Atte Päärni

TUOTEKATALOGIN TIEDONHALLINTA PILVIYMPÄRISTÖSSÄ: AWS AURORA JA DYNAMODB TIETOKANTAPALVELUIDEN SUORITUSKYKYVERTAILU

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2019

TIIVISTELMÄ

Päärni, Atte
Tuotekatalogin tiedonhallinta pilviympäristössä: AWS Aurora ja DynamoDB tietokantapalveluiden suorituskykyvertailu
Jyväskylä: Jyväskylän yliopisto, 2019, 77 s.
Tietojärjestelmätiede, pro gradu-tutkielma
Ohjaajat: Luoma, Eetu, Seppänen, Ville & Taipalus, Toni

Tutkielmassa on vertailtu kahden pilviympäristöön suunnitellun tietokannan hallintajärjestelmän suorituskykyä tuotekatalogin tietokantana pilviympäristössä. Tuotekatalogin tiedon saatavuuden suorituskyky on merkittävä tekijä menestyksekkääseen kaupankäyntiin verkossa. Pilvilaskenta toi mullistuksen verkkosovellusten suorituskykyyn tarjoamalla illuusion laskentaresurssien ehtymättömyydestä. Perinteiset tietokannan hallintajärjestelmät soveltuvat huonosti pilviympäristöön. Vastaamaan tähän haasteeseen syntyi NoSQL-tietokantakonsepti, johon kuuluu joukko pilviympäristöön kehitettyjä tietokannan hallintajärjestelmiä, joita yhdistää relaationaalisesta tietomallista luopuminen. Relationaalisesta tietomallista luopuminen mahdollisti skaalautumisen pilviympäristössä, mutta samalla luovuttiin vahvoista tapahtumanhallintaominaisuuksista, jotka ovat pakollisia monien yritysten liiketoiminnassa. NewSQL-tietokantakonsepti syntyi vastaamaan näihin haasteisiin uudella tietokannan hallintajärjestelmäarkkitehtuurilla, joka pyrkii tarjoamalla relationaalista tietomallia ja tapahtumanhallintaominaisuuksia pilviympäristössä. Tutkielmassa on selvitetty, voidaanko näillä moderneilla tietokannan hallintajärjestelmillä toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta, ja kumman pilvipalveluna tarjottavan tietokantapalvelun suorituskyky on parempi tuotekatalogin tietokantana: AWS Aurora- vai DynamoDB-tietokantapalvelun. Tutkimus toteutettiin suunnittelutieteellisenä konstruktiivisena tutkimuksena. Kahta tuotekatalogin tietokantatoteutusta evaluoitiin ja vertailtiin keskenään verkkosovellusten suorituskykytestauksen periaatteiden mukaisesti. Molemmat tuotekatalogitoteutukset skaalautuvat pilviympäristössä kuormituksen kasvaessa, mutta kumpikaan tietokanta ei yksinään pysty täyttämään kaikkia tuotekatalogiin kohdistuvia tiedon saannin tarpeita. Molemmat tarkastellut tietokannat soveltuvat huonosti tiedon hakemiseen vapaalla hakusanalla, missä hakusanaa pitäisi verrata useaan kenttään. Suorituskykyisempi tietokantapalveluista oli Aurora, joka oli DynamoDB-tietokantapalveluun toteutettua tuotekatalogia suorituskykyisempi, kun tarkasteltiin tiedon saatavuutta.

Asiasanat: pilvilaskenta, tietokannan hallintajärjestelmä, NoSQL, NewSQL, CAP-teoreema, ACID-ominaisuudet, BASE-ominaisuudet

ABSTRACT

Päärni, Atte
Product catalog data management in cloud environment: Performance evaluation of AWS Aurora and DynamoDB database services
Jyväskylä: University of Jyväskylä, 2019, 77 p.
Information Systems Science, Master's Thesis
Supervisors: Luoma, Eetu, Seppänen, Ville & Taipalus, Toni

The performance of two database management systems designed to cloud environment is evaluated in this study from the view of product catalog data management. Data availability of product catalog is significant factor in successful e-commerce. Cloud computing revolutionized the performance of web applications by providing illusion of endless computing resources. Traditional database management systems scale poorly in cloud environment. To answer this challenge numerous new database management systems designed for cloud environment emerged that belonged to new NoSQL-database concept. These new database management systems abandoned relational data model, which enabled them to scale in cloud environment, but they also lacked transaction management properties that are required in many businesses. NewSQL-database concept aims to answer these challenges with new database management systems architecture that provides relational data model and transaction management that scales in cloud environment. The aim of this research is to study if a product catalog database that scales in cloud environment can be developed with these modern database management systems. Furthermore, which of these cloud database services perform better as product catalog database: AWS Aurora or DynamoDB-database service. The research was conducted as design science study. Two designed product catalog implementations were evaluated by web application performance testing practices. Both product catalog database implementations did scale in cloud environment, but neither could fulfil all data availability requirements of a product catalog database. Both databases performed poorly when products were queried with search term. Aurora-database service had superior data availability performance compared to DynamoDB-database service as product catalog database.

Keywords: product catalog, cloud computing, database management system, NoSQL, NewSQL, CAP theorem, BASE properties, ACID properties

ESIPUHE

Haluan kiittää yliopistonlehtori Eetu Luomaa kärsivällisyydestä, ymmärryksestä, tuesta ja ohjauksesta.

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Internetin yleistyessä kasvoi vaihdannan tarve Internetin välityksellä (Hirai, Kimura, Fukushima, Kihara, Nakamura, & Nagata, 2001). Tämä johti verkkokaupankäynnin syntymiseen, mitä voidaan pitää yhtenä merkittävimmistä Internetin sovellusten aiheista (Hirai ym., 2001). Suorituskyky ja skaalautuvuus ovat tärkeitä tekijöitä menestyksekkääseen kaupankäyntiin verkossa (Bochamann ym., 2002). Verkkokauppasovelluksen suorituskyvyn kannalta oleellinen tekijä on katalogitiedon saatavuus, koska 80-prosenttia verkkokauppasovelluksen pyynnöistä liittyy tuotteiden hakuun (Bochamann ym., 2002).

Pilvimaailma luo uusia haasteita ohjelmistoille. Toimiakseen nopeasti ja kustannustehokkaasti niiden pitää pystyä skaalautumaan nopeasti käyttöasteen kasvaessa ja laskiessa (Armbrust ym., 2009). Tämä skaalautuvuus alaspäin on uusi haaste myös tietokannan hallintajärjestelmille (Feuerlicht, 2010). Tietokannan hallintajärjestelmän pitää pystyä tarvittaessa skaalautumaan horisontaalisesti tuhansille eri palvelimille ja samalla pitämään huolta tiedon saatavuudesta ja yhteneväisyydestä (Feuerlicht & Pokorný, 2011). Vastaamaan näihin haasteisiin on kehitetty useita kaupallisia ja avoimeen lähdekoodiin perustuvia NoSQL–tietokantatuotteita (Feuerlicht & Pokorný, 2011).

NoSQL on sateenvarjo, jonka alle kuuluu useita tietokantatuotteita (Hecht & Jablonski, 2011). NoSQL - tietokannat eivät ole relaatioihin perustuvia, eivätkä käytä SQL-kieltä tiedon käsittelemiseen (Feuerlicht & Pokorný, 2011). Tästä johtuen NoSQL–tietokannoista puuttuu perinteisten relaatiotietokannan hallintajärjestelmien tapahtumanhallintaominaisuuksia (Feuerlicht & Pokorný, 2011). Tätä voidaan pitää vaihtoehtoiskustannuksena tiedon paremmalle saatavuudelle (Feuerlicht & Pokorný, 2011).

Monille Internet-sovelluksille, kuten pankki- ja finanssialan sovelluksille, takeet tiedon eheydestä ovat pakollisia tietokannan hallintajärjestelmän ominaisuuksia (Feuerlicht & Pokorný, 2011). Tästä syystä on tarve tietokannan hallintajärjestelmälle, joka yhdistäisi NoSQL–tietokantojen tiedon saatavuuden ja perinteisten relaatiotietokannan hallintajärjestelmien tapahtumanhallintaominaisuudet (Feuerlicht & Pokorný, 2011). Vastaamaan näihin haasteisiin on kehitetty

NewSQL-tietokantakonsepti, joka pyrkii tarjoaman skaalautuvuutta pilviympäristössä sekä relationaalisen tietokannan mahdollistavia tapahtumanhallintaominaisuuksia (Bernstein, 2014).

NoSQL-tietokantojen suorituskykyä on vaihtelevin tuloksin verrattu perinteisiin relationaalisin tietokantoihin (Boicea, Radulescu & Agapin, 2012, Florato, Teletia, DeWitt, Patel & Zhang, 2012, Li & Manoharan, 2013, Abotourabi, Rezapour, Moradi & Ghadiri, 2015). Tässä tutkielmassa ei verrataan NoSQL-tietokannan suorituskykyä perinteiseen relationaaliseen tietokantaan vaan modernin NewSQL-tietokannan suorituskykyyn. Aiemmat tutkimukset on tyypillisesti tehty myös joko yksittäisellä koneella tai suljetussa klusteroidussa ympäristössä (Floratou ym., 2012, Abotourabi ym., 2015). Tässä tutkielmassa tietokantojen suorituskykyä tutkitaan todellisessa pilviympäristössä, johon testattavat tietokannan hallintajärjestelmät ovat suunniteltu. NoSQL-tietokantakonseptin alle kuuluu erityyppisiä tietokannan hallintajärjestelmiä ja tarkoituksenmukainen tietokanta valitaan sovelluksen tarpeiden mukaan (Hecht & Jablonski, 2011). Tutkimuksen tiedonhallinnan näkökulmaksi on valittu tuotekatalogin tiedonhallinta, koska se on merkittävä tekijä verkkokauppasovelluksen suorituskykyyn ja näin oleellinen vaikuttaja menestyksekkääseen verkkokaupankäyntiin (Bochamann ym., 2002).

Tämän tutkielman tarkoituksena on selvittää tuotekatalogin tiedonhallinnan vaatimuksia ja selvittää, millaisia pilvisovelluksille soveltuvia tietokantahallintajärjestelmiä on kehitetty, jotka voisivat vastata tuotekatalogin tiedonhallinnan vaatimuksiin. Tämän jälkeen vertaillaan kahta pilvipalveluna tarjottavan tietokannan suorituskykyä tuotekatalogin tietokannan hallintajärjestelmänä. Tämä tutkielman tutkimuskysymykset voidaan esittää seuraavasti:

- Miten voidaan toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta?
- Miten Amazon Aurora-tietokantapalvelu suorituskyky vertautuu Amazon DynamoDB-tietokantapalvelun suorituskykyyn tuotekatalogin tietokantana?

Ensimmäisen kysymyksen tutkimiseksi on selvitettävä mitä vaatimuksia tuotekatalogin tiedonhallinta asettaa ja mitä pilviympäristössä skaalautuvia tietokannan hallintajärjestelmiä on olemassa. Lisäksi on selvitettävä mitä verkkosovellusten suorituskykytestauksen periaatteita ja ohjeistuksia on olemassa, jotta voidaan evaluoida tutkimuksessa kehitettyä artefaktia, eli pilviympäristössä skaalautuvaa tuotekatalogin tietokantaa. Näitä kysymyksiä tutkitaan kirjallisuuskatsauksena. Kirjallisuuskatsauksen aineistoa on haettu ACM:n (Association for Computing Machinery) ja IEEE:n (Institute for Electrical and Eletronics Engineeriing) digitaalisista kirjastoista muun muassa seuraavilla hakusanoilla: *ecommerce, product catalog, cloud computing, database, NoSQL, NewSQL, performance testing*.

Kirjallisuuskatsauksen löydösten pohjalta tehdään konstruktiivinen tutkimus, jolla pyritään vastaamaan ensimmäiseen tutkimuskysymykseen. Koska NoSQL-tietokannat tarjoavat vaihtoehtoisia tapoja tiedon mallintamiseen ja NewSQL-tietokannat tarjoavat perinteistä relationaalista tietomallia pilviympä-

ristössä, on olemassa vaihtoehtoisia tapoja toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta. Tästä syystä konstruktiivisessa tutkimuksessa kehitetään kaksi vertailtavaa toteutusta pilviympäristössä skaalautuvasta tuotekatalogin tietokannasta ja toisen tutkimuskysymyksen tarkoituksena on vertailla kahta vaihtoehtoista tapaa toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta. Molemmat tietokannan hallintajärjestelmät ovat Amazon Web Services-pilvipalvelutarjoajan tietokantapalveluja. AWS Aurora-tietokantapalvelu valittiin edustamaan NewSQL-tietokantakonseptin mukaista relationaalista tietokantaa, ja AWS DynamoDB-tietokantapalvelu valittiin edustamaan NoSQL-tietokantakonseptin mukaista dokumenttivarastoa. Kehitettyjä tuotekatalogin tietokantojen suorituskykyä evaluoidaan verkkosovellusten suorituskykytestauksen periaatteiden avulla.

Tutkielma koostu kahdeksasta luvusta. Luvussa 2 selvitetään mitä erityisvaatimuksia on tuotekatalogin tiedonhallinnalle. Luvussa 3 käsitellään pilvimaailman käsitteitä, pilvilaskennan keskeisiä ominaisuuksia, sekä tietokannan hallintajärjestelmien haasteita pilvimaailmassa. Lisäksi tutustutaan NoSQL-tietokantakonseptiin ja tyypillisiin NoSQL-tietokantojen tietomalleihin. Luvun lopussa esitellään NewSQL-tietokantakonsepti. Luvussa 4 esitetään suorituskykytestauksen periaatteita käsittelemällä suorituskykytestauksen tyypit ja aktiviteetit. Luvussa 5 esitellään tutkimuksessa käytettävää tutkimusmenetelmää, eli suunnittelutiedettä, ja miten sitä on sovellettu tutkimuksessa. Luvussa 6 esitetään konstruktiivisen tutkimuksen pohjalta syntyneet artefaktit, eli AWS Aurora-tietokantapalveluun toteutettu tuotekatalogin tietokanta ja AWS DynamoDB-tietokantapalveluun toteutettu tuotekatalogi tietokanta. Luvussa 7 esitetään evaluointi, eli kuvataan, miten luotuja tuotekatalogeja vertaillaan keskenään suorituskykytestauksen avulla ja esitetään evaluoinnin tulokset. Tutkielman päätteeksi esitetään pohdinta ja yhteenveto.

2 TUOTEKATALOGIN TIEDONHALLINTA

Tässä luvussa esitetään ensimmäisenä tuotekatalogin tiedonhallinnan vaatimuksia. Tämän jälkeen kuvataan entiteetti-attribuutti-arvo -tietomalli, jolla on pyritty vastaamaan näihin vaatimuksiin relaatiotietokannan hallintajärjestelmissä. Tämän jälkeen esitetään perinteisen verkkokauppasovelluksen arkkitehtuuri ja skaalautuvuuden haasteita. Lopuksi esitetään yhteenveto.

2.1 Tuotekatalogin tiedonhallinnan vaatimuksia

Jhingran (2000) näkemyksen mukaan tuotekatalogin tiedonhallinta on tärkein verkkokauppasovelluksia tukeva tietokantatutkimuksen ala. Hän listaa kolme spesifimpää aluetta, jotka hän uskoo olevan erityisen tärkeitä:

- **Katalogien integrointi.** Kaupallisilla toimijoilla voi olla jopa 80 000 tavarantoimittajaa. Kuinka yhdistää ja ylläpitää kunkin tavarantoimittajan katalogit?
- **Vertikaalinen skeema.** Tyypillinen 100 000 varastonimikettä sisältävä tuotekatalogi voi sisältää 10 000 eri attribuuttia ja relationaaliset tietokannat eivät hallinnoi hyvin näin montaa taulua tai yhtä yleistaulua, joka sisältää paljon tyhjiä arvoja. Kuinka tukea 10 000 taulun näkymää sovellukselle, mutta hallinnoida yhtä vertikalisoitua pöytää tietokantatasolla?
- **Haku.** Tietoa haetaan tyypillisesti katalogihierarkiaa selailemalla tai määrätyillä hakukriteereillä. Näistä tiedonhaun näkökulmasta haastavampi on määrätyillä hakukriteereillä haku, erityisesti relaatiotietokannan hallintajärjestelmien kohdalla. (Jhingran, 2000)

Näistä kolmesta tuotekatalogin tiedonhallinnan ongelmasta voidaan johtaa kaksi keskeistä tuotekatalogin tiedonhallinnan vaatimusta: skeeman joustavuus ja tiedon saatavuus.

2.1.1 Tiedon saatavuus

Toisin kun esimerkiksi inventaarion hallinta tai käyttäjätiedon hallinta, katalogidata on melko staattista tietoa, jossa päivitykset tehdään määrättyjen hallintaprosessien kautta. Katalogidataa ei päivitetä, mutta haetaan usein käyttäjän toimesta. Ostoskorin tietoja päivitetään usein käyttäjän toimesta, mutta yksi käyttäjä hakee vain yhden ostoskorin tietoja, kun taas samoja tuotteita voi samanaikaisesti hakea useita käyttäjiä. (Bochamann ym., 2003)

2.1.2 Skeeman joustavuus

Katalogin hallinta on haastavaa ympäristössä, jossa tavarantoimittajia voi olla jopa kymmeniä tuhansia. Toimittajien katalogit on pystyttävä yhdistämään yhteen verkkokaupassa esitettävään hierarkiaan. Uusia toimittajia tulee usein ja vanhoja poistuu, mikä tekee katalogin integroinnista entistä haastavampaa. Kaikkien katalogien tallentaminen omiin tauluihin relationaalisen tietokantaan ei ole mielekästä, mutta tiedon yhdistäminen yhteen tauluun ei ole ongelmatonta. (Jhingran, 2000, Agrawal, Somani & Xu, 2001)

Tuotedata on hajanaista sisältäen mahdollisesti tuhansia attribuutteja. Uudet tavarantoimittajat tuovat mahdollisesti mukanaan taas uusia attribuutteja, joten tietokannan skeemalta vaaditaan joustavuutta. Relationaalisessa tietokannassa perinteisesti varastoidaan tietoa horisontaalisessa skeemassa, jossa kullekin attribuutille luodaan oma sarake. Agrawal ym. (2001) kohtasivat seuraaviin ongelmiin yrittäessään varastoida tuotedataa horisontaalisessa skeemassa:

- **Sarakkeiden suuri lukumäärä.** Relaatiotietokannan hallintajärjestelmät (IBM DB2, Oracle Database) eivät salli yli 1012 sarakkeen tauluja, vaikka tarvetta olisi ollut 5000 attribuutin tallentamiseen.
- **Hajanaisuus.** Vaikka tietokanta olisi sallinut 5000 sarakkeen tauluja, olisi taulu sisältänyt paljon tyhjiä arvoja. Tyhjät arvot aiheuttavat tallennustilan ylijäämää ja kasvattavat indeksien kokoa.
- **Skeeman evoluutio.** Taulua pitäisi muokata aina uusien tuotteiden ja kategorioiden ilmetessä ja skeeman muokkaaminen on raskasta nykyisissä tietokannan hallintajärjestelmissä.
- **Suorituskyky.** Vaikka tietueen kyselyssä käsiteltäisiin vain muutamaa saraketta, on laajojen tietueiden kysely raskasta. (Agrawal ym., 2001)

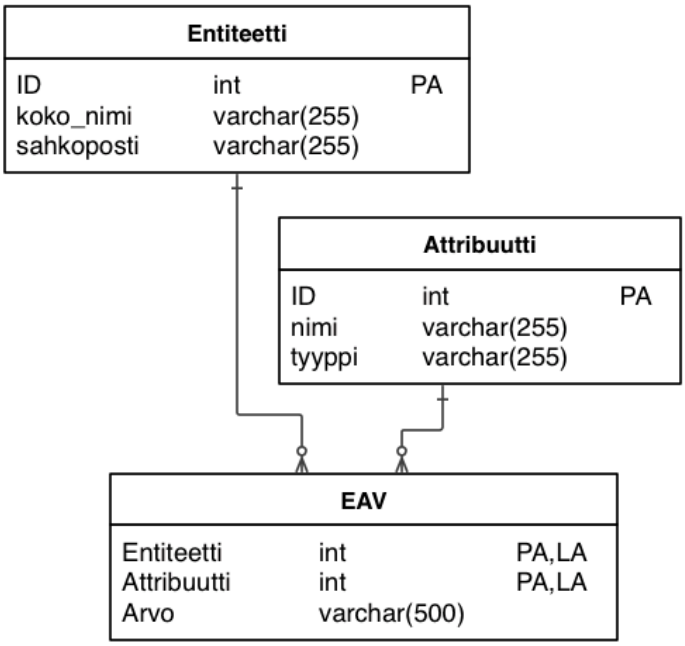
Ongelmien ratkaisemiseksi monet verkkokappasovellukset käyttävät vertikalisoitua skeemaa jossa tuoteattribuutit tallennetaan omaan kolmitietueiseen tauluun, joka sisältää tuotteen avaimen, sekä attribuutin nimen ja arvon. Relationaalisessa tietokannassa tämän on todettu olevan horisontaalista skeemaa tehokkaampi ratkaisu. (Agrawal ym., 2001)

2.2 Entiteetti-attribuutti-arvo -tietomalli

Entiteetti-attribuutti-arvo -tietomallia (EAV, entity-attribute-value) käytettiin alun perin lääketieteellisissä tietokannoista, joissa voi olla lukuisia potilasta koskevia attribuutteja, joista suurin osa koskee vain pientä osaa potilasta (Jastrow & Preuss, 2015).

EAV-tietomalli koostuu yksinkertaisimmillaan kolmesta tietokantataulusta: Entiteetti, attribuutti ja EAV-taulusta. Entiteetti-tauluun tallennetaan entiteetin yleiset tiedot, joiden ei odoteta muuttuvan usein. Attribuutti-tauluun

tallennetaan attribuutin metatiedot. Tähän kuuluu attribuutin nimi ja tyyppi, mutta myös muita metatietoja voidaan tarvittaessa tallentaa. EAV-tauluun tallennetaan attribuutin arvo, sekä liitosavain (LA) entiteetti-taulun ja attribuuttitaulun perusavaimiin (PA). Kolmesta tietokantataulusta koostuva EAV-tietomalli on kuvattu kuviossa 1. (Jastrow & Preuss, 2015)



KUVIO 1 EAV-tietomalli (Jastrow & Preuss, 2015, s. 495)

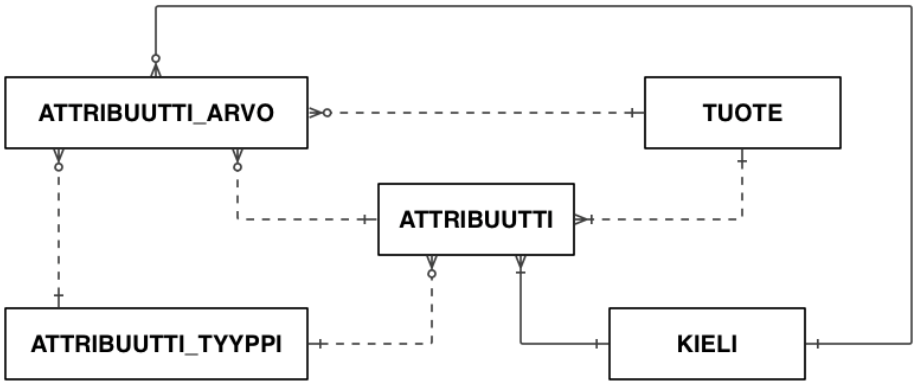
EAV-tietomallia käytetään myös verkkokauppasovellusten tietokannassa tuotekatalogin attribuuttien tallentamiseen (IBM, 2018, Magento, 2018). EAV-tietomalli voidaan nähdä jatkumona aikaisemmin esitettyyn vertikalisoituun skeemaan, jossa tuoteattribuutit tallennetaan kolmitietueiseen tauluun.

IBM WebSphere Commerce-verkkokauppasovelluksen tuoteattribuuttien tietomalli koostuu viidestä taulusta:

- tuote (catentry)
- attribuutti (attribute),
- attribuutti_arvo (attr_value)
- attribuutti_tyyppi (attr_type)
- kieli (language) (IBM, 2018)

IBM WebSphere Commerce-verkkokauppasovelluksen attribuutti-tietomalli jatkaa yksinkertaisesta kolmetauluista EAV-tietomallia kahden taulun osalta. *Kieli*-taulua tarvitaan tukemaan monikielisyyttä, jolloin kukin attribuutti tallennetaan jokaisella verkkokaupan tuetulla kielellä. Toinen uusi taulu on *attribuutin_tyyppi*-taulu, johon tallennetaan attribuutin metatieto, attribuutin tyyppi. Varsinainen EAV-taulu on *attribuutti_arvo*-taulu, johon attribuutin arvo ja tuotteen ja attribuutin liitosavain. *Attribuutti_arvo*-taulussa on varattu attribuutin arvolle kolumni

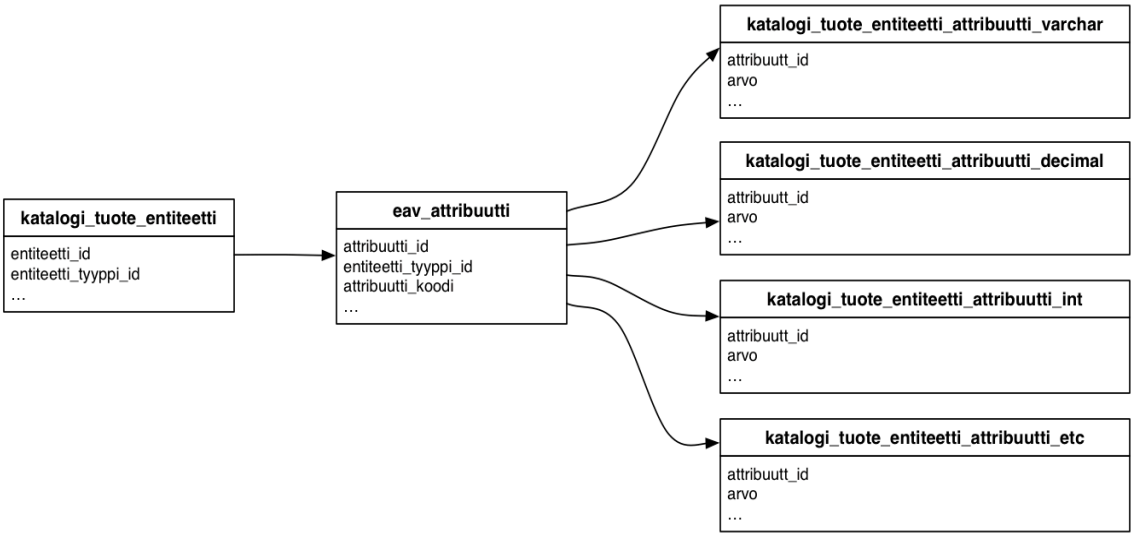
jokaiselle tuetulle tietotyypille: kokonaisluvulle, desimaalille ja merkkijonolle. Liitos *attribuutti_tyyppi*-tauluun määrittää mitä tietotyyppiä ja kolumnia käytetään kyseisen attribuutin tallentamiseen ja kyselyyn. IBM WebSphere Commerce-verkkokauppasovelluksen attribuutti-tietomalli on kuvattu kuviossa 2. (IBM, 2018)



KUVIO 2 IBM Commerce EAV-tietomalli (IBM, 2018)

Perustelu attribuutin tyyppi-metatiedon siirtämiseen omaan tauluun on todennäköisesti ollut vähentää turhaa tiedon toistuvuutta, koska attribuutin tyyppejä on hyvin rajallinen määrä. Toisaalta *attribuutti_arvo*-taulun sisältäessä attribuutin arvolle kolme varattua kolumnia, joista vain yksi on käytössä, sisältää kukin *attribuutti_arvo*-tietue vähintään kaksi tyhjää arvoa.

Magento-verkkokauppasovelluksen attribuutti-tietomalli jatkaa perinteistä kolmitauluista EAV-tietomallia useammalla EAV-taululla. Magento-verkkokauppasovelluksen tietomallissa tietotyypille on oma taulu, johon tallennetaan attribuutin arvo. Magento-verkkokauppasovelluksen EAV-tietomalli on kuvattu kuviossa 3. (Magento, 2018)

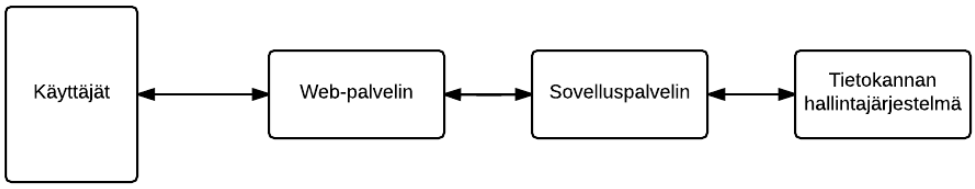


KUVIO 3 Magento EAV-tietomalli (Magento, 2018)

Magento-verkkokauppasovelluksen EAV-tietomallin toteutus poikkeaa IBM WebSphere Commerce-verkkokauppasovelluksen vastaavasta. Tyhjien arvojen tallentamiselta on vältytty luomalla jokaiselle attribuuttityypille oma taulu, mutta tämä lisää taulujen määrää ja oletettavasti lisää kyselylogiikan kompleksisuutta. Valitettavasti Magento-verkkokauppasovelluksen tietokannan tuoteattribuuttien tiedon mallintamisesta ei löytynyt julkista dokumentaatiota samalla tasolla, kun IBM WebSphere Commerce-verkkokauppasovelluksen toteutuksesta, joten yksityiskohtaisempi tietomallien vertailu on pelkästään dokumentaatiota tulkitsemalla vaikeaa.

2.3 Verkkokauppasovelluksen palvelinarkkitehtuuri ja skaalautuvuus

Verkkokauppasovellukselle perustuu tyypillisesti kolmitasoiseen palvelinarkkitehtuuriin. Ensimmäisellä tasolla web-palvelin käsittelee käyttäjän kutsuja ja palauttaa verkkosivuja. Keskimmäisessä tasolla on sovelluspalvelin, jossa ajetaan sovelluksen logiikkaa, ja joka tekee kyselyitä tietokanta-tasolle. Alimmalla tasolla on tietokannan hallintajärjestelmä, jota sovellus käyttää tiedon pysyvyyteen ja monimutkaisiin kyselyihin. Verkkokauppasovellukselle tyypillinen kolmitasoinen arkkitehtuuri on kuvattu kuviossa 4. (Jhingran, 2000, Bochamann ym., 2003)



KUVIO 4 Kolmitasoinen arkkitehtuuri (Bochamann ym., 2003)

Kolmitasoista arkkitehtuuria voidaan skaalata korkeammalle käyttöastelle klusteroimalla palvelintasoja. Käytännössä tämä tarkoittaa lisäämällä palvelimia kullekin tasolle ja jakamalla kuormaa eri palvelimille. Palvelinarkkitehtuurin klusteroinnin tekee haasteelliseksi rinnakkaisten tietokannan hallintajärjestelmien käyttäminen. Tietoa joko replikoidaan, eli kopioidaan palvelimien kesken, tai ositetaan tietokantojen välillä. Tiedon replikointi parantaa tiedon saatavuutta, mutta aiheuttaa haasteita tiedon eheydelle. Tiedon osittamisella omiksi itsenäisiksi kokonaisuuksiin palvelimille voidaan tukea tiedon täydellistä eheyttä, mutta tieto on saatavilla ainoastaan yhdeltä palvelimelta. (Bochamann ym., 2003)

2.4 Yhteenveto

Tässä luvussa ollaan esitelty tuotekatalogin tiedonhallinnan vaatimuksia, joita on tiedon saatavuus ja skeeman joustavuus. Lisäksi ollaan kuvattu EAV-tietomalli, jota on käytetty verkkokauppasovellusten tuoteattribuuttien tallentamiseen relationaaliseen tietokantaan. Lopuksi esiteltiin verkkokauppasovelluksen tyypillinen palvelinarkkitehtuuri ja kyseisen palvelinarkkitehtuurin skaalautuvuuden haasteita.

Tuotekatalogin tietokannan hallintajärjestelmän skeeman on oltava riittävän joustava, jotta useiden toimittajien katalogit saadaan yhdistettyä verkkokaupassa näytettäväksi tiedoksi. Katalogitiedon ylläpitäminen perinteisessä relationaalisessa tietokannassa on haastavaa joustavan skeeman ja tuoteattribuuttien suuresta määrästä johtuen. Haasteisiin on vastattu soveltamalla EAV-tietomallia, mutta ottaen huomioon, että eri verkkokauppasovellukset soveltavat tietomallia eri tavalla, voidaan olettaa, ettei tietomallin soveltaminen ole täysin haasteetonta.

Tuotekatalogin tiedon saatavuuden vaatimus on hyvin suoraviivainen ja yksinkertaistaa tiedonhallintaa. Tietoa päivitetään hallituin prosessein, joten tuotekatalogi ei aseta vahvoja vaatimuksia tietokannan hallintajärjestelmän tapahtumanhallintaominaisuuksille. Sen sijaan tiedon pitää olla käyttäjälle helposti ja nopeasti saatavilla.

Skaalautuvuus luo omat haasteensa tuotekatalogin tiedonhallinnalle. Perinteistä verkkokauppasovelluksen kolmitasoista palvelinarkkitehtuuria voidaan skaalata vain klusteroimalla, eli lisäämällä palvelin-instansseja, mikä luo omat haasteensa tiedon hallinnalle. Tietoa voidaan joko replikoida tai osittaa eri palvelimille riippuen halutaanko tukea tiedon saatavuutta vai eheyttä.

Monet tässä luvussa listatut tuotekatalogin tiedonhallinnan haasteet on esitetty jo 2000-luvun alussa (Agrawal ym., 2001, Bochamann ym., 2003, Jhingran, 2000). Tuotekatalogin tiedonhallinnan vaatimukset tuskin ovat muuttuneet vuosien saatossa, mutta tietokannan hallintajärjestelmät ovat todennäköisesti kehittyneet. Tuotekatalogin tiedonhallinnan vaatimuksia olisi syytä peilata modernien tietokannan hallintajärjestelmien ominaisuuksiin.

3 TIETOKANNAN HALLINTAJÄRJESTELMÄT PILVIYMPÄRISTÖSSÄ

Tässä luvussa kuvataan mitä pilvilaskenta tarkoittaa, miten se on muuttanut informaatioteknologiateollisuutta ja minkälaisia haasteita se luo tietokannan hallintajärjestelmille. Tämän jälkeen esitellään NoSQL-tietokantakonsepti ja NewSQL-tietokantakonsepti, jotka pyrkivät vastaamaan pilviympäristön haasteisiin. Lopuksi esitetään yhteenveto.

3.1 Pilvimaailman käsitteitä

Vaikka pilvilaskennasta on jo vuosia puhuttu, kirjoitettu blogeissa ja se on ollut useiden työpajojen, konferenssien ja lehtien otsikoissa, on ollut epäselvää mitä se tarkalleen tarkoittaa ja milloin se on hyödyllistä (Armbrust ym., 2009). Pilvilaskenta tuli terminä laajalti käyttöön vuosituhannen vaihteessa, jolloin siitä käytettiin aluksi IT-palvelukonseptin mukaista termiä SaaS (Software as a Service), eli verkkopalvelusovellus (Korpinen, 2011). Nykyään pilvi, pilvilaskenta ja pilvipalvelut termeinä ovat laajasti käytettyjä ja tarkoittavat monia eri asioita (Korpinen, 2011). Seuraavaksi määritellään keskeiset pilvimaailman käsitteet.

Pilven määritelmä pohjautuu tässä tutkielmassa yleisesti hyväksytyn ja viitattuun Vaqueron, Rodero-Merinon, Caceresin ja Lindnerin (2009) määritelmään (Korpinen, 2011). He kokosivat yhteen tieto- ja viestintätekniikan alan asiantuntijoiden määritelmiä pilvestä ja esittivät näiden pohjalta oman määritelmänsä pilvelle:

> ”Pilvet ovat suuri helposti käytettävä ja helppopääsyinen kokonaisuus virtualisoituja resursseja, kuten laitteistot, kehitysympäristöt ja palvelut. Nämä resurssit voivat dynaamisesti sopeutua muuttuneeseen kuormitukseen, mahdollistaen optimoidun resurssien käytön. Tästä resurssikokonaisuudesta maksetaan tavallisesti käytön mukaan. Käytön saatavuuden takaa pilvipalveluntarjoaja sovitun palvelutaso-sopimuksen mukaan.” (Vaquero ym., 2009, s. 51)

Määritelmä on varsin yleisluontoinen, ja kokoaa keskeisiä pilvilaskennan ominaisuuksia.

Pilvipalvelulla on viitattu sekä Internetin välityksellä palveluna tarjottuun ohjelmistoon että näiden palveluntarjoajien datakeskuksien laitteistoon ja järjestelmiin. Palveluna tarjottua ohjelmistoa kutsutaan verkkosovelluspalveluksi, ja siihen on jo pitkään viitattu termillä SaaS. Datakeskusten laitteistoa ja järjestelmiä kutsutaan pilveksi. Pilvipalvelujen tuottajat, kuten Amazon Web Services, Googlen AppEngine ja Microsoftin Azure, tarjoavat pilveä julkiseen käyttöön jatkuvan laskutuksen periaatteella. Tarjottua palvelua kutsutaan pilvilaskennaksi. (Armbrust ym.,2009)

Tässä tutkielmassa pilviympäristöllä tarkoitetaan pilvilaskentaa käyttäviä alustoja. Toisin sanoen pilviympäristössä skaalautuva sovellus sijaitsee pilvessä yksittäisen datakeskuksen sijasta.

3.1.1 Pilvilaskenta

Pilvilaskenta on uusi termi IT-alalla pitkään olleesta unelmasta tarjota tietokonelaskentaa yleishyödykkeen tapaan, mistä on nyt tullut todellista liiketoimintaa. Pilvilaskennalla on mahdollisuus muokata suurta osaa IT-alasta, koska se tekee ohjelmistojen toteuttamisesta verkkosovelluspalveluna entistä houkuttelevamman. Se vaikuttaa siihen, miten IT-laitteistoja suunnitellaan ja ostetaan. (Armbrust ym., 2009)

SaaS-verkkosovelluspalveluiden hyödyt niin loppukäyttäjälle kuin palveluntarjoajalle ymmärretään hyvin. Verkkosovelluspalveluntarjoajat pitävät keskitetystä ohjelmiston asennuksesta, ylläpidosta ja versionhallinnasta. Verkkosovelluspalvelun käyttäjät pääsevät käyttämään palvelua milloin tahansa ja mistä tahansa, sekä pystyvät helposti jakamaan dataansa ja varastoimaan sen turvallisesti palveluun. Pilvilaskenta mahdollistaa sen, ettei ohjelmistosuunnittelijoiden tarvitse enää sijoittaa suurta pääomaa palvelinlaitteistoon ja niiden ylläpitämiseen verkkosovelluspalveluidensa saataville saattamiseksi. Pilven resurssien skaalautuvuus tarpeen mukaan mahdollistaa myös sen, ettei pilvilaskennan asiakkaan tarvitse arvioida verkkosovelluspalvelunsa käyttöastetta etukäteen. Samaan tapaan kuin verkkosovelluspalveluiden käyttäjät voivat sysätä osan ongelmistaan verkkosovelluspalveluiden tarjoajille, voivat verkkosovelluspalvelun tarjoajat sysätä osan ongelmistaan pilvilaskennan tarjoajille, joita ovat esimerkiksi Amazon Web Services, Google AppEngine ja Microsoft Azure. (Armbrust ym., 2009)

Vaqueron ym. (2009) kokoamien ICT-alan ammattilaisten määritelmistä useimmin mainitut pilven ominaisuudet olivat skaalautuvuus, jatkuvan laskutuksen periaate ja virtualisointi. Samoja ominaisuuksia ilmenee myös Armbrustin ym. (2009 s. 4) raportissa, jossa listataan laitteiston kannalta kolme uutta näkökulmaa, jotka pilvilaskenta on tuonut:

1. Illuusio tietokoneresurssien ehtymättömyydestä tarvittaessa, mikä eliminoi pilvilaskennan asiakkaiden tarpeen varautua pitkälle etukäteen.
2. Pilven käyttäjien etukäteissitoutumisen poistuminen, mikä mahdollistaa sen, että yritykset voivat aloittaa pienestä ja lisätä laitteistoresursseja tarpeen mukaan.
3. Mahdollisuus maksaa tietokoneresursseista käytön mukaan lyhyellä aikavälillä, ja mahdollisuus vapauttaa tarpeettomia resursseja, näin palkiten säästeliäisyydestä. (Armbrust ym., 2009, s. 4)

Armbrust ym. (2009) mukaan kaikki kolme ovat tärkeitä tekijöitä pilvilaskennan mahdollistamaan tekniseen ja taloudelliseen muutokseen. Aiemmissa epäonnistuneissa yrityksissä tehdä tietokonelaskennasta hyötypalvelua on joku näistä kolmesta ominaisuudesta puuttunut (Armbrust ym., 2009).

3.2 Tietokannan hallinta pilvessä

Pilvilaskennan tietokoneresurssien elastisuus ja jatkuvan laskutuksen periaate on luonut uusia haasteita ohjelmistoille. Toimiakseen nopeasti ja kannattavasti pilvisovellusten pitää pystyä skaalautumaan nopeasti käyttöasteen kasvaessa ja myös laskiessa (Armbrust ym., 2009). Tämä on uusi haaste myös tietokannan hallintajärjestelmille (Feuerlicht, 2010).

Perinteisesti organisaatioiden datakeskukset ovat koostuneet suhteellisen pienestä määrästä tehokkaita tietokantapalvelimia (Feuerlicht & Pokorný, 2011). Relationaalisille tietokannoille on voitu luoda skaalautuvuutta vertikaalisesti lisäämällä prosessoritehoa ja käyttämällä nopeampia muistilaitteita (Pokorný, 2011). Pilven infrastruktuuri koostuu taas sadoista tuhansista edullisista palvelimista ja muistilaitteista (Feuerlicht & Pokorný, 2011). Pilvessä on käytännöllisempää ja halvempaa skaalautua horisontaalisesti osittamalla tietokanta tuhansille dynaamisesti lisättäville koneille (Pokorný, 2011). Tietokannan hallintajärjestelmän pitää samalla pystyä hallinnoimaan tiedon saatavuutta ja ristiriidattomuutta (Feuerlicht & Pokorný, 2011).

Abadi (2009) on maininnut kolme tiedonhallinnan kannalta relevanttia pilvilaskennan ominaisuutta:

- **Laskentateho on elastista, mutta ainoastaan jos työtehtävä voidaan tehdä rinnakkain**. Esimerkiksi kausittaisiin tai odottamattomiin verkkosovelluspalveluiden kysynnän piikkeihin voidaan allokoida lisää tietokoneresursseja minuuteissa. Tästä ei ole kuitenkaan mitään hyötyä, jos sovellus ei pysty jakamaan osaa työstään uudelle palvelininstanssille, joka toimii rinnakkain vanhojen palvelin instanssien kanssa. Tällaiseen ympäristöön soveltuvat parhaiten Shared-nothing –arkkitehtuurin mukaiset sovellukset, joissa itsenäiset koneet suorittavat tehtävän resurssien minimaalisella limittäisyydellä.
- **Data on varastoitu epäluotettavalle isännälle.** Vaikka pilvilaskennan tarjoajan liiketoiminnan kannalta asiakkaan datan yksityisyyden loukkaaminen ei vaikuta järkevältä, tekee sen mahdollisuus osan potentiaalisista asiakkaista hermostuneiksi. Data on aina fyysisesti varastoituna johonkin maahan ja on alistettu kyseisen maan laille ja säädöksille. Esimerkiksi Yhdysvalloissa hallituksen on mahdollista vaatia pääsyä minkä tahansa tietokoneen dataan, jolloin pilvilaskennan asiakkaan dataa voidaan antaa eteenpäin sen tietämättä. Asiakas ei voi muuta kuin pelätä pahinta, että jos dataa ei ole salattu avaimella, joka ei sijaitse isännän koneella, dataan voidaan päästä käsiksi ilman asiakkaan tietämystä.

- **Dataa kopioidaan usein yli pitkien maantieteellisten välimatkojen.** Datan saatavuus ja pysyvyys on tärkeintä pilvivarastojen tarjoajille, koska datan häviäminen tai saavuttamattomuus on vahingollista niin palvelun käyttäjälle kuin tarjoajan liiketoiminnan maineelle. Datan saatavuus ja pysyvyys on tyypillisesti saavutettu kopioimalla tietoa automaattisesti ilman asiakkaan pyyntöä. Datakeskusten ollessa ympäri maailmaa on virheensietokykyä parannettu tiedon kopioimisella ja hajauttamalla yli pitkien välimatkojen. (Abadi, 2009)

Kaikki nämä ominaisuudet sopivat heikosti transaktioperusteiseen (transactional) tiedonhallintaan. Tällä termillä Abadi (2009) viittaa tietokantojen keskeisimpiin sovelluksiin, joita pankkien, lentoyhtiöiden varausjärjestelmien, verkkokauppojen ja toimitusketjujen tiedon hallinnassa käytetään. Nämä tietokannat luottavat ACID (atomicity, consistency, isolation, durability) – tapahtumanhallintaominaisuuksiin, eli tiedon atomisuuteen, ristiriidattomuuteen, eristyneisyyteen ja pysyvyyteen. Nämä tietokannat perustuvat tyypillisesti Shared-everything – arkkitehtuuriin, jossa skaalautuvuus on huonompaa verrattuna Shared-nothing – arkkitehtuurin sovelluksiin. Nämä tietokannat sisältävät usein myös kaiken operatiivisen datan liiketoimintaprosessien läpivientiin, ja ne voivat sisältää myös arkaluontoisia asiakastietoja, kuten luottokorttitietoja. Näiden tietojen säilyttämiseen ulkopuolisena palveluna voi liittyä riski. (Abadi, 2009)

3.2.1 CAP-teoreema

ACID–ominaisuuksia on vaikea taata pilviympäristössä, jossa tietoa kopioidaan hajautetusti ympäri maailmaa (Abadi, 2009). Tilanteessa voidaan soveltaa Brewerin (2000) esittämää CAP-teoreema hajautetuille järjestelmille. CAP teoreema käsittelee kolmea ominaisuutta:

- **Ristiriidattomuus** (Consistency) tarkoittaa, että tieto näkyy samana kaikille käyttäjille.
- **Saatavuus** (Availability) tarkoittaa, että data on aina saatavilla ja muokattavissa odotetussa vasteajassa.
- **Osittamistoleranssi** (Partition tolerance) tarkoittaa, että tietokantaan voidaan suorittaa kyselyitä ja päivityksiä, vaikka osa tietokannasta on saavuttamattomissa. (Brewer, 2000)

CAP-teoreeman mukaan hajautettu järjestelmä, joka käyttää jaettua dataa, voi ylläpitää täydellisesti korkeintaan kahta ominaisuutta (Brewer, 2000). Brewer (2012) myöhemmin tarkensi, ettei CAP-ominaisuuksien välillä tehdä absoluuttisia valintoja, vaan kyse on ominaisuuksien balanssista. Perinteiset tietokannan hallintajärjestelmät ovat yleensä suosineet ristiriidattomuutta (Pokorný, 2011). CAP-teoreeman pohjalta voidaan kuitenkin todeta, että pilviympäristön kaltaisessa hajautetussa järjestelmässä täydellistä ristiriidattomuutta voidaan ylläpitää ainoastaan tiedon saatavuuden tai osittamistoleranssin kustannuksella (Brewer,

2000). Pilven koostuessa sadoista tuhansista edullisista koneista on osittamistoleranssi pakollinen ominaisuus tietokannan hallintajärjestelmälle, joten CAP-teoreeman pohjalta joudutaan pilvimaailmassa tekemään valintoja ristiriidattomuuden ja saatavuuden väliltä (Pritchett, 2008, Pokorný, 2011).

3.2.2 BASE-ominaisuudet

Relaationaaliset tietokannan hallintajärjestelmät ovat käytännössä aina tukeneet täydellisesti ACID-ominaisuuksia ja jatkuvaa ristiriidattomuutta (Pokorný, 2011). Monet web-sovellukset sietävät tiedon väliaikaista ristiriitaisuutta, mutta kyselyiden kasvavat vasteajat tai tiedon saavuttamattomuus voi olla kallista (Feuerlicht & Pokorný, 2011, Abadi, 2009). Näille sovelluksille on kehitetty uusi tapahtumanhallintaominaisuuksien malli, jota kutsutaan BASE-ominaisuuksiksi (Pritchett, 2008). BASE-ominaisuuksia tukeva tietokanta on aina saatavilla (basically available), ei ole välttämättä yhtenevässä tilassa (soft state), mutta takaa että tietokanta saavuttaa lopulta ristiriidattoman tilan (eventual consistency) (Pritchett, 2008).

3.3 NoSQL-tietokantakonsepti

Pilvimaailman tietokannanhallinnan haasteisiin on kehitetty useita kaupallisia ja avoimeen lähdekoodiin perustuvia NoSQL-tietokantatuotteita, joiden tarkoituksena on tarjota elastisuutta, automatisoitua tiedon saatavuuden hallintaa (automatic provisioning) ja virheensietokykyä. NoSQL -termin alle kuuluvia tietokantatuotteita ovat esimerkiksi Redis, Membase, MongoDB, CouchDB, Cassandra, Hbase, Neo4j ja GraphDB (Hecht & Jablonski, 2011).

Yhteinen tekijä kaikille NoSQL-tietokantatuotteille on se, etteivät ne perustu relaatioihin, eivätkä ne käytä SQL-kieltä kyselykielenään (Feuerlicht & Pokorný, 2011). SQL-kieltä saatetaan kuitenkin käyttää osana tietokannan hallintajärjestelmää, joten NoSQL-termin on monesti tulkittu tarkoittavan, ettei kyseisen termin alle kuuluvat tietokantasovellukset käytä pelkästään SQL-kieltä (not-only-SQL) (Pokorný, 2011). Horisontaalisen skaalautuvuuden mahdollistaminen NoSQL-tuotteissa on johtanut relaationaalisesta tietomallista luopumiseen, jonka tilalle tarjotaan yksinkertaisempia tietomalleja (Pokorný, 2011).

3.3.1 Avain-arvo-varastot

Avain-arvo-varastot (key-value store) tallentavat tietoa yksilöivän avaimen perusteella. Tietoa hallinnoidaan pelkästään avaimen perusteella, joten lisäys-, päivitys- ja poisto-operaatiot tehdään avaimen perusteella. Arvot voivat olla tietokannan hallintajärjestelmän kannalta mitä tahansa ja ne ovat toisille näkymättömiä. Tietomallissa arvoilla ei ole liitoksia toisiin arvoihin. Yksinkertainen tietorakenne tarjoaa skeemasta vapaan tietovaraston, jolloin uusia arvoja voidaan lisätä ajon aikana ilman konflikteja niiden tyypistä riippumatta. Mahdolliset arvojen

liitokset voidaan toteuttaa esimerkiksi tallentamalla tietovarastoon arvo, joka sisältää kokoelman avain-arvo-pareja ja tekemällä tarvittavat tietoliitokset sovellustasolla. (Hecht & Jablonski, 2011)

Avain-arvo-varastot usein säilyttävät tietovarastoaan tietokoneen muistissa, joten ne soveltuvat hyvin sovelluksille, jotka vaativat hyvää suorituskykyä uniikin muuttujan perusteella tehdyille luku- ja kirjoitusoperaatioille. Esimerkiksi käyttäjäkohtaisen web-sivun laskemista voidaan nopeuttaa hakemalla käyttäjäkohtainen tieto avain-arvo-varastosta. Avain-arvo-varastot soveltuvat huonosti monimutkaisille tietokantakyselyille, mutta niitä käytetään usein myös relaatiotietokannan hallintajärjestelmän rinnalla raskaiden SQL-kyselyiden välimuistina. (Hecht & Jablonski, 2011)

3.3.2 Dokumenttivarastot

Dokumenttivarastot tallentavat tiedon uniikin avaimen perusteella. Avain-arvo-varastoista poiketen arvot on tallennettu rakenteellisessa muodossa. Usein rakenne on tallennettu JSON-muodossa (JavaScript Object Notation) tai JSON:n kaltaisessa muodossa. Tämä mahdollistaa myös monimutkaisempien tietorakenteiden, esimerkiksi listojen ja sisäkkäisten olioiden, tallennukseen. (Hecht & Jablonski, 2011)

Yksilöivänä avaimena toimii dokumentin sisäinen kenttä. Arvo haetaan avaimen perusteella ja tieto indeksoidaan avaimen perusteella. Arvot eivät ole myöskään näkymättömiä varastolle, joten kyselyitä voidaan tehdä myös dokumentin kenttien perusteella tai tiettyyn kenttään kohdistetun arvovälin perusteella. Dokumenteilla ei ole skeema-rajoitteita, joten uusia kenttiä voidaan lisätä dokumenteille ajon aikana ilman konflikteja. (Hecht & Jablonski, 2011)

Dokumenttivarastot tukevat useampaan dokumentin kenttään kohdistuvia avain-arvo-kyselyitä, missä avain-arvot-parit voivat olla erityyppisiä. Tämän vuoksi dokumenttivarastot soveltuvat erityisen hyvin tiedon integraatioon ja skeemojen migraatioon. (Hecht & Jablonski, 2011)

3.3.3 Sarakeperhevarastot

Sarakeperhevarastot pohjautuvat Googlen Bigtable-tallennusjärjestelmään (Hecht & Jablonski, 2011). Bigtable-tallennusjärjestelmässä tietokanta koostuu *Bigtable-tauluista*. Bigtable-taulu on harva (sparse), hajautettu, pysyvä, moniulotteinen ja lajiteltu kartta (map). Kartta on indeksoitu kolmiulotteisesti riviavaimen, sarakeavaimen ja aikaleiman mukaisesti. Yksittäistä alkiota kolmen ulottuvuuden risteyskohdassa kutsutaan *soluksi* (cell). Solujen arvot ovat tulkitsemattomia merkkijonoja. (Chang ym., 2008)

Rivit on sanakirjamaisesti (lexicographic) järjestetty riviavaimen mukaan. Tämä mahdollistaa, että tietovaraston käyttäjät voivat riviavaimen vallinnalla järjestää toisiinsa liittyvä tieto lähelle toisiaan, millä voidaan tehostaa rivijoukkoon kohdistuvia kyselyitä. Riviavain voi olla jopa 64 kilotavua pitkä. Kaikki yksittäiseen riviin kohdistuvat transaktiot ovat sarjallistuvia. Sen sijaan rivijoukon

osalta sarjallistuvuus ei toteudu. Peräkkäisiä rivejä ryhmitellään *tableteiksi* (tablet), joita hallinnoimalla voidaan tasata järjestelmän kuormitusta. (Chang ym., 2008)

Sarakeavaimet ovat ryhmitelty *sarakeperheisiin* (columnfamily), jotka muodostavat tiedon saantiyksikön. Tiedon pakkaamisen helpottamiseksi samantyyppinen tieto tallennetaan yleensä samaan sarakeperheeseen. Datan käsittely tapahtuu sarakeperheittäin, mikä mahdollistaa, että eri kolumneille voidaan antaa käyttäjästä riippuen erilaisia oikeuksia datan käsittelemiseen. (Chang ym., 2008)

Tiedoista voidaan tallentaa useita versioita, jotka tunnistetaan aikaleiman (vrt kuvion 1 t_3, t_5, t_6 -merkinnät) avulla. Versiot järjestetään päivityksen mukaan alkaen uusimmasta, jolloin uusin versio luetaan ensimmäisenä. Käyttäjä voi määrittää kuinka monta versioita säilytetään, tai kuinka vanhoja versioita säilytetään. (Chang ym., 2008)

3.3.4 Verkkotietokannat

Perinteisistä relationaalisista tietokannoista ja avaimen mukaan hallinnoivista tietovarastoista poiketen verkkotietokannat ovat erikoistuneet vahvasti linkitetyn tiedon hallinnoimiseen. Verkkotietokannat soveltuvat sovelluksille, jotka käyttävät paljon keskinäisiä suhteita sisältävää tietoa. Tietoa on tehokkaampi hakea liitoksia seuraamalla kuin rekursiivisten taululiitosten avulla. (Hecht & Jablonski, 2011)

3.4 NewSQL-tietokantakonsepti

Relaatioista luopuminen NoSQL-tietokantatuotteissa on tarkoittanut myös, että käyttäjälle jää suurempi vastuu tiedon oikeellisuuden hallinnassa. Tärkeimpänä puutteena voidaan kuitenkin pitää ACID-ominaisuuksien puuttumista (Pokorný, 2011). Yritysten liiketoiminnassa käytettäville tietokannoille ACID-ominaisuudet ovat pakollisia (Pokorný, 2011). Tämän ongelman ratkaisemiseksi on kehitetty NewSQL-tietokantakonsepti, joka tarjoaa relationaalisen tietomallin, SQL-kielen kyselykielenä, ACID-ominaisuuksia ja horisontaalista skaalautuvuutta (Bernstein, 2014). Pavlo & Aslett (2016) määrittelevät NewSQL:n seuraavasti

> "Meidän määritelmän mukaan NewSQL on modernien tietokannan hallintajärjestelmien joukko, joka pyrkii tarjoamaan NoSQL:n tasoista skaalautuvuutta tuotantotietokannan (OLTP) luku- ja kirjoitus kuormille, tukien samalla ACID-ominaisuuksia" (Pavlo & Aslett, 2016, s. 46)

Määritelmä ottaa kantaa mihin haasteisiin NewSQL tietokannan hallintajärjestelmät pyrkivät vastaamaan, mutta ei ota vielä kantaa niiden tekniseen toteutukseen. Teknisen toteutuksen näkökulmasta Pavlo & Aslett (2016) luokittelee NewSQL tietokannan hallintajärjestelmät kolmeen kategoriaan:

1. Uusi järjestelmä, joka on rakennettu alusta lähtien uutta arkkitehtuuria käyttäen.
2. Järjestelmät, jotka käyttävät uudenlaista väliohjelmistoa (middleware), joka tukee tiedon sirpalointia (sharding) useammalle palvelimelle.
3. Palveluna tarjottavat tietokannat (DBaaS, Database-as-a-service), jotka perustuvat uudelle arkkitehtuurille. (Pavlo & Aslett, 2016)

Uuden arkkitehtuurin NewSQL-tietokannan hallintajärjestelmät on rakennettu kokonaan uuden koodipohjan päälle, eivätkä sen takia kanna perinnejärjestelmien teknistä velkaa. Nämä tietokannan hallintajärjestelmät perustuvat hajautettuun arkkitehtuuriin, jossa operoidaan resursseja jakamatta (shared-nothing), ja joka sisältää komponentteja, jotka tukevat usean solmun samanaikaisuuden ohjausta (multi-node concurrency control), virheensietokykyä replikoinnin avulla, tietovuon ohjausta (flow control) ja hajautettua kyselyiden prosessointia. Hajautetulle ajolle rakennettujen tietokannan hallintajärjestelmien etu on se, että kaikki järjestelmän osat voidaan optimoida usean solmun järjestelmiin (multi-node environments). (Pavlo & Aslett, 2016)

Uudenlaiset väliohjelmistot mahdollistavat tiedon sirpaloinnin (sharding) palvelin klusterille, jossa sirpaleet hajautetaan yksittäisille solmuille, joissa ajetaan yksittäistä tietokannan hallintajärjestelmän instanssia. Sirpalointi poikkeaa perinteisistä 1990-luvun tietokannan hallintajärjestelmien liityntä tekniikoista, koska kukin solmu ajaa samaa tietokannan hallintajärjestelmää, joka sisältää vain osan kokonaisesta tietokannasta, eikä se ole suunniteltu usean erillisen sovelluksen tiedon käyttöön ja päivittämiseen. Väliohjelmisto ohjaa kyselyt, koordinoi transaktioita ja hallitsee tiedon sijoitusta, replikointia ja osittamista solmujen välillä. Sirpalointia tukevien väliohjelmistojen etu on, että ne esittävät sovellukselle yhden loogisen tietokannan, ja korvaavat siten helposti jo yksittäistä tietokantaa käyttävän sovelluksen olemassa olevan tietokannan hallintajärjestelmän. (Pavlo & Aslett, 2016)

On olemassa pilvipalvelun tuottajia, jotka tarjoavat NewSQL-tietokantatuotteita palveluna. Näiden palvelujen avulla organisaatioiden ei tarvitse ylläpitää tietokannan hallintajärjestelmiä omalla laitteistolla tai pilvipalveluina tarjottavissa virtuaalikoneissa. Tietokantapalveluntarjoaja on vastuussa tietokannan fyysisen konfiguraation ylläpidosta, kuten järjestelmän hienosäädöstä, replikoinnista ja varmuuskopioinnista. Monet tietokantapalvelut ovat vain ylläpidettyjä perinteisiä yhden solmun (single-node) tietokannan hallintajärjestelmiä, mutta NewSQL-tietokantapalveluiksi lasketaan vain ne tietokantapalvelut, jotka perustuvat NewSQL-tietokantakonseptin mukaiseen uuteen arkkitehtuuriin. Näistä merkittävin on Amazonin Aurora MySQL RDS, joka käyttää loki-pohjaista tallennushallintaohjelmaa parantaakseen luku- ja kirjoitusoperaatioiden rinnakkaisuutta. (Pavlo & Aslett, 2016)

3.5 Yhteenveto

Pilviympäristö luo uusia haasteita tietokannan hallintajärjestelmille. Perinteisten relationaalisten tietokantojen osittaminen useammalle palvelimelle ei ole nähty riittävän tehokkaana tapana tarjota moderneilta verkkosovelluksilta vaadittavaa skaalautuvuutta. Relationaalisesta tietomallista luopuminen on mahdollistanut useampien tarvekohtaisten tietomallien kehityksen. Ajatus, että yksi tietokannan hallintajärjestelmä toimisi kaikille sovelluksille koetaan vanhentuneeksi, joten lukuisia uusia tietokantasovelluksia on kehitetty vastaamaan tiettyihin tarpeisiin. Tietokannan hallintajärjestelmän valinnassa ei ole enää kyse siitä keneltä toimittajalta perinteinen relaatiotietokannan hallintajärjestelmä valitaan. Sovelluksen tietokannan hallintajärjestelmän valinta perustuu nykyään sovelluksen käyttötarkoitukseen ja siitä johdettuihin vaatimuksiin tietokannalle.

NoSQL-tietokannat tarjoavat lukuisten tietomallien lisäksi horisontaalista skaalautuvuutta pilviympäristössä. Tietokannan hajauttaminen mahdollistaa tiedon parempaa saatavuutta, mutta luovat haasteita tiedon ristiriidattomuudelle. Pilviympäristössä joudutaan tekemään valintoja tiedon saatavuuden ja ristiriidattomuuden välillä. Näitä ominaisuuksia painotetaan sovelluksesta riippuen. Sosiaalisen median sovellusten on tarjottava aina hyvää tiedon saatavuutta, mutta tiedon väliaikainen ristiriidattomuus on sallittavaa. Toisaalta kaupan ja pankkialan sovelluksilta vaaditaan tiedon jatkuvaa ristiriidattomuutta.

Pitkään uskottiin, ettei tapahtumanhallintaominaisuuksia ja relationaalista tietomallia voida skaalata pilviympäristössä. NewSQL-tietokantakonsepti on haastanut tätä käsitystä muuttamalla perinteisten relationaalisten tietokantojen arkkitehtuuria. NewSQL-tietokannat pyrkivät ratkaisemaan niitä ongelmia, mitkä ovat estäneet transaktiokriittisten sovellusten siirtymistä pilviympäristöön.

4 VERKKOSOVELLUSTEN SUORITUSKYKYTESTAUS

Tässä luvussa kuvataan verkkosovellusten suorituskykytestaamiseen eri tyypit ja siihen kuuluvat vaiheet. Luvun rakenne pohjautuu Meier, Farre, Bansode, Barber & Rea (2009) teokseen, jossa esitetään ohjeistus verkkosovellusten suorituskykytestaukseen. Ensimmäiseksi kuvataan eri verkkosovellusten suorituskykytestauksen tyypit ja tämän jälkeen suorituskykytestaukseen liittyvät aktiviteetit. Lopuksi esitetään yhteenveto, jossa otetaan myös kantaa ohjeistuksen soveltuvuudelle pilvipalveluna tarjottavan tietokantatuotteiden suorituskykyvertailulle tuotekatalogin tietokantana.

4.1 Suorituskykytestauksen tyypit

Suorituskykytestaus on yleistermi, jolla voidaan viitata eri tyyppisiin suorituskykyyn liittyviin testeihin, joista kukin on suunnattu spesifille ongelma-alueelle. Meier ym. (2009) mukaan yleisimmät verkkosovellusten liittyvät suorituskykyyn liittyvät kysymykset ovat:

- Onko se riittävän nopea?
- Pystyykö se palvelemaan kaikkia asiakkaitani?
- Mitä tapahtuu, jos jokin menee vikaan?
- Mihin pitää varautua asiakasmäärän kasvaessa? (Meier ym., 2009)

Nämä kysymykset luovat perustan neljälle erityyppiselle verkkosovellusten suorituskykytestaukselle:

1. Suorituskykytestaus
2. Kuormitustestaus
3. Stressitestaus
4. Kapasiteettitestaus. (Meier ym., 2009)

Suorituskykytestaus on tekninen selvitystyö, jonka tarkoitus on määrittää tai validoija järjestelmän nopeus, skaalautuvuus ja vakaus, tai osa näistä ominaisuuksista. Tavoitteena on selvittää, onko järjestelmän käyttäjä tyytyväinen järjestelmän suorituskykyominaisuuksiin, ja vastaavatko odotukset järjestelmän suorituskyvystä todellisuutta. Suorituskykytestaus tukee järjestelmän hienosäätöä, optimointia ja kapasiteettisuunnittelua. Suorituskykytestaus ei välttämättä löydä kaikkia funktionaalisia vikoja, jotka ilmenevät järjestelmää kuormittaessa. Mikäli suorituskykytesti ei ole huolellisesti suunniteltu ja validoitu, sen tulokset ilmaisevat suorituskyvystä, joka koskee ainoastaan pientä osaa tuotannon skenaarioista. (Meier ym., 2009)

Kuormitustestauksen tavoite on todentaa järjestelmän toimintakyky normaalilla ja huippukorkealla kuormituksella. Tarkoitus on verifioida, että järjestelmä vastaa sille asetettuja suorituskykytavoitteita, jotka usein määritellään järjestelmän palveluntasosopimuksessa. Kuormitustestauksella mitataan järjestelmän vasteaikoja, suoritustehoa ja resurssien käyttöastetta, sekä voidaan määrittää järjestelmän hajoamispiste, mikäli se ilmenee ennen huippukuormitusta. Yksi kuormitustestin muodoista on kestävyystesti, jossa pyritään määrittämään tai validoimaan järjestelmän suorituskykyominaisuuksia kuormittamalla sitä pitkiä aikajaksoja kuormitusmalleilla ja -tasoilla, jonka oletetaan vastaavan todellista käyttäjäkuormaa todellisessa ympäristössä. Kuormittavuustestauksen pääasiallinen tarkoitus ei ole keskittyä vasteaikojen nopeuteen, vaan tuloksia on syytä verrata toisiin vastaaviin kuormittavuustesteihin. (Meier ym., 2009)

Stressitestauksen tavoitteena on selvittää, miten järjestelmä käyttäytyy epänormaalin korkealla kuormituksella. Tarkoituksena on paljastaa muistivuodoista, synkronointiongelmista tai resurssien kilpailutilanteesta johtuvia järjestelmävirheitä, jotka ilmenevät vain hyvin korkealla kuormituksella. Tavoitteena on selvittää voiko data korruptoitua tai paljastuuko tietoturva-aukkoja, jos järjestelmää ylikuormitetaan. Stressitestauksen avulla voidaan määrittää, mitä monitorointityökaluja kannattaa asettaa ja minkälaisiin järjestelmävirheisiin on hyvä suunnitelmallisesti varautua. Stressitestauksessa käytettävän kuormituksen määrää on vaikea arvioida etukäteen. Huonosti eristetty testiympäristö saattaa myös johtaa järjestelmä- tai tietoverkkovirheistä johtuviin odottamattomiin häiriöihin. Osa sidosryhmistä saattaa vähätellä stressitestauksen tuloksia, koska lähtökohtaisesti stressitestauksella testataan epänormaalin korkeaa kuormitusta. (Meier ym., 2009)

Kapasiteettitestauksella pyritään selvittämään, kuinka suurella kuormalla järjestelmä pystyy palvelemaan käyttäjiä suorituskykytavoitteiden mukaisesti. Kapasiteettitestausta tehdään yhdessä kapasiteettisuunnittelun kanssa. Kapasiteettisuunnittelun tarkoitus on varautua tulevaan kasvuun, kuten käyttäjämäärien tai datan määrän kasvuun. Suunnittelun tarkoitus on selvittää mitä resursseja, kuten prosessoreita, muistia, levytilaa tai tietoverkon siirtonopeutta, tarvitaan, jotta järjestelmä pystyy palvelemaan käyttäjiä tulevilla käyttöasteilla. Kapasiteettitestauksen avulla voidaan määrittää järjestelmän nykyinen käyttöaste ja kapasiteetti, sekä voidaan validoija ja vertailla kapasiteetti suunnittelun malleja ja ennustuksia järjestelmän kapasiteetista. Kapasiteettimallien validoivien testien toteuttaminen on kuitenkin monimutkaista ja vaikea toteuttaa kaiken kattavasti silloin kun niiden validoiminen olisi arvokkainta. (Meier ym., 2009)

4.2 Suorituskykytestauksen aktiviteetit

Suorituskykytestaus on monitahoinen toiminto, josta on vaikea muodostaa yleistä lähestymistapaa, jota voisi soveltaa kaikissa, tai edes useimmissa tilanteissa (Meier ym., 2009). On kuitenkin joitain aktiviteetteja, jotka toistuvat lähes kaikissa projektiluontoisessa suorituskykytestauksessa (Meier ym., 2009). Meier ym. (2009) listaa nämä testausaktiviteetit seuraavasti:

1. Identifioi testiympäristö
2. Identifioi hyväksymiskriteerit
3. Suunnittele suorituskykytesti
4. Konfiguroi testiympäristö
5. Toteuta suunniteltu suorituskykytesti
6. Suorita suorituskykytesti
7. Analysoi tulokset, raportoi ja testaa uudelleen. (Meier ym., 2009)

Oleellista näiden aktiviteettien tehokkaaseen toteuttamiseen ei ole aktiviteettien ajoitus, päällekkäisyys tai iteraatiomalli, vaan tärkeintä on ymmärtää jokaisen vaiheen konsepti ja niiden toteuttaminen tavalla, joka tuo eniten lisäarvoa projektiympäristöön, missä suorituskykytestiä toteutetaan (Meier ym., 2009). Seuraavaksi kuvataan tarkemmin kukin vaihe.

4.2.1 Testiympäristön tunnistaminen

Kaikki suorituskykytestin suorittamiseen tarvittavat työkalut ja laitteisto koostavat testiympäristön. Mikäli suorituskykytestin tarkoitus on määrittää verkkosovelluksen suorituskyky ominaisuuksia tuotantoympäristössä, on ideaali tilanteessa testiympäristö kuormitus- ja monitorointityökaluja lukuun ottamatta täydellinen kopio tuotantoympäristöstä. Täydelliset ympäristöjen kopiot ovat kuitenkin harvinaisia, ja poikkeavuuden aste on tärkeä ottaa huomioon päätettäessä, mitä testejä suoritetaan ja millä kuormitusasteella. (Meier ym., 2009)

Testiympäristön identifioimisen kannalta kriittisiä tekijöitä Meierin ym. (2009) mukaan on:

- **Laitteisto**
  - Konfiguraatiot
  - Fyysinen laitteisto (Prosessorit, Muisti jne.)
- **Verkko**
  - Verkon arkkitehtuuri ja loppukäyttäjän sijainti
  - Kuormantasauksen osallisuus
  - Klusteri ja nimipalvelin konfiguraatio
- **Työkalut**
  - Kuormitustyökalun rajoitukset
  - Monitorointityökalujen vaikutus ympäristöön
- **Ohjelmisto**
  - Muut asennetut tai käynnissä olevat ohjelmistot jaetussa tai virtuaalisessa ympäristössä
  - Ohjelmisto lisenssien rajoitukset tai poikkeavuudet
  - Tallennus kapasiteetti ja ympäristöön syötetty data
  - Lokin keräämisen tasot
- **Ulkoiset tekijät**
  - Muu liikenne verkossa
  - Ajastetut eräajot tai varmuuskopiointi prosessit
  - Vuorovaikutus muiden järjestelmien kanssa (Meier ym., 2009)

Testiympäristöä kuvattaessa on tärkeä identifioida järjestelmän kriittiset komponentit. Näitä on esimerkiksi komponentit, joilla on tiedetty merkitys suorituskykyyn tai komponentit, joita ei voida kontrolloida testin yhteydessä. Tärkeää on myös identifioida järjestelmään syötetyn datan tyyppi ja määrä, jolla emuloidaan todellisen maailman tilannetta. (Meier ym., 2009)

4.2.2 Hyväksymiskriteerien tunnistaminen

Verkkosovelluksen toivottuja suorituskykyominaisuuksia on hyvä aloittaa määrittelemään tai vähintään arvioimaan hyvissä ajoin kehitysprosessia. Käyttäjien tai sidosryhmien tyytyväisyyteen vaikuttavat suorituskykyominaisuudet voidaan jakaa Meier ym. (2009) mukaan kolmeen luokkaan:

- **Vasteaika**. Esimerkiksi tuotekatalogin oltava nähtävillä alle kolmessa sekunnissa.
- **Suoritusteho**. Esimerkiksi järjestelmästä on pystyttävä tilamaan 25 kirjaa sekunnissa.
- **Resurssien käyttö**. Esimerkiksi prosessorin käyttöaste ei saa olla yli 75 prosenttia. (Meier ym., 2009)

Näiden ominaisuuksien lisäksi on hyvä määritellä suorituskykytestauksen onnistumiskriteerit; esimerkiksi suotuisimpien suorituskykyominaisuuksien mahdollistavien konfiguraatioiden löytäminen suorituskykytestauksen avulla. (Meier ym., 2009)

4.2.3 Testien suunnitteleminen

Suunniteltaessa testiä, jonka tarkoituksena on luonnehtia verkkosovelluksen tuotannon suorituskykyä, tavoite on luoda simulaatio todellisesta maailmasta. Testisuunnitelmien perustuminen todelliseen maailmaan lisää testin tuloksien luotettavuutta ja käytettävyyttä, mikä antaa mahdollisuuden organisaatiolle tehdä valistuneita liiketoimintapäätöksiä. Meierin ym. (2009) mukaan testisuunnitteluun kuuluu:

- Käyttöskenaarioiden tunnistaminen
- Käyttäjien vaihtelevuuden määrittäminen
- Testidatan tunnistus ja luonti
- Mittareiden määrittäminen (Meier ym., 2009)

Keskeiset käyttöskenaariot yleensä nousevat esille määriteltäessä verkkosovelluksen toivottuja suorituskykyominaisuuksia. Keskeisiä käyttöskenaarioita voidaan tunnistaa myös tarkastelemalla esimerkiksi palvelusopimuksen velvoittamia käyttöskenaarioita, liiketoimintakriittisiä käyttöskenaarioita, suoritusraskaita käyttöskenaarioita tai teknisesti haasteellisia käyttöskenaarioita. (Meier ym., 2009)

Oikein määritellyt mittarit ja niistä oikein kerätty ja raportoitu tieto auttavat sovelluksen todellisen suorituskyvyn ja toivotun suorituskyvyn vertailussa. Mittarit voivat auttaa myös ongelmien ja pullonkaulojen löytämisessä. Testin suunnitteluvaiheessa on oleellista tunnistaa ne mittarit, jotka liittyvät tunnistettuihin hyväksymiskriteereihin, jotta toteutusvaiheessa nämä mittarit voidaan ottaa osaksi testiä. (Meier ym., 2009)

4.2.4 Testiympäristön konfigurointi

Kuormitus- ja monitorointityökalujen valmistelu on usein oletettua haastavampaa. Ongelmia voi nousta esiin hyvin erilaisista konfiguraation kohteista, esimerkiksi verkkoympäristöstä, laitteiston hankinnassa, testiin tarvittavien IP-osoitteiden koordinoimisessa, työkalujen ohjelmistoversioiden yhteensopivuudesta tai palvelinten käyttöjärjestelmistä. Testiympäristön konfigurointi on hyvä aloittaa aikaisin, jotta voidaan varmistaa, että kaikki ongelmat on selvitetty ennen testien aloittamista. Testiympäristöä on syytä myös määräajoin uudelleenkonfiguroida ja ylläpitää, koska vaikka itse verkkosovellus pysyisikin samana, tarkastelun kohteena olevat mittarit voivat muuttua. Testiympäristöä konfiguroidessa on tärkeää määrittää se kuormituksen taso, missä kuormitustyökalusta tulee testin pullonkaula. Tyypillisesti kuormitustyökalun pullonkaula tulee vastaan laskentaresurssien näkökulmasta ensin muistin ja sitten prosessorin osalta. (Meier ym., 2009)

4.2.5 Testisuunnitelman toteuttaminen

Suorituskykytestin toteuttaminen tyypillisesti koostuu yksittäisten käyttöskenaarioiden ohjelmoinnista komentosarjoiksi, kehittämisestä ja yhdistämisestä muihin skenaarioihin, kunnes saadaan lopullinen kuormitusmalli. Suoritettavan suorituskykytestin luonnin yksityiskohdat ovat erittäin työkaluriippuvaisia, ja työkalut ovat vääjäämättä kehittyviä teknologioita ja käytäntöjä perässä. Usein suorituskykytestauksen suurin haaste onkin ensimmäisen suhteellisen realistisen testin toteuttaminen, missä testattava sovellus ei pysty erottamaan simuloidun ja oikean käyttäjän eroa. Testisuunnitelmaa toteuttaessa on tärkeä varmistaa, että syötetty data kuormitustyökaluun ja testattavaan sovellukseen on toteutettu oikein. Datan on oltava riittävän kattavaa, mutta usein osajoukko tuotannon datasta riittää. Tärkeää on myös varmistaa, että testin suorittamat transaktiot validoijaan oikein. Web-palvelin saattaa raportoida transaktion onnistuneeksi, vaikka se ei kokonaisuudessaan olisikaan onnistunut odotetulla tavalla. Tässä vaiheessa on myös hyvä varmistaa suorituskykymittareiden monitorointi. (Meier ym., 2009)

4.2.6 Testin suorittaminen

Testin suorittaminen on huomattavasti monitahoisempi prosessi kuin testin aloittaminen ja sen monitorointi. Testin suorittaminen voidaan nähdä seuraavien osatehtävien kombinaationa:

1. Testin suorittamisen ja monitoroinnin koordinointi
2. Testien, konfiguraatioiden, testiympäristön tilan sekä datan validointi
3. Testin suorituksen aloittaminen.
4. Komentosarjojen, järjestelmien ja datan monitorointi testin suorituksen aikana
5. Testin tulosten nopea läpikäynti testin suorituksen päätyttyä virheellisen testin tunnistamiseksi
6. Testin, testidatan, tulosten ja muun testin toistamiseen tarvittavan datan arkistointi
7. Aloitus- ja päätösaikojen, sekä tulosten kirjaaminen (Meier ym., 2009)

Ennen varsinaisen testauksen aloittamista on hyvä vielä varmistaa, että testiympäristön konfiguraatio vastaa suunniteltua konfiguraatiota, ja että sekä kuormitustyökalun että testiympäristön mittarit ovat konfiguroitu oikein. (Meier ym., 2009)

4.2.7 Tulosten analysointi ja raportointi

Testattavan sovelluksen sidosryhmille testin tulokset ovat sellaisenaan riittämättömät. Tarvitaan johtopäätöksiä ja koostettua dataa, joka tukee näitä johtopäätöksiä, sekä analyysejä, vertailuja ja yksityiskohtia datan keräämisestä. Suorituskykytestin tuloksia analysoidessa tulisi tutkia hyväksymiskriteereihin liitettyjen mittareiden tuloksia ja määrittää onko järjestelmän suorituskyky kehittymässä suorituskykytavoitteiden mukaisesti vai vastaisesti. Testin tuloksia analysoimalla voidaan myös löytää pullonkauloja, joiden korjaamisen jälkeen testi olisi syytä toistaa uudelleen. Testi on myös syytä toistaa silloin, kun testin tulokset eivät riitä selvittämään haluttuja suorituskykyominaisuuksia, jolloin asetettuja mittareita täytyy muuttaa uuden tiedon saamiseksi. (Meier ym., 2009)

4.3 Yhteenveto

Tässä luvussa on käyty läpi verkkosovelluksen suorituskykytestauksen eri tyyppejä ja aktiviteetteja Meierin ym. (2009) mukaan. Voidaan todeta, että verkkosovellusten suorituskykytestaus on monimuotoinen prosessi, mikä on ymmärrettävää, kun ottaa huomioon kuinka monimuotoinen joukko sovelluksia kuuluu verkkosovellus käsitteen alle. Samat suorituskykytestauksen aktiviteetit toistuvat kuitenkin verkkosovelluksesta riippumatta. Samoja suorituskyvyn mitta-

reita voi käyttää myös pilvipalveluna tarjottavan tietokantapalvelun suorituskyvyn mittaamiseen. Kun tietokantapalvelu tarjotaan verkon välityksellä sovellusalustana (DBaaS), on vasteajasta ja suoritustehosta johdetut mittarit ainoat sovellettavissa olevat suorituskyvyn ilmaisimet, koska laskentaresurssien käyttöä hallinnoi ja monitoroi pilvipalvelun tarjoaja. Kun tiedon hallinnan tarpeita tarkastellaan sovelluskohtaisesti, voidaan myös löytää käyttöskenaarioita, jotka voidaan ohjelmoida komentosarjoiksi ja lopulta kantakyselyiksi. Yksittäisestä käyttöskenaariosta voidaan myös ohjelmoida kunkin tietokannan kyselykielen mukainen komentosarja, jolloin yksittäisen käyttöskenaarion suorituskykyä voidaan vertailla eri kyselykielen omaavien tietokantojen kesken.

5 TUTKIMUSMENETELMÄ

Tässä luvussa esitetään tutkimukseen liittyvät aikaisemmat tutkimukset, kerrataan tutkimuksen tavoitteet, kuvataan suunnittelutiedettä ja kuvataan miksi ja miten suunnittelutiedettä käytetään tutkimuksen tutkimusmenetelmänä.

5.1 Liittyvät tutkimukset

Boicea ym. (2012) vertailivat NoSQL-dokumenttivarastoa, MongoDB:tä, ja relaatiotietokannan hallintajärjestelmää, Oracle Database SQL-tietokantaa. Tietokantojen suorituskykyä vertailtiin tietueiden lisäys-, päivitys- ja poisto-operaatioiden osalta, mutta ei haku-operaation osalta. Boicea ym. (2012) löydösten mukaan MongoDB-tietokanta suoriutui kyselyistä nopeammin, mutta erot olivat merkittäviä vasta kun kyselyt kohdistuivat 100 000 tietueeseen, jolloin vasteajat olivat kertaluokkaa suurempia. Tutkimuksessa ei kerrottu, millä laitteistolla tietokannan hallintajärjestelmiä ajettiin. (Boicea ym., 2012)

Floratou ym. (2012) vertailevat NoSQL-tietokantojen ja relationaalisten tietokantojen suorituskykyä nykyaikaisen tuotantotietokanta (”new OLTP”) ja päätöksentekojärjestelmien (Decision Support System) näkökulmasta. Tutkimuksessa vertailtiin MongoDB-dokumenttivaraston ja Microsoft SQL Server-tietokannan hallintajärjestelmän suorituskykyä 16 koneen klusteroidussa ympäristössä. Tietokantojen vertailuun käytettiin YCSB-vertailututkimusta (benchmark), joka keskittyy kyselyiden viiveeseen järjestelmää kuormittaessa. Lähes jokaisessa testissä SQL Server-tietokannan hallintajärjestelmä suoriutui MongoDB:tä nopeammin. Floratou ym. (2012) löydösten mukaan relaatiotietokannan hallintajärjestelmät tarjoavat vielä vastaavia NoSQL-tietokannan hallintajärjestelmiä parempaa suorituskykyä, vaikka NoSQL-vaihtoehdot ovatkin joissain tapauksissa kilpailukykyisiä. Tutkimuksessa nostetaan kuitenkin esille, että NoSQL-tietokannat keskittyvä eri ominaisuuksiin, kun relaatiotietokannan hallintajärjestelmät, kuten erilaisiin tietomalleihin, automaattiseen tiedon sirpalointiin ja kuorman tasaukseen ja vaihtoehtoisiin ristiriidattomuuden malleihin. (Floratou ym., 2012)

Li & Manoharan (2013) vertailevat NoSQL-tietokannan hallintajärjestelmiin ja relaatiotietokannan hallintajärjestelmiin toteutettujen avain-arvo-varastojen suorituskykyä. Vaikka NoSQL-tietokannan hallintajärjestelmät ovat yleensä optimoitu avain-arvo-varastoiksi, eivät kaikki NoSQL-tietokannan hallintajärjestelmät suoriutuneet relaatiotietokannan hallintajärjestelmiä paremmin Li & Manoharan (2013) tutkimuksessa. Esimerkiksi luku-operaatioissa NoSQL-tietokannoista ainoastaan MongoDB ja Couchbase olivat nopeampia kuin relationaalinen Microsoft SQL Server Express-tietokannan hallintajärjestelmä, kun sitä hitaampia NoSQL-tietokantoja olivat Hybertable, CouchDB, Cassandra ja RavenDB. Li & Manoharan (2013) eivät myöskään havainneet MongoDB:n ja Microsoft SQL Server Express-tietokannan hallintajärjestelmän välillä kertaluokkaa suurempia vasteaikoja 100 000 tietueen lisäyksessä, kuten Boicea ym. (2012) havaitsivat tut-

kimuksessaan vertaillessaan MongoDB ja Oracle SQL-tietokannan hallintajärjestelmiä. Tutkimuksessa ei oteta kantaa millä laitteistolla tietokannan hallintajärjestelmiä ajettiin. (Li & Manoharan, 2013)

Abotourabi ym. (2015) vertailevat MongoDB- ja Microsoft SQL Server-tietokannan hallintajärjestelmien suorituskykyä käyttäen verkkokaupankäyntiin tarkoitettua dataa. Tutkimuksessa MongoDB-tietokanta suoriutuu Microsoft SQL Server-tietokannan hallintajärjestelmää paremmin lukuun ottamatta koostekyselyissä, jossa relaationaalinen MS SQL Server-tietokannan hallintajärjestelmä suoriutui paremmin. Tietokannan hallintajärjestelmiä ajettiin yksittäisellä tietokoneella. (Abotourabi ym., 2015)

5.2 Tutkimuksen tavoitteet

NoSQL-tietokantakonseptin mukaisia tietokannan hallintajärjestelmiä kehitettiin vastaamaan pilvilaskennan haasteisiin. Kuitenkin NoSQL-tietokantoja on pääasiassa tutkittu perinteisissä palvelinympäristöissä pilviympäristön sijaan. Tämän tutkimuksen tavoitteena on tutkia moderneja pilviympäristöön suunniteltuja tietokannan hallintajärjestelmiä pilviympäristössä. Vertailevat tutkimukset ovat toistaiseksi vertailleet NoSQL-tietokantoja perinteisiin relaatiotietokannan hallintajärjestelmiin. Tässä tutkimuksessa NoSQL-tietokantakonseptin mukaista dokumenttivarastoa ei verrata perinteiseen relationaaliseen tietokantaan vaan NewSQL-tietokantakonseptin mukaiseen pilviympäristöön suunniteltuun tietokannan hallintajärjestelmään.

NoSQL-sateenvarjoon kuuluu eri tietomalleja tukevia tietokannan hallintajärjestelmiä, jolloin sopivan tietokannan hallintajärjestelmän valinta riippuu sovelluksen tarpeista. Tutkimuksen tiedonhallinnan näkökulmaksi on valittu verkkokaupan tuotekatalogi. Verkkokaupankäynti on yksi merkittävimmistä Internetin sovellusten aiheista, ja skaalautuvuus ovat tärkeitä tekijöitä menestyksekkääseen kaupankäyntiin verkossa (Hirai ym., 2001, Bochamann ym., 2002). Erityisesti oleellista verkkokauppasovelluksen suorituskyvyn kannalta on katalogitiedon saatavuus, koska 80 -prosenttia verkkokauppasovelluksen pyynnöistä liittyy tuotteiden hakuun (Bochamann ym., 2002).

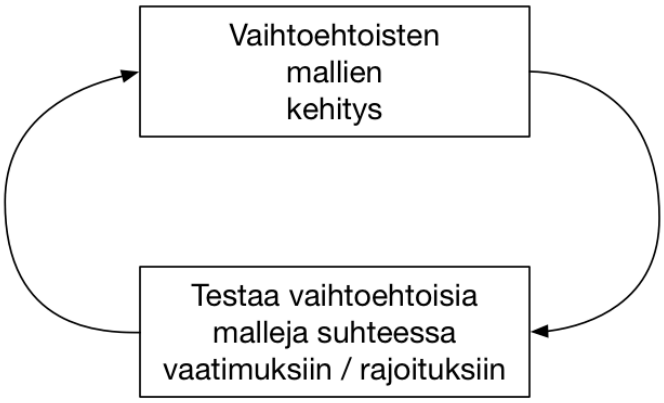
Tutkimuskysymykset voidaan esittää seuraavasti:

- Miten voidaan toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta?
- Miten Amazon Aurora-tietokantapalvelu suorituskyky vertautuu Amazon DynamoDB-tietokantapalvelun suorituskykyyn tuotekatalogin tietokantana?

5.3 Suunnittelutiede

Suunnittelutieteellisen tutkimuksen tuloksena syntyy artefakti, joka voi olla esimerkiksi konstruktio, malli, menetelmä tai ilmentymä. Artefakti on esitettävä ja kuvattava niin, että sitä voidaan arvioida ja verrata aikaisempiin samaan tarkoitukseen luotuihin artefakteihin. Artefaktin luonnilla voidaan osoittaa mitä on mahdollista toteuttaa. Suunnittelutieteen kriittinen laatu perustuu uusien teknisten kyvykkyyksien tunnistamiseen, mikä laajentaa tietojärjestelmien maailmaa. Löydökset ovat merkittäviä tietojärjestelmätutkimukselle ainoastaan, jos on epäilys kyvykkyydestä luoda kyseistä artefaktia, on epävarmuutta sen kyvykkyydestä suoriutua tarpeenmukaisesti ja automatisoitu tehtävä on tärkeä tietojärjestelmäyhteisölle. (Hevner, March, Park & Ram, 2004)

Suunnittelu on pohjimmiltaan iteratiivista ja inkrementaalista toimintaa ilman selkeitä päättymisen sääntöjä. Kehitetty artefakti on valmis, kun se täyttää ratkaistavan ongelman asettamat vaatimukset ja rajoitukset. Suunnittelun evaluointivaihe antaa oleellista palautetta kehitysvaiheelle suunnitteluprosessin ja –artefaktin laadusta. Suunnitteluprosessia voidaan kuvata kehitysvaiheen ja evaluointivaiheen kiertokulkuna. Kehitys/testaus –kiertokulku on kuvattuna kuviossa 5. (Hevner ym., 2004)



KUVIO 5 Kehitys/testaus-kiertokulku (Hevner ym., 2004 s. 14)

Evaluointi on tärkeä osa kehitysprosessia, koska suunnittelutieteiden päämäärä on tuottaa hyötyä. Liiketoimintaympäristö asettaa vaatimukset, joita vasten kehitettyä artefaktia evaluoidaan. Evaluointi edellyttää tarkoituksenmukaisten mittarien määrittämistä, ja datan keräämistä ja analysointia. Tietojärjestelmäartefakteja voidaan evaluoida funktionaalisuuden, kokonaisuuden, yhteneväisyyden, tarkkuuden, suorituskyvyn, luotettavuuden, käytettävyyden ja muiden relevanttien laadullisten ominaisuuksien mukaan. Kehitettyjen artefaktien tehokas evaluointi vaatii tietoa tutkimusmenetelmistä. Evaluointimenetelmän täytyy soveltua sekä kehitetyn artefaktin että asetettujen mittareiden osalta. (Hevner ym., 2004)

5.4 Tutkimuksen toteutus ja rakenne

Suunnittelutiede soveltuu tutkimuksen tutkimusmenetelmäksi, koska voidakseen vastata ensimmäiseen tutkimuskysymykseen, on kehitettävä uusi artefakti: pilviympäristössä skaalautuva tuotekatalogin tietokanta. Koska modernit pilviympäristöön suunnitellut tietokannan hallintajärjestelmät tarjoavat erilaisia tapoja tiedon mallintamiseen, on olemassa vaihtoehtoisia tapoja toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta. Tämä on otettu huomioon konstruktiivisessa tutkimuksessa, jossa luodaan kaksi artefaktia: NoSQL-tietokantakonseptin mukaiseen dokumenttivarastoon toteutettu tuotekatalogin tietokanta ja NewSQL-tietokantakonseptin mukaiseen relationaaliseen tietokantaan toteutettu tuotekatalogin tietokanta. Tämä herättää toisen tutkimuskysymyksen, jonka tarkoituksena on vertailla kahden eri tietomallin omaavan tietokannan suorituskykyä tuotekatalogin tietokantana. Tuotekatalogien suunnitteluvalinnat perustuvat kirjallisuuskatsauksen löydöksiin, tietokantapalvelukohtaiseen dokumentaatioon ja suunnitteluprosessin kehitysvaiheen löydöksiin ja niiden synnyttämiin iteraatioihin. Artefakteja evaluoidaan vertailemalla niiden suorituskykyä keskenään. Artefaktien suorituskyvyn evaluointiin käytetään kirjallisuuskatsauksessa esiteltyjä suorituskykytestauksen periaatteita.

Tutkielman rakenteessa sovelletaan Gregorin & Hevnerin (2013) esittämään suunnittelutieteellisten tutkimusten julkaisujen skeemaan. Ensimmäisessä luvussa esitettiin tutkimusongelma ja perusteltiin sen merkittävyys. Toinen, kolmas ja neljäs luku koostavat kirjallisuuskatsauksen tutkimuksen aiheeseen liittyen. Toisessa luvussa katsastettiin kirjallisuutta tuotekatalogin tiedonhallinnasta. Kolmannessa luvussa katsastettiin kirjallisuutta tietokannan hallintajärjestelmistä pilviympäristössä ja neljännessä luvussa katsastettiin kirjallisuutta verkkosovellusten suorituskykytestauksesta. Tässä luvussa esitettiin suunnitelutieteen tutkimusmenetelmä, ja kuinka sitä sovelletaan tietokantapalveluiden suorituskyvyn tutkimiseen tuotekatalogin tietokantana. Kuudennessa luvussa esitetään tutkimuksessa luodut tuotekatalogitoteutukset Amazon Aurora ja DynamoDB-tietokantapalveluihin, eli konstruktiivisen tutkimuksen kehitetyt artefaktit. Seitsemännessä luvussa kuvataan, miten luotuja tuotekatalogitoteutuksia evaluoidaan suorituskykytestauksen avulla ja tulkitaan evaluoinnin tuloksia. Viimeisessä kahdeksannessa luvussa esitetään pohdinta ja yhteenveto.

6 AWS AURORA JA DYNAMODB-TIETOKANTAPALVELUT TUOTEKATALOGIN TIETOKANTANA

Tässä luvussa esitetään miten tutkielmassa kehitetyt artefaktit, Amazon Aurora- ja DynamoDB-tietokantapalveluihin toteutetut tuotekatalogit, on toteutettu. Ensimmäiseksi kuvataan, miten tuotekatalogin tuotemassa on luotu. Tämä jälkeen kuvataan, miten tuotekatalogi on toteutettu kahteen AWS-pilvipalvelutarjoajan (Amazon Web Services) palveluna tarjoamaan tietokantatuotteeseen: Aurora- ja DynamoDB-tietokantapalveluun. Lopuksi esitetään yhteenveto.

6.1 Tuotekatalogi

Tuotekatalogin tuotemassan mallintamiseen on käytetty Logan, Humeau & Singh (2017) tarjoamaa datakokoelmaa, joka on tarkoitettu harjoitussyötteeksi järjestelmille, jotka tunnistavat tuoteattribuutteja kuvista. Kyseinen datakokoelma soveltuu hyvin testattavan tuotekatalogin tuotemassaksi, koska se sisältää riittävän laajan variaation attribuuteissa. Tuotekatalogin tuotemassa on Loganin ym. (2009) datakokoelman lisäksi rikastettu Googlen (2015) tuotetaksonomian mukaisilla kategorialiitoksilla.

Tuotekatalogin tuotemassan koko on mallinnettu todellisista verkkokaupoista. Taulukossa 1 on vertailtu tutkielman testikatalogin tuotteiden, attribuuttien, kategorioiden ja lapsikategorioiden määrää kolmen todellisen verkkokauppojen vastaaviin määriin. Lapsikategorialla tarkoitetaan tuotekatalogin kategoriahierarkian alinta kategoriatasoa. Testattava tuotekatalogi sisältää 1 000 000 tuotetta, 3 414 593 attribuuttia, 2 000 kategoriaa ja 1 701 lapsikategoriaa. Tuotemäärällisesti suurin tuotekatalogi vertailtavista todellisista verkkokaupoista on verkkokaupassa B, joka sisältää 11 788 786 tuotetta ja 88 370 103 attribuuttia. Kategoriamäärällisesti verkkokauppa B:n tuotekatalogi on pienin 814 kategorialla ja 626 lapsikategorialla. Eniten kategorioita on verkkokauppa C:n tuotekatalogissa, jossa on 5 355 kategoriaa ja 4 525 lapsikategoriaa. Vähiten tuotteita on verkkokauppa A:n tuotekatalogissa, jossa on 72 371 tuotetta ja 25 493 attribuuttia.

TAULUKKO 1 Tuotekatalogin tuotemassan vertailu

	Verkkokauppa A	**Verkkokauppa B**	**Verkkokauppa C**	**Testikatalogi**
Tuotteita	72 371	11 788 786	106 623	1 000 000
Attribuutteja	25 493	88 370 103	59 946	3 414 593
Kategorioita	1 201	814	5 355	2 000
Lapsikategorioita	1 108	626	4 525	1 701

Taulukossa 2 on kuvattu vertailuverkkokauppojen ja testikatalogin attribuuttien ja tuotteiden, tuotteiden ja kategorioiden, sekä kategorioiden ja lapsikategorioiden suhdelukuja. Kaikki kolme verkkokauppaa ovat eri toimialoilta, mikä näkyy myös näiden suhdelukujen vaihteluvälissä. Tästä voidaan nähdä, että tuotekatalogeja on hyvin erilaisia, riippuen toimialasta. Testikatalogin suhdeluvut ovat kuitenkin minimi- ja maksimiarvojen välissä, joten voidaan olettaa, että testikatalogi voisi vastata todellista verkkokaupan tuotekatalogia.

TAULUKKO 2 Tuotekatalogin suhdelukujen vertailu

	Verkkokauppa A	**Verkkokauppa B**	**Verkkokauppa C**	**Testikatalogi**
Attribuutteja /Tuote	0,35	7,50	0,56	3,41
Tuotteita /Kategoria	60	14482,54	19	500
Kategorioita /Lapsikategoria	1,08	1,30	1,18	1,18

6.2 AWS Aurora

Amazon Aurora on RDS tuoteperheeseen kuuluva pilviympäristöön rakennettu MySQL- ja PostgreSQL-yhteensopiva relaationaalinen tietokanta (Amazon Web Services, 2018a). AWS:n mukaan (2018b) Aurora on jopa viisi kertaa nopeampi kuin tavallinen MySQL-tietokanta ja kolme kertaa nopeampi kuin tavallinen PostgreSQL-tietokanta. Aurora on täysin ylläpidetty tietokantapalvelu, joka palvelitonta (Serverless) konfiguraatiota käytettäessä skaalautuu automaattisesti sovelluksen tarpeisiin (Amazon Web Services, 2018a).

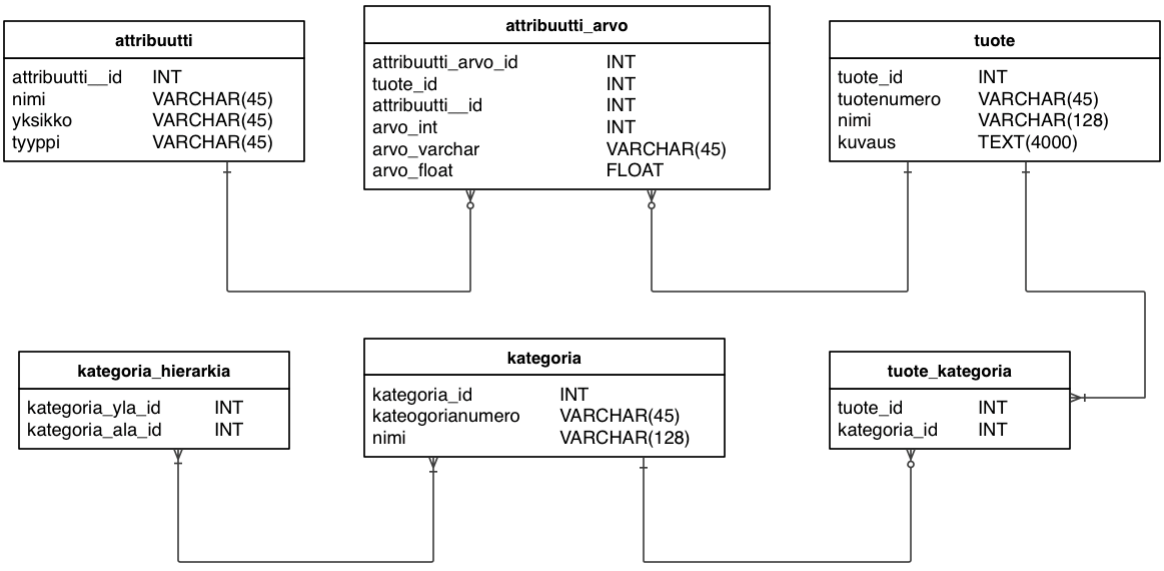
Aurora valittiin vertailtavaksi tietokantatuotteeksi, koska se pyrkii tarjoamaan pilviympäristössä skaalautuvaa relationaalista tietomallia. Luvussa 2 esitetyt tuotekatalogin tiedonhallinnan haasteet ja niihin kehitetyt ratkaisut, kuten vertikaaliset skeemat ja EAV-tietomallit, liittyvät erityisesti relationaalisiin tietokantoihin. Tarkoituksena on testata näiden ratkaisujen toimivuutta ja suorituskykyä modernissa NewSQL-tietokantatuotteessa, joka pyrkii skaalamaan relationaalista tietomallia pilviympäristössä.

6.2.1 Tietomalli

Testiympäristön Aurora tietokantaan toteutettu tuotekatalogi pohjautuu luvussa 2 esitettyyn kolmitauluiseen EAV-tietomalliin. EAV-taulua vastaa *attribuutti_arvo*-taulu, josta löytyy liitos *attribuutti*-tauluun ja *tuote*-tauluun, sekä attribuutin arvo, jolle on IBM:n WebSphere Commerce verkkokauppasovelluksen attribuutti tietomallia mukaillen varattu jokaiselle tietotyypille (kokonaisluvulle,

merkkijonolle ja desimaalille) oma sarake (IBM, 2018). IBM:n attribuuttimallista poiketen attribuutin metatieto *tyyppi* tallennetaan *attribuutti*-tauluun Jastrow & Preussin (2015) esittämän EAV-tietomallin mukaan. Testituotekatalogissa ylläpidetään vain kolmea erilaista attribuuttien tyyppiä, jotka vastaavat tietokannan tietotyyppejä. Tämä lisää tiedon toistuvuutta *tyyppi*-sarakkeessa, mutta attribuuttien kyselyissä tarvitaan yksi liitos vähemmän, kun tyyppi-metatieto on tallennettuna *attribuutti*-tauluun. Toinen tallennettava attribuutin metatieto on attribuutin yksikkö, joka tallennettaan *attribuutti*-taulun *yksikko*-sarakkeeseen. Tämä metatieto määrittää esimerkiksi onko kokonaislukuna esitetty pituus-attribuutin arvo millimetreinä vai senttimetreinä. Tyyppi ja yksikkö-metatietojen lisäksi *attribuutti*-tauluun on tallennettu attribuutin nimi. EAV-tietomallin entiteetti-taulua vastaa *tuote*-taulu. *Tuote*-tauluun on tallennettu ne tiedot, jotka eivät muutu usein: *nimi*, *tuotenumero* ja *kuvaus*.

EAV-tietomallin lisäksi tuotekatalogin tietomalli tukee kategoriahierarkian ja tuotteiden kategorialiitosten ylläpitämistä. *Kategoria*-tauluun tallennetaan kategorian nimi ja yksilöivä kategorianumero. Kategorioiden välistä hierarkiaa ylläpidetään *kategoria_hierarkia*-taulussa, jonka *kategoria_yla_id* ja *kategoria_ala_id*-sarakkeet kuvaavat kahden kategorian keskinäistä suhdetta ja toimivat samalla liitosavaimina *kategoria*-tauluun. Tuotteiden kategorialiitoksia ylläpidetään *tuote_kategoria*-taulussa, johon tallennetaan liitosavaimet *tuote*-tauluun ja *kategoria*-tauluun. Tuotekatalogin tietomalli kokonaisuudessaan on kuvattu kuviossa 6.



KUVIO 6 Aurora tietokannan tuotekatalogin tietomalli

Taulujen perusavaimet (primary key) ja yksilöivät hakemistot (unique index) on listattuna taulukossa 3.

TAULUKKO 3 Tietokantataulujen perusavaimet ja yksilöivät indeksit

Taulu	*Perusavain*	*Yksilöivä indeksi*
attribuutti	attribuutti_id	
attribuutti_arvo	attribuutti_arvo_id, tuote_id, attribuutti_id	
kategoria	kategoria_id	kategorianumero
kategoriahierarkia	ala_kategoria_id, yla_kategoria_id	
tuote	tuote_id	tuotenumero
tuote_kategoria	tuote_id, kategoria_id	

Tuotekatalogiin kohdistuvien tietokantahakujen tehostamiseksi tauluihin on asetettu hakuindeksejä. Suorituskykytestissä käytettävät tietokantakyselyt esitetään luvussa 7. Tauluihin asetetut hakuindeksit ovat listattuna taulukossa 4.

TAULUKKO 4 Tietokantataulujen perusavaimet ja yksilöivät hakemistot

Taulu	*Indeksit*	*Yhdistelmäindeksi*
attribuutti	nimi	
attribuutti_arvo	tuote_id, attribuutti_id, arvo_varchar	attribuutti_id, arvo_varchar
kategoria		
kategoriahierarkia	ala_kategoria_id	
tuote	tuote_id	
tuote_kategoria	tuote_id, kategoria_id	

Tietokantataulujen luontilauseet ovat kokonaisuudessaan esitettynä liitteessä 1.

6.2.2 Data

Tuotemassa on luettu tietokantaan sisään niin, että jokainen tiedoiltaan (nimi, yksikko, tyyppi) vastaava tuoteattribuutti on luettu *attribuutti*-tauluun vain kerran ja jokaisen attribuutin arvo ja tuoteliitos luettu *attribuutti_arvo*-tauluun. Tämän johdosta erilaisia attribuutteja, eli *attribuutti*-taulun tietuetta, on vain 2 709, kun taas attribuuttien arvoja on *attribuutti_arvo*-taulussa 3 414 593 tietuetta. Kaikkien taulujen tietueiden lukumäärä on esitettynä taulukossa 5.

TAULUKKO 5 Tietokantataulujen tietueiden lukumäärät

Taulu	*Tietueita*
attribuutti	2 709
attribuutti_arvo	3 414 593
kategoria	2 000
kategoriahierarkia	1 991
tuote	1 000 000
tuote_kategoria	1 000 000

Sisällöltään mielenkiintoisin taulu EAV-tietomallissa on attribuutin, attribuutin arvon ja entiteetin yhdistävä EAV-taulu, joka testiympäristön Aurora-tietokannan tietomallissa vastaa *attribuutti_arvo*-taulua. Koska kyseiseen tauluun on varattu attribuutin arvolle sarake kolmelle eri tietotyypille, on taulussa odotetusti paljon tyhjiä arvoja. *Attribuuti_arvo*-taulun tyhjät sekä uniikit arvot sarakkeittain on esitetty taulukossa 6.

TAULUKKO 6 *Attribuutti_arvo*-taulun arvoja

Kolumni	*Tyhjiä arvoja*	*Erilaisia arvoja*
attribuutti_arvo_id	0	3 414 593
tuote_id	0	1 000 000
attribuutti_id	0	2 709
arvo_int	2 857 342	1 015
arvo_varchar	597 875	12 432
arvo_float	3 373 969	233

Toinen taulu, joka sisältää odotetusti paljon tyhjiä arvoja on *attribuutti*-taulu, johon on varattu attribuutin nimen lisäksi sarakkeet attribuutin yksikkö ja tyyppi metatiedoille. Yksiköt on parsittu attribuuttien arvoista tunnistamalla numeerista arvoa seuraava merkkijono, joka on tulkittu yksiköksi. *Attribuutti*-taulun 2709:stä tietueesta 363 sisältää *yksikko*-arvon. Toinen huomioitava seikka on *tyyppi*-sarake, jossa voidaan havaita odotettua tiedon toistuvuutta. *Attribuuti*-taulun tyhjät sekä uniikit arvot kolumneittain on esitetty taulukossa 7.

TAULUKKO 7 *Attribuutti*-taulun arvoja

Kolumni	*Tyhjiä arvoja*	*Erilaisia arvoja*
attribuutti_id	0	2 709
nimi	0	2 114
yksikko	2346	195
tyyppi	0	3

EAV-tietomallin entiteetti-taulua vastaava *tuote*-taulu vastaa sisällöltään odotettua normalisoitua tilannetta, jossa taulut eivät sisällä tyhjiä arvoja, vaan paljon

erilaisia arvoja. Samaa voi sanoa kategoriahierarkian toteuttavista *kategoria*-, *kategoriahierarkia*- ja *tuote_kategoria*-tauluista. Taulujen tyhjät sekä uniikit arvot ovat listattuna taulukoissa 8,9,10 ja 11.

TAULUKKO 8 *Tuote*-taulun arvoja

Kolumni	*Tyhjiä arvoja*	*Erilaisia arvoja*
tuote_id	0	1 000 000
tuotenumero	0	1 000 000
nimi	0	795 851
kuvaus	0	785 243

TAULUKKO 9 *Kategoria*-taulun arvoja

Kolumni	*Tyhjiä arvoja*	*Erilaisia arvoja*
kategoria_id	0	2 000
kategorianumero	0	2 000
nimi	0	1 992

TAULUKKO 10 *Tuote_kategoria*-taulun arvoja

Kolumni	*Tyhjiä arvoja*	*Erilaisia arvoja*
tuote_id	0	1 000 000
kategoria_id	0	1 700

TAULUKKO 11 *Kategoriahierarkia*-taulun arvoja

Kolumni	*Tyhjiä arvoja*	*Erilaisia arvoja*
ala_kategoria_id	0	1991
yla_kategoria_id	0	299

6.2.3 Konfiguraatio

AWS:n RDS on täysin ylläpidetty tietokantapalvelu, jossa palvelun käyttäjän ei tarvitse huolehtia esimerkiksi palvelun laitteiston tai käyttöjärjestelmien konfiguraatiosta (Amazon Web Services, 2018b). AWS tarjoaa kuitenkin käyttäjälle konfigurointiasetuksia, jotka vaikuttavat esimerkiksi tietokannan yhteensopivuuteen ja skaalautuvuuteen. RDS tarjoaa perinteisiä tietokannan hallintajärjestelmiä, mutta tietokantapalvelun voi myös konfiguroida käyttämään modernimpaa automaattisesti skaalautuvaa palvelitonta arkkitehtuuria. Palvelitonta arkkitehtuuria tukee toistaiseksi vain Aurora MySQL tietokantamoottorin versio 5.6. Testiympäristön Aurora tietokanta on konfiguroitu käyttämään tätä tietokantamoottoria, jotta voidaan hyödyntää automaattisesti skaalautuvaa palvelitonta kapasiteettityypin asetusta. Palvelittoman kapasiteettityypin asetusta käytettäessä on asetettava skaalautuvuuden minimi ja maksimi kapasiteettiyksiköt, jotka

on testiympäristössä asetettu kahteen ja 64 kapasiteettiyksikköön. Testiympäristön Aurora-tietokantapalvelun konfiguraatio kokonaisuudessaan on esitetty taulukossa 12.

TAULUKKO 12 Aurora tietokantapalvelun konfiguraatio

Konfiguraatio	*Asetus*
Tietokantamoottori	Aurora MySQL
Versio	5.6.10a
Kapasiteettityyppi	Palveliton
Minimi kapasiteettiyksikkö	2 Aurora kapasiteettiyksikköä
Maksimi kapasiteettiyksikkö	64 Aurora kapasiteettiyksikköä

6.3 AWS DynamoDB

DynamoDB on avain-arvo- ja dokumenttivarasto, jonka luvataan horisontaalisen skaalautuvuuden avulla tukevan käytännöllisesti katsoen minkä tahansa kokoisia tauluja. DynamoDB on suunniteltu suorituskykyisille Internet-tason sovelluksille, jotka Amazonin (2018) mukaan ylikuormittaisivat perinteiset relaationaaliset tietokannat. DynamoDB on täysin ylläpidetty tietokantapalvelu, joka pystyy tarvittaessa skaalautua tukemaan petatavuja dataa ja kymmeniä miljoonia luku- ja kirjoitusoperaatioita sekunnissa. DynamoDB:ssä on joustava skeema, eli jokaisella rivillä voi olla milloin tahansa kuinka monta tahansa saraketta. (Amazon Web Services, 2018c)

DynamoDB valittiin toiseksi vertailtavaksi tietokantatuotteeksi edustamaan NoSQL-tietokantakonseptin mukaista dokumenttivarastoa. DynamoDB:n joustava skeema soveltuu hyvin tuotekatalogin tiedon hallinnan vaatimuksiin ja Amazon Web Services lupaa DynamoDB-tietokantapalvelulle todella korkeaa suorituskykykapasiteettia.

6.3.1 Tietomalli

NoSQL-tietokantojen suunnittelu poikkeaa relationaalisten tietokantojen suunnittelusta. Kun relationaalisessa tietokannassa voidaan luoda normalisoitu tietomalli pohtimatta, miten tietoa haetaan tietokannasta, DynamoDB-tietokannan tietomalli suunnitellaan siihen kohdistuvien kyselyiden pohjalta. Toinen tärkeä konsepti on pitää toisiinsa liittyvä data lähellä toisiaan. Sen sijaan, että toisiinsa liittyvät tietoalkiot hajautetaan useampaan tauluun, ne pitäisi pitää lähellä toisiaan. Yleisenä sääntönä voidaan pitää, että DynamoDB-sovelluksen pitäisi ylläpitää mahdollisimman vähän tauluja, ja parhaiten suunnitellut sovellukset vaativat ainoastaan yhden taulun. (Amazon Web Services, 2018d)

Testiympäristön DynamoDB-tietokantaan toteutettu tuotekatalogin tietomalli mukailee AWS:n (2016) DynamoDB-tietokantapalvelun suunnittelumalleja

käsittelevässä webinaarissa esitettyyn hierarkkisten tietorakenteiden tietomalliin. Tietomallissa attribuutit ja tuotteet on tallennettu samaan *tuotekatalogi*-tauluun. Taulun perusavain on yhdistelmäavain ositusavaimesta (partition key) ja järjestysavaimesta (sort key). Ositusavain on tuotteen tuotenumero ja järjestys avain on tyyppi-koodi, joka on tuote-tietueilla ”TUOTE” ja attribuutti-tietueilla attribuutin järjestysnumero ”ATT_”-etuliitteellä. Kategoria-liitos tallennettaan *kategoria*-kenttään, jota ylläpidetään sekä tuote- että attribuutti-tietueilla. *Tuotekatalogi*-taulu on esitetty kuviossa 7.

Perusavain		Attribuutit			
Ositusavain	Järjestysavain				
tuotenumero	tyyppi_koodi				
‘000001’	‘TUOTE’	kategoria	nimi	kuvaus	
		0123	‘Juomapullo’	‘Lorem…’	
‘000001’	‘ATT_1’	kategoria	att_nimi	arvo_s	
		0123	‘Väri’	‘Punainen’	
‘000001’	‘ATT_2’	kategoria	att_nimi	arvo_s	
		0123	‘Materiaali’	‘Muovi’	
‘000001’	‘ATT_3’	kategoria	att_nimi	arvo_n	yksikko
		0123	‘Tilavuus’	1	‘Litra’
‘000002’	‘TUOTE’	kategoria	nimi	kuvaus	
		3456	‘Muki’	‘Lorem…’	
‘000002’	‘ATT_1’	kategoria	att_nimi	arvo_s	
		3456	‘Väri’	‘Punainen’	
‘000003’	‘TUOTE’	kategoria	nimi	kuvaus	
		7890	‘Kahvipaketti’	‘Lorem…’	
‘000003’	‘ATT_1’	kategoria	att_nimi	arvo_n	yksikko
		7890	‘Paino’	500	‘grammaa’
‘000003’	‘ATT_2’	kategoria	att_nimi	arvo_s	
		7890	‘Paahto’	‘Tumma’	
‘000003’	‘ATT_3’	kategoria	att_nimi	arvo_s	
		7890	‘Alkuperämaa’	‘Kenia’	

KUVIO 7 DynamoDB-tietokantapalvelun *tuotekatalogi*-taulu

Tuotekatalogi-taulun lisäksi tietomalliin kuuluu myös *kategoria*-taulu, jossa ylläpidetään kategoriahierarkiaa. Kategoria-taulu on kuvattu kuviossa 8.

Perusavain		Attribuutit
Kategorianumero	nimi	
0123	‘Huonekalut’	
3456	‘Säilytys’	kategoria_yla
		0123
3457	‘Sohvat’	kategoria_yla
		0123
7890	‘Hyllyt’	kategoria_yla
		3456
8901	‘Vuodesohvat’	kategoria_yla
		3457

KUVIO 8 DynamoDB-tietokantapalvelun *kategoria*-taulu

6.3.2 Toisiohakemistot

DynamoDB-tietokantapalvelu tarjoaa nopean perusavaimen määrittämiseen perustuvan tiedon haun. Perusavaimeen kohdistuvan kyselyn lisäksi DynamoDB tarjoaa skannausoperaation, joka lukee kaikki taulun tai hakemiston tietueet ja ennen vastausta suodattaa niistä käyttäjän määrittelemän rajauksen mukaisesti. Skannausoperaatioita tulisi välttää suurien taulujen tapauksessa niiden hitauden vuoksi, sekä kyselyn kustannuksen vuoksi. Pelkästään yksi skannausoperaatio saattaa kuluttaa suuren taulun koko lukukapasiteetin. Taulut ja hakemistot tulisi suunnitella niin, että sovellus voi käyttää perusavaimeen kohdistuvaa kyselyä skannausoperaatioiden sijaan. (Amazon Web Services, 2019a)

Moni sovellus voisi hyötyä perusavaimen ulkopuolisiin attribuutteihin perustuvasta tehokkaasta tiedonsaannista. Tämän mahdollistamiseksi DynamoDB:n käyttäjä voi luoda tauluille yhden tai useamman toisiohakemiston, johon voi kohdistaa kyselyt varsinaisen taulun sijasta. Toisiohakemisto on tietorakenne, joka sisältää osajoukon taulun attribuuteista ja vaihtoehtoisen perusavaimen. DynamoDB tukee kahdenlaisia hakemistoja: globaaleja toisiohakemistoja (global secondary index) ja lokaaleja toisiohakemistoja (local secondary index). Globaalissa toisiohakemistossa ositusavain ja järjestysavain voivat molemmat poiketa kantataulun vastaavista, ja sitä pidetään globaalina, koska kyseiseen hakemistoon kohdistuvat kyselyt voivat kattaa kaiken kantataulun datan jokaiselta osiolta. Lokaalissa toisiohakemistossa ositusavaimen on vastattava kantataulun vastaavaa, ja sitä pidetään lokaalina, koska toisiohakemisto on ositettu kantataulun ositusten mukaisesti. Lokaalit hakemistot tukevat tiedon ristiriidattomuutta ja niitä suositellaan käytettäväksi ainoastaan tilanteissa, joissa kyselyiltä vaaditaan vahvaa ristiriidattomuutta. Muussa tapauksessa suositellaan käytettäväksi globaaleja toisiohakemistoja. (Amazon Web Services, 2019b)

Testiympäristön DynamoDB-tietokantapalvelun *tuotekatalogi*-taulusta on luotu kolme toisiohakemistoa. Toisiohakemistot on luotu luvussa 6 esitettyjen käyttötapausten pohjalta ja ne ovat kukin tyypiltään globaaleja toisiohakemistoja, koska tuotekatalogin tiedonhallinta ei aseta vaatimuksia tiedon ristiriidattomuudelle.

Kategoriakohtaisten tuote- ja tuoteattribuuttikyselyiden tukemiseksi on luotu *kategoriaTuotteetIndex*-toisiohakemisto. Hakemiston ositusavain on *kategoria*-attribuutti ja järjestysavain *tyyppi*-attribuutti, mikä mahdollistaa tietueiden haun kategorian perusteella ja rajaamaan tarvittaessa tietueen tyypillä. Hakemistoa on tarkoitus käyttää kategorioiden tuotelistauksessa. Hakemistoon on projektoitu kaikki *tuotekatalogi*-taulun attribuutit pois lukien *kuvaus*-attribuutti, joka on tarpeeton kategoriasivun tuotelistauksessa. *KategoriaTuotteetIndex*-toisiohakemisto on kuvattu kuviossa 9.

Perusavain		Projektoidut attribuutit			
Ositusavain	Järjestysavain				
kategoria	tyyppi				
0123	‘TUOTE’	tuotenumero	nimi		
		‘000001’	‘Juomapullo’		
0123	‘ATT_1’	tuotenumero	att_nimi	arvo_s	
		‘000001’	‘Väri’	‘Punainen’	
0123	‘ATT_2’	tuotenumero	att_nimi	arvo_s	
		‘000001’	‘Materiaali’	‘Muovi’	
0123	‘ATT_3’	tuotenumero	att_nimi	arvo_n	yksikko
		‘000001’	‘Tilavuus’	1	‘Litra’
3456	‘TUOTE’	tuotenumero	nimi		
		‘000002’	‘Muki’		
3456	‘ATT_1’	tuotenumero	att_nimi	arvo_s	
		‘000002’	‘Väri’	‘Punainen’	
7890	‘TUOTE’	tuotenumero	nimi		
		‘000003’	‘Kahvipaketti’		
7890	‘ATT_1’	tuotenumero	att_nimi	arvo_n	yksikko
		‘000003’	‘Paino’	500	‘grammaa’
7890	‘ATT_2’	tuotenumero	att_nimi	arvo_s	
		‘000003’	‘Paahto’	‘Tumma’	
7890	‘ATT_3’	tyyppi	att_nimi	arvo_s	
		‘000003’	‘Alkuperämaa’	‘Kenia’	

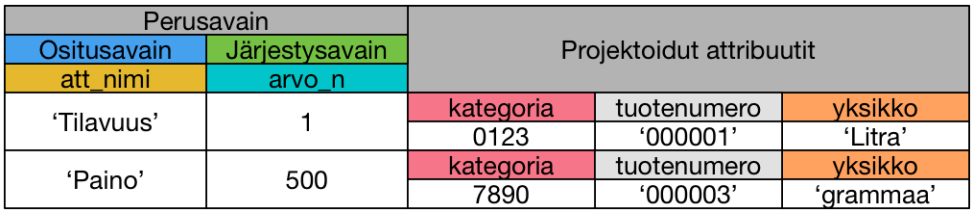
KUVIO 9 DynamoDB-tietokantapalvelun *kategoriaTuotteetIndex*-hakemisto

Tuoteattribuuttirajauksiin perustuvien kyselyiden tukemiseksi on luotu kaksi attribuutin arvon tietotyyppiin perustuvaa toisiohakemistoa. DynamoDB-taulujen attribuuttien tietotyyppi on määritettävä joko merkkijonoksi tai numeraaliseksi, mikäli attribuutti on osana perusavainta. Tästä johtuen *tuotekatalogi*-taulussa on tuoteattribuuttien arvoille luotu kaksi kenttää: *arvo_s* ja *arvo_n,* jotta tuoteattribuutin arvoa voidaan käyttää osana toisiohakemiston perusavainta. Tuoteattribuuttien toisiohakemistojen tarpeettomien tietueiden määrän rajoittamiseksi *tuotekatalogi*-taulun tuoteattribuuttien nimeä kuvaava kenttä on eri kuin tuotteen nimeä kuvaava kenttä. Tuoteattribuutin nimeä kuvaava *att_nimi*-kenttä on kummankin tuoteattribuuttien toisiohakemiston ositusavain. *AttribuutiSIndex*-toisohakemiston järjestysavain on *arvo_s*-kenttä, eli *attribuutiSIndex*-hakemiston tarkoitus on tarjota tehokas merkkijonopohjaisten tuoteattribuuttien tiedonsaanti. Hakemistoon on projektoitu *tuotenumero*-kentän lisäksi *kategoria*-kenttä, jotta attribuutteja voidaan tarvittaessa suodattaa kategorian mukaan. *AttribuutiSIndex*-toisiohakemisto on kuvattu kuviossa 10.

Perusavain		Projektoidut attribuutit	
Ositusavain	Järjestysavain		
att_nimi	arvo_s		
‘Väri’	‘Punainen’	kategoria	tuotenumero
		0123	‘000001’
‘Väri’	‘Punainen’	kategoria	tuotenumero
		3456	‘000002’
‘Materiaali’	‘Muovi’	kategoria	tuotenumero
		0123	‘000001’
‘Paahto’	‘Tumma’	kategoria	tuotenumero
		7890	“000003”
‘Alkuperämaa’	‘Kenia’	kategoria	tuotenumero
		7890	“000003”

KUVIO 10 DynamoDB-tietokantapalvelun *attribuutiSIndex*-hakemisto

AttribuutiNIndex-toisiohakemiston tarkoitus on tarjota tehokas numeraalisten tuoteattribuuttien tiedonsaanti, joten sen järjestysavain on *arvo_n*-kenttä. Hakemistoon on projektoitu *tuotenumero*-kentän ja *kategoria*-kentän lisäksi numeraalisiin tuoteattribuutteihin liittyvä *yksikko*-kenttä. *AttribuuttiNIndex*-toisiohakemisto on kuvattu kuviossa 11.



KUVIO 11 DynamoDB-tietokantapalvelun *attribuutiNIndex*-hakemisto

6.3.3 Data

DynamoDB-tietokantapalveluun on ajettu sisään täysin Aurora RDS-tietokantapalvelua vastaava tuotemassa. Koska tuotteet ja attribuutit tallennetaan samaan *tuotekatalogi*-tauluun, on sen tietueiden määrä tuotteiden ja attribuuttien yhteismäärä, eli 4 414 593 tietuetta. *KategoriaTuotteetIndex*-toisiohakemisto on käytännössä projektio samasta taulusta, joten sen tietueiden määrä on *tuotekatalogi*-taulua vastaava. *AttribuuttiSIndex*-toisiohakemistoon on projektoitu ainoastaan *arvo_s*-kentän sisältävät tietueet, ja *attribuuttiNIndex*-toisiohakemistoon on projektoitu ainoastaan *arvo_n*-kentän sisältävät tietueet. Tietueiden määrät ovat listattuna taulukossa 13.

TAULUKKO 13 DynamoDB-taulujen ja hakemistojen tietueiden lukumäärät

Taulu / hakemisto	*Tietueita*
tuotekatalogi	4 414 593
kategoriaTuotteetIndex	4 414 593
attribuuttiSIndex	2 816 923
attribuuttiNIndex	597 670
kategoria	2 000

6.3.4 Konfiguraatio

DynamoDB on täysin ylläpidetty tietokantapalvelu, jossa käyttäjä voi konfiguroida ainoastaan palvelun kapasiteettia. Käyttäjä voi valita kahdesta kapasiteettitilasta: provisioitu (provisioned) tai tilausperusteinen (on-demand). Tilausperusteista kapasiteettitilaa käytettäessä AWS on täysin vastuussa kapasiteetin skaalautuvuudesta ja palvelun laskutus perustuu kyselyiden määrään. Provisioitua kapasiteettitilaa käytettäessä käyttäjä määrittää luku- ja kirjoituskapasiteetti tasot. Konfigurointi tapahtuu määrittämällä taulu- ja indeksikohtaiset

luku- ja kirjoituskapasiteettiyksiköt (read/write capacity unit). Yksi lukukapasiteettiyksikkö vastaa yhtä ristiriidatonta kyselyä sekunnissa, tai kahta lopulta ristiriidatonta kyselyä sekunnissa 4 kilotavuun asti. Yksi kirjoituskapasiteettiyksikkö vastaa yhtä kirjoitusta sekunnissa 1 kilotavuun asti. Testiympäristön DynamoDB-tietokantapalvelu on kustannusten ennustamisen helpottamiseksi konfiguroitu käyttämään provisioitua kapasiteettitilaa. Koska suorituskykytestin tarkoitus on testata tiedon saatavuutta, eikä se sisällä kirjoitusoperaatioita, on kunkin taulun ja toisiohakemiston kirjoituskapasiteetti asetettu yhteen yksikköön. Taulujen ja toisiohakemistojen lukukapasiteettiyksiköt on arvioitu suorituskykytestin kuormituksen jakoa analysoimalla ja validoitu ajamalla niin sanottu savu-testaus, jossa suorituskykytestin toiminta validoijaan lyhyellä matalan kuormituksen testiajolla. Kapasiteettiyksiköiden tasot ovat listattuna taulukossa 14.

TAULUKKO 14 DynamoDB-taulujen ja hakemistojen kapasitettiyksiköt

Taulu / hakemisto	*Lukukapasiteettiyksiköitä*	*Kirjoituskapasiteettiyksiköitä*
tuotekatalogi	200	1
kategoriaTuotteetIndex	1600	1
attribuuttiSIndex	400	1
attribuuttiNIndex	200	1
kategoria	1	1

6.4 Yhteenveto

Tässä luvussa on esitetty, miten tutkimuksessa kehitetyt artefaktit, Amazon Aurora- ja DynamoDB-tietokantapalveluihin toteutetut tuotekatalogit, on toteutettu. Luvussa on esitetty, miten tuotekatalogia mallintava tuotemassa on luotu. Tämän jälkeen on kuvattu, miten tuotekatalogi on toteutettu Aurora- ja DynamoDB-tietokantapalveluihin.

Testattava tuotemassa ei ole kopio tai otanta varsinaisesta tuotekatalogista, mutta sen luonnissa on otettu huomioon todellisen tuotekatalogin tuotteiden, attribuuttien ja kategorioiden määrä. Kolmen verkkokaupan otannan perusteella verkkokauppojen tuotekatalogit ovat hyvin yksiköllisiä tuotteiden, attribuuttien ja kategorioiden määrien suhteen. Tutkielman tuotekatalogi ei ole keskiarvo näistä todellisista verkkokauppojen tuotekatalogeista, mutta sen tuotteiden, attribuuttien ja kategorioiden määrä, sekä keskinäinen suhdeluku, on vertailtavien tuotekatalogien maksimi ja minimi arvojen sisällä. Voidaan siis olettaa, että tutkielman tuotekatalogi tuotemassaltaan mallintaa riittävällä tasolla todellista tuotekatalogia.

Amazon Aurora-tietokantapalvelu valittiin edustamaan suorituskykyvertailun NewSQL-tietokantasovellusta, eli testaamaan relationaalisen tietomallin

skaalautuvuutta pilviympäristössä. Aurora-tietokantaan toteutettu tuotekatalogi noudattaa EAV-tietomallia, jota tyypillisesti käytetään tuotekatalogiin tiedon mallintamisessa relationaaliseen tietokantaan. Testattava Aurora-tietokantapalvelu on konfiguroitu käyttämään palvelitonta arkkitehtuuria, jossa Amazon pilvipalvelun tarjoajana hallinnoi tietokantapalvelun skaalautumisesta.

DynamoDB-tietokantapalvelu valittiin edustamaan suorituskykyvertailun NoSQL-tietokantasovellusta. DynamoDB on dokumenttivarasto, joka ominaisuuksiltaan pitäisi sopia hyvin tuotekatalogin tiedonhallinnan vaatimuksiin, eli tietomallin joustavuuteen ja tehokkaaseen tiedonsaantiin. DynamoDB-tietokantapalveluun toteutetun tuotekatalogin tietomalli myötäilee Amazonin dokumentaatiota hierarkkisen tiedon mallintamiseen DynamoDB-tietokantaan. Tiedon tehokasta saatavuutta on edistetty toisiohakemistojen avulla, jotka on luotu luvussa 7 esitettyjen käyttötapauksista johdettujen kantakyselyiden pohjalta.

Artefaktien luonnilla ollaan osittain vastattu ensimmäiseen tutkimuskysymykseen: Miten voidaan toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta? Luvussa on kuvattu, miten tuotekatalogi on toteutettu kahteen pilvipalveluna tarjottavaan tietokantapalveluun, jotka molemmat Amazonin mukaan tarjoavat skaalautuvuutta pilviympäristössä. Jotta voidaan todentaa toteutettujen tuotekatalogien skaalautuvuus pilviympäristössä, on niiden suorituskykyä testattava.

7 AWS AURORA JA DYNAMODB-TIETOKANTAPALVELUIDEN EVALUOINTI

Tässä luvussa esitetään kehitettyjen artefaktien evaluointi. Kehitettyjä artefakteja, Amazon Aurora ja DynamoDB-tietokantapalveluihin toteutettuja tuotekatalogeja, evaluoidaan suorituskykytestauksen avulla. Ensimmäiseksi kuvataan suorituskykytestin tavoitteet. Tämän jälkeen esitetään käyttötapaukset, johon suorituskykytesti perustuu, sekä miten suorituskykytestissä nostetaan kuormitusta ja simuloidaan satunnaisuutta. Tämä jälkeen esitetään suorituskykytestin tulokset molempien tietokantapalvelujen osalta, minkä jälkeen tuloksia vertaillaan keskenään. Lopuksi esitetään yhteenveto.

7.1 Tavoitteet

Suorituskykytestin tavoitteena on vertailla kahden pilvipalveluna tarjottavan tietokantatuotteen suorituskykyä tuotekatalogin tietokantana. Oleellista tietokantojen suorituskyvyn arvioinnissa on tuotekatalogin tiedon hallintaan kohdistuvat vaatimukset, jotka on esitetty luvussa 2. Koska tuotekatalogin data on staattista hallituin prosessein päivittyvää dataa, ei suorituskykytestissä oteta huomioon tapahtumanhallintaominaisuuksia, vaan keskitytään tiedon saatavuuden suorituskykyyn. Tavoitteena ei myöskään ole validoida jotain tiettyä kuormituksen tasoa, mitä tietokantapalvelut pystyvät tukemaan, vaan ainoastaan vertailemaan niiden keskinäisiä suorituskykyeroja.

7.1.1 Skaalautuvuus

Pilvilaskenta mahdollistaa illuusion ehtymättömistä tietokoneresursseista tarjoamalla lähes loputonta horisontaalista skaalautuvuutta (Armbrust ym., 2009). Suorituskykytestin tavoitteena ei ole testata tätä konseptia. Tavoitteena on kuitenkin havainnoida pilvipalveluna tarjottavien tietokantatuotteiden skaalautuvuutta. Skaalautuvuuden havainnoinnilla tarkoitetaan analysointia kuormituksen noston vaikutuksesta vasteaikojen kehittymiseen. Tavoitteena ei myöskään ole löytää sitä kuormituksen tasoa, jolla skaalautuvuuden esteeksi tulee pilvipalveluihin asetetut kapasiteettirajat. Kahden eri palvelun kapasiteettirajojen vertailu on vaikeaa, koska niiden kapasiteettiyksiköt eivät vastaa toisiaan eri palveluissa. Siksi ei ole mielekästä etsiä näitä keinotekoisesti asetettuja skaalautuvuuden rajoja, koska niiden vertailu ei kertoisi koko totuutta palveluiden skaalautuvuudesta tai kapasiteetista. Mikäli kuitenkin asetetut kapasiteettirajat tulevat suorituskykytestissä esille, on se otettava huomioon tuloksia analysoitaessa.

7.1.2 Mittarit

Meier ym. (2009) jakaa suorituskykyominaisuudet kolmeen luokkaan: vasteaika, suoritusteho ja resurssien käyttö. Ainoa tapa hallinnoida testiympäristön tietokantapalveluiden resurssien käyttöä on asetetut kapasiteettirajat. Nämä kapasiteettirajat eivät ole suorituskykytestauksen kohteena, mutta niiden konfiguroinnissa on otettu huomioon testauksen tavoitteeksi esitetty skaalautuvuuden havainnointi. Toisin sanoen, kapasiteettirajat on asetettu semmoiselle tasolle, jonka pitäisi mahdollistaa palveluiden skaalautuvuuden kuormaa kasvattaessa.

Koska pyritään havainnoimaan suorituskykyeroja, eikä todentamaan esimerkiksi kyvykkyyttä tietyn käyttäjäkuorman palvelemiseen, voidaan vasteaikaa ja suoritustehoa käyttää itsessään suorituskyvyn mittarina. Tavoitteena on löytää eroja tietokantapalvelujen vasteajassa kuormitusta kasvattamalla. Suoritustehoa säädellään kuormitusta kasvattamalla, eli lisäämällä tietokantakyselyiden määrää tietyllä aikavälillä. Tavoitteena ei ole määrittää maksimi suoritustehokapasiteettia, koska testiympäristössä sen määrittää palveluille asetetut kapasiteettirajat, jotka eivät ole tutkimuksen kohteena.

7.2 Käyttötapaukset

Suorituskykytestin käyttötapaukset ovat johdettu tutkimalla todellisen verkkokaupan käyttöliittymää ja HTTP-pyyntölokia. Tyypillisin käyttötapaus on tuotesivulle siirtyminen, joka on nimetty *KT1*-käyttötapaukseksi. Toinen yleinen tapa selata tuotekatalogia on kategoriapuun kautta kategoriasivulle siirtyminen, joka on nimetty *KT2*-käyttötapaukseksi. Tyypillinen tapa rajata tuotemassaa kategoriasivun tuotelistauksessa on suodattaa näkymää tuoteattribuuttirajauksella. Tämä käyttötapaus on nimetty *KT3*-käyttötapaukseksi. Käyttötapaukset ovat listattuna taulukossa 15.

TAULUKKO 15 Suorituskykytestin käyttötapaukset

Lyhenne	*Hakulauseke*	*Käyttötapaus*
KT1	Hae tuote X	Näytä tuotteen tiedot
KT2	Hae kaikki kategoriaan Y kuuluvat tuotteet ja attribuutit.	Hae kaikki ulkotakit
KT3	Hae kaikki kategoriaan Y kuuluvat tuotteet ja suodata yhden attribuutin mukaan.	Hae kaikki ulkotakit, jotka ovat kokoa L

Koska tietokantojen kyselykielet poikkeavat toisistaan, joudutaan suorituskykytestin kyselyt kirjoittamaan kummallekin tietokannalle erikseen. Kyselyt johdetaan samoista käyttötapauksista, joiden keskinäinen suhteellinen jakautuminen pysyy vakiona molemmissa suorituskykyajoissa.

7.2.1 Käyttötapausten keskinäinen jakautuminen

Suorituskykytesti pyrkii simuloimaan mahdollisimman tarkasti todellista tuotekatalogin tietokannan kohtaamaa kuormaa. Tässä oleellista on käyttötapausten keskinäinen suhteellinen jakautuminen mahdollisimman todenmukaisesti. Käyttötapausten suhteellisen jakautumisen määrittämisessä on vertailukohtana käytetty todellisen verkkokaupan yhden vuorokauden (30.9.2018) HTTP-pyyntölokeja. Pyyntölokeja analysoimalla voidaan todeta, että noin 52-prosenttia tuotekatalogiin kohdistuvista pyynnöistä oli tuotesivulle siirtymisiä, mikä vastaa suorituskykytestin käyttötapausta *KT1.* 18-prosenttia pyynnöistä oli kategoriasivulle siirtymisiä, mikä vastaa suorituskykytestin käyttötapausta *KT2* ja 18-prosenttia pyynnöistä oli hakutulossivulle siirtymisiä. Verkkokaupan hakutulossivu ja kategoriasivu molemmat mahdollistavat samankaltaisen käyttöliittymän tuotemassan rajaamiseen attribuuttien avulla ja noin 12-prosenttia tuotekatalogiin kohdistuvista pyynnöistä oli tuotemassan attribuuttirajauksia. Mielenkiintoista attribuuttirajauksien jakautumisessa oli, että vain kolmasosa näistä pyynnöistä tapahtui hakutulossivulla ja kaksi kolmasosaa attribuuttirajauksista tehtiin kategoriasivulla. Tietokantatoteutuksien kehitysvaiheessa kävi ilmi, että sekä DynamoDB että Aurora-tietokantapalvelu soveltuvat huonosti hakusanaan perustuvalle hauille. Pelkästään minuutin kuormitusajo pienellä kuormituksella hakusanakyselyjä johti molemmilla tietokantapalveluilla tilanteeseen, jossa yli 90-prosenttia kyselyistä kesti yli 30 sekuntia, mitä voidaan pitää tässä käyttötarkoituksessa kestämättömänä. Verkkokauppasovelluksissa tämä ominaisuus on usein toteutettu sille tarkoitetun erillisen hakumoottorin avulla, joten suorituskykytestissä ei oteta huomioon hakusanaan perustuvia rajauksia (IBM, 2019, Magento, 2019). Kun tämä 18-prosentin osuus hakusanakyselyistä ja neljän prosentin osuus attribuuttirajauksista hakutulossivulla jätetään huomioimatta, jää *KT1*-käyttötapauksen suhteelliseksi osuudeksi 67-prosenttia, *KT2*-käyttötapauksen suhteelliseksi osuudeksi 23-prosenttia ja *KT3*-käyttötapauksen suhteelliseksi osuudeksi 10-prosenttia. Suorituskykytestin käyttötapausten keskinäisen suhteellisen kuormituksen jakautuminen on listattuna taulukossa 16.

TAULUKKO 16 Käyttötapausten kuormituksen jakautuminen

Käyttotapaus	*Suhteellinen kuormitus*
KT1	67 %
KT2	23 %
KT3	10 %
Yhteensä	100 %

7.2.2 DynamoDB tietokantakyselyt

AWS suosittelee valmiiden kielikohtaisten ohjelmistokehityspakin (Software Development Kit, SDK) käyttöä pyyntöjen lähettämiseen DynamoDB-tietokantapalvelun sovellusliittymään (Application Programming Interface, API) (Amazon

Web Services, 2018e). Jmeter on Java-pohjainen kuormitustyökalu, joten suorituskykytestin DynamoDB-kyselyt on toteutettu AWS:n Java SDK (V 1.11.460) ohjelmistokehityspakkia käyttäen.

KT1-käyttötapauksen kyselyssä haetaan tuotteen tiedot *tuotekatalogi*-taulusta ositusavainta, eli tuotenumeroa käyttäen. Näin yhdellä kyselyllä saadaan sekä tuote- että attribuutti-tietueet.

KT2-käyttötapauksen kyselyt on jaettu kahteen osaan. Ensimmäisessä *KT2a*-kyselyssä haetaan *kategoriaTuotteetIndex*-indeksistä *kategoria*-ositusavainta käyttäen tiettyyn kategoriaan kuuluvat tuotteet. Järjestysavaimen, eli *tyyppi*-kentän avulla voidaan rajata, että ensimmäisessä kyselyssä haetaan pelkästään tuotteita, jolloin kyselyn kuormitusta voidaan säädellä rajaamalla palautettujen tietueiden, eli tuotteiden määrää. Suorituskykytestin *KT2a*-kyselyssä tuotteiden määrä on rajattu 25 tuotteeseen. Suorituskykytestissä ei ole toteutettu tuotemassan sivutusta, jonka sovellusliittymä kuitenkin mahdollistaisi. *KT2b*-kyselyssä haetaan kaikki kategoriaan liittyvät attribuutit. *KT2b*-kysely vastaa muutoin *KT2a*-kyselyä, mutta järjestysavaimella on rajattu, että *kategoriaTuotteetIndex*-indeksistä haetaan attribuutit. Haettavien attribuuttien määrä ei ole rajattu, jotta käyttäjälle voidaan luoda näkymä, josta voidaan valita kategorian kaikki mahdolliset attribuuttirajaukset.

KT3-kysely nähdään *KT2*-käyttötapauksen kyselyiden jatkumona. DynamoDB-tietokantapalvelu tarjoaa perinteisiä relationaalisia tietokantoja rajallisemmat kyselymahdollisuudet, minkä takia esimerkiksi taulu-liitokset joudutaan tekemään sovellustasolla. Tästä johtuen esimerkiksi useamman attribuutin rajaukset ovat käytännössä perättäisiä tietokantakyselyitä, jotka yhdistetään sovellustasolla. Tästä johtuen suorituskykytestin *KT3*-kysely käsittelee käytännössä yhtä attribuuttia ja palauttaa listan tuotenumeroita, jonka avulla rajataan *KT2*-vaiheen kategorianäkymää sovellustasolla. *KT3*-kysely kohdistuu attribuutin tyypistä riippuen joko *attriibuuttiNIndex*-indeksiin, mikäli attribuutin arvo on numeerinen tai *attribuuttiSIndexiin*, mikäli attribuutin arvo on merkkijono. Kyselyssä attribuutit ja niihin liittyvät tuotenumerot haetaan käyttäen indeksin ositusavainta, *att_nimi*-kenttää, ja järjestysavainta, a*rvo_s-/arvo_n*-kenttää. Kaikkien DynamoDB-kyselyiden Java-koodipalaset löytyvät liitteestä 2.

7.2.3 Aurora tietokantakyselyt

Aurora-tietokantapalvelun tietokantakyselyt ovat perinteisiä SQL-kyselyitä. Vaikka relationaalinen tietomalli ja SQL-kyselyt mahdollistaa rikkaampien tietokantakyselyiden luomisen, on suorituskykytestissä tehty samat oletukset sovellustason kyvykkyydelle liittää tietoa useammasta perättäisestä tietokantakyselystä.

KT1-käyttötapauksen kyselyt on jaettu kahteen osaan, jotta sekä tuotteen, että sen attribuutit saadaan noudettua. *KT1a*-kyselyssä haetaan *tuote*-taulusta käyttäen *tuotenumero*-rajausta ja *KT1b*-kyselyssä haetaan tuotteen attribuutit *tuotenumero*-rajauksella liittämällä *tuote*, *attribuutti_arvo* ja *attribuutti*-taulut.

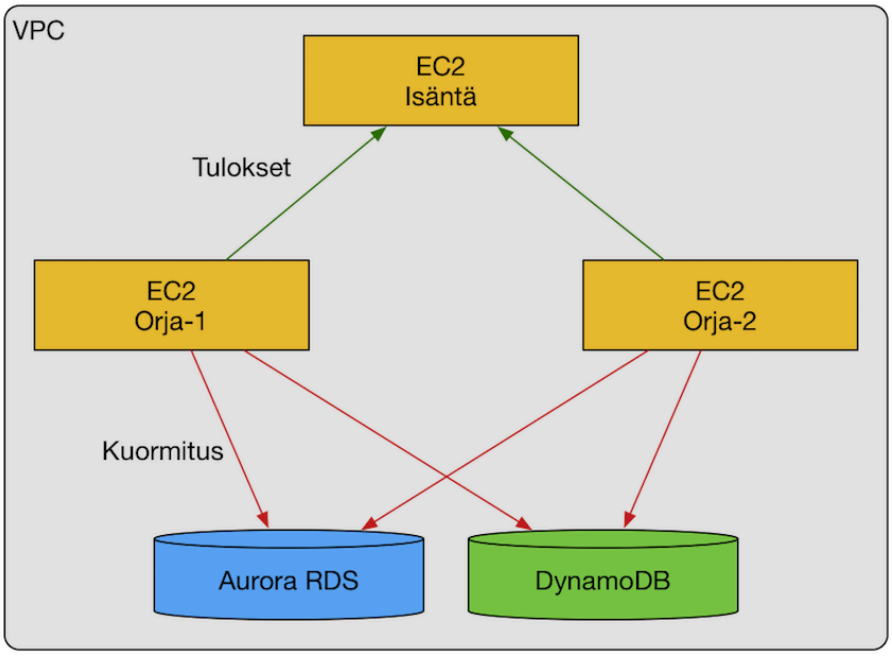
KT2-käyttötapauksen kyselyt on myös jaettu kahteen osaan ja ne myötäilevät vastaavia DynamoDB-kyselyitä. *KT2a*-kyselyssä haetaan kaikki kategorian

tuotteet *kategorianumero*-rajauksella liittämällä *tuote*, *tuote_kategoria* ja *kategoria*-taulut ja kyselyssä on rajaus 25 ensimmäiselle tietueelle. *KT2b*-kyselyssä haetaan kaikki kategorian attribuutit *kategorianumero*-rajauksella liittämällä *attribuutti*, *attribuutti_arvo*, *tuote*, *tuote_kategoria* ja *kategoria*-taulut.

KT3-käyttötapauksen kyselyssä tehdään sama oletus sovellustason kyvykkyydestä suodattaa *KT2*-vaiheen tuotteita ja kysely palauttaa vain listan tuotenumeroita. *KT3*-kysely vastaakin rajauksilta ja liitoksiltaan *KT2a*-kyselyä, jossa on vielä sisäkkäin kysely *attribuutti_arvo*-tauluun, mikä rajaa tuotteita attribuutin mukaan. Esimerkit kaikista SQL-kyselyistä löytyy liitteestä 3.

7.3 Kuormitussuunnitelma

Suorituskykytesti ajetaan AWS:n Frankfurtin (eu-central-1) konesaliin konfiguroidussa yksityisessä pilviympäristössä (Virtual Private Cloud, VPC). Molemmat testattavat tietokantapalvelut ovat konfiguroitu samaan yksityiseen pilveen, ja kuormituksen mallintamiseksi samaan yksityiseen pilveen on konfiguroitu kolme EC2-virtuaalikonetta. EC2-virtuaalikoneet mallintavat verkkosovelluksen kolmitasoisen-arkkitehtuurin sovellustasoa, joka käyttää tiedonhallintaan samassa suljetussa verkossa olevaa tietokantaa. EC2-virtuaalikoneisiin on asennettu Jmeter (v5.0.0) kuormitustyökalu hajautetulla konfiguraatiolla, jossa yksi koneista toimii isäntä-koneena, joka orkestroi suorituskykytestin ajoa kahden orjakoneina toimivien EC-virtuaalikoneiden kautta. Isäntä-kone lähettää kuormitussuunnitelman orja-koneille, jotka toteuttavat kuormitussuunnitelman kuormittamalla testattavaa tietokantapalvelua. Orja-koneet lähettävät suorituskykytestin tulokset isäntä-koneelle, joka kokoaa tulokset yhteen. Testiympäristö on kokonaisuudessaan kuvattu kuviossa 12.



KUVIO 12 Testiympäristö

Kuormituksen mallintamiseen on käytetty hajautettua Jmeter-konfiguraatiota kahdella orja-koneella, jotta riittävän korkeaa kuormitusta voidaan mallintaa. Kukin EC2-virtuaalikone perustuu AWS:n t2.medium-virtuaalikoneinstanssiin, joka sisältää kaksi virtuaalista prosessoriydintä ja 8 gigatavua keskusmuistia. EC2-virtuaalikoneiden konfiguraation on esitetty taulukossa 17.

TAULUKKO 17 EC2-tietokoneiden konfiguraatio

EC2-virtuaalikoneiden konfiguraatio	
instanssi	t2.medium
alusta	Amazon Linux
vCPUs	2
RAM (GiB)	8.0
alue	eu-central-1

7.3.1 Kuormituksen nosto

Jmeter-kuormitustyökalulla kuormitusta kasvatetaan lisäämällä säikeiden määrää. Suorituskykytesti aloitetaan 20 säikeellä per orja-instanssi, eli yhteensä 40 säikeellä. Säikeiden määrää nostetaan vaiheittain kaksinkertaistamalla säikeiden määrä neljä kertaa, jolloin korkein kuormituksen taso on 320 säiettä per orja-instanssi, eli yhteensä 640 säiettä. Kuormitusta nostetaan lisäämällä kymmenen sekunnin aikana 40 säiettä, kunnes ollaan saavutettu haluttu kuormituksen taso, jota ajetaan yhteensä kymmenen minuuttia. Jokaisen kuormitusvaiheen välissä on minuutin jäähdytysvaihe, jossa säikeet ajetaan alas.

7.3.2 Satunnaisuuden simuloiminen

Jotta suorituskykytesti simuloisi mahdollisimman tarkasti oikeaa kuormitusta, on kyselyiden suorittamiseen mallinnettu satunnaisuutta. Satunnaisuuden mallintaminen on otettu huomioon tuotteiden sisään luvussa tietokantoihin. Koska tuotenumero on tuotteiden sisään luvussa inkrementoituva järjestysnumero, voidaan *KT1*-käyttötapauksen tuotenumero arpoa jokaisen säikeen alussa 0 ja 1 000 000 väliltä. Kaikki lapsikategoriat, joilla on tuoteliitoksia, on tallennettu tuotekatalogien sisäänluvussa omaan tiedostoon, josta luetaan jokaiselle säikeelle satunnainen kategoria *KT2*-käyttötapauksen kyselyihin. *KT3*-kyselyt tapahtuvat aina *KT2b*-kyselyn jälkeen ja *KT3*-kyselyssä käytettävä attribuutti on satunnainen *KT2b*-kyselyn palauttama attribuutti.

7.4 Tulokset

Seuraavaksi esitetään suorituskykytestin tulokset. Ensimmäisenä esitetään Aurora-tietokantapalvelun suorituskykytulokset, jonka jälkeen esitetään DynamoDB-tietokantapalvelun suorituskykytulokset. Tämän jälkeen vertaillaan suorituskykytestien tuloksia.

7.4.1 Aurora suorituskykytulokset

Aurora-tietokantapalvelun suorituskykyä testaava testi ajettiin 29.12.2018. Taulukossa 18 on esitetty kuormitustasokohtaisesti kyselyiden vasteaikojen keskiarvo, vasteaikojen mediaani, suoritusteho ja virheprosentti. Tuloksista voidaan havaita skaalautuvuutta kuormituksen noustessa 320 säikeestä 640 säikeeseen, jolloin vasteaikojen mediaani tippuu 89 millisekunnista 19 millisekuntiin. Myöskin virheprosentti tippuu 10.91-prosentista 0.02-prosenttiin. 320 säikeen kuormitustaso on ensimmäinen taso, jossa kyselyt päätyvät virheisiin. Kaikki suorituskykytestissä havaitut virheet liittyivät tietokantayhteysvarannon ehtymiseen, jolloin Aurora-tietokantapalvelu hylkäsi yhteyden. Kuormitustason noustessa 640 säikeeseen Aurora-tietokantapalvelu salli kuitenkin lähes kaikki yhteydet. Tästä voidaan tehdä oletus, että palvelittoman konfiguraation Aurora-tietokantapalvelu skaalaa tietokantayhteysvarantoa automaattisesti kuormituksen kasvaessa. AWS:n dokumentaatiosta ei löytynyt vahvistusta tälle oletukselle, mutta löytyi parametri, jolla määritellään palvelininstanssitasolla tietokantayhteyksien maksimimäärä Aurora-tietokannalle, joka käyttää perinteistä esimääritettyä palvelin-konfiguraatiota (Amazon Web Services, 2019c).

TAULUKKO 18 Aurora tuloksia

Säikeiden määrä	Vasteajan keskiarvo (ms)	Vasteajan mediaani (ms)	Suoritusteho	Virheprosentti
40	34	4	556.1/sek	0.00%
80	70	4	480.4/sek	0.00%
160	344	22	150.6/sek	0.00%
320	497	89	145.2/sek	10.91%
640	600	19	123.3/sek	0.02%

Kun suorituskykytestin tuloksia tarkastellaan kyselykohtaisesti, voidaan havaita, että selkeästi raskain kysely on *KT2b*, jonka vasteajan mediaani on 262 millisekuntia. Seuraavaksi raskain kysely on *KT3*, jonka vasteajan mediaani on 15 millisekuntia. Loput käyttötapaukset jäävät alle kymmenen millisekuntiin ja kaikkien kyselyiden vasteajan mediaani on kuusi millisekuntia. Taulukossa 19 on esitetty kyselykohtaisesti kyselyiden vasteaikojen keskiarvo, vasteaikojen mediaani, suoritusteho ja virheprosentti.

TAULUKKO 19 Aurora tuloksia kyselykohtaisesti

Kysely	Vasteajan keskiarvo (ms)	Vasteajan mediaani (ms)	Suoritusteho	Virheprosentti
KT1a	87	4	94.1/sek	1.13%
KT1b	76	3	94.8/sek	1.13%
KT2a	131	8	32.4/sek	1.13%
KT2b	790	262	32.5/sek	1.14%
KT3	69	15	14.3/sek	0.42%
Yhteensä	173	6	267.4/sek	1.09%

7.4.2 DynamoDB suorituskykytylokset

Amazon DynamoDB-tietokantapalvelun suorituskykyä testaava testi ajettiin 29.12.2018. Taulukossa 20 on esitetty kuormitustasokohtaisesti kyselyiden vasteaikojen keskiarvo, vasteaikojen mediaani, suoritusteho ja virheprosentti. Myös DynamoDB-tietokantapalvelun tuloksista voidaan havaita skaalautuvuutta kuormituksen noustessa 320 säikeestä 640 säikeeseen, jolloin vasteaikojen mediaani tippui 208 millisekunnista 125 millisekuntiin. Tuloksista voidaan myös havaita, että kyselyt päätyivät virheisiin ainoastaan 640-säikeen kuormitustasolla ja tällöinkin ainoastaan 0.02-prosenttia kyselyistä. Toisaalta mediaanista poiketen vasteajan keskiarvot nousivat huomattavasti kuormituksen kasvaessa, mikä johtui siitä, että osassa kyselyistä vasteaika kasvoi huomattavan korkeaksi. 90-prosenttia kyselyistä jäi vasteaika alle 10 307 millisekuntiin, mutta rajan noustessa 99-prosenttiin kasvaa vasteaika 46 502 millisekuntiin.

TAULUKKO 20 DynamoDB tuloksia

Säikeiden määrä	Vasteajan keskiarvo (ms)	Vasteajan mediaani (ms)	Suoritusteho	Virheprosentti
40	55	6	288.4/sek	0.00%
80	575	40	48.9/sek	0.00%
160	1158	91	45.0/sek	0.00%
320	1852	208	41.9/sek	0.00%
640	3483	125	50.4/sek	0.02%

Kun suorituskykytestin tuloksia tarkastellaan kyselykohtaisesti, voidaan havaita, että myös DynamoDB-tietokantapalvelun tapauksessa selkeästi raskain kysely on *KT2b,* jonka vasteajan mediaani on 213 millisekuntia. Seuraavaksi raskain kysely on *KT3*, jonka vasteajan mediaani on 33 millisekuntia. Loput käyttötapaukset jäävät alle kymmeneen millisekuntiin ja kaikkien kyselyiden vasteajan

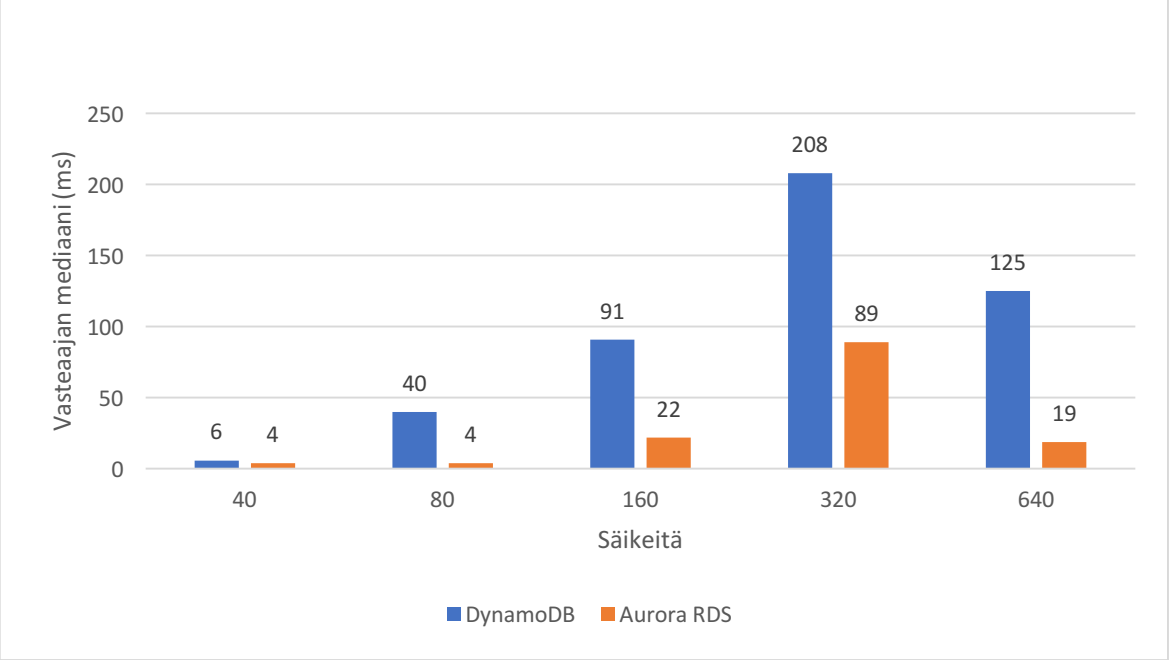
mediaani on 16 millisekuntia. Taulukossa 21 on esitetty kyselykohtaisesti kyselyiden vasteaikojen keskiarvo, vasteaikojen mediaani, suoritusteho ja virheprosentti.

TAULUKKO 21 DynamoDB tuloksia kyselykohtaisesti

Kysely	Vasteajan keskiarvo (ms)	Vasteajan mediaani (ms)	Suoritusteho	Virheprosentti
KT1	257	6	47.4/sek	0.00%
KT2a	547	8	16.3/sek	0.00%
KT2b	2343	213	16.2/sek	0.00%
KT3	767	33	7.2/sek	0.02%
Yhteensä	742	16	87.0/sek	0.00%

7.4.3 Tietokantapalveluiden suorituskykytulosten vertailu

Suorituskykytestin kyselyiden mediaaniaikoja tietokonepalveluittain vertaillessa voidaan todeta, että Aurora-tietokantapalvelun tietokantakyselyiden mediaani jää jokaisella kuormitustasolla DynamoDB-tietokantapalvelun kyselyiden mediaania pienemmäksi. Pienimmällä 40 säikeen kuormitustasolla molempien tietokantapalveluiden kyselyiden vasteajan mediaani jäi alle kymmeneen millisekuntiin, mutta jo seuraavalla kuormitustasolla voidaan nähdä selkeä ero vasteajoissa, kun DynamoDB-tietokantapalvelun vasteajan mediaani oli 40 ms ja Aurora-tietokantapalvelun vasteaikojen mediaani oli neljä millisekuntia. Tietokantapalveluiden vasteaikojen mediaanien suhteellinen ero kuitenkin tasoittuu seuraavilla kuormitustasoilla. Molempien tietokantapalveluiden korkeimmat vasteajat ilmenivät 320 säikeen kuormitustasolla, jolloin DynamoDB-tietokantapalvelun vasteajan mediaani oli 208 millisekuntia ja Aurora-tietokantapalvelun vasteajan mediaani oli 89 millisekuntia. Molempien tietokantapalveluiden kohdalla voidaan olettaa tapahtuneen pilvilaskennalle ominaista automaattista skaalautumista kuormituksen kasvaessa, koska molempien tietokantapalveluiden vasteaikojen mediaani laski huomattavasti korkeimmalla 640 säikeen kuormitustasolla, jolloin DynamoDB-tietokantapalvelun vasteaikojen mediaani oli 125 millisekuntia ja Aurora RDS-tietokantapalvelun vasteaikojen mediaani oli 19 millisekuntia. Molempien tietokantapalvelujen vasteaikojen mediaani kuormitustasoittain on kuvattu kuviossa 13.



KUVIO 13 Kyselyiden vasteaikojen vertailu

Tietokantapalveluiden vasteaikoja vertaillessa on otettava huomioon, että samojen käyttötapausten suorittamiseksi Aurora-tietokantapalveluun jouduttiin suorittamaan enemmän tietokantakyselyitä kuin DynamoDB-tietokantapalveluun. *KT1*-käyttötapaus on Aurora-tietokantapalvelun yhteydessä jaettu kahteen tietokantakyselyyn, kun DynamoDB-tietokantapalvelusta käyttötapauksen tyydyttävät tiedot saadaan yhdellä tietokantakyselyllä. Jotta tietokantapalveluiden suorituskykyä tuotekatalogin tietokantana voidaan verrata keskenään, on verrattava kyselyiden vasteaikojen sijaan käyttötapausten vasteaikoja, toisin sanoen koostaa kyselyjen vasteajat käyttötapauskohtaisesti ja laskea keskiarvokäyttötapauksen vasteajan mediaani ottaen huomioon käyttötapausten keskinäinen jakautuminen. Taulukossa 22 on listattuna DynamoDB-tietokantapalvelun vasteaikoja käyttötapauskohtaisesti ja taulukossa 23 on listattuna Aurora-tietokantapalvelun vasteaikoja käyttötapauskohtaisesti. Kummassakin taulukossa vasteajat on listattu kuormitustasoittain ja taulukkoon on laskettuna keskiarvovasteaika käyttötapaukselle, missä on otettu huomioon käyttötapaisten keskinäinen jakautuminen, eli *KT1*-käyttötapauksen painotus on 67-prosenttia, *KT2*-käyttötapauksen 23-prosenttia ja *KT3*-käyttötapauksen painotus on 10-prosenttia. Tuloksista voidaan huomata, että yleisimmän *KT1*-käyttötapauksen yhteydessä DynamoDB-tietokantakapalvelu hyötyy tietomallista, joka mahdollistaa tarvittavan tiedon hakemisen yhdellä kyselyllä Aurora-tietokantapalvelun kahden kyselyn sijaan. *KT1*-käyttötapaus suoriutui DynamoDB-tietokantapalvelua vasten nopeammin kolmella kuormitustasolla viidestä. Kuitenkin kullakin kuormitustasolla sekä *KT2* että *KT3* käyttötapaukset suoriutuivat nopeammin Aurora-tietokantapalvelua vasten.

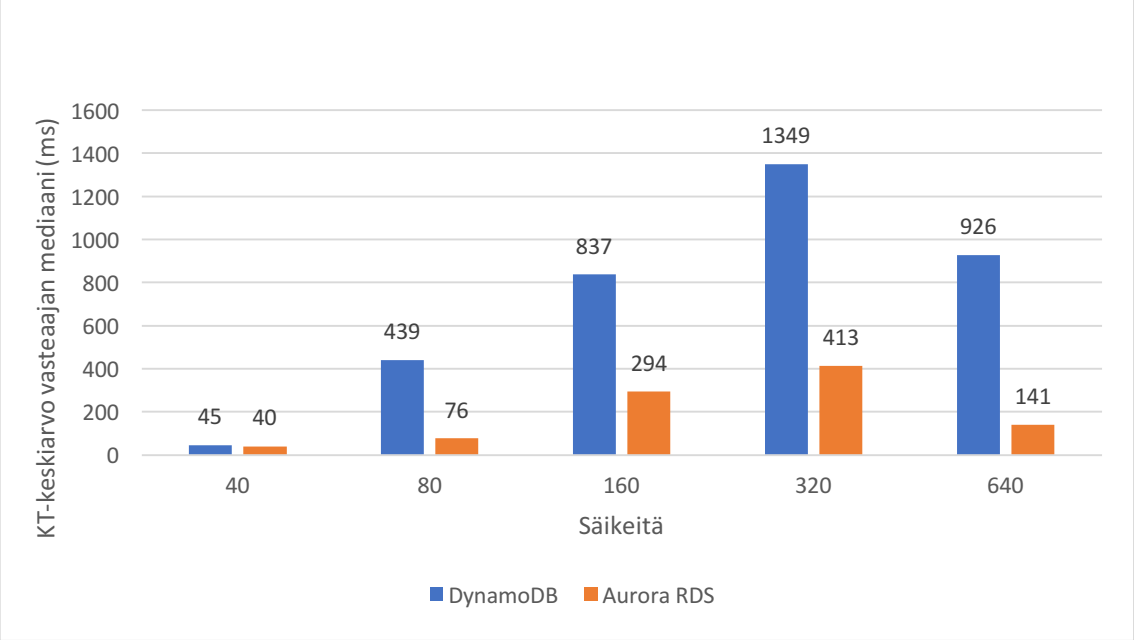
TAULUKKO 22 DynamoDB vasteaikoja (ms) käyttötapauskohtaisesti

Säikeiden määrä	KT1	KT2	KT3	KT-keskiarvo
40	4	175	21	45
80	20	1810	95	439
160	23	3517	126	837
320	51	5640	178	1349
640	27	3877	161	926

TAULUKKO 23 Aurora vasteaikoja (ms) käyttötapauskohtaisesti

Säikeiden määrä	KT1	KT2	KT3	KT-keskiarvo
40	5	156	12	40
80	6	308	9	76
160	40	1153	25	294
320	98	1492	43	413
640	17	557	18	141

Painotetun keskiarvokäyttötapauksen vasteajan mediaanista voi huomata, että Aurora-tietokantapalvelu suoriutuu käyttötapauksista kullakin kuormitustasolla DynamoDB-tietokantapalvelua nopeammin. Ero on merkittävin 320-säikeen kuormitustasolla, jolloin DynamoDB-tietokantapalvelua vasten suoritetun keskiarvokäyttötapauksen vasteajan mediaani on 1349 millisekuntia ja Aurora-tietokantapalvelua vasten suoritetun keskiarvokäyttötapauksen vasteajan mediaani on 413 millisekuntia. Toisaalta on huomioitava, että kyseisellä kuormitustasolla noin 10-prosenttia Aurora-tietokantapalvelua vasten tehdyistä kyselyistä päätyivät yhteysvirheisiin, eli vasteajat eivät kerro koko totuutta suorituskykyeroista kyseisellä kuormitustasolla. Suorituskykyero on kuitenkin huomattava myös 640-säikeen kuormituksella, jossa molempien tietokantapalveluiden kyselyiden virheprosentti oli 0.02. Tällä kuormitustasolla DynamoDB-tietokantapalvelua vasten suoritetun keskiarvokäyttötapauksen vasteajan mediaani oli 926 millisekuntia ja Aurora-tietokantapalvelua vasten suoritetun kesiarvokäyttötapauksen vasteajan mediaani oli 141 millisekuntia. Kuviossa 14 on verrattu tietokantapalveluiden keskiarvo käyttötapauksen vasteaikojen mediaaneja kuormitustasoittain.



KUVIO 14 Käyttötapauskohtaisten vasteaikojen vertailu

7.5 Yhteenveto

Tässä luvussa on esitetty miten kehitettyjä artefakteja, Amazon Aurora ja DynamoDB-tietokantapalveluihin toteutettuja tuotekatalogeja, evaluoidaan suorituskykytestauksen avulla. Luvussa on esitetty suorituskykytestin tavoitteet ja näitä tavoitteita mittaavat mittarit. Luvussa on esitetty testattavat käyttötapaukset ja niistä johdetut tietokantakyselyt tietokannoittain. Luvussa on esitetty suunnitelma kuormituksen nostosta ja kuvattu, miten suorituskykytestissä on simuloitu satunnaisuutta. Tämän jälkeen on esitetty suorituskykytestin tulokset ja vertailtu molempien tietokantapalvelujen tuloksia keskenään.

Suorituskykytestin tavoitteena oli löytää suorituskykyeroja tietokantapalveluiden välillä ja havainnoida skaalautuvuutta. Suorituskykytestin tuloksista voidaan löytää sekä suorituskykyeroja tietokantapalveluiden välillä, sekä molempien tietokantapalvelujen tuloksissa on havaittavissa palvelun skaalautumista kuormituksen kasvaessa. Tarkasteltiin sitten tietokantakyselyiden vasteaikojen mediaania tai käyttötapauksien vasteaikojen mediaania, oli Aurora-tietokantapalveluun toteutettu tuotekatalogi jokaisella kuormitustasolla DynamoDB-tietokantapalveluun toteutettua tuotekatalogia suorituskykyisempi tiedon saatavuuden näkökulmasta. Molempien tietokantapalvelujen kohdalla vasteaikojen mediaani laski kuormituksen noustessa 320-säikeestä 640 säikeeseen. Koska kyselyiden virheprosentti ei nouse merkittävästi kyseisten kuormitustasojen välillä, voidaan olettaa, että selittävä tekijä vasteajan laskulle kuormituksen noustessa on tietokantapalvelujen skaalautuminen, jolloin pilvipalvelu on varannut enemmän laskentaresursseja tietokantapalvelun käyttöön kuormituksen kasvaessa.

8 POHDINTA JA YHTEENVETO

Tutkielmassa on vertailtu kahden pilviympäristöön suunnitellun tietokannan hallintajärjestelmän suorituskykyä tuotekatalogin tietokantana pilviympäristössä. Tutkimusasetelma on merkittävä, koska suorituskyky erityisesti tuotekatalogin tiedon saatavuuden näkökulmasta on merkittävä tekijä menestyksekkääseen kaupankäyntiin verkossa, joka on Internetin yleistymisestä saakka ollut yksi merkittävimmistä Internetin sovellusten aiheista (Bochamann ym., 2002, Hirai ym., 2001). Pilvilaskenta toi mullistuksen verkkosovellusten suorituskykyyn tarjoamalla illuusion laskentaresurssien ehtymättömyydestä (Armbrust ym., 2009). Perinteiset tietokannan hallintajärjestelmät soveltuivat kuitenkin huonosti pilviympäristöön, missä laskentaresursseja skaalataan horisontaalisesti ottamalla lisää palvelimia käyttöön kuormituksen kasvaessa (Feuerlicht & Pokorný, 2011). Vastaamaan tähän haasteeseen kehitettiin joukko pilviympäristöön kehitettyjä tietokannan hallintajärjestelmiä, joita yhdisti relationaalisesta tietomallista luopuminen, ja tätä tietotokantakonseptia kutsuttiin nimellä NoSQL (Pokorný, 2011). Relationaalisesta tietomallista luopuminen mahdollisti tietokannan helpomman osittamisen useammalle palvelimelle, mutta samalla luovuttiin vahvoista tapahtumanhallintaominaisuuksista, jotka ovat kuitenkin pakollisia monien yritysten liiketoiminnassa (Pokorný, 2011). NewSQL-tietokantakonsepti pyrkii vastaamaan uudella tietokannan hallintajärjestelmäarkkitehtuurilla näihin haasteisiin tarjoamalla relationaalista tietomallia ja tapahtumanhallintaominaisuuksia pilviympäristössä (Bernstein, 2014, Pavlo & Aslett, 2016). Tutkielman tarkoitus oli selvittää, miten näillä moderneilla tietokannan hallintajärjestelmillä voidaan toteuttaa tuotekatalogin tietokanta ja näin tarjota pilvilaskennan suorituskykyä tuotekatalogin tiedon saatavuudelle. Aikaisemmat aiheeseen liittyvät tutkimukset ovat verranneet NoSQL-tietokantoja perinteisiin relaatiotietokannan hallintajärjestelmiin, eikä niitä ole toteutettu pilviympäristössä (Boicea ym., 2012, Floratou ym., 2012, Li & Manoharan, 2013, Abotourabi ym., 2015). Tutkielmassa tavoitteena oli selvittää, miten voidaan toteuttaa pilviympäristössä skaalautuva tuotekatalogin tietokanta ja kumman pilvipalveluna tarjottavan tietokantapalvelun suorituskyky on parempi tuotekatalogin tietokantana: AWS Aurora- vai DynamoDB-tietokantapalvelun.

Tutkimus toteutettiin suunnittelutieteellisenä konstruktiivisena tutkimuksena. Kehitettyjä artefakteja syntyi kaksi: tuotekatalogin tietokantatoteutus sekä Aurora- että DynamoDB-tietokantapalveluihin. Tuotekatalogien tietokantatoteutuksien kehitys perustui aikaisempaan kirjallisuuteen tuotekatalogin tiedonhallinnasta, sekä Amazonin tietokantapalvelukohtaiseen dokumentaatioon ja ohjeistukseen. Tuotekatalogin tietokantatoteutuksia evaluoitiin ja vertailtiin keskenään verkkosovellusten suorituskykytestauksen periaatteiden mukaisesti. Suorituskykytestauksessa testattiin tietokantatoteutuksien tiedon saatavuutta ja jätettiin huomioimatta tietokantojen tapahtumanhallintaominaisuudet. Suorituskykytestauksessa tietokantatoteutuksia kuormitettiin tietokantakyselyillä, jotka oli johdettu todellisen verkkokaupan HTTP-pyyntölokeja ja käyttöliittymää analysoimalla johdetuista käyttötapauksista.

Suorituskykytestin jälkeen voidaan edelleen vastata vain osittain tutkielman ensimmäiseen tutkimuskysymykseen. Molemmat tuotekatalogitoteutukset kyllä skaalautuvat kuormituksen kasvaessa, mutta kumpikaan tietokanta ei yksinään pysty täyttämään kaikkia tuotekatalogiin kohdistuvia tiedon saannin tarpeita. Molemmat tarkastellut tietokannat soveltuvat huonosti tiedonhakuun vapaalla hakusanalla, missä hakusanaa pitäisi verrata useaan tietokannan kenttään. Perinteiset relationaalista tietokantaa käyttävät verkkokauppasovellukset ovat yleensä käyttäneet erillistä hakumoottoria tuotteiden hakemiseen vapaalla hakusanalla. Kumpikaan tarkastelluista tietokantapalveluista ei poista tarvetta erilliselle hakumoottorille, jotta kaikki tuotekatalogin tiedon saannin vaatimukset voidaan täyttää. Vaikka DynamoDB-tietokantapalvelun joustava tietomalli soveltuu hyvin tuotekatalogin tiedonhallintaan, tarjoaa se ainoastaan perusavaimeen perustuvaa tehokasta tiedonhakua. Ehkä on olemassa NoSQL-dokumenttivarastoja, jotka tarjoavat joustavampaa tiedonhakua ja soveltuisivat näin paremmin tuotekatalogin tietokannaksi. Vaikka Aurora-tietokantapalvelu tarjoaa pilviympäristössä skaalautuvaa relationaalista tietomallia, soveltuu se huonosti myös pilviympäristössä tuotekatalogitoteutukseen, jossa tuotteita haetaan vapaalla hakusanalla, joka kohdistuu useampaan sarakkeeseen useammasta taulusta. Pilvilaskennan skaalautuvuus mahdollisesti tarjoaa kuitenkin suorituskykyetua perinteisiin relaationaalisiin tietokannan hallintajärjestelmiin, mikä on merkittävä löydös esimerkiksi verkkokaupankäynnille, jossa tietokannan hallintajärjestelmältä vaaditaan vahvoja transaktionhallintaominaisuuksia.

Toiseen tutkimuskysymykseen voidaan vastata helpommin suorituskykytestin tulosten pohjalta. Jokaisella kuormitustasolla Aurora-tietokantapalveluun toteutettu tuotekatalogi oli DynamoDB-tietokantapalveluun toteutettua tuotekatalogia suorituskykyisempi, kun tarkasteltiin tiedon saatavuutta.

Tutkimukseen löydökset ovat merkittäviä johtuen ainutlaatuisesta tutkimusasetelmasta. Moderneja pilviympäristöön suunniteltuja tietokannan hallintajärjestelmiä tutkittiin pilviympäristössä, jossa laskentaresursseja skaalataan horisontaalisesti ottamalla käyttöön lisää tietokoneresursseja. Tutkimuksen löydökset vahvistavat näkemystä, että laskentaresursseja voidaan dynaamisesti skaalata kuormituksen kasvaessa, ja että modernit tietokannan hallintajärjestelmät pystyvät hyödyntämään tätä horisontaalista laskentaresurssien skaalautuvuutta. Löydös myös tukee NewSQL-tietokantakonseptin rohkeaa lupausta tarjota pilviympäristössä skaalautuva relaatiotietokannan hallintajärjestelmä.

Kustannussyistä tutkimuksessa kuormitettiin tietokantapalveluita hyvin lyhytkestoisesti, eikä kuormitusajoa toistettu. Tästä huolimatta tuloksista voidaan havaita laskentaresurssien dynaamista skaalautumista. Jotta ilmiölle saataisiin lisää vahvistusta, tai mikäli haluttaisiin testata illuusiota ehtymättömistä laskentaresursseista, kuormitusta pitäisi ajaa pitkäkestoisemmin ja nostaa kunnes kyselyiden vasteajat tai virheprosentti lähtee huomattavaan nousuun.

Tutkimuksen tietokantapalvelut valittiin Amazonin pilvipalvelutuotevalikoimasta, koska tutkimuksen tavoitteena oli löytää helposti vertailtavissa olevat NoSQL-tietokanta ja NewSQL-tietokanta. Aurora ja DynamoDB ovat molemmat pilvipalveluna tarjottava tietokantatuote, joten vertailuasetelma kahden tietokantapalvelun välillä on otollinen, koska kummassakaan palvelussa ei tarvitse

palvelinresursseja määrittää tarkasti etukäteen. DynamoDB on NoSQL-tietokantakonseptin mukainen dokumenttivarasto, joka kategoristen ominaisuuksien perusteella pitäisi sopia hyvin tuotekatalogin tiedonhallintaan. DynamoDB-tietokantapalvelun tiedonhaun joustavuus ei kuitenkaan tämän tutkielman perusteella riitä kokonaisuudessa täyttämään tuotekatalogin tiedon saatavuuden vaatimuksia, kuten ei myöskään Aurora-tietokantapalvelunkaan. Jatkotutkimusaiheena voisi olla esimerkiksi jonkun toisen dokumenttivaraston tutkiminen tuotekatalogin tietokantana ja vertailukohtana voisi olla tuotekatalogitoteutus, joka on NewSQL-tietokannan hallintajärjestelmän lisäksi laajennettu hakumoottorilla.

LÄHTEET

Abadi, D. J. (2009). Data management in the cloud: Limitations and opportunities. *Bulletin of the Technical Committee on Data Engineering, 32*, 1, 3-12.

Aboutorabi, S. H., Rezapour, M., Moradi, M., & Ghadiri, N. (2015). Performance evaluation of SQL and MongoDB databases for big e-commerce data. *Computer Science and Software Engineering (CSSE), 2015 International Symposium on,* 1-7.

Agrawal, R., Somani, A. & Xu, Y. (2001). Storage and querying of e-commerce data. Vldb, (149-158).

Amazon Web Services (2016) Design Patterns using Amazon DynamoDB [online] Amazon Web Services [viitattu 31.12.2018] saatavilla www-osoitteessa <https://www.slideshare.net/AmazonWebServices/design-patterns-using-amazon-dynamodb>

Amazon Web Services (2018a) Amazon Aurora - Relational Database Build For The Cloud - AWS [online] Amazon Web Services [viitattu 7.12.2018] saatavilla www-osoitteessa <https://aws.amazon.com/rds/aurora/ >

Amazon Web Services (2018b) Amazon Aurora Serverless - On-Demand, Auto-Scalind Relational Database - AWS [online] Amazon Web Services [viitattu 7.12.2018] saatavilla www-osoitteessa <https://aws.amazon.com/rds/aurora/serverless/>

Amazon Web Services (2018c) Amazon DynamoDB - Features [online] Amazon Web Services [viitattu 7.12.2018] saatavilla www-osoitteessa <https://aws.amazon.com/dynamodb/features/>

Amazon Web Services (2018d) NoSQL Design for DynamoDB – Amazon DynamoDB [online] Amazon Web Services [viitattu 31.12.2018] saatavilla www-osoitteessa <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-general-nosql-design.html>

Amazon Web Services (2018e) Welcome - Amazon DynamoDB [online] Amazon Web Services [viitattu 31.12.2018] saatavilla www-osoitteessa <https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/Welcome.html>

Amazon Web Services (2019a) Best practices fro Querying and Scanning Data [online] Amazon Web Services [viitattu 3.1.2019] saatavilla www-osoitteessa

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-scan.html>

Amazon Web Services (2019b) Improving Data Access with Secondary Indexes [online] Amazon Web Services [viitattu 3.1.2019] saatavilla www-osoitteessa <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html>

Amazon Web Services (2019c) Managing Performance and Scaling for Amazon Aurora MySQL [online] Amazon Web Services [viitattu 21.1.2019] saatavilla www-osoitteessa <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Managing.Performance.html>

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M. (2009). *Above the clouds: A berkeley view of cloud computing (Report UCB/EECS-2009-28), Electronic Engineering and Computer Sciences Department, University of California, Berkeley.*

Arnold, J., Glavic, B. & Raicu, I. (2015). HRDBMS: A NewSQL database for analytics. Cluster Computing (CLUSTER), 2015 IEEE International Conference, (519-520).

Bernstein, D. (2014). Today's tidbit: VoltDB. Cloud Computing, IEEE, 1(1), 90-92.

Bochmann, G. V., Bourne, D., Evans, D., Kerhervé, B., Lau, T. C, Salem, M. M. & Ye, H. (2003). Scalability of web-based electronic commerce systems. IEEE Communications Magazine, 41(7), 110-115.

Bochmann, G. V., Bourne, D., Kerhervé, B. & Ye, H. (2002). Towards database scalability through efficient data distribution in e-commerce environments. Electronic Commerce, 2002. Proceedings. Third International Symposium on, 87-95.

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

Brewer, E. A., (2000). Towards Robust Distributed Systems. PODC 2000. Haettu 4.4.2012 osoitteesta <http://openstorage.gunadarma.ac.id/~mwiryana/Kuliah/Database/POD C-keynote.pdf>

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

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., et al. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems, 26*, 2, 4:1-4:26.

Feuerlicht, G. (2010). Database trends and directions: current challenges and opportunities. Teoksessa J. Pokorný, V.Snásel & K. Richta (toim.) *DATESO 2010, (s. 163-174)*

Feuerlicht, G., Pokorný, J. (2011). Can relational DBMS scale-up to the cloud? *Presented in ISD 2011 Conference, Edinburgh, Scotland.*

Floratou, A., Teletia, N., DeWitt, D. J., Patel, J. M., & Zhang, D. (2012). Can the elephants handle the NoSQL onslaught? Proc.VLDB Endow., 5(12), 1712-1723.

Google (2015) Google_Product_Taxonomy_Version: 2015-02-19 [online] Google [viitattu 1.10.2018] saatavilla www-osoitteessa <https://www.google.com/basepages/producttype/taxonomy-with-ids.en-US.txt>

Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. MIS Quarterly, 37, 337-355.

Hecht, R. & Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. Cloud and Service Computing (CSC), 2011 International Conference on, (336-341).

Hevner, A., March, S., Park, J., & Ram, S., (2004). Design science in information systems research. *Management Information Systems Quarterly, 28*, 75.

Hirai, S., Kimura, F., Fukushima, G., Kihara, R., Nakamura, S., & Nagata, M. (2001). Benchmark test of clustered transaction processing for electronic commerce. 2001 IEEE International

IBM (2018) IBM Knowledge Center - Attribute Data Model [online] International Business Machines Corp. [viitattu 1.12.2018] saatavilla www-osoitteessa <https://www.ibm.com/support/knowledgecenter/SSZLC2_9.0.0/com.ibm.commerce.database.doc/refs/rdb_datamodel_attribute.htm>

IBM (2019) IBM Knowledge Center - WebSphere Commerce search [online] International Business Machines Corp. [viitattu 31.1.2019] saatavilla www-osoitteessa <https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdsearch.htm>

Jastrow, T., & Preuss, T. (2015). The entity-attribute-value data model in a multi-tenant shared data environment. *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC),* 494-497.

Jhingran, A. (2000). Moving up the food chain: Supporting E-commerce applications on databases. ACM SIGMOD Record, 29(4), 50-54.

Korpinen, S. (2011) Sovelluksen siirtäminen pilviympäristöön, Tietojenkäsittelytieteiden laitos, Pro Gradu-tutkielma, Jyväskylän yliopisto, Jyväskylä

Li, Y & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on,* 15-19.

Magento (2018) Magento for Developers: Part 7—Advanced ORM: Entity Attribute Value [online] Magento [viitattu 1.12.2018] saatavilla www-osoitteessa <https://devdocs.magento.com/guides/m1x/magefordev/mage-for-dev-7.html>

Magento (2019) How to Install and Configure the Solr Search Engine With Magento Enterprise Edition (EE) 1.x [online] Magento [viitattu 31.1.2019] saatavilla www-osoitteessa <https://devdocs.magento.com/guides/m1x/other/ht_magento-solr.html>

Maia, F., Armendáriz-iñigo, J., Ruiz-Fuertes, M., & Oliveira, R. (2010). Scalable transactions in the cloud: Partitioning revisited. Teoksessa R. Meersman (toim.) *On the move to meaningful internet systems, OTM 2010 (s. 785-797) Springer Berlin / Heidelberg.*

Meier, J., Farre, C., Bansode, P., Barber, S. & Rea, D., (2007) Performance Testing Guidance for Web Applications: Patterns & Practices. Microsoft Press.

Pavlo, A., & Aslett, M. (2016). What's really new with NewSQL? SIGMOD Rec., 45(2), 45-55.

Pokorný, J. (2010). Databases in the 3rd millennium: Trends and research directions. *Journal of Systems Integration, 1*, 1-2

Pokorný, J. (2011). NoSQL Databases: a step to database scalability in Web environment. *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and* Services, Ho Chi Minh City, Vietnam (s. 278-283)

Pritchett, D. (2008). BASE: An Acid Alternative. *ACM Queue, 6, 3, 48-55*

Vaquero, L. M., Rodero-Merino, L., Caceres, J. & Lindner, M. (2009). A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communications Review, 39(1)*, 50–55.

LIITE 1 TIETOKANTATAULUJEN LUONTILAUSEET

```
-- -----------------------------------------------------
-- Schema tuotekatalogi
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS tuotekatalogi DEFAULT CHARACTER SET utf8 ;

USE tuotekatalogi ;

-- -----------------------------------------------------
-- Table tuotekatalogi.tuote
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS tuotekatalogi.tuote (
  tuote_id INT NOT NULL AUTO_INCREMENT,
  tuotenumero VARCHAR(45) NULL,
  nimi VARCHAR(256) CHARACTER SET utf8mb4 NULL,
  kuvaus TEXT(4000) CHARACTER SET utf8mb4 NULL,
  PRIMARY KEY (tuote_id) ,
  INDEX fk_tuote_tuotenumero_idx (tuotenumero ASC),
  FULLTEXT fk_tuote_nimi1_idx (nimi),
  FULLTEXT fk_tuote_kuvaus1_idx (kuvaus)
)ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table tuotekatalogi.attribuutti
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS tuotekatalogi.attribuutti (
  attribuutti_id INT NOT NULL AUTO_INCREMENT,
  nimi VARCHAR(128) CHARACTER SET utf8mb4 NULL,
  yksikko VARCHAR(45) CHARACTER SET utf8mb4 NULL,
  tyyppi VARCHAR(45) CHARACTER SET utf8mb4 NULL,
  PRIMARY KEY (attribuutti_id),
  FULLTEXT fk_attribuutti_nimi1_idx (nimi)
)ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table tuotekatalogi.attribuutti_arvo
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS tuotekatalogi.attribuutti_arvo (
  attribuutti_arvo_id INT NOT NULL AUTO_INCREMENT,
  tuote_id INT NOT NULL,
  attribuutti_id INT NOT NULL,
  arvo_int INT NULL,
  arvo_varchar VARCHAR(128) NULL,
  arvo_float FLOAT NULL,
  PRIMARY KEY (attribuutti_arvo_id, tuote_id, attribuutti_id),
  INDEX fk_attribuutti_arvo_tuote1_idx (tuote_id ASC),
  INDEX fk_attribuutti_arvo_attribuutti1_idx (attribuutti_id ASC),
  FULLTEXT fk_attribuutti_arvo_varchar1_idx (arvo_varchar ASC),
  INDEX fk_attribuutti_arvo_composite1_idx (tuote_id, arvo_varchar),
  CONSTRAINT fk_attribuutti_arvo_tuote1
```

```
    FOREIGN KEY (tuote_id)
    REFERENCES tuotekatalogi.tuote (tuote_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_attribuutti_arvo_attribuutti1
    FOREIGN KEY (attribuutti_id)
    REFERENCES tuotekatalogi.attribuutti (attribuutti_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table tuotekatalogi.kategoria
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS tuotekatalogi.kategoria (
  kategoria_id INT NOT NULL AUTO_INCREMENT,
  nimi VARCHAR(128) NULL,
  kategorianumero INT NOT NULL,
  PRIMARY KEY (kategoria_id),
  UNIQUE INDEX kategorianumero_UNIQUE (kategorianumero ASC)
)ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table tuotekatalogi.tuote_kategoria
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS tuotekatalogi.tuote_kategoria (
  tuote_id INT NOT NULL,
  kategoria_id INT NOT NULL,
  PRIMARY KEY (tuote_id, kategoria_id),
  INDEX fk_tuote_kategoria_tuote1_idx (tuote_id ASC),
  INDEX fk_tuote_kategoria_kategoria1_idx (kategoria_id ASC),
  CONSTRAINT fk_tuote_kategoria_tuote1
    FOREIGN KEY (tuote_id)
    REFERENCES tuotekatalogi.tuote (tuote_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_tuote_kategoria_kategoria1
    FOREIGN KEY (kategoria_id)
    REFERENCES tuotekatalogi.kategoria (kategoria_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table tuotekatalogi.kategoriahierarkia
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS tuotekatalogi.kategoriahierarkia (
  ala_kategoria_id INT NOT NULL,
  yla_kategoria_id INT NOT NULL,
  PRIMARY KEY (ala_kategoria_id, yla_kategoria_id),
  INDEX fk_kategoriahierarkia_kategoria1_idx (ala_kategoria_id ASC),
```

```
  CONSTRAINT fk_kategoriahierarkia_kategoria1
    FOREIGN KEY (ala_kategoria_id)
    REFERENCES tuotekatalogi.kategoria (kategoria_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_kategoriahierarkia_kategoria2
    FOREIGN KEY (yla_kategoria_id)
    REFERENCES tuotekatalogi.kategoria (kategoria_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
)ENGINE = InnoDB;
```

LIITE 2 DYNAMODB-KYSELYT

```
//KT1
HashMap<String, String> nameMap = new HashMap<String, String>();
nameMap.put("#tn", "tuotenumero");

HashMap<String, Object> valueMap = new HashMap<String, Object>();
valueMap.put(":nro", Integer.parseInt(valueStr));

QuerySpec querySpec = new QuerySpec().withKeyConditionExpression("#tn = 
:nro").withNameMap(nameMap)
    .withValueMap(valueMap);

ItemCollection<QueryOutcome> items =  table.query(querySpec);

//KT2a
HashMap<String, String> nameMap = new HashMap<String, String>();
nameMap.put("#kn", "kategoria");
nameMap.put("#ty", "tyyppi");

HashMap<String, Object> valueMap = new HashMap<String, Object>();
valueMap.put(":nro", Integer.parseInt(valueStr));
valueMap.put(":typ", "TUOTE");

QuerySpec querySpec = new QuerySpec().withKeyConditionExpression("#kn = :nro
and begins_with(#ty, :typ)").withNameMap(nameMap)
    .withValueMap(valueMap);
    querySpec.setMaxResultSize(25);

ItemCollection<QueryOutcome> items = kategoriaIndex.query(querySpec);

//KT2b
HashMap<String, String> nameMap = new HashMap<String, String>();
nameMap.put("#kn", "kategoria");
nameMap.put("#ty", "tyyppi");

HashMap<String, Object> valueMap = new HashMap<String, Object>();
valueMap.put(":nro", Integer.parseInt(valueStr));
valueMap.put(":typ", "ATT");

QuerySpec querySpec = new QuerySpec().withKeyConditionExpression("#kn = :nro
and begins_with(#ty, :typ)").withNameMap(nameMap)
    .withValueMap(valueMap);

ItemCollection<QueryOutcome> items = kategoriaIndex.query(querySpec);

//KT3
HashMap<String, String> nameMap = new HashMap<String, String>();
nameMap.put("#an", "att_nimi");
```

```
nameMap.put("#av", attrValField);

HashMap<String, Object> valueMap = new HashMap<String, Object>();
valueMap.put(":atni", attributeName);
valueMap.put(":val", attributeValue);
valueMap.put(":kategoria", category);

QuerySpec querySpec = new QuerySpec().withKeyConditionExpression("#an = :atni
and #av = :val").withNameMap(nameMap)
    .withValueMap(valueMap).withFilterExpression("kategoria = :kategoria");

ItemCollection<QueryOutcome> items = attribuuttiIndex.query(querySpec);
```

LIITE 3 SQL-KYSELYT

```
-- KT1a
select tuote.tuotenumero,
tuote.nimi,
tuote.kuvaus
from tuote where tuote.tuotenumero = 0000000002;

-- KT1b
select attribuutti.nimi,
attribuutti.yksikko,
attribuutti.tyyppi,
attribuutti_arvo.arvo_int,
attribuutti_arvo.arvo_float,
attribuutti_arvo.arvo_varchar
from attribuutti_arvo
inner join tuote on tuote.tuote_id = attribuutti_arvo.tuote_id
inner join attribuutti on attribuutti_arvo.attribuutti_id = attribuutti.attri-
buutti_id
where tuote.tuotenumero = 0000000002;

-- KT2a
select tuote.nimi, tuote.tuotenumero
from tuote
inner join tuote_kategoria on tuote.tuote_id = tuote_kategoria.tuote_id
inner join kategoria on tuote_kategoria.kategoria_id = kategoria.kategoria_id
where kategoria.kategorianumero = 4990 limit 25;

-- KT2b
select attribuutti.attribuutti_id,
attribuutti.nimi,
attribuutti.yksikko,
attribuutti.tyyppi,
attribuutti_arvo.arvo_int,
attribuutti_arvo.arvo_float,
attribuutti_arvo.arvo_varchar
from attribuutti
inner join attribuutti_arvo on attribuutti_arvo.attribuutti_id = attri-
buutti.attribuutti_id
inner join tuote on tuote.tuote_id = attribuutti_arvo.tuote_id
inner join tuote_kategoria on tuote.tuote_id = tuote_kategoria.tuote_id
inner join kategoria on tuote_kategoria.kategoria_id = kategoria.kategoria_id
where kategoria.kategorianumero = 4990

-- KT3
select
tuote.tuotenumero
from tuote
inner join tuote_kategoria on tuote.tuote_id = tuote_kategoria.tuote_id
```

```
inner join kategoria on tuote_kategoria.kategoria_id = kategoria.kategoria_id
where kategoria.kategorianumero = 4990 and tuote.tuote_id in (
    select tuote_id
    from attribuutti_arvo
    where attribuutti_arvo.attribuutti_id = 64 and attribuutti_arvo.arvo_var-
char = 'Manual'
    );
```