietokanta voisi olla jokin muukin.

Ensimmäisenä sovelletaan ensimmäistä sääntöä, jossa aloitussymboli Q merkitsee jälleen koko haettavaa tekstimuotoista lausetta. Sen jälkeen sovelletaan kolmatta sääntöä kahden solmun ja yhden suhteen tilanteessa. Säännössä 3 symboli Node kuvautuu solmulle Persons (Name = Jalkanen), Edge kuvautuu suhteelle OWNS* ja M2 solmulle Outlets (Name = Jalkanen). Attribuutissa *return*(M1) muuttuja x1 tulee ensimmäisestä solmusta, x2 toisesta solmusta ja suhteesta tähän attribuuttiin tulevat muuttujat e1, y ja e2. Säännön 3 yhteydessä on huomionarvoista, että *query*(Edge) sisältää kaksi Cypher-suhdetta (OWNS*0..), joiden välissä on solmu y. Tämä tarvitaan, jotta transitiivisen kyselyn kaikki solmut saadaan palautettua ja ne generoituvat myöhemmin säännön 11 yhteydessä.

1. Q → M

Q = Persons (Name = Jalkanen) OWNS* Outlets (Name = Aamulehti)

query(Q) = `MATCH` ⊕ *query*(M) ⊕ `RETURN` ⊕ *return*(M)

```
query(Q) = MATCH (x1:Persons{Name:'Jalkanen'})
           -[e1:OWNS*0..]-> (y) -[e2:OWNS*0..]->
           (x2:Outlets{Name:'Aamulehti'})
           RETURN x1, e1, y, e2, x2
```

3. M1 → Node Edge M2

M1 = Persons (Name = Jalkanen) OWNS* Outlets (Name = Aamulehti)

query(M1) = *query*(Node) ⊕ *query*(Edge) ⊕ *query*(M2)

```
query(M1) = (x1:Persons{Name:'Jalkanen'})
            -[e1:OWNS*0..]-> (y) -[e2:OWNS*0..]->
            (x2:Outlets{Name:'Aamulehti'})
```

return(M1) = *return*(Node) ⊕ , ⊕ *return*(Edge) ⊕ , ⊕ *return*(M2)

return(M1) = `x1, e1, y, e2, x2`

Ensimmäisen solmun määrittelyyn sovelletaan viidettä ja kuudetta sääntöä, sillä henkilön määrittelevällä solmulla on annettuna ominaisuus. Tässä tilanteessa solmu määritellään täysin samoin kuin aiemmissa hakutapauksissa.

5. Node → Node_label Property

Node = Persons (Name = Jalkanen)

query(Node) = `( ⊕ x: ⊕ Node_label ⊕ { ⊕ query(Property) ⊕})`

*query(*Node) = `(x1:Persons{Name:'Jalkanen'})`

return(Node) = `x1`

6. Property → P_name = Value

Property = P_name = Jalkanen

*query(*Property) = `P_name ⊕ :' ⊕ Value ⊕ '`

query(Property) = `Name:'Jalkanen'`

Suhteeseen sovelletaan kerran sääntöä 11 sekä sääntöä 13 kahdesti. Säännössä 11 muodostettava attribuutti *query*(Edge) koostuu kahdesta suhteesta ja niiden välissä olevasta solmusta. Tämä yhdistelmä sitoo yhteen kaikki haettavat polut. Muuttuja y kuvaa solmua suhteiden välissä. Molemmissa suhteissa niiden pituudeksi on määritelty nolla tai enemmän. Molempiin *query*(Edge_name)-attribuutteihin sovelletaan sääntöä 13 erikseen ja näin molempiin tulee omat muuttujansa. Uudet muuttujat liitetään myös attribuuttiin *return*(Edge).

11. Edge → Edge_name*

Edge = OWNS*

query(Edge) = `-[ ⊕ query(Edge_name1) ⊕ *0.. ⊕ ]-> ⊕`
`(⊕ x ⊕) ⊕ -[ ⊕ query(Edge_name2) ⊕ *0.. ⊕ ]->`

query(Edge) = `-[e1:OWNS*0..]-> (y) -[e2:OWNS*0..]->`

return(Edge) = *return*(Edge_name1) `⊕ , ⊕ x ⊕ , ⊕` *return*(Edge_name2)

return(Edge) = `e1, y ,e2`

13. Edge_name ∈ edge_labels

Edge_name = OWNS

query(Edge_name) = `x ⊕ : ⊕ Edge_name`

query(Edge_name) = `e1:OWNS`

return(Edge_name) = `e1`

13. Edge_name ∈ edge_labels

Edge_name = OWNS

query(Edge_name) = `x ⊕ : ⊕ Edge_name`

query(Edge_name) = `e2:OWNS`

return(Edge_name) = `e2`

Viimeisen solmun jäsentämisessä sovelletaan toista, viidettä ja kuudetta sääntöä. Toinen sääntö oli jo aiemminkin mainittu sääntö, josta pääsee määrittelemään itse solmua. Viidettä ja kuudetta solmua sovelletaan solmulle samoin kuin aiemmin ja solmuun määritellään nimiketieto sekä ominaisuuden arvo.

2. M → Node

M = Outlets (Name = Aamulehti)

query(M) = *query*(Node) = `(x2:Outlets{Name:'Aamulehti'})`

return(M) = *return*(Node) = `x2`

5. Node → Node_label Property

Node = Persons (Name = Aamulehti)

query(Node) = `( ⊕ x: ⊕ Node_label ⊕ { ⊕ query(Property) ⊕})`

*query(*Node) = `(x2:Outlets{Name:'Aamulehti'})`

return(Node) = `x2`

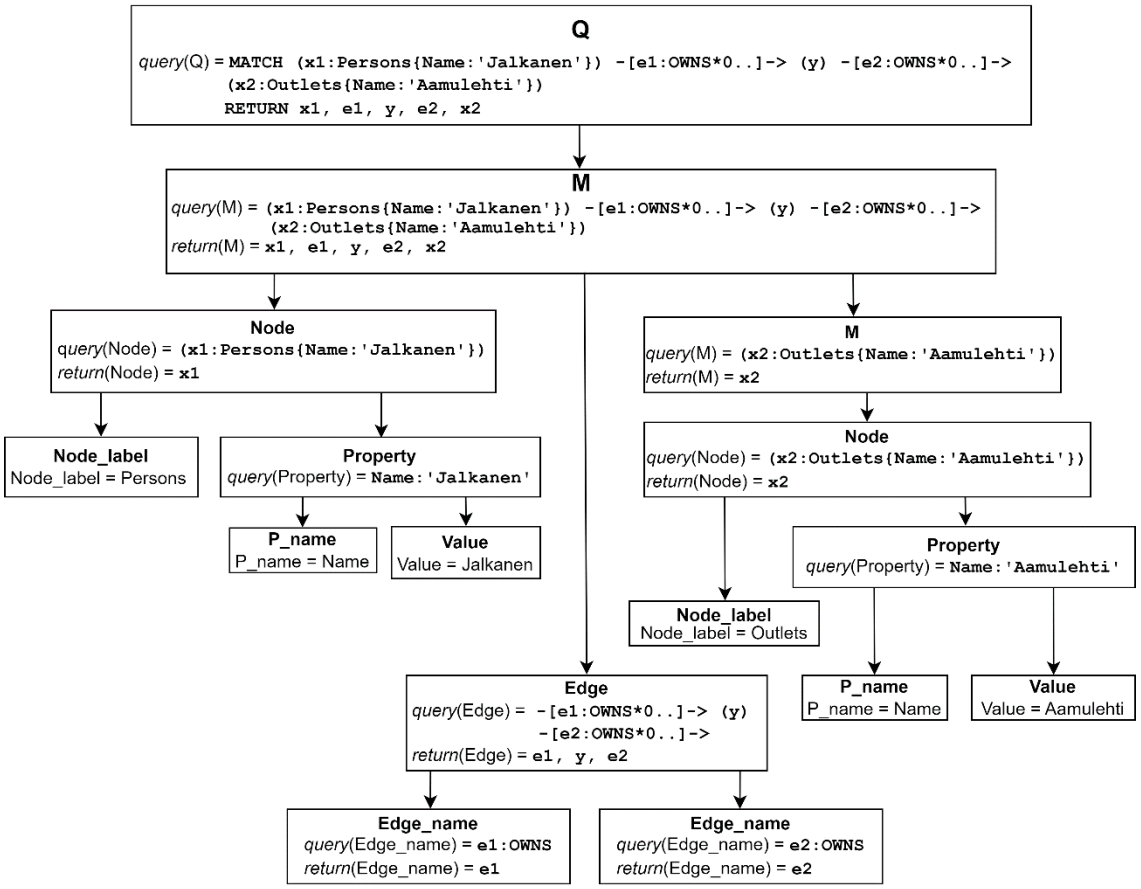
6. Property → P_name = Value

Property = P_name = Aamulehti

*query(*Property) = `P_name ⊕ :' ⊕ Value ⊕ '`

query(Property) = `Name:'Aamulehti'`

Kun säännöt on käyty läpi, muodostuu kokonaisuudesta valmis Cypher-kysely: `MATCH (x1:Persons{Name:'Jalkanen'}) -[e1:OWNS*0..]-> (y) -[e2:OWNS*0..]-> (x2:Outlets{Name:'Aamulehti'}) RETURN x1, e1, y, e2, x2`. Kuvassa 22 on esitettynä samojen sääntöjen läpikäynti jäsennyspuuna.



Kuva 22. Persons (Name = Jalkanen) OWNS* Outlets (Name = Aamulehti) haun jäsennyspuu.

9 Pohdinta

Tässä tutkielmassa tavoitteena oli luoda kielioppi, jolla voidaan kääntää tekstimuotoisia hakulauseita Cypher-kyselykielelle. Lisäksi tutkielmaa varten kehitettiin sovellus, joka hyödyntää muodostettua s-attribuoitua kielioppia lauseiden kääntämisessä ja tiedonhaussa. Sovellukseen kehitetty perushaku ei suoraan vastaa tähän tavoitteeseen luoda omaa kielioppia hakua varten, mutta se toteutettiin sovelluksen käytön helpottamiseksi ja käytön mahdollistamiseksi mahdollisimman monelle henkilölle.

Edistynyt haku sen sijaan vastaa tutkielman tavoitteeseen. Luotu kielioppi on selkeä kaava käyttäjän antamien tietojen muuttamiseksi Cypher-kyselylauseeksi. Koska tietokannan tiedot tässä tutkielmassa olivat yksinkertaiset ja erityyppisiä solmuja oli vain neljää erilaista, kielioppi pysyi selkeänä ja ytimekkäänä. Toisaalta kuitenkin kielioppia on hieman typistettykin ja muutama ominaisuus jätettiin toteutettavaksi tulevaisuudessa.

Tulevaisuuden kehitykseen jätetty ominaisuus on esimerkiksi ajateltu mahdollisuus hakea myös muilla solmun ominaisuuksilla kuin nimitiedolla. Nyt kaikille solmutyypeille on toteutettu mahdollisuus antaa haettavan solmun nimitieto, mutta solmujen ominaisuuksien ollessa paljon monipuolisempiakin hakua voisi laajentaa mahdollistamaan myös muiden ominaisuuksien perusteella hakemisen. Yksinkertaisin lisäys voisi olla mahdollisuus hakea uutismedian veronumeron perusteella. Toinen merkittävä laajentaminen hakuun voisi olla myös suurempi kuin (>)- ja pienempi kuin (<)-merkkien käyttö ominaisuuksien määrittelyssä. Muun muassa omistussuhteilla on tutkielman tietokannassa ominaisuus, joka kertoo omistuksen suuruuden prosenttiosuutena. Näin yksi hakumahdollisuus voisi olla haku niistä henkilöistä, jotka omistavat tiettyä uutismediaa yli tai alle määritellyn prosenttiosuuden. Suurempi kuin- ja pienempi kuin-merkkien käyttö soveltuu myös muihin numeeristen arvojen määrittelyyn, kuten esimerkiksi uutismedian henkilöstön määrän antamiseen.

Toinen tulevaisuuden kehityskohde on tuloskuvan laajentaminen solmua klikkaamalla. Sovelluksen toteutus mahdollistaa laajentamisen niin, että kun käyttäjä klikkaa solmua, tuloskuva muuttuu. Uudessa tuloskuvassa klikattu solmu olisi keskiössä ja sen suhteet vierussolmuihin olisivat näkyvissä. Toteutettu edistyksellinen hakutapa on sovellettavissa myös muihin kuin käytettyyn EurOMo-tietokantaan. Tämä vaatii nimike- ja ominaisuustietojen päivittämisen sovellukseen. Kehityksen kohteena voi olla myös näiden automaattinen haku tietokannasta, jolloin ohjelmapäivitystä ei tarvittaisi tietokannan vaihtuessa.

Luvussa 5.2 esitelty Bloom-visualisointityökalu tuo mielenkiintoisen vertailumahdollisuuden sen ja tutkielmassa kehitetyn hakumallin välille. Näiden tavoite on yhteinen: auttaa käyttäjää löytämään etsimäänsä tietoa ja esittää se mielekkäällä tavalla visuaalisesti. Bloom ei ole kuitenkaan tarkoitettu verkkosovellusten kehittämiseen, mikä oli tutkielman sovelluksen edellytys. Toisaalta Bloom kuitenkin sisältää mahdollisuuden tehdä

hakuja ilman Cypher-kyselykielen osaamista. Solmut ja suhteet määritellään Bloomissa graafisesti sen antamien ehdotusten perusteella tai luonnollisella kielellä kirjoittamalla. Kuitenkin käyttäjältä vaaditaan solmujen ja suhteiden ominaisuuksien tuntemusta, jotta niitä voi hyödyntää hauissa. Bloom näyttää vastauksena kyselyyn ensisijaisesti koko graafin vain tarkentaen haettuihin solmuihin tai suhteisiin, mikä ei kehitetyn sovelluksen näkökulmasta ole toimiva tapa. Lisäksi tutkielman sovelluksen ja luodun kieliopin vaatima transitiivisten suhteiden hakeminen tietokannasta ei onnistu Bloomilla.

Graafitietokantojen monipuolisuus ja erityisesti niiden antaman visuaalisen kuvan hyödyntäminen tiedon tulkinnassa ja hahmottamisessa tuovat laajat mahdollisuudet kehittää sovelluksia niiden pohjalta. Graafitietokantojen sovellusalueiden laaja-alaisisuus ja varsinkin sosiaalisen median sovellusten määrän voimakas lisääntyminen tuovat mahdollisuuden graafitietokantojen käytön lisääntymiselle myös tulevaisuudessa. Graafien visualisoinnilla ja hakutulosten analysoinnilla algoritmisten esikäsittelyiden avulla on lähes loputtomat mahdollisuudet tuoda uusia näkökulmia, mahdollisuuksia ja sovellusalueitakin.

Yleisesti graafitietokantojen tehokkuus ja suorituskyky eri tilanteissa on aihe, jota on tutkittu jo paljon, mutta vaatii edelleen lisätutkimusta. Luvussa 3.2 esitellyt graafitietokantojen hyödyt käsittävät juuri joidenkin tutkimusten mukaisesti tehokkaampaa tiedonhakua ja -käsittelyä. Kuitenkin, koska toisenlaisiakin tuloksia tutkimuksista graafitietokantojen tehokkuudesta on, lisätutkimus tästä on aiheellista. Tarkempaa tietoa tarvitaan liittyen muun muassa siihen, millaisella tiedolla ja kyselyrakenteilla graafitietokanta olisi tehokkaampi vaihtoehto.

Käytettyjen tietokantojen suosiosta ja käytön määrästä kertova listaus, jota DB-Engines ylläpitää, on kattava katsaus kulloisenkin hetken trendeihin tietokannoissa. Toisaalta suosioiden kehityksen näkee myös listauksista hyvin, sillä aiempien vuosien listauksia on mahdollista vertailla nykyiseen. Eri tietokantojen perusominaisuuksien vertailun mahdollisuus on tärkeä ominaisuus sivustolla ja tätä voikin hyödyntää, kun miettii tietokannan valintaa.

10 Johtopäätökset

Tutkielmassa toteutettiin sovellukseen kaksi erilaista visuaalista kyselytapaa. Perushaku on riippuvainen tietokannan sisältämästä tiedosta ja on siten sovellusriippuvainen. Se on suunnattu EurOMo-hankkeen median omistussuhteiden kartoittamiseen ja se on siksi nopea keino tutustua tietokantaan ja tehdä hakuja. Sovellus on tulevaisuudessa mahdollisesti pohjana EurOMo-hankkeen virallisilla verkkosivuilla julkaistavalle hakutoiminnolle, joten tavallisempaa ja nopeampaa tiedonhakumahdollisuuttakin tarvittiin.

Edistynyt haku taas on sovellusalueesta riippumaton, sillä siinä Cypher-kyselyiden luomisessa käytetään tutkielmassa luotua kielioppia. Se mahdollistaa *ad hoc* -kyselyt ja hakupolun muodostamisen pala palalta. Luotu s-attribuoitu kielioppi sopii hyvin tekstimuotoisten hakulauseiden kääntämiseen Cypher-kyselykielelle.

Tutkielmaa ja kehitettyä sovellusta varten tutustuttiin erilaisiin graafitietokantojen visualisoinnin työkaluihin. Neovis.js oli ainoa, joka täytti asetetut vaatimukset ja oli yhteensopiva verkkosovellukseen. Esimerkiksi Neo4j:n oma Browser täytti muut vaatimukset hyvin, mutta sen integroiminen verkkosovellukseen ei onnistu.

Kokonaisuutena kehitetty sovellus on eheä ja siihen valitut teknologiat toimivat hyvin yhdessä ja täydentävät toisiaan. Haasteena kehityksessä oli dokumentaation puute, joten tietokannan, sovelluksen sekä visualisointityökalun yhteen toimiminen on useiden erilaisten kokeilujen tulos. Sovellus vastaa EurOMo-hankkeen tarpeeseen saada visualisoitua tietoa myös käyttäjille. Haun tehokas käyttö edellyttää kuitenkin käyttäjältä ymmärrystä tietokannan sisältämästä tiedosta sekä ajatusta siitä, millaista tietoa tai yhteyttä lähtee haulla etsimään.

Tutkielman toteutus mahdollistaa jatkokehityksen niin kielioppiin kuin sovellukseenkin. Kielioppia laajentamalla voidaan luoda vankka perusta hakumallille, joka voitaisiin toteuttaa eri ohjelmointikielillä.

Viiteluettelo

Angles, Renzo, and Claudio Gutierrez. 2008. Survey of graph database models. *ACM Computing Surveys*, Vol. 40, No. 1, 1-39.

Angles, Renzo. 2012. A comparison of current graph database models. *IEEE 28th International Conference on Data Engineering Workshops*, 171-177.

Angles, Renzo, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2017. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, Vol. 50, No. 5, 1-40.

Barceló, Pablo, Leonid Libkin, and Juan L. Reutter. 2014. Querying regular graph patterns. *Journal of the ACM*, Vol. 61, No. 1, 1-54.

Baton, Jerome, and Rik van Bruggen. 2017. *Learning Neo4j 3.x -Second Edition: Run Blazingly Fast Queries on Complex Graph Datasets with the Power of the Neo4j Graph Database*. O'Reilly.

Calvanese, Diego, Moshe Vardi, Giuseppe de Giacomo, and Maurizio Lenzerini. 2003. Reasoning on regular path queries. *ACM SIGMOD Record*, Vol. 32, No. 4, 83-92.

DB-Engines. *https://db-engines.com/en/*. Luettu 30.1.2023.

Euromo, Euromedia Ownership Monitor (EurOMo). *https://media-ownership.eu/*. Luettu 15.2.2023.

Euromo2, Euromedia Ownership Monitor (EurOMo). *https://media-ownership.eu/about/methodology/*. Luettu 15.2.2023.

Foulds, L. R. 1995. *Graph Theory Applications*. Springer New York.

Holzschuher, Florian, and Rene Peinl. 2013. Performance of graph query languages: Comparison of cypher, gremlin and native access in Neo4j. *In Proceedings of the Joint EDBT/ICDT 2013 Workshops, Genova, Italy.*

Huang, Weidong, Jing Luo, Tomasz Bednarz, and Henry Duh. 2018. Making graph visualization a user-centered process. *Journal of Visual Languages and Computing,* Vol. 48, 1-8.

Karol, Sven. 2006. *An introduction to attribute grammars*. Department of Computer Science, Technische Universität Dresden.

Khan, Wisal, Waqas Ahmad, Bin Luo, and Ejaz Ahmed. 2019. SQL database with physical database tuning technique and NoSQL graph database comparisons. *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC 2019)*.

Knuth, Donald. 1968. Semantics of context-free languages. *Mathematical Systems Theory*, Vol 2, No 2, 127-145.

Koivisto, Pertti ja Riitta Niemistö. 2018. *Graafiteoriaa*. Tampereen yliopisto. Informaatiotieteiden yksikkö.

Kotiranta, Petri, Marko Junkkari, and Jyrki Nummenmaa. 2022. Performance of graph and relational databases in complex queries. *Applied Sciences*, Vol. 12, 6490.

von Landesberger, Tatiana, Arjan Kuijper, Tobias Schreck, Joern Kohlhammer, J.J van Wijk, Jean-Daniel Fekete and D. Fellner. 2011. Visual analysis of large graphs: state-of-the-art and future research challenges. *Computer Graphics Forum*. Vol. 30, No. 6, 1719-1749.

Libkin, Leonid, Wim Martens, and Domagoj Vrgoč. 2016. Querying graph with data. *Journal of the ACM*, Vol. 63, No. 2, 1-53.

Ma, Kwan-Liu, and Chris Muelder. 2013. Large-scale graph visualization and analytics. *Computer*, Vol. 46, No. 7, 39-46.

Neo4j. *https://neo4j.com/*http://www.neo4j.com/. Luettu 23.1.2023.

Neovis. *https://neo4j.com/developer-blog/graph-visualization-with-neo4j-using-neovis-js/*. Luettu 28.3.2023.

React. *https://reactjs.org/docs/faq-structure.html*. Luettu 24.2.2023.

Robinson, Ian, Eifrem Emil, and Webber James. 2015. *Graph Databases: New Opportunities for Connected Data*. O'Reilly.

de Virgilio, Roberto, Antonio Maccioni, and Riccardo Torlone. 2013. Converting relational to graph databases. *First International Workshop on Graph Data Management Experiences and Systems*, 1-6.

Visualization. *https://neo4j.com/developer-blog/15-tools-for-visualizing-your-neo4j-graph-database/.* Luettu 28.3.2023.