NewSQL-tietokannat

Oulun yliopisto
Tieto- ja sähkötekniikan tiedekunta
Tietojenkäsittelytiede
Kandidaatin tutkielma
Aleksi Hytönen
26.08.2019

Tiivistelmä

Tämä kandidaatin tutkielma käsittelee NewSQL-tietokantoja. Tutkielmassa perehdytään siihen, kuinka NewSQL-tietokannat kykenevät takaamaan ACID-transaktiot. Tämä selvitetään tutkimalla erilaisia NewSQL-tietokantoja. Valitsin tämän aiheen henkilökohtaisen mielenkiinnon vuoksi. Lisäksi aihe on tuore ja mahdollisesti tulevaisuudessa merkittävä. Käsiteltävän tiedon määrä on nopeassa kasvussa, joten yritykset tarvitsevat luotettavia ratkaisuja tallentaa ja hallinnoida suurta määrää tietoa. Varsinkin koneoppiminen vaatii suuren datamäärän käsittelyä. Tällä hetkellä on tärkeää varmistaa NewSQL-tietokantojen luotettavuus.

Tutkimuskysymyksen vastausta lähestytään erilaisten NewSQL-tietokantaratkaisujen avulla. Tällaisia ovat esimerkiksi VoltDB ja MemSQL, joiden ratkaisut ACID-transaktioiden takaamiseen esitellään tässä tutkielmassa pääpiirteisesti.

NewSQL-tietokantojen kehitys on ollut jatkumo jo 1960-luvulta lähtien alkaen relaatiotietokannoista. Käsiteltävän tiedon kasvun takia tietokantoja on jouduttu laajentamaan ja hajauttamaan, mikä on aiheuttanut haasteita tietokantojen luotettavuuteen. Tämän takia tutkielmassa käydään myös läpi, millaisia vaikutuksia tietokantojen hajautuksella on ollut niiden luotettavuuteen.

Avainsanat
NewSQL, tietokannan luotettavuus, ACID-transaktio, tietokannan hajauttaminen

Ohjaaja
Juha Iisakka

Sisällysluettelo

1. Johdanto

Perinteiset relaatiotietokannat eivät enää nykyään kykene vastaamaan yritysten tarpeisiin. Suuret yritykset keräävät ja käsittelevät suurta määrää dataa, minkä vuoksi tietokantoja on jouduttu kasvattamaan nopeaan tahtiin. Relaatiotietokantajärjestelmät ovat suunniteltu ajalla, jolloin tiedon määrä ei ollut yhtä massiivinen, joten niiden käyttäminen ei ole enää tehokasta suurille, hajautetuille organisaatioille (Grolinger, Higashino, Tiwari, & Capretz, 2013).

Tietovarastojen fyysinen kasvattaminen esimerkiksi kovalevyjen määrän lisäämisellä ei kuitenkaan ole kannattavaa johtuen suurista kustannuksista. Esimerkiksi taulujen yhdistäminen suurille, hajautetuille tauluille on tehotonta ja vie usein liian paljon aikaa (Leavitt, 2010). Tämän takia tietokantoja on jouduttu hajauttamaan hyödyntäen NoSQL-tietokantoja (Sandborg, 2016). Näiden tietokantojen avulla yritykset ovat kyenneet skaalaamaan tietovarastojaan horisontaalisesta tehden laajennuksista kustannustehokkaita. Hajautettu ja tehokas tietokantajärjestelmä on erityisen hyödyllinen suurille, hajautetuille yrityksille, joiden toimipisteet sijaitsevat ympäri maailmaa. Järjestelmän solmut voidaan sijoittaa useisiin paikkoihin ympäri maailmaa ja yksiköt kykenevät tekemään kyselyitä kaikista läheisimpään solmuun ilman, että koko järjestelmää tarvitsisi käydä läpi tai tieto pitäisi hakea kaukaa (Cattell, 2010).

NoSQL-tietokantojen heikkoutena on eheyden heikentyminen, jonka vuoksi nämä tietokantaratkaisut eivät ole yhtä luotettavia, kuin perinteiset relaatiotietokannat. Lisäksi ne eivät tue SQL-kyselykieltä. Tänä päivänä yritykset keräävät suuria määriä tietoa, jonka pohjalta ne tekevät ennusteita. Tästä johtuen käsiteltävän tiedon tulee olla luotettavaa. Tästä johtuen yritykset ovat kehittäneet uusia NewSQL-tietokantajärjestelmiä, jotka tukevat nopeiden kyselyiden tekemistä hajautettuun tietokantajärjestelmään tehokkaasti ja luotettavasti. Nämä järjestelmät ovat sopivia yrityksille ja organisaatioille, joiden päätöksentekoon tarpeelliset järjestelmät tarvitsevat tietokannan, jonka tieto on aina oikeaa tietoa ja sen on aina saatavilla siitä huolimatta, että järjestelmän solmut on hajautettu ympäri maailmaa (Silvestre, Sauvanaud, Kaaniche, & Kanoun, 2015). Jotta voidaan olla varmoja siitä, että NewSQL-tietokannat ovat luotettavia, niiden ACID-transaktioiden tukemista tulee varmistaa ja tehokasta skaalautumista täytyy mitata.

NewSQL –tietokantojen perusajatuksena on tarjota tehokkaasti skaalautuvan tietokantajärjestelmän NoSQL –tietokantojen tapaan luopumatta relaatiotietokantojen tarjoamasta ACID-ominaisuudesta. Tutkimuksen tarkoituksena on selvittää, kuinka nykyiset NewSQL -tietokantajärjestelmät saavuttavat nämä ominaisuudet. Lisäksi selvitetään, kuinka tietokantojen hajautus vaikuttaa niiden luotettavuuteen ja kuinka NewSQL-tietokannat pitävän tallennetun tiedon eheänä.

Vaikkakin tutkimuksessa käydään läpi erilaisia NewSQL-tietokantoja, niiden käytännön implementointiin ja käyttötarkoituksiin ei kiinnitetä huomiota. Tietokantojen arkkitehtuuria käsitellään ainoastaan tutkimuskysymykseen peilattuna relevanteista näkökulmista. Empiiristä tutkimusta ei tehdä, vaan kaikki materiaali kerätään aiemmin aiheesta tehdyistä tutkimuksista.

Luvussa 2 käydään läpi ACID-transaktioita ja millaisia ongelmia tietokantojen hajautus on aiheuttanut näiden transaktioiden takaamiseksi. Luvussa 3 käydään läpi BASE-transaktioita ja NoSQL-tietokantoja, jotka ovat pyrkineet ratkaisemaan tietokantojen hajautuksessa esiintyviä ongelmia. Luvussa 4 käydään läpi NewSQL-tietokantoja ja niiden luotettavuutta. Lisäksi luvussa vastataan edellä mainittuun tutkimuskysymykseen.

Tämä tutkielma koostuu tähän aiheeseen liittyvistä tutkimuksista, jotka olen koonnut yhteen. Tiedonhaun työkaluna olen käyttänyt ACM-kirjastoa sekä Scopusta. Käytettyjä hakulauseita ovat ”NewSQL AND database*” (Scopus 60 osumaa, ACM 1 osuma), ”relational AND database*” (Scopus 26888 osumaa, ACM 267 osumaa), ”NoSQL AND database*” (Scopus 1896 osumaa, ACM 3 osumaa) ja “CAP AND theorem” (Scopus 337 osumaa, ACM 31 osumaa). Osumat laitoin järjestykseen relevanssin mukaan ja tarkastelin kymmentä ensimmäistä osumaa.

2. ACID, BASE ja tietokantojen hajautus

Relaatiotietokantojen vahvuutena on ACID –transaktioiden takaaminen. Tämä ominaisuus tekee tietokantajärjestelmistä luotettavia. ACID tulee englannin kielen sanoista 'Atomicity', 'Consistency', 'Isolation' ja 'Durability'. Relaatiotietokantaan tehtävät luku- ja kirjoitusoperaatiot suoritetaan joko kokonaisuudessaan loppuun saakka tai niitä ei suoriteta ollenkaan (atomicity). Kun tietokantaan tehdään kirjoitusoperaatio, tietokanta sisältää tämän jälkeen juuri tämän tiedon, eikä virheellistä tai vanhaa tietoa (Consistency). Kun tietokannan joku solmu haluaa suorittaa jonkin operaation, muille solmuille ei anneta lupaa suorittaa mitään operaatioita. Näin ainoastaan yhdellä solmulla kerrallaan on oikeus tehdä operaatioita tietokantaan (isolation). Kun tietokantaan tehdään jokin operaatio alusta loppuun, se on pysyvä, eikä sitä voi jälkeenpäin muuttaa ilman uutta operaatiota (durability). Näiden ominaisuuksien ansiosta relaatiotietokantajärjestelmät ovat luotettavia, koska ne sisältävät vain oikeaa tietoa ja niiden operaatiot suoritetaan aina loppuun saakka, mikäli operaation tekeminen on mahdollista (Leavitt, 2010).

2.1 Tietokantojen hajautus

Vuonna 2000 Berkeleyn yliopiston professori Eric A. Brewer esitteli kehittämänsä CAP-teoreeman, joka jakaa hajautettujen tietojärjestelmien ominaisuudet eheyteen, saatavuuteen ja verkon pirstoutumisen sietokykyyn. Brewerin mukaan hajautetut tietojärjestelmät voivat taata näistä ainoastaan kaksi ominaisuutta (Ahsan & Gupta, 2016). Hajautettujen tietojärjestelmien ominaisuuksia kuvataan teoreeman nimen mukaisin kirjaimin sen mukaan, minkä ominaisuuden tietojärjestelmä on jättänyt pois. Esimerkiksi tietojärjestelmän luopuessa eheydestä se määritellään AP-tietojärjestelmäksi.

CAP-teoreeman ominaisuuksien saavuttamiseksi hajautetun tietojärjestelmän tulee suorittaa erilaisia toimenpiteitä tietyn ominaisuuden kannalta. Jotta järjestelmä olisi aina eheä, sen tulee suorittaa eri solmuille tulevat operaatiot joko kokonaan loppuun tai ei ollenkaan. Näin saavutetaan tilanne, jossa kaikki järjestelmän solut sisältävät yhteneväistä tietoa eli järjestelmän tietokanta on eheä. Saatavuus saavutetaan siten, että järjestelmä on aina käytössä, vaikka operaatioissa tapahtuisi virheitä. Verkon pirstoutumisen sietokyky saavutetaan siten, että järjestelmän solmut jatkavat tietojen käsittelyä, vaikka niillä ei olisi yhteyksiä yksittäisiin solmuihin tai koko verkko olisi poikki (Stonebraker, 2010).

2.2 Hajautuksen vaikutus eheyteen

Tietokannan eheys on merkittävä tekijä tietokannan luotettavuuden kannalta. Tietokanta on eheä silloin, kun kaikki sen sisältämä tieto on uusinta tietoa. Toisin sanoen tietokannan eheys on varmistettu, mikäli jokaisella transaktiolla on käytössään samanlaista tietoa, jokainen transaktio ajan kuluessa saadaan valmiiksi ja lopullinen tietokanta kaikkien transaktioiden valmistuessa sisältää uusinta tietoa. Yksittäisissä tietokannoissa tämä saavutetaan, kun jokainen operaatio suoritetaan yksi kerrallaan joko kokonaan tai ei ollenkaan. Hajautetussa tietokannassa taas samaan tietokantaan tehdään useita transaktioita yhtä aikaa (Silberschatz & Kedem, 1980).

Kun transaktioita lähdetään hajauttamaan, niiden loppuun suorittamisen varmistamiseksi relaatiotietokannoissa käytetään 2PC-protokollaa (two-phase commit). Tämän protokollan avulla ACID-transaktio kyetään varmistamaan, vaikka transaktio jouduttaisiin hajauttamaan. Käytännön toteutus tapahtuu nimensä mukaisesti kahdessa vaiheessa. Ensimmäistä vaihetta kutsutaan äänestysvaiheeksi ja toista vaihetta päätösvaiheeksi. Äänestysvaiheessa yksi transaktioon osallistuva solmu kysyy muilta solmuilta, ovatko ne valmiita sitoutumaan. Solmu äänestää ”kyllä”, mikäli se voi taata transaktion loppuun suorituksen. Mikäli solmu ei kykene sitoutumaan transaktion suoritukseen, se äänestää ”ei”. Jos kaikki transaktioon tarpeelliset solmut äänestivät ”kyllä”, transaktio suoritetaan. Jos yksikin tarpeellinen solmu äänesti ”ei”, tai vastausta ei saada määrätyn ajan kuluessa, transaktio keskeytetään (Samaras, Britton, Citron, & Mohan, 1995).

2PC-protokolla mahdollistaa transaktioiden tekemisen hajautettuun tietokantaan ja samalla varmistaa tietokannan eheyden. Ongelmana kuitenkin on CAP-teoreemassa esitelty tietokannan saatavuus. Tietokantaan ei kyetä tekemään luku- tai kirjoitusoperaatioita, mikäli yhteys solmujen välillä on poikki. Tällä saattaa olla suuriakin vaikutuksia varsinkin, jos tietokantajärjestelmä on merkittävässä roolissa jonkin yrityksen toiminnassa. Esimerkiksi jos tietokanta on toiminnassa ja saatavilla 99.9% ajasta, voi tämä tarkoittaa jopa 45 minuutin katkoksia joka kuukausi (Pritchett, 2008).

2.3 BASE ja NoSQL

Joillekin järjestelmille on järkevämpää tietokantaratkaisua ajatellessa valita tietokanta, joka valitsee saatavuuden eheyden sijasta. Näin ollen tietokantaan kyetään tekemään operaatioita, vaikka tieto ei olisi täysin eheää. Esimerkiksi sosiaaliset mediat käyttävät tällaista ratkaisua. ACID-transaktioita tukevista tietokannoista on siirrytty käyttämään BASE-oikeellisuusmallia. Tämä malli luopuu ACID:in tarjoamasta eheydestä ja eristyneisyydestä ja sen tilalle tarjoaa korkeaa saatavuutta ja tehokkuutta (Strauch, Sites, & Kriha, 2011).

BASE-termi tulee englannin kielen sanoista ’Basically available’, ’Soft state’ ja ’Eventually consistent’. Tämä tarkoittaa sitä, että tietokanta on käytännössä aina saatavilla, vaikka siinä oleva tieto ei olisi ajantasaista (basically available). Solmussa oleva tieto saattaa olla myös vanhentunut, vaikka sillä hetkellä ei tehtäisi mitään kirjoitusoperaatioita. Tämä johtuu tietokannan automaattisesta solmujen välisistä prosesseista, esimerkiksi replikoinnista (soft state). Ajan kuluessa koko tietokanta tulee sisältämään oikeaa ja ajantasaista tietoa automaattisen replikoinnin ansiosta, mikäli kirjoitusoperaatioita ei tehdä replikoinnin aikana (eventually consistent). Näiden ominaisuuksien avulla CAP –teoreeman mukaisesti yhdestä ominaisuudesta ei tarvitse luopua, mutta tietokantajärjestelmän luotettavuus kärsii (Pritchett, 2008).

Vaikkakin BASE-tietokannat eivät ole yhtä luotettavia kuin ACID-tietokannat, niiden hyödyllisyys riippuu käyttötarkoituksesta. Jos esimerkiksi haettavan tiedon ei tarvitse olla aivan uusinta tietoa, BASE-tietokanta voi olla parempi vaihtoehto. Tästä esimerkkinä sosiaaliset mediat, joissa tiedon ei tarvitse olla juuri uusinta tietoa, vaan on riittävää, että tieto on uusinta tietoa kohtuullisessa ajassa. Mikäli tiedon pitäisi olla aina oikeaa, sosiaalisen median ylläpitäminen olisi mahdotonta, koska ACID-tapauksessa ainoastaan yksi käyttäjä voisi julkaista päivityksen kerrallaan ja sen jälkeen kaikki käyttäjät joutuisivat odottamaan, että päivitys replikoituu kaikkiin solmuihin. Tämä taas poistaa tietokannan saatavuuden. BASE-malli antaa suunnittelijalle mahdollisuuden luoda tehokkaita tietojärjestelmiä, mikäli tiedon ei tarvitse olla reaaliaikaista (Bailis & Ghodsi, 2013).

2.4 NoSQL-tietokannat

BASE-oikeellisuusmallia toteuttavia tietokantaratkaisuja kutsutaan NoSQL-tietokannoiksi. Termi ”NoSQL” valittiin kuvaamaan vaikeasti määriteltäviä ei-relaatiotietokantoja. Suurin osa tällaisista tietokannoista ei tue SQL-kyselykieltä. Jotkin kuitenkin käyttävät, minkä vuoksi termi on harhaanjohtava. Usein termi tulkitaan ”not only SQL”. Näiden tietokantojen rakenne on suunniteltu yksinkertaiseksi ja joustavaksi. NoSQL-tietokantoihin kyetään tallentamaan useita erilaisia malleja hyödyntäen paljon sekä strukturoitua että huonosti strukturoitavaa tietoa. Esimerkiksi avain-arvo-mallilla kyetään tallentamaan tietoa pareina, tai kaaviomallilla kyetään tallentamaan erilaisia kaavioita visualisoimaan tietoa. Lisäksi mallit ovat hyvin joustavia ja tietokannan kasvattaminen tapahtuu automaattisesti (Cattell, 2010).

NoSQL-tietokannoille on ominaista skaalautua relaatiotietokantoja tehokkaammin jakamalla tallennetun tiedon usealle koneelle. Osittaminen tapahtuu pääosin avainten perusteella siten, että toisiinsa yhdistettävät tiedot on helppo etsiä. Tietokannan käyttämää hajautusfunktiota ei kuitenkaan usein pysty määrittelemään etukäteen, koska tallennettavien tietojen piirteet voivat vaihdella paljonkin. Tästä johtuen tiedon ositus toteutetaan automaattisesti, kun esimerkiksi taulujen koko kasvaa suuresti. Tällaiset taulut hajotetaan paloiksi ja tallennetaan eri koneille muistikuorman vähentämiseksi (Silberschatz, Korth, & Sudarshan, 2010).

Perinteisiin relaatiotietokantoihin verrattuna NoSQL-tietokantojen valttina on ollut, että skaalautuu tehokkaasti ja on aina saatavilla. Tietokantojen eheys kuitenkin kärsii, koska tallennettu tieto ei välttämättä olen uusinta tietoa. Tämä on merkittävä heikkous, mikäli yritys haluaa tehokkaan skaalautuvuuden lisäksi luotettavaa tietoa esimerkiksi päätöksenteon tueksi. Yritykset haluavat kerätä ja analysoida erittäin suurta määrää dataa, jotta mahdollisimman tarkkoja ennusteita päätöksien vaikutuksista voitaisiin tehdä. Relaatiotietokannat taas ovat tehottomia suuren tietomäärän käsittelemiseen, vaikkakin relaatiotietokanta olisi luotettava. Tämän ongelman punnitseminen on yrityksille vaikea paikka erityisesti silloin, jos tietokantajärjestelmä on erittäin merkittävässä roolissa päätöksenteon kannalta (Boicea, Radulescu, & Agapin, 2012).

3. NewSQL

NewSQL-tietokanta on yleinen nimitys sellaisille tietokannoille, jotka yhdistävät perinteisten relaatiotietokantojen ACID-transaktiot ja NoSQL-tietokantojen tehokkaan vaakasuuntaisen skaalautuvuuden. NewSQL-tietokanta on hyvä ratkaisu sellaisille sovelluksille, jotka vaativat perinteisten relaatiotietokantojen eheyttä ja saatavuutta, mutta samaan aikaan tietokannan tulisi olla tehokkaasti skaalautuva. Eri NewSQL-tietokantajärjestelmillä on omat keinonsa ratkaista ongelmia liittyen ACID-transaktioiden varmistamiseen ja tietokannan skaalautuvuuteen, mutta yhteistä näillä tietokantajärjestelmillä on tunnetun SQL-hakukielen tukeminen (Aslett, 2011).

Tarve NewSQL-tietokannoille syntyi, kun yritykset tarvitsivat tietokantajärjestelmiä, jotka käsittelevät paljon yrityksen päätöksentekoon merkittävää tietoa ja samaan aikaan skaalautuu tehokkaasti. Lisäksi NoSQL-tietokanta ei ole ollut mahdollisuus, koska yritykset vaativat luotettavia transaktioita ja eheyttä. Aikaisemmin tällaisilla yrityksillä oli mahdollisuus joko ostaa tehokas yhden palvelimen tietokanta tai kehittää oma keskitason järjestelmä, joka jakaa kyselyitä eri palvelimille. Molemmat vaihtoehdot ovat kalliita, joten yritykset ovat halunneet käyttää uusia NewSQL-ratkaisuja (Aslett, 2011).

3.1 Ominaisuudet

NewSQL-tietokannat voidaan jakaa kolmeen ryhmään. Ensimmäinen ryhmä koostuu tietokannoista, jotka ovat täysin uusia ja ne on rakennettu käyttämättä vanhoja ratkaisuja. Esimerkkejä tällaisista tietokantajärjestelmistä ovat VoltDB ja NuoDB. Nämä tietokannat toimivat käyttäen ”shared-nothing” –solmuja, jotka skaalautuvat dynaamisesti lisäämällä uusia solmua järjestelmään. Toiseen ryhmään kuuluvat tietokannat, jotka hyödyntävät vanhoja relaatiotietokantoja ja ne ovat rakentuneet näiden järjestelmien päälle. Ne toimivat kuten vanhat relaatiotietokannat, mutta ne skaalautuvat tehokkaammin. Kolmanteen ryhmään kuuluvat tietokannat, jotka hyödyntävät väliohjelmistoja automaattisesti osittamaan tietokanta eri solmuille. Tällaisia tietokantoja ovat esimerkiksi dbShards ja ScaleBase (Silva, Almeida, & Queiroz, 2016).

NewSQL-tietokantajärjestelmien toimivuus perustuu sisäänrakennettuihin mekanismeihin, joiden tarkoituksena on suojata järjestelmä erilaisilta virhetilanteilta, kuten esimerkiksi tiedon katoamiselta tai solmujen yhteysvirheiltä. Tällaisia mekanismeja ovat esimerkiksi tiedon hajauttaminen eri solmujen välille, tiedon replikoiminen, redundantit verkkotopologiat tiedonsiirtoon silmujen välillä ja kuormituksen tasaaminen eri solmujen välille hakujen nopeuttamiseksi (Silvestre, Sauvanaud, Kaaniche, & Kanoun, 2015). NewSQL-tietokannat tarjoavat tehokkuutta pitämällä kaiken tiedon solmujen keskusmuistissa. Tietokantaa kyetään laajentamaan joko lisäämällä solmuja klusteriin, lisäämällä koko järjestelmään uusia klustereita tai lisäämällä yksittäisten solmun muistin määrää. Tietokantajärjestelmät tekevät kyselyitä siten, että se vaikuttaa ainoastaan yhteen laitteeseen, jotta kyselyjen suorittamiseksi operaatioita ei tarvitse hajauttaa eri laitteille (Mohmmed & Osman, 2017).

Yksi merkittävimmistä NewSQL-tietokannan ominaisuuksista on mahdollisuus käyttää hakujen tekemiseen SQL-kieltä. SQL on tehokas kieli tietokannan tiedon käsittelemiseen ja hallinnointiin. Sen lisäksi kieli on yrityksille, ja niiden työntekijöille tuttu, minkä vuoksi tietokantajärjestelmä on helppo ottaa käyttöön yrityksessä. NewSQL-tietokannat mahdollistavat eri taulujen liittämisen toisiinsa, mutta tällä hetkellä liitosten tekeminen on vielä tehotonta, mikä hidastaa suurten operaatioiden tekemistä. Tämä johtuu siitä, että eri taulut voivat sijaita eri solmuille, joten niiden etsiminen on tietokannalle työläs prosessi. Jotkin liitosoperaatiot saattavat olla jopa mahdottomia (Cattell, 2010).

3.2 Ratkaisut

NewSQL-tietokannat hyödyntävät hajautettua tietokanta-arkkitehtuuria, jossa tietokanta hajautetaan klustereihin, jotka sisältävät solmuja. Nämä solmut toteutetaan käyttäen ”shared-nothing” –arkkitehtuuria, solmut eivät jaa muistia, vaan jokainen solmu sisältää oman tiedon. Näin ollen klusterissa samankaltaiset tiedot talletetaan yhdelle solmulle ja klusteri kokonaisuudessaan muodostaa yhden taulun, joka sisältää yhteyksiä toisiin tauluihin. Tämän avulla kyetään varmistamaan tehokkaiden transaktioiden suorittaminen siten, että tiedon oikeellisuus varmistetaan ja tietokantaan kyetään tekemiään useita kyselyitä yhtä aikaa. NewSQL-tietokantajärjestelmät hyödyntävät lukotonta rinnakkaisuuden hallintaa siten, että lukeminen ei aiheuta konflikteja kirjoitusten kanssa. Tämän avulla tiedon lukeminen on mahdollista, vaikka johonkin klusteriin ollaan tekemässä kirjoitusoperaatiota samanaikaisesti (Pavlo, 2016).

Jotta kyselyjen tekeminen olisi tehokasta hajautetuissa tietokantajärjestelmissä, NewSQL-tietokantajärjestelmät tallentavat tietoa tauluihin siten, että hakuja tehtäessä kyetään etsimään tarvittava tieto ainoastaan yhdestä lähteestä. Esimerkiksi yhdelle solmulle voidaan tallentaa tietoja taulukkopohjaisella mallilla (column-oriented schema) siten, että yhteen taulukkoperheeseen (column family) talletetaan tietoa käyttäjän ID:n perusteella. Näin ollen käyttäjän etsiessä tietoa itsestään, kaikkia tieto löytyy yhdestä paikasta. Tämän ominaisuuden vuoksi tietokantajärjestelmän ei tarvitse varmistaa transaktion loppuun suorittamista, koska luku- tai kirjoitusoperaatio koskee ainoastaan yhtä solmua (Pavlo, 2016).

Kaksi hyvää esimerkkiä NewSQL-tietokannoista ovat VoltDB ja MemSQL. VoltDB on NewSQL-tietokantajärjestelmä, joka hyödyntää tiedon tallentamisessa tietokoneen päämuistia. Se on rakennettu H-Store –tietokannan päälle ja se on suunniteltu ajatellen OLTP –järjestelmiä. VoltDB hoitaa transaktioiden atomisuuden hajottamalla kysely pienempiin operaatioihin ja tämän jälkeen suorittaa operaatioita yksi kerrallaan. Näin ollen järjestelmän ei tarvitse huolehtia transaktioiden yhtäaikaisuudesta, mikä parantaa järjestelmän tehokkuutta. Lisäksi VoltDB varmistaa tiedon pysyvyyden esimerkiksi varmuuskopioilla. Näiden seikkojen avulla VoltDB varmistaa ACID-transaktiot (Stonebraker & Weisberg, 2013). MemSQL on NewSQL-tietokantajärjestelmä, joka koostuu kaksitasoarkkitehtuurista. Ylemmällä tasolla on yhdistelmäsolmuja ja niiden alla lehtisolmuja. Yhdistelmäsolmu tietävää lehtisolmujen sijainnit ja jakaa kyselyitä näiden lehtien kesken. MemSQL takaa talletetun tiedon pysyvyyden VoltDB:n tapaan tekemällä tiedosta varmuuskopioita. Yhtäaikaisten transaktioiden kontrollointiin MemSQL käyttää MVCC-protokollaa, jotta tiedon eheys ei kärsisi ja kirjoittaminen ei estäisi lukuoperaatioita (Chen et al., 2016).

On kuitenkin havaittavaa, että jotkin NewSQL-tietokannoiksi luokitellut järjestelmät eivät tue vahvoja ACID-transaktioita, vaan jokin osa vaatimuksesta saatetaan keventää. Esimerkiksi NuoDB käyttää tiedon replikoimiseen asynkronisia operaatioita klusterin solmujen välillä, tehden yhdestä klusterista ajan myötä yhtenäisen (vrt. BASE), eikä aina yhtenäisen (vrt. ACID). (Monteiro, Brayner, & Tavares, 2016).

4. Yhtäaikaisten operaatioiden käsittely hajautetussa tietokannassa

Kuten aikaisemmin olen maininnut, tallennettavan tiedon määrän nopeasta kasvusta johtuen tietokantoja on jouduttu hajauttamaan. Yksi merkittävimmistä ongelmista tähän liittyen on yhtäaikaisten operaatioiden käsittely. Hajautettuun tietokantaan voi tehdä yhtäaikaisesti useita luku- tai kirjoitusoperaatioita yhtä aikaa. Jos jokainen operaatio suoritetaan yksi kerrallaan, hakujen tekeminen on hyvin hidasta. Jos taas operaatioita suoritetaan yhtä aikaa, saatu tieto voi olla vanhentunutta. Jotta voidaan varmistaa tiedon luotettavuus ja tietokannan saatavuus, tulee yhtäaikaisten operaatioiden käsittelyyn keksiä sellainen ratkaisu, jossa uusin tieto on aina saatavilla useissa eri solmuissa. Tässä luvussa tarkastellaan, millaisia ratkaisuja yhtäaikaisten operaatioiden käsittelyyn on kehitetty.

4.1 Lukot

Lukkojen hyödyntäminen operaatioiden käsittelyssä tarkoittaa, että tietty osa, esimerkiksi yksi rivi tai olio lukitaan siksi aikaa, kunnes siihen tehtävä operaatio suoritetaan loppuun. Lukkojen avulla kyetään estämään konfliktit luku- ja kirjoitusoperaatioissa. Tämä kuitenkin estää lukuoperaatioilta pääsyn lukittuun tietoalkioon, mikä voi aiheuttaa huomattavaa viivettä, mikäli tietokantaa käyttää suuri määrä käyttäjiä yhtä aikaa. Suurin osa tämän päivän tietokantajärjestelmistä hyödyntää lukkoja yhtäaikaisten operaatioiden käsittelyssä (Sang, Li, Liu, & Kong, 2012).

Lukitustekniikat voi jakaa optimistisiin ja pessimistisiin tekniikkoihin. Optimistisissa tekniikoissa kohdetta ei lukita koko operaation ajaksi, eli siihen on mahdollista tehdä lukuoperaatioita, vaikka kirjoitus olisi kesken. Ennen kuin kirjoitusoperaatio suoritetaan loppuun, tietokanta tarkastaa, ettei kirjoitusoperaation aikana ole tehty, tai olla tekemässä, uusia kirjoitusoperaatioita. Jos ei, kirjoitusoperaatio suoritetaan loppuun. Muussa tapauksessa operaatio keskeytetään. Tietokanta siis optimistisesti olettaa, ettei kirjoituksen aikana yritetä tehdä uusia kirjoitusoperaatioita. Pessimistisissä tekniikoissa taas tietokanta olettaa, että kirjoitusoperaation aikana siihen tullaan tekemään yhtä aikaa useita kirjoitusoperaatioita, joten operaation kohde lukitaan koko prosessin ajaksi. Näin ollen kohteena olevaan tietueeseen ei voi tehdä mitään operaatioita ennen kuin kirjoitusoperaatio on suoritettu loppuun. Pessimistiset lukot sopivat tietokantoihin, joihin tullaan todennäköisesti tekemään paljon operaatioita, jotka aiheuttavat konflikteja. Nämä lukot voivat kuitenkin herkästi aiheuttaa lukkiutumisia, joissa kaksi tai useampi operaatio jäävät jumiin, koska ne odottavat toisiaan. Optimistiset lukot taas sopivat tietokantoihin, joihin voidaan olettaa konfliktien määrän olevan vähäinen (Gupta, Arora, & Bhati, 2018).

NewSQL-tietokantoihin on suunniteltu uudenlaisia lukitusmenetelmiä, joiden avulla yhtäaikaisia operaatioita kyetään suorittamaan jatkuvasti tietokantaan ilman, että luotettavuus kärsii. Esimerkiksi Google Spanner käyttää ratkaisua, jossa lukkoja käytetään ainoastaan luku- ja kirjoitusoperaatioihin. Lukot eivät vaikuta operaatioihin, jotka ainoastaan lukevat tietoa tietokannasta. Tämä mahdollistetaan tallentamalla useita versioita samasta tietoalkiosta. Yksi versio merkitään turvalliseksi lukea, kun taas muihin versioihin kyetään tekemään kirjoitusoperaatioita. Kun kirjoitusoperaatio on valmis, uudesta tiedosta luodaan taas useita versioita (Grolinger et al., 2013).

4.2 Paxos-tilakone

Jotta hajautettuun tietokantaan tallennettu tieto olisi luotettavaa, kaikilla tietokannan solmuilla tulee olla yhteisymmärrys, eli konsensus, tietoalkioista. Konsensuksen saavuttamiseksi on kehitetty algoritmeja, jotka yleensä saavuttavat konsensuksen, kun yli puolet koko klusterin solmuista on saatavilla. Mahdollisesti kuuluisin konsensusalgoritmi on Leslie Lamportin kehittämä Paxos-algoritmi, joka on saanut viitteitä fiktiivisestä lainsäädäntöprotokollasta Kreikan Paxos-saarella. Algoritmi on kuitenkin hyvin monimutkainen ja se on vaikea ottaa käyttöön, joten algoritmista on tehty useita erilaisia implementaatioita. Yksinkertaistetussa versiossa solmut jaetaan kolmeen eri rooliin. Näitä ovat ehdottaja, hyväksyjä ja oppija. Algoritmi olettaa, että jokainen solmu toimii mielivaltaisella nopeudella ja kuka tahansa solmuista saattaa yhtäkkisesti poistua, esimerkiksi kaatua tai kadota verkosta. Ehdottajat lähettävät kahdenlaisia viestijä hyväksyjille, valmistavia ja varsinaisia ehdotuksia. Lisäksi ehdottaja liittää ehdotuksen pariksi järjestysnumeron. Algoritmi käy läpi kaksi vaihetta. Ensimmäisessä vaiheessa ehdottaja valitsee ehdotuksen ja lähettää valmistavan ehdotuksen suurimmalle osalle hyväksyjistä. Kun hyväksyjä vastaanottaa valmistavan ehdotuksen, se vastaa ehdottajalle lupaamalla, ettei enää hyväksy varsinaisia ehdotuksia, joiden järjestysnumero on pienempi, kuin kyseisen ehdotuksen. Lisäksi hyväksyjä liittää vastaukseen korkeimman ehdotuksen, jonka se on hyväksynyt. Toisessa vaiheessa ehdottaja vastaanottaa viestejä hyväksyjien enemmistöltä. Ehdottaja valitsee varsinaiseksi ehdotukseksi arvon, jonka järjestysnumero on korkein hyväksyjien vastauksista. Ehdottaja lähettää tämän jälkeen varsinaisen ehdotuksen hyväksyjille, jotka hyväksyvät tämän ehdotuksen, mikäli ne eivät ole saaneet uutta valmistavaa ehdotusta, jonka järjestysnumero on vähintään yhtä suuri, kuin kyseisen varsinaisen ehdotuksen. Kun valitusta arvosta on päästy yhteisymmärrykseen, se lähetetään yhdelle oppijalle, joka tiedottaa valitusta arvosta muille oppijoille (Lamport, 2001).

Google Spanner käyttää useita Paxos-tilakoneita käsittelemään luku- ja kirjoitusoperaatioita. Spanner varmistaa tietokannan yhtenäisyyden käyttämällä kaksivaiheista sitouttamista (2PC), joka heikentää tietokannan saatavuutta. Kaksivaiheinen sitouttaminen suoritetaan kuitenkin Paxos-ryhmissä. Ryhmät sisältävät edellä mainitulla tavalla ehdottajia, hyväksyjiä ja oppijoita. Kun tietokantaan halutaan tehdä luku- ja kirjoitusoperaatio, operaatio lähetetään yhdelle ryhmälle. Ryhmä suorittaa edellä mainitun prosessin ja päättää joko suorittaa operaation loppuun tai hylätä sen kokonaan. Koska vain yli puolet ryhmän hyväksyjistä tulee olla saatavilla konsensuksen saavuttamiseksi, vältytään kaksivaiheisen sitouttamisen heikkoudelta, jossa kaikkien klusterin solmujen tulee olla valmiita sitoutumaan operaation loppuun suorittamiseksi. Spanner käyttää useita Paxos-ryhmiä, mikä mahdollistaa useiden luku- ja kirjoitusoperaatioiden tekemisen yhtä aikaa, kunhan ne eivät koske samaa taulua (Corbett et al., 2013).

Paxos-algoritmin huono puoli on sen monimutkaisuus. Edellä mainittu prosessi on yksinkertaistettu versio, jossa on vain kaksi vaihetta. Alkuperäisessä versiossa vaiheita on useampia ja niitä suoritetaan yhtä aikaa. Paxos-algoritmin kokonaisvaltainen ymmärtäminen vaatii hyvin paljon omistautumista ja vaivaa. Monimutkaisuudesta johtuen se on hyvin vaikeasti implementoitavissa. Tästä johtuen algoritmista on tehty useita erilaisia sovelluksia eri käyttötarkoituksia varten. Kaikki nämä sovellukset pohjautuvat Paxos-algoritmiin, mutta eroavat merkittävästi toisistaan. Tällaisia sovelluksia ovat esimerkiksi Chubby ja Raft (Stonebraker, M., 2010).

4.3 Moniversiointi

Multi-Version Concurrency Control (MVCC) on hyvin yleinen yhtäaikaisten operaatioiden käsittelymenetelmä. Tämä menetelmä mahdollistaa sen, että luku- ja kirjoitusoperaatiot eivät koskaan lukitse toisiaan. Menetelmä on siis optimistinen. Kun tietokantaan tehdään kirjoitusoperaatio, sen lisäämään tai päivittämään riviin lisätään järjestysnumero. Tapahtumista pidetään tarpeeksi pitkää historiaa, jotta lukuoperaatiot kykenevät käyttämään yhtä sitoutettua tilannekuvaa. Menetelmässä on kaksi tasoa, joiden avulla se kykenee eristämään luku- ja kirjoitusoperaatiot. *Latest Commited*-tasolla luetaan uusimmasta sitoutetusta versiosta, joka ei ole kirjoituslukossa. *Snapshot*-tasolla luetaan uusimmasta versiosta, joka ei ole kirjoituslukossa, vaikka se ei olisi sitoutettu. Näistä kahdesta tasosta huomataan, että menetelmä palvelee ainoastaan lukuoperaatioita. Yhtäaikaisia kirjoitusoperaatiota ei voida suorittaa loppuun, mikäli samaan tauluun ollaan tekemässä kirjoitusoperaatiota (Neumann, Mühlbauer, & Kemper, 2015).

Google Megastore hyödyntää multiversiointia ja tiedon replikointia saavuttaakseen ACID-transaktiot. Järjestelmä tarjoaa luku- ja kirjoitusoperaatioiden tekemisen nykyiseen kopioon, tilannekuvaan tai eheettömään kopioon. Jos käyttäjä haluaa suorittaa operaation nykyiseen kopioon, järjestelmä varmistaa, että kaikki tietoalkioon sitoutetut kirjoitusoperaatiot on saatu valmiiksi. Tämän jälkeen järjestelmä suorittaa lukuoperaation loppuun. Tämä varmistaa aina tiedon eheyden, mutta saatavuus kärsii, mikäli tietoalkioon ollaan tekemässä paljon kirjoitusoperaatioita. Kun suoritetaan operaatio tilannekuvaan, järjestelmä etsii uusimman kopion, johon ei olla tekemässä kirjoitusoperaatiota. Mikäli lähin kopio on lukittu, se hakee tiedon seuraavaksi lähimmästä kopiosta. Näin ollen luku- ja kirjoitusoperaatioiden tekeminen on mahdollista samaan tietoalkioon. Mikäli käyttäjällä on suuri tarve saada uusin, vaikkakin vahvistamaton, tieto käsiinsä, on hänellä mahdollisuus tehdä lukuoperaatio eheettömään kopioon. Tällöin operaatio suoritetaan aina uusimpaan tietoon, vaikka kirjoitusoperaatio olisi kesken. Tämä mahdollistaa tehokkaiden operaatioiden suorittamisen loppuun, vaikkakin tieto ei ole varmasti uusinta. Jokaisessa eri mallissa uusi kirjoitettu tieto replikoidaan kaikille klusterin solmuille. (Baker et al., 2011)

5. Yhteenveto

Tämän tutkielman tarkoituksena oli selvittää, kuinka NewSQL-tietokannat kykenevät saavuttamaan tehokkaan skaalautuvuuden luopumatta ACID-transaktioista. Lisäksi tarkoituksena oli selvittää, kuinka NewSQL-tietokannat pitävät tallennetun tiedon eheänä. Ratkaisuja on monenlaisia, mutta suosituimpana ratkaisuna on tallentaa samankaltaista tietoa yhteen klusteriin ja suorittaa transaktiot atomisesti klusterin sisällä. NewSQL-tietokantojen pohjalla on perinteinen relaatiotietokanta, jonka päälle on rakennettu hallinnointikerros pitämään huolen tiedon eheydestä. Lisäksi klustereiden solmut on rakennettu käyttäen hyväksi ”shared-nothing”-arkkitehtuuria, eli ne eivät jaa muistia muiden solmujen kanssa. Näin ollen ainoastaan kyseinen solmu voi suorittaa operaatioita siihen tallennettuun tietoon. NewSQL-tietokantajärjestelmät hyödyntävät yhtäaikaisten operaatioiden käsittelyssä erilaisia lukkoja, moniversiointia ja konsensusalgoritmeja takaamaan tietokannan saatavuus ja luotettavuus, vaikka tietokanta olisi hajautettu.

Vaikkakin NewSQL-tietokannan luokittelussa mainitaan ACID-transaktioiden tukeminen, osa NewSQL-tietokannoista joustaa tästä vaatimuksesta keventämällä jostakin ACID:in osasta. Tämä tekee tietokannasta tehokkaamman, mutta epäluotettavamman verrattuna sellaisiin NewSQL-tietokantoihin, jotka tukevat vahvoja ACID-transaktioita. Näiden tietokantojen kevennetyn ominaisuuden vaikutukset tietokannan luotettavuuteen tulisi selvittää, jotta kyettäisiin varmistamaan, että nämä tietokannat ovat oikeasti NewSQL-tietokantoja, eivätkä NoSQL-tietokantoja, joihin on implementoitu joitakin ACID:in ominaisuuksia. Tämä olisi tärkeää, jotta NewSQL-tietokannan määritelmä olisi selkeä ja yritykset voisivat olla varmoja tietokantajärjestelmän luotettavuudesta.

Tutkimuksessa ongelmaksi muodostui edellisten tutkimusten suppea määrä. Koska NewSQL on melko tuore käsite, tehtyjä tutkimuksia ei ole paljoa. Suurin osa tutkimuksista vertaa NewSQL-tietokantoja relaatiotietokantoihin tai NoSQL-tietokantoihin ainoastaan tehokkuuden osalta ja luotettavuus jätetään pois. Lisäksi tutkimuksissa ei ole käsitelty ACID-transaktioiden tukemista.

Tässä tutkielmassa olen käsitellyt ACID-transaktioihin liittyviä ominaisuuksia. Jatkotutkimuksena tämän tutkielman pohjalta olisi mahdollista tehdä käytännön tutkimus, jossa peilataan relaatiotietokantoja tällä hetkellä käytössä oleviin NewSQL-tietokantajärjestelmiin ja tarkastella, kuinka tehokkaasti nämä järjestelmät kykenevät takaamaan ACID-transaktiot. Lisäksi olisi mahdollista tutkia näiden järjestelmien skaalautuvuutta peilaten NoSQL-tietokantajärjestelmiin. Vaikka NewSQL-tietokantajärjestelmät sisältävät erilaisia keinoja virhetilanteiden hoitamiseen, kaikkia virheitä ei voida saada kiinni pelkästään sisäänrakennetuilla työkaluilla. Tästä johtuen näiden järjestelmän käyttöympäristöä tulee tutkia, jotta kyetään valmistautumaan mahdollisimman useaan virhetilanteeseen riskien madaltamiseksi.

6. Lähdeluettelo

Ahsan, S. B., & Gupta, I. (2016). (2016). The cat theorem and performance of transactional distributed systems. Paper presented at the Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, doi:10.1145/2955193.2955205 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85054864921&doi=10.1145%2f2955193.2955205&partnerID=40&md5=bbb0a84952a9748304a7f05949cd986b

Aslett, M. (2011). How will the database incumbents respond to NoSQL and NewSQL.

Bailis, P., & Ghodsi, A. (2013). Eventual consistency today: Limitations, extensions, and beyond. Communications of the ACM, 56(5), 55-63. doi:10.1145/2447976.2447992

Baker, J., Bond, C., Corbett, J. C., Furman, J. J., Khorlin, A., Larson, J., . . . Yushprakh, V. (2011). Megastore: Providing scalable, highly available storage for interactive services.

Boicea, A., Radulescu, F., & Agapin, L. I. (2012). (2012). MongoDB vs oracle--database comparison. Paper presented at the 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, 330-335.

Cattell, R. (2010). Scalable SQL and NoSQL data stores. SIGMOD Record, 39(4), 12-27. doi:10.1145/1978915.1978919

Chamberlin, D. D., & Boyce, R. F. (1974). (1974). Sequel: A struciured english query language. Paper presented at the Proceedings of the ACM SIGMOD International Conference on Management of Data, 249-264. doi:10.1145/800296.811515 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85034850329&doi=10.1145%2f800296.811515&partnerID=40&md5=f9d197b3240b7e45defdd62d8a723a48

Chen, J., Jindel, S., Walzer, R., Sen, R., Jimsheleishvilli, N., & Andrews, M. (2016). The MemSQL query optimizer: A modern optimizer for real-time analytics in a distributed database. Proceedings of the VLDB Endowment, 9(13), 1401-1412.

Codd, E. F. (1990). The relational model for database management: Version 2 Addison-Wesley Longman Publishing Co., Inc.

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., . . . Hochschild, P. (2013). Spanner: Google's globally distributed database. ACM Transactions on Computer Systems (TOCS), 31(3), 8.

Elmasri, R., & Navathe, S. (2014). Fundamentals of database systems (Sixth edition, Pearson new international edition ed.). Harlow: Pearson Education. Retrieved from https://oula.finna.fi/Record/oula.1471686

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. M. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing, 2(1) doi:10.1186/2192-113X-2-22

Gupta, M. K., Arora, R. K., & Bhati, B. S. (2018). Study of concurrency control techniques in distributed DBMS. International Journal of Machine Learning and Networked Collaborative Engineering, 02, 180–187. Retrieved from https://www.researchgate.net/profile/Manoj_Gupta51/publication/333394671_Study_of_Concurrency_Control_Techniques_in_Distributed_DBMS/links/5ceb7a79a6fdccc9ddd22ec1/Study-of-Concurrency-Control-Techniques-in-Distributed-DBMS.pdf

Haerder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR), 15(4), 287-317. doi:10.1145/289.291

Lamport, L. (2001). Paxos made simple. ACM Sigact News, 32(4), 18-25.

Leavitt, N. (2010). Will NoSQL databases live up to their promise? Computer, 43(2), 12-14.

Martin, J. (1976). Principles of data-base management.

Minsky, N. (1974). Another look at data-bases. ACM SIGMOD Record, 6(4), 9-17. doi:10.1145/983082.983084

Neumann, T., Mühlbauer, T., & Kemper, A. (2015). (2015). Fast serializable multi-version concurrency control for main-memory database systems. Paper presented at the Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 677-689.

Pavlo, A. (2016). What's really new with NewSQL? ACM SIGMOD Record, 45(2), 45-55. Retrieved from https://oula.finna.fi/PrimoRecord/pci.acm3003674

Petticrew, M. (2001). Systematic reviews from astronomy to zoology: Myths and misconceptions. British Medical Journal, 322(7278), 98-101. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-0035852458&partnerID=40&md5=8d8428c697fbe89c46af15a5592617db

Pokorny, J. (2013). NoSQL databases: A step to database scalability in web environment. International Journal of Web Information Systems, 9(1), 69-82. doi:10.1108/17440081311316398

Pritchett, D. (2008). Base an acid alternative. Queue, 6(3), 48-55. doi:10.1145/1394127.1394128

Samaras, G., Britton, K., Citron, A., & Mohan, C. (1995). Two-phase commit optimizations in a commercial distributed environment. Distributed and Parallel Databases, 3(4), 325-360.

Sandborg, J. (2016). NoSQL tietokannat. Helsingin Yliopisto.Tietojenkäsittelytieteen Laitos.Haettu, 22(5)

Sang, C., Li, Q., Liu, Z., & Kong, L. (2012). VGL: Variable granularity lock mechanism in the shared storage multi-tenant database. Emerging computation and information teChnologies for education (pp. 481-487) Springer.

Silberschatz, A., & Kedem, Z. (1980). Consistency in hierarchical database systems. Journal of the ACM (JACM), 27(1), 72-80.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database system concepts (6. ed ed.). New York: McGraw-Hill. Retrieved from https://oula.finna.fi/Record/oula.1077545

Silva, Y. N., Almeida, I., & Queiroz, M. (2016). (2016). SQL: From traditional databases to big data. Paper presented at the Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 413-418.

Silvestre, G., Sauvanaud, C., Kaaniche, M., & Kanoun, K. (2015). In Fantechi A., &Pelliccione P.(Eds.), Tejo: A supervised anomaly detection scheme for newSQL databases Springer Verlag. doi:10.1007/978-3-319-23129-7_9 Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84945955133&doi=10.1007%2f978-3-319-23129-7_9&partnerID=40&md5=c26e9ccc8c48531a6295a87a081f1c29

Stonebraker, M. (2010). In search of database consistency. Communications of the ACM, 53(10), 8-9. doi:10.1145/1831407.1831411

Stonebraker, M., & Cattell, R. (2011). Ten rules for scalable performance in simple operation datastores. Communications of the ACM, 10

Stonebraker, M., & Weisberg, A. (2013). The VoltDB main memory DBMS. IEEE Data Eng.Bull., 36(2), 21-27.

Strauch, C., Sites, U. S., & Kriha, W. (2011). NoSQL databases. Lecture Notes, Stuttgart Media University, 20

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). (2010). A comparison of a graph database and a relational database: A data provenance perspective. Paper presented at the Proceedings of the 48th Annual Southeast Regional Conference, 42.