Tampereen yliopisto

Sauli Nevalainen

RELAATIOTIETOKANTAKIRJASTOT NODE.JS-AJOYMPÄRISTÖSSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2021

TIIVISTELMÄ

Sauli Nevalainen: Relaatiotietokantakirjastot Node.js-ajoympäristössä
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2021

Tutkielman tavoite on perehtyä Node.js-ajoympäristössä käytettäviin relaatiotietokantakirjastoihin, selvittää kirjaston valinnasta aiheutuvat seuraukset ja siten auttaa valintaa. Tutkielma on pääasiassa kirjallisuuskatsaus, mutta sisältää myös empiirisen osuuden, jossa selvitettiin relaatiotietokantakirjastojen tehokkuutta.

Relaatiotietokantakirjastolla tarkoitetaan kirjastoa, jonka avulla Node.js-ajoympäristössä relaatiotietokantaa käsitellään. Tämä tutkielma keskittyi kolmeen kirjastoon, jotka rakentuvat toinen toisensa päälle: matalimmalla tasolla node-postgres, sitten Knex.js ja korkeimmalla tasolla Bookshelf.js.

Suorituskykykokeen tulokset vastasivat odotetusti aiempia muissa ympäristöissä toteutettuja kokeita: kyselyiden suoritusaika kasvoi siirryttäessä korkeammille abstraktiotasoille. Suorituskykymenetyksen vastineeksi korkeamman tason kirjastot tarjoavat lisäominaisuuksia ja työkaluja kehittäjille. Nämä ominaisuudet ja työkalut voivat oikein käytettynä helpottaa ohjelmistokoodin ylläpidettävyyttä ja nopeuttaa kehittämistä.

Avainsanat: Node.js, SQL, ORM, relaatiotietokanta

Sisällysluettelo

1 Johdanto

Web-ohjelmoinnin suosio on tänä päivänä valtava ja edelleen kasvava. Yhdysvaltojen Bureau of Labor Statistics arvioi, että web-kehittäjänä työskentelevien määrä tulee kasvamaan 8 % vuosina 2019–2029, joka on keskimääräistä työllisyyden kasvua huomattavasti suurempi (BLS, 2021). Myös Stack Overflow, joka on suosittu tietotekniisiin aiheisiin erikoistunut keskustelufoorumi, on teettänyt kyselytutkimuksen, joka osoittaa web-ohjelmoinnin suosiota: web-ohjelmointikielenä pidetty JavaScript on tutkimuksen mukaan suosituin ohjelmointikieli jo kahdeksatta vuotta. Kyselyyn vastasi lähes 65000 ohjelmistokehittäjää. (SO, 2021)

Yksi JavaScriptiä hyödyntävä teknologia on Node.js (Node.js, 2021). Se on tekijöidensä mukaan asynkroninen JavaScript-ajoympäristö, joka on suunniteltu skaalautuvien verkkosovellusten kehitykseen. (Node.js, 2021) Node.js on saavuttanut suurta suosiota olemalla jo kaksi vuotta peräkkäin käytetyin teknologia ohjelmistokehityksessä (SO, 2021).

Olennainen osa verkkosovelluksia ovat myös tietokannat. Tietokannat mahdollistavat kaikenlaisen datan tallentamisen, mikä puolestaan mahdollistaa muun muassa dynaamisten web-sovellusten luomisen. Suosituimmat neljä tietokantaa Stack Overflown kyselytutkimuksen mukaan ovat relaatiotietokantoja (SO, 2021).

Tämä tutkielma käsittelee kolmea Node.js:lle tehtyä relaatiotietokantakirjastoa: node-postgres (PG, 2021), Knex.js (Knex, 2021) ja Bookshelf.js (Bookshelf, 2021). Kirjastot on edellä lueteltu abstraktiotasolta matalimmasta korkeimpaan ja ne rakentuvat toistensa päälle. Knex.js ja Bookshelf tarjoavat rajapinnan useampaan eri relaatiotietokantaan, mutta node-postgres on vain PostgreSQL-tietokannalle (PostgreSQL, 2021) kehitetty ajuri. PostgreSQL-ajuri valittiin, sillä nimenomaisesta tietokannasta kirjoittajalla on eniten kokemusta. Tiedot ovat kuitenkin oletettavasti suurimmilta osin yleistettävissä muihinkin relaatiotietokanta-ajureihin. Knex.js valikoitui, sillä se on ainoa kirjoittajan tietämä Node.js:lle tehty kyselyrakentaja; sen avulla voidaan rakentaa tietokantakyselyitä dynaamisesti. Bookshelf.js on olio-relaatiomallintaja (ORM), joka luo uuden abstraktiotason Knex.js:n päälle. Kolmen kirjaston yhteys toisiinsa vaikutti mielenkiintoiselta, joten Bookshelf.js valikoitui mukaan tähän tutkimukseen. Eniten viikoittaisia latauksia kirjoitushetkellä kerännyt olio-relaatiomallintaja, Sequelize, jäi tutkielman ulkopuolelle, sillä Sequelize ei rakennu Knex.js:n päälle (OpenBase, 2021; Sequelize, 2021).

Tämän tutkielman on tarkoitus selvittää näiden kolmen kirjaston eroavaisuuksia ja antaa tietoa valinnan tueksi. Vertailua tehdään tehokkuuden, käytettävyyden ja turvallisuuden näkökulmasta.

Tutkielman alussa esitellään tutkimusmenetelmät, jonka jälkeen siirrytään teknologioiden esittelyihin. Ensin esitellään relaatiotietokannat yleisesti ja selitetään niiden tarkoitus ja toiminta. Seuraavaksi esitellään Node.js ja JavaScript sekä näiden olennaisimmat piirteet. Pohjustuksen jälkeen siirrytään esittelemään valitut tietokantakirjastot lyhyesti. Näistä esitellään niiden erityispiirteet. Esittelyiden jälkeen siirrytään vertailuun. Rajapintoja vertaillaan selvittämällä niiden suorituskykyä, käytettävyyttä ja turvallisuutta. Pienenä kuriositeettina esitellään suorituskyvyn käsittelyn yhteydessä myös pieni empiirinen suorituskykykoe. Lopuksi vertailun tulokset ja johtopäätökset esitetään kootusti yhteenvedossa.

2 Tutkimusmenetelmä

Tämä kandidaattitutkielma on pääasiallisesti kirjallisuuskatsaus, vaikka sisältääkin lyhyen empiirisen osan. Tutkielman rajauksessa käytettiin apuna Node.js:n paketinhallintajärjestelmän, npm:n, verkkosivustoa, josta haettiin tietoa ”ORM”-hakusanalla löytyvien pakettien suosiosta.

Kirjallisuutta yritettiin hakea useista tietokannoista, mutta hyvin nopeasti ilmeni useiden tietokantojen hyödyttömyys. Hyödyllisiltä hakukoneilta vaikuttivat ACM Digital Library (ACM DL, 2021), Google Scholar (GS, 2021), Elsevier (Elsevier, 2021), Scopus (Scopus, 2021) ja internetlähteiden etsinnässä Google (Google, 2021). Hakutuloksista yritettiin poimia mahdollisimman uusia julkaisuja: lähteiden tulisi ehdottomasti olla vähintään 2010-luvulta. Tuoreita lähteitä pidettiin tärkeinä teknologioiden nopean kehityksen vuoksi; tieto vanhentuu nopeasti ja Node.js on ympäristönä uusi. Muutamien lähteiden osalta 2010-luvun vaatimuksesta kuitenkin poikettiin, sillä relaatiotietokannat ovat jo lähes 50 vuotta vanhaa teknologiaa.

Tietoa yritettiin aluksi hakea vain Node.js-ympäristöön liittyen. Tutkimuksia ei kuitenkaan juuri ole hakutulosten olemattomuuden perusteella tehty, joten hakuja päätettiin alkaa tekemään myös yleisemmin tietokantoihin liittyen. Haut tehtiin englanniksi, sillä suurin osa tutkimuksista on englanninkielisiä. Lisäksi englanninkielinen termistö on vakiintuneempi kuin esimerkiksi suomenkielinen.

Olio-relaatiomalli on kuitenkin jo yli 20 vuotta vanha konsepti, joten tutkimuksen aineistoksi päätettiin hyväksyä yleisesti ORM:iin liittyviä artikkeleita. Tehokkuuden vertailusta löytyi muun muassa PHP-ympäristössä (PHP, 2021) tehty empiirinen tutkimus, joka otettiin mukaan, sillä ORM:ien ongelmat ovat kyseisen tutkimuksenkin mukaan tyypillisesti yleismaallisia (Procaccianti, Lago, & Diesveld, 2016).

Pakettien turvallisuuden vertailuun käytettiin Snyk-nimisen tietoturvayrityksen tarjoamaa Advisor-hakukonetta (Snyk, 2021), josta löytyy muun muassa npm-pakettien versiohistoria tietoturvahaavoittuvuuksineen.

Empiirisenä osuutena tutkielmassa tehtiin suorituskykymittauksia valituilla relaatiotietokantakirjastoilla. Empiirinen koe tehtiin Procacciantin ja muiden (2016) tekemää tutkimusta mukaillen, jotta saavutettaisiin jonkinlainen vertailukelpoisuus toiselle ohjelmointikielelle tehtyyn tutkimukseen. Vertailukelpoisuuden vuoksi myös eroavaisuudet ja perustelut niihin on pyritty kuvaamaan.

3 Käsitteitä

Tässä kappaleessa esitellään tutkielman kannalta olennaisia asioita: relaatiotietokannat, JavaScript, Node.js ja olio-relaatiomallinnus

3.1 Relaatiotietokannat

Tässä kappaleessa käsitellään ensiksi relaatiotietokantojen kehitystä historiasta nykyhetkeen, jonka jälkeen esitellään relaatiotietokantojen toimintaa ja SQL-kyselykieli.

3.1.1 Historia

Relaatiotietokantojen historian voidaan nähdä alkaneen 1970, kun E.F. Codd julkaisi relaatiomallin (Codd, 1970). Codd oli matemaatikko ja relaatiomallikin pohjautui matemaattisiin relaatioihin. Codd määritteli myös normaalimuodot, joiden tarkoituksena oli auttaa vähentämään redundanttia dataa. Relaatiotietokannan oli tarkoitus piilottaa käyttäjältä tietokannan todellinen toteutus, kun aiemmat relaatiotietokantoja edeltäneet tietokannat vaativat ohjelmoijilta tarkkaa tietämystä tietokannan rakenteesta ja toteutuksesta. (Wade & Chamberlin, 2012)

3.1.2 Toiminta

Relaatiotietokannat pohjautuvat relaatiomalliin. Tietokantoihin tallennettu tieto on säilöttynä *relaatioihin*, jotka tunnetaan myös tauluina. Taulut koostuvat sarakkeista ja riveistä. Sarakkeet ovat nimettyjä, ja jokaisen taulun sarakkeen nimen tulee olla uniikki. Saman nimisiä sarakkeita voi olla kuitenkin muissa tauluissa, jolloin ne voidaan yksilöidä lisäämällä sarakkeen nimen eteen taulun nimi. (Harrington, 2016, ss. 89–91)

Vastaavasti myös jokaisen taulun sisältämän rivin tulee olla uniikki, ja täten yksilöitävissä. Rivit voivat sisältää osittain samaa dataa keskenään, mutta yhden sarakkeen tai sarakkeiden yhdistelmän tulee muodostaa rivin yksilöivä *pääavain*. (Harrington, 2016, s. 91)

Tietokannan loogisen rakenteen suunnitelmaa kutsutaan skeemaksi (schema). Skeema on yksinkertaistettuna säiliö tietokannan tauluille. Niiden avulla voidaan pitää samalla palvelimella useita eri tietokantoja toisistaan eroteltuina loogisina kokonaisuuksina. (Harrington, 2016, s. 194)

3.1.3 Structured Query Language

Lähes jokaisen relaatiotietokantajärjestelmän ohjaamiseen käytetään *Structured Query Languagea* (SQL). SQL:stä on julkaistu useita standardeja vuodesta 1986 lähtien. Mikään ei kuitenkaan velvoita tietokantajärjestelmien toteuttajia noudattamaan standardeja, joten eri ohjelmistojen käyttämissä SQL-kielissä on eroja. (Harrington, 2016, ss. 183–186)

3.2 JavaScript

Nimestään huolimatta JavaScriptillä on hyvin vähän tekemistä Javan kanssa. Lukuun ottamatta osittaista syntaktista yhtäläisyyttä, ne ovat täysin eri kieliä. JavaScriptin standardoitu nimitys on oikeasti ECMAScript. (Flanagan, 2020, Luku 1) Vakiintuneen käytännön vuoksi tutkielmassa käytetään kuitenkin nimitystä JavaScript.

JavaScript on pääasiallisesti web-ohjelmointikieli. JavaScript kehitettiin alun perin dynaamisten web-sivustojen kehittämiseksi, mutta seuraavassa kappaleessa käsiteltävä Node.js toi JavaScriptin yleisesti käyttöön myös palvelinohjelmoinnissa. JavaScript on tulkattava korkean tason ohjelmointikieli, joka sisältää sekä olio- että funktionaalisista ohjelmointikielistä tuttuja ominaisuuksia. (Flanagan, 2020, Luku 1)

Yksi JavaScriptin olennaisimmista ominaisuuksista, myös tämän tutkielman kannalta, ovat oliot (object). Olio sisältää useita avain-arvo-pareja, joissa avaimena toimii merkkijono. Lukuun ottamatta merkkijonoja (string), numeroita (number), symboleita (symbol), totuusarvoja *true* ja *false*, sekä erityisiä tyyppejä *null* ja *undefined,* kaikki muuttujat ovat JavaScriptissä olioita. (Flanagan, 2020, Luku 6.1)

Asynkronisuus on JavaScriptille tyypillistä. Asynkronisuus tarkoittaa, että ohjelman ei tarvitse jäädä odottamaan toisen aliohjelman suoritusta; koodia ei toisin sanoen suoriteta ”rivi riviltä”. Asynkronisuuden käsittelyyn JavaScript tarjoaa useita tapoja, joista vanhin on takaisinkutsu- eli *callback*-funktiot. Takaisinkutsufunktiot ovat funktioita, jotka annetaan parametrina asynkroniselle funktiolle ja ne tulevat suoritetuksi, kun asynkroninen operaatio on valmis. ECMAScript 6 eli ES6-standardi toi kieleen *promise*-objektit ja niiden tarkoitus oli yksinkertaistaa asynkronista ohjelmointia. Promiset yksinkertaistavat ohjelmakoodin rakennetta ja helpottavat virheenkäsittelyä takaisinkutsufunktioihin verrattuna. ES2017-standardi esitteli *async*- ja *await*-avainsanat, jotka yksinkertaistavat promisejen käyttöä entisestään. Uusien avainsanojen avulla asynkroninen koodi voidaan kirjoittaa aivan kuten synkroninenkin. (Flanagan, 2020, Luku 13)

3.3 Node.js

Node.js on vuonna 2009 esitelty yhdessä säikeessä suoritettava avoimen lähdekoodin JavaScript-ajoympäristö (Node.dev, 2021). Se kykenee liittymiensä kautta keskustelemaan käyttöjärjestelmän kanssa tarjoten vaihtoehdon myös perinteisille shell-skripteille (Flanagan, 2020, Luku 16). Alkujaan Node.js on kuitenkin erityisesti websovelluksia varten kehitetty ajoympäristö. (Node.js, 2021).

Tärkeä osa Node.js-ympäristöä on sen mukana saatava *npm*-pakettimanageri. Pakettimanagerin avulla on mahdollista ottaa ohjelmassaan käyttöön muiden ohjelmoimia *moduuleja* eli *paketteja* (Flanagan, 2020, Luku 16.1). Npm:ssä oli tammikuussa 2017 tarjolla yli 350000 erilaista pakettia; nykyään osa paketeista on kuitenkin tarkoitettu selainsovelluksille Node.js:n sijaan (Node.dev, 2021). Tutkielmassa käsiteltävät relaatiotietokantakirjastot ovat npm:ssä saatavilla olevia paketteja.

Npm pitää huolta myös kehitettävän ohjelman *riippuvuuksista* (dependencies). Kun lataa ja asentaa paketin npm:n avulla, se lisätään oletuksena automaattisesti ohjelman riippuvuudeksi. Tämä mahdollistaa ohjelman yksinkertaisen käyttöönoton myös muissa ympäristöissä, sillä tarvittavat riippuvuudet voidaan asentaa yhdellä komennolla. (Node.dev, 2021)

3.4 Olio-relaatiomallinnus

Relaatiotietokantojen ja olio-ohjelmointikielten eriävien paradigmojen vuoksi niiden yhteensovittaminen on hankalaa. Ongelmia, jotka ilmenevät näitä kahta yhteensovittaessa kutsutaan englanniksi termillä *object-relational impedance mismatch problem* (IMP). (Torres, Galante, Pimenta, & Martins, 2017). Olio-relaatiomallintajat (ORM) on suunniteltu ratkomaan näitä ongelmia luomalla abstraktiotason olio-ohjelmointikielen ja tietokannan datan välille. Kun oliokielellä määritellyn olion tilaa muutetaan, päivittyy tieto samalla myös tietokantaan. Käytännössä ORM siis muuntaa oliolle tapahtuvat operaatiot edelleen SQL-kyselyiksi. (Chen ja muut, 2016)

4 Relaatiotietokantakirjastot

Tässä kappaleessa esitellään lyhyesti vertailtavaksi valitut relaatiotietokantakirjastot. Kukin kirjasto esitellään pintapuolisesti erityisine ominaisuuksineen. Jokaisesta esitetään seuraavaa SQL-kielistä kyselyä vastaava komento yksinkertaisena esimerkkinä syntaksista.

```
1. SELECT nimi, julkaisuvuosi, ohjaaja FROM elokuvat
```

4.1 Node-postgres

Node-postgres on matalan tason tietokantakirjasto PostgreSQL-tietokannalle. Node-postgres on kirjoitettu JavaScriptillä, mutta tarjoaa myös mahdollisuuden käyttää C-kielellä kirjoitettua *libpq*-kirjastoa natiivin tietokantayhteyden muodostamiseksi. (PG, 2021, osa Welcome, 2021, osa Native Bindings)

Node-postgres mahdollistaa tietokantakyselyiden suorittamisen, ja vastaus on mahdollista käsitellä takaisinkutsufunktioiden tai promise-objektien avulla. Tietokantakyselyn vastaus palautetaan objekteja sisältävänä taulukkona (array of objects). Tietokantakyselyt kirjoitetaan SQL:nä. (PG, 2021, osa Queries)

```
1. client.query(`SELECT nimi, julkaisuvuosi, ohjaaja
2. FROM elokuvat`)
```

4.2 Knex.js

Knex.js käyttää taustallaan node-postgres-pakettia, joten se on node-postgresistä seuraava abstraktiotaso. PostgreSQL-ajurin lisäksi Knex.js tarjoaa mahdollisuuden käyttää tietokantana muun muassa MySQL- ja SQLite-tietokantoja. Knex.js on niin kutsuttu kyselyrakentaja: sen avulla voidaan muodostaa SQL-kyselyjä käyttäen JavaScript-tyylistä syntaksia. Knexin metodien nimet ovat tuttuja SQL:stä. (Knex, 2021). Aiemmin esitelty SQL-kysely olisi Knexillä seuraava:

```
1. knex.select('nimi', 'julkaisuvuosi', 'ohjaaja')
2. .from('elokuvat')
```

Kyselyrakentajan lisäksi Knex sisältää myös skeemarakentajan. Skeemarakentaja mahdollistaa tietokannan rakenteen luomisen ja päivittämisen Knexin avulla aiemman kaltaista syntaksia käyttäen. Jokainen muutos voidaan tallentaa erilliseen tiedostoon niin kutsuttuun *migraatioon*, joka sisältää tiedon muutoksesta ja kuinka muutos peruutetaan. Tietokannan rakenne voidaan täten palauttaa aikaisempaan tilaan yksinkertaisesti. (Knex, 2021)

Knexin metodit palauttavat oletuksena promiseja, mutta erillisen metodin avulla sitä voi käyttää myös callback-funktioiden kanssa. Virallinen dokumentaatio ohjeistaa käyttämään promiseja. (Knex, 2021)

4.3 Bookshelf.js

Bookshelf.js on Knexin päälle rakennettu olio-relaatiomallintaja eli se on Knexistä seuraava abstraktiotaso. Bookshelfin luvataan toimivan PostgreSQL:n, MySQL:n ja SQLite3:n kanssa eli se ei tarjoa yhtä laajaa järjestelmätukea kuin sen käyttämä Knex. Olio-relaatiomallintajana se eroaa merkittävästi aiemmin mainituista kirjastoista, mutta se myös mahdollistaa taustalla olevan Knexin suoran käytön tarvittaessa. (Bookshelf, 2021)

Bookshelfin kanssa ei lähtökohtaisesti käytetä enää SQL:ää vaan puhdasta JavaScriptiä. Ennen kyselyjen muodostamista Bookshelfin avulla tulee muodostaa malleja, joissa tietokannan taulut yhdistetään JavaScript objektiin. (Bookshelf, 2021)

Aiemman kaltaisen esimerkin luomiseen Bookshelfissa tulee ensin määrittää elokuvalle malli seuraavalla tavalla

```
1. const Elokuva = bookshelf.model('Elokuva', {
2. tableName: 'elokuvat'
3. })
```

Mallin määrittelyn jälkeen voidaan tulos hakea seuraavasti.

```
1. Elokuva.fetch({columns: ['nimi', 'julkaisuvuosi',
2. 'ohjaaja']})
```

5 Vertailu

Tässä kappaleessa vertaillaan aiemmin esiteltyjä relaatiotietokantakirjastoja kolmella mittarilla: suorituskyky, käytettävyys, ja turvallisuus. Suorituskykyä vertaillaan energiatehokkuuden ja operaatioiden nopeuden avulla. Käytettävyydestä kerrotaan, mitä työkaluja ja apuvälineitä mikäkin kirjasto tarjoaa ohjelmoijalle niin ohjelmoinnin kuin ylläpidettävyydenkin näkökulmasta. Käytettävyyden vertailu tehdään pintapuolisesti esittelemällä valittujen tietokantakirjastojen olennaisimmiksi koettuja ominaisuuksia. Turvallisuus-kappaleessa tarkastellaan yleisten relaatiotietokantojen tietoturvaongelmien estämistä kussakin kirjastossa.

5.1 Suorituskyky

Tässä kappaleessa käsitellään suorituskykyä ensin aikaisempien tutkimusten pohjalta, jonka jälkeen esitellään tutkielmaa varten toteutettu suorituskykykoe ja sen tulokset

5.1.1 Aiempi tutkimus

Procaccianti ja muut (2016) ovat tutkineet olio-relaatiomallintajien energiatehokkuutta PHP-ympäristössä. Energiatehokkuus on luonnollisesti suoraan yhteydessä suorituskykyyn: tehokkaammat operaatiot vaativat vähemmän suoritusaikaa, joka tarkoittaa pienempää energiakulutusta. Procaccianti ja muut (2016) tutkivat tehokkuutta kolmen muuttujan näkökulmasta: valitun operaation, valitun tietokantakirjaston ja tietokantataulun koon.

Aiempaan tutkimukseen mukaan valitut kehykset vastaavat melko hyvin kuvauksiltaan tämän tutkimuksen Node.js-ympäristön tietokantakirjastoja, joten tulokset lienevät verrannollisia. Aiemman tutkimuksen mukaan olio-relaatiomallintajan avulla suoritetut operaatiot ovat merkittävästi hitaampia kuin kyselyrakentajan tai tietokanta-ajurin avulla suoritetut. Eroa tietokanta-ajurilla suoritettuun kyselyyn oli jopa 70 %. (Procaccianti ja muut, 2016)

Samaan tulokseen ORM:ien suorituskyvystä ovat tulleet myös Colley, Stanier ja Asaduzzaman (2018). Colley ja muut (2018) analysoivat suorituskykyä tarkemmin: he selvittivät minkälaisen suoritussuunnitelman (execution plan) kyselyrakentaja tuottaa ja vertasivat sitä hyväksi todettuihin tapoihin. He havaitsivat ORM:n käyttävän enemmän muistia kuin vertailukohtana oleva kysely ja toteavat sen vaikuttavan potentiaalisesti koko järjestelmän suorituskykyyn yhden kyselyn sijaan.

5.1.2 Suorituskykykoe

Tässä aliluvussa esitellään kokeen tutkimuskysymys, menetelmät ja laitteisto. Tutkimuksen tulosten käsittely löytyy omasta aliluvustaan.

5.1.2.1 Kokeen määrittely

Kokeen tarkoituksena on tutkia eri relaatiotietokantakirjastojen tehokkuutta ohjelmistokehittäjän näkökulmasta. Kyseinen näkökulma on valittu, jotta saadaan kehittäjille tietoa heidän tekemiensä valintojen vaikutuksista ohjelmiston tehokkuuteen. Tutkimuskysymys on siis *”Mikä on eri relaatiotietokantakirjastojen suorituskyky?”*.

Suorituskyvyn mittariksi valittiin *suoritusaika*, kuten myös Procaccianti ja muut (2016) ovat valinneet. Heillä oli mittarina myös energiankulutus, mutta koska tässä tutkimuksessa tutkitaan vain tehokkuutta eikä energiatehokkuutta, ei se ole tässä kokeessa olennainen.

5.1.2.2 Muuttujien valinta

Suoritusaika toimii kokeessa riippuvana muuttujana ja selittäväksi muuttujaksi relaatiotietokantakirjasto. Procaccianti ja muut (2016) valitsivat tutkimuksessaan selittäväksi muuttujaksi lisäksi taulun koon ja kyselyn tyypin. He eivät kuitenkaan huomanneet taulun koolla olevan oleellista merkitystä, joten se on jätetty tästä kokeestakin pois. Kyselyn tyypin vertailua ei myöskään nähty olennaisena, kun tarkastellaan kirjastojen kokonaistehokkuutta, joten se on jätetty pois tutkimuksen yksinkertaistamiseksi.

5.1.2.3 Kokeen malli

Jokaiselle relaatiotietokantakirjastolle toteutettiin neljän tyyppisiä kyselyitä: luonti (create), luku (read), päivitys (update) ja poisto (delete), joten lopputuloksena oli 12 mitattavaa yhdistelmää. Kunkin kirjaston kaikkien eri kyselyiden tulokset käsiteltiin kuitenkin yhdessä, jotta voitiin vertailla kirjastojen kokonaistehokkuutta keskenään.

5.1.2.4 Näytteen valinta

Yrityksistä huolimatta Procaccianti ja muut (2016) käyttämää tietokantaa ei ollut löydettävissä, ja palvelu, josta tietokanta oli, on myös jo kadonnut internetistä. Koska Procaccianti ja muut (2016) totesivat, että taulun koolla ei ole tehokkuuden kannalta merkitystä, päätettiin näytteenä toimiva tietokantataulu generoida Filldb-palvelun avulla (FillDB, 2021). Palvelu tuottaa MySQL-yhteensopivia lausekkeita, jotka muokattiin käsin PostgreSQL-sopiviksi. Lopullinen tietokantataulu sisälsi kuusi saraketta ja 10000 riviä.

5.1.2.5 Tutkimusvälineet

Koe toteutettiin huhtikuussa 2021 ja käytössä oli silloin viimeisin PostgreSQL-tietokanta versioltaan 13.6. Kaikki testit ajettiin tutkimuksen tekijän kotitietokoneella, jossa on kuusiydinprosessori ja 16 GB keskusmuistia. Käyttöjärjestelmänä oli 64-bittinen Windows 10 ja PostgreSQL-palvelinta ajettiin Docker-säiliössä. Suoritusajat mitattiin lisäämällä ajan mittaaminen Node.js:n koodiin kyselyiden ympärille, jolloin esimerkiksi kirjastojen

lataamiseen tarvittava aika ei vaikuta lopputulokseen. Kokeessa ajetut suorituskykytestit ja tietokantataulun luontilauseet ovat julkisesti jaossa GitHubissa (Nevalainen, 2021).

5.1.2.6 Kokeen suoritus

Koska kokeen suoritukseen oli käytössä vain kotitietokone, on mahdollista, että muut ohjelmat vaikuttavat suorituskykyyn. Mahdollisimman moni ylimääräinen ohjelma pyrittiin kuitenkin sulkemaan häiriöiden välttämiseksi. Yhdessä testisuorituksessa ajettiin 10000 kyselyä, paitsi lukuoperaatioissa 30000 kyselyä, ja suorituksen aika mitattiin nanosekunteina. Jokaisen testisuorituksen jälkeen tietokantataulu palautettiin alkuperäiseen tilaan. Jokainen testisuoritus ajettiin viisi kertaa.

5.1.2.7 Hypoteesin muodostaminen

Hypoteeseja muodostetaan vain yksi. Hypoteesia testataan merkitsevyystasolla $\alpha = 0{,}05$, joten tuloksille suoritettiin varianssianalyysi.

- **Nollahypoteesi** H_0: Suoritusajassa ei ole merkittävää eroa kirjastojen välillä.

$$\mu_{t,\text{node-postgres}} = \mu_{t,Bookshelf.js} = \mu_{t,Knex.js}$$

- **Vaihtoehtoinen hypoteesi** H_a: Suorituskyvyssä on merkitsevä ero ainakin kahden relaatiotietokantakirjaston välillä.

$$\exists\, i, j \in \{\text{node-postgres}, \text{Bookshelf.js}, \text{Knex.js}\} \mid i \neq j \wedge \mu_{t,i} \neq \mu_{t,j}$$

5.1.3 Suorituskykykokeen tulokset

Tässä kappaleessa esitellään suorituskykykokeen tulokset. Aluksi esitellään dataa kuvailevan tilastoanalyysin avulla. Sen jälkeen testataan alkuperäistä hypoteesia. Tietojen tulkinnan avuksi on lisätty graafeja.

5.1.3.1 Kuvailevat tunnusluvut

Koedatan tunnusluvut on esitetty Taulukossa 1. Valitut tunnusluvut ovat keskiarvo, keskihajonta ja varianssi.

Koedatalle tehtiin Shapiro-Wilk -testi normaalijakautuneisuuden testaamiseksi. Testin nollahypoteesia ei voitu hylätä (*W(60)=0,973, p=0,215*), joten mittausdatan voidaan olettaa noudattavan normaalijakaumaa.

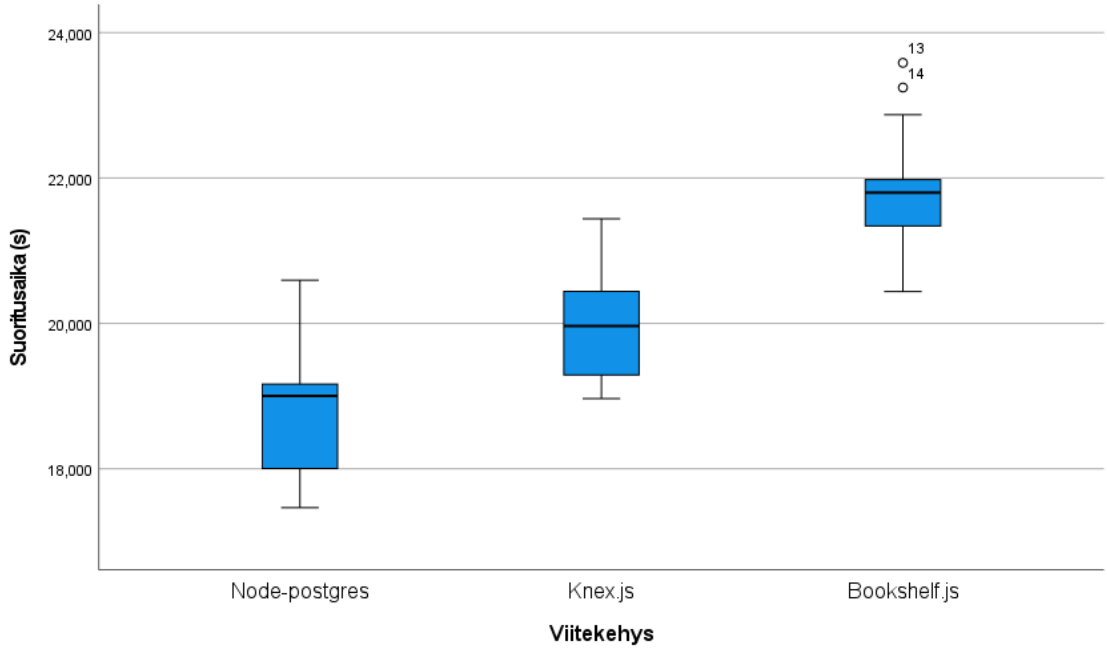
	N	Keskiarvo	Keskihajonta	Varianssi
Suoritusaika (s)	60	20,17378	1,472721	2,169

Taulukko 1. Koedatan tunnusluvut.

5.1.3.2 Hypoteesin tulos

Hypoteesin avulla voidaan arvioida valittujen kirjastojen tehokkuutta suoritusajassa mitattuna. Kuvassa 1 on kuvattu suoritusajat viitekehityksen mukaan ryhmiteltynä laatikkojanakuviossa. Varianssianalyysin perusteella kirjastojen välillä on merkitsevä ero (*F(2)=76,884, p<0,0005*), joten nollahypoteesi hylätään.

Suoritusaikojen keskiarvot ryhmiteltynä viitekehyksen mukaan olivat node-postgres 18,76 s, Knex.js 19,96 s (+6 % verrattuna node-postgresiin) ja Bookshelf.js 21,80 s (+16 % verrattuna node-postgresiin).



Kuva 1. Suoritusaika sekunneissa viitekehyksittäin.

5.1.3.3 Keskustelu

Suorituskykykoe osoittaa, että kirjastojen välillä on olemassa selkeä ero tehokkuudessa. Koe on kuitenkin tehty hyvin yksinkertaisia operaatioita käyttämällä ja monimutkaisempien kyselyiden suorituskyvyn mittaaminen olisi olennainen jatkotutkimus.

5.2 Käytettävyys

Tässä kappaleessa listataan relaatiotietokantakirjastojen käytettävyyteen liittyviä asioita kehittäjän näkökulmasta.

5.2.1 Kieli

Yksi syy olio-relaatiomallintajan tai kyselyrakentajan käyttöön on tarve pelkälle JavaScriptille. Kehittäjien ei tarvitse opetella SQL-kieltä tietokantakutsujen tekemiseksi. (Ireland ja muut, 2009). Chen ja muut (2016) kuitenkin toteavat, että myös ORM:ien käyttö on opeteltava.

Kuten aiemmin todettu, SQL-kielestä on useita variaatioita; lähes jokaisella eri tietokannanhallintajärjestelmällä (DBMS) omansa. SQL-kielisiä kutsuja saattaa siis joutua kirjoittamaan uudestaan, jos esimerkiksi taustalla oleva DBMS päätetään vaihtaa toiseen. Knex, ja myös sitä hyödyntävä Bookshelf, tekevät kyselyt oikealla SQL:n variaatiolla,

kunhan käytössä oleva DBMS on oikein määritelty ja luonnollisesti myös tuettu. DBMS on siis yksinkertaisempaa vaihtaa käytettäessä Knexiä tai Bookshelfiä.

5.2.2 Ylläpidettävyys

Knex sisältää migraatiotyökalun, jota käytetään tietokantaskeemojen määrittelyyn ja muuttamiseen. Kyseinen migraatiotyökalu löytyy myös Bookshelfistä. (Bookshelf, 2021) Migraatiot helpottavat tietokannan ylläpitoa, kun tietokannan tilaa voidaan muuttaa yhdellä komennolla, ja jos esimerkiksi tietokannan tilan päivitys onkin virheellinen, voidaan tila palauttaa aiempaan (bin Uzayr, Cloud, & Ambler, 2019). Node-postgresia käytettäessä muutokset täytyy tehdä käsin SQL-kieltä käyttäen. Knex pitää myös huolta, että tarvittavat taulut luodaan tietokantaan, jos ne eivät ole jo olemassa. Tästäkin tulee node-postgresin kanssa huolehtia itse.

Toinen Knexin tarjoama ominaisuus ovat *seed*-tiedostot. Seed-tiedostot ovat migraatioiden kaltaisia sillä erolla, että seed-tiedostojen avulla voidaan määritellä dataa tietokannan rakenteen sijaan. Seed-tiedostot suorittamalla tietokanta voidaan täyttää esimerkiksi testausta varten. (bin Uzayr ja muut, 2019; Knex, 2021)

Bookshelf tuo mukaan mallit (models). Bookshelfin mallin määrittely esiteltiin aiemmin Bookshelfin esittelyn ohessa. Mallien käyttö näyttäytyy yksinkertaisessa esimerkissä SQL-kielistä kyselyä ja Knexin avulla tehtyä kyselyä monimutkaisempana. Malleista saadaan hyötyä, kun niiden välille muodostetaan relaatioita.

Chen ja muut (2016) huomasivat tutkimuksessaan, että ORM:n käyttö saattaa vaikeuttaa ohjelman ylläpitoa. Heidän mukaansa ORM:t eivät välttämättä ole niin tietokantaagnostisia kuin ne väittävät; tietokannan vaihtaminen toiseen ei siis ole välttämättä yksinkertaista. Tämän lisäksi ORM:iin liittyvä koodi on heidän mukaansa luonteeltaan pirstaleista ja monimutkaista, joka vaikeuttaa ohjelmakoodin ylläpitoa.

Bookshelf ja Knex myös mahdollistavat olioiden yksinkertaisen käytön esimerkiksi päivitys- tai luontioperaatioiden yhteydessä. Otetaan esimerkkinä suorituskykykokeen sisältämää koodia päivitysoperaatiosta Bookshelfin avulla:

```
1. for(i = 1; i <= 10000; i++){
2.     await new Post({'id': i}).save({...update}, {patch: true})
3. }
```

Yllä olevassa esimerkissä **update** on olio, joka sisältää tietokantaan päivitettävän datan. Olion attribuutit on nimetty tietokantataulun sarakkeiden mukaan. Kyseisessä koodiesimerkissä update-olio on muodostettu alla olevasta JSON-tiedostosta.

```
1. {
2.     "author_id": 6,
3.     "title": "Qui inventore fugiat quisquam neque esse et.",
4.     "description": "Quia enim suscipit repel-
   lat ex. Nemo modi officia quos corporis.",
5.     "content": "Doloribus quia iure debitis voluptatibus est fu-
   giat est. Dolores saepe aliquid ut fugit voluptates dolore. Eve-
   niet dolorem et aut eos iste facilis possimus. Vel si-
   milique itaque voluptates.",
6.     "date": "1997-04-30"
7. }
```

Vaikka JSON-tiedostosta poistettaisiin yksi attribuutti, esimerkiksi *date*, toimisi esitelty Bookshelfin päivityskoodi edelleen; päivämäärä ei vain enää päivittyisi tietokantaan.

Otetaan toisena esimerkkinä node-postgresin avulla tehty vastaava koodinpätkä:

```
1. for(i = 1; i <= 10000; i++) {
2.     await client.query(`
3.     UPDATE posts
4.     SET author_id=$2,
5.     title=$3,
6.     description=$4,
7.     content=$5,
8.     date=$6
9.     WHERE id = $1`,
10.    [i, update.author_id, update.title, update.description, up-
   date.content, update.date])
11.  }
```

Jos date-attribuutti puuttuisi, ei yllä oleva koodi toimisi halutulla tavalla. Kyselylle annettava parametri **update.date** saisi arvokseen *undefined*, joka muutettaisiin tietokannalle tyhjäksi arvoksi *NULL*.

Knex toimii olioiden kanssa vastaavalla tavalla kuin Bookshelf. Tässä on selkeä heikkous node-postgresissä: kyselyt on kirjoitettava uudestaan, jos tietokantaan syötettävän datan rakenne muuttuu.

5.3 Turvallisuus

Snykin Advisor-palvelun mukaan node-postgres on kärsinyt vakavasta tietoturva-aukosta viimeksi elokuussa 2017 ja korjattu versio paketista on julkaistu seuraavana päivänä. Knexin syyskuussa 2019 ja sitä aiemmin julkaistut versiot sen sijaan kärsivät SQL-injektiohaavoittuvuudesta. Haavoittuvuus on korjattu jo ennen sen julkituloa, mutta haavoittuvuus ehti olla paketissa yli kuusi vuotta. Vaikka Bookshelf käyttää hyväkseen Knexiä, ei Bookshelfistä ole raportoitu yhtään haavoittuvuutta. (Snyk, 2021)

Verkkosovellusten tietoturvaan keskittynyt OWASP listaa injektiohyökkäykset suurimpana tietoturvariskinä (OWASP, 2021). Kaikki vertailuun valitut viitekehykset mahdollistavat parametrisoitujen kyselyjen tekemisen. Parametrisoitujen kyselyiden käyttö lisää ohjelmiston tietoturvaa ehkäisemällä injektiohyökkäyksiä (Clarke-Salt, 2009, s. 371). Node-postgresiä käytettäessä osaamaton kehittäjä saattaa kuitenkin luoda mahdollisuuden SQL-injektiolle, jos kehittäjä rakentaa SQL-kyselyn merkkijonon dynaamisesti itse, eikä käytä kyselyparametreja potentiaalisesti haitallisen syötteen käsittelyyn. Alla olevassa esimerkissä kysely on muodostettu oikeaoppisesti kyselyparametriä käyttäen:

```
1. client.query('SELECT * FROM posts WHERE id = $1', [i])
```

Jos saman esimerkin kirjoittaisi vaarallisessa muodossa, se voisi näyttää esimerkiksi tältä:

```
1. client.query(`SELECT * FROM posts WHERE id = ${i}`)
```

Jos esimerkeissä oleva muuttuja **i** saadaan esimerkiksi käyttäjän syötteestä, on sen avulla mahdollista suorittaa haitallista koodia jälkimmäisessä esimerkissä.

6 Yhteenveto

Tutkimuksessa huomattiin, että erilaiset relaatiotietokantakirjastot eroavat toisistaan niin tehokkuuden, käytettävyyden, turvallisuuden ja ylläpidettävyydenkin osalta. Koska tarkasteltavat tietokantakirjastot rakentuivat toistensa päälle, vaikuttaa enemmän ominaisuuksia sisältävän kirjaston valinta negatiivisesti tehokkuuteen. Kirjaston valinnan vaikutusta projektiin voi kuitenkin olla hankala arvioida, ja lienee olennaisinta ymmärtää kirjaston valinnan vaikutukset, jotta mahdollisiin ongelmiin osataan varautua.

Knex.js lienee hyvä kompromissi tehokkuuden ja ominaisuuksien välillä, jos valinta on muuten hankalaa. On myös hyvä muistaa, että monimutkaisemmat kirjastot tarjoavat myös matalamman tason rajapinnan käyttöön, jolloin tehokkuutta vaativia kyselyjä olisi mahdollista suorittaa puhtaan SQL:n avulla myös Knexillä ja Bookshelfillä. Tämä tuo kuitenkin oman haasteensa monimutkaisemman lähdekoodin muodossa.

Aivan kuten Procaccianti ja muut (2016) totesivat tutkimuksessaan, myös tämän tutkimuksen suorituskykykoe osoittaa, että monimutkaisemmilla kirjastoilla on negatiivinen vaikutus suorituskykyyn. Erot suorituskyvyssä eivät kuitenkaan ole yhtä suuret kuin aiemmassa PHP-ajoympäristössä toteutetussa tutkimuksessa.

Lähdeluettelo

ACM DL. (2021). ACM Digital Library. Noudettu 5. kesäkuuta 2021, osoitteesta https://dl.acm.org/

bin Uzayr, S., Cloud, N., & Ambler, T. (2019). Knex and Bookshelf. Teoksessa S. bin Uzayr, N. Cloud, & T. Ambler, *JavaScript Frameworks for Modern Web Development* (ss. 377–426). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-4995-6_10

BLS. (2021). Bureau of Labor Statistics. Web Developers and Digital Designers: Occupational Outlook Handbook. Noudettu 5. maaliskuuta 2021, osoitteesta https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm

Bookshelf. (2021). Bookshelf.js | Home. Noudettu 20. maaliskuuta 2021, osoitteesta https://bookshelfjs.org/

Chen, T.-H., Shang, W., Yang, J., Hassan, A. E., Godfrey, M. W., Nasser, M., & Flora, P. (2016). An empirical study on the practice of maintaining object-relational mapping code in Java systems. *Proceedings of the 13th International Conference on Mining Software Repositories*, 165–176. Austin Texas: ACM. https://doi.org/10.1145/2901739.2901758

Clarke-Salt, J. (2009). *SQL Injection Attacks and Defense*. Elsevier.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387. https://doi.org/10.1145/362384.362685

Colley, D., Stanier, C., & Asaduzzaman, M. (2018). The Impact of Object-Relational Mapping Frameworks on Relational Query Performance. *2018 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, 47–52. https://doi.org/10.1109/iCCECOME.2018.8659222

Elsevier. (2021). Site search. Noudettu 5. kesäkuuta 2021, osoitteesta Elsevier.com website: https://www.elsevier.com/search-results

FillDB. (2021). Dummy data for MYSQL database. Noudettu 16. huhtikuuta 2021, osoitteesta Fill Database website: http://filldb.info/

Flanagan, D. (2020). *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language.* O'Reilly Media, Incorporated.

Google. (2021). Google. Noudettu 5. kesäkuuta 2021, osoitteesta https://www.google.fi/

GS. (2021). Google Scholar. Noudettu 5. kesäkuuta 2021, osoitteesta https://scholar.google.com/

Harrington, J. L. (2016). *Relational Database Design and Implementation, Fourth Edition*. Noudettu osoitteesta http://www.books24x7.com/marc.asp?bookid=113450

Ireland, C., Bowers, D., Newton, M., & Waugh, K. (2009). A Classification of Object-Relational Impedance Mismatch. *2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications*, 36–43. https://doi.org/10.1109/DBKDA.2009.11

Knex. (2021). Knex.js—A SQL Query Builder for Javascript. Noudettu 20. maaliskuuta 2021, osoitteesta http://knexjs.org/

Nevalainen, S. (2021). *Snottis/node-sql-testing* [JavaScript]. Noudettu osoitteesta https://github.com/snottis/node-sql-testing

Node.dev. (2021). Introduction to Node.js. Noudettu 20. maaliskuuta 2021, osoitteesta Introduction to Node.js website: https://nodejs.dev/

Node.js. (2021). Node.js. Noudettu 6. maaliskuuta 2021, osoitteesta https://nodejs.org/en/

OpenBase. (2021). 24 Most Popular Node.js PostgresSQL ORM Libraries in 2021. Noudettu 5. kesäkuuta 2021, osoitteesta Openbase website: https://openbase.com/categories/js/best-nodejs-postgressql-orm-libraries

OWASP. (2021). OWASP Top Ten Web Application Security Risks | OWASP. Noudettu 18. huhtikuuta 2021, osoitteesta https://owasp.org/www-project-top-ten/

PG. (2021). Node-postgres documentation. Noudettu 20. maaliskuuta 2021, osoitteesta https://node-postgres.com/

PHP. (2021). PHP: Hypertext Preprocessor. Noudettu 5. kesäkuuta 2021, osoitteesta https://www.php.net/

PostgreSQL. (2021, kesäkuuta 5). PostgreSQL. Noudettu 5. kesäkuuta 2021, osoitteesta PostgreSQL website: https://www.postgresql.org/

Procaccianti, G., Lago, P., & Diesveld, W. (2016). Energy Efficiency of ORM Approaches: An Empirical Evaluation. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10. Ciudad Real Spain: ACM. https://doi.org/10.1145/2961111.2962586

Scopus. (2021). Scopus preview—Scopus—Welcome to Scopus. Noudettu 5. kesäkuuta 2021, osoitteesta https://www.scopus.com/home.uri

Sequelize. (2021). Sequelize. Noudettu 5. kesäkuuta 2021, osoitteesta Sequelize ORM website: https://sequelize.org/

Snyk. (2021). Snyk Open Source Advisor | Snyk. Noudettu 7. huhtikuuta 2021, osoitteesta https://snyk.io/advisor/

SO. (2021). Stack Overflow. Developer Survey 2020. Noudettu 5. maaliskuuta 2021, osoitteesta https://insights.stackoverflow.com/survey/2020/

Torres, A., Galante, R., Pimenta, M. S., & Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications

on application design. *Information and Software Technology*, *82*, 1–18. https://doi.org/10.1016/j.infsof.2016.09.009

Wade, B. W., & Chamberlin, D. D. (2012). IBM Relational Database Systems: The Early Years. *IEEE Annals of the History of Computing*, *34*(4), 38–48. https://doi.org/10.1109/MAHC.2012.48