Artturi Laine

RELAATIOMALLIN VERTAILUA VAIHTOEHTOISIIN TIETOKANTAMALLEIHIN

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2024

TIIVISTELMÄ

Laine, Artturi
Relaatiomallin vertailua vaihtoehtoisiin tietokantamalleihin
Jyväskylä: Jyväskylän yliopisto, 2024, 34 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Kuusio, Ari

Relaatiomalli on pitkään ollut hallitseva tietokantamalli tietokannanhallintajärjestelmien alalla. Tietokantoja kohtaan muuttuneet tarpeet etenkin massadatan suhteen ovat luoneet vaatimuksia, joita relationaaliset tietokannanhallintajärjestelmät eivät pysty palvelemaan tehokkaasti. Vaihtoehtoisien tietokantamallien suosion nousun myötä relaatiomallin aseman oikeellisuutta suosituimpana tietokantamallina on mahdollista kyseenalaistaa. Tässä tutkielmassa tarkastellaan relaatiomallin, NoSQL:n ja NewSQL:n ominaisuuksia ja vertaillaan niitä keskenään. Keskeisenä havaintona huomattiin, että eri tietokantamallit täydentävät toisiaan korvaamisen sijaan. Tutkielman toteutuksesta teki haasteellista se, että monia tietokannanhallintajärjestelmiä on haastavaa ellei jopa mahdotonta kategorisoida yhteen tietokantamalliin.

Asiasanat: Tietokanta, relaatiomalli, tietokantamalli, NoSQL, NewSQL, tietokannanhallintajärjestelmä

ABSTRACT

Laine, Artturi
Comparing the relational model to alternative database models
Jyväskylä: University of Jyväskylä, 2024, 34 pp.
Information Systems, Bachelor's Thesis
Supervisor: Kuusio, Ari

The relational model has been the leading database model within database management systems for a long time. The evolving needs for databases, especially regarding big data, have created requirements that relational database management systems can't fulfill efficiently. The rise of the alternative database models' popularity may raise doubt about the rightfulness of the relational model's position as the most popular database model. In this thesis, the features of the relational model, NewSQL and NoSQL are explained, and then compared with each other. A notable finding was that these database models complement each other instead of replacing. Carrying out this thesis was made challenging by the fact that a lot of database management systems are difficult if not impossible to categorise as a single database model.

Keywords: Database, relational model, dbms, database model, NoSQL, NewSQL

TAULUKOT

SISÄLLYS

1 JOHDANTO

Tietokannat ovat olennainen osa nykyaikaista tietotekniikkaa ja tietojenkäsittelyä, sillä tiedon merkitys nyky-yhteiskunnassa on erittäin suuri, tietomäärät ovat valtavia ja tietoa tallennetaan tyypillisesti tietokannoissa. Tietokannat ovat organisoituja kokoelmia tiedosta, mitkä on suunniteltu siten, että tietoa on helppo hakea, muokata ja hallinnoida. *Tietokannanhallintajärjestelmät* (TKHJ, eng. Database Management System, DBMS) ovat ohjelmistoja, joilla tietokoneellistettuja tietokantoja voidaan luoda, muokata ja operoida (Elmasri & Navathe, 2016, s. 6). TKHJ:istä ja niiden hallinnoimista tietokannoista puhutaan usein puhekielessä tietokantoina (Taipalus 2024, s. 2), vaikka tarkoitetaankin tietokannanhallintajärjestelmän ja tietokannan muodostamaa kokonaisuutta eli tietokantajärjestelmää. TKHJ:t kategorisoidaan niiden rakenteen mukaan, ja tietyissä määrin myös niiden käyttötarkoituksen mukaan (Gartner, 2024).

Relaatiomallin toi markkinoille E. F. Codd vuonna 1970 (Codd, 1970), ja se on edelleen suosituin saatavilla olevista tietokantamalleista (DB-Engines, 2024). Kyseessä on kuitenkin hyvin vanhaa tekniikkaa suhteessa nopeasti kehittyvään tietotekniikanalaan, ja vuosikymmenten varrella tietokantojen vaatimukset ovat muuttuneet merkittävästi etenkin sekä käsiteltävän datan (Khan ym., 2023, s. 1), että tietokantaa yhtäaikaisesti käyttävien käyttäjien (Pavlo & Aslett, 2016, s. 1) määrän suhteen. Relaatiomallille vaihtoehtoisista malleista NewSQL ja varsinkin NoSQL ovat nostaneet suosiotaan merkittävästi viimeisen kymmenen vuoden aikana (DB-Engines Ranking, 2024). NoSQL on enemmänkin useamman tietokantamallin sisältävä lähestymistapa tietokantasuunnitteluun kuin yksittäinen tietokantamalli, mutta tässä tutkielmassa NoSQL:ää käsitellään yhtenä tietokantamallina vertailun helpottamiseksi. Relaatiotietokantojen suosio on samaan aikaan hieman vähentynyt, mutta suurta muutosta ei ole havaittavissa.

Vaihtoehtoisten mallien suosion nousun vuoksi on syytä tarkastella, mitä eroavaisuuksia eri tietokantamalleilla on, ja mitä syitä tietyn mallin valintaan nykyisin löytyy. Tässä tutkielmassa relationaalisille tietokannoille vaihtoehtoina tarkastellaan NoSQL- ja NewSQL-tietokantoja. Muitakin vaihtoehtoja on olemassa, kuten vektori- ja oliotietokannat, mutta niiden tarkempi käsittely jätetään tutkielmasta pois aiheen laajuuden rajaamiseksi. Valitut vaihtoehtoiset

tietokannat nähdään varteenotettavimpina vaihtoehtoina relaatiotietokannoille. Objektiorientoituneet tietokannat eivät koskaan saavuttaneet laajaa suosiota, mutta monia niiden tuomista ideoista kuitenkin aikanaan hyödynnettiin sekä relationaalisissa että NoSQL-tietokannoissa (Pavlo & Aslett, 2016, s. 45). Pavlon ja Aslettin (2016, s. 46) mukaan NewSQL-tietokannat voidaan luokitella yhdeksi relaatiomallin luokista, mutta tässä tutkielmassa relaatiomallia ja NewSQL:ää käsitellään toisistaan erillisinä malleina. Myös Taipalus (2024) käsittelee niitä erikseen sen sijaan, että NewSQL olisi vain relaatiomallin alaluokka.

Tässä tutkielmassa vertaillaan relaatiomallia vaihtoehtoisiin tietokantamalleihin. Tutkimusongelmaa selvitetään seuraavien tutkimuskysymysten avulla:

1. Millaisia ominaispiirteitä perinteisellä relaatiomallilla on, ja mitkä ovat tämän mallin vahvuuksia ja heikkouksia?
2. Millaisia vaihtoehtoisia malleja relaatiomallille voidaan nähdä, ja mitkä ovat näiden mallien vahvuuksia ja heikkouksia?
3. Miten relaatiomalli vertautuu vaihtoehtoisiin tietokantamalleihin?

Tutkielmassa noudatetaan seuraavia rajauksia. Valittujen tietokantamallien ominaispiirteitä vertaillaan yleisellä tasolla menemättä tarkemmin eri TKHJ-tuotteiden (esim. Oracle, MySQL, MongoDB) yksityiskohtiin. Tutkielmassa ei vertailla eri tietokantamalleista aiheutuvia kuluja toisiinsa.

Tutkielma toteutetaan kuvailevana kirjallisuuskatsauksena, jonka tarkoituksena on tarjota lukijalle laaja kuva käsiteltävästä aiheesta (Salminen, 2011, s. 6). Tiedonhaussa on käytetty apuna JYKDOKia, Scopusta ja Google Scholaria. Lähteiksi valikoitui erityisesti JUFO-luokituksen omaavia julkaisuja sekä työelämässä luotettavaksi tunnistettuja tekijöitä kuten IBM, Gartner ja MongoDB. Keskeisiä hakusanoja lähteitä etsiessä ovat olleet: “NoSQL”, “NewSQL”, “SQL”, “database”, “RDBMS”, “relational model”.

Tutkielma etenee seuraavasti. Toisessa luvussa tarkastellaan relaatiomallin piirteitä ja ominaisuuksia, minkä jälkeen kolmannessa luvussa siirrytään valittujen vaihtoehtoisten tietokantamallien, NoSQL:n ja NewSQL:n, keskeisiin piirteisiin ja ominaisuuksiin. Neljännessä luvussa vertaillaan esiteltyjen mallien keskeisiä piirteitä. Tutkielma päättyy yhteenvetolukuun, jossa tiivistetään tutkielmassa tehdyt havainnot ja esitetään niiden pohjalta jatkotutkimusaiheita.

Tämän tutkielman perusteella ei ole mahdollista todeta yhden tietokantamallin olevan ehdottomasti muita parempi. Sen sijaan eri tietokantamallien ominaisuuksia, vahvuuksia ja heikkouksia vertaillaan monesta eri näkökulmasta. Vertailun tulokset ovat yleisluontoisia ja suuntaa antavia, sillä tietokannanhallintajärjestelmien kehitys on jatkuvaa, ja paras mahdollinen TKHJ-tuote kuhunkin käyttötarkoitukseen voi olla haastavaa määrittää. Kaikkia TKHJ-tuotteita ei myöskään ole helppoa lokeroida tiettyyn kategoriaan kuten Taipaluksen (2024) kirjallisuuskatsauksesta käy ilmi, minkä lisäksi Pavlo ja Aslett (2016, s. 54) uskovat TKHJ:ien kehittyvän niin monipuolisiksi, että niiden kategorisoinnista tulee turhaa.

2 RELAATIOMALLIN OMINAISUUKSIA

Tämän luvun aluksi tarkastellaan relaatiomallin kehitysvaiheita, minkä jälkeen siirrytään relaatiomallin keskeisiin piirteisiin. Tämän jälkeen tarkastellaan relaatiomallin yleisiä käyttökohteita ja luku päättyy yhteenvetoon.

2.1 Relaatiomallin kehitysvaiheita

Relaatiomalli julkaistiin aikaisessa vaiheessa TKHJ:ien elinkaarta, mutta ensimmäinen tietokantamalli se ei kuitenkaan ollut. Charles Bachmanin vuonna 1963 julkaisemaa IDS:ää (Integrated Data Store) voidaan pitää historian ensimmäisenä TKHJ:nä, vaikka TKHJ vakinaistui terminä vasta myöhemmin (Haigh, 2016, s. 26). IDS:n käyttämä *tietomalli*[1] (eng. data model) määriteltiin vasta myöhemmin verkkomalliksi (eng. network model) (Haigh, 2016, s. 29). Toinen suosittu tietomalli aikaisissa tietokantajärjestelmissä oli hierarkkinen malli. Verkkomallia tai hierarkkista tietokantamallia noudattavat tietokannat luokitellaan navigationaalisiksi (eng. navigational) tietokannoiksi (DB-Engines, 2024). Navigationaalisissa

[1] Tsichritzis ja Lochovsky (1976, s. 106) määrittelevät tietomallin menetelmänä, jota noudattamalla data voidaan järjestää loogisesti kyseisestä datasta löytyvien yhteyksien perusteella. Elmasrin ja Navathen (2016, s.32) määritelmän mukaan tietomalli on kokoelma konsepteja, jolla voidaan kuvailla tietokannan rakenne. Tässä tutkielmassa tietomallilla tarkoitetaan mallia, joka kuvastaa tietokantaan tallennettavan datan rakennetta loogisella tasolla, vaikka suuri osa käsiteltävistä tietomalleista kuvastaakin suuntaa antavasti tietokannan rakennetta myös fyysisellä tasolla (Elmasri & Navathe, 2016, s.33). Usein tietokantamalli on rajoitettu käyttämään samannimistä tietomallia, mutta tietomallin käyttämistä ei ainakaan teorian tasolla ole rajoitettu tietyn tietokantamallin kanssa käytettäväksi.

tietokannoissa datajoukkoihin pääsee käsiksi ainoastaan linkitettyjen tietueiden kautta (DB-Engines, 2024). Silberschatz ym. (1996, s. 107) kuvailevat näitä kahta mallia samankaltaisiksi sillä erolla, että hierarkkisessa mallissa linkkien tulee muodostaa puurakenne, kun taas verkkomalli sallii mielivaltaisia käyriä (eng. arbitrary graphs). Elmasrin ja Navathen (2016, s. 51) mukaan monet perinteisistä järjestelmistä (eng. legacy systems) nojautuvat nykyäänkin verkkomalliin tai hierarkkiseen malliin pohjautuviin tietokantajärjestelmiin. Nykyään nämä mallit voidaan tuntea myös perinteisinä tietomalleina (eng. legacy data model) (Elmasri & Navathe, 2016, s. 53). Relaatiomallia edeltäneet perinteiset mallit eivät ole hävinneet markkinoilta täysin, mutta nykypäivänä kyseisiä malleja edustavista TKHJ:istä suosituin, hierarkkinen IMS, on marraskuussa 2024 TKHJ:ien suosiota mittaavalla listauksella vasta sijalla 142 (DB-Engines Ranking, 2024). Tämän lisäksi perinteisten mallien markkinaosuus tietokannanhallintajärjestelmien keskuudessa tippui 10,1 prosentilla vuonna 2022 vuoteen 2021 verrattuna, kun samaan aikaan TKHJ:ien markkina-arvo kokonaisuudessaan kasvoi 91 miljardiin dollariin, mikä tarkoittaa 14,4 prosentin kasvua (Gartner, 2023).

Yksi suurimmista ongelmista aikaisissa tietokantajärjestelmissä oli tietokannan fyysisen ja loogisen rakenteen sekoittaminen. Fyysisen ja loogisen rakenteen sekoittuessa tietokannassa tehtävät kyselyt ja transaktiot saattoivat olla hyvinkin tehokkaita, jos tietokanta oli suunniteltu niiden käsittelyä varten. Tämä lähestymistapa ei kuitenkaan tarjonnut tarpeeksi joustavuutta silloin, kun kyselyiden tarpeet tai sovelluksen vaatimukset muuttuivat. Toinen aikaisten tietokantajärjestelmien merkittävistä puutteista oli se, että ne tarjosivat rajapinnan ohjelmointikielille, mutta kyselykielten käyttäminen ei ollut mahdollista. Kyselykieli on ohjelmointikieli, jota käytetään tietokannan muokkaamiseen ja sieltä tiedon noutamiseen. Tämä johti siihen, että uusien kyselyiden ja transaktioiden luominen oli sekä aikaavievää että kallista. (Elmasri & Navathe, 2016, s. 23-24)

Relaatiomallin julkaisi IBM:n tutkija Edgar F. Codd vuonna 1970 (Elmasri & Navathe, 2016, s. 149; Codd, 1970). Ennen relaatiomallia tietokannat olivat tietomalliltaan tyypillisesti navigationaalisia (Codd, 1970, s. 1; Haigh, 2016). Nämä järjestelmät olivat monimutkaisia ja vaikeakäyttöisiä, ja Codd (1970, s. 1) motivoikin relaatiomallin kehittämistä sillä, että tietokantojen käyttäjien ei tarvitsisi tuntea tietokannan fyysistä rakennetta. Toisin kuin kilpailevat mallit, relaatiomalli kehitettiin teoriapohjaisesti täysin irrallaan tietokannoista, kun taas kilpailevat mallit julkaistiin valmiin järjestelmän ohessa.

Relaatiomallin käyttöönottoa jarruttivat Haighin (2016, s. 29) mukaan sen tietokoneilta vaatima tehokkuus, ja mallin käyttöönottoon liittyneet "äärimmäisen haastavat" ongelmat. Haigh (2016) ei kuitenkaan kuvaile ongelmia tarkemmin. Relaatiomallin kaupallistamisessa tuli vastarintaa monien yritysten suunnalta IBM mukaan lukien, vaikka mallin esittämiä ideoita pidettiinkin hyvinä, sillä kyseisillä yrityksillä oli sijoituksia navigationaalisiin TKHJ:iin, joista luopuminen ei ollut helppoa (Wade & Chamberlin, 2012, s. 39). IBM:n hierarkkinen

TKHJ IMS dominoi TKHJ-markkinoita yli 20 vuoden ajan vuosien 1965 ja 1985 välillä (Elmasri & Navathe, 2016, s. 53). Aikaiset relaatiomallin toteutukset noudattivat relaatiomallia vain osittain, ja osa TKHJ-tuotteista otti relaatiomallista valikoituja ominaisuuksia käyttöön vain siksi, että tuotetta voitaisiin markkinoida relationaalisena (Codd, 1985a, s. 1; Codd, 1985b, s. 53; Wade & Chamberlin, 2016, s. 39-40). *Relationaalisten tietokannanhallintajärjestelmien* (RTKHJ) suosion kasvusta huolimatta, vuonna 1985 yhtäkään olemassaolevaa TKHJ:ä ei voinut vielä Coddin (1985a, s. 2; 1985b, s. 49) mukaan rehellisesti kutsua täysin relationaaliseksi. Tätä kehityssuuntaa vastustaakseen Codd esitti 12 sääntöä, joita TKHJ:n tulee noudattaa ollakseen relationaalinen (Codd, 1985a; Wade & Chamberlin, 2012, s. 40). Esitetyt säännöt ovat nykypäivänäkin käypä standardi järjestelmän relationaalisuuden mittaamisessa, vaikka ne ovatkin aiheuttaneet keskustelua (Wade & Chamberlin, 2016, s. 40).

Relaatiomalli toi mukanaan perustan korkean tason kyselykielille (Elmasri & Navathe, 2016, s. 24; Codd, 1970, s. 1). IBM julkaisi kyselykieli SQL:n (Structured Query Language) muutama vuosi Coddin relaatiomallin julkaisun jälkeen (Chamberlin & Boyce, 1974). SQL luotiin relaatiomallisten tietokantojen datan hallintaan, mutta sillä kesti aikansa pystyä vastaamaan täysin relaatiotietokantojen tarpeisiin, sillä vielä vuonna 1985 SQL ei tukenut relaatioiden muodostamisen kannalta kriittisiä viiteavaimen (eng. Foreign key) ja perusavaimen (eng. Primary key) konsepteja (Codd, 1985b, s. 52). SQL on ollut kyselykielien standardi relaatiotietokannoille jo vuodesta 1986 lähtien (Kelechava, 2018), ja nykyään relaatiotietokantojen sijaan puhutaan usein SQL-tietokannoista niin puhekielessä kuin tieteellisissä julkaisuissakin, vaikka relaatiotietokannan ei tarvitse käyttää tai edes tukea SQL:ää ollakseen relationaalinen.

Kokeellisia relationaalisia järjestelmiä kehitettiin jo 1970-luvun loppupuolella, mutta ensimmäiset kaupalliset RTKHJ:t julkaistiin vasta 1980-luvun taitteessa. Relaatiomallin aikaisimpiin toteuttajiin lukeutuu mm. IBM ja Oracle (ent. RSI), joista jälkimmäisen samanniminen TKHJ, Oracle, on nykyään kaikista suosituin TKHJ (DB-Engines Ranking, 2024). On kuitenkin hyvä huomata, että Oracle-TKHJ tarjoaa relaatiomallin lisäksi muitakin tietokantamalleja saman TKHJ:n alaisuudessa, minkä vuoksi Oraclen relationaalisen TKHJ:n suosion suuruus on haastavaa tietää tarkalleen. Aikaiset RTKHJ:t olivat vielä melko hitaita, mutta tiedontallennus- ja indeksointitekniikoiden kehityksen myötä suorituskykyä saatiin paremmaksi. (Elmasri & Navathe, 2016, s. 24, s. 149)

1980-luvulla erilaiset ohjelmointikielet olivat intensiivisen tutkimuksen kohteena, minkä myötä olio-orientoituneet ohjelmointikielet (eng. object-oriented programming language, OOLP) eli olio-ohjelmointikielet nostivat suosiotaan niin, että termistä "olio-orientoitunut" tuli suosittu muotisana (Wegner, 1990, s. 20). Olio-ohjelmointikielien suosion nousun innoittamana syntyivät myös olio-orientoituneet tietokannanhallintajärjestelmät (eng. Object-Oriented Database Management System, OOTKHJ), jotka suunniteltiin olio-ohjelmointikielellä kehitettyjen sovellusten kanssa yhteensopiviksi, sillä perinteisten tietokantojen

käyttäminen olio-ohjelmointikielellä luotujen sovellusten kanssa voi joskus olla hankalaa (Elmasri & Navathe, 2016, s. 364). Maier ym. (1986, s. 472) perustelivat tarvetta OOTKHJ:n kehittämiselle sekä RTKHJ:ien että muiden tietuepainotteisten TKHJ:ien rajoitteilla, jotka aiheutuvat sallittujen tietotyyppien rajallisuudesta ja datan *normalisoinnin* (eng. normalization) vaatimisesta.

Oliotietokannat on tarkoitettu sovelluksille, jotka vaativat talletetulta datalta tavallista monimutkaisempia rakenteita. Tämänlaisia sovelluksia ovat mm. tietokannat telekommunikaatiota ja multimediaa varten. Oliotietokantoihin tallennettava data voi olla mm. kuvan, videon, äänen tai dokumentin muodossa. (Elmasri & Navathe, 2016, s. 363)

OOTKHJ:t eivät koskaan saavuttaneet erityisen laajaa suosiota, mutta useat niiden tuomista ominaisuuksista nähtiin tarpeellisina, minkä vuoksi monet RTKHJ:t sisällyttivät niitä uusiin versioihinsa. Tämä johti järjestelmiin, joita voidaan kutsua olio-relationaalisiksi tietokannanhallintajärjestelmiksi (eng. object-relational DBMS, ORTKHJ). Olio-ominaisuuksien onnistunut lisääminen relaatiotietokantoihin oli yksi syy OOTKHJ:ien saaman suosion rajallisuuteen. Olio-ominaisuuksia lisättiin SQL:ään ensimmäisen kerran vuonna 1999. Joitakin oliotietokantojen tuomia ominaisuuksia on sisällytetty myös nykyaikaisissa NoSQL-järjestelmissä. (Elmasri & Navathe, 2016, s. 363-364)

Sagiroglun ja Sinancin (2013, s. 42) mukaan vuonna 2003 maailmassa oli dataa viiden eksatavun (10^18 tavua) verran, mutta vuonna 2013 sama määrä dataa tuotettiin kahden päivän aikana. "*How Much Data is Generated Every Day*" -sivun (What's the Big Data, 2024) mukaan vuonna 2024 päivittäin tuotetun datan määrä on 328,77 eksatavua. Massadatan keskeisimmät ominaisuudet voidaan määritellä ns. kolmen V:n avulla. Chen ja Zhang (2014, s. 314) määrittelevät *3V-mallin* seuraavanlaisesti:

- Määrä (*eng. Volume*), jossa määrällä tarkoitetaan datajoukon kokoa.

- Nopeus (*eng. Velocity*), jossa nopeudella tarkoitetaan sisään ja ulos menevän datan nopeutta.

- Monimuotoisuus (*eng. Variety*), jossa monimuotoisuudella tarkoitetaan datan tietotyyppien ja lähteiden laajaa kirjoa.

Chenin ja Zhangin (2014, s. 314) mukaan ihmiset joskus lisäävät malliin myös neljännen V:n omiin erityisvaatimuksiinsa sopien. Khan ym. (2023, s. 2) puhuvat massadatan yhteydessä viidestä V:stä, joista 3V-mallin näkökulmasta uusia ovat arvo (eng. Value) ja todenmukaisuus (eng. Veracity). Madden (2012, s. 4) suosii määritelmää, jossa massadata kuvaillaan liian suurena, liian nopeana ja liian vaikeana senaikaisten työkalujen käsiteltäväksi. Yaqoobin ym. (2016, s. 1243) mukaan perinteinen relaatiomalli ei pysty saavuttamaan massadatan vaatimaa korkean tason suorituskykyä suuria datamääriä käsiteltäessä.

Vuosikymmenten mittaisesta historiastaan huolimatta Dieun ym. (2023, s. 1) mukaan relaatiomallilla on edelleen potentiaalia parantaa.

2.2 Relaatiomallin keskeisiä piirteitä

Relaatiomalli ei koostu pelkästään tiedonvarastoimisen rakenteesta, eli *skeemasta*[2], vaan se koostuu myös mm. datan muokkaamisen säännöistä, sekä tietokannan sisältämän datan eheyden (eng. Data integrity) varmistamisesta (Codd, 1985a, s. 2). Datan eheydellä tarkoitetaan datan loogisuuden ja paikkansapitävyyden lisäksi sitä, että data ei saa muuttua tarkoituksetta.

Relaatiomalli ei itsessään ole pelkästään tietomalli, vaikka siinä täytyykin noudattaa sen omaa, relationaalista tietomallia. Relaatiomallissa tieto tallennetaan tauluihin eli relaatioihin. Jokainen relaation rivi eli *monikko* (eng. tuple) kuvastaa kokoelmaa toisiinsa liittyvistä arvoista. Jokaisessa relaatiossa valitaan perusavaimeksi attribuutti, jonka avulla monikot voidaan tunnistaa toisistaan. Relaatiolla voi olla monta sopivaa vaihtoehtoa perusavaimeksi, mutta yhdellä relaatiolla voi olla vain yksi perusavain. (Elmasri & Navathe, 2016, s. 159-160) Viiteavain on relaation attribuutti, joka viittaa toiseen relaatioon. Viiteavaimeksi käy vain uniikit attribuutit eli viiteavain on usein toisen relaation perusavain, mutta se ei ole välttämätöntä. Sarakkeiden nimiä, eli attribuuttien nimiä, ja relaatioiden nimiä käytetään apuna relaation sisältämien arvojen merkityksen ymmärtämiseksi. Yhden relaation sisältämää joukkoa attribuuttien nimistä kutsutaan relaation otsakkeeksi (eng. header). Attribuuttien arvoiksi sopivaa joukkoa voidaan rajata määrittelyjoukon (eng. domain) avulla. Määrittelyjoukko määrittelee arvot, jotka on mahdollista syöttää attribuutin arvoksi. Esimerkiksi henkilön syntymävuotta kuvaavan attribuutin arvoksi on hyvä rajata vain kokonaisluvut, minkä lisäksi rajauksen voi asettaa tarkemmaksi niin, että tietokannan sisältämän datan kannalta loogisesti mahdottomat syntymävuodet eivät ole sallittuja arvoja. Nykyään relaatiomallia noudattaakseen, relaation tulee noudattaa ensimmäistä normaalimuotoa (eng. First normal form, 1NF), jolloin relaatio ei saa sisältää moniarvoisia attribuutteja (Elmasri & Navathe, 2016, s. 477). Relaatio ei saisi sisältää esimerkiksi attribuuttia “lasten_nimet”, koska attribuutin arvo voisi silloin koostua monesta nimestä yhden sijaan. Tässä tilanteessa relaatio täytyy normalisoida.

Normalisointi (eng. normalization) on prosessi, jonka avulla relaatiot saadaan noudattamaan haluttua normaalimuotoa. Prosessissa relaatioille tehdään testejä, joiden pohjalta relaatiota joudutaan muokkaamaan, ja joissakin tapauksissa jakamaan useampaan relaatioon testin vaatimuksien noudattamista varten. Codd (1972) esitti kolme normaalimuotoa normalisointiprosessin esittämisen yhteydessä, ja nykyään normaalimuotoja on viisi (Elmasri & Navathe, 2016, s. 475).

[2] Tässä tutkielmassa skeemalla tarkoitetaan tietokannan tai sen osan (esim. relaation tai dokumentin) loogista rakennetta.

Mitä edistyneempää normaalimuotoa tietokanta noudattaa, sitä vähemmän tietokanta sisältää tiedon toistoa.

Relaatiotietokannat noudattavat ACID-periaatetta, joka Härderin ja Reuterin (1983) mukaan koostuu seuraavista osista:

- *Atomisuus* (eng. Atomicity). Jokainen transaktio on atominen, mikä tarkoittaa sitä, että jokainen transaktio suoritetaan joko kokonaan tai ei lainkaan. Eli esimerkiksi tilinsiirtotapahtumassa sekä yhdeltä tililtä poistamisen että toiselle tilille lisäämisen täytyy onnistua, tai kumpikin jää tapahtumatta.
- *Eheys* (eng. Consistency). Tietokanta siirtyy eheästä tilasta toiseen eheään tilaan onnistuneen transaktion myötä. Toisin sanoen jokainen transaktio johtaa vain ja ainoastaan hyväksyttäviin tuloksiin.
- *Eristyneisyys* (eng. Isolation). Jokainen transaktio toteutetaan eristyksissä toisistaan. Toisin sanoen yhtä aikaa toteutettavat transaktiot ovat toisiltaan piilossa, eivätkä vaikuta toisiinsa, kunnes transaktio on suoritettu loppuun asti.
- *Pysyvyys* (eng. Durability). Kun transaktio on suoritettu loppuun asti, sen tekemät muutokset ovat lopullisia ja ne pysyvät tallessa, vaikka järjestelmä esimerkiksi kaatuisi.

ACID-periaate ei ole muuttunut vuosien saatossa, mutta Taipaluksen (2024, s. 2) mukaan nykyaikaisissa tietokantatoteutuksissa ainakin eristyneisyydestä voidaan usein tinkiä vaihtelevin määrin.

2.3 Relaatiomallin käyttökohteita

Codd (1970) ei mainitse relaatiomallin julkaisun yhteydessä mitään selkeää kohdeyleisöä tai käyttötarkoitusta relationaalisille tietokannoille. Yksi Coddin (1970) mainitsemista relaatiomallin tavoitteista oli jaettujen tietokantojen käytettävyyden parantaminen tilanteissa, joissa tietokannan rakenteeseen tehdään muutoksia. Relationaalisia tietokannanhallintajärjestelmiä (RTKHJ) on pidetty yleisesti hyväksyttävinä geneerisinä ratkaisuina tiedonhallintaan monenlaisille sovelluksille vuosikymmenten ajan (Vogt ym., 2017, s. 2490). RTKHJ:iä on myös kritisoitu niiden yleisluontoisesta lähestymistavasta tietokantasuunnitteluun, vaikka moniin niistä on onnistuttu sisällyttämään nykyaikaisten vaatimusten mukaisia ominaisuuksia (Valduriez ym., 2021, s. 6).

Relaatiotietokannat toimivat parhaiten valmiiksi tauluihin sopivan rakenteellisen datan, kuten myyntilukujen kanssa (Leavitt, 2010, s. 12). Monimutkaisen datan käsittelyssä RTKHJ:t ovat Valduriezin ym. (2021, s. 6) mukaan tehottomia. Perinteiset relaatiotietokannat on optimoitu käsittelemään kyselyitä, jotka koskevat vain pientä osaa tietokannasta, sekä transaktioita, jotka käsittelevät

muutamia tietueita per relaatio lisäyksiä tai päivityksiä varten (Elmasri & Navathe, 2016, s. 1102-1103). Relaatiomalli palvelee transaktiopainotteista työkuormaa hyvin, mutta tuki analyyttisille kyselyille on vajavaista muihin tarjolla oleviin vaihtoehtoihin verrattuna.

2.4 Yhteenveto

Relaatiomallilla on pitkä ja monivaiheinen historia tietokantateknologian alalla. Relaatiomalli sai heti julkaisuaikanaan positiivista huomiota, mutta ensimmäisten relationaalisten tietokannanhallintajärjestelmien saapuminen markkinoille vei aikansa. Relaatiomalli nousi nopeasti suosituimmaksi tietokantamalliksi kilpailijoidensa rinnalla. Suosio on kestänyt nykypäivään asti, vaikkakin edellisen 15 vuoden aikana tapahtuneet kehitysaskeleet ovat tuoneet markkinoille varteenotettavia kilpailijoita.

Relaatiomalli on kehittynyt tietokantateknologian mukana mukautuen ajan saatossa muuttuneisiin tarpeisiin. Relaatiomallin käyttämisen tukena on olemassa paljon asiantuntijoita ja suuria toimijoita. Pitkälle kehitetyt standardit tekevät relaatiomallista luotettavan niin tietokannan toimivuuden kuin myös odotusarvojen saavuttamisen kannalta. Relaatiomalli tarjoaa laajan valikoiman TKHJ:iä, joiden ominaisuudet palvelevat monia tarpeita hyvin. Relaatiomallin yleisluonteisuus voi luoda haasteita, jos tietokannalle on erityisen tarkat tai erikoiset vaatimukset, koska malli ei ole kovin joustava.

3 VAIHTOEHTOISIA TIETOKANTAMALLEJA

Tässä luvussa esitellään vaihtoehtoisia tietokantamalleja relaatiomallille. Aluksi tarkastellaan perusteita valita jokin muu tietokantamalli relaatiomallin sijaan, minkä jälkeen siirrytään valittujen vaihtoehtoisten tietokantamallien ominaisuuksien ja piirteiden esittelyyn. Vaihtoehtoisista tietokantamalleista esitellään *NoSQL* ja *NewSQL*, jotka ovat saaneet osakseen sekä markkinasuosiota että tuoretta tieteellistä tutkimusta.

3.1 Perusteita vaihtoehtoisille tietokantamalleille

Monet yritykset tarjoavat sovelluksia, jotka säilövät suuria määriä dataa. Esimerkkinä tämänlaisesta sovelluksesta voidaan käyttää ilmaista sähköpostisovellusta, kuten Gmailia. Käyttäjiä sovelluksella voi olla miljoonia, joista jokaisella voi olla tuhansia sähköpostiviestejä. Tällöin on selkeä tarve jonkinlaiselle järjestelmälle sähköpostiviestien hallinnoimista varten. Tähän tarpeeseen perinteinen relationaalinen tietokannanhallintajärjestelmä eli RTKHJ ei kuitenkaan ole optimaalinen ratkaisu, sillä RTKHJ:t tarjoavat paljon palveluita, joista ei ole hyötyä esimerkin kaltaisessa käyttötarkoituksessa. Tämän lisäksi relationaalinen tietomalli saattaa olla liian rajoittava. (Elmasri & Navathe, 2016, s. 884)

Relaatiomallilla koetaan olevan parantamisen varaa massiivisten datamäärien käsittelyn suhteen. Liun ym. (2014, s. 1247) mukaan on olemassa suuria määriä semirakenteellista dataa, jota ei voi tehokkaasti mallintaa relationaalisiksi skeemoiksi. Edellisen vuosikymmenen aikana sekä tiede että kaupalliset toimijat ovat kyseenalaistaneet TKHJ-teknologioiden yleisluonteista, ns. “one-size-fits-all" -luonnetta (Valduriez ym., 2021, s. 6), mikä osaltaan johti NoSQL:n kehittämiseen (Khan ym., 2023, s. 1-2). Tietokantateknologiaa haluttiin kehittää enemmän tiettyjä käyttötarkoituksia varten sen sijaan, että tietokantamallilla pyrittäisiin palvelemaan kaikkia tarpeita.

3.2 NoSQL

NoSQL on relaatiomalliin verrattuna monimuotoisempi, sillä se sisältää usean erilaisen tietomallin sisällään, mutta relaatiomalli ei ole yksi niistä. NoSQL:n tietomallit ovat relationaalista tietomallia yksinkertaisempia (Leavitt, 2010, s. 14). Suurin osa NoSQL-järjestelmistä on hajautettuja tietokantoja, joiden huomio keskittyy semirakenteellisen (eng. semistructured) datan tallettamiseen, korkeaan suorituskykyyn, saatavuuteen (eng. availability), datareplikaatioon ja skaalautuvuuteen (Elmasri & Navathe, 2016, s. 883). Elmasrin ja Navathen (2016, s. 3) mukaan toinen kutsumanimi NoSQL-järjestelmille on massadatajärjestelmä, viitaten NoSQL-järjestelmien kykyyn käsitellä massadataa.

Leavittin (2010, s. 14) mukaan SQL:ää ei pysty käyttämään NoSQL-järjestelmissä, mutta se ei pidä paikkaansa ainakaan enää, sillä nykyään termin tulkitaan tarkoittavan "Not only SQL" sen sijaan, että SQL:ää ei käytettäisi - tai edes voitaisi käyttää - lainkaan (Elmasri & Navathe, 2016, s. 883; Davoudian ym., 2018, s. 4). Leavitt (2010, s. 12) ei näe NoSQL-tietokantoja omana luokkanaan, sillä hänen mukaansa NoSQL-tietokanta tarkoittaa yksinkertaisesti ei-relationaalista tietokantaa. Tämän määritelmän mukaan relaatiomallia edeltäneet verkkomallia ja hierarkkista mallia käyttävät tietokannat ovat NoSQL-tietokantoja. Elmasri ja Navathe (2016, s. 885) huomauttavat, että myös olio- ja XML-tietokannat on mahdollista luokitella NoSQL-tietokannoiksi, mutta tässä tutkielmassa niitä ei käsitellä NoSQL-tietokantoina.

NoSQL-tietokantamallien olennaisia ominaisuuksia ovat *skeematon*[3] (eng. schema-less) rakenne sekä datapresentaatioiden (eng. data representation) tehokkaan ja dynaamisen kasvun salliminen (Khan ym., 2023, s. 1). Skeemattomassa tietomallissa datan tietorakenteet (eng. data structure) voivat olla mielivaltaisia, koska niitä ei määritellä suoraan tiedonmäärittelykielellä (eng. Data Definition Language, DDL). Näiden lisäksi Khan ym. (2023, s. 1) mainitsevat keskeisenä ominaisuutena vaakasuuntaisen skaalaamisen (eng. horizontal scaling) suurikokoisten rykelmien (eng. cluster) ylitse datareplikaatiokokoelmia (eng. data replication collection) ja pirstalointia (eng. sharding) hyödyntäen. NoSQL-järjestelmien "data ensin, skeema myöhemmin tai ei koskaan" -lähestymistapa helpottaa semirakenteellisen datan, kuten dokumenttien, sähköpostiviestien tai laskentataulukkojen käsittelyä (Liu ym., 2014, s. 1247). Davoudianin ym. (2018, s. 4) mukaan NoSQL-tietokantojen käyttämät tietomallit ovat joustavia, mutta skeemattomuus ei ole ehdotonta. Seuraavaksi tarkastellaan NoSQL:n tietomallien keskeisiä ominaisuuksia ja piirteitä. Käsiteltävät tietomallit ovat *avain-arvo -malli, saraketietokannat, dokumenttitietokannat* ja *graafitietokannat*.

NoSQL:n tietomalleista Davoudianin ym. (2018, s. 5) mukaan suosituin ja samalla myös yksinkertaisin on *avain-arvo –malli* (eng. Key-value). Avain-arvo –mallia noudattavissa tietokannoissa data hallinnoidaan ja esitetään "(avain, arvo)"

[3] Skeemattoman tietomallin tietokannassa tietokantaan on mahdollista syöttää dataa, vaikka tietokannan looginen rakenne olisikin keskeneräinen tai kokonaan ilmoittamatta.

-pareissa, jotka on tallennettu tehokkaisiin ja hyvin skaalautuviin avainpohjaisiin hakurakenteisiin (eng. Key-based lookup structure). Tietokantaan tallennettava data on skeematonta eli sillä ei ole ennalta määriteltyä rakennetta tietokannassa. "(Avain-arvo)" –parissa avain on ainutlaatuinen tunniste, jonka avulla arvoon päästään käsiksi. Arvo kuvaa dataa, jonka tyyppi, rakenne ja koko ovat mielivaltaisia. Arvo voi olla tyypiltään esimerkiksi merkkijono, dokumentti tai kuva. Teoriassa avain-arvo –tietokannat eivät tue arvoihin pohjautuvia kyselyitä, mutta käytännössä monet edistyneet avain-arvo –tietokannat mahdollistavat indeksoinnin ja kyselyiden tekemisen tiettyjen tietotyyppien arvoihin pohjautuen. Avain-arvo -TKHJ:istä esimerkiksi Redis ja Aerospike tukevat List-tietotyyppiä. (Davoudian ym., 2018, s. 5)

Saraketietokannat (eng. column-based store, wide column store) osittavat taulun sarakkeiden mukaan sarakeperheisiin (eng. column family), joista jokainen tallennetaan omaan tiedostoonsa (Elmasri & Navathe, 2016, s. 888). Saraketietokannoissa data esitetään tabulaarisesti riveissä ja sarakeperheissä. Sarakeperhe koostuu mielivaltaisesta määrästä toisiinsa loogisesti liittyvistä sarakkeista, jotka yleensä avataan yhtaikaisesti. Tämän takia saraketietokannan data tallennetaan fyysisesti sarakeperheittäin rivien sijaan. Sarakeperheen skeema on joustava, sillä sen sarakkeita voidaan lisätä tai poistaa suoritusaikana. Sarakkeiden ja sarakeperheiden rakenteet huomioon ottaen jokainen rivi kuvastaa erittäin rakenteellista tietoalkiota (eng. data item), jonka ainutlaatuisena tunnisteena on "merkkijono rivi-avain" (eng. string row-key). Davoudian ym. (2018, s. 7) kuvailevat saraketietokantoja laajennettuina avain-arvo –tietokantoina, joissa arvo esitetään sarjana (eng. sequence) sisäkkäisiä "(avain-arvo)" –pareja. (Davoudian ym., 2018, s. 6-7)

Saraketietokantojen datan osittaminen voidaan tehdä tehokkaasti sekä pystysuunnassa (sarakeperheittäin), että vaakasuunnassa (riveittäin) (Davoudian ym. 2018, s. 8). Tämän ominaisuuden vuoksi saraketietokannat ovat soveliaita valtavien datajoukkojen (eng. dataset) tallettamiseen. Ennalta määritelty sarakeperheiden joukko saattaa vaikeuttaa saraketietokantojen käyttämistä sovelluksilla, joiden skeema on monimutkainen ja kehittyvä, sekä sovelluksilla, joissa halutaan vahvaa eheyttä laajan replikaation yhteydessä (Davoudian ym., 2018, s. 9; Corbett ym., 2013, s. 2).

Davoudianin ym. (2018, s. 9) mukaan myös *dokumenttitietokannat* (eng. document store) ovat laajennettuja versioita avain-arvo –tietokannoista, missä arvot esitetään puolirakenteellisina dokumentteina. Dokumentit voivat olla tallennusmuodoltaan mm. XML-, JSON- (JavaScript Object Notation) tai BSON-dokumentteja (Binary JSON) (Davoudian ym., 2018, s. 9; Elmasri & Navathe, 2016, s. 888). Dokumenttitietokannoille ei ole olemassa standardia, mutta Valduriezin ym. (2021, s. 7) mukaan JSON-dokumentit ovat saavuttamassa epävirallista standardin asemaa.

Dokumentin skeema on joustava, sillä sen attribuutteja voidaan lisätä tai poistaa suoritusaikana, jos attribuutilla on nimen lisäksi vähintään yksi arvo. Dokumenttitietokannat tietävät dokumenttiensa tallennusmuodon, minkä vuoksi kyselyiden tekeminen attribuuttien nimien ja arvojen perusteella on mahdollista,

toisin kuin avain-arvo –tietokannoissa. Datan noutaminen dokumentista ilman kokonaisen dokumentin noutamista ja tarkastelua on myös mahdollista. (Davoudian ym., 2018, s. 9-10)

Dokumenttitietokannat sopivat sovelluksille, joiden data voidaan esittää yksinkertaisesti dokumenttien muodossa. Tämänlaisia sovelluksia ovat mm. sisällönhallintajärjestelmät ja bloggausalustat.

Tässä tutkielmassa tähän mennessä esiteltyjen NoSQL-tietokantojen tarkoituksena on ollut informaation tallettaminen entiteeteistä binäärisinä arvoina, riveinä moniulotteisissa tauluissa tai dokumentteina, mutta *graafitietokannoissa* (eng. graph database) data talletetaan graafien muodossa, jotka käsittävät myös datan väliset suhteet (Davoudian ym., 2018, s. 11). Graafitietokannat pohjautuvat vahvaan teoreettiseen perustaan, jossa graafi koostuu entiteettejä kuvaavista solmuista (eng. vertex) ja solmujen välisiä suhteita kuvaavista kaarista (eng. edge) (Davoudian ym., 2018, s. 11). Graafitietokantojen suosion nousu perustuu graafiorientoituneiden datajoukkojen määrän kasvamisella, mikä on luonut tarpeen tehokkaille ER-läpikäynneille (eng. entity-relationship traversal) (Davoudian ym., 2018, s. 11). ER-läpikäynti on prosessi, jossa graafin jokainen solmu tarkistetaan ja/tai päivitetään. Jokaisessa solmussa vieraillaan vähintään kerran, mutta vierailuja voi tulla enemmänkin riippuen läpikäynnin suorittavan algoritmin optimoinnin tasosta.

Muista NoSQL-malleista poiketen graafitietokannat käsittelevät nimenomaan vahvasti toisiinsa liittyviä datajoukkoja. Graafitietokantojen käyttämien talletustekniikoiden perusteella graafitietokannat voidaan jakaa kahteen kategoriaan, ei-natiiveihin ja natiiveihin. Ei-natiivissa graafitietokannassa looginen malli toteutetaan toisen tietokannan päälle, kuten vaikka dokumentti- tai relaatiotietokannan. Ei-natiivit graafitietokannat joutuvat käyttämään indeksejä graafien läpikäyntien suorittamista varten, mikä saattaa varsinkin suurempien graafien kohdalla johtaa suorituskyvyn heikentymiseen. Natiiveissa graafitietokannoissa graafien tallettaminen on graafitietomallien ominaisuuksiin pohjautuen mukautettua. Graafitietomallit, joihin natiivit graafitietokannat pohjautuvat, mahdollistavat luonteeltaan monimutkaisten kyselyiden nopean prosessoinnin ja helpon ilmaisemisen vahvasti toisiinsa liittyviä entiteettejä käsiteltäessä. (Davoudian ym., 2018, s. 12-15)

Graafiprosessointi on ollut kasvavan mielenkiinnon kohteena koneoppimisen ja tekoälyn aloilla (Sakr ym., 2021, s. 64). Graafi-TKHJ:t ovat todella tehokkaita, jos tietokanta voidaan keskittää yhdelle laitteelle ja replikoida muille laitteille. Tehokkuutta menetetään, jos tietokantaa hajautetaan ja ositetaan. (Valduriez ym., 2021, s. 8)

Taulukko 1 on Davoudianin ym. (2018, s. 17) esittämä NoSQL:n tietomalleja vertaileva taulukko suomenkieliseksi käännettynä. Taulukko toimii yhteenvetona NoSQL-tietomallien vahvuuksista, heikkouksista ja sopivista käyttötilanteista.

TAULUKKO 1 NoSQL-tietomallien vertailua.

Tietomalli	Sopivat skenaario(t)	Vahvuudet	Rajoitukset/heikkoudet
Avain-arvo	Dataan käsiksi pääsyyn riittää vain yksi avain, kyselyiden tuloksien välimuistiin tallentaminen (eng. Object caching) ja toisiinsa liittymätön data.	Skaalautuvuus, todella nopea hajasaanti (eng. random access) avaimen kautta, datan osittamisen helppous.	Sovelluksille jäävä vastuu arvojen mallintamisesta, indeksointi ja kyselyiden tekeminen vain avaimiin pohjautuen, käyttäjän täytyy tietää haettavan datan avain kyselyn tehdäkseen.
Sarake	Suurien koottujen datajoukkojen rinnakkainen prosessointi erä-orientoituneesti.	Kyselyiden työkuormalle on suunniteltu aggregaattien hierarkia, joka tekee kyselyiden suorittamisesta tehokkaampaa. Sopiva malli valtavien datamäärien tallettamiseen tehokkaan pystysuuntaisen sekä vaakasuuntaisen osittamisen vuoksi.	Mahdollisuudet "ad-hoc" -kyselyihin ovat rajoitettuja, sillä muutokset sovellustason ominaisuuksiin vaikuttavat malliin merkittävästi; ennaltamääritelty sarakeperheiden joukko vaikeuttaa saraketietokannan käyttämistä sovelluksilla, joiden skeemat kehittyvät.
Dokumentti	Data on jatkuvasti kehittyvää ja helppo esittää dokumentteina.	Rikas tietomalli, johon voi tallentaa mielivaltaisen monimutkaista dataa, kuten sisäkkäisiä rakenteita, taulukkoja ja skalaarisia arvoja; kuhunkin dokumentin osaan voi päästä käsiksi toisarvoisten indeksien avulla.	Ei standardia kyselykieltä tai sovellusrajapintaa.
Graafi	On tarve läpikäydä useita datan välisten suhteiden tasoja syvästi toisiinsa liittyvässä datassa.	Helppo ja nopea kyselyiden tekeminen linkitetyillä datajoukoilla, ER-kaavioiden helppo kuvastaminen.	Ei standardia kyselykieltä tai sovellusrajapintaa. Suurien graafien ositus heikentää suorituskykyä solmujen välisestä kommunikoinnista johtuen.

Davoudian ym. (2018, s. 17)

NoSQL-tietokannanhallintajärjestelmiä on tällä hetkellä jopa yli 225 (Hosting Data, 2023) ja ne tarjoavat monille eri käyttötarkoituksille optimoituja vaihtoehtoja. Davoudian ym. (2018, s. 4) näkevät järjestelmien määrän suurena haasteena heille, jotka haluavat siirtyä käyttämään NoSQL:ää olemassaolevan TKHJ-ratkaisunsa sijaan. Omalle käyttötarpeelleen sopivan NoSQL-järjestelmän valitseminen vaatii seikkaperäistä ymmärrystä niiden tarjoamista ominaisuuksista ja haasteista.

3.3 NewSQL

NewSQL on tietokantamalli, joka pyrkii yhdistämään perinteisen relaatiomallin ja NoSQL:n parhaat puolet. Pavlon ja Aslettin (2016, s. 46) määritelmä NewSQL:stä ilmaisee NewSQL:n yhtenä relaatiomallin luokkana, jonka tarkoituksena on tarjota NoSQL:n skaalautuvuus OLTP-kyselyissä ja samaan aikaan säilyttää ACID-periaate transaktioissa. NewSQL oli n. viisi vuotta sitten Valduriezin ym. (2021, s. 2) mukaan uusinta teknologiaa massadatan hallinnoinnin alalla. NewSQL-järjestelmille tyypillisiä ominaisuuksia Valduriezin ym. (2021, s.

2) mukaan ovat tuki relationaaliselle tietomallille ja SQL:lle, ACID-transaktiot, skaalautuvuus datan osituksen avulla sekä korkea saatavuus datareplikaatiota hyödyntäen. NewSQL-järjestelmiä on montaa tyyppiä, joista jokainen on suunnattu eri työkuormia ja käyttötarkoituksia varten (Valduriez ym., 2021, s. 9). Tästä voisi tehdä johtopäätöksen, että Valduriezin ym. (2021) mukaan yksittäinen NewSQL-järjestelmä ei pyri palvelemaan laajaa asiakaskuntaa yhtaikaisesti.

NewSQL:n ominaisuuksista seuraavia voidaan pitää uusina: skaalautuvat ACID-transaktiot, tuki keskusmuistille, polyvarasto (eng. Polystore) ja tuki HTAP:lle (Hybrid Transactional/Analytical Processing). Jokaista näistä ominaisuuksista voidaan kuvailla sopivaksi johonkin massadatan kolmesta V:stä (*Määrä, Nopeus, Monimuotoisuus*). Polyvarasto on järjestelmä, joka tarjoaa integroidun pääsyn moniin tietovarastoihin, kuten HDFS-tiedostoihin, relationaalisiin tietokantoihin ja NoSQL-tietokantoihin, yhtä tai useampaa kyselykieltä käyttämällä. Polyvarastot ovat tyypillisesti rajoitettu ainoastaan read-kyselyihin, koska hajautettujen transaktioiden tukeminen kyseisessä ympäristössä on todella haastavaa. NewSQL-järjestelmät usein luokitellaan sen ominaisuuden perusteella, jossa ne ovat parhaimmillaan, vaikka yleensä ne tukevatkin useampaa kuin yhtä edellä mainituista ominaisuuksista. (Valduriez ym., 2021, s. 9-10)

HTAP (*Hybrid Transactional/Analytical Processing*) on Gartnerin vuoden 2014 raportissa esittämä sovellusarkkitehtuuri, joka käyttää hyväkseen keskusmuistia hyödyntäviä tietojenkäsittelyteknologioita (eng. in-memory computing technology) mahdollistaakseen OLAP- ja OLTP-työkuormien käsittelyn samanaikaisesti (Li & Zhang, 2022, s. 2483). Tässä tutkielmassa OLAP- ja OLTP-työkuormia tarkastellaan lisää luvussa 4.1.2. Lin ja Zhangin (2022, s. 2483) tulkinnan mukaan Gartnerin myöhemmin tekemä laajennus HTAP:n konseptiin tarkoittaa sitä, että HTAP ei enää ole rajoitettu keskusmuistia hyödyntäviin teknologioihin. HTAP-järjestelmät käyttävät hyväkseen moniversiointia (eng. multiversioning) välttääkseen konflikteja pitkien analyyttisten kyselyiden ja päivitysten välillä (Valduriez ym., 2021, s. 10). Valduriezin ym. (2021, s. 10) mukaan sekä OLTP:n että OLAP:n samanaikaisesti tehokkaaksi saaminen on todella hankalaa, eikä moniversiointi itsessään vielä riitä siihen.

Yksi NewSQL-tietokantateknologian suurimmista kulmakivistä on keskusmuistin hyödyntäminen. *Keskusmuistitietokannat* ovat tietokantoja, joissa tietokannan kaikki data on säilötty keskusmuistiin. Joitakin NewSQL:n ominaisuuksia omaavia keskusmuistitietokantoja on ollut olemassa jo kauan ennen NewSQL:ää (Valduriez ym., 2021, s. 9). Pavlon ja Aslettin (2016, s. 48) mukaan NewSQL toi keskusmuistitietokannoille uutena mahdollisuuden siirtää osa tietokannasta pysyvään muistiin, jolloin TKHJ pystyy tukemaan saatavilla olevaa keskusmuistia suurempia tietokantoja ilman, että tarvitsee vaihtaa levyorientoituneeseen tietokanta-arkkitehtuuriin. On yleisesti hyväksyttyä, että tiedon hakeminen keskusmuistista on vähintään neljä suuruusluokkaa nopeampaa kuin tiedon hakeminen levyltä (Taipalus, 2024, s. 4). Tämä tarkoittaa siis sitä, että jos tiedon hakeminen keskusmuistista kestäisi minuutteja, kestäisi se levyltä hakiessa kuukausia (Taipalus, 2024, s. 4). Taipaluksen (2024, s. 4) mukaan tämän vuoksi TKHJ:iä optimoitaessa pyritään keskusmuistin käytön maksimoimiseen.

Keskusmuistitietokannat tarjoavat huomattavasti nopeampaa tiedonhakua tietokannasta, mutta vastapainona tulee ottaa huomioon suuri riski siitä, että dataa voi hävitä esimerkiksi palvelinvian ilmetessä, koska data on ns. ei-pysyvää. Monet keskusmuistitietokannat tarjoavat nykyään pysyvyysmekanismin tämän riskin lievittämiseksi, mutta monessa tapauksessa tätä riskiä ei silti kannata ottaa. Parhaiten keskusmuistitietokanta sopii tapauksissa, joissa dataa haetaan tietokannasta suurella volyymilla, ja datan häviäminen on hyväksyttävää. On suositeltavaa käyttää tämänlaista tietokantaa vain osassa sovellusta, jolloin tärkeämpi tieto voi olla säilöttynä toisenlaiseen tietokantaan. (MongoDB, 2024)

Analyyttiset keskusmuistitietokannat ovat todella nopeita, mutta tietokannan koko on niiden suurin rajoite, sillä tietokannan kokonaisuudessaan, kaiken apudatan (eng. auxiliary data) ja välitulosten (eng. intermediate results) täytyy mahtua keskusmuistiin (Valduriez ym., 2021, s. 9). Nykyajan muistikapasiteeteilla ja hinnoilla on kuitenkin mahdollista säilöä melko suuriakin tietokantoja keskusmuistiin, lukuunottamatta suurimpia OLTP-tietokantoja (Pavlo & Aslett, 2016, s.48). Pavlon ja Aslettin (2016, s. 48) mukaan keskusmuistiarkkitehtuuria hyödyntävät NewSQL-tietokannanhallintajärjestelmät ovat huomattavasti parempia OLTP-työkuorman käsittelyssä kuin levyorientoituneet järjestelmät.

NewSQL-tuotteet eivät tähän mennessä vielä ole saavuttaneet laajaa suosiota kilpailijoihinsa nähden, kun vertaillaan mitä tietokantamallia eri TKHJ-tuotteet käyttävät. Taipaluksen (2024) TKHJ:ien suorituskykyä vertailua tarkasteleva kirjallisuuskatsaus kattaa 117 TKHJ:iä vertailevaa tutkimusta, joissa käsiteltiin yhteensä 44 TKHJ:ä. Näistä TKHJ:istä Taipalus (2024, s. 8) luokitteli NewSQL-tuotteiksi vain neljä: VoltDB, CockroachDB, MemSQL (nyk. SingleStoreDB) ja NuoDB. Tämä voisi viitata siihen, että tieteellisen tutkimuksen saralla NewSQL-tuotteita ei ole nähty yhtä kiinnostavana kuin NoSQL- ja relationaalisia tuotteita. Taipaluksen (2024) kirjallisuuskatsauksessa käsiteltävistä tutkimuksista valtaosa on julkaistu vuonna 2015 tai myöhemmin, uusimpien julkaisujen ollessa vuoden 2022 maaliskuulta. Pavlo ja Aslett (2016, s. 53) kuvailevat NewSQL:n omaksumista markkinoilla hitaahkoksi varsinkin NoSQL:ään verrattuna. NewSQL-TKHJ:t on suunniteltu palvelemaan transaktiopainotteisia työkuormia, joita löytyy enimmäkseen yritysohjelmistoista (eng. enterprise applications). Pavlo ja Aslett (2016, s. 53) arvelevat, että tämänlaisille ohjelmistoille tehtävät TKHJ-valinnat ovat konservatiivisempia kuin uusien verkkosovellusten työkuormia varten tehtävät valinnat. Tämän lisäksi Pavlon ja Aslettin (2016, s. 53) mukaan NewSQL-TKHJ:iä usein käytetään jo käyttöönotetun RTKHJ:n täydentämiseksi tai korvaamiseksi, kun taas NoSQL:ää otetaan useammin käyttöön kokonaan uusien työkuormien käsittelyyn.

3.4 Yhteenveto

Tämän luvun aluksi tarkasteltiin syitä vaihtoehtoisten tietokantamallien kehittämiseksi. Vaihtoehtoisia malleja ryhdyttiin kehittämään muuttuneiden tarpeiden

vuoksi. Yksi suurimmista tekijöistä tarpeiden muuttumisessa on datan määrän eksponentiaalinen kasvu. Tämän lisäksi tietokantoja kehitettiin enemmän tiettyjä käyttötarkoituksia varten sen sijaan, että tietokannat pyrkisivät palvelemaan mahdollisimman laajasti eri tarpeita. Edellä mainitut asiat yhdistettynä relaatiomallin haasteiden havaitsemiseen etenkin suurien datamäärien käsittelyssä edesauttoivat uusien tietokantamallien kehittämistä.

Vaihtoehtoisista tietokantamalleista tarkasteluun valittiin NoSQL ja NewSQL, joiden ominaisuudet vastaavat hyvin nykyaikaisten tarpeiden mukaisia vaatimuksia. NoSQL-tietokantoja on monenlaisia, eikä yksittäisen tietokannan kuvaileminen NoSQL-tietokantana ole yksinään kovinkaan tarkka määritelmä. NoSQL-tietokantojen saama suosio on ollut huomattavan laajaa, kun taas NewSQL-tietokantojen suosio on ainakin toistaiseksi jäänyt suhteellisen pieneksi. NewSQL pyrkii yhdistämään relaatiomallin ja NoSQL:n parhaat ominaisuudet yhdessä TKHJ:ssä, mutta NewSQL:n esittämä potentiaali ei ole vielä näkynyt TKHJ-markkinoilla merkittävästi. Yksi NewSQL:n rajoitteista on tietokannan koko, sillä kaikkein suurimpia tietokantoja ne eivät pysty säilömään. Molemmat vaihtoehtoisista tietokantamalleista on suunniteltu suurien datamäärien käsittelyyn, mutta etenkin NoSQL suoriutuu “ketterässä” ympäristössä toimimisesta hyvin.

4 TIETOKANTAMALLIEN OMINAISUUKSIEN VERTAILUA JA TULOSTEN POHDINTAA

Tässä luvussa vertaillaan aikaisemmin tutkielmassa esiteltyjen tietokantamallien ominaisuuksia keskenään kategorioittain. Vertailusta tehdään yhteenveto, joka sisältää pohdintaa tehdyistä havainnoista.

4.1 Vertailua

Edellisissä luvuissa tuotiin esille eri tietokantamallien ominaispiirteitä ja käyttökohteita olemassa olevaan kirjallisuuteen pohjautuen. Tämän perusteella tässä alaluvussa vertaillaan relaatiomallia vaihtoehtoisiin tietokantamalleihin. Seuraavaksi tarkastellaan eri tietokantamallien vahvuuksia ja heikkouksia eri näkökulmista.

4.1.1 Hajautettu ympäristö

Hajautettuja tietokantajärjestelmiä on kahdenlaisia: maantieteellisesti hajautettuja ja yksittäisen sijainnin (eng. single location) toteutuksia. Maantieteellisesti hajautetuissa järjestelmissä eri sijainnit ovat yhdistetty laajaverkoilla (WAN), joille pitkät viestiviiveet ja korkeahkot virhemäärät ovat tyypillisiä. Yksittäisen sijainnin hajautettujen järjestelmien solmut sijaitsevat lähellä toisiaan, mikä mahdollistaa paljon nopeamman kommunikoinnin johtaen lyhyempiin viestiviiveisiin ja pienempään määrään virheitä. Yksittäisen sijainnin hajautetut tietokantajärjestelmät ovat rinnakkaisia järjestelmiä, joissa tyypillisesti hyödynnetään tietokonerykelmiä (eng. computer clusters) yksittäisessä datakeskuksessa. Näissä tietokantajärjestelmissä painotetaan rinnakkaisuuden hyödyntämistä sekä suorituskyvyn että saatavuuden parantamiseksi. Tietokantajärjestelmiä voidaan skaalata joko pysty- tai vaakasuunnassa. Pystysuuntaisessa skaalaamisessa olemassa olevaa palvelinta päivitetään tehokkaammaksi esim. muistia ja suorittimia lisäämällä, mikä tarkoittaa sitä, että pystysuuntainen skaalaaminen rajoittuu

palvelimen enimmäiskoon mukaan. Vaakasuuntaisessa skaalaamisessa lisätään palvelimien määrää, jolloin skaalausta voidaan tehdä lähes rajoittamattomasti. Hajautetut tietokantajärjestelmät voivat tarjota molempia skaalaustapoja. Nykyään suurin osa käytössä olevista TKHJ:istä on hajautettuja tai rinnakkaisia. (Valduriez ym., 2021, s. 1-3)

Brewerin (2000) CAP-teoreeman mukaan *jaetun datan tietokannat* (eng. shared-data systems) pystyvät vastaamaan vain kahteen kolmesta esitetystä keskeisestä tarpeesta. Tarpeet ovat seuraavat:

- Eheys (*eng. consistency*)
- Saatavuus (*eng. availability*)
- Osittamisen toleranssi (*eng. partition tolerance*)

Brewer (2000) käytti *hajautettua tietokantaa* (eng. distributed database) yhtenä esimerkkinä tietokannasta, joka luopuu saatavuudesta ja on täten ns. CP-järjestelmä. Khan ym. (2023, s. 2) kuitenkin näkevät CAP-teoreeman olevan nimenomaan hajautetuille järjestelmille tarkoitettu malli, jossa hajautettu järjestelmä voi noudattaa mitä tahansa täsmälleen kahden tarpeen yhdistelmää. Myöhemmin Brewer (2012) toteaa artikkelissaan '*CAP Twelve Years Later: How the "Rules" Have Changed*', että hänen esittämänsä CAP-teoreema on yliyksinkertaistettu, ja että tiukka "kaksi kolmesta" -sääntö on harhaanjohtava. Tästä huolimatta CAP-teoreemaan viitataan toistuvasti tuoreissakin tieteellisissä julkaisuissa. Tässä tutkielmassa käytetyistä lähteistä CAP-teoreema esitellään ainakin seuraavissa: Khan ym. (2023, s. 2-3), Davoudian ym. (2018, s. 34) ja Valduriez ym. (2021, s. 6).

Monet RTKHJ:istä ovat saaneet alkunsa keskitettyinä järjestelminä (Elmasri & Navathe, 2016, s. 49). Anjomshoaan ja Tjoan mukaan (2011, s. 8) relaatiomallia ei ole suunniteltu hajautettuja tietokantoja varten, mutta relationaalisten tietokantojen hajauttaminen on sittemmin mahdollistettu. Hajautettu relationaalinen tietokanta tosin vaatii monimutkaisia ja kalliita protokollia joidenkin yksinkertaisten tehtävien, kuten datan synkronoinnin, suorittamiseen.

NewSQL ja NoSQL ovat molemmat suunniteltu hajautetussa ympäristössä käyttämiseen. Massadatan hallinnoinnin kannalta tietokannan korkea skaalautuvuus on yksi tärkeimmistä ominaisuuksista. Keskitetty ympäristö ei pysty kilpailemaan hajautetun ympäristön kanssa skaalautuvuuden suhteen.

4.1.2 Työkuormien käsittely: OLTP, OLAP ja HTAP

Tietokantojen työkuormat jaetaan usein kahteen kategoriaan: *verkkopohjaiseen transaktioiden käsittelyyn* (eng. Online Transactional Processing, OLTP) ja *verkkopohjaiseen analytiikan käsittelyyn* (eng. Online Analytical Processing, OLAP). OLTP-kategoria sisältää enimmäkseen yksinkertaisia kyselyitä, joissa suoritetaan vain pientä osaa tietokannasta koskevat luku- ja muokkausoperaatiot (Vogt ym., 2017, s. 2490). OLAP-kategoriaan luokiteltavat kyselyt ovat yleensä monimutkaisempia lukuoperaatioita, joiden suorittaminen kestää kauemmin, joissakin tapauksissa jopa minuutteja (Pavlo & Aslett, 2016, s. 46). Aikaisemmin TKHJ:iä

suunniteltiin yleisluonteisella tavalla, jolloin tiettyihin käyttötarkoituksiin erikoistuneet järjestelmät olivat harvinaisempia. Sittemmin TKHJ:ien optimointi on usein kohdennettu palvelemaan joko OLAP- tai OLTP-työkuormaa. Vogtin ym. (2017, s. 2490) mukaan tämä jaottelu on kuitenkin ongelmallinen, koska kaikkia kyselyitä ei pysty jakamaan näihin kahteen kategoriaan. TKHJ:n optimointi ainoastaan joko OLAP- tai OLTP-työkuormia varten ei välttämättä palvele näitä väliinputoavia kyselyitä (Vogt ym., 2017, s. 2490). *HTAP* on sovellusarkkitehtuuri, jonka avulla OLAP- ja OLTP-työkuormien yhtaikainen käsittely on mahdollista. HTAP-sovellusarkkitehtuuria on käsitelty tässä tutkielmassa ensimmäisen kerran luvussa 3.3.1.

Perinteiset relaatiotietokannat on optimoitu käsittelemään kyselyitä, jotka koskevat vain pientä osaa tietokannasta, ja transaktioita, jotka käsittelevät muutamia tietueita per relaatio lisäyksiä tai päivityksiä varten (Elmasri & Navathe, 2016, s. 1102-1103). Relaatiomalli tukee siis OLTP-työkuorman käsittelyn lisäksi myös OLAP-tyypin kyselyitä pienemmässä mittakaavassa, mutta Elmasrin ja Navathen (2016, s. 1103) mukaan RTKHJ:iä ei voi optimoida OLAP-työkuormaa varten. Myös Taipaluksen (2024, s. 2) mukaan suurissa tietokannoissa tehtävät monimutkaiset kyselyt voivat olla haastavia, sillä relationaalisissa tietokannoissa tarvittavan tallennustilan määrä pyritään minimoimaan, mikä yleensä aiheuttaa kyselyiden hidastumista taulujen välisten liitoksien suuren määrän vuoksi. Valduriezin ym. (2021, s. 6) mukaan RTKHJ:t ovat suhteellisen tehottomia monimutkaisen datan käsittelyssä relationaalisen tietomallin jäykkyydestä johtuen. Myös graafimallisen datan käsittely on haastavaa, vaikka se voidaankin mallintaa relaatioiksi (Valduriez ym., 2021, s. 6). RTKHJ:issä graafien läpikäyntejä aiheuttavat kyselyt voivat viedä todella paljon aikaa, varsinkin jos läpikäytävä graafi on suuri (Valduriez ym., 2021, s. 6). Gyorodin ym. (2015, s. 1) mukaan relationaalinen tietokanta on tehokas, jos tietokannan koko on rajallinen.

NoSQL-järjestelmät ovat alunperin optimoitu OLAP-työkuormaa varten. OTLP-työkuormaa varten NoSQL-järjestelmistä ainakin MongoDB tarjoaa tuen ACID-transaktioille, mutta NoSQL:n prioriteettina on optimointi OLAP-työkuormaa varten. NoSQL on todella joustava talletettavan datan muodon suhteen, minkä lisäksi moni NoSQL-tietokannanhallintajärjestelmä on optimoitu graafimallisen datan käsittelemiseen. Khanin ym. (2023, s. 14) toteuttama relaatiomallin ja NoSQL:n tietokantateknologiaa vertaileva kirjallisuuskatsaus puoltaa NoSQL-tietokantojen käyttöä, kun kyseessä on verkkosovellus käsiteltävän datan määrän ollessa valtava.

NewSQL-järjestelmät ovat Pavlon ja Aslettin (2016, s. 46) mukaan suunnattu sovelluksille, joiden työkuorma on pääasiassa read-write –transaktioita, jotka (1) ovat lyhytkestoisia, (2) käsittelevät vain pientä dataosajoukkoa indeksihakuja (eng. index lookup) käyttäen ja (3) toistuvat useasti. NewSQL:n ja perinteisen relaatiomallin kohdistamilla sovelluksilla voidaan siis todeta olevan samankaltaisuuksia. OTLP-työkuorman lisäksi NewSQL-järjestelmät tukevat HTAP-sovellusarkkitehtuuria (Valduriez ym., 2021; Pavlo & Aslett, 2016). SAP

HANA ja MemSQL olivat ensimmäiset NewSQL-TKHJ:t, jotka markkinoivat itseään HTAP-järjestelminä (Pavlo & Aslett, 2016, s. 52). Pavlon ja Aslettin (2016, s. 54) näkemyksen mukaan suuret OLAP-tietokannat jäänevät historiaan HTAP-tietokantojen nousun myötä.

4.1.3 Suunnittelu, käyttöönotto ja ylläpito

Tietokantojen loogisen rakenteen suunnittelu, käyttöönotto ja ylläpito ovat huomioonotettavia asioita tietokantamallia ja TKHJ:ää valittaessa. Tietokannan loogisen rakenteen suunnittelu *Skeemakehitys* (eng. schema evolution) on klassinen haaste tietokantatutkimuksessa (Chillón ym., 2024, s. 2774). Tietokantojen elinkaaren aikana tietokantojen skeemoja täytyy muokata esimerkiksi silloin, kun uusia tietokantaan kohdistuvia vaatimuksia ilmenee, jolloin tietokantasovellusten koodi ja tietokantaan tallennettu data täytyy päivittää mukautumaan uuteen skeemaan (Chillón ym., 2024, s. 2774). Chillónin ym. (2024, s. 2774) mukaan on tavoiteltavaa, että skeemamuutoksissa datan ja koodin *rinnakkaiskehitys* (eng. co-evolution) on automatisoitua data- ja sovellusvirheiden sekä ylimääräisen vaivan välttämiseksi. Brahmian ym. (2024, s. 39) toteuttaman kirjallisuuskatsauksen mukaan olemassa olevat TKHJ-tuotteet eivät tarjoa skeemakehityksen käsittelyyn tehokasta tukea, minkä vuoksi skeemamuutoksen onnistunut toteuttaminen vaatii paljon tietokantojen ylläpitäjiltä ja sovelluskehittäjiltä.

Relaatiomallisen tietokannan skeemalle on tarkat vaatimukset, mikä rajoittaa valinnanvaraa skeemaa suunniteltaessa. Relationaalinen tietomalli on monimutkainen, mutta suunnittelua tukee mm. normalisointiprosessi ja useat oppaat. Skeemakehitystä varten relaatiotietokannoille on olemassa rinnakkaiskehityksen automatisoinnin tueksi useita julkaisuja ja työkaluja (Chillón ym., 2024, s. 2774), mutta toisaalta Davoudianin ym. (2018, s. 3) mukaan relaatiotietokannan kehittäminen on kallista johtuen relationaalisen datan muuntamisen monimutkaisuudesta. Khanin ym. (2023, s. 4) mukaan skeeman monimutkaisuus saattaa aiheuttaa ongelmia myös tietokannasta toiseen siirtymisessä, jolloin ongelmat voi ilmetä esimerkiksi ristiriitaisena tai toistuvana datana. Relaatiotietokantojen suuremmaksi skaalaaminen on sekä kallista että väliaikaista saatavuuden puutetta aiheuttavaa, sillä se vaatii tietokantojen siirtämistä erillisille palvelimille, joiden laitteisto on tehokkaampaa (Davoudian ym., 2018, s. 3).

NoSQL on tietokannan muokkaamisen ja kehittämisen kannalta ideaali, sillä sen profiloivia ominaisuuksia ovat ketteryys ja joustavuus. NoSQL-tietokannan suunnittelu on paljon vapaamuotoisempaa kuin relationaalisen tietokannan suunnittelu. Suunnittelun vapaamuotoisuus tekee tietokannan suunnittelusta tietyllä tavalla helpompaa ja vähemmän rajoittavaa, mutta samalla tietokanta voi altistua uhkille, jotka voidaan automaattisesti välttää relationaalisen tietokannan suunnittelun ohjeita noudattamalla. NoSQL-tietokantaa suunnitellessa esimerkiksi tiedon toistumisesta huolehtiminen jää suunnittelijan omalle vastuulle. Tiedon tarpeettoman toistumisen salliminen voi olla tietoinen päätös tietokannassa usein tehtävien operaatioiden nopeuden parantamiseksi. NoSQL-tietokantojen

monimuotoisuuden vuoksi lähestymistapojen ja työkalujen tarjoaminen NoSQL-tietokantojen skeemakehityksen tueksi on haastavampaa kuin relaatiotietokannoille (Chillón ym., 2024, s. 2774). Skeemakehityksen huomioonottaminen on tärkeää valitusta NoSQL-tietokannasta riippumatta, koska Chillónin ym. (2024, s. 2774) mukaan skeemamuutoksia tapahtuu myös ns. skeemattomille NoSQL-tietokannoille, sillä niiden skeema määrittyy epäsuorasti tietokannan datan ja tietokantasovellusten koodin perusteella. NoSQL-tietokannat tukevat pystysuuntaista skaalaamista ja pystyvät käsittelemään suuria määriä rakennetonta dataa tehokkaasti (Khan ym., 2023, s. 1).

NewSQL-tietokannat noudattavat relationaalista tietomallia, joten NewSQL-tietokannan loogisen rakenteen suunnittelu on hyvin samankaltaista kuin relationaalisen tietokannan suunnittelu. Relationaalisen tietomallin noudattaminen helpottaa myös skeemakehitystä huomattavasti, koska relaatiomallin skeemakehityksen tueksi on jo olemassa huomattava määrä työkaluja ja tehtyjä tutkimuksia. NewSQL-tietokannat pyrkivät NoSQL-tietokantojen skaalautuvuuteen Read- ja Write-operaatioissa (Pavlo & Aslett, 2016, s. 46).

4.1.4 Riskit ja luotettavuus

Tietokannan käyttötarkoituksesta riippuen tietokannalta vaadittu luotettavuuden ja eheyden taso on vaihtelevaa. Esimerkiksi Pavlon ja Aslettin (2016, s. 46) mukaan on yleistä, että kriittistä dataa käsittelevät yritysjärjestelmät, kuten taloudelliset ja tilaustenkäsittelyjärjestelmät, eivät voi joustaa ACID-periaatteen tuomasta luotettavuudesta. Kriittisellä datalla tarkoitetaan dataa, jota pidetään keskeisenä organisaation menestyksen kannalta, tai dataa, jota täytyy säilyttää säännösten vuoksi.

Relationaalisille tietokannanhallintajärjestelmille ACID:in tukeminen on itsestäänselvyys. Relaatiomallin pitkä historia ja vakiintunut asema markkinoilla tuo myös turvaa osaavien työntekijöiden ja tuen löytymiseen. NewSQL-tietokannat säilyttävät relaatiomallin tapaan ACID-transaktiot ja niiden tuoman luotettavuuden.

NoSQL lähtökohtaisesti tukee BASE[4]-periaatetta ACID:in sijaan, mutta jotkin NoSQL-TKHJ:t, kuten MongoDB ja Redis, tarjoavat tuen ACID-transaktioille. ACID-periaatteen noudattamisella NoSQL-tietokannoissa on kuitenkin kääntöpuolensa, sillä se tarkoittaa Taipaluksen (2024, s. 4-5) mukaan suorituskyvyn heikkenemistä, jolloin NoSQL:n tarjoamat edut eivät ole niin huomattavia. Pritchettin (2008, s. 51) mukaan BASE on täysi vastakohta ACID:lle. Pritchett (2008, s. 51) näkee ACID:lle ominaisen jokaisen operaation jälkeisen eheyden varmistamisen pessimistisyytenä, kun taas BASE on "optimistinen" salliessaan vaihtelevan eheyden tietokannassa. BASE:n noudattaminen mahdollistaa skaalauksen laajemmalla tasolla, kuin minkä ACID pystyy saavuttamaan. BASE:n korkea saatavuus mahdollistetaan osittaisia järjestelmäkatkoksia (eng. partial failure)

[4] Basically Available, Soft state and Eventually consistent

tukemalla niin, että järjestelmän täysi kaatuminen kuitenkin estetään. Yksinkertaisena esimerkkinä Pritchett (2008, s. 51) esittää tilanteen, jossa tietokannan käyttäjät ovat jakautuneet viidelle eri tietokantapalvelimelle. BASE-periaatteen mukaisesti toteutetussa järjestelmässä tietokannassa tapahtuva häiriö vaikuttaisi vain siihen 20 prosenttiin käyttäjistä, jotka käyttävät kyseistä häiriönalaista palvelinta. Tämä kokonaisuudessaan johtaa järjestelmän korkeampaan saatavuuteen.

NoSQL-tietokannoissa arkaluonteisen datan salaaminen pilvipalveluntarjoajilta voi olla haastavaa NoSQL-tietokantojen monimuotoisen luonteen vuoksi. Käyttäjien henkilökohtaisten tietojen suojaaminen on suuri huolenaihe pilvitallennussiirtymissä (eng. cloud storage transfer). Tämänhetkisiä pilvipalveluntarjoajamalleja (eng. cloud service provider designs) kehitettäessä ei ole otettu yhteentoimivuutta (eng. interoperability) ja siirrettävyyttä huomioon, minkä vuoksi yhtenäisen pilvipalvelun kehittäminen ja tarjoaminen NoSQL-tietomalleille on hankalaa. (Khan ym., 2023, s. 3-4)

4.2 Yhteenveto & pohdintaa

Perinteisen relaatiomallin vahvuudet eroavat vaihtoehtoisista tietokantamalleista selkeästi, vaikka NewSQL:llä. NoSQL ja NewSQL ovat relaatiomallia sopivampia monelle nykyaikaiselle käyttötarkoitukselle, sosiaalisen median sovelluksille. Davoudian ym. (2018, s. 4) eivät kuitenkaan näe NoSQL:ää korvaajana relaatiotietokannoille, vaan ennemminkin korjauskeinona niiden haasteisiin massadatan käsittelyn kanssa. Brewerin (2012) näkemys tukee myös tätä, sillä hänen mukaansa NoSQL:n tarkoituksena on nimenomaan jättää tietokannan eheys toissijaiseksi sen saatavuuteen nähden, jolloin ACID-periaatteen noudattaminen ei olisi loogista.

Taipaluksen (2024) kokoamasta datasta tietokannanhallintajärjestelmien suorituskykyä vertailevien tutkimusten tuloksista on huomioitavissa, että suurimpia menestyjiä vertailuissa ovat mm. MySQL, MongoDB, Redis ja Cassandra. Kyseisistä tuotteista MySQL on relationaalinen, ja muut ovat NoSQL-tuotteita. On kuitenkin otettava huomioon, että suosituimmat tietokannanhallintajärjestelmät todennäköisesti esiintyvät tutkimuksissa useammin, jolloin niiden menestys on todennäköisempää. Esimerkiksi NewSQL-tietokannanhallintajärjestelmistä VoltDB on päihittänyt suorituskykyvertailuissa sekä MongoDB:n että MySQL:n, mutta se on ollut osana huomattavasti harvempaa tutkimusta, jolloin sillä ei ole ollut mahdollisuutta olla yhtä suuri “menestyjä”, kuin suosituimmilla järjestelmillä. Tietokantamallia ja TKHJ:ä valittaessa tietokannanhallintajärjestelmien suorituskykyvertailuihin ei kannata luottaa automaattisesti, sillä Taipaluksen (2024, s. 4) mukaan ne eivät välttämättä heijasta oikean elämän käyttötilanteita, riippuen vertailussa käytetyistä metodeista.

Pavlo ja Aslett (2016, s. 54) odottavat, että tulevaisuudessa kaikki merkittävimmät TKHJ:t tulevat tukemaan relaatiomallia ja SQL:ää jossakin muodossa,

kuin myös OLAP- ja OLTP-työkuormaa HTAP-tietokantojen tapaan, minkä tapahduttua nykyisenkaltainen TKHJ:ien lokeroiminen on merkityksetöntä. TKHJ:ien lokerointi eri luokkiin voi jo nyt olla haastavaa, sillä Taipaluksen (2024, s. 6) mukaan niiden määritelmät ovat muuttuvia ja tarkkuudeltaan puutteellisia. Taipalus (2024) luokitteli kirjallisuuskatsauksessaan viisi TKHJ:ä "Muu" -kategoriaan. Suurimpia TKHJ-tuotteita, kuten Oraclea, tarkasteltaessa onkin helppo huomata, että TKHJ:t voivat jo nyt tarjota paljonkin omasta tietokantamallistaan poikkeavia ominaisuuksia. Tietokantamallien rajojen hämärtyminen vaikeuttaa tietokantamallien suosion vertailua, minkä lisäksi tietokantamallien keskeisten ominaisuuksien vertailu ei välttämättä ole niin merkityksellistä markkinoilla olevia TKHJ-tuotteita ajatellen.

5 YHTEENVETO

Relaatiomalli on ollut tietokannanhallintajärjestelmien suosituin tietokantamalli vuosikymmenten ajan. Relaatiomallin keskeisimpiä ominaisuuksia ovat relationaalinen tietomalli sekä tietokannan ja sen sisältämän datan eheyden varmistaminen mm. normalisointiprosessin ja ACID-periaatteen noudattamisen avulla.

Tutkielmassa tarkasteltiin kahta vaihtoehtoista tietokantamallia relaatiomallille, NoSQL:ää ja NewSQL:ää. NoSQL:n keskeisiä ominaisuuksia ovat joustava tietokantasuunnittelu, korkea skaalautuvuus ja skeemattomat tietomallit. NoSQL:n joustavuus ja monimuotoisuus mahdollistaa tietokantojen optimoimisen myös silloin, kun tietokannan vaatimukset ovat tarkat. NoSQL-tietokantojen tehokkuus ja joustavuus toteutetaan eheyden ja luotettavuuden kustannuksella. NoSQL-tuotteita on tarjolla kattavasti, mutta omalle käyttötarpeelle TKHJ-valintaa tehdessä täytyy omata seikkaperäinen ymmärrys eri järjestelmien ominaisuuksista ja haasteista. NewSQL-järjestelmät pyrkivät yhdistämään relationaalisten järjestelmien ja NoSQL-järjestelmien parhaita puolia. NewSQL tarjoaa NoSQL:n skaalautuvuuden ja relaatiomallin ACID-periaatteen OLTP-työkuorman transaktioissa. Tutkielman tuloksista huomataan NewSQL:n ja NoSQL:n sopivan relaatiotietokantoja paremmin nykyaikaisiin massadatan tuomiin vaatimuksiin, mutta relaatiotietokannoilla on edelleen omat vahvuutensa, joiden korvaaminen on haastavaa varsinkin NoSQL-tietokannoille.

Tutkielman tulosten luotettavuutta on pyritty edistämään seuraavasti: tutkielma toteutettiin kirjallisuuskatsauksena, jonka lähteet on tarkoin valittu Julkaisufoorumin portaalia hyödyntäen. Ne tässä tutkielmassa käytetyt lähteet, joille ei ole saatavilla JUFO-arvosanaa, ovat joko työelämän puolella arvostettuja merkittäviä toimijoita tai yksittäisiä julkaisuja, joiden viittausmäärät ovat suuria. Jälkimmäiseen kategoriaan sopivia lähteitä ovat esimerkiksi Brewer (2000) ja Elmasri & Navathe (2016). Tutkielma on uudelleentoistettava, sillä eri tietokantamallien keskeisimmät piirteet eivät ole nopeasti muuttuvia. Tässä tutkielmassa käytettyjen lähdeviittausten tarkkuus helpottaa lähdeviittausten paikkansapitävyyden tarkistamista. Samasta aiheesta tehty tutkielma voi kuitenkin olla sisällöltään erilainen valittujen tietokantamallien suhteen etenkin silloin, jos tietokantoihin kohdistuvat tarpeet muuttuvat merkittävästi (vrt. massadata ja web-sovellukset).

Tutkielmaa tehdessä ilmeni joitakin rajoituksia. Tietokantamallien vertailuun ei löytynyt yksiselitteistä listaa vertailtavista ominaisuuksista, joten vertailuun käytetyt näkökulmat tuli valita itse. Tutkielmassa käytetty vertailukriteerien lista on alustava. Sen lisäksi monet käytetyistä tuoreemmista lähteistä tarkastelevat tietokantoja uusien vaihtoehtojen näkökulmasta, jolloin relaatiomallista muodostunut kuva saattaa olla negatiivisesti vinoutunut. Relaatiomallin nykytilan vahvuuksia ja heikkouksia käsittelevää kirjallisuutta oli haastavaa löytää, minkä lisäksi tietokantamallien välisten rajojen hämärtyminen vaikeutti työtä. Tutkielman alussa tehdyt rajaukset olisivat voineet myös olla selkeämpiä ja tiukempia.

Jatkotutkimuksissa voisi tarkastella vaihtoehtoisia kriteeristöjä tietokantamallien ja ennen kaikkea tietokannanhallintajärjestelmien vertailuun. Tämän lisäksi eri NoSQL-tietomalleja yhdistelevän pilvipalvelun kehittämisen edellytyksien ja mahdollisuuksien tarkastelulla olisi potentiaalia.

LÄHTEET

Anjomshoaa, A., & Tjoa, A. M. (2011). How the cloud computing paradigm could shape the future of enterprise information processing. *Proceedings of the 13th International Conference on Information Integration and Web-Based Applications and Services*, 7–10. https://doi.org/10.1145/2095536.2095539

Brewer, D. E. A. (2000). *Towards Robust Distributed Systems*.

Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. *Computer*, *45*(2), 23–29. https://doi.org/10.1109/MC.2012.37

Chamberlin, D. D., & Boyce, R. F. (1976). SEQUEL: A structured English query language. *Proceedings of the 1976 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control - FIDET '76*, 249–264. https://doi.org/10.1145/800296.811515

Chillón, A. H., Klettke, M., Ruiz, D. S., & Molina, J. G. (2024). A Generic Schema Evolution Approach for NoSQL and Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, *36*(7), 2774–2789. IEEE Transactions on Knowledge and Data Engineering. https://doi.org/10.1109/TKDE.2024.3362273

Codd, E. F. (1970). *A Relational Model of Data for Large Shared Data Banks*. *13*(6).

Codd, E. F. (1985a). Is your DBMS really relational. *ComputerWorld*, *19*(41), 1-2.

Codd, E. F. (1985b). Does your DBMS run by the rules. *ComputerWorld*, *21*.

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., … Woodford, D. (2013). Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems*, *31*(3), 1–22. https://doi.org/10.1145/2491245

Davoudian, A., Chen, L., & Liu, M. (2018). A Survey on NoSQL Stores. *ACM Computing Surveys*, *51*(2), 40:1-40:43. https://doi.org/10.1145/3158661

DB-Engines. *DB-Engines Ranking*. Noudettu 8. marraskuuta 2024, osoitteesta https://db-engines.com/en/ranking

DB-Engines. *Navigational DBMS - DB-Engines Encyclopedia*. Noudettu 14. huhtikuuta 2024, osoitteesta https://db-engines.com/en/article/Navigational+DBMS

Dieu, G., Lerat, J.-S., & Cremer, S. (2023). Relational Wide Column Store: A new Big Data paradigm. *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 211–212. https://doi.org/10.1145/3587135.3592183

Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems* (Seventh edition). Pearson.

Gartner. *Definition of DBMS (Database Management System) – Gartner Information Technology Glossary*. Noudettu 10. marraskuuta 2024, osoitteesta

https://www.gartner.com/en/information-technology/glossary/dbms-database-management-system

Gartner. (2023, 17. toukokuuta) *Market Share: Database Management Systems, Worldwide, 2022*. https://www.gartner.com/en/documents/4366299

Gyorodi, C., Gyorodi, R., Pecherle, G., & Olah, A. (2015). A comparative study: MongoDB vs. MySQL. *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, 1–6. https://doi.org/10.1109/EMES.2015.7158433

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

Haigh, T. (2016). How Charles Bachman invented the DBMS, a foundation of our digital world. *Communications of the ACM*, *59*(7), 25–30. https://doi.org/10.1145/2935880

Hosting Data. (2023, 17. marraskuuta) *NoSQL Databases List by Hosting Data – Updated 2023*. https://hostingdata.co.uk/nosql-database/

Kelechava, B. (2018, lokakuuta 5). The SQL Standard—ISO/IEC 9075:2023 (ANSI X3.135)—ANSI Blog. *The ANSI Blog*. https://blog.ansi.org/sql-standard-iso-iec-9075-2023-ansi-x3-135/

Khan, W., Kumar, T., Zhang, C., Raj, K., Roy, A. M., & Luo, B. (2023). SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review. *Big Data and Cognitive Computing*, *7*(2), Article 2. https://doi.org/10.3390/bdcc7020097

Liu, Z. H., Hammerschmidt, B., & McMahon, D. (2014). JSON data management: Supporting schema-less development in RDBMS. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 1247–1258. https://doi.org/10.1145/2588555.2595628

Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? *Computer*, *43*(2), 12–14. Computer. https://doi.org/10.1109/MC.2010.58

Li, G., & Zhang, C. (2022). HTAP Databases: What is New and What is Next. *Proceedings of the 2022 International Conference on Management of Data*, 2483–2488. https://doi.org/10.1145/3514221.3522565

Madden, S. (2012). From Databases to Big Data. *IEEE Internet Computing*, *16*(3), 4–6. https://doi.org/10.1109/MIC.2012.50

Maier, D., Stein, J., Otis, A., & Purdy, A. (1986). Development of an object-oriented DBMS. *ACM SIGPLAN Notices*, *21*(11), 472–482. https://doi.org/10.1145/960112.28746

MongoDB. *In-Memory Databases Explained.* Noudettu 10. marraskuuta 2024, osoitteesta https://www.mongodb.com/resources/basics/databases/in-memory-database

Pavlo, A., & Aslett, M. (2016). What's Really New with NewSQL? *ACM SIGMOD Record*, *45*(2), 45–55. https://doi.org/10.1145/3003665.3003674

Philip Chen, C. L., & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, *275*, 314–347. https://doi.org/10.1016/j.ins.2014.01.015

Pritchett, D. (2008). BASE: An Acid Alternative: In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability. *Queue*, *6*(3), 48–55. https://doi.org/10.1145/1394127.1394128

Sagiroglu, S., & Sinanc, D. (2013). Big data: A review. *2013 International Conference on Collaboration Technologies and Systems (CTS)*, 42–47. https://doi.org/10.1109/CTS.2013.6567202

Salminen, A. (2011). *Mikä kirjallisuuskatsaus? : Johdatus kirjallisuuskatsauksen tyyppeihin ja hallintotieteellisiin sovelluksiin*. Vaasan yliopisto. https://osuva.uwasa.fi/handle/10024/7961

Silberschatz, A., Korth, H. F., & Sudarshan, S. (1996). Data models. *ACM Computing Surveys*, *28*(1), 105–108. https://doi.org/10.1145/234313.234360

Taipalus, T. (2024). Database management system performance comparisons: A systematic literature review. *Journal of Systems and Software*, *208*, 111872. https://doi.org/10.1016/j.jss.2023.111872

Tsichritzis, D. C., & Lochovsky, F. H. (1976). Hierarchical Data-Base Management: A Survey. *ACM Computing Surveys*, *8*(1), 105–123. https://doi.org/10.1145/356662.356667

Valduriez, P., Jimenez-Peris, R., & Özsu, M. T. (2021). Distributed Database Systems: The Case for NewSQL. Teoksessa A. Hameurlain & A. M. Tjoa (Toim.), *Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVIII: Special Issue In Memory of Univ. Prof. Dr. Roland Wagner* (ss. 1–15). Springer. https://doi.org/10.1007/978-3-662-63519-3_1

Vogt, M., Stiemer, A., & Schuldt, H. (2017). Icarus: Towards a multistore database system. *2017 IEEE International Conference on Big Data (Big Data)*, 2490–2499. https://doi.org/10.1109/BigData.2017.8258207

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

Wegner, P. (1990). Concepts and paradigms of object-oriented programming. *ACM SIGPLAN OOPS Messenger*, *1*(1), 7–87. https://doi.org/10.1145/382192.383004

Yaqoob, I., Hashem, I. A. T., Gani, A., Mokhtar, S., Ahmed, E., Anuar, N. B., & Vasilakos, A. V. (2016). Big data: From beginning to future. *International Journal of Information Management*, *36*(6, Part B), 1231–1247. https://doi.org/10.1016/j.ijinfomgt.2016.07.009