Antti Juhani Rapa

Relaatio- ja epärelaatiotietokantojen suorituskykyvertailu: MySQL ja MongoDB

Tietotekniikan pro gradu -tutkielma

1. kesäkuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Antti Juhani Rapa

Yhteystiedot: +358 44 576 4410, antti.j.rapa@student.jyu.fi, antti.rapa@gmail.com

Ohjaajat: Vesa Lappalainen ja Jonne Itkonen

Työn nimi: Relaatio- ja epärelaatiotietokantojen suorituskykyvertailu: MySQL ja MongoDB

Title in English: Relational databases and non-relational databases performance comparison: MySQL and MongoDB

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 61 + 31

Tiivistelmä: Tutkielmassa esitellään relaatio- ja epärelaatiotietokantoja sekä paneudutaan niiden ominaisuuksiin. Nykyään erilaisia tietokantahallintajärjestelmiä on paljon. Ongelmaksi tulee se, että mikä tietokantahallintajärjestelmä on järkevin valinta uudelle ohjelmistoprojektille. Valinnan tärkeyttä korostaa vielä se, että kehitettävä ohjelma tulee olemaan sidottu kyseiseen tietokantaan koko projektin aikajänteen ja sen vaihtaminen projektin edetessä tulee päivä päivältä kalliimmaksi. Alkuvaiheessa tehtävät valinnat voivat mennä hyvin helposti pieleen, koska tarpeellista toimialakohtaista osaamista ja kokonaisuuden hahmotusta ei ole välttämättä pystytty sisäistämään täydellisesti.

Tutkielman tarkoituksena on tarjota yksinkertaiset ohjeet, joita seuraamalla pystytään valitsemaan oikea tietokantahallintajärjestelmä uudelle projektille. Tutkimuksen teoreettinen tausta pohjautuu hyvin pitkälti relaatio- ja epärelaatiotietokantojen teoriaan. Näistä relaatiotietokantojen teorian pohja on luotu jo varhain 1970 -luvulla, kun taas epärelaatiotietokantojenteoria on kehittynyt vasta 2000 -luvulla. Tutkielmassa vertaillaan relaatio- ja epärelaatiotietokantoja keskenään hyvin suorituskykypainotteisesti. Koska eri tietokantahallintajärjestelmien kirjo on laaja, rajasin tutkittavat kohteet kahteen tuotteeseen molemmista päätyypeistä: MySQL ja MongoDB. Valintaa edesauttoi myös se, että minulla on henkilökohtaista kokemusta molemmista tietokantahallintajärjestelmistä.

Avainsanat: Tietokannat, relaatiotietokannat, nosql, nosql-tietokannat, relaatiomalli, web-kehitys

Abstract: This master thesis presents relational- and non-relational databases and their features. Nowadays there are variety of different database management systems and the problem is that which one of them is the most rational choice for a new software project. Software project will be bound to this choice it's whole timeframe and changing it will cost more and more when a project continues. The choices that people make in the beginning of a software project can go easily wrong. There can be a lack of domain specific knowledge or the overall view of the project is blurred.

This thesis will provide a simple instructions what you can follow and make right decisions about database management systems in the beginning of a software project. Research theory is based on relational- and non-relational databases theory. Relational database theory has been created in early 1970 and non-relational database in early 2000. In this thesis I will compare relational and non-relational databases through benchmark tests. Because there is so many different database management systems, I will focus on two products: MySQL and MongoDB. Choice has been made purely based on my knowledge about those products.

Keywords: Databases, relational databases, nosql, nosql databases, relational model, web development

Esipuhe

Haluan kiittää vanhempiani tuesta ja avustuksesta, mitä he ovat koko opiskeluajan minulle tarjonneet. Lisäksi suuri kiitos menee veljelleni jatkuvasta tuesta ja motivoimisesta. Ilman teitä en tiedä olisiko tutkielma valmistunut koskaan. Suuri kiitos myös kaikille ystäville, jotka ovat olleet mukana nämä viimeisimmät vuodet.

Jyväskylässä 1.7.2015

Antti Rapa

Termiluettelo

ACID	Atomicity, Consistency, Isolation, Durability on tietokantajärjestelmien periaate, jonka avulla turvataan tietojen eheys jokaisessa tilanteessa.
Atomisuus	(eng. Atomicity) Yksi ACID-ominaisuuksien periaatteista. Takaa sen, että transaktio suoritetaan aina onnistuneesti tai sitä ei suoriteta ollenkaan.
BASE	Basic Availability, Soft-state, Eventually consistency on tietokantajärjestelmien periaate, jonka avulla pyritään mahdollisimman kovaan suorituskykyyn ja saatavuuteen.
CAP	CAP on lyhenne sanoista eheys (eng. consistency), saatavuus (eng. availability) ja osituksen sietokyky (eng. partion tolerance). Hajautettu järjestelmä voi toteuttaa mainituista ominaisuuksista enintään kaksi.
Deadlock	On tilanne, missä esimerkiksi tapahtumaketjut yrittävät pyytää ristiin kirjoitusoikeutta tietokannan tauluihin. Tämän takia molemmat tapahtumaketjut jäävät jumittuneeseen tilaan.
Eheys	(eng. Consistency) Yksi CAP-teoreeman ja ACID-ominaisuuksien periaatteista. Pitää huolta, että tallennettavat tiedot noudattava määrättyjä oikeellisuussääntöjä.
Eristyneisyys	(eng. Isolation) Yksi ACID-ominaisuuksien periaatteista. Pitää huolta siitä, että samaa tietoa ei voida muokata yhtä aikaa useista eri prosesseista.
Indeksi	Indeksi pitää sisällään tiedon siitä, missä indeksoitu tietue sijaitsee tietokannassa. Tämän avulla nopeutetaan kyselyiden suorittamista.
NoSQL	”Not only SQL”. Tarkoitetaan tietokantoja, joiden kyselykielenä ei käytetä SQL-kieltä.

Osituksen sietokyky	(eng. Partion Tolerance) Yksi CAP-teoreeman periaatteista. Tarkoittaa sitä, että hajautettu järjestelmä pystyy jatkamaan toimintaa, vaikka solmujen välinen kommunikaatio ei toimi.
Perusavain	Jokainen relaatio (taulu) sisältää perusavaimen (eng. primary key), joka yksilöi relaation sisältämät tietueet.
Pysyvyys	(eng. Durability) Yksi ACID-ominaisuuksien periaatteista. Takaa sen, että tallennetut tiedot ovat aina saatavilla eikä ne tuhoudu järjestelmävirheden seurauksena.
Relaatio	Tietokanta koostuu yhdestä tai useammasta relaatiosta, jotka sisältävät tallennettua tietoa.
Saatavuus	(eng. Availability) Yksi CAP-teoreeman periaatteista. Takaa sen, että hajautetun järjestelmän solmun kaaduttua järjestelmä pysyy silti toiminnassa.
Tietue	Yksittäinen tietoalkio, joka on relaation sisällä. (kts. Relaatio)
Tiiviste	Tiiviste (eng. hash)
Transaktio	Transaktio on yhtenä toimenpiteenä suoritettava tapahtumaketju, jossa jonkin tapahtuman epäonnistuminen peruuttaa koko tapahtumaketjun.
Yhdistelmä avain	Kts. perusavain. Perusavainta kutsutaan yhdistelmä avaimeksi, jos se muodostuu kahdesta tai useammasta tietueen kentästä.
JSON	JavaScript Object Notation. Yksinkertainen kuvauskieli lähettää ja vastaanottaa dataa.
XML	Extensible Markup Language. Rakenteellinen ja monimutkaisempi (vrt. JSON) kuvauskieli jonka avulla voidaan lähettää ja vastaan ottaa dataa.

Kuviot

Taulukot

Sisältö

1 Johdanto

Tietokannat ovat kulkeneet tietotekniikan mukana hyvin varhaisesta ajasta lähtien. Ensimmäiset nykyisiä tietokantoja vastaavat tietokannat kehitettiin 1960-luvulla. Tällöin yhdysvaltalainen Charles W. Bachman vaikutti paljon niiden kehittymiseen. Myöhemmin vuonna 1973 hän sai työstään tunnustuksen Turingin palkinnon muodossa. Näihin aikoihin tietotekniikka oli vielä hyvin nuori tieteenala joten vahva tieteellinen näyttö puuttui tai oli hyvin puutteellinen. Näin ollen Charles W. Bachman oli ensimmäinen Turingin palkinnon voittanut, joka ei ollut koulutukseltaan tohtori. [1] Samoihin aikoihin E. F Codd, matemaattinen tiedemies, tutki miten relaatiomallia voidaan hyödyttää tietokantojen suunnittelussa. Tästä muodostui nykyään käytettävien relaatiotietokantojen perusteoria, jotka mallintavat konkreettisia arkipäivän asioita relaatioiden ja niiden välisten suhteiden muodossa. Vaikka Coddia pidetään nykyaikaisten relaatiotietokantojen ”isänä”, Bachmanin saavutuksia ei kuitenkaan pidä vähätellä. Molemmat henkilöt ovat omalla panoksellaan mahdollistaneet tietokantojen kehittymisen. [1]

Näiden jälkeen 1990-luvun lopussa Carlo Strozzi käytti ensimmäisen kerran termiä ”NoSQL” hänen kehittämälleen tietokannalle, jossa ei ollut käytössä SQL-kieltä. Myöhemmin noin kymmenen vuotta myöhemmin järjestettiin tapahtuma, jonka tarkoituksena oli löytää yhteinen nimitys tietokannoille, jotka eivät pohjautuneet perinteiseen relaatiomalliin ja ne eivät käyttäneet kyselykielenä SQL-kieltä. [2] Tapahtuma on yksi merkittävimmistä tapahtumista NoSQL-tietokantojen syntymiselle. Nykyään NoSQL-tietokannat ovat tulleet varteen otettavaksi vaihtoehdoksi relaatiotietokannoille. Ongelmaksi kuitenkin alkaa muodostua erilaisten tietokantahallintajärjestelmien määrä. Esimerkiksi pelkästään erilaisia NoSQL-tietokantoja on tällä hetkellä 150 kappaletta. [3] Lisäksi NoSQL-tietokannat ovat tällä hetkellä kategorisoitu tyypeittäin seuraaviin kategorioihin riippuen niiden ominaisuuksistaan: Sarake-, Dokumentti-, Avain-Arvo-, Verkko-, Monimalli-, Olio-ja XML-tietokantoihin. Lisäksi on monia tietokantoja, mitkä eivät suoraan mene mihinkään edellisistä kategorioista. [3]

Ohjelmistoprojektien alkaessa on mietittävä, minkälainen tietojen tallennusratkaisu kyseiseen projektiin sopii. Päätöstä ei ole välttämättä pakko tehdä heti, mutta jossain välissä se tulee kuitenkin eteen. Valittavana on useita erilaisia ratkaisuja, joista jokaisella on omia hyviä ja huonoja puolia. Mieleen voi tulla seuraavanlaisia kysymyksiä. Olemmeko toteuttamassa perinteistä työpöytäsovellusta vai verkkosovellusta? Haluammeko tallentaa tiedot keskitetysti vai hajautetusti? Minkä tyylistä tietoa ollaan tallentamassa? Näiden kysymysten jälkeen meillä on ehkä pieni käsitys siitä, minkälainen tietokanta sopii toteutettavalle sovellukselle.

Tutkielmassa esitellään perinteinen relaatiomalli, jota sovelletaan relaatiotietokannoissa ja tämän jälkeen paneudutaan NoSQL-tietokantojen toimintaperiaatteisiin. Lopullisena tarkoituksena on avata molempien tietokantojen hyviä ja huonoja puolia, jotta sovelluskehittäjien ja projektipäälliköiden on suhteellisen helppo valita sopiva tietokanta uudelle projektille ja välttää mahdolliset vaikeudet ja ongelmat projektin edetessä. Tutkielma sisältää paljon esimerkkejä ja kuvia, jotka olen pyrkinyt tekemään mahdollisimman yksinkertaisiksi ja helposti omaksuttaviksi.

2 Tietokannat

Monet arkipäiväiset asiat käyttävät hyväkseen tietokantoja. Esimerkiksi nykyisen asuntosi osoite on tallennettu johonkin tietokantaan, jotta posti löytää perille. Samalla myös tiedetään, että juuri sinä asut kyseisessä osoitteessa eikä kukaan muu. Tietokanta voi myös pitää sisällään tiedot edellisistä asukkaista, jotka asuivat ennen sinua kyseisessä osoitteessa. Jos jatketaan ajattelua vielä hieman pidemmälle, voidaan törmätä muihin itsestään selvänä pidettäviin asioihin, joiden takana on tietokanta. Olet kaupassa ostamassa ruokaa viikonlopuksi ja päätät maksaa maksukortilla. Asetat maksukortin sille tarkoitettuun lukijaan ja syötät PIN-koodin. Tämän jälkeen maksukortinlukija lähettää varmennuspyynnön palveluun, joka käy varmentamassa sen hetkisen maksukorttia vastaavan tilin katteen. Miten varmennus voi tietää kuinka paljon sinun tilillä on tällä hetkellä katetta? Tilin katetta on pakko säilyttää keskitetyssä tietokannassa, jotta pankki voi olla varma, että et käytä enemmän rahaa kuin mitä sinulla on.

Edellisten esimerkkien perusteella voidaan sanoa karkeasti, että tietokannat ovat kokoelma tietoa, jotka liittyvät johonkin reaalimaailman kohteeseen. Tietokanta voi olla keskitetty yhteen tietokoneeseen tai se voi olla hajautettu useammalle tietokoneelle. Hajautettu tietokanta voi lisäksi sijaita maantieteellisesti eri paikoissa. [4, p. 3]

3 Relaatiotietokannat

Tässä kappaleessa syvennymme relaatiotietokantojen ominaisuuksiin kuten siihen, että mitä tarkoitetaan perinteisellä relaatiomallilla ja mitä ovat relaatiotietokantojen keskeiset käsitteet. Kaikkia relaatiotietokantojen keskeisiä käsitteitä en tässä tutkielmassa ole käsitellyt vaan olen valinnut tutkielmaan muutaman omasta mielestäni tärkeän kohdan. Tutkielmasta puuttuu esimerkiksi relaatiotietokannoissa käytetyn SQL-kielen perusteet, tietokannansuunnittelu, näkymät, tietoturva, kyselyjen käsittelyt ja indeksointi.

3.1 Relaatiomalli

Relaatiotietokantojen tietomallin perustana on E.F. Coddin kehittämä relaatiomalli, joka esiteltiin ensimmäistä kertaa 1970 -luvulla. Relaatiomalli tarjoaa tavan mallintaa tietoa luonnollisessa muodossa ilman tarpeettomia lisärakenteita tietokoneita varten. [5] Lisäksi malli mahdollistaa tiedon johdannaisuuksien hallitsemisen, toistuvuuden vähentämisen ja relaatioiden ristiriidattomuuden. [5] Relaatiomallin peruskäsitteisiin kuuluvat relaatio (relation), sarake (field, attribute), tietue (record, tuple). On tärkeää huomata, että relaatiotietokannoista puhuttaessa relaatiolla viitataan tauluun ja tietueella riviin. [4, p. 36] Coddin määritysten mukaisesti jokainen relaatio koostuu nollasta tai useammasta tietueesta. Tietue jakaantuu useampaan sarakkeeseen, mikä määrittää relaation asteen. Esimerkiksi HENKILO -relaation asteluku on viisi (Kuva 1). Coddin mukaan relaatiolla on oltava seuraavat ominaisuudet:

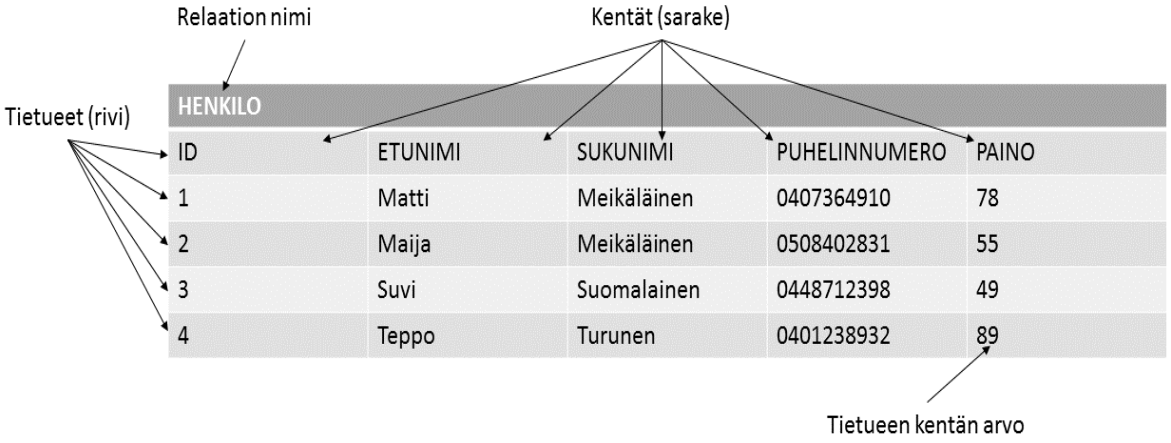
1. Jokainen tietue sisältää yhden tai useamman sarakkeen
2. Tietueiden järjestys taulussa on epäoleellinen
3. Jokainen tietue on erilainen
4. Sarakkeiden järjestyksellä on väliä
5. Sarakkeiden nimet pitäisi nimetä käytettävän asiayhteyden perusteella [5] [6]

Määritysten perusteella voimme määritellä seuraavanlaisen relaation (Kuva 1). Relaation nimeämisessä on syytä käyttää samoja perusteita kuin sarakkeiden nimeämisissä. Lisäksi

erikoismerkkejä ja skandinaavisia kirjaimia tulisi välttää mahdollisuuksien mukaan. [7] Määritelty HENKILO -relaatio sisältää viisi saraketta: ID, ETUNIMI, SUKUNIMI, PUHELINNUMERO ja PAINO, joten HENKILO -relaation asteluku on viisi. Lisäksi Codd määrittelee kolme syytä siihen miksi relaatioiden sarakkeiden nimet täytyvät olla toisistaan eroavat saman relaation sisällä:

1. Nimeämisellä on tarkoitus välittää käyttäjälle sarakkeen tietueiden sisältö. [6, p. 3] Esimerkiksi edellisessä HENKILO -relaatiossa mainittu sarake SUKUNIMI, joka antaa olettaa sen, että kaikki sarakkeiden tietueiden sisällöt ovat henkilöiden sukunimiä eikä esimerkiksi puhelinnumeroita.
2. Sarakkeiden nimeämisellä mahdollistetaan se, että käyttäjän ei tarvitse muistaa sarakkeiden järjestystä. [6, p. 3] Käytännössä relaatiomalli ei ota kantaa onko esimerkiksi HENKILO -taulun ETUNIMI -sarake ennen SUKUNIMI -saraketta vai toisinpäin, koska sarakkeiden sisällöt palautetaan nimen eikä järjestyksen mukaan.
3. Sarakkeiden nimet erottavat sarakkeet toisistaan ja liittävät ne asiayhteyteen eli mallinnettavaan kohteeseen. [6, p. 3] Esimerkiksi etunimi ja sukunimi ovat henkilöiden ominaisuuksia. Näin ollen on loogista nimetä henkilöitä mallintavan relaation sarakkeiden nimeksi ETUNIMI ja SUKUNIMI.

Codd määrittelee myös nimeämisen eduksi sen, että ei ole kovinkaan epätodennäköistä, että relaatio sisältää kymmeniä ellei satoja sarakkeita. Näin ollen sarakkeiden käsitteleminen järjestyksen mukaan ei ole kovinkaan mielekästä ja käyttäjäystävällistä. [6, p. 3] Edellisessä kappaleessa määrittelemä HENKILO -relaatio sisältää vain viisi saraketta, joten sarakkeiden järjestys olisi tässä tapauksessa vielä suhteellisen helposti muistettavissa. Reaalimaailman tapauksessa HENKILO -relaatio voisi pitää sisällään tiedon henkilön asuinpaikasta kuten postinumero, postiosoite ja postitoimipaikka tai siihen voisi sisällyttää vielä lisää henkilön ominaisuuksia kuten pituus, hiusten väri, sukupuoli, jne. Näin ollen HENKILO -relaation asteluku kasvaa nopeasti ja järjestyksen muistamien olisi työlästä.



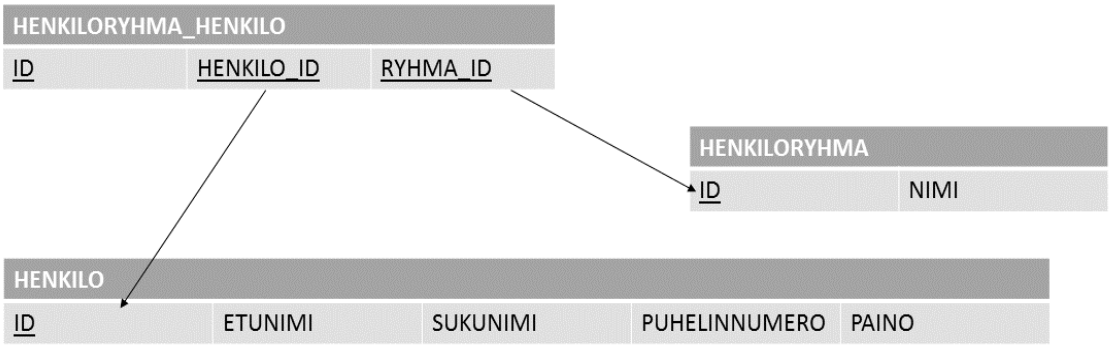
Kuva 1: Yksinkertainen HENKILO -relaatio

3.2 Peruskäsitteet

Seuraavissa alakappaleissa käsitellään relaatiotietokantoihin liittyviä keskeisiä asioita. Kuten sitä, että mitä tarkoitetaan relaatiolla ja miten niiden sisältö rakentuu. Lisäksi kerrotaan hieman relaatiotietokannan ominaisuuksista kuten mitä tarkoittaa ACID-ominaisuus ja miten relaatiotietokantoja normalisoidaan, jotta voidaan vähentää tiedon toistuvuutta tietokannassa.

3.2.1 Relaatiot

Relaatiomallissa tietokanta koostuu yhdestä tai useammasta relaatiosta, joiden välillä voi olla riippuvuuksia (Kuva 2). [7] Tietokanta on tyhjä, jos se ei sisällä yhtään relaatiota. Relaatiot pitävät sisällään vaihtelevan määrän tietueita. Relaatio taas on tyhjä, jos se ei sisällä yhtään tietuetta. Näin ollen voidaan määrittää, että tietokanta on kokoelma relaatioiden sisältämiä tietueita. Jokaiseen relaatioon täytyy määrittää yksikäsitteinen perusavain (primary key), jonka tarkoituksena on yksilöidä taulun sisältämät tietueet. [7] Perusavaimesta kerrotaan tarkemmin kappaleessa 3.2.3



Kuva 2: Kuva mallintaa relaatioiden väliset liitokset.

3.2.2 Sarake

Jokainen relaatio sisältää sen asteenluvun määräämän verran sarakkeita, joista jokainen sarake on homogeeninen [5]. Homogeenisuuden mukaan sarakkeen sisältämä tieto on tyypiltään sama katsottuna miltä riviltä tahansa. Relaatiotietokannoissa sarakkeet sisältävät erilaisia ominaisuuksia, mitkä määrittävät sarakkeen käyttäytymisen ja tietosisällön. [7] Näitä ominaisuuksia ovat esimerkiksi:

- Sarakkeen nimi, jonka tulisi olla mahdollisimman kuvaava
- Tietotyyppi, joka määrittää minkälaista tietoa kyseiseen sarakkeeseen voidaan tallentaa. Yleisiä tietotyyppejä ovat mm. merkkijono (VARCHAR, TEXT), numeerinen (INT, FLOAT, DECIMAL), aika (DATE, DATETIME) ja looginen (BOOLEAN)
- Maksimipituus, joka määrittää miten pitkä esimerkiksi merkkijono voi olla.
- Tarkkuus, joka määrittää kuinka monella desimaalilla esimerkiksi numeerinen – tietotyyppi tallennetaan.
- Oletusarvo, joka määrittää mitä sarakkeen sisällöksi laitetaan, jos tallennettava tieto tulee tyhjänä.

3.2.3 Avaimet

Jokaiselle tietokannan relaatiolle tulee määritellä yksikäsitteinen perusavain (primary key). Perusavaimen tehtävänä on yksilöidä jokainen rivi tietokannan taulun sisällä. [5, p. 380] Yleensä perusavaimeksi valitaan yksittäinen kenttä, jonka sisältämää arvoa ei esiinny muilla tietueilla. Sen arvoja voi olla mm. juokseva numerointi ykkösestä ylöspäin, henkilötunnus tai uniikki tiiviste (hash). Perusavain voi myös olla yhdistelmä avain (composite key), jos perusavaimeksi valitaan useampi kenttä ja nämä kentät yhdessä yksilöivät tietueen. [5, p. 380]

Perusavaimia käytetään tietojen ristiviittauksiin eri relaatioiden välillä. [5, p. 380] Esimerkiksi HENKILO -relaatio (Kuva 1), joka sisältää perusavaimen ID, mikä yksilöi jokaisen henkilön tiedot, voidaan liittää toiseen relaatioon käyttämällä hyväksi perusavainta. Tämä tapahtuu niin, että liitettävään relaation tietueeseen lisätään sarake, joka viittaa HENKILO -relaation perusavaimeen. Sarakkeen sisältöä, joka liittää relaatiot yhteen kutsutaan nimellä viiteavain (foreign key).

3.2.4 Normalisointi

Relaatiomallissa tietokantaan voidaan tallentaa tietoa eri tavoilla. Käytännössä tämä tarkoittaa sitä, että tietueet voidaan jakaa relaatioihin usealla eri tavalla. Esimerkiksi mikään ei estä, että HENKILO -relaation ETUNIMI- ja SUKUNIMI -sarakkeet voitaisiin yhdistää NIMI -sarakkeeksi. Normalisoinnin tarkoituksena on vähentää tiedon toistuvuutta (redundancy) ja helpottaa tietojen päivitystä tietokannassa. [8, p. 93] Tarkoittaen, että jokainen tieto tulisi esiintyä vain yhden kerran tietokannassa. Näin ollen tieto pitäisi päivittää vain yhteen tietueeseen eikä useampaan. Tietokanta voidaan normalisoida ainakin kahdella eri tavalla. Yleisin tapa on mallintaa tietokanta ER-kaavioksi. ER-kaavion avulla tietokanta voidaan normalisoida noudattamalla ennalta määriteltyjä sääntöjä. Toinen monimutkaisempi normalisointi tapa on käyttää teoreettisia malleja relaatioiden suunnittelun. [8, p. 93] Tietokannan taulut voivat olla useassa eri normaalimuodossa riippuen siitä, mitä ominaisuuksia taulut toteuttavat. Jos tietokanta ei toteuta mitään normaalimuotojen määrittelemiä ehtoja, on tietokanta normalisoimaton. (Kuva 3: Normalisoimaton HENKILO -taulu.

Ensimmäinen normaalimuoto (1NF) määrä sen, että taulujen attribuuttien on oltava atomisia (atomic). Tämä tarkoittaa sitä, että attribuutin sisältö ei voi olla rakenteellinen ja sisältää useita eri arvoja. [4, p. 207] Esimerkiksi alapuolella määritelty taulu (Kuva 3) ei ole ensimmäisessä normaalimuodossa, koska sen sisältämä HENKILORYHMA -kenttä on rakenteellinen. Taulu voidaan normalisoida ensimmäiseen normaalimuotoon hajottamalla HENKILORYHMA -kentän sisältö kahdeksi kentäksi.

HENKILO					
ID	ETUNIMI	SUKUNIMI	PUHELINNUMERO	PAINO	HENKILORYHMA
1	Matti	Meikäläinen	0407364910	78	Opiskelija, false
1	Matti	Meikäläinen	0407364910	78	Opettaja, false
3	Maija	Meikäläinen	0508402831	55	Passiiviset Opettajat, true
4	Suvi	Suomalainen	0448712398	49	Opiskelija, false
5	Teppo	Turunen	0401238932	89	Passiiviset Opiskelijat, true

Kuva 3: Normalisoimaton HENKILO -taulu.

HENKILO						
ID	ETUNIMI	SUKUNIMI	PUHELINNUMERO	PAINO	HENKILORYHMA	HENKILORYHMA_PASSIIVINEN
1	Matti	Meikäläinen	0407364910	78	Opiskelija	false
1	Matti	Meikäläinen	0407364910	78	Opettaja	false
3	Maija	Meikäläinen	0508402831	55	Passiiviset Opettajat	true
4	Suvi	Suomalainen	0448712398	49	Opiskelija	false
5	Teppo	Turunen	0401238932	89	Passiiviset Opiskelijat	true

Kuva 4: HENKILO -taulu ensimmäisessä normaalimuodossa.

Taulu voi olla toisessa normaalimuodossa (2NF), jos se toteuttaa ensimmäisen normaalimuodon ja lisäksi kaikki attribuutit, jotka eivät ole perusavaimia ovat funktionaalisesti riippuvia perusavaimesta. (Kuva 5) Ensimmäisestä normaalimuodosta toiseen normaalimuotoon päästään luomalla uusi taulu HENKILORYHMA, joka sisältää henkilöryhmään liittyvät tiedot. Toinen normaalimuoto ei ole kuitenkaan täydellinen, koska Matti Meikäläisen tiedot ovat HENKILO -taulussa useamman kerran. Esimerkiksi puhelinnumeron päivittäminen jouduttaisiin tekemään kahteen eri tietueeseen.

HENKILO					
ID	ETUNIMI	SUKUNIMI	PUHELINNUMERO	PAINO	HENKILORYHMA_ID
1	Matti	Meikäläinen	0407364910	78	1
1	Matti	Meikäläinen	0407364910	78	2
3	Maija	Meikäläinen	0508402831	55	4
4	Suvi	Suomalainen	0448712398	49	1
5	Teppo	Turunen	0401238932	89	3

HENKILORYHMA		
ID	NIMI	PASSIIVINEN
1	Opiskelija	false
2	Opettaja	false
3	Passiiviset Opiskelijat	true
4	Passiiviset Opettajat	true

Kuva 5: HENKILO- ja HENKILORYHMA -taulu toisessa normaalimuodossa.

Toisesta normaalimuodosta kolmanteen normaalimuotoon päästään, kun poistetaan tietueen arvot, jotka eivät ole riippuvaisia perusavaimesta. Käytännössä siis HENKILO –taulussa (Kuva 5) oleva HENKILORYHMA_ID ei ole riippuvainen ID –kentästä, joka toimii kyseisessä taulussa perusavaimena. Tästä syystä Matti Meikäläinen (ID = 1) joudutaan pitämään taulussa kahteen kertaan, koska hän kuuluu kahteen henkilöryhmään. Normalisointisääntöjen mukaisesti tässä tapauksessa HENKILORYHMA_ID –kenttä täytyy sijoittaa uuteen tauluun HENKILO_HENKILORYHMA. (Kuva 6) Kyseinen taulu toimii liitoksena HENKILO- ja HENKILORYHMA –taulun väillä. Tämän ansiosta vältymme useamman rivin päivitykseltä esimerkiksi Matti Meikäläisen puhelinnumeroa muuttamalla.

HENKILO				
ID	ETUNIMI	SUKUNIMI	PUHELINNUMERO	PAINO
1	Matti	Meikäläinen	0407364910	78
3	Maija	Meikäläinen	0508402831	55
4	Suvi	Suomalainen	0448712398	49
5	Teppo	Turunen	0401238932	89

HENKILO_HENKILORYHMA		
ID	HENKILO_ID	HENKILORYHMA_ID
1	1	1
2	1	2
3	3	4
4	4	1
5	5	3

HENKILORYHMA		
ID	NIMI	PASSIIVINEN
1	Opiskelija	false
2	Opettaja	false
3	Passiiviset Opiskelijat	true
4	Passiiviset Opettajat	true

Kuva 6: Taulut toisessa normaalimuodossa.

3.2.5 Indeksointi

Indeksit ovat tietokoneen keskusmuistiin ladattava tietorakenne, jonka tehtävänä on tehostaa tietokannasta tiedon hakemista. Tietokannan indeksit tallennetaan levylle, joista ne ladataan tarvittaessa keskusmuistiin myöhempää käyttöä varten. Levylle tallentaminen mahdollistaa

sen, että palvelimen uudelleen käynnistämisen jälkeen indeksit voidaan lukea takaisin muistiin eikä ne katoa pysyvästi. [9]

Tietokannan indeksi koostuu joukosta indeksejä, joita on yksi jokaiselle taulun riville tietokannassa. Olettaen, että taululle on määritelty indeksi. Käytännössä indeksit säilötään taulumaiseen rakenteeseen, mikä sisältää kaksi saraketta: indeksi avain ja osoite tietoon, mistä indeksi on luotu. [9]

Indeksien käyttämistä voidaan ajatella niin, että etsitään henkilöitä kaupungista. Kaupungin jokaisen asunnon osoitteet ja niissä asuvat henkilöt on tallennettu tietokantaan. Indeksointi on hoidettu niin, että jokaisen henkilöiden nimet on kirjoitettu aakkosjärjestyksessä paperille. Lisäksi nimen perässä on tieto siitä, missä kyseinen henkilö asuu. Näin ollen voimme etsiä paperilta henkilöä Matti Meikäläinen ja näin ollen saamme kyseisen henkilön osoitteen selville. Hakua voidaan myös toteuttaa ilman indeksejä niin, että käymme kaikki asunnot kaupungista läpi ja kysymme jokaiselta henkilöltä, että oletko sinä Matti Meikäläinen.

Lisäksi indekseillä on suuri merkitys kyselynoptimoinnissa. [9] En kuitenkaan paneudu tässä tutkielmassa kyseiseen aiheeseen, joten siitä voi lukea enemmän esimerkiksi kirjasta Database: Principles, Programming, Performanse (Patrick O'Neill)

3.3 ACID – ominaisuudet

ACID on akronyymi ja se muodostuu sanoista: atomisuus (Atomicity), eheys (Consistency), eristyneisyys (Isolation) ja pysyvyys (Durability). [10, 11] ACID -ominaisuudet määrittävät kokonaisuudessaan, miten transaktion tulisi toimia. [11] Transaktiota voidaan kuvata, että se on joukko tapahtumia, mitkä johtavat johonkin pysyvään lopputulokseen. [11] Havainnollistetaan tätä esimerkin kautta. Kaksi ihmistä neuvottelevat asuntokaupoista, missä toisen henkilön tavoitteena on myydä asunto ja toisen ostaa. Atomisuus takaa sen, että kaupat syntyvät tai ne perutaan. Eheys taas pyrkii siihen, että neuvottelu hoidetaan käyttämällä sovittuja protokollia. Sovittujen protokollien käyttäminen takaa sen, että neuvottelun jälkeen transaktio on siirtynyt eheästä tilasta eheään tilaan (kauppojen syntymiseen tai perumiseen). Eristyneisyydellä pyritään välttämään useiden neuvottelujen käyminen samaan aikaan (vrt.

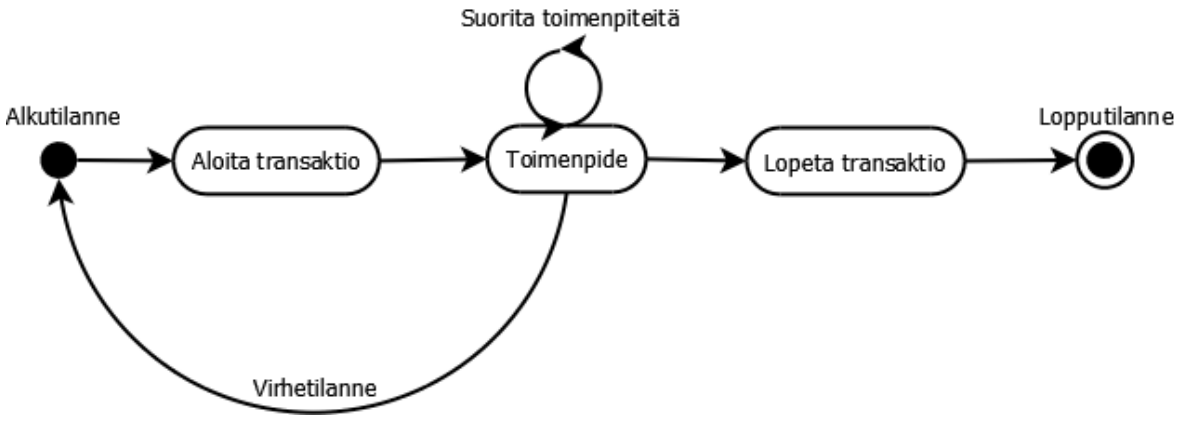
myyjä yrittää myydä asuntoa yhtäaikaisesti monelle henkilölle, jonka jälkeen kaikki yrittää ostaa asuntoa). Viimeinen ominaisuus eli pysyvyys tarkoittaa sitä, että kauppojen tekemisen myötä kauppoja ei voida kumota, joten kaupat ovat pysyvät (peruminen voidaan hoitaa uudella transaktiolla). ACID-ominaisuuksista kerrotaan tarkemmin seuraavissa alaluvuissa.

3.3.1 Atomisuus

Atomisuus takaa sen, että transaktio suoritetaan aina alusta loppuun tai sitä ei suoriteta olleenkaan. Kaikki virheet, mitkä tapahtuvat transaktion aikana tai tapahtumat, jotka rikkovat muita ACID-ominaisuuksia, keskeyttävät transaktion ja tietokanta palaa tilaan, missä se oli transaktion alettua. Alkutilaan palaaminen voidaan ajatella niin, että transaktiota ei ole koskaan edes aloitettu suorittamaan. [4, p. 767] Transaktion keskeytykselle on useita syitä, joita Kifer, Bernstein ja Lewis on luetellut teoksessaan. Transaktio voi keskeytyä, koska

- sen jatkaminen voisi vahingoittaa tietokannan eheyttä,
- jatkamisella rikotaan eristyneisyyttä (kuvattu kappaleessa 3.3.3),
- se on joutunut lukittuneeseen tilaan (deadlock),
- järjestelmä kaatuu suoritus hetkellä. [4, pp. 767-768]

Seuraavassa kuvassa on esitetty atominen transaktio, joka päätyy joko lopputilanteeseen tai alkutilanteeseen (Kuva 7). Transaktion suorittamien aloitetaan aloituspyynnöllä, jonka jälkeen seuraa toimenpiteet. Toimenpiteiden aikana transaktio suorittaa sille ennalta määrättyjä toimenpiteitä (esimerkiksi tietueen arvon muuttaminen). Toimenpiteiden jälkeen transaktion suoritus lopetetaan ja muutokset hyväksytään. Virheiden sattuessa siirrytään toimenpiteestä suoraan alkutilanteeseen ja tuhotaan transaktio.



Kuva 7: Atomisen transaktion prosessi.

3.3.2 Eheys

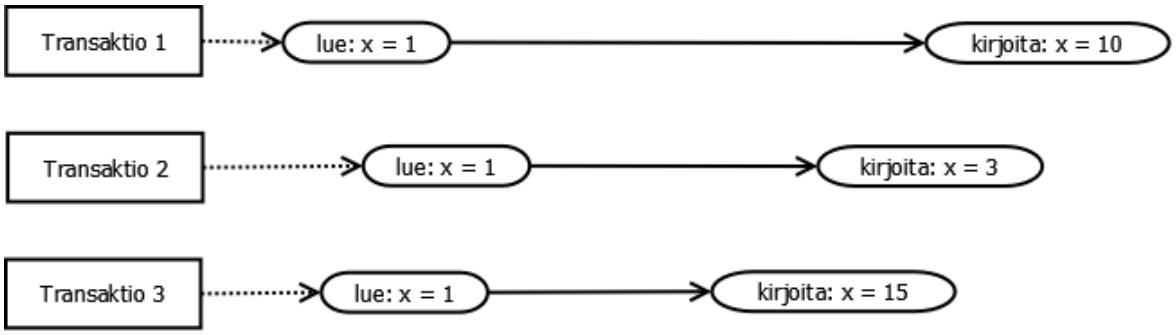
Tietokannan eheys on ominaisuus, joka kuvaa tietokannan sen hetkistä tilaa verrattuna reaalimaailman ympäristöön. [4, p. 764] Reaalimaailman ympäristöllä tarkoitetaan kohdetta reaalimaailmasta, mitä tietokanta mallintaa. Esimerkiksi HENKILO -taulu (Kuva 1). Eheyden kaksi näkökulmaa, mitkä tietokannan on aina täytettävä, ovat seuraavat [4, p. 764]

- Sisäinen eheys, joka pyrkii siihen, että tietokannan mallintamat kohteet noudattavat annettuja sääntöjä. Esimerkiksi HENKILO -taulu ei voi sisältää samaa henkilöä moneen kertaan.
- Ympäristön säännöt, jotka määrittyvät reaalimaailman kautta. Jos mallinnettavalla kohteella on reaalimaailmassa sääntöjä, tulee niitä noudattaa myös tietokantatasolla. Esimerkiksi HENKILO -taulussa olevan henkilön paino täytyy olla suurempi kuin nolla tai puhelinnumero ei voi sisältää kirjaimia.

Jokaisen transaktioketjun on noudatettava kaikkia eheyden vaatimia sääntöjä. Tarkoittaen sitä, että tietokannan on oltava eheässä tilassa ennen transaktion alkua ja jäätävä eheään tilaan transaktion loputtua. [4, pp. 764-765]

3.3.3 Eristyneisyys

Eristyneisyys tarkoittaa sitä, että usean transaktion samanaikainen suorittaminen täytyy päätyä samaan tilanteeseen kuin transaktioiden suorittaminen järjestyksessä. Tämä voidaan saavuttaa sillä, että transaktiot voivat lukita tietokannan tauluja tai rivejä sitä mukaan, kun niitä käsitellään. Lukitseminen estää yhtäaikaisen muokkauksen ja näin ollen takaa sen, että transaktion jälkeen tietokanta on yhä eheässä tilassa. [4, pp. 769-772] Eristyneisyyden tärkeyttä voidaan selventää esimerkin kautta (Kuva 8). Kuvassa on mallinnettu usean transaktion suorittamista samanaikaisesti. Transaktio 1 (T1) aloittaa lukuoperaatiolla, joka kohdistuu tietueeseen x. Seuraavaksi transaktio 3 (T3) ja transaktio 2 (T2) suorittaa oman lukuoperaation samaan tietueeseen. Koska luku- ja kirjoituslukkoja ei käytetä tässä esimerkissä, transaktiot voivat syöttää tuloksen tietueeseen x nopeusjärjestyksessä, jolloin viimeisin arvo jää voimaan. Esimerkki kuvan tapauksessa T1:sen kirjoittama arvo jää voimaan. T2:sen ja T3:sen muutokset katoavat T1:sen kirjoituksen takia.



Kuva 8: Useampi transaktio ja eristyneisyyden puuttuminen

3.3.4 Pysyvyys

Pysyvyys määrittelee sen, että tietokantaan tallennetut tiedostot ovat käytettävissä eikä tietokanta hävitä tallennettuja tietoja missään vaiheessa. Tietokannan tulee, myös toipua äkillisestä tilanteesta kuten järjestelmän sammumisesta tai laiteviasta. [4, pp. 768-769] Pysy-

vyyteen liittyy hyvin vahvasti muut ACID -ominaisuudet. Esimerkiksi juuri ennen vikatilannetta onnistuneesti suoritetun transaktion muutokset täytyy löytyä myös vikatilanteen korjaannuttua tietokannasta.

4 NoSQL -tietokannat

Edellisessä kappaleessa käsiteltiin perinteisiä relaatiomallia noudattavia tietokantoja ja niiden ominaisuuksia. Tässä kappaleessa esitellään relaatiomallista poikkeavia tietokantoja, joita kutsutaan yleisemmin nimellä NoSQL-tietokannat. NoSQL-termi tulee sanoista ”Not only SQL” eli tietokantoja, joiden kyselykielenä ei ole SQL-kieli. Termi tuli ensimmäisen kerran tutuksi vuonna 1998, jolloin Carlo Strozzi käytti kyseistä nimeä hänen vapaaseen lähdekoodiin perustuvassa tietokannassa, jossa ei ollut käytössä SQL-kieltä. [12] Tämän jälkeen vasta yli kymmenen vuotta myöhemmin heinäkuussa 2009 järjestettiin tapahtuma, jossa tarkoituksena oli löytää yhteinen nimitys kasvavalle määrälle tietokantajärjestelmiä, jotka eivät sopineet enää perinteiseen relaatiomalliin. [2] NoSQL-tietokannat ovat suunniteltu niin, että ne ovat horisontaalisesti helposti skaalautuvia hajautettuja tietokantoja ja pystyvät vastaamaan nykyajan kasvavaan informaation määrän käsittelyyn. [12] Toisin kuin relaatiotietokannoissa NoSQL-tietokannoissa ei ole erikseen määriteltyä skeemaa. Eli tietokannan rivit eivät ole välttämättä homogeenisia. Näiden ominaisuuksien avulla NoSQL-tietokannoista pystytään tekemään suorituskykyisiä, saavutettavissa olevia ja skaalautuvia. Ominaisuuksiin pureudutaan tarkemmin alakappaleissa 4.2 ja 4.3.

4.1 Peruskäsitteet

Tässä kappaleessa käsitellään NoSQL-tietokantoihin liittyviä keskeisiä asioita. Kuten sitä, että mitä tarkoitetaan skaalautuvuudella ja sirpaloinnilla. Lisäksi käsittelemme hiukan NoSQL-tietokannoissa käytettävää BASE –oikeellisuusmallia, joka on verrattavissa relaatiotietokannoissa olevaan ACID-ominaisuuksiin. Lisäksi kerron hieman CAP-teoreemastta, joka on tärkeä osa hajautettuja tietokantoja.

4.1.1 MapReduce

MapReduce on ohjelmointimalli, jonka on kehittänyt Google Oy vuonna 2003. Sen tarkoituksena on helpottaa ja nopeuttaa suurien tietomäärien hajautettua prosessointia ja generointia. [13, p. 1] [14, p. 1] MapReducen toiminta perustuu siihen, että käyttäjä määrittää kaksi

funktiota map ja reduce. Map-funktion tarkoitus on muodostaa avain-arvo -pareja ennalta määritellyn toimintalogiikan perusteella. Tulokseksi saatu avain-arvo -parijoukko välitetään reduce-funktiolle. Tämän jälkeen Reduce-funktion tehtävänä on yhdistää syötteenä saadut parit lopulliseksi tulosjoukoksi. [13, p. 1] [14, p. 1]

Havainnollistetaan toimintalogiikka esimerkin kautta, jossa käytämme hyväksi HENKILO -taulun tietoja (Kuva 1). Esimerkissä oletamme, että henkilötietoja on määritelty useita tuhansia kuvassa määritellyn neljän henkilön lisäksi. Esimerkin tarkoituksena on hakea kuinka monta henkilöä kuuluu määrättyihin painoryhmiin. Määritellään painoryhmät seuraaviksi:

- alle 50 kg
- 50–60 kg
- 60–70 kg
- 70–80 kg
- 80–90 kg
- yli 90 kg.

Näiden pohjalta voidaan märittää seuraava pseudokoodimuotoinen map-funktio:

```
/**
 * Map-funktio henkilöiden läpikäymiseen ja
 * painoryhmien esiintymistiheyksien laskemiseen.
 *
 * @param avain    Dokumentin avain
 * @param henkilo  Dokumentin sisältö (ts. avain)
 */
function map(avain, henkilo) {
     if (henkilo.paino < 50) {
         emit("alle 50 kg", 1);
     } else if (henkilo.paino >= 50 && henkilo.paino < 60) {
         emit("50-60 kg", 1);
     } else if (henkilo.paino >= 60 && henkilo.paino < 70) {
         emit("60-70 kg", 1);
     } else if (henkilo.paino >= 70 && henkilo.paino < 80) {
         emit("70-80 kg", 1);
     } else if (henkilo.paino >= 80 && henkilo.paino < 90) {
         emit("80-90 kg", 1);
     } else {
         emit("yli 90kg", 1);
     }
}
```

Esimerkissä map-funktio tekee jokaiselle henkilölle painoluokka vertailun ja muodostaa siitä avain-arvo -parin. Avaimena käytetään tässä tapauksessa painoryhmän nimeä ja arvona kokonaislukua yksi. Map-funktion suorittamisen jälkeen saadaan tulokseksi lista, joka sisältää kaikkien henkilöiden painoluokan ja kokonaisluvun yksi. (Taulukko 1)

Avain	Arvot
alle 50 kg	[1, 1]
50–60 kg	[1, 1, 1, 1, 1, 1]
60–70 kg	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
70–80 kg	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
80–90 kg	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
yli 90 kg	[1, 1, 1, 1, 1, 1, 1, 1]

Taulukko 1: Map-funktiolta saatu tulosjoukko.

Tämän jälkeen tulosjoukkoa käytetään hyväksi reduce –funktion suorituksessa. Reduce –funktio ottaa vastaan kaksi parametria, joita ovat avain (key) ja sitä vastaavat arvot (values). Esimerkki tapauksessa voitaisiin reduce –funktiota käyttää hyväksi summaamaan jokaisen ryhmän esiintymät. Reduce –funktio voisi olla seuraavanlainen.

```
/**
 * Recude –funktio painoryhmien esiintymisien summaamiseen
        * ja kokonaistuloksen laskemiseen
 *
 * @param avain    Dokumentin avain
 * @param arvot    Dokumentin arvot
 */
function reduce(avain, arvot) {
     return Array.sum(arvot);
}
```

Tässä esimerkissä reduce –funktio saisi syötteenä rivi riviltä map –funktion tuloksen, jonka seurauksena saadaan lopullinen tulosjoukko, joka sisältää avaimet ja rivin arvot summattuna yhteen tulokseen. (Taulukko 2)

Avain	Tulos
alle 50 kg	2
50–60 kg	6
60–70 kg	11
70–80 kg	15
80–90 kg	12
yli 90 kg	8

Taulukko 2: Reduce –funktiolta saatu tulosjoukko.

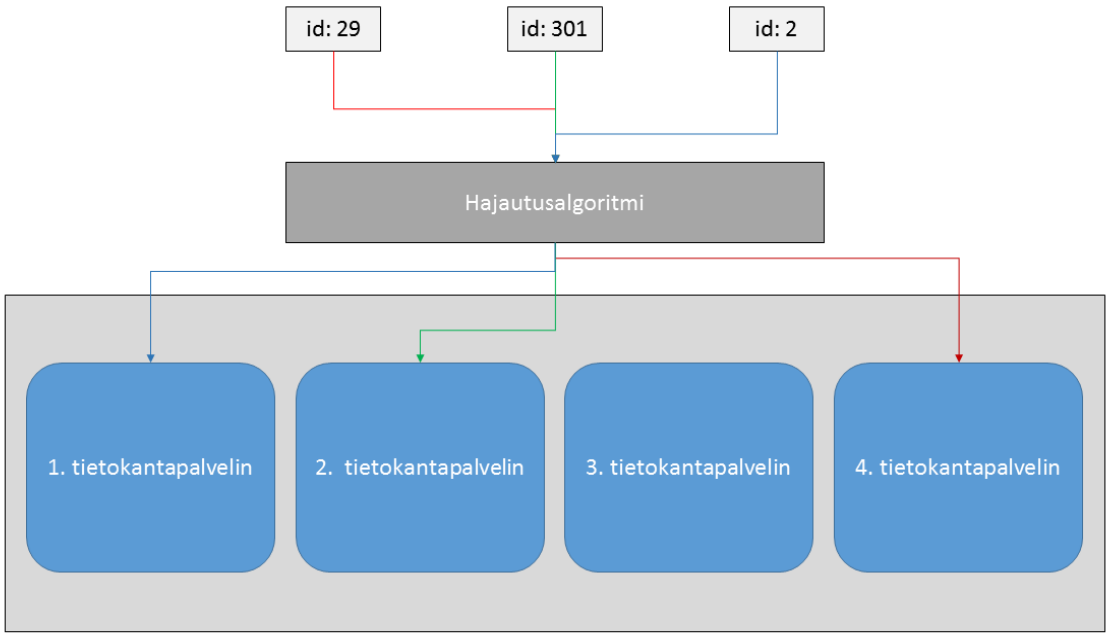
4.1.2 Skaalautuvuus

Tietokantojen skaalautuvuus on yksi isoista kysymyksistä tällä vuosikymmenellä. Nykyiset pilvipalvelut säilövät tietoa suuria määriä, että perinteisellä yhden tietokantapalvelin pitämisellä ei päästä kovin pitkälle. Ongelmaksi tässä tulee se, että käyttäjä- ja tietomäärän kasvaessa palvelimen kapasiteetti ei riitä palvelemaan kaikkia. Tilanteen voi ratkaista kahdella eri tavalla: vertikaalisesti tai horisontaalisesti skaalautumalla. [15]

Relaatiotietokannat voidaan skaalata vertikaalisesti. Vertikaalinen skaalautuminen tapahtuu lisäämällä palvelimeen lisää muistia, prosessoreita ja tallennustilaa. Vertikaalinen skaalautuminen on hyvin rajallista verrattuna horisontaaliseen skaalautumiseen, koska yhtä palvelinta ei voi loputtomasti laajentaa kustannustehokkaasti. Relaatiotietokannoissa horisontaalinen skaalautuminen tapahtuu yleensä replikoimalla tietokanta useammalle palvelimelle. [15]

Relaatiotietokantojen skaalautuminen horisontaalisesti on haastavaa ja kustannustehotonta. Tästä syystä epärelaatiokantojen tärkeimpiin ominaisuuksiin kuuluu horisontaalinen skaa-

lautuvuus. Esimerkiksi The Apache Software Foundation kehittämä Cassandra tietokantahallintajärjestelmä skaalautuu lineaarisesti horisontaalisessa suunnassa. [16] Käytännössä tämä tarkoittaa siis sitä, että jos yksi tietokantapalvelin ei riitä palvelemaan kaikkia asiakkaita niin toinen tietokantapalvelin voidaan laittaa sen rinnalle. Tätä ei kuitenkaan tehdä replikoimalla ensimmäisen palvelimen tietoja uudelle palvelimelle vaan hajauttamalla tallennettava tieto eri palvelimille. Hajautuksessa käytetään tyypillisesti hajautustauluja (Hash table). [15]

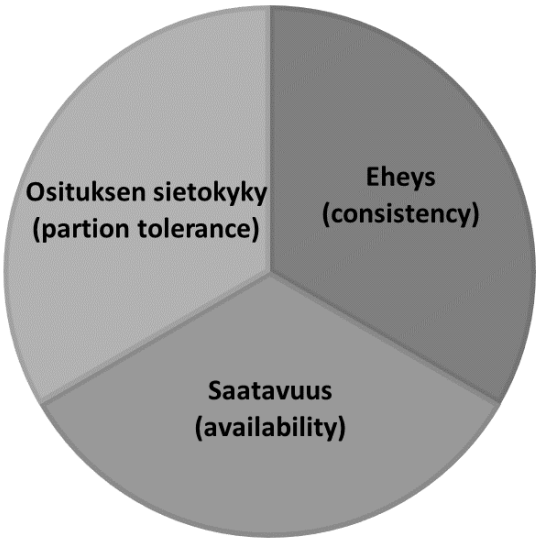


Kuva 9: Esimerkki horisontaalisesti skaalatusta tietokannasta käyttäen hajautusalgoritmia.

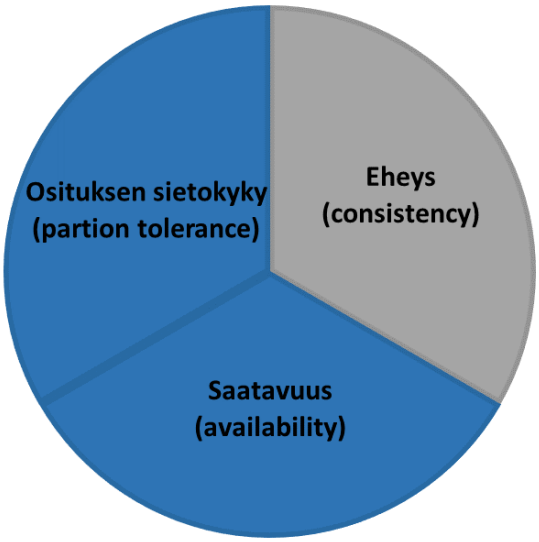
4.2 CAP-teoreema

CAP-teoreema on lyhenne sanoista eheys (consistency), saatavuus (availability) ja osituksen sietokyky (partion tolerance). CAP-teoreeman mukaan hajautettu järjestelmä voi toteuttaa vain kaksi ominaisuutta kolmesta. [17, p. 23] Tarkoittaen sitä, että jos järjestelmän halutaan olevan eheä ja aina saatavilla niin osituksen sietokykyä ei voida toteuttaa. (Kuva 12) Jos

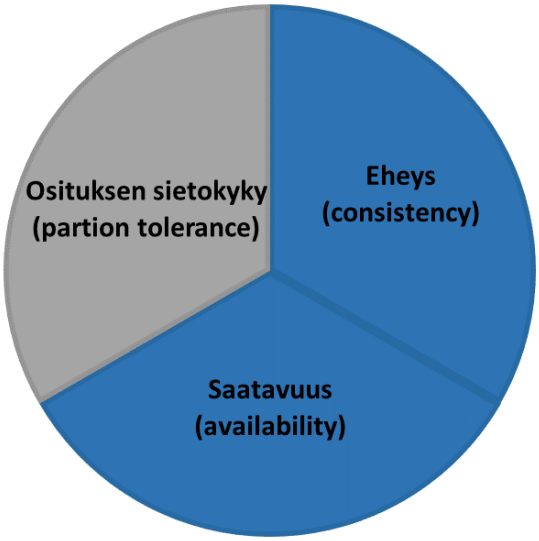
järjestelmän halutaan olevan saatavilla ja siihen halutaan lisätä osituksen sietokyky, eheyttä ei voida toteuttaa. (Kuva 11) Viimeisenä vaihtoehtona on se, että jos järjestelemän täytyy olla eheä ja sen täytyy sietää osittautumista niin saatavuutta ei voida taata jokaisessa (Kuva 13)



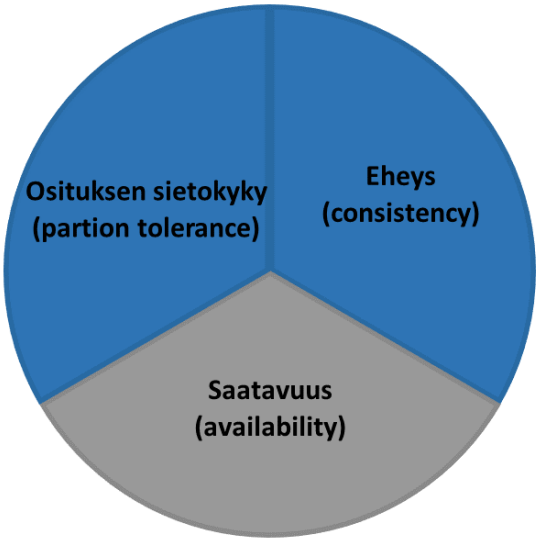
Kuva 10: CAP-teoreeman ominaisuudet



Kuva 11: CAP-teoreeman ominaisuudet, joista on valittu osituksen sieto-kyky ja saatavuus



Kuva 12: CAP -teoreeman ominaisuudet, joista on valittu eheys ja saata-vuus



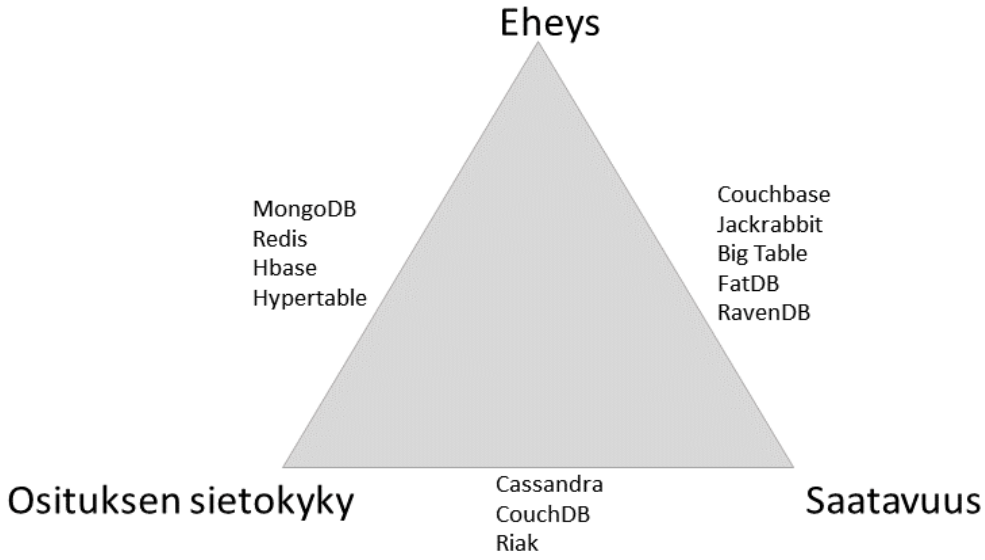
Kuva 13: CAP-teoreeman ominaisuudet, joista on valittu osituksen sietokyky ja eheys

Eheyden (consistency) tarkoituksena on, että hajautetun järjestelmän jokainen solmu olisi eheässä tilassa yhtä aikaa. [18, p. 1] Käytännössä tarkoittaen sitä, että jokaisen operaation jälkeen voidaan kysyä miltä tahansa solmulta vastausta johonkin kyselyyn ja jokainen näistä solmuista palauttaisi saman vastauksen.

Saatavuuden (availability) tarkoituksena on, että hajautettu järjestelmä on aina saatavilla. Yksi tai useampi solmu voi kaatua virheen seurauksena, mutta saatavuuden määritelmän mukaan toinen solmu pystyy korvaamaan tämän. [18, p. 1]

Osituksen sietokyky tarkoittaa sitä, että verkkovirheen sattuessa, missä solmut eivät voi keskustella toistensa kanssa, järjestelmä pysyy silti toiminnassa. Tässä tapauksessa järjestelmä hajoaa ryhmiksi, joista jokainen jatkaa toimintaa itsenäisti. [18, p. 1]

Seuraavassa kuvassa on eriytelty muutamia tunnettuja NoSQL-tietokantoja niiden CAP-teoreeman toteuttavien ominaisuuksien mukaan.



Kuva 14: Esimerkkejä CAP-teoreeman mukaisista NoSQL-tietokannoista [19]

4.3 BASE–oikeellisuusmalli

BASE–oikeellisuusmalli muodostuu sanoista (Basically Available, Soft State, Eventual Consistency). Se on tyypillinen malli, mitä käytetään hajautetuissa tietokannoissa ja on vastakohta luvussa 3.3 mainitulle ACID-ominaisuuksille. Käytännössä ACID-ominaisuudet pakottavat transaktioiden ristiriidattomuuteen eli tietokanta jää jokaisen transaktion jälkeen eheään tilaan. BASE-oikeellisuusmalli tarjoaa mahdollisuuden rikkoa eheyden rajoja. Tarkoittaen sitä, että tietokanta voi olla hetkellisesti ei-eheässä tilassa kuitenkin niin, että se saavuttaa joskus eheän tilan. Tämä taas tarkoittaa sitä, että BASE-oikeellisuusmalli mahdollistaa paremman skaalautuvuuden kuin tietokannat, jotka luottavat ACID-ominaisuuksiin. [10, p. 51] BASE-oikeellisuusmalli muodostuu siis kolmesta eri ominaisuudesta, joita ovat:

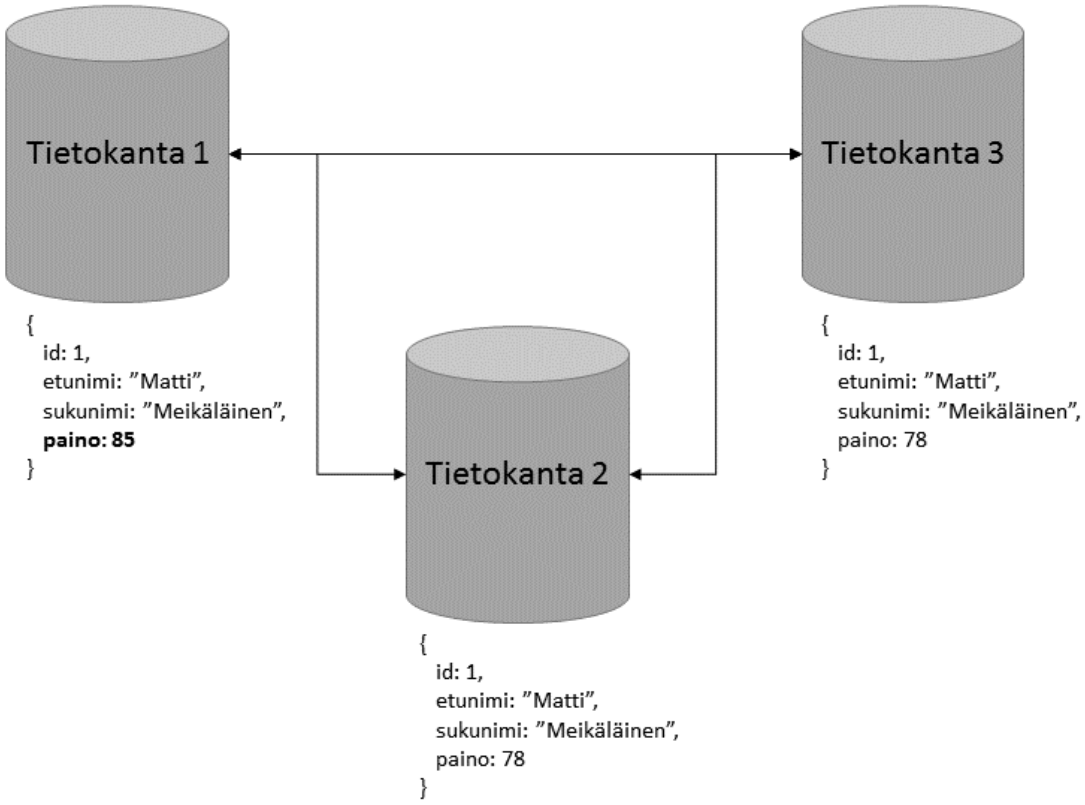
- periaatteessa käytettävissä / saatavilla,
- ei aina oikeellinen
- lopulta oikeellinen.

Edellä mainituilla ominaisuuksilla pyritään siihen, että tietokannan suorituskyky ja saavutettavuus paranee entisestään.

Pariaatteessa käytettävissä (Basically Available) -ominaisuus saavutetaan tukemalla osittaisia virhetilanteita ilman, että koko järjestelmä kaatuu. [10, p. 51] Tarkoittaen sitä, että jos tietokanta on hajautettu useammalle tietokantapalvelimelle ja yksi näistä palvelimista kaatuu, niin kaatuminen ei kaada koko järjestelmää. Palvelimen kaatumisen vaikutukset saattavat näkyä pienelle joukolle loppukäyttäjiä, muuta kokonaiskuvaa katsomalla tietokanta on periaatteessa käytettävissä.

Ei aina oikeellinen (Soft State) -ominaisuus tarkoittaa sitä, että hajautetun tietokannan eri solmut eivät ole välttämättä samassa tilassa. [10, p. 55] Esimerkiksi, jos tietokanta on hajautettu useammalle palvelimelle ja tietoja päivitetään, niin transaktion jälkeen kaikki tietokannat eivät välttämättä ole heti samassa tilassa. Tätä voidaan havainnollistaa seuraavan esimerkkikuvan avulla. (Kuva 15) Käytetään hyväksi HENKILO -relaation (Kuva 1) tietuetta henkilön tiedoista ja sen painosta. Matti Meikäläinen haluaa päivittää oman painonsa uudelleen järjestelmään, koska hän on hieman kasvanut. Päivitysoperaatio suoritetaan loppuun ja

Matin tietoihin päivittyy uusi paino. Tiedot kuitenkin sijaitsevat hajautetun tietokannan tietokannassa 1, kun taas samaan aikaan tietokannat 2 ja 3 sisältävät vielä vanhentunutta tietoa.



Kuva 15: BASE–oikeellisuusmallin ei aina oikeellinen (Soft State) -ominaisuus

Edellinen esimerkki kuvastaa siis tilannetta, jossa hajautetun järjestelmän tietokannat voivat erota hetkittäin toisistaan. Viimeinen BASE-ominaisuus kuitenkin takaa, että jossain vaiheessa tietokannat päätyvät oikeelliseen tilaan. Eli edellisessä esimerkissä muutettu paino päivittyy kaikkiin tietokantoihin.

4.4 Avain-arvo -tietokannat

Avain-arvo -tietokannat tallentavat tiedon nimensä mukaisesti avain-arvo -pareina. [20] Tarkoittaen sitä, että tallennettava data koostuu avaimesta ja tallennettavasta arvosta. Avain-arvo -tietokannat eivät ota kantaa tallennettavan tiedon muotoon vaan se voi olla hyvinkin kompleksinen. [20] Samalla se tekee tarpeettomaksi relaatiotietokannasta tutun taulurakenteen määrittämisen. Tyypillisesti tallennettava data on jokin ohjelmointikielen primääreistä-tietotyypeistä kuten merkkijono (string), kokonaisluku (integer), taulukko, (array). Tai erikoisemmissa tapauksissa se voi olla olio (object) tai hyvinkin monimutkainen JSON- tai XML -dokumentti. [21] Seuraavassa taulukossa on yksinkertainen esimerkki, millainen avain-arvo -tietokantojen tallennettava sisältö voisi olla:

Avain	Arvo
”matti_meikalainen_paino”	78
”matti_meikalainen_puhelinnumero”	”0407364910”

”matti_meikalainen”	{
	paino: 78,
	puhelinnumero: ”0407364910”
	}

Kuva 16: Henkilön tietoja tallennettuna avain-arvo -tietokantaan.

Seuraavassa taulukossa on esitetty muutamia tunnettuja avain-arvo -tietokantoja:

Nimi	Kehittäjä	CAP-teoreeman ominaisuudet
Riak	Basho Technologies	• Saatavuus • Osituksen sietokyky
DynamoDB	Amazon	• Saatavuus • Osituksen sietokyky
Redis	Salvatore Sanfilippo	• Eheys • Osituksen sietokyky

Taulukko 3: Esimerkkejä avain-arvo –tietokannoista.

4.5 Dokumenttitietokannat

Dokumenttitietokannoissa data tallennetaan dokumentteina. Dokumentin sisältö voi olla hyvin heterogeeninen verrattuna homogeenisiin relaatiotietokannan tauluihin. [20] Tämä mahdollistaa joustavan tiedon tallentamisen ja lukemisen, koska tyypillisesti dokumentin sisältö tallennetaan JSON –tyylisenä rakenteena. Tämä on hyvin intuitiivinen tapa tallentaa tietoa, koska esimerkiksi web-kehityksessä JSON –formaatti on hyvin yleinen tapa lähettää ja vastaan ottaa tietoa. Käytännössä siis jokaista dokumenttia voidaan verrata olio-ohjelmoinnista tuttuun olioon eli objektiin. Dokumentit tallennetaan yleensä kokoelmiin (collections), jotka vastaavat relaatiotietokantojen tauluja. [22]

Dokumenttitietokantojen dokumentit sisältävät 1-N –kappaletta erillaisia kenttiä, joita voidaan verrata SQL –tietotyyppeihin. Yleisesti tietotyyppejä ovat mm. merkkijono (string), numero (integer), päivämäärä (date) ja taulukko (array). Verrattuna relaatiotietokantoihin dokumenttitietokannat eivät tarvitse erillistä perusavainta viitatakseen toisessa taulussa olevaan tietoon vaan kyseinen tieto voidaan tallentaa dokumentille alidokumenttina. Käytännössä tämä voidaan ajatella niin, että olion sisällä on toinen olio. [22]

Koska dokumentit voivat sisältää hyvin erilaisia kenttiä saman kokoelman sisällä, niin tämä voi asettaa kehittäjille haasteita. Esimerkiksi henkilö –kokoelman kentät eivät välttämättä sisällä samoja kenttiä eri henkilöiden välillä. (Kuva 17) Käytännössä tämä voi johtaa siihen, että näytettäessä tietoa kehittäjä joutuu rakentamaan tarkastukset kentän olemassa olemiselle, koska dokumenttitietokanta ei pakota dokumenttejä tiettyyn rakenteeseen.


```
[
  {
    "id":1,
    "etunimi":"Matti",
    "sukunimi":"Meikäläinen",
    "paino":78
  },
  {
    "id":2,
    "etunimi":"Maija",
    "sukunimi":"Meikäläinen",
    "pituus":168
  }
]
```


Kuva 17: Esimerkki yksinkertaisista dokumenteistä.

Tässä tutkielmassa verrataan yhtä tämän tietokantatyypin tietokantaa relaatiotietokantaan ja tutkitaan kuinka niiden suorituskyky vertautuu, kun tietokannasta haetaan samanlaista dataa tietokantojen omilla kyselyillä.

Seuraavassa taulukossa on esitetty muutamia tunnettuja dokumenttitietokantoja ja niiden CAP-teoreeman mukaisia ominaisuuksia: [3]

Nimi	Kehittäjä	CAP-teoreeman ominaisuudet
MongoDB	10gen	• Eheys • Osituksen sietokyky
CouchDB	Apache	• Saatavuus • Osituksen sietokyky
SimpleDB	Amazon	• Saatavuus • Osituksen sietokyky

Taulukko 4: Esimerkkejä dokumenttitietokannoista.

4.6 Saraketietokannat

Relaatiotietokannat ovat tyypillisesti arkkitehtuuriltaan rivitietokantoja eli tietueen tiedot tallennetaan peräkkäin levylle. Tämä mahdollistaa rivitietokantojen nopean kirjoitusnopeuden eli ne ovat optimoituja kirjoitusnopeuteen (write-optimized). Järjestelmissä, missä on paljon dataa ja suoritetaan isoja kyselyitä tietokantahallintajärjestelmän tulisi olla optimoitu lukunopeuteen (read-optimized). Tästä syystä ollaan kehitetty rivitietokannoista hieman poikkeavia saraketietokantoja. Saraketietokannoissa jokaisen sarakkeen tiedot tallennetaan peräkkäin levylle järjestyksessä. [23, p. 1] Tästä syystä saraketietokannat eivät vaadi erillistä perusavainta tietojen yhteen liittämiselle hakuvaiheessa. Käytännössä tämä tehdään niin, että tiedot säilötään aina samaan järjestyksen jokaisessa sarakkeessa. Esimerkiksi alapuolella olevassa kuvassa (Kuva 18) Tepon tiedot löytyvät indeksillä 4 jokaisesta sarakkeesta. Lisäksi saraketietokannat käyttävät ainoastaan hauissa määriteltyjä sarakkeita tiedon hakemiseen, joten levyltä luettavan datan määrä on pienempi kuin esimerkiksi rivitietokannoissa.. Huonona puolena saraketietokannoissa on se, että kirjoitus- ja päivitysoperaatiot

kestävät kauemmin kuin rivitietokannoissa. Tämä johtuu yksinomaan siitä, että tieto joudutaan lisäämään useampaan sarakkeeseen. [24]

Seuraavassa kuvassa (Kuva 18) on esitetty perinteinen rivitietokannan tyyli ilmaista henkilön tietoja ja sen alapuolella on samat tiedot saraketietokannassa. Kuvassa on hyvä huomioida se, että erillistä perusavainta ei ole missään sarakkeessa vaan tietojen järjestyksellä on väliä.

HENKILO				
ID	ETUNIMI	SUKUNIMI	PUHELINNUMERO	PAINO
1	Matti	Meikäläinen	0407364910	78
3	Maija	Meikäläinen	0508402831	55
4	Suvi	Suomalainen	0448712398	49
5	Teppo	Turunen	0401238932	89

ETUNIMI
Matti
Maija
Suvi
Teppo

SUKUNIMI
Meikäläinen
Meikäläinen
Suomalainen
Turunen

PUHELINNUMERO
0407364910
0448712398
0401238932
Teppo

PAINO
78
55
49
89

Kuva 18: HENKILO –taulu esitettynä saraketietokannassa.

Seuraavassa taulukossa on esitelty muutamia saraketietokantoja ja niiden CAP-teoreeman mukaisia ominaisuuksia

Nimi	Kehittäjä	CAP-teoreeman ominaisuudet
Casandra	Apache	• Saatavuus • Osituksen sietokyky
HBase	Apache	• Eheys • Osituksen sietokyky
BigTable	Google	• Eheys • Osituksen sietokyky

Taulukko 5: Esimerkkejä saraketietokannoista

4.7 Verkkotietokannat

Vekkotietokannat (Graph Database) käyttävät verkkomaisia rakenteita tiedon mallintamiseen. Verkkotietokantojen ideana on yhdistää solmuja toisiinsa erilaisten suhteiden avulla. Jokainen verkon solmu voi pitää sisällään tarvittavan määrän tietoa avain-arvo-pareina. Lisäksi solmujen väliset suhteet ovat merkitseviä eri solmujen välillä. Näin ollen jokaisella suhteella on suunta, tyyppi, alku- ja loppusolmu. Solmujen välisille suhteille voi myös tallentaa erinäköistä tietoa. Verkkotietokannoilla on yksi sääntö eli tietokanta ei saa sisältää rikkinäisiä suhteita solmujen välillä. Tästä johtuen solmua poistaessa on poistettava kaikki mahdolliset riippuvuudet kyseiseen solmuun. [25]

Seuraavassa taulukossa on esitelty muutamia verkkotietokantoja ja niiden CAP-teoreeman mukaisia ominaisuuksia:

Nimi	Kehittäjä	CAP-teoreeman ominaisuudet
Neo4j	Neo Technology, Inc	• Saatavuus • Eheys
GraphBase	FactNexus	Ei ole varmuutta
Titan	Datastax	• Eheys • Osituksen sietokyky tai • Saatavuus • Osituksen sietokyky

5 Tutkimuksen esittely

Aikaisempien kappaleiden on tarkoitus toimia nopeana ja ytimekkäänä johdatuksena relaatio- ja epärelaatiotietokantoihin. Tästä kappaleesta eteenpäin odotetaan, että lukijalla on hallussa jonkin asteinen ymmärrys molemmista tietokantatyypeistä.

Koska eri tietokantahallintajärjestelmiä on useita, joudun rajaamaan tutkimuksen kahteen tietokantahallintajärjestelmään. Valinnan olen tehnyt sen mukaan, mistä itselläni on henkilökohtaisesti eniten kokemusta. Tässä tutkimuksessa tulen vertailemaan kahta eri tietokantahallintajärjestelmää; MySQL:ää ja MongoDB:tä. Tutkimus on hyvin suorituskykypainotteinen, joita samalla täydennän yleisten näkemysten mukaisesti.

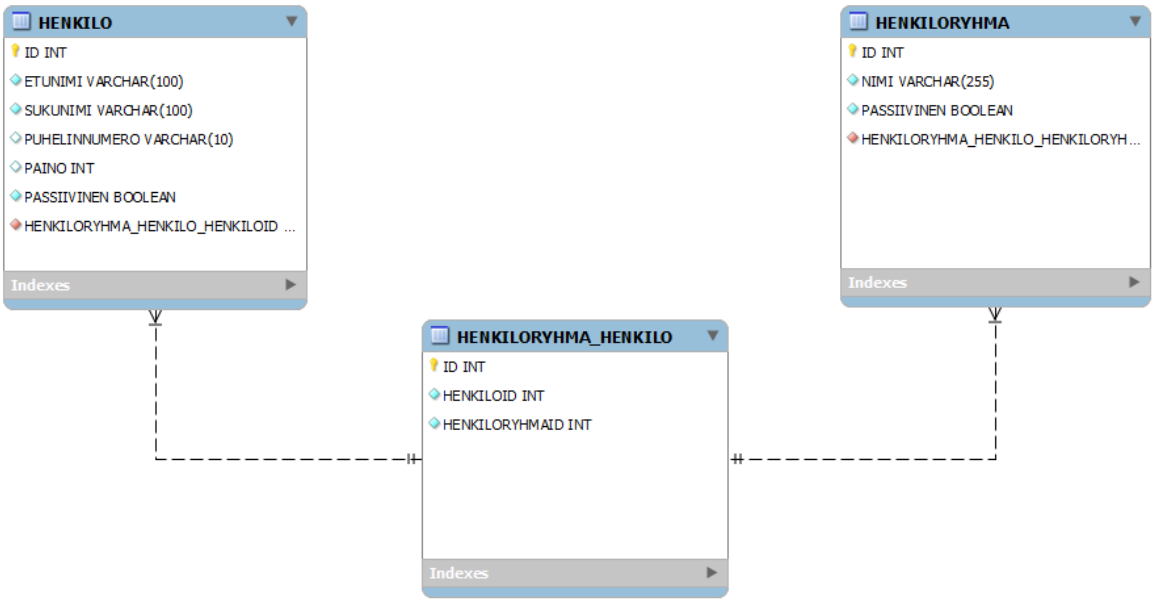
Suorituskykytestaus tehdään virtuaalipalvelimella (VPS), joka sijaitsee Microsoftin Azure -pilvipalvelussa. [26] Virtuaalipalvelimen tekniset tiedot ovat seuraavanlaiset:

- Keskusyksikkö (CPU): AMD Opteron(tm) Processor 4171 HE 2.1 GHz
- Keskusmuisti (RAM): 2 GB
- Kovalevytila: 20 GB
- Käyttöjärjestelmä: Ubuntu 14.10 (GNU/Linux 3.16.0-30-generic x86_64)

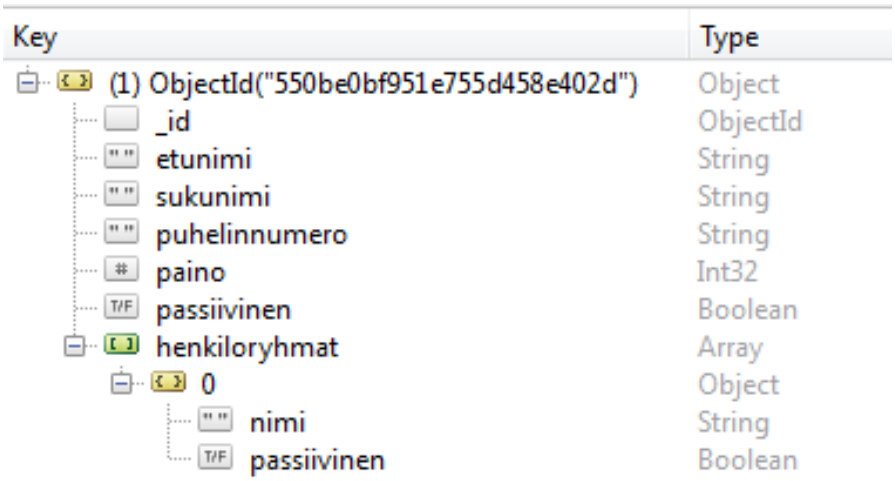
Tutkimuksessa käytetään liitteistä löytyvää esimerkkikoodia, jolla suorituskykytestaukset suoritetaan. Jotta välttyisimme ohjelmakoodin aiheuttamilta viiveiltä, tulee kyselyn kesto kysyä tietokantahallintajärjestelmästä. Lisäksi sama kysely suoritetaan useamman kerran peräkkäin ja näistä tuloksista lasketaan keskiarvo, jotta hetkelliset kuormituspiikit tai muut ulkopuolelta johtuvat viiveet saadaan minimoitua tuloksesta.

Tässä tutkimuksessa on otettu pieni reaalimaailman esimerkki ja sovellettu sitä molempiin tietokantatyyppeihin hyvien käytänteiden mukaisesti. Esimerkkinä toimii siis seuraavanlainen skenaario: Järjestelmään voi lisätä henkilöitä, joilla on henkilölle tyypillisiä tietoja. Henkilön tietoihin kuuluu seuraavat: etunimi, sukunimi, puhelinnumero, paino ja passiivisuustieto. Tämän lisäksi jokainen järjestelmän henkilö voi kuulua tai olla kuulumatta yhteen tai useampaan henkilöryhmään. Henkilöryhmä sisältää nimen ja passiivisuustiedon. Näiden

pohjalta on laadittu ER-kaavio relaatiotietokannan tauluista (Kuva 19) ja vapaamuotoisempi kuvaus epärelaatiokannan kokoelmasta (Kuva 20).



Kuva 19: ER-kaavio tutkimuksessa käytetystä relaatiotietokannan tauluista.



Kuva 20: Tutkimuksessa käytetyn epärelaatiokannan kokoelma.

Edellisten tietojen pohjalta tutkimus toteutetaan suorittamalla esimerkki kyselyitä molempiin tietokantoihin niin, että kumpaankin on ajettu ennalta määritelty määrä henkilön ja henkilöryhmien sekä niiden liitostietoja. Molemmat tietokannat sisältävät käytännössä saman informaation tietokannalle tyypillisessä formaatissa. Kyselyt ovat sanallisesti seuraavassa muodossa: (Taulukko 6)

T1	Haetaan lukumäärä henkilöistä, jotka ovat passiivisia ja kuuluvat ainakin yhteen passiiviseen henkilöryhmään
T2	Haetaan kaikki henkilöt, joiden paino on enemmän tai yhtä suuri kuin 80 kg
T3	Haetaan aktiivisten henkilöiden määrä ryhmiteltynä painoluokkaan yhden kilogramman tarkkuudella.

Taulukko 6: Testeissä käytettävät kyselyt sanallisessa muodossa.

Suorituskykytesteissä kyselyt tehdään indeksoimattomaan ja indeksoituun tietokantaan kummallekkin tietokantahallintajärjestelmälle. Tämän lisäksi verrataan molemman tuotteen suorituskyvyn parantumista indeksien lisäämisen jälkeen. Yksityiskohtaisemmat analyysit tuloksista löytyvät kappaleista 6.1-6.12. Indeksoimattomat testit ovat lisätty tutkielmaan sen takia, että kaikki tietokantaan menevät kyselyt eivät välttämättä voi käyttää indeksejä hyväkseen vaan ne joutuvat käymään jokaisen tietokannan rivin läpi. Esimerkiksi, jos halutaan hakea kaikki henkilöt joiden etunimi sisältää kirjainyhdistelmän ”att”. Tässä tapauksessa tietokannasta hakeminen joutuu käymään kaikki rivit etunimen osalta läpi ja tekemään vertailun tuloksen löytämiseksi.

MySQL:ssä tämä haettaisiin esimerkiksi seuraavalla kyselyllä:

```
SELECT *
FROM `HENKILO` AS `h`
WHERE  `h`.`ETUNIMI` LIKE “%att%”;
```

Samanlainen kysely voidaan myös suorittaa MongoDB:ssä seuraavalla komennolla:

```
db.henkilo.find(
    {
        etunimi: {$regex: ".*att.*"}
    }
);
```

5.1 Suorituskykytesti 1 – Kyselyt

Seuraavaksi esitellään ensimmäinen suorituskykytesti ilmaistuna SQL- ja JavaScript -kyselynä:

SQL:

```
SELECT COUNT(*) AS `maara`
FROM `HENKILO` AS `h`
JOIN `HENKILORYHMA_HENKILO` AS `hh` ON `h`.`ID` = `hh`.`HENKILOID`
JOIN `HENKILORYHMA` AS `hr` ON `hh`.`HENKILORYHMAID` = `hr`.`ID`
WHERE  `h`.`PASSIIVINEN` = 1 AND `hr`.`PASSIIVINEN` = 1;
```

Kyselyssä käytetään SQL:ssä olevaa COUNT()-funktiota. Sen tarkoituksena on palauttaa kyselyn tuottamien rivien lukumäärä ottamatta kantaa siihen sisältävätkö rivit NULL arvoja vai ei. [27] Seuraavaksi tarvitsemme liitoksen HENKILORYHMA_HENKILO -tauluun, joka pitää sisällään tiedon siitä, mihin henkilöryhmään kukin henkilö kuuluu. Tämän voimme tehdä käyttämällä JOIN-operaatiota. [28] Edellisen liitoksen jälkeen meillä on tieto, mikä henkilö liittyy mihinkin henkilöryhmään. Kuitenkin HENKILORYHMA_HENKILO -taulu sisältää vain viitteen HENKILORYHMA -tauluun, joten joudumme liittämään vielä yhden taulun, jotta saamme henkilöryhmien passiivisuuden selville. Tämä tapahtuu käyttämällä samaa JOIN-operaatiota. Tämän testin on tarkoitus tutkia, miten nopeasti MySQL suoriutuu kyselyistä, joissa joudutaan hakemaan tietoja useammasta taulusta.

JavaScript (MongoDB):

```
db.henkilo.count(
    {
        passiivinen: true,
        henkiloryhmat: {
            $elemMatch: {
                passiivinen: true
            }
        }
    }
);
```

MongoDB:stä hakiessa, käytämme JavaScriptiä kyselykielenä. [29] Kyselyn tuottamien rivien lukumäärä voidaan MongoDB:ssä kysyä suoraan tietokannan kokoelmalta käyttäen count()-funktiota. Tämän jälkeen määritellään funktion sisään kyselyn ehdot JavaScript oliona. [29] Edellisestä esimerkistä poiketen, liitoksien tekeminen toisiin kokoelmiin on turhaa, koska tietoa voidaan säilyttää henkilön tietojen alla alidokumenttina. Koska henkilöryhmät -alidokumentti voi sisältää 0–n kappaletta henkilöryhmiä, joudumme käyttämään kyselyssä $elemMatch -operaattoria. [30] $elemMatch -operaatio palauttaa dokumentin, jos yksikin listassa olevista alkioista toteuttaa annetun ehdon. [30] Eli käytännössä haemme henkilo -kokoelmasta kaikki dokumentit, mitkä ovat passiivisia (henkilö) ja yksikin henkilöryhmä on passiivinen. Tämän testin tarkoituksena on tehdä vastaavanlainen kysely MongoDB:lle.

5.2 Suorituskykytesti 2 – Kyselyt

Seuraavaksi toinen suorituskykytesti on ilmaistuna SQL- ja JavaScript -kyselynä:

SQL:

```
SELECT *
FROM   `HENKILO` AS `h`
WHERE  `h`.`PAINO` >= 80
```

Kysely on hyvin yksinkertainen SQL-kysely, jossa haetaan HENKILO -taulusta kaikki henkilöt, joiden paino on enemmän tai yhtä kuin 80. Testin tarkoituksena on katsoa, miten nopeasti MySQL hakee tietoja yksinkertaisella hakukriteerillä.

JavaScript (MongoDB):

```
db.henkilo.find(
    {
        paino: {
            $gte: 80
        }
    }
);
```

Vastaavasti sama kysely voidaan esittää JavaScriptillä MongoDB:n vaatimassa formaatissa. Nyt vertailuoperaattorina joudutaan käyttämään $gte -operaattoria, joka tarkoittaa, että haemme isompaa tai yhtä suurta arvoa kuin annettu arvo. [31]

5.3 Suorituskykytesti 3 – Kyselyt

Seuraavaksi kolmas suorituskykytesti on ilmaistuna SQL- ja JavaScript -kyselynä:

SQL:

```
SELECT COUNT(`h`.`PAINO`) AS `maara`
FROM   `HENKILO` AS `h`
WHERE  `h`.`PASSIIVINEN` = 0
GROUP  BY `h`.`PAINO`
```

Kyselyn tarkoituksena on testata, miten hyvin MySQL suorittaa kyselyn, jossa on käytetty ryhmittelyä painon suhteen. GROUP BY -funktion tarkoituksena on ryhmitellä saatu tulosjoukko painoittain. [32] Käytännössä kysely palauttaa henkilöiden määrän painoryhmittäin.

JavaScript (MongoDB):

```
db.users.aggregate(
    [
        {
            $match: {
                passiivinen: false
            }
        },
        {
            $group: {
                _id: "$paino",
                count: {
                    $sum: 1
                }
            }
        }
    ]
);
```

MongoDB:ssä vastaavanlaisen kyselyn suorittaminen vaatii aggregate-funktion käyttämistä. Aggregate-operaatiot ovat operaatioita, jotka prosessoivat dataa ja palauttaa lasketut arvot. [33] Aggregate-funktiolle voidaan välittää kaksi parametria, joista ensimmäiseen määritellään kyselyehdot ja toiseen ryhmittely ehdot. Käytännössä siis $match -määritys vastaa normaalia kyselyehtoa ja $group -määritys ryhmittelee tulosjoukon annetuilla ehdoilla. [34]

5.4 Tietokannan alustus ja indeksien luominen

Tässä kappaleessa käydään läpi, millä tavalla ja miten tietokanta on alustettu suorituskykytestejä varten. Käytännössä siis kappale esittelee luontilauseet, millä MySQL:n taulut ja indeksit luodaan sekä kerrotaan, miten tämä hoituu MongoDB:ssä.

MySQL:ssä HENKILO, HENKILORYHMA_HENKILO ja HENKILORYHMA taulut alustetaan testejä varten seuraavalla tavalla:

```
CREATE TABLE HENKILO (
     ID             INT(11)              NOT NULL AUTO_INCREMENT,
     ETUNIMI        VARCHAR(100)         NOT NULL,
     SUKUNIMI       VARCHAR(100)         NOT NULL,
     PUHELINNUMERO  VARCHAR(10)          NOT NULL,
     PAINO          INT(11)              NOT NULL,
     PASSIIVINEN    INT(1)               NOT NULL DEFAULT 0,
     PRIMARY KEY (ID)
) ENGINE=MYISAM;

CREATE TABLE HENKILORYHMA_HENKILO (
     ID              INT(11)             NOT NULL AUTO_INCREMENT,
     HENKILOID       INT(11)             NOT NULL,
     HENKILORYHMAID  INT(11)             NOT NULL,
     PRIMARY KEY (ID)
) ENGINE=MYISAM;

CREATE TABLE HENKILORYHMA (
     ID           INT(11)                NOT NULL AUTO_INCREMENT,
     NIMI         VARCHAR(255)           NOT NULL,
     PASSIIVINEN  INT(1)                 NOT NULL DEFAULT 0,
     PRIMARY KEY (ID)
) ENGINE=MYISAM;
```

Taulujen alustuksien lisäksi indeksoiduissa testeissä ajetaan indeksit seuraavalla tavalla:

```
CREATE INDEX PAINO ON HENKILO (PAINO)

CREATE INDEX PASSIIVINEN ON HENKILO (PASSIIVINEN)

CREATE INDEX HENKILOID ON HENKILORYHMA_HENKILO (HENKILOID)

CREATE INDEX HENKILORYHMAID ON HENKILORYHMA_HENKILO (HENKILORYHMAID)

CREATE INDEX PASSIVINEN ON HENKILORYHMA (PASSIIVINEN)
```

MongoDB:ssä erillistä kokoelman luomista ei tarvitse tehdä, koska niille ei tarvitse määrittää erillistä skeemaa, kuten MySQL:ssä vaan riittä, että luomme kokoelman nimellä ja lisäämme sinne tietueita. [35] Indeksoiduissa testeissä joudumme kuitenkin lisäämään indeksit seuraavalla tavalla: [36]

```
db.users.createIndex({ ”paino”: 1 });
db.users.createIndex({ ”passiivinen”: 1 });
db.users.createIndex({ ”henkiloryhmat.passiivinen”: 1 });
```

Käytännössä edelliset komennot luovat indeksit seuraaville kentille ja viimeinen numero (tässä tapauksessa 1) määrittelee onko indeksi laskeva vai nouseva. [36]

6 Tulokset ja analysointi

Seuraavissa kappaleissa esitellyt suorituskykytestien tulokset ovat mallinnettu kuvaajiin. Kuvaajissa y-akselilla on aika sekunteina ja x-akselilla on tietueiden määrä. Käytännössä tietueiden määrä kuvastaa sitä kuinka monta henkilöä tietokantaan on tallennettu. MongoDB:ssä tämä on suoraan verrannollinen dokumenttien määrään, mutta MySQL:ssä tiedot henkilöiden henkilöryhmistä ovat erillisissä liitostauluissa, joten todellinen tietueiden määrä on suurempi. Testeissä kuitenkin tärkeintä on se, että sillä pyritään mallintamaan samaa tutkittavaa asiaa mahdollisimman tasapuolisesti.

Käytännössä kuvaajia voidaan tulkita niin, että mitä korkeammalla kuvaajan pisteet ovat y-akselilla, sitä hitaammin se on suoriutunut suorituskykytestissä. Eli toisin päin sanottuna alempi kuvaaja on suoriutunut nopeammin. Taulukoissa on myös lisätty vertailua varten ”Erotus %” –sarake, joka ilmoittaa kuinka monta kertaa suurempi tai pienempi arvo ensimmäisen sarakkeen tulos on verrattuna toiseen sarakkeeseen. Tämä saadaan laskettua seuraavalla kaavalla:

$$P = 100 \cdot \frac{(b - a)}{a}$$ [1]

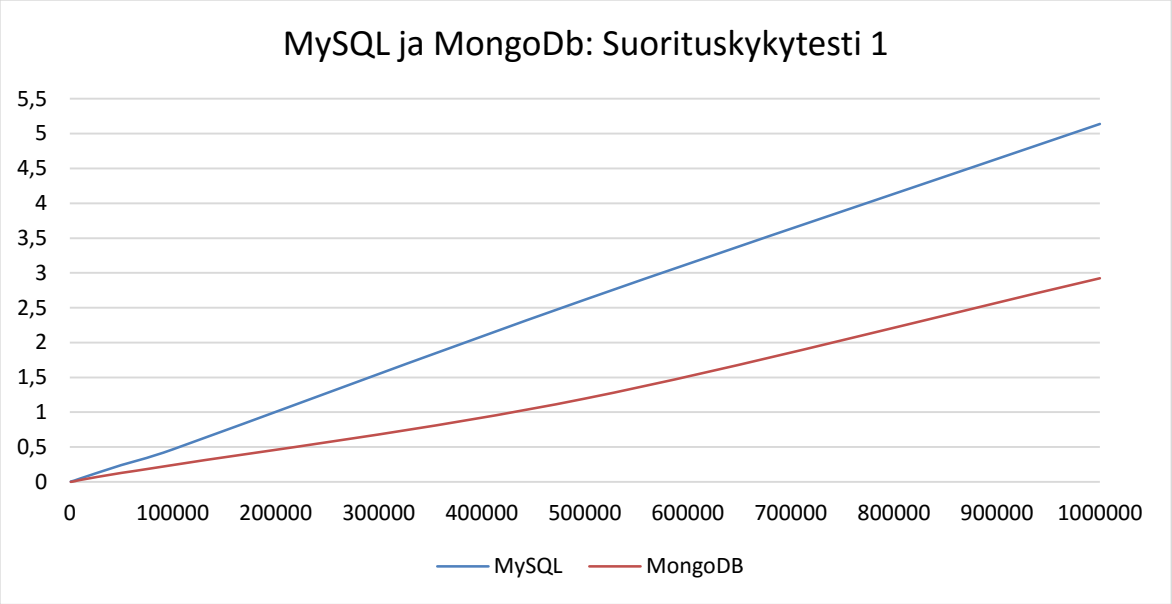
Tuloksesta voimme päätellä kuinka monta prosenttia nopeampi tai hitaampi suorituksellisesti toinen tietokanta on verrattuna toiseen.

6.1 Suorituskykytesti 1 – tulokset (indeksoimaton)

Ensimmäisessä suorituskykytestissä ilman indeksejä saatiin seuraavat tulokset. Kuten kuvaajasta huomaamme niin MongoDB:n suorituskyky on nopeampi kuin MySQL:n. MySQL:ssä saman tuloksen saamiseen joudutaan yhdistelemään tietoja useammasta eri tauluista, mutta MongoDB:ssä nämä tiedot löytyvät suoraan dokumentin sisältä. Eli erillisiä

[1] a = MongoDB: n tulos ja b = MySQL:n tulos.

liitoslauseita ei tässä tapauksessa tarvitse. Huomattavaa tässä on se, että MongoDB on testausmäärässä hitaimmillaankin 64% nopeampi kuin MySQL. (Kuva 21) (Taulukko 7)



Kuva 21: Ensimmäisen suorituskykytestin tulokset kuvaajassa (ilman indeksejä).

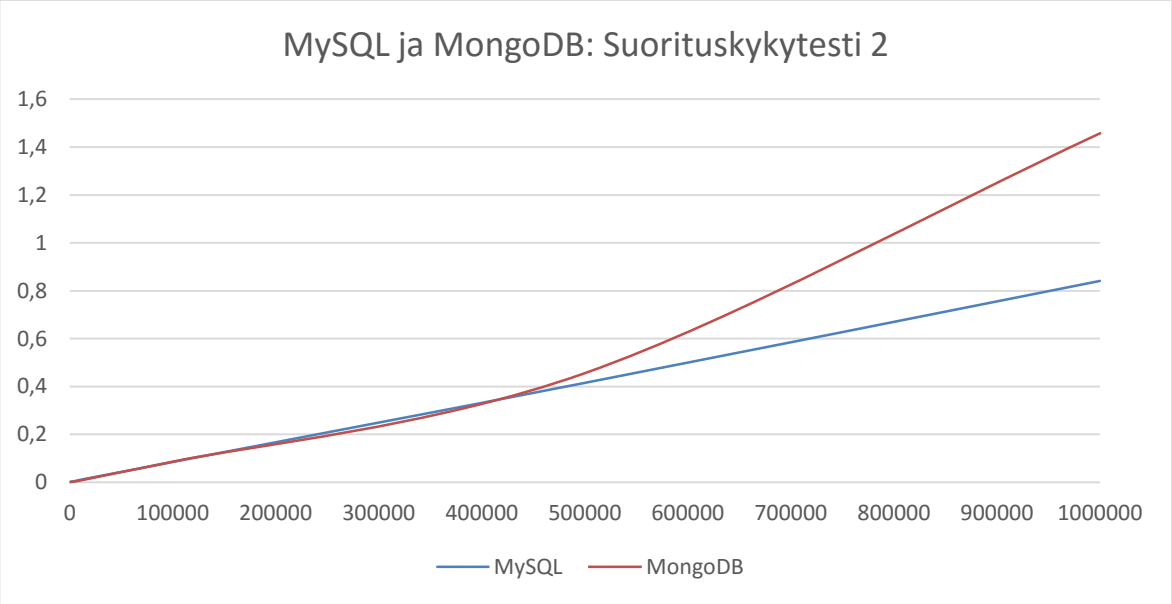
Tietueiden määrä	MySQL (sekuntia)	MongoDB (sekuntia)	Erotus %
1000	0,0043941	0,002	120
5000	0,0207184	0,0114	82
10000	0,0470385	0,0287	64
50000	0,240388	0,1303	84
100000	0,4657713	0,2431	92
500000	2,6142552	1,1962	119
1000000	5,1370533	2,9225	76

Taulukko 7: Ensimmäisen suorituskykytestin tulokset taulukossa (ilman indeksejä).

6.2 Suorituskykytesti 2 – tulokset (indeksoimaton)

Toisessa suorituskykytestissä ilman indeksejä saatiin seuraavat tulokset. Tässä testissä huomaamme selkeän eron, joka kasvaa kun alkioiden määrä tietokannassa lisääntyy. Miljoonan alkion kohdalla MySQL on 42% nopeampi kuin MongoDB, vaikka aluksi näyttäisi siltä, että

MongoDB tulee suoriutumaan nopeammin. MongoDB:n hitaus aiheutuu $elemMatch –operaattorin toiminnasta, joka joutuu käymään jokaisen ”henkiloryhmat” –taulukon sisällä olevat alkiot läpi ja tekemään vertailuja annetun ehdon mukaisesti. [30] (**Error! Reference source not found.**) (Taulukko 8)



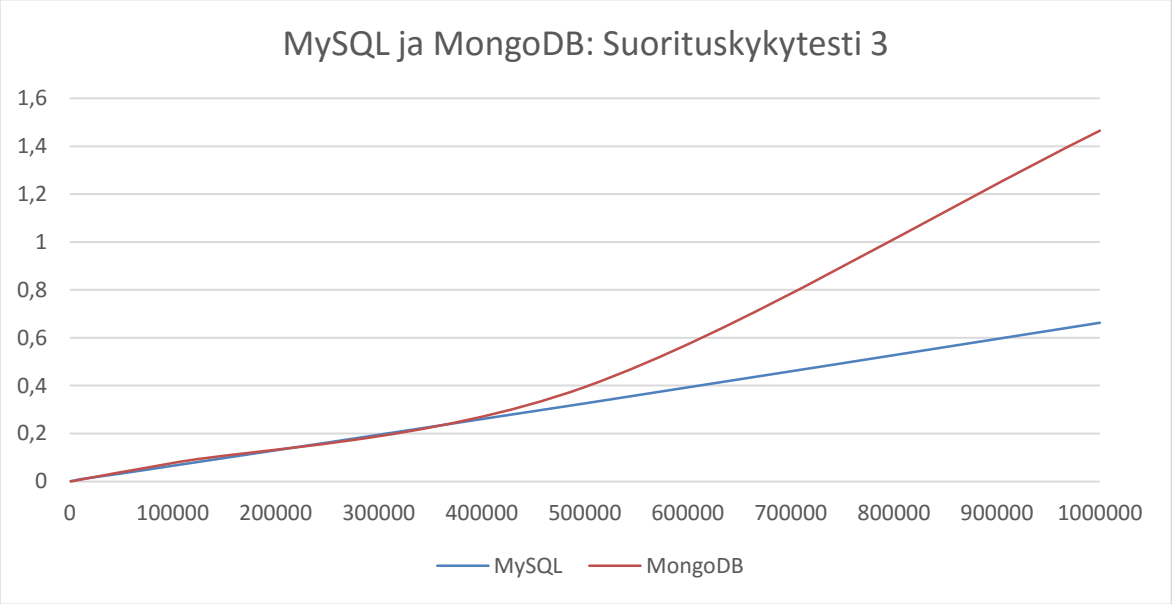
Kuva 22: Toisen suorituskykytestin tulokset kuvaajassa (ilman indeksejä)

Tietueiden määrä	MySQL (sekuntia)	MongoDB (sekuntia)	Erotus %
1000	0,00213	0,0015	42
5000	0,00466	0,0037	26
10000	0,00972	0,0075	30
50000	0,04268	0,0429	-1
100000	0,08457	0,0863	-2
500000	0,41527	0,4558	-9
1000000	0,84150	1,4570	-42

Taulukko 8: Toisen suorituskykytestin tulokset taulukossa (ilman indeksejä).

6.3 Suorituskykytesti 3 – tulokset (indeksoimaton)

Kolmannessa suorituskykytestissä alkuun MongoDB suoriutuu nopeammin kuin MySQL. MongoDB:n suorituskyky kuitenkin hidastuu huomattavasti, kun alkioiden määrä saavuttaa 50000 rajan. Tämän jälkeen MySQL on hieman nopeampi kunnes miljoonan alkion jälkeen MySQL on yli kaksi kertaa MongoDB:tä nopeampi. (Kuva 24) (Taulukko 9)



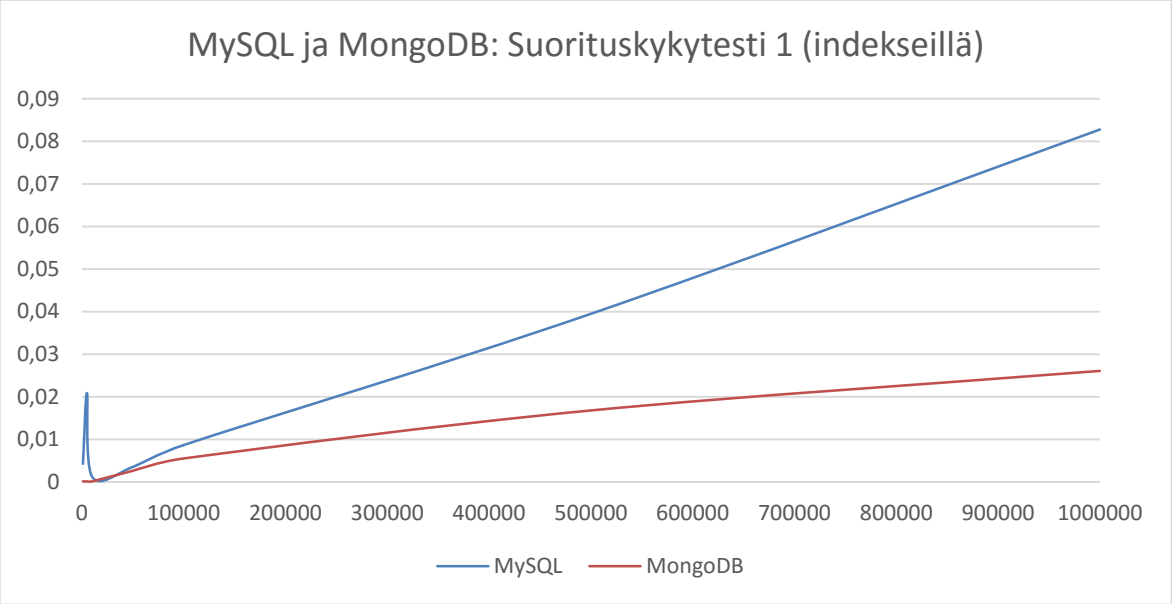
Kuva 23: Kolmannen suorituskykytestin tulokset kuvaajassa (ilman indeksejä)

Tietueiden määrä	MySQL (sekuntia)	MongoDB (sekuntia)	Erotus %
1000	0,001069	0,0010	7
5000	0,004018	0,0033	22
10000	0,009196	0,0074	24
50000	0,033322	0,0390	-15
100000	0,065778	0,0770	-15
500000	0,326286	0,3945	-17
1000000	0,663310	1,4654	-55

Taulukko 9: Kolmannen suorituskykytestin tulokset taulukossa (ilman indeksejä)

6.4 Suorituskykytesti 1 – tulokset (indekseillä)

Ensimmäinen suorituskykytesti indeksien kanssa antaa meille seuraavanlaiset tulokset. Käytännössä siis MongoDB on koko testin ajan nopeampi kuin MySQL. Lisäksi tässä on hyvä huomata se, että MySQL:n suorituskyky tippuu 5000 alkion kohdalla dramaattisesti ja parantuu hiljalleen sen jälkeen. En löytänyt tähän mitään selitystä, miksi MySQL käyttäytyy kyseisellä tavalla indeksien kanssa. Testitulos on hyvin samankaltainen verrattuna testiin ilman indeksejä (Kappale 6.1) (Kuva 24) (Taulukko 10).



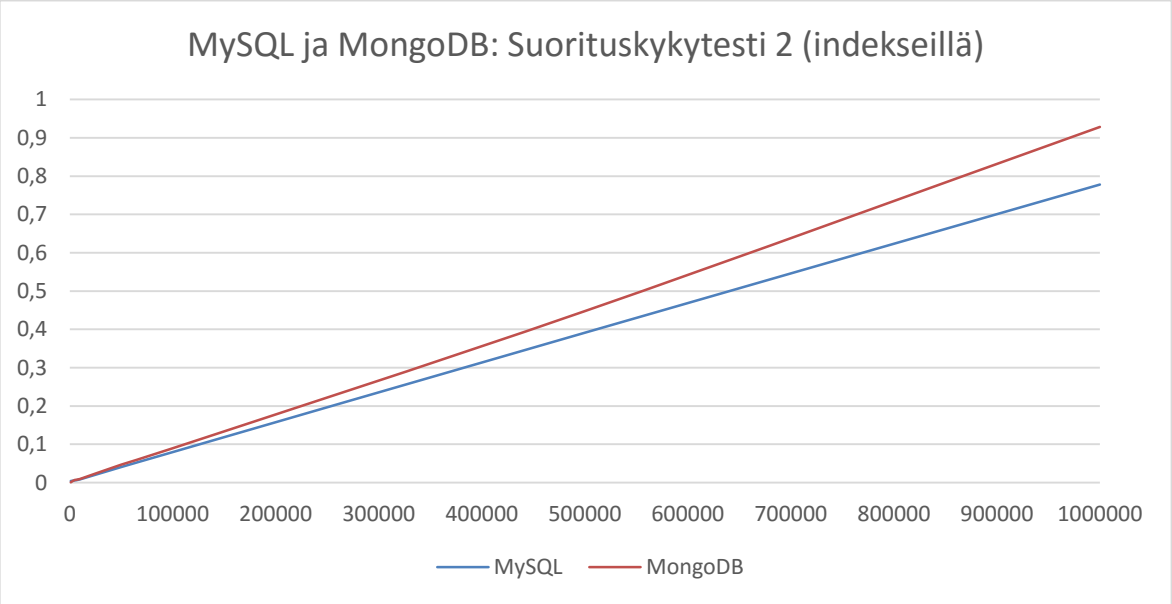
Kuva 24: Ensimmäisen suorituskykytestin tulokset kuvaajassa (indeksien kanssa)

Tietueiden määrä	MySQL (sekuntia)	MongoDB (sekuntia)	Erotus %
1000	0,0042659	0,0001	4166
5000	0,0208450	0,0001	20745
10000	0,0012246	0,0001	1125
50000	0,0035115	0,0026	35
100000	0,0086700	0,0055	58
500000	0,0394902	0,0168	135
1000000	0,0827709	0,0261	217

Taulukko 10: Ensimmäisen suorituskykytestin tulokset taulukossa (indeksien kanssa).

6.5 Suorituskykytesti 2 – tulokset (indekseillä)

Toisessa suorituskykytestissä indekseillä saatiin seuraavat tulokset. Käytännössä MySQL suoritutuu hiukan MongoDB:tä nopeammin kyselyistä. Kuten aikaisemmassa testeissä ilman indeksiä kerroimme, että MongoDB:n hitaus aiheutuu siitä, että $elemMatch –operaattori joutuu käymään jokaisen ”henkiloryhmat” –taulukon sisällä olevat alkiot läpi ja tekemään vertailuja annetun ehdon mukaisesti. [30] (Kuva 25) (Taulukko 11)



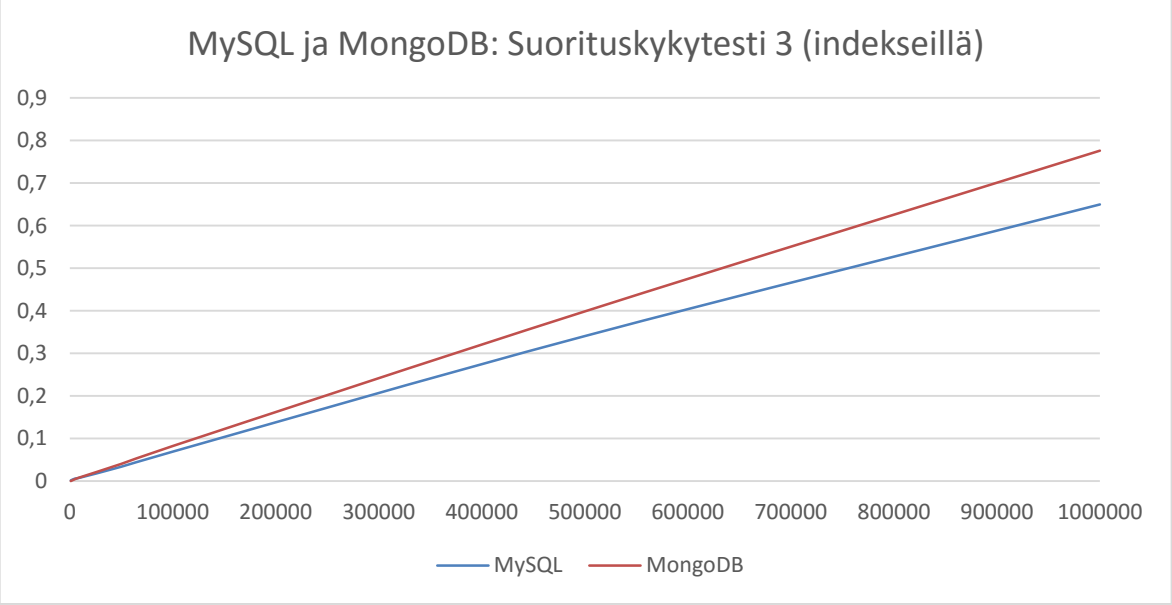
Kuva 25: Toisen suoristuskykytestin tulokset kuvaajassa (indeksien kanssa)

Tietueiden määrä	MySQL (sekuntia)	MongoDB (sekuntia)	Erotus %
1000	0,00418	0,0012	248
5000	0,00588	0,0073	-19
10000	0,00846	0,0098	-14
50000	0,04070	0,0472	-14
100000	0,07969	0,0900	-11
500000	0,39117	0,4478	-13
1000000	0,77794	0,9284	-16

Taulukko 11: Toisen suorituskykytestin tulokset taulukossa (indeksien kanssa).

6.6 Suorituskykytesti 3 – tulokset (indekseillä)

Kolmannessa suorituskykytesteissä indekseillä saatiin seuraavat tulokset. (Kuva 26) (Taulukko 12) Tässä testissä MongoDB suoritutuu pienillä määrillä dataa nopeammin kuin MySQL. Tietueiden määrän kasvaessa yli 5000 MongoDB:n suorituskyky alkaa heikentyä verrattuna MySQL:ään. Taulukosta huomaamme, että MongoDB on hitaimmillaan 17% hitaampi kuin MySQL. Huomattavaa tässä on kuitenkin se, että verrattuna ilman testiin joka suoritettiin ilman indeksejä, tietokantojen välinen suorituskykyero on huomattavasti pienempi. (Kuva 23)



Kuva 26: Kolmannen suoristuskykytestin tulokset kuvaajassa (indeksien kanssa)

Tietueiden määrä	MySQL (sekuntia)	MongoDB (sekuntia)	Erotus %
1000	0,00147	0,0002	634
5000	0,00558	0,0046	21
10000	0,0080	0,0090	-12
50000	0,03365	0,0405	-17
100000	0,06905	0,0825	-16
500000	0,34065	0,3987	-15
1000000	0,65000	0,7763	-16

Taulukko 12: Kolmannen suorituskykytestin tulokset taulukossa (indeksien kanssa).

6.7 Suorituskykytesti 1 – MySQL:n yhdistetyt tulokset (ilman indeksejä ja indekseillä)

Tässä kappaleessa on koottu yhteen MySQL:n tulokset ensimmäisestä suorituskykytestistä indekseillä ja ilman indeksejä. Kuvaajasta huomaamme, että jo 5000 tietueen kohdalla indeksin käytöstä on merkittävää hyötyä ja 10000 tietueen kohdalla kyselyt suoriutuvat yli 400% nopeammin kuin ilman indeksejä. Tämä johtuu käytännössä siitä, että ilman indeksejä MySQL joutuu käymään koko taulun läpi (eng. full table scan) [37] Indeksien käyttö rajaa käytännössä läpikäytävien rivien määrä, jolloin suorituskyky paranee.

Kuva 27: Ensimmäisen suorituskykytestin tulokset kuvaajassa (MySQL)

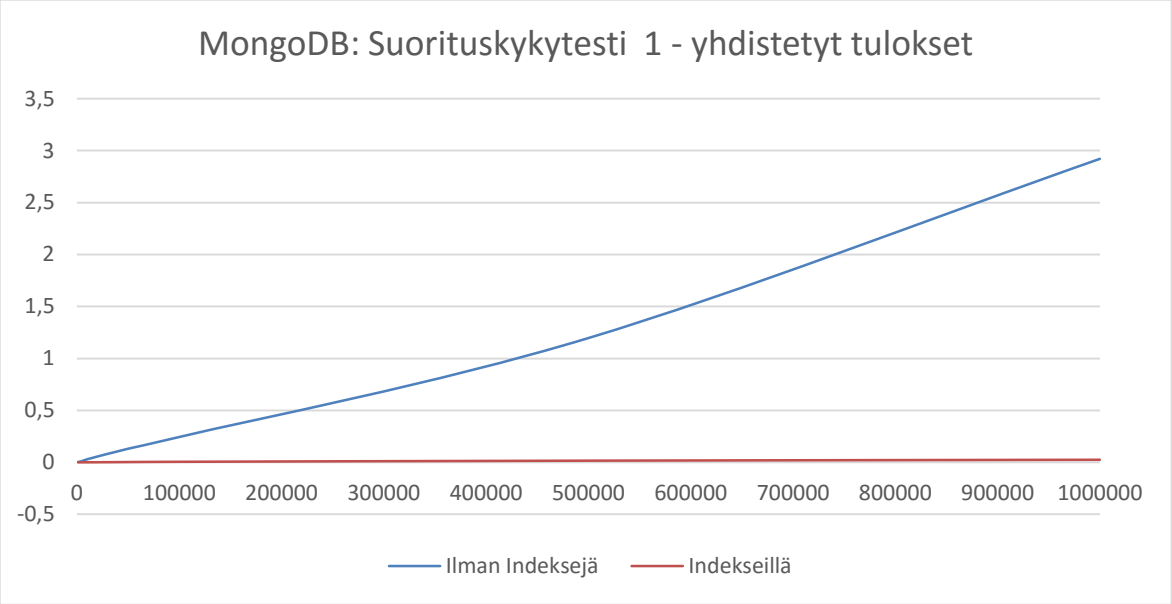
Tietueiden määrä	MySQL (sekuntia)[2]	MySQL (sekuntia)[3]	Erotus %
1000	0,0043941	0,0041818	5
5000	0,0207184	0,0058771	253
10000	0,0470385	0,0084593	456
50000	0,2403880	0,0406998	491
100000	0,4657713	0,0796937	484
500000	2,6142552	0,3911711	568
1000000	5,1370533	0,7779442	560

Taulukko 13: Ensimmäisen suorituskykytestin tulokset taulukossa (MySQL)

6.8 Suorituskykytesti 1 – MongoDB:n yhdistetyt tulokset (ilman indeksejä ja indekseillä)

Tässä kappaleessa on koottu yhteen MongoDB:n tulokset ensimmäisestä suorituskykytestistä indekseillä ja ilman indeksejä. Kuvaajasta huomaamme, että indeksejä lisäämällä kyselyt suoriutuvat merkittävästi nopeammin. Tosin alle 10000 tietueen suorituskyky vertailuja ei ole mielekästä vertailla keskenään, koska indeksien lisäämisien jälkeen suoritusaika on liian pieni mitattavaksi. Kuitenkin miljoonan alkion kohdalla suorituskyky on indeksien kanssa yli 11 kertaa parempi kuin ilman indeksejä. Tässä esimerkissä huomataan samanlainen tilanne, kun edellisessä kappaleessa MySQL:n kanssa. Käytännössä siis ilman indeksejä MongoDB joutuu käymään koko kokoelman läpi (eng. collection scan) [36] Tässäkin tapauksessa indeksien käyttö rajaa läpikäytävien dokumenttien määrää jolloin suorituskyky nopeutuu huomattavasti.

[2] MySQL ilman indeksejä.
[3] MySQL indeksien kanssa.



Kuva 28: Ensimmäisen suorituskykytestin tulokset kuvaajassa (MongoDB)

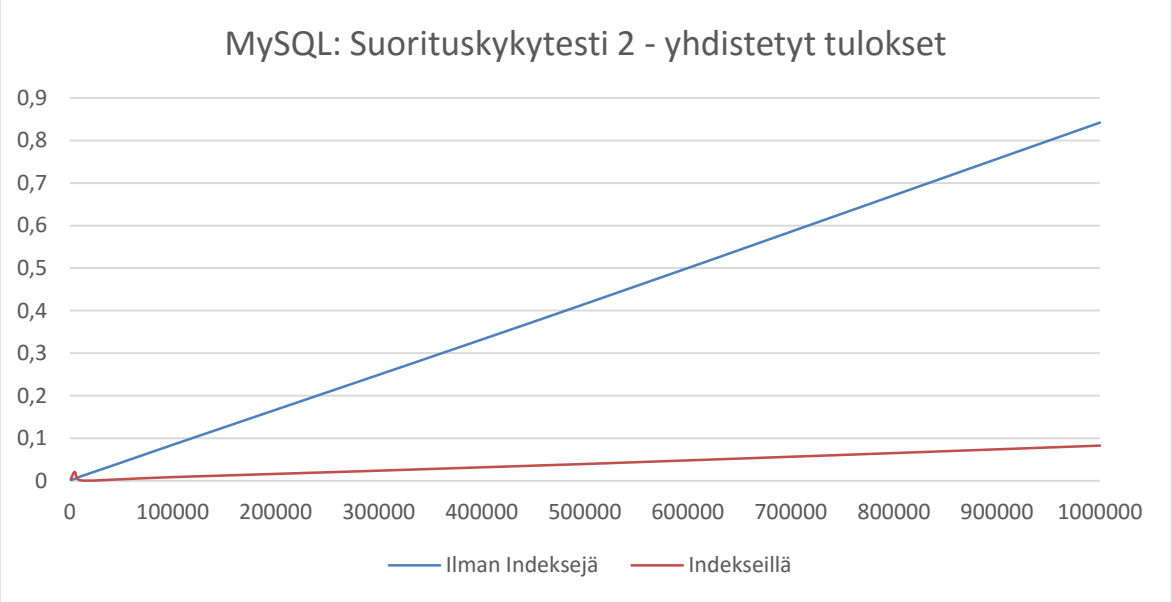
Tietueiden määrä	MongoDB (sekuntia)[4]	MongoDB (sekuntia)[5]	Erotus %
1000	0,0020	0,0001	1900
5000	0,0114	0,0001	11300
10000	0,0287	0,0001	28600
50000	0,1303	0,0026	4912
100000	0,2431	0,0055	4320
500000	1,1962	0,0168	7020
1000000	2,9225	0,0261	1197

Taulukko 14: Ensimmäisen suorituskykytestin tulokset taulukossa (MongoDB)

[4] MongoDB ilman indeksejä.
[5] MongoDB indeksien kanssa.

6.9 Suorituskykytesti 2 – MySQL:n yhdistetyt tulokset (ilman indeksejä ja indekseillä)

Tässä kappaleessa on koottu yhteen MySQL:n tulokset toisesta suorituskykytestistä indekseillä ja ilman indeksejä. Kuvaajasta voimme päätellä, että alle 5000 tietueen määrällä on nopeampi tehdä kyselyitä ilman indeksejä kuin indeksien kanssa. Näin ollen merkittävä hyöty indekseistä saadaan vasta, kun tietueiden määrä kasvaa yli 5000:nen. Tämän jälkeen jo miljoonan alkion kohdalla suorituskyky on nopeutunut noin yhdeksän kertaiseksi. Tässä esimerkissä huomaamme kuinka PAINO –kentän indeksoiminen vaikuttaa suorituskykyyn positiivisesti. Käytännössä tämä estää edellisissä kappaleissa mainitun koko taulun läpikäynnin ja mahdollistaa sen, että kyselyn tulos saadaan käymällä läpi pienempi joukko rivejä, jolloin suorituskykykin on huomattavasti parempi. [37]



Kuva 29: Toisen suorituskykytestin tulokset kuvaajassa (MySQL)

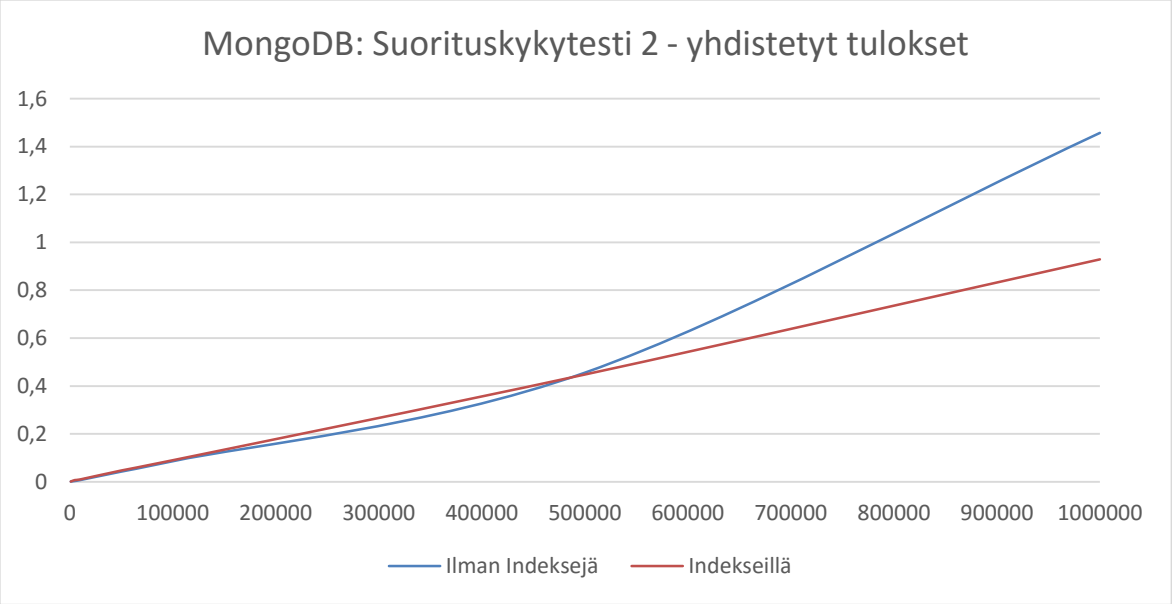
Tietueiden määrä	MySQL (sekuntia)[6]	MySQL (sekuntia)[7]	Erotus %
1000	0,0021299	0,0042659	-50
5000	0,0046576	0,0208450	-78
10000	0,0097164	0,0012246	693
50000	0,0426751	0,0035115	1115
100000	0,0845650	0,0086700	875
500000	0,4152680	0,0394902	952
1000000	0,8414989	0,0827709	917

Taulukko 15: Toisen suorituskykytestin tulokset taulukossa (MySQL)

6.10 Suorituskykytesti 2 – MongoDB:n yhdistetyt tulokset (ilman indeksejä ja indekseillä)

Tässä kappaleessa on koottu yhteen MongoDB:n tulokset toisesta suorituskykytestistä indekseillä ja ilman indeksejä. Kuten kuvaajasta huomaamme niin indeksien käytöllä ei saavuteta tässä tilanteessa ja käytössä olevalla datamäärällä merkittävää etua verrattuna edelliseen esimerkiksi edelliseen testiin. Ero oikeastaan alkaa näkyä vasta miljoonan alkion kohdalla.

[6] MySQL ilman indeksejä.
[7] MySQL indeksien kanssa.



Kuva 30: Toisen suorituskykytestin tulokset kuvaajassa (MongoDB)

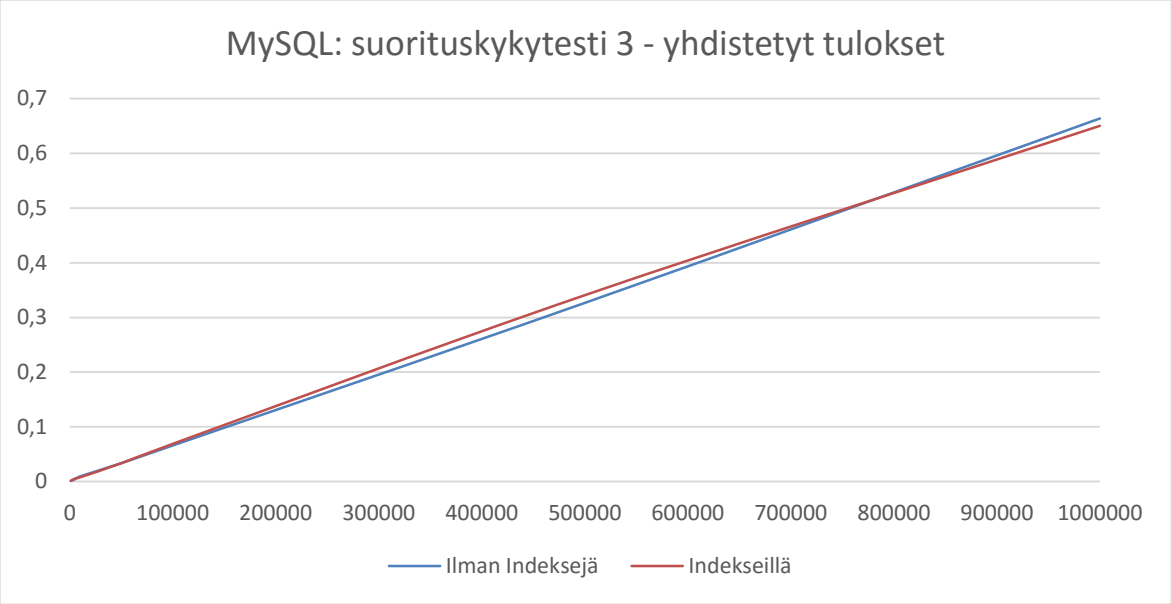
Tietueiden määrä	MongoDB (sekuntia)[8]	MongoDB (sekuntia)[9]	Erotus %
1000	0,0015	0,0012	25
5000	0,0037	0,0073	-49
10000	0,0075	0,0098	-23
50000	0,0429	0,0472	-9
100000	0,0863	0,0900	-4
500000	0,4558	0,4478	2
1000000	1,4570	0,9284	57

Taulukko 16: Toisen suorituskykytestin tulokset kuvaajassa (MongoDB)

[8] MongoDB ilman indeksejä.
[9] MongoDB indeksien kanssa.

6.11 Suorituskykytesti 3 – MySQL:n yhdistetyt tulokset (ilman indeksejä ja indekseillä)

Tässä kappaleessa on koottu yhteen MySQL:n tulokset kolmannesta suorituskykytestistä indekseillä ja ilman indeksejä. Tämän testin yhdistetyissä tuloksissa huomaamme sen, että indeksien lisäämisellä ei ole merkitystä, koska molemmat kuvaajat ovat lähes päällekkäin.



Kuva 31: Kolmannen suorituskykytestin tulokset kuvaajassa (MySQL)

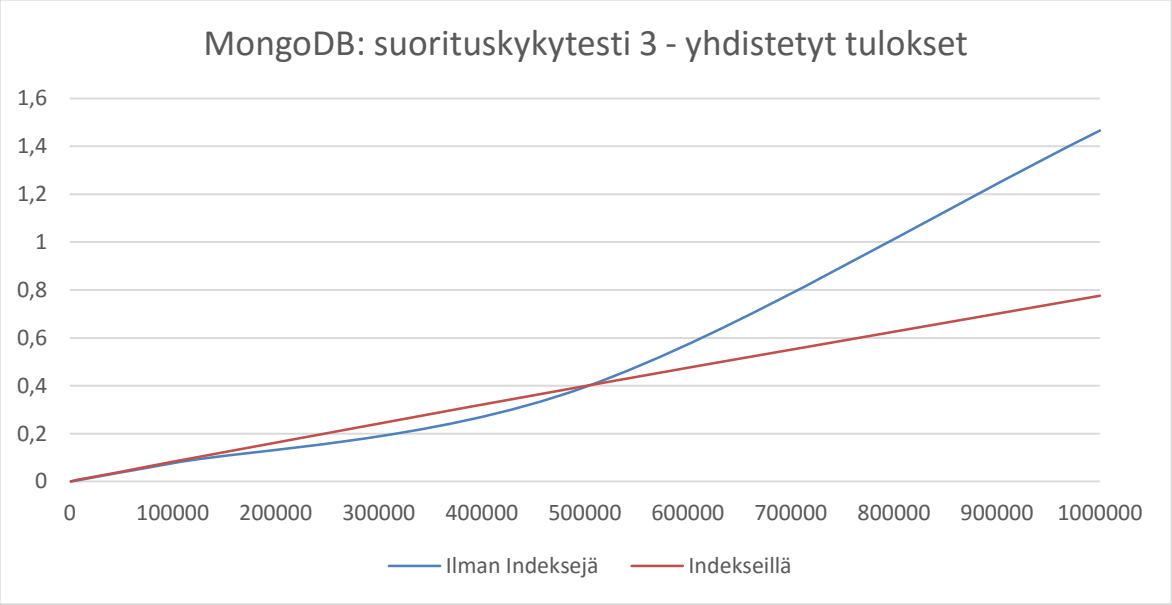
Tietueiden määrä	MySQL (sekuntia)[10]	MySQL (sekuntia)[11]	Erotus %
1000	0,0010693	0,0014674	-27
5000	0,0040175	0,0055747	-28
10000	0,0091962	0,0079561	16
50000	0,0333219	0,0336488	-1
100000	0,0657778	0,0690479	-5
500000	0,3262864	0,3406453	-4
1000000	0,6633103	0,6500039	2

[10] MySQL ilman indeksejä.
[11] MySQL indeksien kanssa.

Taulukko 17: Kolmannen suorituskykytestin tulokset taulukossa (MySQL)

6.12 Suorituskykytesti 3 – MongoDB:n yhdistetyt tulokset (ilman indeksejä ja indekseillä)

Tässä kappaleessa on koottu yhteen MongoDB:n tulokset kolmannesta suorituskykytestistä indekseillä ja ilman indeksejä. Tuloksesta huomaamme, että ensimmäisen testin aika indeksien kanssa ja jopa ilman indeksejä on niin pieni, että sitä ei välttämättä ole mielekästä verrata. Testistä kuitenkin huomaamme sen, että indeksien käytöstä on vasta merkitystä, kun tietueiden määrä saavuttaa miljoonan tietueen rajapyykin. Tässä tapauksessa suoritus on noin 1,9 kertaa nopeampaa kuin ilman indeksejä.



Kuva 32: Kolmannen suorituskykytestin tulokset kuvaajassa (MongoDB)

Tietueiden määrä	MongoDB (sekuntia)[12]	MongoDB (sekuntia)[13]	Erotus %
1000	0,0010	0,0002	400
5000	0,0033	0,0046	-28
10000	0,0074	0,0090	-18
50000	0,0390	0,0405	-4
100000	0,0770	0,0825	-7
500000	0,3945	0,3987	-1
1000000	1,4654	0,7763	89

Taulukko 18: Kolmanne suorituskykytestin tulokset taulukossa (MongoDB)

[12] MongoDB ilman indeksejä.
[13] MongoDB indeksien kanssa.

7 Yhteenveto

Tutkielmassa käsiteltiin kaksi toisistaan eroavaa tietokantatyyppiä: relaatio- ja epärelaatiotietokannat. Relaatiotietokannat perustuvat 1970 –luvulla esiteltyyn relaatiomalliin, kun taas epärelaatiotietokannat ovat tulleet tutuksi suurelle yleisölle vasta 2000 –luvun puolella. Epärelaatiotietokannat, joista käytetään useasti nimeä NoSQL –tietokannat ovat suunniteltu toimimaan nopeasti suurilla datamäärillä (Big Data). Käytännössä tämä tarkoittaa usein sitä, että relaatiotietokannan teorian mukaisista ACID –ominaisuuksista on luovuttu ja otettu käyttöön epärelaatiokannoille tyypillinen BASE –oikeellisuusmalli. BASE –oikeellisuusmalli tarjoaa mahdollisuuden rikkoa tietokannan eheyttä mahdollistaen sen, että tietokanta on paremmin skaalattavissa useammalle palvelimelle. Useamman palvelimen ryväs taas mahdollistaa paremman suorituskyvyn. BASE -oikeellisuusmalli tarkoittaa käytännössä sitä, että tietokanta voi olla hetkellisesti epäeheässä tilassa kuitenkin niin, että se palautuu loppujen lopuksi eheään tilaan. Tämä saavutetaan niin, että heikennetään ACID –ominaisuuksista eheyttä ja eristyneisyyttä.

Erillaisia NoSQL -tietokantoja on tällä hetkellä olemassa yli 225 kappaletta. Suuren määrän lisäksi ne jakautuvat useampaan eri kategoriaan, joita ovat mm. saraketietokannat, dokumenttitietokannat, avain-arvo –tietokannat, verkkotietokannat, oliotietokannat, xml-tietokannat, aikasarjatietokannat ja monia muita jotka eivät sovi edellä mainittuihin kategorioihin. Verrattuna relaatiotietokantojen erilaisten kategorioiden määrä on huimaava ja jokaisella tietokannalla on omanlaisensa käyttötarkoituksensa.

Tässä tutkielmassa tarkasteltiin molemmista tietokantatyypeistä yhtä tuotetta ja verrattiin niitä keskenään toisiinsa suorituskyvyn merkeissä. Valitsin tutkittaviksi tietokantahallintajärjestelmiksi MySQL:n ja MongoDB:n, koska niistä minulla on henkilökohtaisesti eniten kokemusta. Tutkimuksessa mallinnettiin molemmille tietokannoille tyypillisessä muodossa seuraavanlainen skenaario: ”Järjestelmään voi lisätä henkilöitä, joilla on henkilölle tyypillisiä tietoja. Henkilön tietoihin kuuluu seuraavat: etunimi, sukunimi, puhelinnumero, paino ja passiivisuustieto. Tämän lisäksi jokainen järjestelmän henkilö voi kuulua tai olla kuulu-

matta yhteen tai useampaan henkilöryhmään. Henkilöryhmä sisältää nimen ja passiivisuustiedon.” Kyseisen skenaarion valitsin sen takia, koska se on suhteellisen yksinkertainen ja sillä on suora kytkös reaalimaailmaan. Lisäksi MySQL:ssä joudutaan luomaan tiedon tallennusta varten useampi taulu ja MongoDB:ssä useampia alidokumentteja.

Suorituskykytestien perusteella voidaan sanoa, että MySQL ja MongoDB eroavat toisistaan erityyppisen kyselyiden suoritusnopeudessa. Testien perusteella MongoDB on suorituskykyisempi sellaisissa tilanteissa, joissa tietojen hakeminen MySQL:ssä ei onnistu ilman JOIN –lauseita. Tämä johtuu yksinkertaisesti siitä, että MongoDB:ssä voidaan liitostietoja hakea suoraan dokumentin alidokumentilta (Kuva 20) [35] Kuitenkin tilanteet, missä hakuehto muodostetaan niin, että tulosjoukko saadaan suoraan yhden taulun tai dokumentin alta, MySQL:n suorituskyky on parempi. On kuitenkin myös hyvä muistaa, että suorituskykytesteissä tietokannassa olleiden alkioiden määrä on hyvin pieni ja tuloksia ei välttämättä voida yleistää isoille datamäärille. Lisäksi testeissä ei huomioitu millään tavalla kirjoitusoperaatioita tai tietokantojen hajauttamista useammalle palvelimelle. Niiden lisääminen testeihin antaisi parmman kuvan kokonaissuorituskyvystä.

Valittaessa näiden kahden tuotteen väliltä täytyy muistaa, että kumpikaan ei ole ylitse toisen. Esimerkiksi MongoDB on suunniteltu esimerkiksi logien tallennukseen, raportteihin, tuotekatalogeihin, kommenttien, metadatan ja ominaisuuksien, kategoria hierarkian tallentamisen ja tavaraluettelon hallintaan. Edellä mainituista käyttötapauksista MongoDB tarjoaa vielä kattavan dokumentaation verkkosivuillaan. [36] Yhtenäistä eri käyttötapojen välillä on se, että kaikki tarvittava data on yhden kokoelman alla riippuen esitystavasta eli käytännössä relaatioita useisiin kokoelmiin ei muodosteta. Lisäksi kannattaa huomata, että esimerkiksi logittaminen, voi sisältää hyvin heterogeenistä dataa saman kokoelman eri dokumenttien välillä.

MySQL taas soveltuu tilanteisiin, missä vaatimukset ovat selvästi määritelty ja datan eheydestä ei voida tinkiä. Käytännössä siis tarkoittaa tilanteita, missä täytyy ottaa huomioon ACID –ominaisuuksien vaatimukset. Tyypillisesti näitä vaatimuksia pitää noudattaa rahan-

siirrossa ja muissa transaktioita vaativissa sovelluksissa. [37] Vaikka usein luullaan, että relaatiotietokannat eivät ole helposti vertikaalisesti skaalattavissa, niin tämä väite on väärin. Esimerkiksi MySQL tarjoaa version, joka mahdollistaa klusteroimisen. [38]

Lähteet

[1] AMC, ”A.M. Turing Award,” [Online]. Available: http://amturing.acm.org/award_winners/bachman_1896680.cfm. [Haettu 15 5 2015].

[2] Eventbrite, ”NoSQL meetup,” 11 6 2009. [Online]. Available: http://www.eventbrite.com/e/nosql-meetup-tickets-341739151. [Haettu 13 2 2015].

[3] ”NoSQL Databases,” [Online]. Available: http://nosql-database.org/. [Haettu 6 3 2015].

[4] M. Kifer ja L. M. P. Bernstein Arthur, Database Systems An Application-Oriented Approach, New York: Pearson, 2006.

[5] E. F. Codd, ”A Relational Model of Data for Large Shared Data Banks,” *Communications of the ACM*, osa/vuosik. 13, nro 6, pp. 377-387, Kesäkuu 1970.

[6] E. Codd, The Relational Model For Database Management, osa/vuosik. Version 2, Addison-Wesley Publishing Company, 1990.

[7] T. Lahtonen, SQL Toolkit, Vantaa: Docendo Finland Oy; SanomaWSOY-konserni, 2003.

[8] J. L. Harrington, Relational Database Design Clearly Explained, New York: Morgan Kaufmann, 2002.

[9] P. O'Neil, Database: Principles, Programming, Performance, San Francisco: Morgan Kaufmann Publishers, Inc, 1994.

[10] D. Pritchett, ”BASE: An Acid Alternative,” *Queue*, osa/vuosik. 3, nro 6, pp. 48 - 55, 2008.

[11] J. Gray, ”The Transaction Concept: Virtues and Limitations,” Tandem Computers Incorporated, Cupertino, 1981.

[12] w3resource, ”NoSQL,” 1 28 2015. [Online]. Available: http://www.w3resource.com/mongodb/nosql.php. [Haettu 13 2 2015].

[13] D. Jeffrey ja G. Sanjay, ”MapReduce: Simplified Data Processing on Large Clusters,” Google, Inc., San Francisco, 2004.

[14] D. Jeffrey ja G. Sanjay, ”MapReduce: A Frelxible Data Processing Tool,” *Communications of the ACM,* osa/vuosik. 53, nro 1, pp. 72-77, 2010.

[15] P. Jaroslav, ”NoSQL databases: a step to database scalability in web environment,” *International Journal of Web Information Systems,* osa/vuosik. 1, nro 9, pp. 69-82, 2013.

[16] The Apache Software Foundation, ”https://cassandra.apache.org/,” The Apache Software Foundation, [Online]. Available: https://cassandra.apache.org/. [Haettu 17 5 2015].

[17] E. Brewer, ”CAP Twelve Years Later: How the "Rules" Have Changed,” IEEE Computer Society, Berkeley, 2012.

[18] M. Stonebraker, ”Errors in Database Systems, Eventual Consistency, and the CAP Theorem,” Communications of the ACM, 2010.

[19] I. FindTheBest.com, ”Compare NoSQL Databases,” FindTheBest.com, Inc., [Online]. Available: http://nosql.findthebest-sw.com/. [Haettu 4 2 2015].

[20] N. Leavitt, ”Will NoSQL Databases Live Up to Their Promise,” *Computer,* pp. 12-14, 2 2010.

[21] M. Seeger, ”Key-Value stores: a practical overview,” medien informatik, Stuttgart, 2009.

[22] MongoDB Inc., ”Top 5 Considerations When Evaluating,” MongoDB Inc., New York, Palo Alto, Washington D.C, London, Dublin, Barcelona, Sydney, Tel Aviv, 2015.

[23] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O’Neil, P. O’Neil, A. Rasin, N. Tran ja S. Zdonik, ”C-Store: A Column-oriented DBMS,” tekijä: *Proceedings of the 31st VLDB Conference*, Trondheim, Norway, 2005.

[24] D. J. Adabi, S. R. Madden ja N. Hachem, ”Column-Stores vs. Row-Stores: How Different Are They?,” ACM, Vancouver, 2008.

[25] Neo Technology, Inc., ”What is a Graph Database?,” [Online]. Available: http://neo4j.com/developer/graph-database/. [Haettu 16 03 2016].

[26] Microsoft, ”Microsoft Azure,” [Online]. Available: http://azure.microsoft.com/en-us/. [Haettu 13 3 2015].

[27] Oracle Corporation and/or its affiliates, ”MySQL - Reference Manual - Counting Rows,” Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/refman/5.1/en/counting-rows.html. [Haettu 22 05 2015].

[28] Oracle Corporation, ”MySQL - Reference Manual - JOIN Syntax,” Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/refman/5.0/en/join.html. [Haettu 22 05 2015].

[29] MongoDB, Inc., ”MongoDB CRUD Introduction,” 2015. [Online]. Available: http://docs.mongodb.org/manual/core/crud-introduction/. [Haettu 22 05 2015].

[30] MongoDb, Inc., ”MongoDB $elemMatch (query),” MongoDb, Inc., 2015. [Online]. Available: http://docs.mongodb.org/manual/reference/operator/query/elemMatch/. [Haettu 22 05 2015].

[31] MongoDB, Inc., ”$gte,” MongoDB, Inc., 2015. [Online]. Available: https://docs.mongodb.com/manual/reference/operator/query/gte/. [Haettu 28 05 2016].

[32] Oracle Corporation, ”MySQL - Reference Manual - GROUP BY (Aggregate) Functions,” Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/refman/5.0/en/group-by-functions.html. [Haettu 22 05 2015].

[33] MongoDB, Inc., ”MongoDB Aaggregation Introduction,” MongoDB, Inc., 2015. [Online]. Available: https://docs.mongodb.org/manual/core/aggregation-introduction/. [Haettu 22 05 2015].

[34] MongoDB, Inc., ”MongoDB $group (aggregation),” MongoDB, Inc., 2015. [Online]. Available: http://docs.mongodb.org/manual/reference/operator/aggregation/group/. [Haettu 22 05 2015].

[35] Mongo DB, Inc., ”Introduction to MongoDB,” Mongo DB, inc, [Online]. Available: https://docs.mongodb.com/getting-started/shell/introduction/#documents. [Haettu 28 05 2016].

[36] Mongo DB, Inc, ”Indexes with the mongo Shell,” Mongo DB, Inc, [Online]. Available: https://docs.mongodb.com/getting-started/shell/indexes/. [Haettu 28 05 2016].

[37] ”8.2.1.17 How to Avoid Full Table Scans,” Oracle Corporation, 2016. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/how-to-avoid-table-scan.html. [Haettu 28 05 2016].

[38] MongoDB, inc, ”The MongoDB 2.6 Manual”.

[39] MongoDB, Inc, ”Use cases,” [Online]. Available: https://docs.mongodb.org/ecosystem/use-cases/. [Haettu 28 3 2016].

[40] C. Buckler, ”SQL vs NoSQL: How to Choose,” 23 9 2015. [Online]. Available: http://www.sitepoint.com/sql-vs-nosql-choose/. [Haettu 28 3 2016].

[41] Oracle Corporation, ”18.2 MySQL Cluster Installation,” [Online]. Available: http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-installation.html. [Haettu 28 2 2016].

[42] IBM, ”SQL Guide: Tables, rows, columns,” IBM Knowledge Center.

[43] R. Elmasri ja N. B. Shamkant, Database Systems: Models, Languages, Design and Application Programming, Pearson, 2011.

[44] S. Harizopoulos, D. Abadi ja P. Boncz, *Column-Oriented Database Systems,* VLDB 2009 Tutorial , 2009.

Liitteet

A src/run.php

```
#!/usr/bin/php
<?php
/**
 * This scripts runs benchmark tests against MySQL and MongoDB. You can run this
 * script with following command:
 *      $ php run.php
 *
 * Tested with PHP 5.5.
 */
use classes\commands\interfaces\ExecutableCommand;

try {
     // Require init.php file.
     require_once "init.php";

     /**
     * @var ExecutableCommand[] $commands Executable commands.
     */
     $commands = array();

     // Open commands directory.
     if ($handle = opendir(COMMANDS_DIR)) {
          while (($entry = readdir($handle)) !== false) {
               // Read command files.
               if ($entry !== "." && $entry !== ".." && !is_dir("./commands/" .
$entry) &&
                    strpos($entry, "Command") === false && strpos($entry,
".php")
               ) {
                    // Get class name
                    $class = "\\classes\\commands\\" . str_replace(".php", "",
$entry);
                    // Create new instance
                    $commands[] = new $class();
               }
          }
          closedir($handle);
     }

     echo sprintf(
          "This is a script that helps creating databases for benchmarking tests
and you can run queries with it.
=================
Available commands:
---
"
     );

     foreach ($commands as $key => $command) {

```

```
          echo sprintf(
               "[%1\$s] - %2\$s - %3\$s\n",
               /** 1 */ $key,
               /** 2 */ $command->getName(),
               /** 3 */ $command->help()
          );
     }
     echo sprintf(
          "---
==================
"
     );
     $selectedCommand = (int)readline("Execute command > ");
     // Get selected command.
     $command = $commands[$selectedCommand];
     // Execute and get results.
     $results = $command->benchmark();

     if (!empty($results)) {
          foreach ($results as $key => $result) {
               echo sprintf(
                    "[%1\$s] %2\$s\n",
                    /** 1 */ $key,
                    /** 2 */ $result
               );
          }

          echo sprintf(
               "AVG = %1\$s\n",
               bcdiv(array_sum($results), count($results), 6)
          );
     }
} catch (\Exception $e) {
     echo sprintf(
          "
==================
Error:
---
Message:    %1\$s
File:       %2\$s
Line:       %3\$s
Code:       %4\$s
==================
",
          $e->getMessage(),
          $e->getFile(),
          $e->getLine(),
          $e->getCode()
     );
}
```

B src/init.php

```php
<?php

define("RANDOM_SEED", 1234567890);
define("COMMANDS_DIR", dirname(__FILE__) . DIRECTORY_SEPARATOR . "classes/commands");
define("DEBUG", true);

define("MONGO_DATABASE_NAME", "gradu");
define("MONGO_COLLECTION_NAME", "users");

define("MYSQL_DATABASE_NAME", "gradu");
define("MYSQL_HOST", "localhost");
define("MYSQL_USER", "xxxx");
define("MYSQL_PASS", "xxxx");

define("TEST_COUNT", 10);

/**
 * Default loader for classes.
 *
 * @param string $className Class name
 */
function classLoader($className)
{
    // Split class name.
    $parts = explode("\\", $className);
    $path = "";

    // Get first part of the class path.
    while (count($parts) > 1) {
        $path .= strtolower(array_shift($parts)) . DIRECTORY_SEPARATOR;
    }

    // Get actual class name.
    $path .= array_shift($parts) . '.php';

    // Check that we can access the file and require it.
    if (is_readable($path) && is_file($path)) {
        require_once $path;
    }
}

// Init default class loader.
spl_autoload_register('classLoader');

// Default seed for rand() -function.
srand(RANDOM_SEED);
```

C src/classes/database/Connection.php

```php
<?php
/**
 * \classes\database\Connection.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\database;

use classes\database\interfaces\DatabaseConnection;

/**
 * Class Connection
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\database
 */
abstract class Connection implements DatabaseConnection
{
     /**
     * Connection instance.
     *
     * @var null|DatabaseConnection
     */
     protected static $instance = null;

     /**
     * Returns connection instance.
     *
     * @return DatabaseConnection
     */
     public static function getInstance()
     {
          $class = get_called_class();

          if (is_null(Connection::$instance[$class])) {
               Connection::$instance[$class] = new $class();
          }

          return Connection::$instance[$class];
     }
}
```

D src/classes/database/MongoDBConnection.php

```php
<?php
/**
```

```
 * \classes\database\MongoDBConnection.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\database;

use MongoClient;
use MongoCollection;
use MongoDB;

/**
 * Class MongoDBConnection
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\database
 */
class MongoDBConnection extends Connection
{
     /**
     * @var null|MongoClient
     */
     private $mongoClient = null;

     /**
     * @var null|MongoDB
     */
     private $mongoDB = null;

     /**
     * @var null|MongoCollection
     */
     private $collection = null;

     /**
     * Connects to database.
     *
     * @return void
     */
     public function connect()
     {
          $this->mongoClient = new \MongoClient();
     }

     /**
     * Disconnects from database.
     *
     * @return void
     */
     public function disconnect()
     {
          if ($this->mongoClient->connected) {
               $this->mongoClient->close();
          }
     }

```

```
61        /**
62        * Makes queries to database using parameters.
63        *
64        * @param array    $parameters    Query parameters.
65        *
66        * @return void
67        */
68        public function insert(array $parameters)
69        {
70             if (DEBUG) {
71                  $output = "INSERT ";
72
73                  $output .= $this->debugQuery($parameters);
74
75                  echo $output . "\n";
76             }
77
78             $this->collection->insert($parameters);
79        }
80
81        /**
82        * Makes indexes to database.
83        *
84        * @param array     $parameters    Parameters
85        *
86        * @return void
87        */
88        public function ensureIndex(array $parameters)
89        {
90             $this->collection->ensureIndex($parameters);
91        }
92
93        /**
94        * Selects database using name.
95        *
96        * @param string    $databaseName    Name of the database
97        *
98        * @return void
99        */
100       public function selectDB($databaseName)
101       {
102            $this->mongoDB = $this->mongoClient->selectDB($databaseName);
103       }
104
105       /**
106       * Selects database collection.
107       *
108       * @param string    $collectionName     Name of the collection
109       *
110       * @return void
111       */
112       public function selectCollection($collectionName)
113       {
114            $this->collection = $this->mongoDB->selectCollection($collection-
115 Name);
116       }
117
118       /**
```

```
119       * Helper method for debugging mongo queries.
120       *
121       * @param array     $parameters     Query parameters.
122       *
123       * @return string
124       */
125      private function debugQuery(array $parameters)
126      {
127           $output = "";
128
129           foreach ($parameters as $key => $parameter) {
130                $output .= sprintf(
131                     "(%1\$s => %2\$s)",
132                     /** 1 */ $key,
133                     /** 2 */ is_array($parameter) ? $this->debugQuery($parame-
134 ter) : $parameter
135                );
136           }
137
138           return $output;
139      }
140
141      /**
142      * @return MongoCollection|null
143      */
144      public function getCollection()
145      {
146           return $this->collection;
147      }
148
149      /**
150      * @return MongoDB|null
151      */
152      public function getMongoDB()
153      {
154           return $this->mongoDB;
     }
}
```

E src/classes/database/MySQLConnection.php

```
1  <?php
2  /**
3   * \classes\database\MySQLConnection.php
4   *
5   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
6   * @version 1.0
7   */
8  namespace classes\database;
9
10 use mysqli;
11
```

```php
/**
 * Class MySQLConnection
 *
 * @method static MySQLConnection getInstance()
 *
 * @package classes\database
 */
class MySQLConnection extends Connection
{
     /**
     * Database connection.
     *
     * @var null|mysqli
     */
     private $connection = null;

     /**
     * Connects to database.
     *
     * @return void
     *
     * @throws \Exception
     */
     public function connect()
     {
          if (is_null($this->connection)) {
               $this->connection = mysqli_connect(MYSQL_HOST, MYSQL_USER,
MYSQL_PASS);

               if (!$this->connection) {
                    throw new \Exception(mysqli_connect_error());
               }

               $this->query("CREATE DATABASE IF NOT EXISTS " . MYSQL_DATA-
BASE_NAME);
               $this->selectDB(MYSQL_DATABASE_NAME);
          }
     }

     /**
     * Disconnects from database.
     *
     * @return void
     *
     * @throws \Exception
     */
     public function disconnect()
     {
          $status = mysqli_close($this->connection);

          if (!$status) {
               throw new \Exception(mysqli_error($this->connection));
          }

          $this->connection = null;
     }

     /**
```

```
70        * Makes queries to database using parameters.
71        *
72        * @param string|array $query Database query.
73        * @param array $parameters Query parameters.
74        *
75        * @return mixed
76        *
77        * @throws \Exception
78        */
79       public function query($query, array $parameters = array())
80       {
81            $pattern = $replacement = array();
82
83            foreach ($parameters as $key => $value) {
84                 $pattern[] = $key;
85                 $value = (is_numeric($value)) ? $value : ("'" . $value . "'");
86                 $replacement[] = $value;
87            }
88
89            $query = preg_replace('/\s\s+/', ' ', $query);
90
91            $query = str_replace($pattern, $replacement, $query);
92
93            if (DEBUG) {
94                 echo $query . "\n";
95            }
96
97            $result = mysqli_query($this->connection, $query);
98
99            if (!$result) {
100                throw new \Exception(mysqli_error($this->connection));
101           }
102
103           return $result;
104      }
105
106      /**
107       * Selects database using name.
108       *
109       * @param string $databaseName Name of the database
110       *
111       * @return void
112       *
113       * @throws \Exception
114       */
115      public function selectDB($databaseName)
116      {
117           if (is_null($this->connection)) {
118                throw new \Exception("Use connect() -function before selecting
119 database");
120           }
121
122           $status = mysqli_select_db($this->connection, $databaseName);
123
124           if (!$status) {
125                throw new \Exception(mysqli_error($this->connection));
           }
```

```
    }
}
```

F src/classes/database/interfaces/DatabaseConnection.php

```php
<?php
/**
 * \classes\database\interfaces\DatabaseConnection.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\database\interfaces;

/**
 * Interface DatabaseConnection
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\database\interfaces
 */
interface DatabaseConnection
{
    /**
     * Returns connection instance.
     *
     * @return DatabaseConnection
     */
    public static function getInstance();

    /**
     * Connects to database.
     *
     * @return void
     */
    public function connect();

    /**
     * Disconnects from database.
     *
     * @return void
     */
    public function disconnect();
}
```

G src/classes/commands/Command.php

```php
<?php
/**
 * \classes\commands\Command.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\commands;

use classes\commands\interfaces\ExecutableCommand;
use ReflectionClass;

/**
 * Abstract class for commands.
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
abstract class Command implements ExecutableCommand
{
     /**#@+
     * Helper constants for class.
     */
     const CLASS_HELP_ANNOTATION_NAME = "@help";
     const TYPE_INTEGER = 'integer';
     const TYPE_BOOLEAN = 'boolean';
     /**#@-*/

     /**
     * @var mixed Database query.
     */
     protected $query = null;

     /**
     * Method returns name of the class.
     *
     * @return string Class name.
     */
     public function getName()
     {
          return get_called_class();
     }

     /**
     * Returns help text for command.
     *
     * @return string
     */
     public function help()
     {
          $reflectionClass = new ReflectionClass(get_called_class());
          $classCommentBlock = $reflectionClass->getDocComment();

```

```
56            $pattern = "/\*\s" . Command::CLASS_HELP_ANNOTATION_NAME . "\s(.*)/";
57            $annotations = array();
58
59            $status = preg_match_all($pattern, $classCommentBlock, $annotations);
60
61            if ($status) {
62                 return array_pop($annotations[1]);
63            }
64
65            return sprintf(
66                 "Add `%1\$s` to `%2\$s` -class comment block",
67                 Command::CLASS_HELP_ANNOTATION_NAME,
68                 get_called_class()
69            );
70       }
71
72       /**
73       * Asks parameter for command.
74       *
75       * @param string    $question   Question
76       * @param string    $type       Parameter type
77       *
78       * @return mixed
79       */
80       public function askParameter($question, $type = Command::TYPE_INTEGER)
81       {
82            $parameter = null;
83
84            while (empty($parameter)) {
85                 $parameter = readline($question . " > ");
86            }
87
88            if ($type === Command::TYPE_INTEGER) {
89                 $parameter = (int)$parameter;
90            } elseif ($type === Command::TYPE_BOOLEAN) {
91                 $parameter = (($parameter === 'Y') ? true : false);
92            }
93
94            return $parameter;
95       }
  }
```

H src/classes/commands/MySQLCommand.php

```
1  <?php
2  /**
3   * \classes\commands\MySQLCommand.php
4   *
5   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
6   * @version 1.0
7   */
8  namespace classes\commands;
```

```php

use classes\database\MySQLConnection;

/**
 * Class MySQLCommand
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
abstract class MySQLCommand extends Command
{
     /**
     * @var MySQLConnection|null
     */
     protected $db = null;

     /**
     * Default constructor for class.
     *
     * @throws \Exception
     */
     public function __construct()
     {
          $this->db = MySQLConnection::getInstance();
          $this->db->connect();
     }

     /**
     * Executes current command and returns result.
     *
     * @return mixed
     */
     public function execute()
     {
          // Set profiling and limit history size.
          $this->db->query("SET profiling = 1");
          $this->db->query("SET profiling_history_size = 1;");

          // Do actual query.
          $this->db->query($this->query);

          // Fetch profile data.
          $resource = $this->db->query("SHOW profiles");

          $result = mysqli_fetch_assoc($resource);

          $this->db->query("SET profiling = 0");
          $this->db->query("FLUSH TABLES;");

          return bcmul($result["Duration"], 1, 6);
     }

     /**
     * Helper method for benchmarking mysql queries.
     *
```

```
67          * @return array
68          *
69          * @throws \Exception
70          */
71          public function benchmark()
72          {
73               $output = array();
74
75               for ($i = 0; $i < TEST_COUNT; $i++) {
76                    $output[] = $this->execute();
77               }
78
79               return $output;
80          }
     }
```

I src/classes/commands/MongoDBCommand.php

```
1   <?php
2   /**
3    * \classes\commands\MongoDBCommand.php
4    *
5    * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
6    * @version 1.0
7    */
8   namespace classes\commands;
9
10  use classes\database\MongoDBConnection;
11
12  /**
13   * Class MongoDBCommand
14   *
15   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
16   * @version 1.0
17   *
18   * @package classes\commands
19   */
20  abstract class MongoDBCommand extends Command
21  {
22       /**
23       * @var MongoDBConnection|null
24       */
25       protected $db = null;
26
27       /**
28       * @var \MongoCollection|null
29       */
30       private $collection = null;
31
32       /**
33       * Default constructor for class.
34       *
```

```
35        * @throws \Exception
36        */
37       public function __construct()
38       {
39            $this->db = MongoDBConnection::getInstance();
40            $this->db->connect();
41            $this->db->selectDB(MONGO_DATABASE_NAME);
42            $this->db->selectCollection(MONGO_COLLECTION_NAME);
43            $this->db->getMongoDB()->setProfilingLevel(0);
44            $this->collection = $this->db->getCollection();
45       }
46
47       /**
48        * Executes current command and returns result.
49        *
50        * @return mixed
51        */
52       public function execute()
53       {
54            $cursor = $this->collection->find($this->query);
55            $stats = $cursor->explain();
56            return (float)bcdiv($stats["millis"], 1000, 6);
57       }
58
59       /**
60        * Helper method for benchmarking mysql queries.
61        *
62        * @return array
63        *
64        * @throws \Exception
65        */
66       public function benchmark()
67       {
68            $output = array();
69
70            for ($i = 0; $i < TEST_COUNT; $i++) {
71                 $output[] = $this->execute();
72            }
73
74            return $output;
75       }
76  }
```

J src/classes/commands/InitMySQL.php

```
1  <?php
2  /**
3   * \classes\commands\InitMySQL.php
4   *
5   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
6   * @version 1.0
7   */
```

```php
namespace classes\commands;

use classes\database\MySQLConnection;

/**
 * Class InitMySQL
 *
 * @help Initializes MySQL database for tests.
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
class InitMySQL extends MySQLCommand
{
    /**
     * Number of user that this script creates.
     *
     * @var int
     */
    private $userCount = 1000;

    /**
     * Whether to add indexes to database or not.
     *
     * @var bool
     */
    private $addIndexes = false;

    /**
     * Executes current command and returns result.
     *
     * @return mixed
     */
    public function execute()
    {
        $this->userCount = $this->askParameter("How many users do you want to
create ");
        $this->addIndexes = $this->askParameter("Do you want to add indexes
[Y/n]", Command::TYPE_BOOLEAN);

        $this->db = MySQLConnection::getInstance();
        $this->db->connect();

        // Create tables.
        $this->createTableUsers();
        $this->createTableUserGroups();
        $this->createTableUserGroupsUser();

        // Add data to tables.
        $this->insertUserGroups();
        $this->insertUsers();
        $this->attachUsersToGroups();
    }

    /**
```

```
66      * Creates HENKILO -table to database.
67      *
68      * @throws \Exception
69      */
70      private function createTableUsers()
71      {
72           $this->db->query("DROP TABLE IF EXISTS HENKILO");
73
74           $query = "
75            CREATE TABLE HENKILO (
76                 ID              INT(11)         NOT NULL AUTO_INCREMENT,
77                 ETUNIMI         VARCHAR(100)    NOT NULL,
78                 SUKUNIMI        VARCHAR(100)    NOT NULL,
79                 PUHELINNUMERO   VARCHAR(10)     NOT NULL,
80                 PAINO           INT(11)         NOT NULL,
81                 PASSIIVINEN     INT(1)          NOT NULL DEFAULT 0,
82                 PRIMARY KEY (ID)
83               ) ENGINE=MyISAM;
84          ";
85
86           $this->db->query($query);
87
88           if ($this->addIndexes) {
89                $this->db->query("CREATE INDEX paino ON HENKILO (PAINO)");
90                $this->db->query("CREATE INDEX passiivinen ON HENKILO (PASSI-
91 IVINEN)");
92           }
93      }
94
95      /**
96      * Inserts users to HENKILO -table.
97      *
98      * @throws \Exception
99      */
100     private function insertUsers()
101     {
102          $query = "
103            INSERT INTO
104                HENKILO
105            SET
106                ETUNIMI       = :firstName,
107                SUKUNIMI      = :lastName,
108                PUHELINNUMERO = :phoneNumber,
109                PAINO         = :weight,
110                PASSIIVINEN   = :isPassive
111          ";
112
113          for ($i = 1; $i <= $this->userCount; $i++) {
114               $bindings = array(
115                    ":firstName"    => "Teppo $i",
116                    ":lastName"     => "Testaaja $i",
117                    ":phoneNumber" => rand(pow(10, 9), pow(10, 10) -1),
118                    ":weight"       => rand(50, 120),
119                    ":isPassive"    => $i % 100 === 0 ? 1 : 0
120               );
121
122               $this->db->query($query, $bindings);
123          }
```

```
124       }
125
126       /**
127       * Creates HENKILORYHMA_HENKILO -table.
128       *
129       * @throws \Exception
130       */
131       private function createTableUserGroupsUser()
132       {
133            $this->db->query("DROP TABLE IF EXISTS HENKILORYHMA_HENKILO");
134
135            $query = "
136             CREATE TABLE HENKILORYHMA_HENKILO (
137                  ID              INT(11) NOT NULL AUTO_INCREMENT,
138                  HENKILOID       INT(11) NOT NULL,
139                  HENKILORYHMAID  INT(11) NOT NULL,
140                  PRIMARY KEY (ID)
141               ) ENGINE=MyISAM;
142           ";
143
144            $this->db->query($query);
145
146            if ($this->addIndexes) {
147                 $this->db->query("CREATE INDEX henkiloId on HENKILORYHMA_HENKILO
148 (HENKILOID)");
149                 $this->db->query("CREATE INDEX henkiloRyhmaId on HENKILO-
150 RYHMA_HENKILO (HENKILORYHMAID)");
151            }
152       }
153
154       /**
155       * Creates HENKILORYHMA -table.
156       *
157       * @throws \Exception
158       */
159       private function createTableUserGroups()
160       {
161            $this->db->query("DROP TABLE IF EXISTS HENKILORYHMA");
162
163            $query = "
164             CREATE TABLE HENKILORYHMA (
165                  ID           INT(11)        NOT NULL AUTO_INCREMENT,
166                  NIMI         VARCHAR(255)   NOT NULL,
167                  PASSIIVINEN  INT(1)         NOT NULL DEFAULT 0,
168                  PRIMARY KEY (ID)
169               ) ENGINE=MyISAM;
170           ";
171
172            $this->db->query($query);
173
174            if ($this->addIndexes) {
175                 $this->db->query("CREATE INDEX passivinen ON HENKILORYHMA (passi-
176 ivinen)");
177            }
178       }
179
180       /**
181
```

```
     * Inserts user groups to database.
     *
     * @throws \Exception
     */
    private function insertUserGroups()
    {
        $query = "
          INSERT INTO
              HENKILORYHMA
          SET
              NIMI        = :groupName,
              PASSIIVINEN = :isPassive
       ";

        for ($i = 1; $i <= 10; $i++) {
            $bindings = array(
                ":groupName"    => "Ryhmä $i",
                ":isPassive"     => $i % 4 === 0 ? 1 : 0
            );

            $this->db->query($query, $bindings);
        }
    }

    /**
     * Attaches users to user groups.
     *
     * @throws \Exception
     */
    private function attachUsersToGroups()
    {
        $query = "
          INSERT INTO
              HENKILORYHMA_HENKILO
          SET
              HENKILOID      = :userId,
              HENKILORYHMAID = :groupId
       ";

        for ($i = 1; $i <= $this->userCount; $i++) {
            $userGroupCount = ($i % 10) + 1;

            for ($j = 1; $j <= $userGroupCount; $j++) {
                $bindings = array(
                    ":userId"   => $i,
                    ":groupId" => $j
                );

                $this->db->query($query, $bindings);
            }
        }
    }
}
```

K src/classes/commands/InitMongoDB.php

```php
<?php
/**
 * \classes\commands\InitMongoDB.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\commands;

use classes\database\MongoDBConnection;

/**
 * Class InitMongoDB
 *
 * @help Initializes MongoDB database for tests.
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
class InitMongoDB extends MongoDBCommand
{
     /**
     * Number of user that this script creates.
     *
     * @var int
     */
     private $userCount = 1000;

     /**
     * Whether to add indexes to database or not.
     *
     * @var bool
     */
     private $addIndexes = false;

     /**
     * Executes current command and returns result.
     *
     * @return mixed
     */
     public function benchmark()
     {
          $this->userCount = $this->askParameter("How many users do you want to
create");
          $this->addIndexes = $this->askParameter("Do you want to add indexes
[Y/n]", Command::TYPE_BOOLEAN);

          $this->db = MongoDBConnection::getInstance();
          $this->db->connect();
          $this->db->selectDB(MONGO_DATABASE_NAME);

          // Add user data.

```

```
56            $this->addUsers();
57        }
58
59        public function addUsers()
60        {
61            $this->db->selectCollection(MONGO_COLLECTION_NAME);
62            $this->db->getCollection()->drop();
63
64            if ($this->addIndexes) {
65                $this->db->ensureIndex(array("paino" => 1));
66                $this->db->ensureIndex(array("passiivinen" => 1));
67                $this->db->ensureIndex(array("henkiloryhmat" => 1));
68            }
69
70            $userGroups = array();
71
72            for ($i = 1; $i <= 10; $i++) {
73                $userGroups[$i] = array(
74                    "nimi"          => "Ryhmä $i",
75                    "passiivinen" => $i % 4 === 0
76                );
77            }
78
79            for ($i = 1; $i <= $this->userCount; $i++) {
80                $userGroupsCount = ($i % 10) + 1;
81                $groups = array();
82
83                for ($j = 1; $j <= $userGroupsCount; $j++) {
84                    $groups[$j] = $userGroups[$j];
85                }
86
87                $this->db->insert(
88                    array(
89                        "etunimi"         => "Teppo $i",
90                        "sukunimi"        => "Testaaja $i",
91                        "puhelinnumero" => rand(pow(10, 9), pow(10, 10) - 1),
92                        "paino"           => rand(50, 120),
93                        "passiivinen"    => $i % 100 === 0,
94                        "henkiloryhmat" => array_values($groups)
95                    )
96                );
            }
        }
    }
```

L src/classes/commands/BenchmarkMySQL1.php

```
1  <?php
2  /**
3   * \classes\commands\BenchmarkMySQL1.php
4   *
5   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
```

```
 * @version 1.0
 */
namespace classes\commands;

/**
 * Class BenchmarkMySQL1
 *
 * @help Fetches passive users that belongs to one or more passive user groups
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
class BenchmarkMySQL1 extends MySQLCommand
{
     /**
     * Default constructor for class.
     *
     * @throws \Exception
     */
     public function __construct()
     {
          parent::__construct();

          $this->query = "
            SELECT
              COUNT(*)
            FROM
              HENKILO as h
            JOIN
              HENKILORYHMA_HENKILO as hh ON h.ID = hh.HENKILOID
            JOIN
              HENKILORYHMA as hr ON hh.HENKILORYHMAID = hr.ID
            WHERE
              h.PASSIIVINEN = 1
            AND
              hr.PASSIIVINEN = 1;
         ";
     }
}
```

M src/classes/commands/BenchmarkMySQL2.php

```
<?php
/**
 * \classes\commands\BenchmarkMySQL2.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\commands;
```

```
9
10  /**
11   * Class BenchmarkMySQL2
12   *
13   * @help Fetches users whose weight is over 80
14   *
15   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
16   * @version 1.0
17   *
18   * @package classes\commands
19   */
20  class BenchmarkMySQL2 extends MySQLCommand
21  {
22       /**
23       * Default constructor for class.
24       *
25       * @throws \Exception
26       */
27       public function __construct()
28       {
29            parent::__construct();
30
31            $this->query = "
32            SELECT
33              *
34            FROM
35              HENKILO
36            WHERE
37              PAINO >= 80
38          ";
39       }
40  }
```

N src/classes/commands/BenchmarkMySQL3.php

```
1   <?php
2   /**
3    * \classes\commands\BenchmarkMySQL3.php
4    *
5    * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
6    * @version 1.0
7    */
8   namespace classes\commands;
9
10  /**
11   * Class BenchmarkMySQL3
12   *
13   * @help Fetches count of active users according to their weight.
14   *
15   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
16   * @version 1.0
17   *
```

```
 * @package classes\commands
 */
class BenchmarkMySQL3 extends MySQLCommand
{
     /**
     * Default constructor for class.
     *
     * @throws \Exception
     */
     public function __construct()
     {
          parent::__construct();

          $this->query = "
             SELECT
               COUNT(h.PAINO)
             FROM
               HENKILO as h
             WHERE
               h.PASSIIVINEN = 0
             GROUP BY
               h.PAINO
          ";
     }
}
```

O src/classes/commands/BenchmarkMongoDB1.php

```
<?php
/**
 * \classes\commands\BenchmarkMongoDB1.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\commands;

/**
 * Class BenchmarkMongoDB1
 *
 * @help Fetches passive users that belongs to one or more passive user groups
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
class BenchmarkMongoDB1 extends MongoDBCommand
{
     /**
     * Default constructor for class.
     *
```

```
25        * @throws \Exception
26        */
27       public function __construct()
28       {
29            parent::__construct();
30
31            $this->query = array(
32                 "passiivinen" => true,
33                 "henkiloryhmat" => array(
34                      '$elemMatch' => array(
35                           "passiivinen" => true
36                      )
37                 )
38            );
39       }
40  }
```

P src/classes/commands/BenchmarkMongoDB2.php

```
1   <?php
2   /**
3    * \classes\commands\BenchmarkMongoDB2.php
4    *
5    * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
6    * @version 1.0
7    */
8   namespace classes\commands;
9
10  /**
11   * Class BenchmarkMongoDB2
12   *
13   * @help Fetches users whose weight is over 80
14   *
15   * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
16   * @version 1.0
17   *
18   * @package classes\commands
19   */
20  class BenchmarkMongoDB2 extends MongoDBCommand
21  {
22       /**
23       * Default constructor for class.
24       *
25       * @throws \Exception
26       */
27       public function __construct()
28       {
29            parent::__construct();
30
31            $this->query = array(
32                 "paino" => array(
33                      '$gte' => 80
```

```
                )
            );
        }
}
```

Q src/classes/commands/BenchmarkMongoDB3.php

```php
<?php
/**
 * \classes\commands\BenchmarkMongoDB3.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\commands;

/**
 * Class BenchmarkMongoDB3
 *
 * @help Fetches count of active users according to their weight.
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands
 */
class BenchmarkMongoDB3 extends MongoDBCommand
{
    /**
     * Default constructor for class.
     *
     * @throws \Exception
     */
    public function __construct()
    {
        parent::__construct();

        $this->query = array(
            array(
                array(
                    '$match' => array(
                        'passiivinen' => false
                    )
                ),
                array(
                    '$group' => array(
                        '_id' => '$paino',
                        'count' => array(
                            '$sum' => 1
                        )
                    )
                )
```

```
                )
            );
        }
}
```

R src/classes/commands/interfaces/ExecutableCommand.php

```php
<?php
/**
 * \classes\commands\interfaces\ExecutableCommand.php
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 */
namespace classes\commands\interfaces;

/**
 * Interface for executable commands.
 *
 * @author Antti Rapa, <antti.j.rapa@student.jyu.fi>
 * @version 1.0
 *
 * @package classes\commands\interfaces
 */
interface ExecutableCommand
{
    /**
     * Executes current command and returns result.
     *
     * @return mixed
     */
    public function execute();

    /**
     * Does actual benchmarking calling execute function.
     *
     * @return array
     */
    public function benchmark();

    /**
     * Returns command name
     *
     * @return string
     */
    public function getName();

    /**
     * Returns help text for command.
     *
     * @return string
     */
```

```
46      public function help();
47  }
```