Tampereen ammattikorkeakoulu

NoSQL-tietokantojen käyttö suurten datamäärien hallinnassa

Lauri Karvinen

OPINNÄYTETYÖ
Maaliskuu 2025

Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

KARVINEN, LAURI:
NoSQL-tietokantojen käyttö suurten datamäärien hallinnassa

Opinnäytetyö 33 sivua, joista liitteitä 0 sivua
Maaliskuu 2025

Opinnäytetyössä tutkittiin NoSQL-tietokantojen käyttöä suurten datamäärien hallinnassa ja suorituskykyä verrattuna SQL-tietokantoihin. Työssä keskityttiin erityisesti MongoDB:n ja MySQL:n vertailuun, jotka ovat suosituimpia esimerkkejä NoSQL- ja SQL-tietokannoista.

Opinnäytetyön tavoitteena oli vertailla NoSQL- ja SQL-tietokantojen välisiä suorituskykyeroja suurten datamäärien hallinnassa. Tutkimuksessa tarkasteltiin tietokantojen tallennus- ja hakunopeuksia. Testit toteutettiin luomalla prototyyppi, jolla simuloitiin tietokantojen käyttöä reaalimaailman tilanteissa tallentamalla ja hakemalla suuria määriä käyttäjätietoja. Testeillä arvioitiin, kuinka hyvin tietokannat suoriutuvat datan tallentamisesta ja hakemisesta.

Tulokset osoittivat, että MySQL suoriutui paremmin hakukyselyissä, kun taas MongoDB:n tehokkuus korostui massatallennuksissa. SQL:n hakukyselyissä yksittäisen käyttäjän hakuaika pysyi tasaisena, mikä viittaa indeksoinnin tehokkuuteen. NoSQL puolestaan suoriutui massatallennuksista paremmin, sillä tallennusajat eivät kasvaneet yhtä jyrkästi kuin MySQL:n tallennusajat. Testien perusteella voidaan todeta, että NoSQL on parempi vaihtoehto suurien rakenteettomien tai puolirakenteellisten datamäärien käsittelyyn. MySQL puolestaan sopii monimutkaisiin hakukyselyihin ja tilanteisiin, joissa datan eheys on tärkeää.

Asiasanat: NoSQL, SQL, tietokannat

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

KARVINEN, LAURI:
Use of NoSQL Databases in Managing Large Datasets

Bachelor's thesis 33 pages, appendices 0 pages
March 2025

The thesis examined the use of NoSQL databases in managing large datasets and their performance compared to SQL databases. The study focused particularly on the comparison between MongoDB and MySQL which are among the most popular examples of NoSQL and SQL databases.

The objective of the thesis was to compare the performance differences between NoSQL and SQL databases in handling large datasets. The research analyzed the storage and retrieval speeds of these databases. The tests were conducted by creating a prototype that simulated real-world scenarios where large amounts of user data were stored and retrieved. The tests aimed to evaluate how well the databases perform in data storage and retrieval.

The results showed that MySQL performed better in query searches, whereas MongoDB excelled in mass data storage. In SQL queries, the retrieval time for a single user remained consistent, indicating the efficiency of indexing. In contrast, NoSQL performed better in large-scale data storage, as its storage times did not increase as steeply as MySQL’s. Based on the tests, it can be concluded that NoSQL is a better option for handling large amounts of unstructured or semi-structured data. On the other hand, MySQL is more suitable for complex queries and situations where data integrity is crucial.

Key words: NoSQL, SQL, databases

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

☐ Ei
☒ Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni tutkielmaprosessin aikana seuraavia tekoälysovelluksia:

Tekoälysovellusten nimet ja versiot: ChatGPT 4o ja o1

Käyttötarkoitus: Koodin suunnittelu ja luonti, analysointi, kielen tarkastus, kääntäminen

Osiot, joissa tekoälyä on käytetty: Kaikki

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

SISÄLLYS

LYHENTEET JA TERMIT

SQL	Relaatiotietokantojen kyselykieli
NoSQL	Tietokantatyyppi, joka ei perustu relaatiomalliin
Scale-out	Vaakasuuntainen skaalautuminen, suorituskykyä lisätään uusilla palvelimilla
Scale-up	Pystysuuntainen skaalautuminen, suorituskykyä lisätään päivittämällä palvelin tehokkaammaksi
ACID	Lyhenne sanoista Atomicity, Consistency, Isolation ja Durability
CRUD	Lyhenne sanoista Create, Read, Update ja Delete
Transaktio	Joukko tietokantakomentoja
MongoDB	Dokumenttipohjainen NoSQL-tietokanta
Skeema	Tietomalli, joka määrittää, miten tietoa tallennetaan
Mongoose	Node.js kirjasto, joka tarjoaa MongoDB:lle käyttöliittymän
Tietomalli	Määrittely, miten data organisoidaan ja tallennetaan
Indeksi	Tietokannan rakenne, joka osoittaa mistä tiedot löytyvät

1 JOHDANTO

Viime aikoina NoSQL-tietokannat ovat kasvattaneet suosiotaan sovelluksissa, jotka käsittelevät suuria ja nopeasti kasvavia datamääriä (NoSQL Explained). Perinteiset SQL-tietokannat ovat hyviä ratkaisuja rakenteellisen datan hallinnassa, mutta ne voivat kohdata ongelmia skaalautuvuuden ja joustavuuden suhteen (Advantages and Disadvantages...). NoSQL-tietokannat ratkaisevat nämä haasteet, sillä ne mahdollistavat tehokkaan hajautettujen tietojen käsittelyn, joustavan skeemarakenteen ja paremman suorituskyvyn massadatan käsittelyssä (Advantages and Disadvantages...).

Yksi tyypillinen käyttökohde NoSQL-tietokannoille on verkkokauppojen käyttäjätietojen hallinta ja migraatiot (Datastax). Verkkokaupoissa on suuria määriä käyttäjädataa ja perinteiset SQL-tietokannat voivat kohdata suorituskykyrajoitteita, erityisesti käyttäjätietoja siirtäessä uuteen järjestelmään. NoSQL-tietokannat, kuten MongoDB mahdollistaa käyttäjätietojen siirtämisen tehokkaasti ilman merkittävää heikkenemistä suorituskyvyssä (Relational Migrator). Tämä muistuttaa opinnäytetyössä suoritettuja testejä, joissa simuloitiin samanlaista tilannetta sekä MongoDB:lla että MySQL:lla.

Opinnäytetyössä tutkitaan NoSQL-tietokantojen käyttöä suurten datamäärien hallinnassa. Työssä tarkastellaan MongoDB:ta, joka on yksi suosituimmista NoSQL-tietokannoista, sekä MySQL:ää, joka on perinteinen relaatiotietokanta. Tavoitteena on vertailla NoSQL- ja SQL-tietokantoja suorituskyvyn ja skaalautuvuuden näkökulmasta.

Käytännön osuudessa toteutetaan prototyyppi, jonka avulla tarkastellaan MongoDB:n ja MySQL:n kirjoitus- ja hakunopeuksia tietokannan kasvaessa. Testit suoritetaan simuloiden tilannetta, jossa suuri määrä dataa tallennetaan rinnakkain, mikä vastaa reaalimaailman kuormitusta. Tulosten avulla tutkitaan, miten tietokannat suoriutuvat suurien datamäärien hallinnasta ja millaisia rajoituksia tai haasteita sen käytössä voi ilmetä.

2 Mikä on NoSQL?

NoSQL-tietokannat (lyhenne sanoista Not Only SQL) ovat tietokantoja, jotka eivät perustu perinteiseen relaatiomalliin. Tyypillisiä piirteitä näille on joustava tietomalli, helppo skaalautuvuus ja kyky käsitellä suuria tai nopeasti kasvavia datamääriä. NoSQL ilmestyi 2000-luvun alkupuolella, jolloin tallennustilan hinta laski nopeasti ja tarve käsitellä suuria datamääriä kasvoi. (NoSQL Explained.)

2.1 NoSQL kategoriat

NoSQL:lle neljä yleisintä tietokantatyyppiä ovat dokumenttipohjaiset tietokannat, avainarvo tietokannat, sarakepohjaiset tietokannat ja graafitietokannat. (NoSQL Explained.)

Dokumenttipohjaisessa tietokannassa data tallennetaan dokumentteina, jotka muistuttavat rakenteeltaan JSON-objekteja. Jokainen dokumentti koostuu kenttien ja arvojen pareista. Kentät voivat sisältää eri tietoja, kuten tekstiä, numeroita, totuusarvoja tai taulukoita. (NoSQL Explained.)

Avainarvotietokannat varastoivat dataa avaimilla, jotka ovat yksilöllisiä. Jokainen avain saa vain yhden arvon. (NoSQL Explained.)

Sarakepohjaisissa tietokannoissa data järjestetään taulukoihin ja riveihin, mutta jokaisella rivillä voi olla joustavasti erilaiset sarakkeet. Tämä eroaa perinteisistä relaatiotietokannoista, joissa kaikilla riveillä on sama ennalta määritetty sarakerakenne. (NoSQL Explained.)

Graafitietokannassa tiedot tallennetaan solmuihin ja näitä solmuja yhdistäviin reunoihin. Solmut kuvaavat jotakin reaalimaailman kohdetta, kuten ihmistä tai paikkaa. Reunat kertovat näiden kohteiden väliset suhteet. (NoSQL Explained.)

2.2 NoSQL käyttötarkoitukset

NoSQL-tietokantoihin siirrytään usein, kun sovelluskehityksessä halutaan ketteryyttä ja joustavuutta datan rakenteen suhteen. Projekti voi vaatia toistuvia muutoksia tietomalliin ja NoSQL-tietokannat sallivat nämä muutokset ilman raskaita skeemapäivityksiä. Tämä nopeuttaa kehitystä, sillä kehittäjät voivat itse päivittää tietorakennetta. (When to use NoSQL.)

NoSQL-tietokannat ovat myös suunniteltu isojen ja vaihtelevien datamassojen käsittelyyn. Ne soveltuvat tilanteisiin, joissa dataa kertyy nopeasti ja paljon. Esimerkiksi verkkopalvelut, joissa käyttäjämäärät ja datamäärät saattavat kasvaa eksponentiaalisesti. Useimmiten skaalautuminen hoidetaan lisäämällä useita tavallisia palvelimia yhteen toimivaksi ryhmäksi (scale-out). Tämä on yleensä halvempi tapa kuin ostaa yksi entistä tehokkaampi palvelin (scale-up). (When to use NoSQL.)

Koska NoSQL-tietokannat on rakennettu scale-out menetelmällä, niiden on helpompi käsitellä isoja liikennemääriä ilman käyttökatkoksia. Esimerkiksi versio- tai rakennepäivityksiä voidaan tehdä samalla, kun palvelu pysyy käynnissä. (When to use NoSQL.)

Monet nykyiset ohjelmistoratkaisut tarvitsevat eri tapoja käsitellä dataa yhtä aikaa. Toiset osat järjestelmästä vaativat nopeaa transaktioiden käsittelyä, kun taas toisissa tarvitaan laajempaa analytiikkaa. NoSQL-tietokannat vastaavat näihin tarpeisiin samassa tietokannassa. (When to use NoSQL.)

Yhteenvetona NoSQL on usein oikea valinta, kun tarvitaan suurta kehitysnopeutta, joustavaa tietomallia, skaalautuvuutta ja vähäisiä käyttökatkoja.

2.3 Vertailu SQL-tietokantoihin

Perinteiset SQL-tietokannat perustuvat relaatiomalliin, jossa data tallennetaan rivien ja sarakkeiden muodostamiin taulukoihin tiukasti määritellyn skeeman mu-

kaisesti. Näitä tietokantoja on käytetty useissa kriittisissä liiketoiminta- ja yritys-sovelluksissa ja niistä tunnetuimpia ovat muun muassa MySQL ja PostgreSQL. (Advantages and Disadvantages...)

NoSQL-tietokannat ovat joukko erilaisia teknologioita, jotka pyrkivät ratkaisemaan suuren tai vaihtelevan datamäärän käsittelyyn liittyviä haasteita. Ne soveltuvat useimmiten tilanteisiin, joissa tiedon rakenne muuttuu usein tai dataa kertyy suuria määriä nopeasti. NoSQL-tietokannoissa skeema ei ole yhtä tiukasti sidottu dataan, mikä mahdollistaa joustavamman kehityksen ja helpomman laajentamisen. Esimerkkejä tällaisista ratkaisuista ovat MongoDB, Cassandra ja Redis. (Advantages and Disadvantages...)

2.3.1 Taulukko vertailusta

TAULUKKO 1. Tietokantojen vertailu. Lähde Geeks for geeks

Ominaisuus	SQL	NoSQL
Tietomalli ja skeema	Tiukka, ennalta määritetty skeema	Joustava tai lähes skeematon rakenne
Skaalautuvuus	Scale-up	Scale-out
Suorituskyky	Suoriutuu hyvin pienillä datamäärillä ja rakenteisen datan kanssa	Suoriutuu hyvin isojen datamäärien ja hajautettujen ympäristöjen kanssa
Transaktiot	ACID yhteensopivuus	Rajoitettu ACID yhteensopivuus
Tietotyyppien käsittely	Hyvä rakenteisen datan kanssa	Sopii puolirakenteiselle ja rakenteettomalle datalle
Tyypilliset käyttökohteet	Sovellukset, joissa suoritetaan paljon transaktioita ja jokaisen tallennuksen on oltava luotettava	Sovellukset, joissa dataa kertyy paljon ja nopeasti tai rakenteet vaihtelevat

2.3.2 Tietomalli ja skeema

SQL-tietokannoissa tiedot tallennetaan ennalta määriteltyihin taulukoihin, joissa skeema määrittelee tarkasti, mitä tietoa tallennetaan ja missä muodossa. Tämä on hyödyllistä, kun halutaan varmistaa datan eheys ja johdonmukaisuus.

NoSQL-tietokannoissa skeema on joustava tai lähes olematon. Tämä tekee kehityksestä nopeampaa, koska tietorakennetta voidaan muuttaa ilman suuria rakenteellisia muutoksia. Joustavan skeeman ansiosta NoSQL on hyvä ratkaisu projekteihin, joissa datan rakenne muuttuu usein.

2.3.3 Skaalautuvuus ja suorituskyky

Skaalautuvuus on yksi merkittävä ero SQL- ja NoSQL-tietokantojen välillä. SQL-tietokannat skaalautuvat enimmäkseen pystysuunnassa (scale-up), mikä tarkoittaa, että suorituskykyä lisätään tehokkaammalla laitteistolla. Tämä voi toimia pienemmissä projekteissa hyvin, mutta isompien datamäärien kanssa voi tulla rajoituksia.

NoSQL-tietokannat on suunniteltu skaalautumaan vaakasuunnassa (scale-out). Uusia palvelimia voidaan lisätä helposti, mikä tekee niistä hyvän valinnan sovelluksiin tai järjestelmiin, joissa datamäärät kasvavat nopeasti.

2.3.4 Transaktiot ja eheys

SQL-tietokannat noudattavat ACID-periaatteita (Atomicity, Consistency, Isolation, Durability), jotka huolehtivat transaktioiden luotettavuudesta ja datan eheydestä. Erityisen tärkeää se on esimerkiksi pankkisovelluksissa ja talousjärjestelmissä, joissa pienikin virhe datassa voi johtaa vakaviin seurauksiin.

NoSQL-tietokannoissa ACID yhteensopivuus on usein rajoitettua. Monet ratkaisut painottavat saatavuutta ja suorituskykyä, minkä seurauksena tiedot eivät aina ole yhdenmukaisia. Tämä ei kuitenkaan aiheuta ongelmia sovelluksissa, jossa saatavuus on tärkeämpää.

2.3.5 Tyypilliset käyttökohteet

SQL-tietokannat soveltuvat parhaiten sovelluksiin, joissa suoritetaan paljon transaktioita ja halutaan varmistaa datan eheys, esimerkiksi rahoitusjärjestelmissä. NoSQL-tietokannat ovat taas hyvä valinta sovelluksiin, joissa dataa kertyy paljon ja nopeasti, kuten sosiaalisen median sovellukset.

2.3.6 Yhteenveto

Valinta SQL:n ja NoSQL:n välillä riippuu sovelluksen tarpeista. NoSQL-tietokantojen skaalautuvuus ja joustavuus on tärkeää nopeasti kehittyvissä projekteissa, kun taas SQL-tietokannat sopivat järjestelmiin, joissa vaaditaan luotettavia transaktioita ja eheää dataa.

3 Prototyyppi MongoDB:lla

3.1 Toteutus

Prototyypin toteutus sisälsi kolme vaihetta, tietokantayhteyden muodostaminen, tietomallin luominen ja CRUD-toimintojen toteuttaminen. Yhteys MongoDB tietokantaan muodostettiin Mongoose-kirjaston avulla. Käytössä oli paikallisesti asennettu MongoDB ja yhteys toteutettiin komennolla mongodb://127.0.0.1:27017/testdb, jossa testdb toimii tietokantana (KUVA 1). Yhteyden muodostamiseksi asennettiin tarvittavat Node.js-paketit komennolla npm install mongoose. Tietokannan tarkasteluun käytettiin MongoDBCompass sovellusta. Koodin suunnittelussa ja toteutuksessa hyödynnettiin tekoälyä (ChatGPT 4o).

```
mongoose.connect('mongodb://127.0.0.1:27017/testdb')

  .then(() => {
  console.log('Yhteys MongoDB:hen muodostettu');
  runTests();
}).catch(err => {
  console.error('Virhe yhteydessä MongoDB:hen:', err);
});
```

KUVA 1. Yhteys tietokantaan.

Käyttäjien tiedot tallennettiin Mongoose-kirjastolla määritellyn skeeman mukaisesti (KUVA 2). Skeema sisälsi käyttäjän nimen, sähköpostiosoitteen, iän sekä ryhmäkohtaisen datan. Nimi ja sähköposti tallennettiin merkkijonoina ja ikä numerona. Käyttäjät jaettiin viiteen eri ryhmään ja jokaisella oli oma lisäkenttä. Esimerkiksi Admin-käyttäjillä oli oikeuksiin liittyvää tietoa ja VIP-käyttäjillä tilaushistoriatietoja. Sähköposti määritettiin uniikiksi, mikä loi sille automaattisesti indeksin. Tämä nopeuttaa yksittäisen käyttäjätiedon palautusta.

```
// Käyttäjän skeema
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  age: Number,
  userType: String,
  accessLevel: String,
  purchaseHistory: [String],
  loginCount: Number,
  lastLogin: Date,
  preferences: Object
});
```

KUVA 2. Käyttäjän skeema.

Käyttäjät lisättiin insertMany()-metodilla. Lisäykset suoritettiin rinnakkain Promise.all()-metodilla simuloimaan oikean elämän tilannetta, jossa käyttäjiä lisätään samanaikaisesti eri tiedoilla. Käyttäjäryhmiä oli Admin, Standard, Guest, Moderator ja VIP. Jokaiselle ryhmälle määriteltiin ominaistietoja. Admin- ja Moderatorkäyttäjille lisättiin käyttöoikeudet, Standard-käyttäjille sisäänkirjautumisten määrä, VIP-käyttäjille ostohistoria ja Guest-käyttäjille sivuston asetustietoja.

Käyttäjien nimet ja sähköpostiosoitteet muodostettiin yhdistämällä järjestysnumero sekä käyttäjäryhmän nimi. Ikä arvottiin satunnaisesti ja tallennusajan mittaamiseen otettiin aika ennen ja jälkeen tallennuksen. (KUVA 3)

```
// 1. Lisää käyttäjiä rinnakkain eri ryhmissä
async function insertManyUsers() {
  try {
    const userTypes = ["Admin", "Standard", "Guest", "Moderator", "VIP"];
    const usersPerGroup = 2000; // 10 000 käyttäjää yhteensä

    console.log("1. Aloitetaan 10 000 käyttäjän lisääminen rinnakkain...");
    const startTime = performance.now();

    // Luo eri käyttäjäryhmille omat listansa
    const userPromises = userTypes.map(userType => {
      let users = [];
      for (let i = 0; i < usersPerGroup; i++) {
        let user = {
          name: `${userType}_User${i}`,
          email: `${userType.toLowerCase()}_user${i}@example.com`,
          age: Math.floor(Math.random() * 60) + 18,
          userType: userType,
          lastLogin: new Date()
        };

        // Ryhmäkohtaiset tiedot
        if (userType === "Admin") {
          user.accessLevel = "Super";
        } else if (userType === "Standard") {
          user.loginCount = Math.floor(Math.random() * 500);
        } else if (userType === "Guest") {
          user.preferences = { theme: "dark", notifications: Math.random() > 0.5 };
        } else if (userType === "Moderator") {
          user.accessLevel = "Limited";
        } else if (userType === "VIP") {
          user.purchaseHistory = [1];
        }
```

KUVA 3. Käyttäjien lisääminen.

Kaikki käyttäjät haettiin find()-metodilla. Haun nopeuttamiseksi käytettiin lean()-metodia (KUVA 4). Oletuksena Mongoose palauttaa raskaita dokumenttiolioita, lean()-metodin käyttö keventää kyselyä palauttamalla pelkän JavaScript-objektin (MongooseJs). Haun suoritusajan mittaamiseksi käytettiin samaa tekniikkaa kuin käyttäjien tallennuksessa.

```
async function testFindUsers() {
  try {
    console.log("2. Aloitetaan kaikkien käyttäjien haku...");
    const startTime = performance.now();

    const users = await User.find({}).lean();

    const endTime = performance.now();
    console.log(`2. Haettiin ${users.length} käyttäjää, kesto: ${(endTime - startTime).toFixed(2)} ms`);
  } catch (error) {
    console.error("Virhe haussa:", error);
  }
}
```

KUVA 4. Kaikkien käyttäjien haku.

Yksittäinen käyttäjä haettiin sähköpostiosoitteella käyttäen findOne()-metodia (KUVA 5). Myös yksittäisen käyttäjän hakuajan mittaamiseen käytettiin samaa tekniikkaa kuin tallennuksessa.

```
async function testFindOneUser() {
  try {
    console.log("3. Aloitetaan yksittäisen käyttäjän haku...");
    const startTime = performance.now();

    const user = await User.findOne({ email: /user500@example.com$/ });

    const endTime = performance.now();
    console.log(`3. Haettiin käyttäjä: ${user ? user.name : "Ei löytynyt"}, kesto: ${(endTime - startTime).toFixed(2)} ms`);
  } catch (error) {
    console.error("Virhe haussa:", error);
  }
}
```

KUVA 5. Yksittäisen käyttäjän haku.

3.2 Suorituskykytestit

MongoDB:n suorituskykyä testattiin mittaamalla tietokannan kirjoitus- ja hakunopeutta. Testit arvioivat, kuinka hyvin MongoDB käsittelee suuria datamääriä ja kuinka nopeasti se pystyy tallentamaan sekä hakemaan tietoa. Testien toteutuksessa käytettiin Node.js ympäristöä ja Mongoose kirjastoa tietokantakyselyihin.

Testikoneessa oli käytössä (Intel Core i5-11400F, 16 Gt RAM, SSD, Windows 10) ja testit suoritettiin paikallisesti.

Testeissä suoritettiin kolme osaa eri käyttäjämäärillä (10 000, 20 000 ja 50 000). Ensimmäisessä vaiheessa tietokantaan lisättiin käyttäjädata ja kirjoituksen kesto mitattiin. Tämä auttaa ymmärtämään, kuinka MongoDB skaalautuu suurien tietomäärien kanssa ja kuinka nopeasti suuri määrä dataa voidaan tallentaa tietokantaan.

Toisessa vaiheessa mitattiin kaikkien käyttäjien hakunopeutta. Tämä testi osoittaa, miten tietokannan koko vaikuttaa MongoDB:n hakunopeuteen.

Kolmannessa vaiheessa suoritettiin yksittäisen käyttäjän haku sähköpostin perusteella. Tämä testi osoittaa, kuinka nopeasti MongoDB pystyy palauttamaan yksittäisen tietueen suuren datamäärän joukosta.

Testien tavoitteena on arvioida MongoDB:n suorituskykyä eri kuormitustasoilla ja tunnistaa mahdollisia ongelmia suurien datamäärien käsittelyssä.

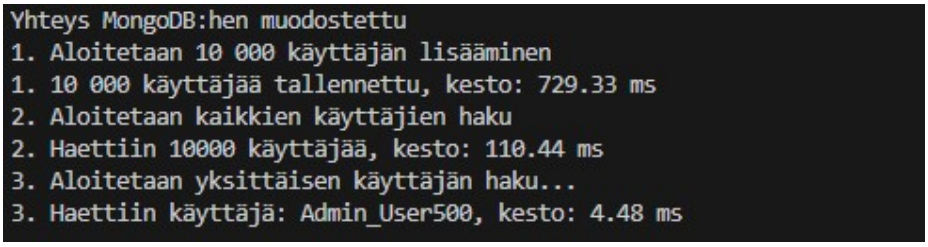
3.2.1 Tulokset

Seuraavassa taulukossa esitellään MongoDB:n testien tulokset eri käyttäjämäärillä (TAULUKKO 2). Taulukko sisältää kolme mittaria, joita ovat tallennusaika, kaikkien hakuaika ja yksittäisen hakuaika. Tulokset ovat millisekunneissa kolmella eri käyttäjämäärällä.

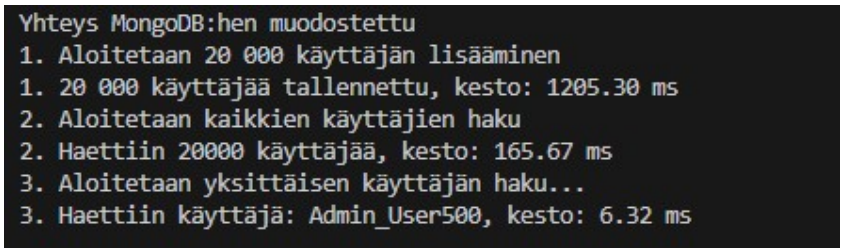
TAULUKKO 2. MongoDB:n suorituskykytestien tulokset

Käyttäjä-määrä	Tallennusaika (ms)	Kaikkien haku (ms)	Yksittäisen haku (ms)
10 000	729.33	110.44	4.48
20 000	1205.30	165.67	6.32
50 000	2387.27	359.99	11.55

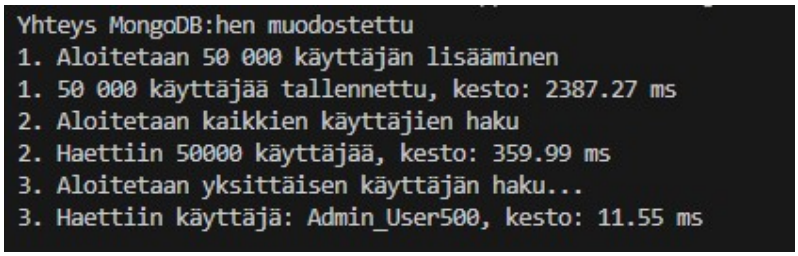
Testitulosten havainnollistamiseksi seuraavat kuvat näyttävät konsolitulostukset testeistä (KUVAT 6-8). Jokaisessa kuvassa näkyvät mitatut ajat tallennukselle, kaikkien käyttäjien haulle ja yksittäisen käyttäjän haulle käyttäjämäärillä 10 000, 20 000 ja 50 000.



KUVA 6. Tulostus 10 000 käyttäjällä.



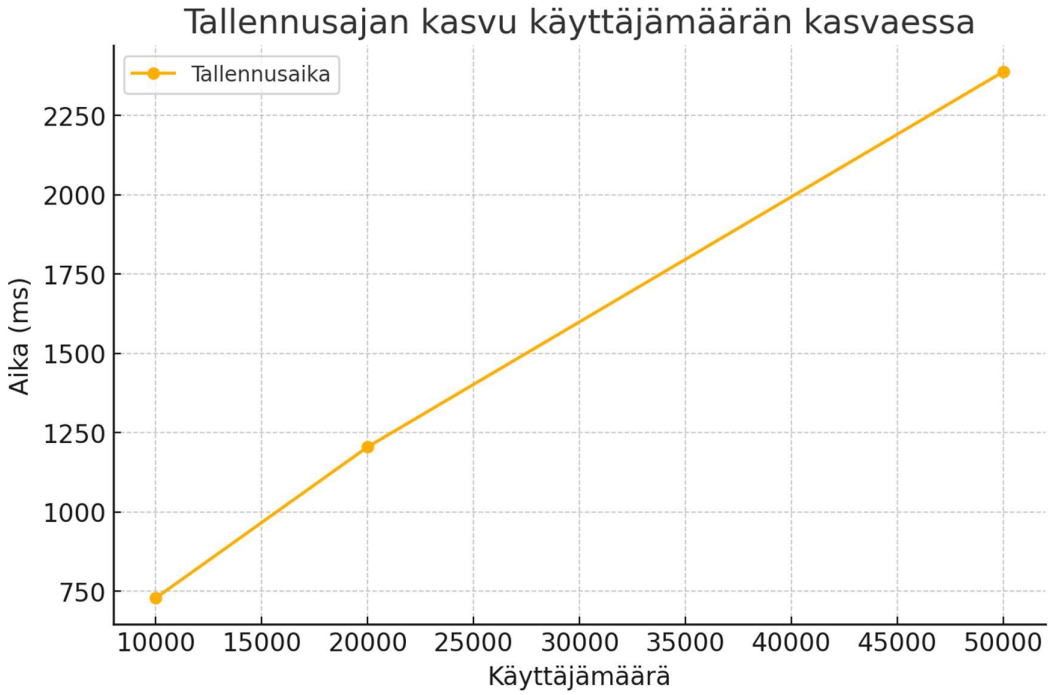
KUVA 7. Tulostus 20 000 käyttäjällä.



KUVA 8. Tulostus 50 000 käyttäjällä.

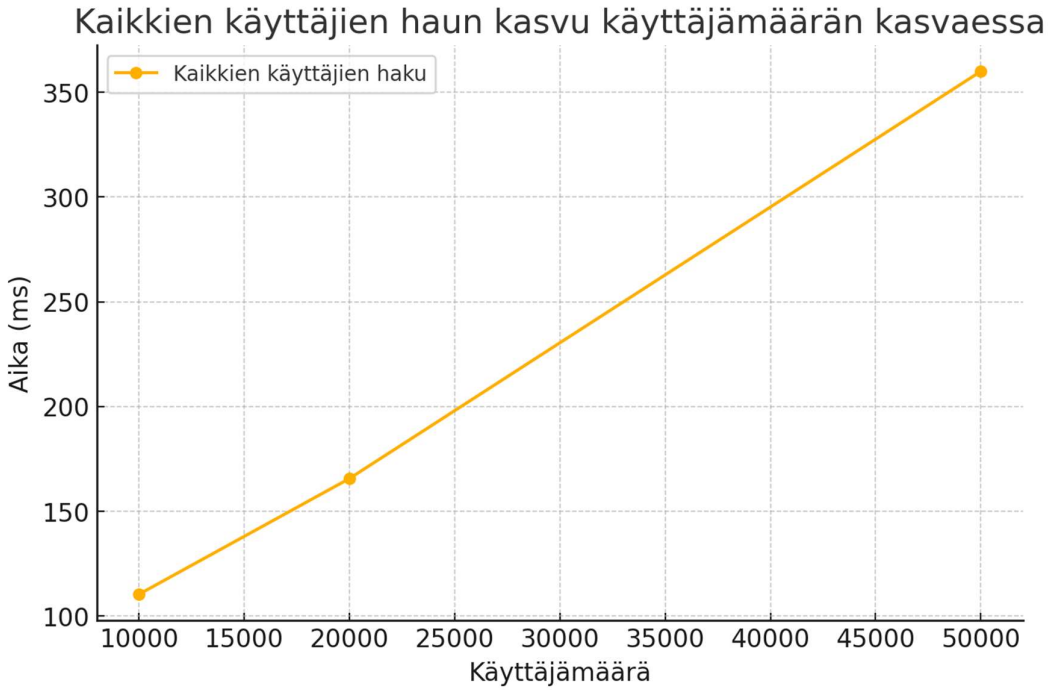
3.2.2 Suorituskyvyn analyysi

Tallennusnopeuden testit osoittavat, että tallennusaika ei kasva täysin lineaarisesti (KUVA 9). Ensimmäisessä testissä lisättiin 10 000 käyttäjää, tallennus kesti 729.33 millisekuntia. Käyttäjien kaksinkertaistuessa (20 000 käyttäjää) tallennus kesti 1205.30 millisekuntia ja 50 000 käyttäjällä tallennus kesti 2387.27 millisekuntia (TAULUKKO 2). Suuremmilla käyttäjämäärillä tallennus näyttää tehostuvan, 50 000 käyttäjän tallennus kesti suhteessa käyttäjämäärään vähemmän kuin 10 000 käyttäjän tallennus. Tämä osoittaa, että MongoDB skaalautuu hyvin ja sen tehokkuus korostuu massatallennuksissa.



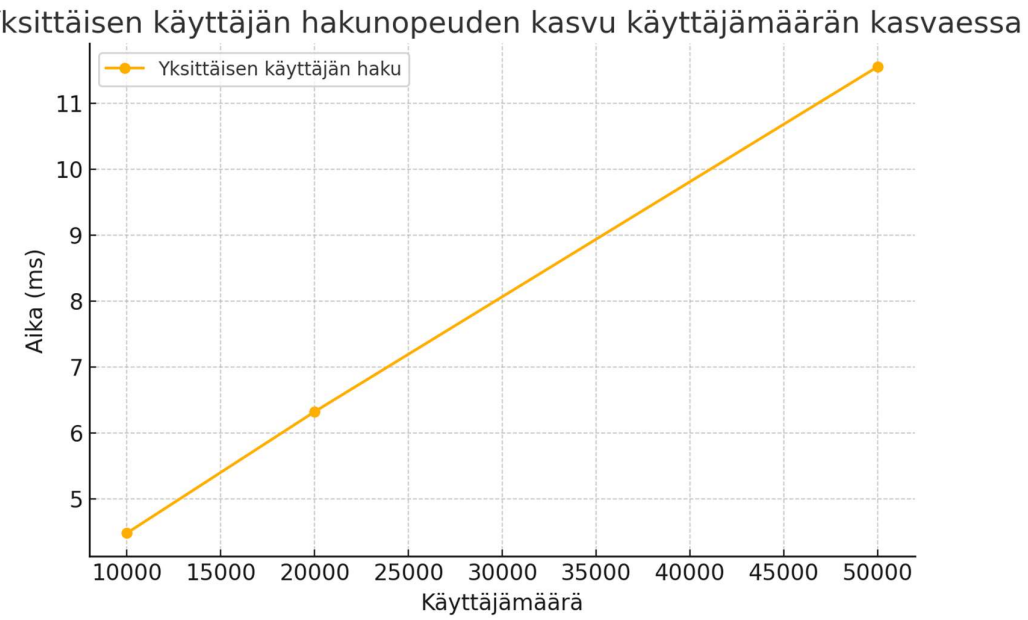
KUVA 9. Tallennusajan kuvaaja käyttäjämäärän kasvaessa (ChatGPT 4o).

Kaikkien käyttäjien haun testit osoittavat, että hakunopeuden kasvu ei myöskään ole lineaarista (KUVA 10). Jos kasvu olisi täysin lineaarista, 20 000 käyttäjän odotettu hakuaika olisi 220.88 millisekuntia ja 50 000 käyttäjän hakuaika 552.2 millisekuntia. Kun haettiin 10 000 käyttäjää, haku kesti 110.44 millisekuntia. 20 000 käyttäjällä hakuaika oli 165.67 millisekuntia ja 50 000 käyttäjällä 359.99 millisekuntia (TAULUKKO 2). Tämä osoittaa MongoDB:n hakujen tehokkuuden suurilla tietomäärillä.



KUVA 10. Kaikkien käyttäjien hakuajan kuvaaja käyttäjämäärän kasvaessa (ChatGPT 4o).

Yksittäisen käyttäjän hakuajan kasvu oli hyvin samankaltaista, kuin tallennus- ja hakuajan kasvu (KUVA 11). Tähän voi vaikuttaa, että haku suoritettiin sähköpostilla, joka määritettiin uniikiksi ja MongoDB loi tälle automaattisesti indeksin. Ilman indeksiä voidaan odottaa, että hakuaika kasvaa enemmän datamäärän kasvaessa enemmän. 10 000 käyttäjän hakunopeus oli 4.48 millisekuntia ja 20 000 käyttäjän 6.32 millisekuntia. 50 000 käyttäjän kohdalla hakunopeus oli 11.55 millisekuntia (TAULUKKO 2).



KUVA 11. Yksittäisen käyttäjän hakuajan kuvaaja käyttäjämäärän kasvaessa (ChatGPT 4o)

3.2.3 Yhteenveto

Tallennus-, kaikkien käyttäjien haku- ja yksittäisen käyttäjän hakuajat eivät kasvaneet lineaarisesti käyttäjämäärän kasvaessa, vaan suuremmilla tietomäärillä suorituskyky parani suhteessa pienempiin tietomääriin. Tämä osoittaa, että MongoDB on optimoitu käsittelemään suuria tietomääriä tehokkaasti.

4 Prototyyppi SQL

4.1 Toteutus

SQL-versiota varten asennettiin MySQL ja MySQL Workbench. Ensin palvelin käynnistettiin käyttäen MySQL Workbench sovellusta. Seuraavaksi luotiin testdb tietokanta suorittamalla SQL-komento CREATE DATABASE testdb;.
Tietokantayhteyden muodostamiseksi asennettiin mysql2-kirjasto komennolla npm install mysql2. Yhteyden muodostamiseen käytettiin mysql2/promise-kirjastoa (KUVA 12). Koodin suunnittelussa ja toteutuksessa hyödynnettiin tekoälyä (ChatGPT 4o).

```
const mysql = require('mysql2/promise');

const db = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: 'salasana',
    database: 'testdb',
    waitForConnections: true,
});
```

KUVA 12. Yhteys tietokantaan.

Yhteyden muodostamisen jälkeen tietokantaan luotiin users-taulu, joka vastasi NoSQL-version käyttäjämallia. Taulussa oli käyttäjätietoja sisältävät sarakkeet, kuten name, email, age sekä käyttäjäryhmät. Sähköpostiosoite määritettiin uniikiksi, jolloin MySQL luo sille automaattisesti indeksin (KUVA 13).

```
async function createTable() {
  const sql = `
    CREATE TABLE IF NOT EXISTS users (
      id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(100) NOT NULL,
      email VARCHAR(255) NOT NULL UNIQUE,
      age INT,
      userType VARCHAR(50),
      accessLevel VARCHAR(50),
      purchaseHistory TEXT,
      loginCount INT,
      lastLogin DATETIME,
      preferences JSON
    );
  `;
```

KUVA 13. Users taulu.

SQL-version toteutus on rakenteeltaan pitkälti samanlainen kuin MongoDB-versio, joten yksittäisiä toiminnallisuuksia ja testien vaiheita ei käsitellä yksityiskohtaisesti. Testien toiminnot toteutettiin samalla periaatteella kuin MongoDB:ssa ja tämän takia tässä osiossa keskitytään enemmän MySQL:n ja MongoDB:n eroihin erityisesti suorituskyvyssä ja skaalautuvuudessa. Koodin suunnittelussa ja toteutuksessa hyödynnettiin tekoälyä (ChatGPT 4o).

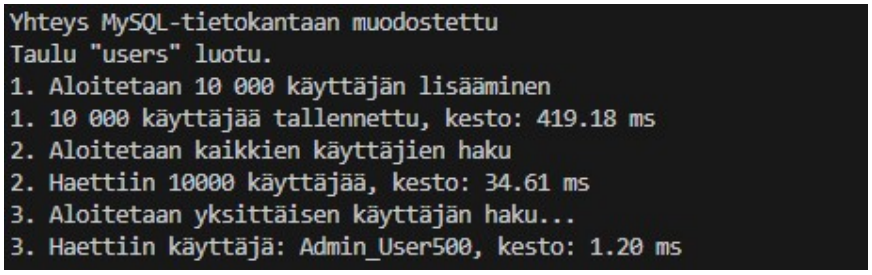
4.2 Tulokset

Seuraavassa taulukossa esitellään MySQL:n testien tulokset eri käyttäjämäärillä (TAULUKKO 3). Taulukko sisältää kolme mittaria, joita ovat tallennusaika, kaikkien hakuaika ja yksittäisen hakuaika. Tulokset ovat millisekunneissa kolmella eri käyttäjämäärällä.

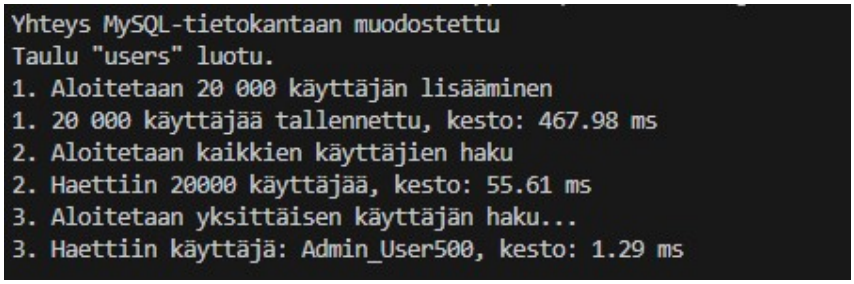
TAULUKKO 3. MySQL:n suorituskykytestien tulokset

Käyttäjä-määrä	Tallennusaika (ms)	Kaikkien haku (ms)	Yksittäisen haku (ms)
10 000	419.18	34.61	1.20
20 000	467.98	55.61	1.29
50 000	890.09	112.27	1.04

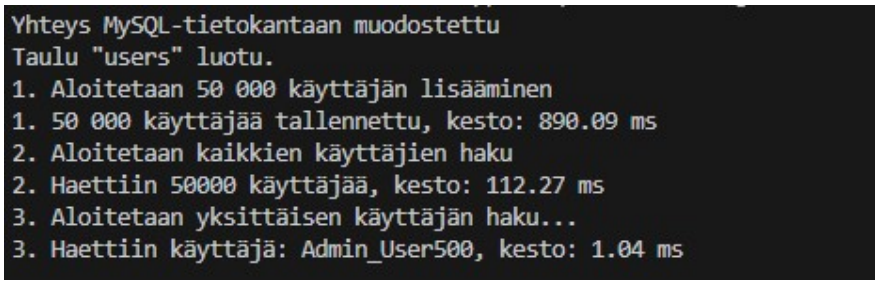
Testitulosten havainnollistamiseksi seuraavat kuvat näyttävät konsolitulostukset testeistä (KUVAT 14-16). Jokaisessa kuvassa näkyvät mitatut ajat tallennukselle, kaikkien käyttäjien haulle ja yksittäisen käyttäjän haulle käyttäjämäärillä 10 000, 20 000 ja 50 000.



KUVA 14. Tulostus 10 000 käyttäjällä.



KUVA 15. Tulostus 20 000 käyttäjällä.

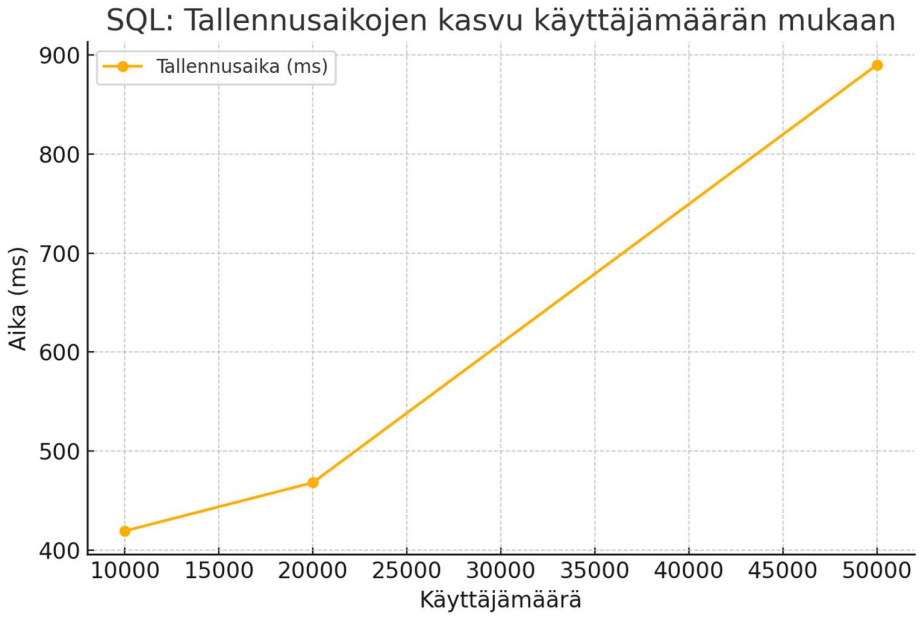


KUVA 16. Tulostus 50 000 käyttäjällä.

4.2.1 Suorituskyvyn analyysi

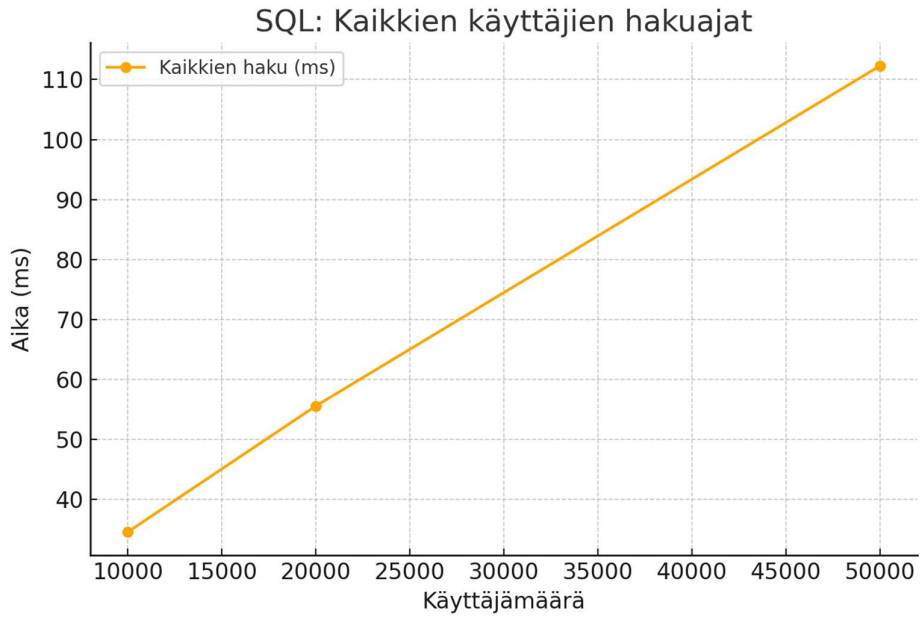
Tallennusajan testit osoittavat, että SQL-tietokannan suorituskyky pysyi tasaisena aluksi, mutta käyttäjämäärän kasvaessa tallennusaika kasvoi huomattavasti (KUVA 17). Ensimmäisessä testissä 10 000 käyttäjän tallennus kesti 419.18 millisekuntia ja 20 000 käyttäjän tallennus kesti 467.98 millisekuntia, mikä on noin 50 millisekunnin nousu. 50 000 käyttäjän kohdalla tallennus kesti 890.09, joka on

lähes kaksinkertainen edelliseen tallennukseen verrattuna. Tämä viittaa siihen, että MySQL:n skaalautuvuus ei ole lineaarista ja suuremmilla tietomäärillä tallennusnopeus alkaa hidastumaan huomattavasti.



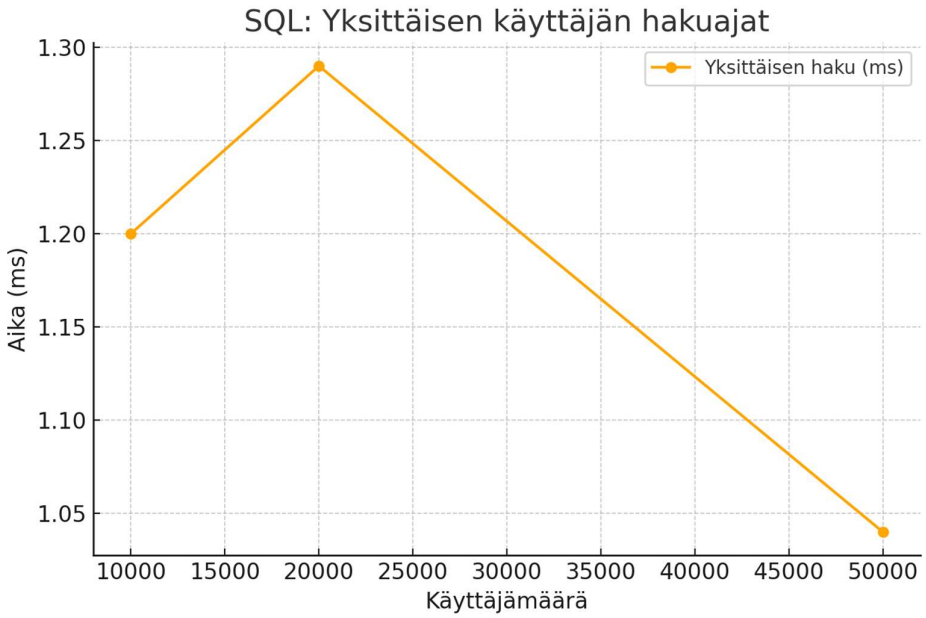
KUVA 17. Tallennusajan kuvaaja käyttäjämäärän kasvaessa (ChatGPT 4o).

Kaikkien käyttäjien hakutestit osoittavat, että hakuajan kasvu pysyy tasaisempana pienemmillä datamäärillä ja se ei kärsi datamäärän suurenemisesta yhtä paljon, kuin tallennusaika (KUVA 18). Ensimmäisessä testissä 10 000 käyttäjän haku kesti 34.61 millisekuntia ja 20 000 käyttäjällä hakuaika kasvoi 55.61 millisekuntiin. 50 000 käyttäjän hakuaika oli 112.27, joka on yli kaksinkertainen 20 000 käyttäjän hakuaikaan verrattuna, mutta suhteessa tämä kasvu on lähellä 10 000 ja 20 000 käyttäjän välistä kasvua. Tämä viittaa siihen, että MySQL-hakukyselyt skaalautuvat hyvin ja suuremmilla tietomäärillä ei ole odotettavissa merkittäviä hidastumisia.



KUVA 18. Kaikkien käyttäjien hakuajan kuvaaja käyttäjämäärän kasvaessa (ChatGPT 4o).

Yksittäisen käyttäjän hakutestit osoittavat, että hakuaika pysyi hyvin matalana kaikilla käyttäjämäärillä (KUVA 19). 10 000 käyttäjän kohdalla yksittäisen käyttäjän haku kesti 1.20 millisekuntia. 20 000 käyttäjällä haku kesti 1.29 millisekuntia, mikä on todella pieni 0.09 millisekunnin nousu. 50 000 käyttäjän kohdalla hakuaika oli 1.04, joka on nopeampi kuin pienemmillä käyttäjämäärillä. Tämä viittaa siihen, että MySQL:n hakukyselyt yksittäisten tietueiden osalta skaalautuu erittäin hyvin, eikä datan määrä vaikuta juurikaan hakunopeuteen. Tämä johtuu todennäköisesti sähköpostin indeksoinnista, joka luotiin automaattisesti määrittelemällä sähköposti uniikiksi. Tästä voidaan päätellä, että SQL:n indeksointi on erityisen tehokasta yksittäisten tietueiden hakemisessa.



KUVA 19. Yksittäisen käyttäjän hakuajan kuvaaja käyttäjämäärän kasvaessa (ChatGPT 4o).

4.2.2 Yhteenveto

Tallennusnopeuden suoritusaika pysyi tasaisena pienemmillä käyttäjämäärillä, mutta 50 000 käyttäjällä suorituskyky alkoi hidastumaan huomattavasti. Tämä viittaa siihen, että suuremmat datamäärät voivat aiheuttaa tallennuksen hidastumista.

Kaikkien käyttäjien hakuaika kasvoi tasaisemmin käyttäjämäärän kasvaessa, kasvu pysyi hallittuna ja suuria harppauksia ajan kasvussa ei ollut. Suuremmilla datamäärillä MySQL-tietokanta skaalautui hyvin, mikä osoittaa, että relaatiotietokannat ovat optimoituja tehokkaisiin hakukyselyihin.

Yksittäisen käyttäjän hakuajat pysyivät erittäin nopeana kaikilla käyttäjämäärillä. Tämä johtuu todennäköisesti sähköpostin automaattisesta indeksoinnista. Tämä osoittaa SQL-indeksien tehokkuuden, sillä hakuaika pysyi lähes samana ja jopa laski 50 000 käyttäjällä.

5 Vertailu ja analyysi

SQL- ja NoSQL-tietokantojen suorituskykyä vertailtiin testitulosten perusteella, joissa tarkasteltiin tallennusnopeutta, kaikkien käyttäjien hakuaikaa ja yksittäisen käyttäjän hakuaikaa. Tulokset osoittavat selkeitä eroja tietokantojen välillä erityisesti tallennuksessa ja yksittäisen tietueen haussa.

SQL:n suorituskyky pysyi tasaisena pienemmillä käyttäjämäärillä, mutta suuremmilla tietomäärillä tallennus hidastui huomattavasti. NoSQL:n tallennusajan kasvu oli tasaisempaa, eikä saanut yhtä jyrkkää nousua datamäärän kasvaessa. Tämä viittaa siihen, että MongoDB on optimoitu paremmin suurten datamäärien käsittelyyn ja MySQL:n tallennukset voivat kohdata rajoituksia suurilla tietomäärillä.

Kaikkien käyttäjien hakutestit osoittivat, että molemmat tietokannat skaalautuivat hyvin. MySQL:n hakuaika kasvoi tasaisesti käyttäjämäärän kasvaessa ilman suuria hidastumisia. MongoDB suoriutui myös hyvin, mutta hakuajat olivat pidempiä verrattuna MySQL:n hakuaikoihin. Molempien hakuaikojen kasvut olivat suhteellisesti hyvin lähellä toisiaan, mutta MongoDB:n kasvu oli hiukan suurempaa kuin MySQL:n kasvu. Tämä viittaa siihen, että SQL:n optimoinnit hakukyselyissä voivat olla hyödyllisempiä suurilla tietomäärillä.

Yksittäisen käyttäjän haku pysyi nopeana molemmissa tietokannoissa, mutta MongoDB:n hakuaika kasvoi käyttäjämäärän kasvaessa toisin kuin MySQL:n hakuaika pysyi lähes vakiona ja jopa nopeutui 50 000 käyttäjällä. Tämä osoittaa, että molemmat tietokannat palauttavat nopeasti yksittäisen tietueen, mutta MongoDB:n hakuajassa on odotettavissa nousua datamäärän kasvaessa.

SQL soveltuu parhaiten tilanteisiin, joissa vaaditaan vahvaa tietojen eheyttä ja monimutkaisia hakukyselyitä. Relaatiotietokannat skaalautuvat tehokkaasti hakutoiminnallisuuksissa. MySQL suoriutui hyvin suurten tietomäärien haussa ja yksittäisen tietueen hausta suuren tietomäärän joukosta. Tämä osoittaa, että MySQL on hyvä ratkaisu sovelluksille, joissa tietojen hakeminen on olennainen osa sovellusta.

NoSQL soveltuu paremmin suuriin ja nopeasti kasvaviin datamääriin, joissa tietomalli voi muuttua nopeasti. MongoDB:n suorituskyky oli MySQL:ää parempi tallennuksessa, mikä viittaa siihen, että MongoDB:n on optimoitu suurien datamäärien nopeaan käsittelyyn. NoSQL-tietokannat soveltuvat tilanteisiin, joissa tallennusnopeus ja skaalautuvuus ovat tärkeämpiä, kuin datan eheys.

Yhteenvetona voidaan todeta, että molemmilla tietokannoilla on omat vahvuutensa ja valinta vaihtelee käyttötarpeiden mukaan. SQL on sopiva vaihtoehto käsittelemään rakenteellista dataa, jossa vaaditaan datan eheyttä ja tehokkaita hakukyselyitä. NoSQL puolestaan on parempi vaihtoehto, kun käsitellään suuria määriä puolirakenteellista tai rakenteetonta dataa, jossa etusijalla on tallennusnopeus ja skaalautuvuus.

6 Pohdinta

Tutkimuksen tulokset osoittavat, että SQL- ja NoSQL-tietokannoilla on omat vahvuutensa riippuen käyttötarkoituksesta. SQL-tietokanta suoriutui paremmin hakukyselyissä, mikä viittaa siihen, että SQL-tietokannat ovat optimoituja tehokkaaseen tiedonhakuun. NoSQL-tietokanta suoriutui paremmin suurten datamäärien tallennuksessa, joka osoittaa sen sopivan massadatan nopeaan käsittelyyn.

Testit suoritettiin paikallisessa kehitysympäristössä, jossa ei ollut monimutkaisia reaalimaailman kuormituksia tai rinnakkaisia käyttäjätoimintoja, kuten yhtäaikaisia tallennuksia tai hakuja eri käyttäjiltä. Tämä tarkoittaa, että testit eivät välttämättä vastaa todellisia tilanteita, joissa järjestelmän kuormitus voi olla huomattavasti suurempi. Lisäksi tekniset rajoitukset, kuten käytetty laitteisto voivat vaikuttaa tuloksiin.

Tulosten perusteella ei voida todeta, että jompikumpi tietokanta olisi yleisesti toistaan parempi. SQL- ja NoSQL-tietokantojen välinen valinta pitää tehdä käyttötarkoituksen perusteella. SQL on hyvä valinta järjestelmiin, joissa on monimutkaisia hakukyselyitä ja datan eheys on tärkeää. Toisaalta, jos halutaan käsitellä suuria ja nopeasti muuttuvia datamääriä ilman tiukkaa skeemarakennetta, NoSQL on hyvä vaihtoehto. Wisal ym. (2023) ja Győrödi ym. (2022) päätyvät samankaltaisiin johtopäätöksiin tutkimuksissaan.

Opinnäytetyön tavoitteena oli tutkia NoSQL-tietokantojen käyttöä suurten datamäärien hallinnassa ja vertailla niiden suorituskykyä SQL-tietokantoihin. Tavoitteeseen päästiin, sillä testit antoivat konkreettisia tuloksia tietokantojen vahvuuksista ja heikkouksista. Jatkokehityksen kannalta olisi hyvä laajentaa testejä suurempiin tietomääriin ja testata tietokantojen suorituskykyä monimutkaisemmissa käyttötilanteissa, kuten rinnakkaisten kyselyiden tai monen yhtäaikaisen käyttäjän kuormituksessa. Mielestäni työ on onnistunut, sillä sain syvennettyä ymmärrystäni tietokantojen toiminnasta ja eroista, vaikka testien laajuutta voisi vielä kehittää.

LÄHTEET

NoSQL explained. n.d. MongoDB. Verkkosivu. Luettu 4.2.2025
https://www.mongodb.com/resources/basics/databases/nosql-explained

When to use NoSQL. n.d. MongoDB. Verkkosivu. Luettu 4.2.2025
https://www.mongodb.com/resources/basics/databases/nosql-explained/when-to-use-nosql

Advantages and Disadvantages of using SQL vs. NoSQL Databases. n.d. Geeks for geeks. Verkkosivu. Luettu 4.2.2025
https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-using-sql-vs-nosql-databases/

Mongoose tutorial. n.d. Geeks for geeks. Verkkosivu. Luettu 5.2.2025
https://www.geeksforgeeks.org/mongoose-tutorial/

Getting started. n.d. Mongoosejs. Verkkosivu. Luettu 5.2.2025
https://mongoosejs.com/docs/index.html

Getting started with MongoDB and Mongoose. n.d. MongoDB. Verkkosivu. Luettu 5.2.2025
https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/

Indexes. n.d. MongoDB. Verkkosivu. Luettu 10.2.2025
https://www.mongodb.com/docs/manual/indexes/#get-started

Index unique. n.d. MongoDB. Verkkosivu. Luettu 10.2.2025
https://www.mongodb.com/docs/manual/core/index-unique/

Unique indexes in MongoDB. n.d. Geeks for geeks. Verkkosivu. Luettu 10.2.2025
https://www.geeksforgeeks.org/unique-indexes-in-mongodb/

Tutorials. n.d. Mongoodejs. Verkkosivu. Luettu 11.2.2025
https://mongoosejs.com/docs/tutorials/lean.html

Datastax. NoSQL use cases. Luettu 24.2.2025
https://www.datastax.com/guides/nosql-use-cases

Relational Migrator. n.d. MongoDB. Verkkosivu. Luettu 24.2.2025
https://www.mongodb.com/docs/relational-migrator/

Wisal, K., Teerath, K., Zhang, C., Kislay, R., Arunabha, M R. & Bin, L. 2023. SQL and NoSQL Databases Software architectures performance analysis and assessments. Big Data and Cognitive Computing 7(2), 97. **https://doi.org/10.3390/bdcc7020097**

Győrödi, C. A., Dumşe-Burescu, D. V., Zmaranda, D. R. & Győrödi, R. Ş. 2022.

A Comparative Study of MongoDB and Document-Based MySQL for Big Data

Application Data Management. Big Data and Cognitive Computing, 6(2), 49. **https://doi.org/10.3390/bdcc6020049**