Vertaileva katsaus MariaDB:n asemaan MySQL:n kilpailijana

Oulun yliopisto
Tieto- ja sähkötekniikan tiedekunta
LuK-tutkielma
Janne Tahkola
15.4.2020

Tiivistelmä

MySQL:n alkuperäinen kehitystiimi hyödynsi samaa avointa lähdekoodia luodakseen MariaDB:n – MySQL:n kanssa binääriyhteensopivan ja monin tavoin parannellun vaihtoehdon. Tarkoituksena oli ensisijaisesti varmistaa MySQL:n lähdekoodin pysyminen avoimena sen siirryttyä Oraclen omistukseen. Nämä samasta koodikannasta veistetyt järjestelmät ovat viime vuosina kehittyneet eri suuntiin, mikä toimi niiden suosion ohella motivoivana tekijänä tämän vertailevan tutkielman laatimiseen.

Tutkielman tutkimuskysymykseksi asetettiin se, mitä perusteita organisaatiolla voisi olla MariaDB:n valinnalle MySQL:n sijaan. Sitä pohjustettiin tutkimalla relaatiotietokantoja sekä yleisesti että MySQL:n kautta. Lopuksi tutkimuskysymystä lähestyttiin tutkimalla järjestelmien yhteensopivuutta sekä vertailemalla niiden merkittävimpiä eroja. Pohjustuksessa ja vertailussa hyödynnettiin olemassa olevaa tutkimusta sekä järjestelmien omia verkkodokumentaatiota.

MariaDB tukee useampia tietokantamoottoreita sekä tarjoaa ilmaiseksi tiettyjä vain MySQL:n kaupallisista versioista löytyviä ominaisuuksia. Erot jäävät kuitenkin perusominaisuuksiksi luokiteltavien seikkojen osalta varsin pieniksi - järjestelmät poikkeavat toisistaan nykyään lähinnä esimerkiksi virtuaalisesti synkronisen replikoinnin kaltaisissa edistyneemmissä ominaisuuksissa. Näistä valtaosa kuuluu varsinaisten tietokantajärjestelmien sijaan yhtiöiden tarjoamiin kaupallisiin tuoteperheisiin, jotka eivät ole tämän tutkielman aihealuetta.

Ennen järjestelmän valintaa organisaation tulee siis peilata tarpeitaan joidenkin perusominaisuuksien lisäksi myös näihin kokonaisiin tuoteperheisiin lisenssivaihtoehtoineen. Huomioitavaa on myös se, kuinka järjestelmien yhteensopivuus heikkeni merkittävästi MySQL:n version 8.0 julkaisun myötä. Järjestelmästä toiseen vaihtaminen ei välttämättä suju enää ongelmitta, joten niiden vertailu ennen varsinaisia investointeja on aiempaa tärkeämpää.

Avainsanat
relaatiotietokannat, relaatiotietokantajärjestelmät, MySQL, MariaDB

Ohjaaja
Juha Iisakka

Sisällysluettelo

1. Johdanto

Datamäärien kasvaessa ja datan luonteen muuttuessa perinteisten relaatiotietokantojen rinnalle on noussut esimerkiksi NoSQL-tietokantoja, kuten MongoDB. Ne soveltuvat erityisesti strukturoimattoman datan käsittelyyn sekä skaalautuvuutta vaativiin tilanteisiin, kuten sosiaalisen median palveluihin tai tieteellisen datan varastoihin (Binani, Gutti ja Upadhyay, 2016). Binanin ja muiden (2016) mukaan ne eivät kuitenkaan pysty aina takaamaan tietokantatransaktioiden luotettavuutta relaatiotietokantojen tavoin, mistä syystä relaatiotietokannat ovat säilyttäneet kärkipaikkansa esimerkiksi pankkien ja sairaaloiden järjestelmissä.

Verrattain uutena vaihtoehtona ovat myös NewSQL-tietokannat, jotka pyrkivät tuomaan yhteen relaatiotietokantojen ACID-transaktiot sekä NoSQL-tietokantojen skaalautuvuuden (Binani ja muut, 2016). NewSQL-tietokantoja ovat esimerkiksi CockroachDB sekä Google Spanner. Ne ovat Binanin ja muiden (2016) mukaan kuitenkin vielä hyvin varhaisessa vaiheessa. Relaatiotietokannoilla sen sijaan on vuosikymmenten mittainen etumatka, mikä heijastuu niiden laajassa käytössä, tarjonnassa sekä tuessa ja dokumentaatiossa.

Relaatiotietokantoja tullaan siis suosimaan vielä toistaiseksi määrittelemättömään ajankohtaan asti ainakin sellaisissa käyttökohteissa, joissa vaaditaan luotettavia transaktioita. Kun tarve on rajattu relaatiotietokantoihin, on seuraavana haasteena sopivan tietokantajärjestelmän valinta. Monille se on hyvinkin helppoa – kokeneen tietokantaratkaisuja tarjoavan yrityksen, Oraclen tukema MySQL on muodostunut monelle oletusarvoiseksi valinnaksi.

Vuodesta 1995 kehityksessä ollut MySQL on hankkinut taakseen mittavan määrän puolesta puhujia. Sen päätyminen Oraclen kehitykseen vuonna 2009 ei kuitenkaan miellyttänyt Nymanin (2013) mukaan kaikkia. Tyytymättömyyttä tunsi kenties kaikista voimakkaimmin MySQL:n alkuperäinen ydinkehitystiimi. Osa heistä jätti MySQL:n kehittääkseen uutta tietokantajärjestelmää – MySQL:n lähdekoodiin pohjautuvaa MariaDB:tä (Nyman, 2013). Sen voidaan katsoa tähdänneen alun perin MySQL:n korvaajaksi ja se toikin mukanaan yhteensopivuuden lisäksi useita parannuksia. Sittemmin sisarusten yhteensopivuus on heikentynyt MariaDB:n kehittyessä omaan suuntaansa. Samalla MySQL on omaksunut monia vain MariaDB:stä aiemmin löytyneitä ominaisuuksia. Kehityksen myötä yhä uudelleen esiin nousevaksi kysymykseksi onkin muodostunut se, missä tilanteessa toinen on toista parempi valinta.

Tutkimuskysymykseksi on asetettu se, mitä perusteita tietokantajärjestelmästä päättävällä taholla voi olla MariaDB:n valinnalle MySQL:n sijaan. Sitä lähestytään vertailemalla järjestelmiä muiden tutkimusten sekä järjestelmien omien verkkodokumentaatioiden pohjalta. Pääpisteenä ovat tekniset ominaisuudet, mutta järjestelmien tulevaisuudennäkymien avaamiseksi myös niihin liittyvät organisaatiot ja yhteisöt huomioidaan. Tarkoituksena ei ole tarkastella jokaista järjestelmien välistä eroa, vaan löytää niistä merkittävimmät ja vastata tutkimuskysymykseen niiden pohjalta. Tarkastelun kohteena ovat pääasiassa MariaDB:n ja MySQL:n tuoreimmat vakaat GA-julkaisut (General Availability), MariaDB 10.4 sekä MySQL 8.0 (MariaDB, 2020f; MySQL, 2020e).

Tutkielman ensimmäisessä luvussa käsitellään relaatiotietokantoja ja relaatiotietokantajärjestelmiä erityisesti MySQL:n kautta sen vakiintuneen aseman johdosta. Tarkoituksena on hyödyntää olemassa olevia tutkimuksia tarjoamaan yleiskuva niiden toiminnasta sekä pohjustamaan toista lukua, jossa vastataan tutkimuskysymykseen tutustumalla MariaDB:n historiaan sekä tutkimalla sen nykyasemaa MySQL:n kilpailijana.

2. Relaatiotietokannat ja MySQL

Relaatiotietokannat syntyivät englantilaisen Edgar F. Coddin esiteltyä relaatiomallinsa vuonna 1970. Relaatiomalliin perustuvissa relaatiotietokannoissa data ja datan sisältämät suhteet esitetään riveistä ja sarakkeista koostuvina tauluina. Datan hallinnoimista helpottamaan luotiin myöhemmin erilaisia relaatiotietokantajärjestelmiä, jotka sisältävät tietokannan lisäksi myös muita ohjelmistoja. (Korth, Silberschatz & Sudarshan, 2011). Niissä säilytettävän datan manipulointiin käytetään yleisimmin SQL:ää (Structured Query Language) (Bhavsar, Mehta, Oza & Shah, 2018).

Helppokäyttöisyytensä vuoksi relaatiotietokannat syrjäyttivät nopeasti edeltäjänsä, kuten esimerkiksi hierarkiset tietokannat sekä verkkotietokannat. 2000-luvulla suosiotaan ovat kasvattaneet erityisesti avoimen lähdekoodin relaatiotietokantajärjestelmät, kuten MySQL. (Korth ja muut, 2011).

2.1 Relaatiomalli

Tauluja kutsutaan relaatiomallin yhteydessä myös relaatioiksi. Taulun riveistä puhutaan joskus monikkoina (eng. tuples) ja sarakkeista attribuutteina (eng. attributes). (Korth ja muut, 2011). Attributtien määrä kertoo relaation asteen; esimerkiksi taulu 1 on 4-asteinen relaatio.

Malli perustuu matemaattiseen relaatioon: relaatio R on joukkojen S_1, S_2, ..., S_n relaatio, mikäli se on joukko n-paikkaisia monikkoja, jossa kunkin monikon ensimmäinen alkio on joukosta S_1, toinen joukosta S_2 ja niin edelleen. (Codd, 1970).

etunimi	***sukunimi***	***syntymävuosi***	***henkilötunnus***
Mikko	Mikkonen	1982	110282-1234
Reino	Reinola	1991	201291-5678
Kaapo	Kaaponen	1985	300385-9010

Taulukko 1. Henkilötieto-relaation tauluesitys

Yllä oleva Henkilötieto-taulu kuvaa ihmisen henkilötietoja muodossa Henkilötieto = {etunimi, sukunimi, syntymävuosi, henkilötunnus}. Se on siis joukko 4-paikkaisia monikkoja.

Coddin (1970) mukaan n-asteisella relaatiolla R on viisi ominaisuutta:

1. Kukin rivi on n-asteinen R:n monikko
2. Rivien järjestyksellä ei ole väliä
3. Kukin monikko on yksilöitävissä
4. Sarakkeiden järjestyksellä ei ole väliä
5. Kukin sarake sisältää arvoja vain sen nimeä vastaavasta arvojoukosta

Lisäksi kaikkien arvojoukkojen tulee Coddin (1970) mukaan olla atomisia eli niiden yksikön tulee olla jakautumaton tai tyhjäarvo. Joukkojen alkiot sisältävät täten tasan yhden joukolle määriteltyä yksikköä vastaavan arvon. Esimerkiksi relaatiossa

Henkiötieto kukin joukon syntymävuosi alkio sisältää yhden numeerisen arvon. Tyhjäarvo (eng. null value) viittaa siihen, että arvoa ei tunneta tai se puuttuu.

2.1.1 Avaimet

Relaation monikkojen yksilöiminen hoidetaan relaatiomallissa avaimilla. Käytännössä tämä tapahtuu Korthin ja muiden (2011) mukaan monikoiden attribuuttien arvojen kautta - kaikkien saman relaation monikoiden tulee poiketa muista vähintään yhden attribuutin arvon osalta. Superavaimeksi kutsutaan sellaista yhden tai useamman attribuutin joukkoa, joka mahdollistaa monikon yksilöimisen muista relaation monikoista. Kandidaattiavain (eng. candidate key) puolestaan on sellainen superavain, jonka mikään osajoukko ei erikseen muodosta superavainta. Se on siis attribuuteiltaan minimoitu superavain.

Esimerkiksi kohdassa 2.1 esitellyssä Henkilötieto-taulussa yksi superavain olisi joukko a = {etunimi, sukunimi, henkilötunnus}. Joukko ei ole kandidaattiavain, koska sen osajoukko b = {henkilötunnus} on myös superavain – ja samalla myös kandidaattiavain. Osajoukko c = {etunimi, sukunimi} puolestaan ole superavain, koska tietokantaan voidaan tulevaisuudessa tallentaa useampi henkilö, joilla on samat etu- ja sukunimet.

Pääavain (eng. primary key) on relaation monikkojen yksilöintiä varten huolella valittu kandidaattiavain. Pääavaimen ei tulisi muuttua käytännössä koskaan. (Korth ja muut, 2011). Esimerkiksi Henkilötieto-taulun kohdalla voisimme valita pääavaimeksi henkilötunnuksen, mutta koska mahdollisilla ulkomaalaisilla asiakkailla ei todennäköisesti ole suomalaista henkilötunnusta, ei se välttämättä riitä. Tällaisten tilanteiden takia päädytään Korthin ja muiden (2011) mukaan usein luomaan oma, vain pääavaimeksi tarkoitettu attribuutti.

Viiteavainta (eng. foreign key) käytetään, kun relaatiosta halutaan viitata toiseen (Korth ja muut, 2011). Olennainen osa henkilötietoja on yleensä kotipaikkakunta. Ensimmäinen vaihtoehto olisi tallentaa tieto Henkilötieto-tauluun suoraan omana attribuuttinaan. Paikkakunnat ovat kuitenkin verrattain pysyviä, joten voimme minimoida esimerkiksi käyttäjien kirjoitusvirheet ylläpitämällä erillistä Paikkakunta-taulua, johon kosketaan harvemmin. Tällöin viittaisimme kyseisen taulun pääavaimeen Henkilötieto-tauluun lisäämästämme “paikkakunta”-attribuutista.

2.1.2 Rajoitteet ja normalisointi

Tietokantamalli (eng. database schema) on tietokantasuunnittelun tulos. Se on relaation looginen kuvaus, joka määrittelee relaation avaimet, rajoitteet (eng. constraints) sekä attribuutit tietotyyppeineen. Tuotettu tietokantamalli täyttää vaatimukset siitä, millaisia luomis-, haku-, päivitys- ja poistamisoperaatiota datalle tullaan suorittamaan. (Korth ja muut, 2011).

Fosterin ja Godbolen (2016) mukaan relaatiotietokantasuunnittelun perusteisiin kuuluvat kaksi toisistaan riippuvaa sääntöä: entiteettieheysrajoite (eng. entity integrity constraint) sekä viite-eheysrajoite (eng. referential integrity constraint). Ensimmäisellä he tarkoittavat sitä, että pääavain ei saa koskaan sisältää attribuutteja, jotka voivat saada tyhjäarvoja. Jälkimmäinen puolestaan asettaa viiteavaimelle rajoitteen, jonka mukaan sen arvon tulee aina olla joko tyhjäarvo tai viitata kohdetaulun olemassa olevan rivin pääavaimeen.

Normalisointi on yksi relaatiotietokantojen suunnittelumetodi. Se varmistaa, että tietokantamalli täyttää esimerkiksi entiteettieheysrajoitteen, välttää tarpeetonta datan toistoa sekä mahdollistaa datan tehokkaan varastoinnin ja haun. Normalisointi tuottaa relaatioita, jotka ovat normaalimuodossa (eng. normal form). (Foster & Godbole, 2016). Mikäli relaation kaikkien attribuuttien arvojoukot ovat atomisia, sanotaan relaation olevan ensimmäisessä normaalimuodossa. Tavallisimmin käytettyjä normaalimuotoja on kuusi. (Korth ja muut, 2011).

2.2 Tietokantajärjestelmät

Korthin ja muiden (2011) mukaan tietokantajärjestelmä (eng. database-management system/DBMS) tarkoittaa tietokantaa eli kokoelmaa toisiinsa liittyvää dataa sekä ohjelmistoja kyseiseen dataan käsiksi pääsemiseksi. Tietokantajärjestelmän tarkoitus on tarjota kätevä ja tehokas tapa datan tallentamiseen ja hakemiseen tietokannasta. (Korth ja muut, 2011). Relaatiotietokantajärjestelmä (eng. relational database-management system/RDBMS) on luonnollisesti tietokantajärjestelmä, joka hyödyntää relaatiotietokantoja.

Tämän hetken suosituimpia relaatiotietokantajärjestelmiä on MySQL. Sen julkaisi vuonna 1995 ruotsalainen MySQL Ab, mutta nykyään sen omistaa Oracle (Nyman, 2013). MySQL:ää käyttävät järjestelmissään muun muassa sosiaalisen median jätti Facebook, elokuvien ja sarjojen suoratoistopalvelu Netflix, maksujenvälitysjärjestelmä PayPal sekä Yhdysvaltojen laivasto (Bhavsar ja muut, 2018). Ohjelmistosivusto Stack Overflow:n vuonna 2019 järjestämään Developer Survey -kyselyyn vastanneista 54 prosenttia kertoi käyttävänsä MySQL:ää. Toisena tuli PostgreSQL 34 prosentilla ja kolmantena Microsoft SQL Server 33 prosentilla. Kyselyyn osallistui lähes 90 tuhatta henkilöä 179:stä maasta.

MySQL:n suosion takana on useita seikkoja. Oracle on hallinnut tietokantamarkkinoita Fosterin ja Godbolen (2016) mukaan jo 20 vuotta, mistä syystä sen nimi herättää luottamusta erityisesti suurissa organisaatioissa. MySQL on myös alustariippumaton, sillä se toimii esimerkiksi Windowsilla, MacOS:llä, Linuxilla, Solariksella ja OS2:lla (Bhavsar ja muut, 2018; Foster ja Godbole, 2016). Lisäksi Foster ja Godbole (2016) huomauttavat, kuinka Oracle on jättänyt MySQL:n palvelinohjelmiston lähdekoodin avoimeksi, mikä saa myös akateemiset piirit sekä pienyritykset suosimaan sitä.

2.2.1 SQL, DML ja DDL

Yleisin tietokantajärjestelmien tarjoama rajapinta datan operointiin on Korthin ja muiden (2011) mukaan jo aiemmin mainittu ohjelmointikieliin verrattavissa oleva SQL. Sen tarjoamien komentojen avulla voidaan luoda tauluja sekä manipuloida niihin tallennettavaa dataa. Käytännössä se voidaan jakaa vielä DML (data-manipulation language) ja DDL (data definition language) -käsitteisiin. DML viittaa niihin SQL:n osiin, joilla datalle suoritetaan esimerkiksi lukuoperaatioita, kun taas DDL keskittyy siihen, miten data kuvataan tauluina.

```
CREATE TABLE IF NOT EXISTS henkilotieto (
    etunimi VARCHAR(255) NOT NULL,
    sukunimi VARCHAR(255) NOT NULL,
    syntymavuosi INT NOT NULL,
    henkilotunnus VARCHAR(255) NOT NULL
) ENGINE = INNODB;
```

Kuva 1. DDL-lauseke, joka määrittelee ja luo Henkilötieto-taulun

Kuva 1 esittelee DDL-lausekkeen, joka määrittelee ja luo Taulukko 1:n mukaisen relaation. Tekstimuotoinen data tallennetaan VARCHAR-tietotyyppinä ja numeromuotoinen kokonaislukua tarkoittavana INT-tietotyyppinä. Lauseke määrittelee myös ettei mikään attribuutti hyväksy tyhjäarvoja.

```
INSERT INTO henkilotieto (etunimi, sukunimi, syntymavuosi, henkilotunnus)
VALUES ('Mikko', 'Mikkonen', 1982, '110282-1234');
```

Kuva 2. DML-lauseke, joka lisää rivin Henkilötieto-tauluun

Tauluun lisätään rivi Kuva 2:n mukaisella DML-lausekkeella. Tietotyypit vastaavat määrittelyä ja jokaiselle attribuutille on annettu arvo.

```
SELECT etunimi, sukunimi, syntymavuosi, henkilotunnus FROM henkilotieto;
SELECT * FROM henkilotieto;
```

Kuva 3. Kaksi SQL-kyselyä, jotka hakevat taulusta kaikkien sarakkeiden arvot

Lopulta taulusta haetaan kaikki sarakkeet SELECT-lausekkeella. Lauseke ei ole DDL eikä DML-lauseke, sillä se ei luo uusia määrittelyjä eikä tee muutoksia. Tähtisymboli * on lyhennetty tapa hakea kaikkien sarakkeiden arvot sen sijaan, että niihin viitattaisiin yksitellen, kuten sen yläpuolella olevassa lausekkeessa.

DDL-kyselyiden pohjalta tietokantajärjestelmä luo DML:llä päivitettäviä datatiedostoja, indeksejä ja datakirjastoja (eng. data dictionary). Datatiedostot sisältävät itse tietokannan tietokantamalleineen, kun taas datakirjastot säilyttävät siihen liittyvää metatietoa. Metatiedolla tarkoitetaan esimerkiksi tietokannan taulujen ja taulujen attribuuttien nimiä sekä mahdollisia rajoitteita. (Korth ja muut, 2011). Indekseihin perehdytään kohdassa 2.2.6.

2.2.2 CAP-teoreema

Hajautetussa tietokannassa (eng. distributed database) data sijaitsee useammissa sijainneissa (eng. site), jotka voivat olla kaukanakin toisistaan. Hajauttamalla voidaan parantaa järjestelmän eheyttä (eng. consistency), saatavuutta (eng. availability) sekä osituksen sietokykyä (eng. partition tolerance). (Korth ja muut, 2011). Näillä kolmella piirteellä on merkittävä rooli hajautettujen tietokantojen suunnittelussa, mistä syystä niihin viitataan usein yhteistermillä CAP. Niihin liittyvät myös hajautettujen tietokantojen suunnittelun kaksi eri lähestymistapaa: replikointi ja ositus.

Brewerin (2012) CAP-teoreeman mukaan hajautettu tietokantajärjestelmä voi täyttää aiemmin mainituista kolmesta piirteestä (eheys, saatavuus ja osituksen sietokyky) kerrallaan vain kaksi. Mikäli kahden sijainnin välinen yhteys katkeaa eli tapahtuu ositus, järjestelmä joutuu valitsemaan saatavuuden ja eheyden väliltä. Jos toiseen sijaintiin voidaan kirjoittaa, toinen ei saa tietoa uudesta datasta – tällöin on valittu

saatavuus. Mikäli halutaan painottaa eheyttä, kirjoittamista ei voida sallia kummassakaan sijainnissa. Brewerin (2012) mukaan saatavuus ja eheys voidaan säilyttää vain silloin, kun osituksen sietokyvystä ei välitetä. Käytännössä valinta on kuitenkin aina saatavuuden ja eheyden välillä, mikäli järjestelmä on hajautettu laajalle alueelle.

2.2.3 Replikointi

Replikoinnissa (eng. replication) kustakin relaatiosta säilytetään identtisiä kopioita kahdella tai useammalla palvelimella. (Korth ja muut, 2011). Sen merkittävänä etuna on saatavuuden paraneminen; jos joku palvelimista menee epäkuntoon, on data edelleen luettavissa toiselta palvelimelta. Varjopuolena taas on päivitysoperaatioiden suorituskyky - kopiot täytyy pitää ajan tasalla, jotta tietokannan eheys säilyy.

Yksi replikointimetodi on asynkroninen master-slave-replikointi, jota suositaan Bhavsarin ja muiden (2018) mukaan yleensä ainakin MySQL:n tapauksessa. Siinä yksi palvelimista toimii pääpalvelimena (eng. master) ja muut sen alaisina (eng. slaves). Alipalvelimet lukevat ja kopioivat tietokantatapahtumat pääpalvelimen ylläpitämästä lokitiedostosta eli tieto kulkee aina pääpalvelimen kautta. Alipalvelimet ovat yleensä vain luettavissa, jotta datan eheys pää- ja alipalvelimien välillä voidaan taata, mikä tuo ilmi ratkaisun pullonkaulan – vain yhden kirjoituksia hyväksyvän palvelimen. Asynkronisuus puolestaan tuo kirjoitusoperaatioihin viivettä, koska tapahtumat luetaan vain määritellyin väliajoin. Niiden näkyminen alipalvelimissa pyritään takaamaan, mutta ajankohtaa ei. Master-slave saattaa myös johtaa datan katoamiseen pääpalvelimen kaatuessa.

Replikointi voi olla myös semisynkronista tai täysin synkronista. Semisynkronisessa replikoinnissa pääpalvelin estää tulevat transaktiot odottaessaan vähintään yhden alaisen kuittausta edellisen transaktion onnistuneesta saapumisesta (Bhavsar ja muut, 2018). Tämä takaa sen, että transaktio näkyy heti vähintään yhdellä alipalvelimella, mutta aiheuttaa luonnollisesti viivettä antamalla kirjoitusoperaatioille mahdollisuuden kasaantua. Synkroninen replikointi tarkoittaa, että pääpalvelin odottaa jokaisen alipalvelimen kuittausta (MariaDB, 2020d). Sen käyttö ei ole suorituskykysyistä suositeltavaa eikä esimerkiksi MySQL tarjoa täysin synkronista master-slave -replikointia ollenkaan. Semisynkronisen ja synkronisen välimalleja on kuitenkin olemassa, kuten esimerkiksi MySQL:n Group Replication ja MariaDB:n Galera Cluster. Näitä kutsutaan joissakin yhteyksissä virtuaalisesti synkronisiksi ratkaisuiksi (MariaDB, 2020d).

2.2.4 Ositus eli fragmentaatio

Osituksessa (eng. partitioning/fragmentation), relaatio jaetaan pienempiin palasiin, fragmentteihin, jotka tallennetaan eri palvelimille. Fragmentit luodaan niin, että alkuperäinen relaatio on edelleen muodostettavissa niiden pohjalta. (Korth ja muut, 2011). Tällainen tarkoituksellinen ositus on siis eri asia kuin CAP-teoreeman ositus, jossa palvelimet menettävät keskinäisen kommunikointikykynsä.

Bhavsarin ja muiden (2018) mukaan ositus voidaan tehdä horisontaalisesti tai vertikaalisesti. Horisontaalisessa osituksessa taulu ositetaan rivien perusteella, kun taas vertikaalisessa osituksessa eri palvelimille jaetaan taulun sarakkeet. MySQL tukee heidän mukaansa vain horisontaalista ositusta. Sen avulla esimerkiksi suomalaisten asiakkaiden tiedot voidaan säilyttää Suomessa, ja amerikkalaisten Yhdysvalloissa. Tämä parantaa paikallisten kyselyiden suorituskykyä. Merkittäviä

suorituskykykustannuksiakin kuitenkin syntyy, mikäli Suomesta haetaan vaikkapa amerikkalaisten tietoja.

2.2.5 Transaktiot ja lukot

ACID-ominaisuudet kuuluvat Pokornyn (2013) mukaan relaatiotietokantajärjestelmien, tai tarkemmin sanottuna niiden suorittamien transaktioiden perusominaisuuksiin. Tietokantatransaktio on yhden tai useamman tietokantaoperaation muodostama kokonaisuus, jossa suoritetaan joko kaikki sen sisältämät operaatiot tai ei mitään niistä (Korth ja muut, 2011). ACID-ominaisuudet toteutuvat, kun transaktiot ovat atomisia, eheitä, eristyneitä ja pysyviä. Korth ja muut (2011) määrittelevät ominaisuudet seuraavasti:

- Atomisuus (eng. atomicity): Tietokantaan tallennetaan joko transaktion kaikkien operaatioiden tulokset tai ei mitään.
- Eheys (eng. consistency): Tietokannan eheys säilyy eli esimerkiksi kaikki määritellyt sille rajoitteet täyttyvät.
- Eristyneisyys (eng. isolation): Muut samaan aikaan ajettavat kyselyt eivät vaikuta transaktioon.
- Pysyvyys (eng. durability): Onnistuneesti suoritetun transaktion tulos näkyy tietokannassa myös järjestelmän virhetilanteissa.

Otetaan ominaisuuksien toiminnasta käytännön esimerkkinä Razzolin (2014) antama verkkokauppa. Verkkokaupan tietokannassa on taulut tilauksille sekä tuotteille ja niiden määrille. Kun asiakas tilaa sieltä farkut, tilaus tallennetaan oikeaan tauluun ja kyseisten farkkujen saldoa vähennetään. Jos järjestelmä syystä tai toisesta kaatuu operaatioiden välillä ja tilaus tallennetaan, mutta saldoa ei vähennetä, ei tietokanta ole enää eheä. Sisällyttämällä operaatiot samaan transaktioon kumpikaan operaatio ei ongelmatilanteessa heijastu tietokantaan. Järjestelmän ei myöskään tarvitse kaatua eheyden menettämiseksi; ilman transaktiota toinen asiakas voi ostaa saman tuotteen ennen kuin saldoa on ehditty vähentää. Tällöin ensimmäisen asiakkaan tilaus menisi läpi, mutta hän jäisi ilman tuotetta. Tämäkin tilanne vältetään transaktiolla, koska ne ovat toisistaan eristettyjä.

Käytännössä tietokantajärjestelmät toteuttavat transaktioiden ACID-ominaisuudet lukoilla (Razzoli, 2014). Esimerkiksi MySQL tukee sisäisiä ja ulkoisia lukkoja (internal and external locks). Nimitykset tulevat niiden hallinnointiin osallistuvien osapuolten perusteella - MySQL:n palvelinohjelmisto on yksin vastuussa sisäisistä lukoista, kun taas ulkoisten lukkojen hallinnoimiseen osallistuu muitakin ohjelmistoja.

Sisäisten lukkojen kohdalla lukot voidaan jakaa rivi- ja taulutason lukkoihin (row and table level locks). (Bhavsar ja muut, 2018). Aiemmin mainitussa verkkokaupan käyttötilanteessa ensimmäisen asiakkaan aloittama transaktio lukitsisi tarvittavat MySQL-taulut tai niiden rivit siten, että toinen asiakas joutuu odottamaan transaktion päättymistä. Rivitason lukkojen hyötynä on luonnollisesti yhtäaikaiset kirjoitusoperaatiot, mikäli ne kohdistuvat eri riveille. Taulutason lukot puolestaan ovat nopeampia ja vaativat vähemmän muistia, koska yhdellä taululla on kerrallaan aina vain yksi lukko (Bhavsar ja muut, 2018).

Tietokantaan kohdistuvan operaation luonteesta riippuen lukot voidaan Razzolin (2014) mukaan jakaa edelleen jaettuihin ja eksklusiivisiin lukkoihin (eng. shared and exclusive locks). Näitä kutsutaan myös luku- ja kirjoituslukoiksi (Bhavsar ja muut, 2018). Rivitason jaettu lukko asetetaan riville siitä luettaessa, jolloin se estää muita yhteyksiä

kirjoittamasta riville. Kirjoittaessa puolestaan käytetään eksklusiivista lukkoa, joka estää rivin lukemisen.

2.2.6 Indeksit

Kun tietokannasta halutaan hakea vain tiettyjä rivejä, on tarpeetonta ja tehotonta käydä läpi niitä kaikkia. Rajoitettuja kyselyitä varten tietokantataulujen attribuutit voidaan indeksoida. Indeksit ovat erillisiä tiedostoja, joita voidaan Korthin ja muiden (2011) mukaan verrata vaikkapa kirjojen indekseihin - kukin siellä listattu käsite viittaa suoraan niille sivuille, joissa käsite esiintyy. Tietokantojen yhteydessä ne ovat tietorakenteita, joihin tallennetaan avaimia osoittamaan varsinaisiin taulun riveihin. Nämä avaimet, kuten esimerkiksi kohdassa 2.1.1 käsitelty pääavain, vastaavat jotain taulun attribuuttia, ja niitä kutsutaan hakuavaimiksi (eng. search key). (Korth ja muut, 2011). MySQL indeksoi pääavaimet automaattisesti (Bhavsar ja muut, 2018). Indeksi järjestetään, jonka jälkeen sille voidaan suorittaa binäärihakuja. Hakuavain voi sisältää myös useampia attribuutteja, jolloin avainta kutsutaan komposiittiavaimeksi (eng. composite key). (Korth ja muut, 2011).

Indeksit olisi hyvä pitää mahdollisimman pieninä, jotta ne mahtuvat kokonaisuudessaan työmuistiin eikä niitä tarvitse lukea kovalevyltä. Tarpeettomat indeksit myös luonnollisesti hidastavat järjestelmän toimintaa, sillä suuresta joukosta hakeminen on hitaampaa, minkä lisäksi indeksit täytyy pitää jatkuvasti ajan tasalla. (Korth ja muut, 2011). Ennen indeksointia tulisi siis miettiä, mitkä taulun attribuuteista ovat sellaisia, joiden avulla taulun rivejä tullaan useimmiten hakemaan.

2.2.7 Tietokantamoottorit

Tietokantamoottori (eng. database engine/storage engine) on Razzolin (2014) mukaan tietokantajärjestelmän komponentti, joka toteuttaa datan käsittelyn fyysisellä tasolla. Esimerkiksi datan käsittelyoperaatiot, indeksit, transaktiot ja lukot, välimuistit sekä viiteavaimet ovat tietokantamoottorin vastuulla (Bhavsar ja muut, 2018). Tietokantamoottoreita on useita eri käyttötarkoituksiin sopivia. Moottori määritellään kullekin taululle taulun luonnin yhteydessä. Esimerkkejä tietokantamoottoreista ovat lukupainotteisiin käyttötarkoituksiin sopiva MyISAM, työmuistin ylittävien datajoukkojen käsittelyyn optimoitu TokuDB sekä yleiskäyttöinen InnoDB. (Razzoli, 2014).

Mikäli tietokanta halutaan osittaa, on Bhavsarin ja muiden (2018) mukaan muistettava, että ositetun taulun kaikkien fragmenttien tulee käyttää samaa tietokantamoottoria. Eri taulut samassa ositetussa tietokannassa voivat kuitenkin käyttää eri moottoreita.

InnoDB on valittu esimerkiksi MySQL:n oletustietokantamoottoriksi. Se on ollut MySQL:n tavoin Oraclen kehityksessä vuodesta 2005, jolloin se ostettiin suomalaiselta Innobase Oy:ltä. Se tukee Bhavsarin ja muiden (2018) mukaan muun muassa ACID-transaktioita sekä rivitason lukkoja, ja on yksi tekijä MySQL:n suosion takana. InnoDB:n lisäksi MySQL tukee myös esimerkiksi MyISAM ja memory -tietokantamoottoreita.

InnoDB tuo mukanaan myös tiettyjä rajoituksia: yhdelle taululle voidaan määritellä enimmillään 64 indeksiä, komposiittiavain voi sisältää enintään 16 attribuuttia ja yhden taulun maksimikoko on 256 teratavua. Lisäksi esimerkiksi viiteavaimia ja ositusta ei voida käyttää yhtä aikaa. (Bhavsar ja muut, 2018).

3. MariaDB

MariaDB on entisten MySQL:n kehittäjien luoma, siihen pohjautuva sekä sen kanssa tiettyihin versioihin asti binääriyhteensopiva tietokantajärjestelmä (Bartholomew, 2012). Teoriassa MySQL voidaan siis missä tahansa järjestelmässä korvata suoraan MariaDB:llä, ja yhteensopivuus pidettiinkin pitkään hyvänä. Nykyään asia on monimutkaisempi. MariaDB syntyi korvaamaan MySQL:n sen alkuperäisten kehittäjien ja uusien omistajien näkemysten erkaantuessa. Sen reilun kymmenen vuoden olemassa olon aikana se on kuitenkin kehittynyt omaan suuntaansa ja nykyään se on ennemminkin kilpailija kuin korvaaja.

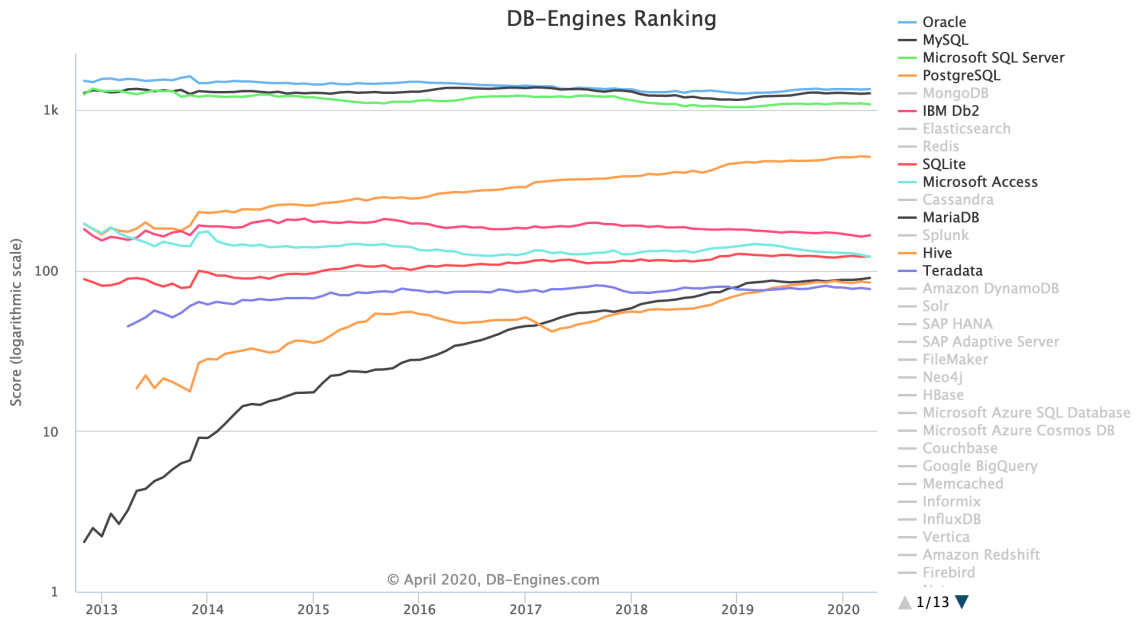
3.1 Historia

Sun Microsystemsin ostettua MySQL Ab:n vuonna 2008, Michael “Monty” Widenius, yksi yhtiön perustajajäsenistä, huolestui ohjelmiston tulevaisuudesta. MySQL Ab:n alkuperäinen osakassopimus kielsi ohjelmiston lisenssin muuttamisen ilman Wideniuksen ja yhtiön toisen perustajajäsenen, David Axmarkin lupaa, suojaten ohjelmistoa esimerkiksi kaupallisilta laajennuksilta. Kun Sun Microsystems osti MySQL Ab:n vuonna 2008, kyseinen sopimuksen kohta mitätöitiin. Widenius katsoi, että MySQL tuli pitää täysin avoimena, mutta kaupan myötä hän menetti lisensointiin liittyvän päätösvaltansa. Hän jätti Sun Microsystemsin syksyllä 2008 ja perusti Monty Program Ab:n, vieden valtaosan MySQL:n ydinkehitystiimistä mukanaan. (Nyman, 2013; Razzoli, 2014).

Monty Program Ab julkaisi MySQL:n avoimeen koodikantaan perustuvan MariaDB:n vuonna 2009. Julkaisu tapahtui samana päivänä, kun Oracle osti Sun Microsystemsin. (Pearce, 2013). Widenius katsoi, että Oraclen omien tietokantatuotteiden ja MySQL:n välillä oli eturistiriita, minkä takia hän pelkäsi Oraclen joko ajavan MySQL:n alas tai muuttavan sen liiketoimintamallin kaupalliseksi, tehden osasta sen lähdekoodia yksityistä. (Nyman, 2013).

Vuonna 2013 Monty Program Ab sekä MariaDB-tavaramerkki siirtyivät SkySQL:n omistukseen, joka vaihtoi seuraavana vuonna nimensä MariaDB Corporation Ab:ksi. MariaDB Corporation vastaa erityisesti MariaDB:n kaupallisesta puolesta sekä teknisestä tuesta ja konsultoinnista. MariaDB:n avoimien osien kehitys tapahtuu nykyään MariaDB Corporationin, MariaDB Foundationin sekä yhteisön avulla. MariaDB Foundation on yleishyödyllinen järjestö, jonka tarkoitus on mm. auttaa yhteisön uusia jäseniä MariaDB:n kehityksessä, pitää MariaDB:n dokumentaatio ajan tasalla sekä tuoda MariaDB:lle näkyvyyttä esimerkiksi puhumalla tapahtumissa. (MariaDB Foundation, 2020).

Vuoden 2013 artikkelissaan, Pearce mainitsee DB-Engines -sivuston suosituimpien tietokantaohjelmistojen listauksen. Vuonna 2013 MySQL oli sen mukaan kolmanneksi suosituin Oraclen ja Microsoft SQL Serverin jälkeen. MariaDB:n sijoitus oli artikkelin mukaan tällöin 35. Tällä hetkellä MySQL on toiseksi suosituin Oraclen jälkeen, kun taas MariaDB on sijalla 12. (DB-Engines, 2020). Stack Overflow:n vuoden 2019 Developer Surveyssä MariaDB tuli seitsemänneksi – 16,5 prosenttia vastaajista kertoi käyttävänsä sitä.



Kuva 4. Kymmenen suosituinta relaatiotietokantajärjestelmää huhtikuussa 2020 (DB-Engines, 2020).

3.2 Arkkitehtuuri

MariaDB-asennus on oletetusti hyvin samanlainen MySQL:n kanssa ja koostuu Bhavsarin ja muiden (2018) mukaan useista eri ohjelmistoista. Varsinainen palvelinohjelmisto on molempien kohdalla nimeltään mysqld. Se on siis taustaprosessi eli daemon, jonka ollessa käynnissä asiakas (eng. client) voi yhdistää siihen esimerkiksi mysql-komentoriviohjelmistolla. Päivitysoperaatioita varten asennus sisältää mysql_upgrade-ohjelmiston, ja varmuuskopiointiin tarjolla on mysqldump.

Palvelinohjelmisto sisältää Razzolin (2014) mukaan muun muassa autentikaattorin, parserin, optimoijan (eng. optimizer) ja tietokantamoottorit. Kun asiakas yhdistää MariaDB:hen, sille suoritetaan autentikaatio, jonka onnistuessa asiakas voi lähettää MariaDB:lle SQL-kyselyitä. Kyselyt ohjataan parserille, jonka tehtävä on ymmärtää ne ja tarkistaa, onko asiakkaalla oikeuksia suorittaa operaatioita. (Razzoli, 2014). Tämän jälkeen vuorossa on kyselyiden optimointi (eng. query optimization), jossa optimoija päättää tehokkaimman tavan niiden suoritukseen (Korth ja muut, 2011). Optimointi sisältää muun muassa päätökset siitä, mitä indeksejä kyselyiden suorittamiseksi kannattaa käyttää. Optimoinnin tulos on kyselysuunnittelma (eng. query plan) tai suoritusstrategia (eng. execution strategy). Tietokantamoottori suorittaa sen perusteella kyselyiden sisältämät luku- ja kirjoitusoperaatiot kovalevylle (Razzoli, 2014).

3.3 Yhteensopivuus MySQL:n kanssa

MariaDB käyttää oletustietokantamoottorinaan InnoDB:tä MySQL:n tavoin. Aiemmin se käytti InnoDB:n parannelluksi versioksi kuvailtua XtraDB:tä, mutta siirtyi takaisin InnoDB:hen versiossa 10.2. Syyksi kerrottiin InnoDB:n ottamat harppaukset, joiden johdosta XtraDB:n pitäminen yhteensopivana MySQL:n tietokantamoottorin kanssa ei enää tuntunut kannattavalta. MariaDB kuitenkin jatkaa XtraDB:stä löytyvien parannusten lisäämistä omaan moottoriinsa, minkä takia implementaatio poikkeaa nykyään jo merkittävästi MySQL:stä. (Gilfillan, 2017). Mikäli tarkoituksena on korvata

olemassa olevassa tietojärjestelmässä käytössä oleva MySQL MariaDB:llä tai toisin päin, tulee yhteensopivuuden kanssa olla tarkkana.

MariaDB:n (2020a) dokumentaation mukaan tietyt MariaDB:n ja MySQL:n versiot luetaan toisiaan vastaaviksi, jolloin yhteensopivuus on hyvä. MariaDB:n versiot 5.1, 5.2 ja 5.3 vastaavat MySQL 5.1:tä, ja versio 5.5 MySQL 5.5:tä. Esimerkiksi tiedostonimet ovat identtisiä, ja data- sekä taulumääritykset binääriyhteensopivia. Myös MySQL:n connectorit esimerkiksi Javalle ja .NET:lle toimivat suoraan MariaDB:n kanssa. Erityisesti näiden versioiden kohdalla MySQL:n korvaaminen MariaDB:llä ja toisin päin on suoraviivaista – poista toinen ja asenna toinen tilalle.

MariaDB:n versiot 10.0 ja 10.1 puolestaan ovat “rajallisesti” yhteensopivia MySQL 5.6:n kanssa, ja versiot 10.2, 10.3 sekä 10.4 MySQL 5.7:n kanssa. Rajallisella yhteensopivuudella dokumentaatio viittaa oletustietokantamoottoriin eli InnoDB:en, joka on näiden versioiden kohdalla edelleen yhteensopiva MySQL:n kanssa. Muita eroja kuitenkin löytyy. Esimerkiksi GTID (Global Transaction Identifier) ei ole yhteensopiva MariaDB 10.3:n ja MySQL 5.7:n välillä, mistä johtuen MySQL-palvelin ei voi toimia MariaDB-palvelimen alaisena replikointiympäristössä (MariaDB, 2020b).

Master→ / Slave ↓	MariaDB-5.5	MariaDB-10.1	MariaDB-10.2	MariaDB-10.3	MariaDB-10.4	MySQL-5.6	MySQL-5.7	MySQL-8.0
MariaDB-5.5	✅	⛔	⛔	⛔	⛔	⛔	⛔	⛔
MariaDB-10.1	✅	✅				✅		
MariaDB-10.2	✅	✅	✅			✅	✅	
MariaDB-10.3	✅	✅	✅	✅		✅	✅	
MariaDB-10.4	✅	✅	✅	✅	✅	✅	✅	
MySQL-5.6						*	*	*
MySQL-5.7						*	*	*
MySQL-8.0						*	*	*

- ✅: This combination is supported.
- ⛔: This combination is **not** supported.
- *: MariaDB can't make any claims about MySQL-only combinations. Refer to the documentation for the specific MySQL version to determine supported combinations.

Kuva 5. Replikoinnin yhteensopivuus MariaDB:n ja MySQL:n eri versioiden välillä (MariaDB, 2020a).

MySQL:n uusin vakaa GA-julkaisu on vuonna 2018 julkaistu 8.0, jonka kanssa asiat mutkistuvat. Ennen MySQL 8.0:aa, tietokantaobjekteista tallennettiin indeksien ja attribuuttityyppien kaltaista metatietoa erillisiin tiedostoihin ja ei-transaktionaalisiin tauluihin. Tätä tietokokonaisuutta kutsutaan kohdassa 2.2.1 mainituksi datakirjastoksi ja sitä hyödynnetään SQL-kyselyiden suorittamisessa. MySQL 8.0 toi mukanaan uuden datakirjastototeutuksen, jonka myötä metatieto tallennetaan suoraan tietokantaan erillisten tiedostojen sijaan. Se parantaa esimerkiksi turvallisuutta ja suorituskykyä kiertämällä tiedostojärjestelmän haavoittuvuudet ja mahdollistamalla datan säilyttämisen nopeammassa työmuistissa. (Bhavsar ja muut, 2018). Samalla se kuitenkin rikkoo yhteensopivuuden aiempien versioiden kanssa eikä MySQL 8.0 ole enää binääriyhteensopiva MariaDB:n tai edes aiempien MySQL-versioiden kanssa.

3.4 Merkittävimmät erot MySQL:ään

Koska MariaDB:n ja MySQL:n uusimpien versioiden väliltä toiseen vaihtaminen ei ole enää yhtä virtaviivaista kuin ennen, on tärkeää valita niistä omia tarkoituksia parhaiten palveleva vaihtoehto ajoissa. MariaDB on MySQL:n ”forkki” eli MySQL:n avoimen

lähdekoodin johdannainen. Se on vielä verrattain nuori, mistä johtuen samankaltaisuudet MySQL:n kanssa ovat edelleen merkittäviä. Aiemmissa luvuissa MySQL:ää sivuten käsitellyt relaatiotietokantojen perusominaisuudet pätevätkin myös siihen.

MySQL 8.0 on ottanut edellä mainitun datakirjastomuutoksen kaltaisia harppauksia suorituskyvyssä, joten MariaDB ei ole sen suhteen enää yhtä selkeästi edellä. Suurimmat erot löytyvät niiden edistyneemmistä ominaisuuksista. Seuraavissa luvuissa pyritään hyödyntämään erityisesti järjestelmien omia verkkodokumentaatioita tärkeimpien MariaDB:n MySQL:stä erottavien seikkojen löytämiseksi.

3.4.1 Näkymät ja näkymättömät sarakkeet

MariaDB esitteli versiossa 10.3.3 näkymättömät sarakkeet, joita ei löydy MySQL:stä. Se mahdollistaa sarakkeiden merkitsemisen näkymättömiksi siten, ettei SQL:n SELECT * -lauseke palauta niitä. Niille ei myöskään tarvitse antaa arvoa INSERT-lausekkeella. Rajoitteena on, ettei taulun jokainen sarake voi olla merkitty näkymättömäksi. (MariaDB, 2020l).

Näkymättömien sarakkeiden käyttötarkoituksiin voi kuulua turvallisuuden parantaminen tai yhteensopivuuden säilyttäminen yhteiskäytössä näkymien kanssa (eng views). Näkymät ovat sekä MariaDB:stä ja MySQL:stä löytyvä tapa tallentaa monimutkaisia SELECT-kyselyjä yksinkertaisen nimen taakse. Tallennettu kysely voidaan näkymän luonnin jälkeen suorittaa lyhyellä SELECT * -lausekkeella. (MariaDB, 2020m).

Kuvitteellisena esimerkkinä voimme ottaa sovelluksen, joka käyttää SELECT * -lausekkeen sisältävää näkymää. Sen kohteena olevaan tauluun lisätään eri käyttötarkoitusta varten uusi sarake, jota ei haluta palauttaa näkymän käyttökontekstissa turvallisuussyistä. Tällöin sarake voidaan jättää pois näkymästä merkitsemällä sarake näkymättömäksi. Voimme myös myöhemmin lisätä uusia sarakkeita ilman tarvetta muuttaa näkymää.

3.4.2 JSON-tietotyyppi

MySQL toteuttaa natiivin JSON-tietotyypin (JavaScript Object Notation). Verrattuna JSON-arvojen tallentamiseen tekstimuodossa, MySQL:n JSON-tietotyypin etuna ovat automaattinen validointi sekä optimoitu tallennusformaatti. Jälkimmäinen mahdollistaa nopean attribuuttihaun, sillä JSON-arvoa ei tarvitse erikseen jäsentää tekstimuodosta. (MySQL, 2020c).

MariaDB puolestaan on päättänyt toteuttaa JSON-tietotyypin aliaksena LONGTEXT-tietotyypille, koska JSON-tietotyypin katsotaan rikkovan SQL-standardia (MariaDB, 2020g). Tämä tarkoittaa sitä, että MySQL-tietokanta, joka sisältää JSON-arvoja voidaan korvata MariaDB:llä. Tällöin JSON-arvot tunnistetaan aliaksen kautta ja muutetaan LONGTEXT-muotoon. Yhteensopivuusongelmaksi jää replikointi MySQL-tietokannasta MariaDB-tietokantaan. Tämä voidaan kiertää muuttamalla JSON-tietotyypit TEXT-tyypeiksi MySQL:n puolella (MariaDB, 2020g).

MariaDB (2020g) väittää suorituskyvyn vastaavan MySQL:n omaa tietotyyppiä. JSON-arvo voidaan MariaDB:n tapauksessakin validoida automaattisesti määrittelemällä JSON-tyypin attribuutille JSON_VALID-rajoite.

Mikäli MariaDB:n suorituskykyväitettä on uskominen, kysymykseksi nousee erityisesti tallennuskoko. MySQL:n mukaan MySQL:n JSON-tietotyypin tallennuskoko on verrattavissa LONGTEXT-tietotyyppiin, joten huomattavia kokoeroja ei pitäisi ilmetä (MySQL, 2020c).

3.4.3 ColumnStore ja muut tietokantamoottorit

MariaDB (2020k) listaa InnoDB mukaan lukien 16 virallisesti tuettua moottoria, joista kuusi ovat tuettuna myös MySQL:ssä. Esimerkkejä vain MariaDB:n virallisesti tukemista moottoreista ovat MyRocks, ja Aria. MyRocks on Facebookin kehittämä moottori ympäristöihin, joissa halutaan säästää levytilassa ja I/O-operaatioissa. Verrattuna InnoDB:en, MyRocks tarjoaa kaksi kertaa tiiviimmän pakkauksen sekä kaksi kertaa vähemmän kirjoitusoperaatioita. Aria puolestaan on MySQL:nkin tukeman MyISAM-moottorin korvaaja paremmalla välimuistitoiminnallisuudella. (MariaDB, 2020k).

Aiemmin mainitut 16 moottoria eivät sisällä ColumnStorea. Se on erittäin suurien datamäärien prosessointiin ja analysointiin optimoitu sarakepohjainen tietokantamoottori. Käytännössä itse tietokantamoottori on kuitenkin osa ColumnStore-komponenttia, joka tulee mukana kaupallisessa MariaDB Platform -paketissa. (MariaDB, 2020j).

ColumnStoren tarkoitus on siis mahdollistaa MariaDB:n tehokas käyttö OLAP-kontekstissa (Online Analytical Processing). Molempien oletusmoottori InnoDB on yleiskäyttöinen ja soveltuu moniin tilanteisiin, mutta valinnanvarassa etu on MariaDB:llä.

3.4.4 Replikointi - Galera Cluster vs. Group Replication

Sekä MariaDB että MySQL:n ilmaisversio tarjoavat sekä tavallisia asynkronisia että semisynkronisia master-slave -replikointiratkaisuja. Lisäksi tarjolla on multi-master/master-master, jossa useampi palvelin hyväksyy aina myös kirjoituksia. Verrattuna master-slave -replikointiin, multi-master parantaa paitsi kirjoitusten suorituskykyä että osituksen sietokykyä, koska tarjolla on useampi pääpalvelin. Se ei kuitenkaan täysin ratkaise master-slave -replikoinnin ongelmia, koska transaktiot voivat edelleen kadota niitä käsittelevän pääpalvelimen kaatuessa.

Edistyneemmät ratkaisut ovat tietokantajärjestelmäkohtaisia. MySQL tarjoaa Group Replication -nimisen replikointimetodin erikseen asennettavana lisäosana. Siinä palvelimet toimivat itsenäisesti ryhmissä, joko yhden tai useamman pääpalvelimen konfiguraatiossa (Bhavsar ja muut, 2018). Group Replication on virtuaalisesti synkroninen siten, että palvelinten enemmistön tulee kuitata saaneensa transaktio. Lisäksi samassa ryhmässä olevat palvelimet osaavat automaattisesti valita uuden pääpalvelimen yhden kaatuessa, mikä parantaa replikointiratkaisun osituksen sietokykyä. Vain InnoDB-tietokantamoottori on tuettu. (MySQL, 2020a).

MariaDB puolestaan toimittaa asennuksen mukana oman toteutuksensa Galera Clusterista. Se on virtuaalisesti synkroninen multi-master -ratkaisu, jossa jokainen palvelin voi ottaa vastaan transaktioita. Galera Clusterissa transaktiot ohjataan heti jokaiselle palvemelle, jotka muodostavat yhdenmukaisen päätöksen sen onnistumisesta (vrt. Group Replicationin enemmistö). Virtuaalinen synkronisuus tulee siitä, että Galera Cluster käyttää sertifikaattipohjaista replikointia. (MariaDB, 2020d). Vaikka ratkaisu näyttää ulos päin synkroniselta, transaktiot suoritetaan todellisuudessa asynkronisesti.

Ulos päin synkronisen toiminnan lisäksi Galera Cluster parantaa eheyttä edelleen verrattuna Group Replicationiin, koska se takaa transaktioiden näkymisen yhtä aikaa kaikilla palvelimilla. Transaktioita ei myöskään tämän johdosta hukata jonkin palvelimen kaatuessa, mikä nostaa osituksen sietokykyä.

Galera Clusterin potentiaaliseksi ongelmaksi jäävät perinteisten synkronisten ratkaisujen tavoin yksittäiset palvelimet. Mikäli yksi niistä saa suoritettavakseen suuria kirjoitusoperaatioita sisältäviä transaktioita, se voi muodostua pullonkaulaksi ja laskea saatavuutta. Ongelma ei ole kuitenkaan yhtä ilmeinen kuin puhtaasti synkronisessa ratkaisussa, kiitos sertifikaattipohjaisen replikoinnin. MySQL Group Replicationin tavoin se ei myöskään tue muita tietokantamoottoreita InnoDB:n lisäksi (MariaDB, 2020e).

3.4.5 Thread Pool

MySQL:n ja MariaDB:n perinteinen tapa luoda jokaiselle tietokantayhteydelle uusi säie voi osoittautua suorituskykyongelmaksi erityisesti, jos yhteyksiä luodaan jatkuvasti ja ne pysyvät elossa keskimäärin hyvin lyhyen ajan. Tällaista työtä kutsutaan termillä OLTP (Online Transactional Processing) ja siihen lukeutuvat esimerkiksi sosiaalisen median palvelut sekä useimmat muut websovellukset. (Razzoli, 2014).

Nykyään yksi säie voi Razzolin (2014) mukaan huolehtia useammasta yhteydestä, mutta yhtäaikaisten yhteyksien kasvaessa ennalta määrätyn kokoinen määrä säikeitä jää auttamatta pullonkaulaksi. Sekä MySQL:ssä että MariaDB:ssä saatavilla oleva paranneltu Thread Pool pyrkii ratkaisemaan ongelman hallitsemalla säikeitä dynaamisesti sekä rajoittamalla säikeiden resurssien käyttöä (MariaDB, 2020c). Käytännössä se pyrkii pitämään elossa aina sopivan määrän säikeitä luomalla uusia nykyisen määrän jäädessä riittämättömäksi ja tuhoten ylimääräisiä kun niitä ei enää tarvita (Razzoli, 2014).

MariaDB:n (2020c) mukaan Thread Pool ei kuitenkaan ole ratkaisu jokaiseen tilanteeseen. Huolimatta kokonaissuorituskyvyn parantumisesta, se saattaa nostaa yksittäisten kyselyiden suoritusaikaa, mistä johtuen myös hyvin yksinkertaiset ja tavallisesti nopeat kyselyt voivat muuttua hitaammiksi. Esimerkkinä MariaDB antaa tilanteen, jossa tietokanta saa pitkin väliajoin suuren määrän kyselyitä. Tällaisissa tilanteissa kyselyiden todennäköisesti odotetaan palaavan mahdollisimman nopeasti, jolloin Thread Pool ei tarjoa haluttua suorituskykyä.

Thread Pool on tietyt sovellusalueet pois luettuna tehokas keino parantaa järjestelmän suorituskykyä. Se on tarjolla MariaDB:ssä ilmaiseksi, mutta MySQL:n osalta vain sen kaupallisissa versioissa.

3.4.6 Lisensointi ja tuki

GNU GPL v2 -lisenssi mahdollistaa sekä MySQL:n että MariaDB:n (tai vain jompaa kumpaa) palvelinohjelmistoa käyttävän sovelluksen jakelun myös kaupallisesti, mikäli sovelluksen toiminta ei riipu puhtaasti niistä eikä sovellus käytä suoraan kummankaan lähdekoodia. Yksityinen, esimerkiksi organisaation sisäinen jakelu, on vapaata kaikissa tapauksissa. Mikäli sovellus ei esimerkiksi toimi ollenkaan ilman MySQL:ää, saa sitä jakaa vain GPL v2 -lisenssillä ellei kehittäjä osta erillistä lisenssiä Oraclelta. (MariaDB, 2020h).

MySQL:n kaupalliset versiot mahdollistavat myös MySQL:stä täysin riippuvaisen sovelluksen jakelun kaupallisesti ilman tarvetta noudattaa GPL v2 -lisenssiä. Lisenssin lisäksi ne sisältävät eritasoisia teknisen tuen paketteja sekä versiosta riippuen erinäisiä lisäosia ja työkaluja, kuten Thread Pool ja MySQL Enterprise Monitor. (MySQL, 2020d) MariaDB:n palvelinohjelmiston kaupallisen version sisältävä MariaDB Platform -paketti ei riitä siitä riippuvaisen sovelluksen jakeluun ilman GPL v2 -lisenssin noudattamista, koska se on MySQL:n johdannainen. Se tarjoaa kuitenkin MySQL:n kaupallisten versioiden tavoin erilaisia teknisen tuen vaihtoehtoja sekä lisäosia ja työkaluja, kuten palomuurin sekä monitorointi- ja diagnostiikkatyökaluja (MariaDB, 2020i).

Koska Oracle omistaa MySQL:n lähdekoodin, sen kaupalliset lisenssivaihtoehdot tarjoavat enemmän joustavuutta. Lisäksi Oraclen tuki on kattavampaa, tarjoten tukipaketista riippuen jatkuvia päivityksiä GA-julkaisusta vielä viiden, kahdeksan tai useamman vuoden päähän (MySQL, 2020b). MariaDB puolestaan on avoimen lähdekoodin asialla ja julkaisi esimerkiksi ColumnStoren GPL-lisenssillä. MariaDB Foundation (2020) lupaa ilmaisia päivityksiä viiden vuoden verran GA-julkaisusta, mutta ei tarjoa laajennettuja tukipaketteja.

4. Yhteenveto

Tutkielman tarkoituksena oli antaa yleiskuva relaatiotietokantojen toiminnasta sekä selvittää MariaDB:n nykyasemaa MySQL:n kilpailijana. Tämän pohjalta tarkoitus oli vastata tutkimuskysymykseen siitä, mitä perusteita organisaatiolla voi olla MariaDB:n valintaan MySQL:n sijaan. MySQL:n käyttäminen relaatiotietokantojen toiminnan avaamiseen perustui MySQL:n asemaan yhtenä suosituimmista tietokantajärjestelmistä. MariaDB puolestaan on sen alkuperäisen kehitystiimin tuottama sekä sen suora johdannainen. Järjestelmien historia ja tulevaisuus ovat kietoutuneet yhteen saman koodikannan kautta, mikä antaa aihetta nykyaseman tarkastelulle.

Taustatutkimusta relaatiotietokannoista oli saatavilla runsaasti, mutta sekä MySQL että MariaDB ovat molemmat kehittyneet viimeisten vuosien aikana. Erityisesti MySQL 8.0:n julkaisun vuonna 2016 voidaan katsoa merkittäväksi muutokseksi järjestelmien kehityssuunnassa, koska binääriyhteensopivuus menetettiin. MariaDB:seen keskittyvien viime aikaisten tutkimusten niukkuus pakotti verkkodokumentaation syvälliseen analysointiin.

Järjestelmissä on edelleen hyvin paljon samaa ja toisen valinnan tulee perustua pääasiassa niiden edistyneempiin ominaisuuksiin sekä kaupalliseen tarjontaan. Yleisesti ottaen MariaDB tarjoaa ominaisuuksistaan useampia ilmaiseksi, mikä sopii erityisesti pienille yrityksille. Myös MariaDB:n taipumus avoimeen lähdekoodiin voi olla valintaperuste erityisesti akateemisissa konteksteissa, sillä avointen lisenssien piirissä on muutakin kuin vain MySQL:ään pohjautuva palvelinohjelmisto.

Jatkossa lähempään tarkasteluun voitaisiin ottaa molempien järjestelmien tuoteperheet kokonaisuudessaan – laajennukset, työkalut sekä kaupalliset erikoisversiot, kuten MariaDB SkySQL. Mitä syvemmälle kummankin tuoteperheen tarjontaan mennään, sitä enemmän eroja niistä löytyy. Tuoteperheiden vertailu tarjoaisi paremman pohjan järjestelmän valintaan.

Lähteet

Bhavsar, A., Mehta, C., Oza, H. & Shah, S. (2018). *MySQL 8 administrator's guide*. Birmingham: Packt Publishing.

Bartholomew, D. (2012). MariaDB vs. MySQL. *Dostopano*, 7(10), 2014.

Binani, S., Gutti, A. & Upadhyay, S. (2016). SQL vs. NoSQL vs. NewSQL – A comparative study. *Communications on Applied Electronics*, 6(1), 43-46.

Brewer, E. (2012). CAP twelve years later: how the "rules" have changed. *Computer*, 45(2): 23-29.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377-387.

DB-Engines. (2020). DB-Engines Ranking. Lainattu 15.4.2019, saatavilla: https://db-engines.com/en/ranking

Gilfillan, I. (2017). Why does MariaDB 10.2 use InnoDB instead of XtraDB? Lainattu 26.11.2019, saatavilla: https://mariadb.com/kb/en/library/why-does-mariadb-102-use-innodb-instead-of-xtradb/

Foster, E. C. & Godbole, S. (2016). *Database systems: a pragmatic approach* (2nd ed.). Apress.

Korth, H. F., Silberschatz, A. & Sudarshan, S. (2011). *Database system concepts* (6th ed.). New York, NY: McGraw-Hill.

MariaDB (2020-a). MariaDB versus MySQL: Compatibility. Lainattu 5.4.2020, saatavilla: https://mariadb.com/kb/en/mariadb-vs-mysql-compatibility/

MariaDB (2020-b). Incompatibilities and Feature Differences Between MariaDB 10.3 and MySQL 5.7. Lainattu 5.4.2020, saatavilla: https://mariadb.com/kb/en/incompatibilities-and-feature-differences-between-mariadb-103-and-mysql-57/

MariaDB (2020-c). Thread Pool in MariaDB. Lainattu 5.4.2020, saatavilla: https://mariadb.com/kb/en/thread-pool-in-mariadb/

MariaDB (2020-d). About Galera Replication. Lainattu 5.4.2020, saatavilla: https://mariadb.com/kb/en/about-galera-replication/

MariaDB (2020-e). MariaDB Galera Cluster - Known Limitations. Lainattu 5.4.2020, saatavilla: https://mariadb.com/kb/en/mariadb-galera-cluster-known-limitations/

MariaDB (2020-f). Downloads. Lainattu 8.4.2020, saatavilla: https://downloads.mariadb.org/mariadb/+releases/

MariaDB (2020-g). JSON Data Type. Lainattu 8.4.2020, saatavilla: https://mariadb.com/kb/en/json-data-type/

MariaDB (2020-h). Licensing FAQ. Lainattu 8.4.2020, saatavilla: https://mariadb.com/kb/en/licensing-faq/

MariaDB (2020-i). MariaDB Platform Enterprise Subscription. Lainattu 8.4.2020, saatavilla: https://mariadb.com/products/platform-subscription/

MariaDB (2020-j). MariaDB ColumnStore. Lainattu 8.4.2020, saatavilla: https://mariadb.com/docs/features/mariadb-columnstore/

MariaDB (2020-k). Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0. Lainattu 8.4.2020, saatavilla: https://mariadb.com/kb/en/incompatibilities-and-feature-differences-between-mariadb-104-and-mysql-80/

MariaDB (2020-l). Invisible columns. Lainattu 8.4.2020, saatavilla: https://mariadb.com/kb/en/invisible-columns/

MariaDB (2020-m). Creating and Using Views. Lainattu 8.4.2020, saatavilla: https://mariadb.com/kb/en/creating-using-views/

MariaDB Foundation (2020). About MariaDB Server. Lainattu 8.4.2020, saatavilla: https://mariadb.org/about/

MySQL (2020-a). Group Replication Background. Lainattu 5.4.2020, saatavilla: https://dev.mysql.com/doc/refman/8.0/en/group-replication-background.html

MySQL (2020-b). MySQL Editions. Lainattu 5.4.2020, saatavilla: https://www.mysql.com/products/

MySQL (2020-c). The JSON Data Type. Lainattu 8.4.2020, saatavilla: https://dev.mysql.com/doc/refman/8.0/en/json.html

MySQL (2020-d). MySQL as an Embedded Database. Lainattu 8.4.2020, saatavilla: https://www.mysql.com/oem/

MySQL (2020-e). MySQL 8.0 Release Notes. Lainattu 8.4.2020, saatavilla: https://dev.mysql.com/doc/relnotes/mysql/8.0/en/

Nyman, L. (2013). Freedom and forking in open source software: the MariaDB story. *Proceedings of the 22nd Nordic Academy of Management Conference*.

Pearce, R. (2013). Dead database walking: MySQL's creator on why the future belongs to MariaDB. Lainattu 24.11.2019, saatavilla: https://www.computerworld.com.au/article/457551/dead_database_walking_mysql_creator_why_future_belongs_mariadb/

Razzoli, F. (2014). *Mastering MariaDB*. Birmingham: Packt Publishing.

Stack Overflow. (2019). Developer Survey Results. Lainattu 24.11.2019, saatavilla: https://insights.stackoverflow.com/survey/2019