

Holopainen Petri

Tietokantarakenteen opettaminen LLM:lle käyttäen RAG-menetelmää

Insinööri (AMK)

Tieto- ja viestintäteknikka

Kevät 2025



**KAMK • University
of Applied Sciences**

Tiivistelmä

Tekijä(t): Holopainen Petri

Työn nimi: Tietokantarakenteen opettaminen LLM:lle käyttäen RAG-menetelmää

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: AI, LLM, RAG, Fine-tuning, SQL, Tiedonhaku tekoälyn avustuksella

Tämän opinnäytetyön tavoitteena oli opettaa LLM:lle toimeksiantajan eli Flow Invoicing Oy:n tietokantarakenne. Rakenteen opettamisella oli tarkoitus helpottaa työntekijöiden taakkaa, jotta tietokantarakennetta ja siihen liittyviä tietoja ei aina tarvitsisi selittää, jos kysytään LLM:ltä kysymyksiä tietokantarakenteeseen liittyen.

Työssä testattiin LLM:n kykyä hyödyntää RAG-menetelmää. RAG on menetelmänä kevyempi kuin LLM:n hienosäätäminen, mutta RAG ei kuitenkaan sulje pois hienosäätämisen tärkeyttä. Hienosäätämällä saadaan muutettua LLM:n käyttäytymistä ja opetettua sille jokin aihealue, mihin se voi erikoistua. RAG taas antaa pääsyn dataan, jonka avulla LLM pystyy generoimaan oman vastauksen käyttäjän tekemiin prompteihin.

Testatut LLM:t olivat Llama 3.2, 3B-parametrin malli ja Llama 3.3, 70B-parametrin malli. Aluksi testattiin, pystyykö LLM tuottamaan vastauksia AnythingLLM-ympäristön RAG:in avulla. Sen jälkeen verrattiin LLM:ien suoritumista prompteista ja lopuksi testattiin toimeksiantajan tietokannalla Llama 3.3, 70B-parametrin mallia.

Testeistä pystyttiin päättämään, että kielimallit pystyvät hyödyntämään RAG-menetelmää, mutta isommat tietokantarakenteet ovat liikaa sellaisinaan. Rakenteiden pilkkominen LLM:lle RAG:issa olisi voinut auttaa tietokantarakenteen hallinnassa. Hienosäätämisen avulla LLM olisi voinut oppia tarvittavat tiedot tietokannan taulujen merkityksistä.

Abstract

Author(s): Holopainen Petri

Title of the Publication: Teaching database structure for LLM using RAG method

Degree Title: Bachelor of Engineering, Information and Communication Technologies

Keywords: AI, LLM, RAG, Fine-tuning, SQL, AI-assisted data retrieval

The purpose of this thesis was to teach the client's database structure for a large language model (LLM). The purpose of teaching the database structure was to ease the burden on the employees so they would not have to explain anything from the database or its structure when sending prompts to the LLM.

The thesis tested how well a LLM can use the Retrieval-Augmented Generation (RAG) method. Compared to fine-tuning, RAG is a lighter method, but RAG does not exclude the importance of fine-tuning. With fine-tuning, the LLM's behavior can be changed, and it can be taught a domain specific area in which it can specialize. RAG, on the other hand, provides access to data that allows LLM to generate its own response to the user's prompts.

The tested LLMs were Llama 3.2, 3B parameter model and Llama 3.3, 70B parameter model. First, it was tested if the LLM could produce the right answers in the test questions using AnythingLLM's RAG. After that, the performance of the LLMs on the tests was compared. Finally, the client's database was tested with Llama 3.3, 70B parameter model.

From the tests, it could be concluded that language models can utilize the RAG method, but larger database structures are too big as they are. Splitting structures for LLM in RAG could have helped in managing the database structure. And with fine-tuning, the LLM could have learned the necessary information about the meanings of the database tables.

Sisällys

| | | |
|-------|--|----|
| 1 | Johdanto | 1 |
| 2 | LLM | 2 |
| 2.1 | LLM fine-tuning | 2 |
| 2.1.1 | Domain-Specific Fine-Tuning | 3 |
| 2.1.2 | Parameter-Efficient Fine-Tuning (PEFT)..... | 3 |
| 2.2 | Retrieval-Augmented Generation (RAG)..... | 3 |
| 2.3 | Text-to-SQL..... | 4 |
| 2.4 | Prompt engineering..... | 4 |
| 2.4.1 | Zero-Shot prompting | 4 |
| 2.4.2 | Few-Shot prompting | 5 |
| 2.4.3 | Chain-of-Thought prompting | 5 |
| 3 | Toteutusympäristö, -välineet ja -suunnitelma | 6 |
| 3.1 | Ollama | 6 |
| 3.2 | SQL Server Management Studio 20 | 7 |
| 3.3 | Toteutussuunnitelma | 7 |
| 4 | Testaus ja tulokset | 9 |
| 4.1 | Yhden taulun tietokannan testit | 10 |
| 4.2 | Kolmen taulun tietokannan testit | 10 |
| 4.3 | Demotietokannan testit | 11 |
| 5 | Loppupohdinta | 13 |
| 5.1 | Testien tulosten pohdinta | 14 |
| 5.2 | Kehityskohteet | 15 |
| | Lähteet | 17 |

Liitteet

Symboliluettelo

Fine-tuning: Fine-tuning eli hienosäätäminen tarkoittaa kielimallien jatkokouluttamista.

Hallusinointi: Hallusinointi tarkoittaa LLM:n virheellistä tai keksimää tietoa, kun se ei pysty tuottamaan oikeaa vastausta.

LLM: Large Language Model eli suuri kielimalli. Esimerkiksi GPT-4 ja Llama 3 ovat suuria kielimalleja, joita on koulutettu massiivisella määrällä dataa.

Luonnollinen kieli: Natural language eli luonnollinen kieli tarkoittaa ihmisten käyttämää kieltä eikä koneiden keskenään käyttämää kieltä.

Pipeline: Pipeline on ketju prosesseja, jossa data kulkee peräkkäisten käsittelyvaiheiden läpi.

Prompt: Prompt on ohje tai kehote, jonka käyttäjä antaa LLM:lle.

RAG: Retrieval-Augmented Generation on menetelmä, jolla LLM hakee tietoa ulkoisista lähteistä.

SQL: SQL eli Structured Query Language on standardikieli, jolla hallitaan tietokantoja.

Tietokanta: Database eli tietokanta on paikka, mihin kerätään dataa ja missä sitä säilytetään. Tietokanta koostuu tauluista, joissa säilytetään tietoa riveinä ja tauluissa on sarakkeita, jotka kertovat, millaista tietoa milläkin rivillä on.

1 Johdanto

Opinnäytetyön toimeksiantaja Flow Invoicing Oy on yritys, joka tuottaa laskutuspalvelua lämpö-, vesi- ja kaasuyhtiöille. Aloitin heillä työt harjoittelujaksolla lokakuussa 2023. Pääsin opettelemaan SQL Server Management Studiota ja tietokantarakenteen monimutkaisuutta. Välillä rakenteen monimutkaisuus tuntui ylivoimaiselta ja myöhemmin sainkin käyttööni ChatGPT:n yritysversion. Sen avulla kyselyiden luonti muuttui paljon helpommaksi, vaikka joka kerta jouduinkin selittämään tarkasti, millaisen kyselyn halusin ChatGPT:n minulle luovan. Harjoittelu- ja kesätyöjakson aikana pohdimme ratkaisuja helpottamaan arkea. Lopulta totesimme, että olisi hyvä, jos voisimme kehittää oman kielimallin, joka osaisi valmiiksi toimeksiantajan tietokantarakenteen, niin sitä ei joka kerta tarvitsisi erikseen selittää, vaan saataisiin tarvittava tieto heti. Tästä ideasta oli hyvä lähteä liikkeelle ja sitä kautta aiheekseni tulikin tietokantarakenteen opettaminen LLM:lle. Myöhemmin opinnäytetyötä tehdessä löydetään RAG-menetelmä, jonka avulla saadaan syötettyä tietokantarakenne LLM:lle. Haaveena tietokantarakenteen opettamisen jälkeen olisi myös jatkokokehtaa kielimallia ymmärtämään tuotteita, mitä yritys kehittää ja täten auttaa kehittäjiä mahdollisimman paljon työssään.

Tavoitteena opinnäytetyöllä on, että työntekijä pystyisi esittämään kysymyksiä LLM:lle tietokantaan liittyen ja LLM osaisi antaa vastauksena valmiin SQL-kyselyn. Aiheeseen itsessään pystyisi käyttämään aikaa paljon, mutta työssä kuitenkin vain keskitytään tietokantarakenteen opettamiseen ja LLM:ien testaamiseen, jotta työstä ei tulisi liian laaja. Työssä lähdetään liikkeelle ensin tutustumalla vaihtoehtoihin, miten LLM:lle annetaan tietokantarakenne. Sitten kun on löydetty hyvä vaihtoehto, lähdetään testaamaan miten LLM suoriutuu sille esitetyistä kysymyksistä tietokantaan liittyen. Jos malli on oppinut tietokantarakenteen, voidaan lähteä testaamaan isommalla mallilla ja lopulta testata oikealla tietokantarakenteella. Mikäli LLM pystyy antamaan luotettavia vastauksia tietokantaan ja sen rakenteeseen liittyen, toimeksiantaja hyötyy opinnäytetyöstä. Samalla työn tekijänä saan opetella ja tutustua menetelmiin millä LLM:ää saadaan optimoitu. LLM:ien hienosäätö ja muu optimointi on kiehtovaa, mutta myös hyödyllistä hallita tulevaisuutta ajatellen. Ympäristön käyttöönoton kanssa palvelimelle, missä LLM:ää olisi tarkoitus käyttää ja muissa siihen liittyvissä työtehtävissä saan apua paikan päältä ja Microsoft Teamsin välityksellä riippuen siitä, teenkö opinnäytetyötä toimeksiantajan toimistolla vai kotoa käsin.

2 LLM

LLM eli large language model on suuri kielimalli, joka on koulutettu massiivisella määrällä dataa ja sen kanssa voi keskustella luonnollisella kielellä. Kun ihminen kysyy LLM:ltä kysymyksen, LLM pyrkii parhaansa mukaan vastaamaan siihen oikein. LLM:t eivät kuitenkaan ole täydellisiä, vaikka niitä on koulutettu paljon. Ne voivat myös tuottaa vihamielistä puhetta, joka perustuu ihmisen kirjoittamiin teksteihin tai virheellisiä vastauksia, eli hallusinaatioita, jotka yleisesti ottaen johtuvat rakenteettomasta datasta. Näitä haittavaikutuksia voidaan kuitenkin ehkäistä kouluttamalla malleja lisää moninaisilla tavoilla, joista kerrotaan myöhemmin lisää. [1.]

LLM:t, joiden hienosäätämiseen opinnäytetyö pohjautuu, pohjautuvat esikoulutettuihin kielimalleihin. Esikoulutusvaiheessa kielimalli saa valtavan määrän merkitsemättömiä tekstejä, jotta kielimalli pääsee opettelemaan kielen rakenteita. Tämän jälkeen mallia lähdetään kouluttamaan kohti omia tehtäviään hienosäätämällä sitä. Hienosäätämisen vaiheessa aineistoa on vähemmän ja se kohdistuu haluttuun tehtävään. LLM:t käyvät läpi saman esikoulutuksen, mutta ennen hienosäätöä kielimalli sovitetaan yhteen ihmisarvojen kanssa. Tämä mahdollistaa luonnollisemman keskustelun ihmisen ja LLM:n välillä. [2.]

Kielimallia valittaessa on tärkeä ottaa huomioon, mitä mallille on koulutettu ja mihin tehtävään se on tarkoitettu. Hyvän pohjan avulla hienosäätämisestä tulee helpompaa ja säästetään aikaa. Seuraavaksi tulee miettiä, millä tekniikalla hienosäädetään. Tekniikan valinnassa joudutaan ottamaan huomioon, millaisessa ympäristössä malli koulutetaan ja kuinka kauan on aikaa kouluttaa malleja. [3.]

2.1 LLM fine-tuning

LLM fine-tuning eli suuren kielimallin hienosäätäminen on prosessi, jossa valmiiksi koulutettua mallia lähdetään kouluttamaan lisää johonkin tiettyyn asiaan tai aihealueeseen liittyen. Tapoja, joilla kielimalleja voidaan hienosäätää, on monta. Näitä tapoja kuitenkin yhdistää se, että pohjalle tarvitaan kielimalli, jota lähdetään kouluttamaan ja kielimalli tarvitsee aineiston, millä sitä lähdetään kouluttamaan. Koulutuksen aikana kielimallin parametrit päivittyvät. Tavallisin tapa on valvottu hienosäätö (Supervised fine-tuning). Valvotussa hienosäätämisessä kielimallille annetaan merkittävää dataa, jolla mallia opetetaan. [4.]

2.1.1 Domain-Specific Fine-Tuning

Domain-Specific Fine-Tuningissa mallia koulutetaan aineistolla, joka sisältää ala- tai aluekohtaista tietoa. Mallia lähdetään kouluttamaan tällä tapaa yleisesti siinä tilanteessa, kun huomataan, että malli ei suoriudu sille esitetyistä kysymyksistä. Malli ei välttämättä tiedä, miten luodaan yritysraportteja tai miten tehdään kirjanpitoa Suomessa, mutta se voi tietää jotakin aiheesta, mutta aiheesta riippumatta raja tulee jossakin vastaan, mitä malli osaa ja mitä se ei osaa. [2.]

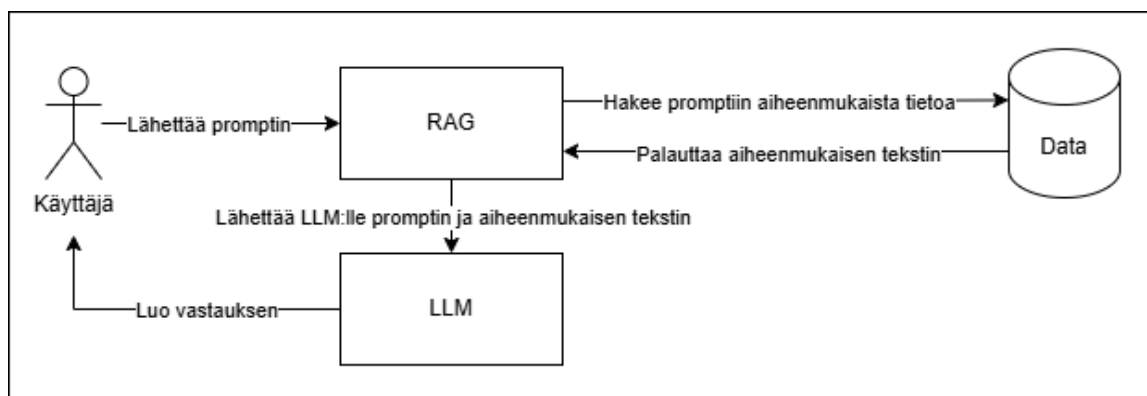
2.1.2 Parameter-Efficient Fine-Tuning (PEFT)

PEFT-tekniikoiden avulla LLM on hienosäädettävissä pienemmillä laskentakustannuksilla. Ideana näillä tekniikoilla on päivittää vain pientä osaa mallin parametreista ja jäädyttää suurin osa esikoulutetun mallin parametreista. Parametrien jäädyttämiseen liittyvän PEFT-tekniikan kautta säävutetaan myös se hyöty, että katastrofaalisen unohtamisen riski pienentyy. Mallin esikoulutettu osa siis pysyy kutakuinkin koskemattomana, kun pyritään opettamaan mallille uutta asiaa. [2.]

2.2 Retrieval-Augmented Generation (RAG)

RAG on LLM:n optimointimenetelmä, jonka avulla saadaan nopeasti syötettyä tietoa LLM:lle ilman erillistä kouluttamista. RAG hakee tietoa sille mahdollistettujen vaihtoehtojen mukaan ja vaihtoehtoja on monia. Esimerkiksi se voidaan yhdistää paikalliseen tietokantaan, tietovarastoon ja vaikka internettiin riippuen siitä, mitä halutaan tehdä LLM:llä. RAG:ille on kuitenkin rakennettava pipelineja, jotta LLM voi käyttää näitä edellä mainittuja tietoja. Kuitenkin RAG:in käyttöä varten on olemassa jo valmiita ratkaisuja, kuten AnythingLLM, jota käytetään tässä opinnäytetyössä. [5.]

Alla olevassa kuvassa 1 on esimerkki, miten RAG toimii. Kuvan käyttäjä käynnistää RAG-prosessin lähettämällä promptin LLM:lle. Ennen kuin viesti menee LLM:lle, RAG käy hakemassa tietoa sille annetuista lähteistä. Näitä lähteitä voivat olla esimerkiksi internet, dokumentit ja tietokanta. Tämän jälkeen RAG lähettää promptin ja kootun aiheenmukaisen tekstin LLM:lle. Tekstin saatuaan LLM luo vastauksen käyttäjän promptiin käyttäen RAG:iltä saamaansa tietoa ja omaa tietoaan, jonka se on saanut koulutuksessaan.



Kuva 1. Malli RAG:in toiminnasta.

2.3 Text-to-SQL

Text-to-SQL on työkalu, jolla luonnollinen kieli käännetään SQL-kyselyksi. Voidaan kysyä, miten haetaan tietokannan tauluista tietoa, kun tiedetään, millainen tietokannan rakenne on. Esimerkiksi voidaan kysyä eri tauluista tietoa ja jos taulut liittyvät toisiinsa voidaan kertoa tauluja yhdistävä tieto, jonka avulla Text-to-SQL saa generoitua meille oikeanlaisen kyselyn. [6.]

2.4 Prompt engineering

Prompt engineering eli kehotteen tai ohjeen suunnittelu on yksi tapa parantaa kielimallin suoriutumista promptiin vastaamisessa. Prompt engineering ei vaadi kielimallin kouluttamista ja sen tavoitteena on auttaa kielimallia tuottamaan tarkempia tuloksia muuttamalla tapaa, miten kerromme asian kielimallille. Promptin suunnittelu on tärkeä osa LLM:ien kanssa keskusteluun, jotta LLM osaa vastata kysymykseen halutulla tavalla. Promptin voi suunnitella monella tapaa ja alla mainitaan yleisimmät prompt engineering -tekniikat. [7.]

2.4.1 Zero-Shot prompting

Ideana Zero-Shot promptingissa on antaa kielimallille prompt ilman tarkempaa ohjeistusta. Sen sijaan kielimalli päättelee itse yleistietoaan hyödyntäen ja antaa promptiin vastauksen. Zero-Shot promptingin ideana on tehdä tehtävä ilman esimerkkejä ja turvautua pelkästään kielimallin

omaan tietoon. Erikoiset ja monimutkaiset tehtävät voivat kuitenkin tuottaa kielimallille haasteita, jolloin kannattaa turvautua erilaiseen lähestymistapaan. [8.]

2.4.2 Few-Shot prompting

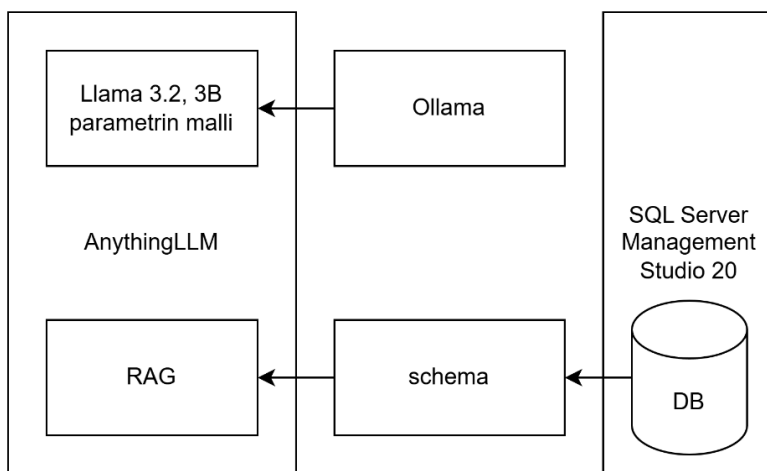
Few-Shot promptingin idea perustuu samaan perusajatukseen kuin Zero-Shot promptingissa. Few-Shot eroaa kuitenkin Zero-Shotista niin, että mallille annetaan muutama esimerkki aiheeseen liittyen, jotta saadaan malli tuottamaan tietyn tyyppisiä vastauksia. Malli ei kuitenkaan välttämättä opi asiaa muutamalla esimerkillä, mikäli se vaatii jotakin erikoistaitoa, -tyyliä tai jonkin tietyn rakenteen ymmärtämistä. Mikäli malli ei ymmärrä, mitä siltä halutaan, mallin kouluttaminen tai hienosäätäminen aineistolla voi ratkaista tämän ongelman. [9.]

2.4.3 Chain-of-Thought prompting

Chain-of-Thought promptingissa ideana on käydä tehtävä pala palalta. Jos halutaan ratkaista jotakin monimutkaista, voidaan antaa esimerkkinä vastaavan kysymys-vastaus-yhdistelmän, jonka avulla kielimalli saa ratkaistua monimutkaisempia kysymyksiä. Kun esitetään kysymys ja vaihe vaiheelta, kuinka saadaan oikea vastaus, saadaan selitettyä kielimallille, miten asia kuuluu ratkaista. Tämän jälkeen voimme kysyä samantyyppisen kysymyksen, johon kielimalli osaa antaa oikeanlaisen vastauksen. Kielimallin suoriutuminen parantuu, koska pakotamme sen käyttämään välivaiheita ja käymään asian läpi askel askeleelta. [10.]

3 Toteutusympäristö, -välineet ja -suunnitelma

Toteutuksessa käytettiin ympäristönä AnythingLLM:ää, missä LLM on käytössä. AnythingLLM on käyttöliittymä LLM:lle ja se tarjoaa käyttöliittymän lisäksi mahdollisuuden hallita LLM:ää paremmin, usealla eri tavalla, joista käytetään RAG-menetelmää. Muita työkaluja olivat SQL Server Management Studio 20 ja Ollama. SQL Server Management Studio 20:stä ja Ollamasta myöhemmin lisää. Alla olevassa kuvassa 2 kuvataan arkkitehtuuria yksinkertaisella tasolla.



Kuva 2. Esimerkkikuva tietokannan ja LLM:n kytköksistä.

3.1 Ollama

Ollama on ympäristö, jossa voidaan käyttää ja ladata erilaisia LLM:iä. Ollaman avulla opinnäytetyössä ladattiin Llama 3.2, 3B-parametrin malli ja Llama 3.3, 70B-parametrin malli. Llama 3.2 pyöri lokaalisti kannettavalla tietokoneella ja Llama 3.3 toimiston palvelimella. Alla taulukossa 1 kerrotaan ympäristöt, joissa LLM:t testattiin.

| | Kannettava tietokone | Toimiston palvelin |
|---------------|------------------------------|---------------------------------|
| Proessori | Intel Core i7-10750H 2.60GHz | Intel Xeon W-1275 2.50 GHz |
| Näytönohjain | NVIDIA GeForce RTX 2070 | Nvidia Quadro P2000 Direct X 12 |
| Keskusmuisti | 15,8 Gt | 94,2 Gt |
| Tallennustila | 475 Gt | 250 Gt |

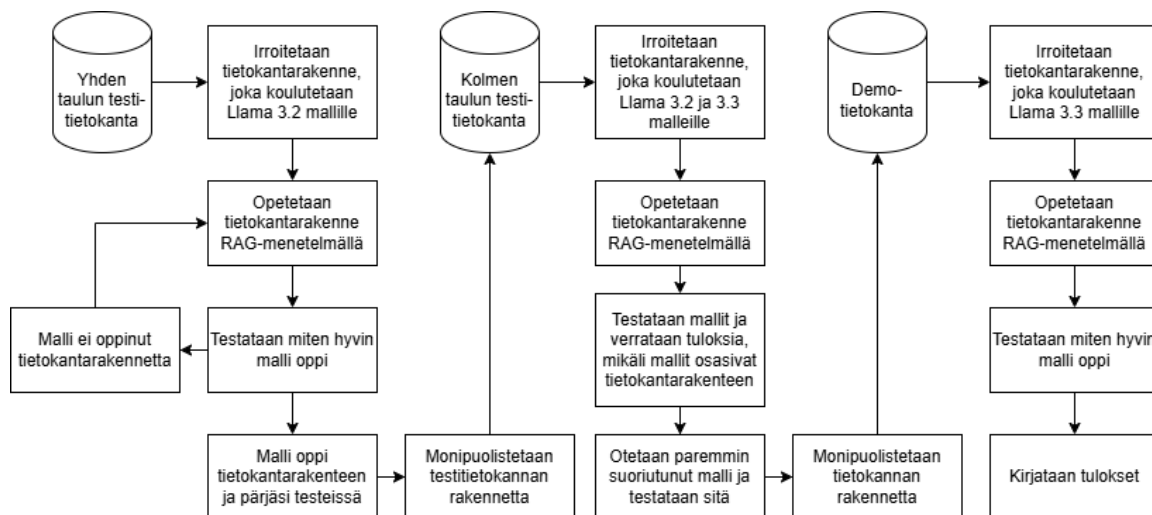
Taulukko 1. Toteutusympäristöt testeille.

3.2 SQL Server Management Studio 20

SQL Server Management Studio 20 on ympäristö, jossa hallitaan SQL-tietokantoja. Ennen testauksen aloittamista, SQL Server management Studio 20:een luotiin yhden taulun tietokanta, jonne lisättiin yksi taulu. Tähän tauluun luotiin neljä saraketta, jotka olivat asiakas, asiakasnumero, maa ja puhelin. Yhden taulun tietokannan testaamisen jälkeen SQL Server management Studio 20:een luotiin uusi tietokanta, josta myöhemmin puhutaan kolmen taulun tietokantana. Uuteen tietokantaan lisättiin yhden taulun tietokannan sisällön lisäksi kaksi taulua lisää. Taulujen nimet olivat kohteet ja sopimukset. Kohteet-taulu sisältää sarakkeet: kohdenumero, sopimusnumero, kohteenosoite ja mittarilukema. Sopimukset-taulu taas sisältää sarakkeet: sopimusnumero, asiakasnumero, sopimuksennimi, alkamispvm ja paattymispvm. Kolmen taulun testien jälkeen viimeisenä testattiin demotietokanta. Demotietokanta tuotiin samaan ympäristöön eli SQL Server management Studio 20:een missä muutkin LLM:t testattiin. Demotietokanta siirrettiin samaan ympäristöön, jotta saataisiin mahdollisimman vertailukelpoisia tuloksia, mutta salassapitosopimuksen vuoksi emme kuitenkaan voi lähteä kertomaan enempää demotietokannasta ja sen rakenteesta. Liitteessä 1 kuvataan tietokantarakenne ja tiedot mitä LLM sai tietokannoista, mutta LLM:lle tiedot syötettiin CSV:nä taulukon sijaan.

3.3 Toteutussuunnitelma

Koulutusvaiheessa SQL Server management Studio 20:een luotiin testitietokannat. Tietokantojen rakenteet irrotettiin LLM:lle SQL-kyselyllä, joka tallennettiin CSV-muodossa. Tämä CSV-tiedosto taas syötettiin AnythingLLM:ään käyttäen RAG-menetelmää. Tämän jälkeen pystyttiin testaamaan LLM:ää, jolle on annettu tietokannan rakenne. Jokaisen testitietokannan testien jälkeen, siirryttiin isompaan tietokantaan ja toistettiin tämä prosessi. Alla kuvassa 3 näytetään testaus-suunnitelma.



Kuva 3. Tietokantojen testaussuunnitelma.

4 Testaus ja tulokset

Testikysymyksiä on 12 kappaletta. Niillä testataan LLM:n käytettävyyttä ja testikysymykset esitetään kielimallille suomeksi ja englanniksi. Opinnäytetyön testikysymykset löytyvät liitteestä 2, ja vastaukset kysymyksiin liitteistä 3 ja 4. Testikysymykset on koottu ajatellen tietokantaa, ja osassa testeistä on tarkoituksellisesti kysytty kertomatta tarkkaa kontekstia, että saadaan simuloitua peruskäyttäjän esittämiä kysymyksiä. Kysymykset on analysoitu käyttäen eri näkökantoja. Ensimmäisenä ”Minimin pitää toteutua”, toisena ”Minimi toteutuu tai prompt on puutteellinen” ja kolmantena ”Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein”. Näillä testeillä on tarkoituksena saada selville, miten hyvin voidaan luottaa tuloksiin ja mitä käyttäjän pitää ottaa huomioon käyttäessään LLM:ää.

Tarkemmin läpikäytynä ensimmäiseksi testataan, miten LLM suoriutuu minimitavoitteestaan eli minimistä. Minimillä tarkastellaan, saako LLM tuotettua oikeita vastauksia, joilla haluttu pyyntö toteutuu. ”Minimi toteutuu tai prompt on puutteellinen” kohdassa tarkastellaan, onko promptissa kerrottu kaikki tarvittava. Jos LLM olisi mahdollisesti saanut toteutettua pyynnön lisätietojen avulla tai sai sen oikein, lasketaan vastaus onnistuneeksi. Tarvittavista lisätiedoista voidaan antaa esimerkkinä, että LLM ei saanut tietoa taulusta, mihin sarakkeeseen se kuuluu, mutta jos LLM:lle olisi kerrottu sijainti, mihin taulun sarakkeeseen tieto tulisi syöttää, kysely olisi mennyt oikein. Viimeisenä tarkastellaan kohdan ”Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein”, kertooko LLM vastatessaan promptiin, mitä se on tehnyt ja voiko peruskäyttäjä sen pohjalta ymmärtää, että LLM:n antama vastaus on väärin.

Alla olevassa taulukossa 2 käydään läpi edellä mainitut asiat. Lisäksi kuvassa kerrotaan testien tulokset prosentteina suomeksi vai englanniksi. Lopuksi kahdessa viimeisessä sarakkeessa näytetään, mitä mallia on käytetty ja miten hyvin malli on suoriutunut testeistä prosentuaalisesti. Jos testeissä ei ole mainittu tuloksia, tietokantaa ei ole testattu kyseisellä mallilla.

| | Testauskieli | Llama 3.2, 3B | Llama 3.3, 70B |
|---|--------------|---------------|----------------|
| Minimin pitää toteutua | Suomi | - | - |
| Minimin pitää toteutua | Englanti | - | - |
| Minimi toteutuu tai prompt on puutteellinen | Suomi | - | - |
| Minimi toteutuu tai prompt on puutteellinen | Englanti | - | - |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Suomi | - | - |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Englanti | - | - |

Taulukko 2. Testin esimerkkikuva, jossa kuvataan mallien suoriutumista.

4.1 Yhden taulun tietokannan testit

Yhden taulun testeissä LLM pystyi kertomaan tietokantarakenteen ongelmitta. Molemmissa eli suomen- ja englanninkielisissä vastauksista kävi hyvin ilmi taulut ja niiden sarakkeet, mitä tietokannassa oli sisällä. LLM vastaa myös hyvin prompteihin, joissa ohjeet ovat mahdollisimmat yksiselitteiset. Promptit, joissa LLM joutuu päättämään tietokantatauluja, nostavat virheiden riskiä suuresti. Testeissä tämä käy hyvin ilmi kysymyksissä kolme ja neljä (liite 2), missä puhutaan ”Asiakas 101” -päivityksistä. Prompteissa, joissa puhutaan ”Asiakas 101”, viitataan asiakas sarakkeeseen ja LLM luulee, että puhutaan asiakasnumerosta, joka on 101. Muissa kysymyksissä on myös havaittavissa hallusinoitua. LLM antaa vastauksissaan taulujen nimiä, joita ei ole olemassa tai ne sisältävät outoja taivutuksia. Liitteestä 3 voi tarkastella yhden taulun testien kysymyksiä lisää. Alla taulukossa 3 tulokset, miten hyvin LLM suoriutui yhden taulun testeistä.

| | Testauskieli | Llama 3.2, 3B | Llama 3.3, 70B |
|---|--------------|---------------|----------------|
| Minimin pitää toteutua | Suomi | 42 % | - |
| Minimin pitää toteutua | Englanti | 50 % | - |
| Minimi toteutuu tai prompt on puutteellinen | Suomi | 67 % | - |
| Minimi toteutuu tai prompt on puutteellinen | Englanti | 92 % | - |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Suomi | 75 % | - |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Englanti | 83 % | - |

Taulukko 3. Yhden taulun tietokannan testien tulokset.

4.2 Kolmen taulun tietokannan testit

Kolmen taulun testeissä vertailtiin Llama 3.2, 3B-parametrin mallia ja Llama 3.3, 70B-parametrin mallia. Kun malleja lähdettiin testaamaan, oli huomattavaa, miten suuri nopeusero kielimallien välillä oli. Llama 3.2 sai luotua vastauksen kysymyksiin muutamissa sekunneissa toisin kuin Llama 3.3. Llama 3.3:lla yhteen kysymykseen vastatessa meni aikaa hyvin vaihtelevasti minuutista yli viiteen minuuttiin. Testien tuloksia läpikäydessä Llama 3.2 oli selvästi heikompi suomenkielisissä testeissä kuin Llama 3.3, mutta englanniksi erot näiden kahden mallin välillä kuroutuivat pienemmiksi. Llama 3.3 pystyi molemmilla kerroilla kertomaan tietokantarakenteen, mutta englanniksi se oli tehnyt kirjoitusvirheen kirjoittaessaan taulun sarakkeen nimeä. Llama 3.2:lta jäi taulun sarakkeita kirjoittamatta tai se hallusinoi tauluja ja sarakkeita lisää. Kysymykset 3 ja 4 (liite 2) aiheuttivat eniten ongelmia malleille. Mallit joutuivat päättämään, mihin tauluun ja sarakkeeseen

”Asiakas 101” kuuluu, eikä kumpikaan malli pystynyt ennustamaan, että ”Asiakas 101” kuuluu asiakkaat-aulun asiakas-sarakkeeseen. Mallit muuten olivat vastaamassa oikein prompteihin. Malleilla oli myös vaikeuksia vastata kysymyksiin 8 ja 10 (liite 2). Llama 3.2 ei pystynyt vastaamaan kysymykseen 8 ollenkaan oikein ja kysymykseen 10 se vastasi englanniksi melkein oikein väärillä taulun nimillä. Llama 3.3 kuitenkin pystyi antamaan oikean vastauksen, mutta taulujen nimet eivät aina olleet oikein. Muut kysymykset olivat riittävän helppoja, että mallit pystyivät vastaamaan niihin oikein. Llama 3.2 tuotti vastauksensa vaikealukuisina verrattuna Llama 3.3. Tämä näkyi paremmin suomenkielisissä vastauksissa ja tuotettu kieli oli ajoittain epäselvää, mitä Llama 3.2 yritti sanoa. Tästä hyvä esimerkki on suomeksi esitetty kysymys 8. Liitteestä 4 voi tarkastella kolmen taulun testien vastauksia lisää. Alla olevasta taulukosta 4 voidaan vertailla tuloksia, miten hyvin mallit suoriutuivat kolmen taulun testeistä.

| | Testauskieli | Llama 3.2, 3B | Llama 3.3, 70B |
|---|--------------|---------------|----------------|
| Minimin pitää toteutua | Suomi | 58 % | 75 % |
| Minimin pitää toteutua | Englanti | 58 % | 58 % |
| Minimi toteutuu tai prompt on puutteellinen | Suomi | 75 % | 92 % |
| Minimi toteutuu tai prompt on puutteellinen | Englanti | 92 % | 100 % |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Suomi | 75 % | 92 % |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Englanti | 83 % | 92 % |

Taulukko 4. Kolmen taulun tietokannan testien tulokset.

4.3 Demotietokannan testit

Demotietokannan rakenne poikkeaa aiemmista tietokannoista, minkä takia testikysymykset eivät ole täysin samat. Kysymyksiä jouduttiin muuttamaan vastaamaan tietokantarakennetta. Suurin muutos kysymyksissä oli taulujen ja sarakkeiden nimien muuttaminen. Tämän lisäksi vain syötettävään tietoon tehtiin muutos, mikäli se oli erityyppinen, kuin aiemmissa testeissä. Kuitenkin kysymykset pysyivät niin samanlaisina, että testeissä voimme käyttää aiemmin käytettyjä testikysymyksiä esimerkkeinä demotietokannan testikysymyksistä.

Testattaessa tietokantaa, kun pyydettiin tietokantarakennetta, tietokanta oli liian suuri Llama 3.3:lle. LLM antoi vastauksessaan vain muutaman taulun, joissa oli oikeat sarakkeet. Kysymykset, joissa kielimallilta kysyttiin taulukohtaisesti pyyntöjä, joissa ohjeet olivat kattavat, onnistuivat.

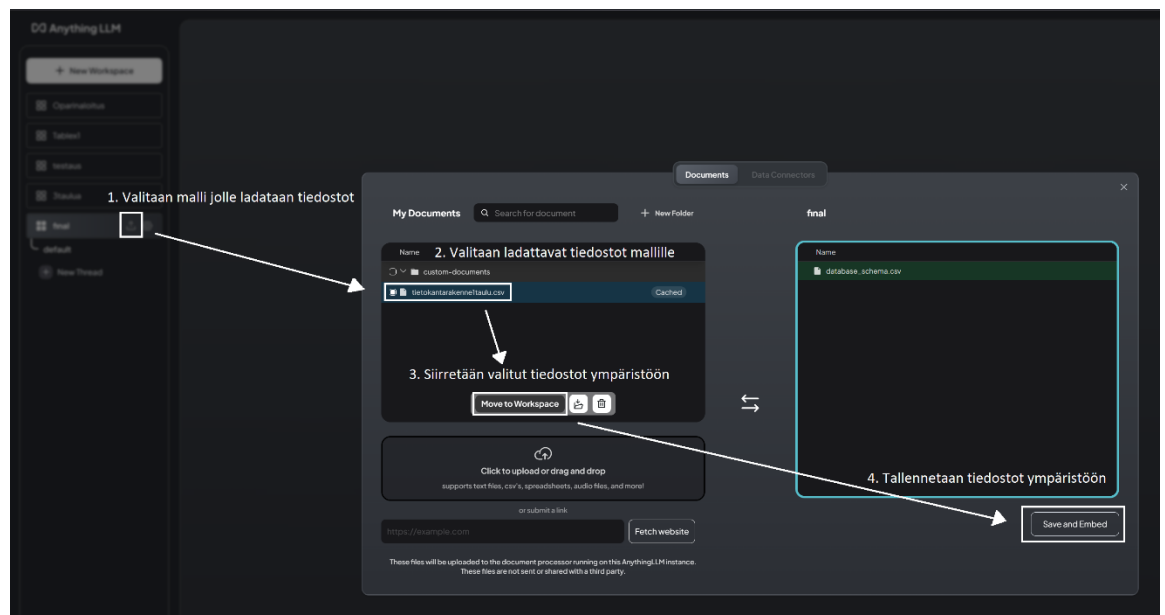
Muut kysymykset, joissa ei ollut annettu taulua tai muokattavaa kenttää, epäonnistuivat. Alla taulukossa 5 kerrotaan, miten hyvin LLM suoriutui demotietokannan testeistä.

| | Testauskieli | Llama 3.2, 3B | Llama 3.3, 70B |
|---|--------------|---------------|----------------|
| Minimin pitää toteutua | Suomi | - | 50 % |
| Minimin pitää toteutua | Englanti | - | 67 % |
| Minimi toteutuu tai prompt on puutteellinen | Suomi | - | 75 % |
| Minimi toteutuu tai prompt on puutteellinen | Englanti | - | 92 % |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Suomi | - | 75 % |
| Minimi toteutuu tai LLM vastauksesta käy ilmi, että vastaus ei ole oikein | Englanti | - | 92 % |

Taulukko 5. Demotietokannan testien tulokset.

5 Loppupohdinta

Mallien hienosäätäminen on yleisesti ottaen raskas prosessi, johon menee paljon aikaa. Toisin kuin hienosäätäminen, RAG-menetelmä tuntui nopealta ja helpolta tavalta suorittaa tietokantarakenteen opettaminen LLM:lle. RAG-menetelmän helppous piilee siinä, että LLM:ää ei tarvitse lähteä jatkokouluttamaan erikseen sille tarkoitettuun tehtävään. LLM:lle tarvitsee vain antaa tiedosto koskien koulutuskohdetta ja LLM pystyy tämän jälkeen tuottamaan paremmin haluttua sisältöä. RAG-menetelmän ansiosta, kun LLM:lle kirjoitetaan prompt, mitä halutaan, ja RAG noutaa LLM:lle syötetyistä tiedostoista vastauksia promptiin. Alla kuvassa 4 kerrotaan vaiheista, miten kielimallille syötetään tiedosto.



Kuva 4. Kuinka kielimallille syötetään tiedosto?

RAG vaatii kuitenkin pohjalle valmiiksi koulutetun LLM:n, ja yleensä isommilla LLM:illä päästään pidemmälle kuin pienillä LLM:illä. Paikallisten LLM:ien käyttö rajoittuu voimakkaasti ympäristön takia, jossa LLM:ää ylläpidetään. Perustietokoneen kanssa, jossa ei ole enempää kuin 16 G RAM-muistia, voidaan käyttää vain pienempiä malleja, kuten LLama 3.2, 3B-parametrin mallia. Isommat mallit vaativat yleensä enemmän muistia, mutta vastaukset ovat yleensä parempia, koska LLM pystyy prosessoimaan isompia kokonaisuuksia ja sitä on koulutettu pidemmälle. Testatessa tämä ilmeni sillä, että suomen kieli parani huomattavissa määrissä LLama 3.2, 3B-parametrin mallin ja LLama 3.3, 70B-parametrin mallin välillä. Myös vastaukset, joita saimme LLama 3.3, 70B-pa-

rametrin mallilta, menivät todennäköisemmin oikein kuin Llama 3.2, 3B-parametrin mallin vastaukset. Llama 3.3, 70B-parametrin mallin vastauksia joutui tosin odottamaan kauan aikaa verrattuna Llama 3.2, 3B-parametrin malliin.

Opinnäytetyö ei kuitenkaan mennyt osaltani hukkaan, vaikka toimeksiantaja nyt ei saanutkaan mitä tilasi. Opin paljon LLM:ien hienosäätämisestä, RAG:istä ja niiden rajoista. RAG oli uusi asia minulle, vaikka tavallaan tiesin mitä se tekee tietämättä, miten LLM tekee hakuja esimerkiksi internettiin. Testaamista olisi voinut tehdä RAG:illä enemmän kuin mitä tein. Jos lähtisin uudestaan yrittämään opinnäytetyötä, aloittaisin testaamaan demotietokantarakenteella, miten hyvin Llama 3.2, 3B-parametrin malli olisi selvinnyt testeistä, kun muutetaan tapaa, miten RAG:illä haetaan tietokantarakenne. RAG-menetelmän testaamisen jälkeen otetaan uusin Llama versio käyttöön suurimmalla parametrimäärällä mitä voidaan käyttää ja otetaan parhaimmin testeissä pärjännyt RAG-menetelmä. Jos tämä ei tuottaisi riittävän hyviä tuloksia, niin sen jälkeen lähdetäisiin hienosäätämään mallia.

5.1 Testien tulosten pohdinta

Yhden taulun testeissä ilmeni ongelma Llama 3.2, 3B-parametrin mallin päättämisen kanssa. Tietokannan taulujen heikko päättely vaikutti tuloksiin negatiivisesti ja suurin osa vastauksista meni tämän takia väärin. Tuloksien avulla nähdään, että promptit olivat vaillinaisia ja LLM olisi tarkkojen ohjeiden avulla saanut vastattua oikein suurimpaan osaan kysymyksistä englanniksi. Suomeksi kysymyksiin vastaaminen osoittautui haastavaksi ja välttämättä hyväkään promptti ei olisi taannut, että LLM olisi tuottanut oikeita vastauksia. Myös suomen kielen osaaminen Llama 3.2, 3B-parametrin mallilla oli heikkoa, joka osakseen vaikutti vastausten heikkoon laatuun.

Heikko suoriutuminen johti toisen mallin valintaan ja mukaan tuli Llama 3.3, 70B-parametrin malli, kun lähdettiin testaamaan kolmella taululla. Kolmen taulun testeissä ilmeni paremmin kielellisten taitojen erot LLM:ien välillä. Llama 3.2, 3B-parametrin mallin suoriutuminen oli selvästi heikompaa, kun tarkasteltiin pelkkiä tuloksia. Myös vastauksista kävi ilmi, että Llama 3.3, 70B-parametrin malli tuotti huomattavasti parempaa suomen kieltä, kuin Llama 3.2, 3B-parametrin malli. Llama 3.3, 70B-parametrin malli osoitti myös olevansa luotettavampi. Pääsääntöisesti virheet johtuivat Llama 3.3, 70B-parametrin mallilla kirjoitusvirheistä, lievistä hallusinoinnista tai

pienistä päättelyvirheistä. Taulujen paloittelu, auki selittäminen ja tarkat tehtäväkuvaukset olisivat auttaneet LLM:ää tekemään parempia vastauksia.

Demotietokantaa testattaessa testikysymyksistä oli huomattavissa, että tietokantarakenne oli liian suuri LLM:lle. Tämä aiheutti muissa testeissä haasteita ja tarve selittää tietokantarakennetta lisääntyi. LLM ei siis pystynyt hyödyntämään tietokantarakennetta RAG:in avulla halutulla tavalla. LLM hyödynsi RAG:iä, kun kielimallille oli annettu valmiiksi taulu tai tieto, mitä saraketta ollaan muuttamassa. Viimeistään tässä vaiheessa taulujen paloittelu ja selittäminen LLM:lle olisi tärkeää, jotta LLM olisi saanut hyödynnettyä tietokantarakennetta oikein.

Testien tulosten perusteella LLM ei siis pystynyt hyödyntämään tietokantarakennetta riittävän hyvin. RAG hakee vain osan tietokantarakenteesta, kun pyydetään koko rakennetta tai jos pyydetään syöttämään asiakastieto tietokantaan, syötetään tieto johonkin kenttään, joka sisältää samanoloisen kentän eikä välttämättä olla edes asiakas-taulussa. Vasta kun kerrottiin mistä tauluista ja kentistä oli kysymys, LLM sai tuotettua oikeanlaisia vastauksia. Toisin sanoen työ epäonnistui tuottamaan haluttuja tuloksia. Kielimalli saatiin asiakkaan palvelimelle, mutta se on hidas ja toimii huonosti. Työskentely opinnäytetyön loppupuolella hidastui merkittävästi, koska isompaa LLM:ää ei pystynyt testaamaan paikallisesti. LLM:lle pitäisi tehdä vielä paljon muutoksia RAG:iin tai kouluttaa LLM:ää eteenpäin, jotta LLM tuottaisi haluttuja SQL-kyselyitä.

5.2 Kehityskohteet

Tietokantarakennetta tulisi pilkkoa sitä syötettäessä LLM:lle. RAG on menetelmänä toimiva, mutta tapa, jolla sitä opinnäytetyössä hyödynnettiin, ei sovellu isoille tietokantarakenteille. Pilkkottuna tietokantarakenne olisi ollut LLM:lle helpompi käsitellä. Myös mallin hienosäätäminen voisi ratkaista ongelmia. Mallille olisi voinut opettaa hienosäätämällä, miten tietokantaa tulisi tulkita ja millaista tietoa lämpö- vesi- ja kaasuyhtiöt pitävät sisällään. Näiden vaihtoehtojen lisäksi muokkaamalla asetuksista löytyvää promptia, voisimme selittää LLM:lle, että se käyttää sille syötettyjä tiedostoja. Tämän muutoksen avulla LLM pystyisi paremmin hakemaan ja hyödyntämään tietoja, mitä sille on syötetty RAG:iin. Alla kuvassa 5 näkyy asetusten paikka, josta saadaan muutettua LLM:n tapaa hakea tietoja.

AnythingLLM

+ New Workspace

Oparinaloitus

Tablex1

1. Mennään asetuksiin

testaus

default

1. Kielimallin pitää pysty...

2. Lisää uusi rivi tauluu...

3. Päivitä olemassa oleva...

4. Poista tietty rivi ehd...

5. Hae kaikki rivit taulu...

6. Hae tietyt sarakkeet, ...

7. Järjestä tulokset tiet...

8. Laske tietyn taulun ri...

9. kahden taulun liittämi...

10. Käytä alikyselyitä SE...

11. Virheellisten SQL-kys...

1. The language model sho...

2. Add a new row to the t...

General Settings Chat Settings Vector Database Agent Configuration

Workspace LLM Provider
The specific LLM provider & model that will be used for this workspace. By default, it uses the system LLM provider and settings.

Ollama
Run LLMs locally on your own machine.

Workspace Chat model
The specific chat model that will be used for this workspace. If empty, will use the system LLM preference.

llama3.2:3b

Chat mode

Chat Query

Chat will provide answers with the LLM's general knowledge *and* document context that is found.

Chat History
The number of previous chats that will be included in the response's short-term memory. *Recommend 20.* Anything more than 45 is likely to lead to continuous chat failures depending on message size.

20

2. Kirjoitetaan ohjeet mallille

Prompt
The prompt that will be used on this workspace. Define the context and instructions for the AI to generate a response. You should to provide a carefully crafted prompt so the AI can generate a relevant and accurate response.

Given the following conversation, relevant context, and a follow up question, reply with an answer to the current question the user is asking. Return only your response to the question given the above information following the users instructions as needed.

Kuva 5. AnythingLLM:n asetukset, joissa voidaan säätää promptia.

Lähteet

1. What are large language models (LLMs)?. IBM. [Internet]. 2.11.2023. [viitattu 1.10.2024]. Saatavilla: <https://www.ibm.com/topics/large-language-models>
2. Parthasarathy V, Zafar A, Khan A, Shahid A. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. arXiv. [Internet] 23.8.2024. [viitattu 2.10.2024]. Saatavilla: <https://arxiv.org/html/2408.13296v1>
3. Raj M, VM K, Warriar H, Gupta Y. Fine-tuning LLMs for Enterprise: Practical Guidelines and Recommendations. arXiv. [Internet]. 23.3.2024. [viitattu 2.10.2024]. Saatavilla: <https://arxiv.org/pdf/2404.10779>
4. Ferrer J. An Introductory Guide to Fine-Tuning LLMs. datacamp. [Internet]. 1.8.2024 [viitattu 3.10.2024] Saatavilla: <https://www.datacamp.com/tutorial/fine-tuning-large-language-models>
5. Belcic Ivan, Stryker Cole. RAG vs. fine-tuning. IBM. [Internet]. 14.8.2024 [viitattu 9.5.2025] Saatavilla: <https://www.ibm.com/think/topics/rag-vs-fine-tuning>
6. Rebei A. Fine-Tuning Language Models for Context-Specific SQL Query Generation. arXiv. [Internet]. 27.2.2024. [viitattu 1.10.2024]. Saatavilla: <https://arxiv.org/html/2312.02251v1>
7. What is Prompt Engineering - Meaning, Working, Techniques. GeeksForGeeks. [Internet]. 4.4.2025 [viitattu 19.5.2025] Saatavilla: <https://www.geeksforgeeks.org/what-is-prompt-engineering-the-ai-revolution/>
8. Ramlochan Sunil. Zero-Shot Prompting: A Powerful Technique for LLMs. Prompt Engineering & AI Institute. [Internet]. 23.7.2024. [viitattu 20.11.2024]. Saatavilla: <https://promptengineering.org/zero-shot-prompting-a-powerful-technique-for-llms/>
9. Ramlochan Sunil. Master Prompting Concepts: Zero-Shot and Few-Shot Prompting. Prompt Engineering & AI Institute. [Internet]. 25.4.2023. [viitattu 21.11.2024]. Saatavilla: <https://promptengineering.org/master-prompting-concepts-zero-shot-and-few-shot-prompting/>

10. Ramlochan Sunil. Master Prompting Concepts: Chain of Thought Prompting. Prompt Engineering & AI Institute. [Internet]. 26.4.2023. [viitattu 21.11.2024]. Saatavilla: <https://promptengineering.org/master-prompting-concepts-chain-of-thought-prompting/>

taulu x 1

| TableName | ColumnName | ColumnDataType | MaxLength | NumericPrecision | NumericScale | IsNullable | IsIdentity | IsPrimaryKey | DefaultConstraint Name |
|-----------|---------------|----------------|-----------|------------------|--------------|------------|------------|--------------|------------------------|
| asiakkaat | asiakas | nvarchar | 100 | 0 | 0 | 1 | 0 | 0 | NULL |
| asiakkaat | asiakasnumero | int | 4 | 10 | 0 | 0 | 0 | 1 | NULL |
| asiakkaat | maa | nvarchar | 100 | 0 | 0 | 1 | 0 | 0 | NULL |
| asiakkaat | puhelin | nvarchar | 100 | 0 | 0 | 1 | 0 | 0 | NULL |

taulu x 3

| Schema | Table | Column | Data Type | Max Length | Is Nullable | Key Type | Referenced Table | Referenced Column |
|--------|------------|----------------|-----------|------------|-------------|-------------|------------------|-------------------|
| dbo | asiakkaat | asiakas | varchar | 50 | YES | None | NULL | NULL |
| dbo | asiakkaat | asiakasnumero | int | NULL | NO | Primary Key | NULL | NULL |
| dbo | asiakkaat | maa | nvarchar | 50 | YES | None | NULL | NULL |
| dbo | asiakkaat | puhelin | nvarchar | 50 | YES | None | NULL | NULL |
| dbo | kohteet | kohdenumero | int | NULL | NO | Primary Key | NULL | NULL |
| dbo | kohteet | sopimusnumero | int | NULL | NO | Foreign Key | sopimukset | sopimusnumero |
| dbo | kohteet | kohteenosoite | nvarchar | 100 | YES | None | NULL | NULL |
| dbo | kohteet | mittarilukema | float | NULL | YES | None | NULL | NULL |
| dbo | sopimukset | sopimusnumero | int | NULL | NO | Primary Key | NULL | NULL |
| dbo | sopimukset | asiakasnumero | int | NULL | NO | Foreign Key | asiakkaat | asiakasnumero |
| dbo | sopimukset | sopimuksennimi | nvarchar | 100 | YES | None | NULL | NULL |
| dbo | sopimukset | alkamispvm | date | NULL | YES | None | NULL | NULL |
| dbo | sopimukset | paattymispvm | date | NULL | YES | None | NULL | NULL |

Kysymykset kielimallille

FI

Kysymysten luonti:

1. Kielimallin pitää pystyä näyttämään tietokannan rakenne.

Näytä minulle SQL-tietokannan rakenne.

2. Lisää uusi rivi tauluun (INSERT).

Haluaisin lisätä nämä tiedot tietokantaan, voisitko luoda SQL-kyselyn: asiakas = Asiakas 101, asiakasnumero = 1101, maa = Suomi, puhelin = +358 40 123 45 101

3. Päivitä olemassa oleva rivi (UPDATE).

Tee minulle SQL-kysely, jolla voin päivittää Asiakas 101 maan Ruotsiksi.

4. Poista tietty rivi ehdon perusteella (DELETE).

Haluaisin poistaa Asiakas 101 tietokannasta. Voisitko tehdä minulle koodin, jolla saan tehtyä sen?

5. Hae kaikki rivit taulusta. (SELECT)

Tee minulle SQL-kysely, jossa haetaan kaikki rivit taulusta Asiakkaat.

6. Hae tietyt sarakkeet, joissa on ehto. (WHERE)

Tee minulle SQL-kysely, jossa haetaan kaikki rivit taulusta Asiakkaat ja joissa on maa Suomi.

7. Järjestä tulokset tietyn sarakkeen mukaan (ASC/DESC).

Haluaisin SQL-kyselyn, jolla voin järjestellä asiakkaat taulun nousevassa järjestyksessä.

Haluaisin SQL-kyselyn, jolla järjestellään asiakkaat taulun laskevassa järjestyksessä.

8. Laske tietyn taulun rivien määrä (COUNT).

Haluaisin tietää, kuinka monen asiakkaan puhelinnumero päättyy numeroon 1.

9. kahden taulun liittäminen yhteen. (JOIN)

Haluaisin hakea tietokannasta listan asiakkaista, joilla on sopimus ja listan heidän puhelinnumeroitaan.

10. Käytä alikyselyitä SELECT-, FROM- ja WHERE-lauseissa.

Haluaisin löytää kaikki asiakkaat, joiden maa on Ruotsi ja järjestellä ne puhelin kentän mukaan.

11. Virheellisten SQL-kyselyiden selvitys.

Tämä SQL-kysely ei toimi. Yritin päivittää asiakas 99 mutta sain vain virheilmoituksen. Tässä kysely: UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;

ENG

Generating questions:

1. The language model should be able to display the database structure.

Show me the structure of the SQL database.

2. Add a new row to the table (INSERT).

I would like to add this information to the database: asiakas = Asiakas 101, asiakasnumero = 1101, maa = Suomi, puhelin = +358 40 123 45 101.

3. Update an existing row (UPDATE).

Please create an SQL query that updates the country of Asiakas 101 to Ruotsi.

4. Delete a specific row based on a condition (DELETE).

I would like to delete Asiakas 101 from the database. Could you make me the code to do that?

5. Fetch all rows from the table (SELECT).

Please create an SQL query that fetches all rows from the table Asiakkaat.

6. Fetch certain columns with a condition (WHERE).

Please create an SQL query that fetches all rows from the table Asiakkaat where maa is Suomi.

7. Order the results by a specific column (ASC/DESC).

I would like to sort the Asiakkaat table in ascending order.

I would like to sort the Asiakkaat table in descending order.

8. Count the number of rows in a table (COUNT).

I would like to know how many customers have a phone number ending with the number 1.

9. Joining two tables together (JOIN).

I would like to fetch from the database a list of customers who have a sopimus and a list of their puhelin numbers.

10. Use subqueries in SELECT, FROM, and WHERE clauses.

I would like to find all customers whose maa is Ruotsi and sort them by the puhelin field.

11. Troubleshooting incorrect SQL queries.

This SQL query does not work. I tried to update asiakas 99 but only got an error. UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;

Testit yhden taulun tietokannalle

Testikysymykset ovat samat kaikissa. Kysymykset on numeroitu ja jos kysymyksiä on kaksi, vastauksia on kaksi kysymysten mukaisessa järjestyksessä. Suomen- ja englanninkielisiin kysymyksiin numeroinnissa mainitaan etuliitteenä joko s (suomeksi) tai e (englanniksi).

Yhden taulun testien vastaukset

- s1. Tiedokanta on seuraava:
asiakkaat (taulu)
asiakas (näyttökielel: nvarchar, pituus: 100, luettavuus: 1)
asiakasnumero (numerointi: int, pituus: 4, arvo: 1, lasketaan)
maa (näyttökielel: nvarchar, pituus: 100, luettavuus: 1)
puhelin (näyttökielel: nvarchar, pituus: 100, luettavuus: 1)
- s2. Tällä on mahdollista luoda SQL-kysely, mutta tietokanta on jo olemassa ja sen nimessä on tietokantarakenne1taulu. Tämä tarkoittaa, että meidän tulee käyttää jo olevien tiedostojen sisältöä luomalla uuden tiedon lisäämisen sijaan.
- Kysely voidaan esittää seuraavasti:
- ```
INSERT INTO asiakkaat (asiakas, asiakasnumero, maa, puhelin) VALUES ('Asiakas 101', 1101, 'Suomi', '+358 40 123 45 101');
```
- Tämä kysely lisää uuden asiakkaan avulla uusi rivi asiakkaat -tauluun.
- s3. UPDATE asiakkaat SET maa = 'Ruotsi' WHERE asiakasnumero = 101;
- s4. Kyllä voisin auttaa sinua! Vaikka tiedän, että olet kyseessä taulukolla ja sille on laskettu metadata, halusin kuitenkin antaa vastauksen ilman laajempia tiedostoja tai koodikehitystä.
- Todennäköisesti niin, että Asiakas 101 poistetaan tietokannasta, käytetään Delete-toimintoa SQL-kommannassa. Kuitenkin ole hyvä tarkkailla, että tässä tapauksessa olet sanoessa "Asiakas 101", koska tiedossa on vain asiakasnumero "asiakasnumero" -kenttä.
- Varmistus: Haluatko poistaa kaikki rivejä, joissa on asiakasnumero 101?
- s5. SELECT \* FROM asiakkaat;
- Tämä kysely hakee kaikki sarakkeet (\*) tai mahdollisesti kaikki tiedot kunkin asiakkaiden rekisteröintiin. Jos haluat esimerkiksi nähdä kaikki asiakkaiden nimet, voit korvata SELECT \* SELECT asiakas lauseeksi.
- Vaikka kyseessä on vain yksi rivi taulussa Asiakkaat, tämän kyselyn tuloksena voi olla useampi rivi jos taulu sisältää muutokset.
- s6. SELECT \* FROM asiakkaat WHERE maa = 'Suomi';
- s7.1. Oletettavissa oleva SQL-kysely:
- ```
SELECT TOP 10 * FROM asiakkaat ORDER BY asiakasnumero DESC;
```
- Tämä kysely hakee ensimmäistä 10 rekistä nousevassa järjestyksessä taulussa asiakkaat, jossa asiakasnumero on alkamismerkintä.
- s7.2. Oletettavissa oleva SQL-kysely:

```
SELECT TOP 10 * FROM asiakkaat ORDER BY asiakasnumero ASC;
```

Tämä kysely hakee ensimmäistä 10 rekistä laskevassa järjestyksessä taulussa asiakkaat, jossa asiakasnumero on alkamismerkintä.

s8. Tarkasteltaessa tiedostoa löytynyt seuraava tieto:

Puhelinnumerot, joilla asiakasnopeus pätee numeroon 1: 11, 21, 31, 41, 51, 61, 71, 81, 91

s9. Tiedostoon on tallennettu seuraavat tiedot:

| | |
|-----------------------|-----------|
| TABLENAME | asiakkaat |
| COLUMNNAME | asiakas |
| COLUMNDATATYPE | nvarchar |
| MAXLENGTH | 100 |
| NUMERICPRECISION | 0 |
| NUMERICSCALE | 0 |
| ISNULLABLE | 1 |
| ISIDENTITY | 0 |
| ISPRIMARYKEY | 0 |
| DEFAULTCONSTRAINTNAME | NULL |

Koska asiakas -kenttä on nimenlaji, ei sen arvo ole tallennettu tietokantaan. Jos haluat hakea listaa asiakkaiden nimistä, joiden on sopimus, kannattaa käyttää seuraavaa pyynnönä:

```
SELECT DISTINCT a.asiakas FROM asiakkaat a JOIN sopimukset s ON a.asiakas = s.asiakas;
```

Tämä pyynnö on kuitenkin vain yksityiskohtaista tietoa, koska sopimukset-taulussa ei ole tallennettu asiakkaiden puhelinnumeroja.

Jos haluat hakea listaa asiakkaita, jotka ovat määrittäneet sopimus ja myös heidän puhelinnumeroidensa, kannattaa käyttää seuraavaa pyynnönä:

```
SELECT DISTINCT a.asiakas FROM asiakkaat a JOIN sopimukset s ON a.asiakas = s.asiakas JOIN asiakasnumeroan aan ON s.asiakas = aan.asiakasnumero;
```

Tämä pyynnö hakee listaa asiakkaiden nimistä, joiden on määritetty sopimus ja myös heidän puhelinnumerot.

s10. Pystyisit tehdä tätä käyttöliittymään, jonka avulla voit listata kaikki asiakasrekisteriisi, joilla on Ruotsia. Tähän tarvitaan:

1. Asiakasrekisterin löytämistä
2. Asiakkaat ja heidän maa-merkintätiedotensa luettamista
3. Puhelinnumeroja järjestelmään listaus

Kun olet listanut kaikki asiakkaiden ja niiden maa-merkintätiedot, voit käyttää tietoja esimerkiksi taulukossa tai piirroskuvassa, jossa voi näkyä listauksesi.

s11. Virheellinen kysely on tarkoitus päivittää rekisteröintitietoa, eikä se ole olemassa SQL:n riveissä. Oletettavasti pituusominaisuudet (MaxLength) ovat määrittelemättömä, jolloin yhteyden luominen ei onnistuisi.

```
Virheellinen kysely UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

Onnistuva kysely

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

Pitää muuttaa MaxLength 100:sta arvoon, joka vastaa suurinta mahdollista pituutta nvarchar-tyyppiselle sanoille.

e1. Based on the provided context, here is the structure of the SQL database:

```
CREATE TABLE asiakkaat (asiakas nvarchar(100) NOT NULL, asiakasnumero int NOT NULL, maa nvarchar(100), puhelin nvarchar(100));
```

Note: The actual data types and constraints may have been modified or added since the original context was provided. This is based on the initial metadata.

Also, please note that in SQL Server (the likely source of this database schema), the column order might be different from what's shown here, as SQL Server uses the IDENTITY property to create the primary key. In this case, the asiakasnumero would be an auto-incrementing integer and would not appear at the beginning of the table definition.

```
CREATE TABLE asiakkaat (asiakasNumero int IDENTITY(1,1) NOT NULL, asiakas nvarchar(100), maa nvarchar(100), puhelin nvarchar(100));
```

e2. Here is the SQL query that adds the specified data to the 'asiakkaat' table:

```
INSERT INTO asiakkaat (asiakas, asiakasnumero, maa, puhelin) VALUES ('Asiakas 101', 1101, 'Suomi', '+358 40 123 45 101');
```

Please note that the puhelin column is of type nvarchar, so it should be enclosed in single quotes.

e3. To update the country of a specific asiakas (customer), we need to specify the asiakas_number and the new value for the maa column, which represents the country in this context.

```
UPDATE asiakkaat SET maa = 'Ruotsi' WHERE asiakasnumero = 101;
```

This query updates the maa (country) of the asiakas with asiakas_number 101 to Ruotsi.

e4. I can provide you with a basic SQL query to delete a record from the asiakkaat table where asiakasnumero is 101, assuming it's an integer data type. Please note that I'll be using T-SQL syntax, which seems to be the default for this document metadata.

```
DELETE FROM asiakkaat WHERE asiakasnumero = 101;
```

This query will delete the entire row from the asiakkaat table where the asiakasnumero column is equal to 101. If you want to delete only the specific columns, you'll need to specify them in the DELETE statement.

If you're using a programming language (e.g., C#, Python) to interact with your database, I'd be happy to provide an example of how to achieve this using that language as well.

e5.

```
SELECT * FROM Asiakkaat;
```

e6.

```
SELECT * FROM Asiakkaat WHERE maa = 'Suomi';
```

This query selects all columns (*) from the Asiakkaat table where the value in the maa column is 'Suomi'. The result will be a set of rows that match this condition.

e7.1. To sort the Asiakkaat table in ascending order, you can use a SQL query such as:

```
SELECT * FROM asiakkaat ORDER BY asiakas ASC;
```

This will sort the columns of the asiakkaat table in ascending order. Note that I'm assuming you want to sort by the asiakas column, if you meant to sort by a different column, replace asiakas with the correct column name.

Also, please note that this is based on the provided context and the SQL syntax may vary slightly depending on the database management system being used.

e7.2. To sort the Asiakkaat table in descending order, you can use a SQL query such as:

```
SELECT * FROM asiakkaat ORDER BY asiakas DESC;
```

This will sort the columns of the asiakkaat table in ascending order by default (based on the provided context), and using DESC means descending order. If you meant to sort by a different column, replace asiakas with the correct column name.

Also, note that since maa is set as not nullable and its sort order is not specified, it will be treated as case-sensitive, so if there are similar values (e.g., "Ma" vs. "ma"), they might not be sorted correctly in a descending order. If you want to treat these cases as equivalent for sorting purposes, you can use the LOWER function:

```
SELECT * FROM asiakkaat ORDER BY LOWER(maa) DESC;
```

This will ensure that values are treated consistently regardless of case during sorting.

e8. You'll need to specify the database management system you're using. Assuming you're using MySQL, here's an example query:

```
SELECT COUNT(*) FROM customers WHERE phone LIKE '%1';
```

This query will count all rows in the customers table where the phone number ends with '1'. Note that this is a case-insensitive comparison due to the % wildcards.

If you want an exact match, remove the %, like so:

```
SELECT COUNT(*) FROM customers WHERE phone LIKE '1';
```

Also note that this query will return a count of all records where the phone column ends with '1', regardless of whether it's the last digit or part of a larger number (e.g., 123-456-7891).

e9. To achieve this, you can use a SQL query that joins the asiakkaat table with itself based on the asiakasnumero column, filtering for rows where the maa column is equal to 'sopimus'. The result will be a list of customers who have a sopimus and their corresponding puhelin numbers. Here's an example query:

```
SELECT a1.asiakas, a2.puhelin FROM asiakkaat a1 INNER JOIN asiakkaat a2 ON a1.asiakasnumero = a2.asiakasnumero AND a1.maa = 'sopimus'
```

This query will return a list of tuples containing the customer's name and their puhelin number, where the customer has a sopimus. Note that this assumes that the maa column is case-sensitive, so if you want to make it case-insensitive, you can modify the query to use the LOWER() function:

```
SELECT a1.asiakas, a2.puhelin FROM asiakkaat a1 INNER JOIN asiakkaat a2 ON LOWER(a1.maa) = 'sopimus' AND a1.asiakasnumero = a2.asiakasnumero
```

- e10. Unfortunately, I don't have enough context about the data you're working with (e.g., database schema, specific fields). However, assuming you have a table or dataset that contains customer information with columns like 'maa' (motherland/maa) and 'puhelin' (phone number), here's a general SQL query to get you started:

```
SELECT * FROM customers WHERE maa = 'Ruotsi' ORDER BY puhelin;
```

Replace customers with the actual table name in your database, and adjust column names as needed. This query will return all rows from the specified table where the value of the 'maa' field is 'Ruotsi', sorted by the 'puhelin' field.

If you're using a programming language like Python or Java to interact with your data, I can provide more specific code examples once I have more context about the libraries and frameworks you're using.

- e11. The issue you're encountering is likely due to the fact that asiakasnumero column in your table has a maximum length of 4 and stores integer values, but you're trying to update it with an integer value (99) directly.

Instead, you can cast the integer value to a string before updating it, like this:

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

Alternatively, if you want to keep asiakasnumero as an integer column and update it with a non-integer value (like a string representation of the number), you would need to convert the input data type before updating it.

It's worth noting that if you're trying to set a default value in SQL Server, the syntax is slightly different. You can use DEFAULT keyword instead of NULL. For example:

```
ALTER TABLE asiakkaat ALTER COLUMN maa nvarchar(100) DEFAULT 'Norja';
```

Also, note that the above solution will modify all rows where asiakasnumero equals 99 to have a maa value of 'Norja'. If you only want to update one specific row, then your original query should work fine:

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

Llama 3.2

s1. Tiedokanta on seuraava:

Asiakkaat:

asiakas (varchar, 50, kyseinen)
 asiakasnumero (int, eikä vastaa minkään
 toista tiedostoa)

Kohdet

kohdenumero (int, ei vastaa minkään toista
 tiedostoa)
 sopimusnumero (int, lukeutuu siihen sopi-
 mukset)
 kohteenosoite (nurk. 100, eikä vastaa min-
 kään toista tietoa)
 mittarilukema (tai/ei vastaa mitään tietoa)

Sopimukset:

sopimusnumero (int, lukeutuu kohde)
 asiakasnumero (int, lukeutuu asiakkaat)
 sopimuksennimi (nurk. 100, ei vastaa min-
 kään toista tietoa)
 alkamispvm (aika, ei vastaa mitään tietoa)
 paattymispvm (aika, ei vastaa mitään tietoa)

s2. Tiedostaa oleva tietokanta sisältää jo niinkauan tiedot, että ne sopivat hyvin soveltuva kysely:

```
INSERT INTO asiakkaat (asiakas, maa, puhelin) VA-
LUES ('Asiakas 101', 'Suomi', '+358 40 123 45 101');
```

Tässä kyselyllä pyritään lisätä uusi rekisteröinti asiakkaat tabelissa. Kyselyn arvoita ovat tämän hetken mukaan Asiakas 101, Suomi ja +358 40 123 45 101.

Llama 3.3

s1. SQL-tietokannan rakenne on seuraava:

Taulut:

asiakkaat

asiakas (nvarchar, 100)
 asiakasnumero (int, Primary Key)
 maa (nvarchar, 100)
 puhelin (nvarchar, 100)

kohteet

kohdenumero (int, Primary Key)
 sopimusnumero (int, Foreign Key -> sopi-
 mukset.sopimusnumero)
 kohteenosoite (nvarchar, 100)
 mittarilukema (float)

sopimukset

sopimusnumero (int, Primary Key)
 asiakasnumero (int, Foreign Key -> asiak-
 kaat.asiakasnumero)
 sopimuksennimi (nvarchar, 100)
 alkamispvm (date)
 paattymispvm (date)

Vierastaulukohtaisuudet:

kohteet.sopimusnumero -> sopimukset.sopi-
 musnumero
 sopimukset.asiakasnumero -> asiakkaat.asia-
 kasnumero

Tämä rakenne osoittaa, että tietokannassa on kolme taulua: asiakkaat, kohteet ja sopimukset. Asiakkaat-taulussa on tietoja asiakkaiden ominaisuuksista, kohteet-taulussa on tietoja kohteiden ominaisuuksista ja sopimukset-taulussa on tietoja sopimuksista. Vierastaulukohtaisuudet määrittävät, miten taulut liittyvät toisiinsa.

s2. INSERT INTO asiakkaat (asiakas, asiakasnumero, maa, puhelin) VALUES ('Asiakas 101', 1101, 'Suomi', '+358 40 123 45 101');

Tämä kysely lisää uuden rivin asiakkaat-tauluun, johon syötetään annetut tiedot. Huomaa, että asiakasnumero on taulun primäärivälikäinen avain, joten sen arvon on oltava yksilöllinen. Jos tällainen asiakasnumero on jo olemassa, saat virheilmoituksen.

Llama 3.2

s3. UPDATE asiakkaat SET maa = 'Ruotsi' WHERE asiakasnumero = 101;

Tämä tietokannassa oleva Asiakas 101 tulee päivittää maanaan "Ruotsi".

s4. Kyseessä on SQL-työntekijä. Poistaa asiakkaat taulukosta voi olla vaikeampaa kuin se näyttää. Tiedokannan asiakas numero 101 poistamiseksi voidaan käyttää seuraavaa SQL-komentia:

```
DELETE FROM asiakkaat WHERE asiakasnumero = 101;
```

Tämä komento poistaa koko rivin, jossa on Asiakas - taulukosta tiedot numero 101. Muuta tiedostettua tietoa ei poisteta.

Poistaesi asiakasta, muista varata backup-tila tai yksityiskohtaiset toimenpiteet asiakkaan poistumisesta!

Llama 3.3

s3. Voit päivittää asiakkaan tiedot seuraavalla SQL-kyselyllä:

```
UPDATE asiakkaat SET maa = 'Ruotsi' WHERE asiakasnumero = 101;
```

Tämä kysely etsii asiakkaan, jonka asiakasnumero on 101, ja päivittää hänen maa-kenttensä arvoksi 'Ruotsi'.

s4. Poistaaksesi asiakkaan tunnuksella 101 tietokannasta, tarvitset DELETE-kyselyn. Tässä on esimerkki siitä, miten voit tehdä tämän:

```
DELETE FROM asiakkaat WHERE asiakasnumero = 101;
```

Tämä poistaa rivin taulukosta asiakkaat, jossa asiakasnumero on 101. Huomaa, että jos asiakasnumero on viiteavain muissa tauluissa (kuten sopimukset-taulukossa), sinun on mahdollisesti poistettava ensin riippuvat rivit näistä tauluista tai muutettava viiteavaimen määrittämiä, jotta poisto voidaan suorittaa ilman virheitä.

Esimerkiksi jos sopimukset-taulukossa on rivejä, joissa asiakasnumero on 101, sinun täytyy ensin poistaa nämä rivit:

```
DELETE FROM sopimukset WHERE asiakasnumero = 101;
```

```
DELETE FROM kohteet WHERE sopimusnumero IN (SELECT sopimusnumero FROM sopimukset WHERE asiakasnumero = 101);
```

```
DELETE FROM sopimukset WHERE asiakasnumero = 101;
```

```
DELETE FROM asiakkaat WHERE asiakasnumero = 101;
```

Tässä esimerkissä oletetaan, että kohteet-taulukossa on viiteavain sopimusnumero, joka viittaa sopimukset-taulukon sopimusnumero-sarakkeeseen. Ensimmäin poistetaan kohteet-taulukosta rivit, jotka liittyvät asiakkaaseen 101, sitten sopimukset-taulukosta poistetaan rivit, jotka liittyvät asiakkaaseen 101, ja lopuksi asiakkaat-taulukosta poistetaan itse asiakas.

Llama 3.2

s5. SQL-kysely:

```
SELECT * FROM asiakkaat;
```

Tämä kysely hakee kaikki rivit asiakkaat-taulukselta, jossa on kaikki sarakkeet (*).

s6. SELECT * FROM asiakkaat WHERE maa = 'Suomi'

s7.1. Tämä voisi tehdä seuraavaa SQL-kyselyä:

```
SELECT * FROM asiakkaat ORDER BY asiakasnumero ASC;
```

Tällä kyselyllä pyritään luokittelemassa asiakkaat nousevassa järjestyksessä asiakasnumero -piirreellä. Järjestys on asetettu alimmasta lukua suurimpaan, koska ASC (kasvava) merkintä on oletettavissa.

s7.2. Tällöin voisi käyttää DESC (laskea) merkintää:

```
SELECT * FROM asiakkaat ORDER BY asiakasnumero DESC;
```

Tämä kysely järjestää asiakkaat laskevassa järjestyksessä asiakasnumero-piirrellä, eli ensimmäisestä lukua suurimpaan.

s8. Todistaa mahdollisuuksien perusteella, että on todennäköistä, että vain yksi asiakkasta (tai vähintään kaksi) on puhelinnumerostaan 1. Oletetaan, että yhteensä asiakasrekisterissä on x Asiakkaat.

Puhelinnumerojako: 0-9 Asiakastavastaan kuuluu kaikki numerot 001-, 011-, 021-, 031-, ..., 091-, 101-, 111-, 121-, 131-, ..., 911-. (Yhteensä 91 toistuva numero)

Puhelinnumerojako: 1-9 Asiakasta koskevat vain 1- ja 81- numerot, jolloin x=2.

Kuitenkin, 101-, 111-, 121-, ... -sarja sisältää myös 91-numeron. Tämä tarkoittaa, että yksi asiakas voi olla puhelinnumerostaan 1.

Tulosta: Asiakkaita on vähintään 2.

Llama 3.3

s5. SELECT * FROM asiakkaat;

s6. SELECT * FROM asiakkaat WHERE maa = 'Suomi';

s7.1. Voit käyttää seuraavaa SQL-kyselyä:

```
SELECT * FROM asiakkaat ORDER BY asiakasnumero ASC;
```

Tämä kysely hakee kaikki sarakkeet (*) asiakkaat-taulusta ja järjestää tulokset asiakasnumero-sarakkeen mukaan nousevassa järjestyksessä (ASC).

s7.2. Voit käyttää seuraavaa SQL-kyselyä:

```
SELECT * FROM asiakkaat ORDER BY asiakasnumero DESC;
```

Tämä kysely hakee kaikki sarakkeet (*) asiakkaat-taulusta ja järjestää tulokset asiakasnumero-sarakkeen mukaan laskevassa järjestyksessä (DESC).

s8. SELECT COUNT(asiakas) FROM asiakkaat WHERE puhelin LIKE '%1'

Llama 3.2

s9. Tiedostossa olevat taulut:

asiakkaat

kohteet

sopimukset

Oletettuna, että sovellus haluaa hakea asiakkaita, joilla on sopimus, ja niiden puhelin-numeroita.

Tiedostossa saatavilla olevat tiedot:

Asiakkaan id (asiakasnumero)

Sopimuksen id (sopimusnumero)

Puhelin-numerosi (puhelin)

Käytännössä tämä voi tehdä seuraavasti SQL-käsitteilylauseella:

```
SELECT a.asiakas, a.puhelin FROM asiakkaat a JOIN
sopimukset s ON a.asiakasnumero = s.asiakasnumero
```

Tämä käsittelylauske johtaa listaukselle asiakkaiden id:n ja puhelin-numerosa, joilla on sopimus.

Lisäksi tulokset voivat sisältää myös kohteiden id:n ja sijaintia (kohteenosoite), mikäli haluat katsotaan kaikkien liitteiden sisältöjä.

s10. Pystyisit kykenemään löytämään asiakkaasi, jotka ovat Ruotsia varten asettamassa puhelua, mutta tällä hetkellä en ole kyennyt lataamaan kaikkia dataa tarpeeseesi. Voisitteko antaa paljon lisää tietoa siitä mitä sinun tarvitsee?

Llama 3.3

s9. Voit hakea tiedot seuraavalla SQL-kyselyllä:

```
SELECT a.asiakas, a.puhelin FROM asiakkaat a JOIN
sopimukset s ON a.asiakasnumero = s.asiakasnumero;
```

Tämä kysely liittyy asiakkaat- ja sopimukset-taulut asiakasnumero-sarakkeen perusteella, jolloin saat listan asiakkaista ja heidän puhelinnumeroistaan, joilla on sopimus.

```
s10. SELECT * FROM customers WHERE country =
'Ruotsi' ORDER BY phone;
```

Llama 3.2

s11. Virheessä on kaksi pienempää ongelmaa:

1. SQL-kyselyssä käytetään virheettömästä "UPDATE" toiminnasta "UDPATE", joka ei ole olemassa tähän tarkoituksiin.
2. Päivittämiseen käytetään oikea sana "UPDATE", mutta päivitettyt tiedot ovat virheelliset, sillä asiakas 99 on jo ollut paikalla koko aikaa.

Virheilmoitus saatuasi on se, että kyseinen asiakas numero on todellisesti poissa käytöstä. Tämän syyttää tietokannassa poistettu rekisteri tai muuten asiakas 99 ei ole olemassa.

Puhutaan "UPDATE" toiminnasta, kun haluat päivittää tietyt arvot tiedostoon. Kuitenkin kyselyssä on virheellinen "UDPATE"-kanta. Sijoitaan sana "UPDATE" ja tarkastellaan, mitä arvoja halutessasi päivitetään.

Päivitetäkseen asiakas 99 sisältyvän tietoja, olisi kuitenkin oltava asiakas numero olemassa. Yritetään päivittää asiakasta, joka on paikalla:

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

Kun tämä ei toimi, on se todennäköisesti syytä tarkastella asiakasrekisteriä ja muuta siihen liittyvää tietoa, jotta voidaan selvittää, missä vaiheessa rekisteröintihistoria on hajoamaton tai poistettu.

Llama 3.3

s11. Kyselyssäsi on pieni kirjoitusvirhe. Oikea komento on UPDATE, ei UDPATE. Oikein kirjoitettu kysely olisi:

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

Tämä korjattu kysely pitäisi toimia, jos tietokannassa on asiakasnumero 99 ja muut edellytykset päivitykselle ovat kunnossa.

Llama 3.2

e1. Based on the provided context, here is a structured representation of the SQL database schema:

```
CREATE TABLE asiakkaat (
  asiasnumero INT NOT NULL,
  asiakas VARCHAR(50) NOT NULL,
  asiakasnumero INT NOT NULL PRIMARY KEY,
  maa NVARCHAR(50),
  puhelin NVARCHAR(50));
```

```
CREATE TABLE kohteet (
  kohdenumero INT NOT NULL PRIMARY KEY,
  sopimusnumero INT NOT NULL,
  kohteenosoite NVARCHAR(100) NOT NULL,
  mittarilukema FLOAT);
```

```
CREATE TABLE sopimukset (
  sopimusnumero INT NOT NULL PRIMARY
  KEY,
  asiakasnumero INT NOT NULL,
  sopimuksennimi NVARCHAR(100) NOT NULL,
  alkamispv DATE,
  paattymispv DATE);
```

```
CREATE TABLE asiakkaat_sopimukset (
  asiakasnumero INT NOT NULL,
  sopimusnumero INT NOT NULL,
  FOREIGN KEY (asiakasnumero) REFERENCES
  asiakkaat(asiakasnumero),
  FOREIGN KEY (sopimusnumero) REFERENCES
  sopimukset(sopimusnumero));
```

Note that the actual SQL code might vary based on the specific database management system being used. This representation is based on the provided context and might need adjustments for a specific DBMS.

Llama 3.3

e1. Based on the provided context, the SQL database structure appears to be as follows:

Tables:

```
asiakkaat (Customers)
  asiakas (nvarchar, 50)
  asiakasnumero (int, Primary Key)
  maa (nvarchar, 50)
  puhelin (nvarchar, 50)
```

kohteet (Sites)

```
kohdenumero (int, Primary Key)
sopimusnumero (int, Foreign Key referenc-
ing sopimukset.sopimusnumero)
kohteenosoite (nvarchar, 100)
mittarilukema (float)
```

sopimukset (Agreements)

```
sopimusnumero (int, Primary Key)
asiakasnumero (int, Foreign Key referencing
asiakkaat.asiakasnumero)
sopimuksennimi (nvarchar, 100)
alkuamispv (date)
paattymispv (date)
```

Relationships:

An agreement (sopimukset) is associated with one customer (asiakkaat) through the asiakasnumero foreign key.
A site (kohteet) is associated with one agreement (sopimukset) through the sopimusnumero foreign key.

Note that this structure is based on the provided context and may not be a comprehensive or complete representation of the entire database.

Llama 3.2

e2. To insert the new data into the database, I would use the following SQL commands:

```
INSERT INTO asiakkaat (asiakas, asiakasnumero) VALUES ('Asiakas 101', 1101);
```

```
UPDATE asiakkaat SET maa = 'Suomi' WHERE asiakasnumero = 1101;
```

```
UPDATE asiakkaat SET puhelin = '+358 40 123 45 101' WHERE asiakasnumero = 1101;
```

These commands first insert the new data into the asiakkaat table, then update the existing record with the new values.

Alternatively, if you want to insert all the columns at once, you can use:

```
INSERT INTO asiakkaat (asiakas, asiakasnumero, maa, puhelin) VALUES ('Asiakas 101', 1101, 'Suomi', '+358 40 123 45 101');
```

This will achieve the same result with less code.

e3. Here is the SQL query to update the country of Asiakas 101:

```
UPDATE asiakkaat SET maa = 'Ruotsi' WHERE asiakasnumero = 101;
```

This query updates the maa column in the asiakkaat table to 'Ruotsi' for the row where asiakasnumero equals 101.

Llama 3.3

e2. To add this information to the database, you would need to insert a new row into the "asiakkaat" table with the provided values. The SQL command for this operation would look like this:

```
INSERT INTO asiakkaat (asiakas, asiakasnumero, maa, puhelin) VALUES ('Asiakas 101', 1101, 'Suomi', '+358 40 123 45 101');
```

This command specifies the table and columns where you want to add the new data, and then provides the values to be inserted in the same order as the columns.

e3. UPDATE asiakkaat SET maa = 'Ruotsi' WHERE asiakasnumero = 101;

Llama 3.2

e4. I don't have direct access to your database. However, I can provide you with a SQL query that you can use to delete an asistajattomasta (client) from your database.

```
DELETE FROM asiakkaat WHERE asiakasnumero = 101;
```

This query deletes the row in the asiakkaat table where the asiakasnumero is equal to 101. Make sure you have the necessary permissions to execute this query, as it permanently deletes data from your database.

Note: Before executing the query, make sure that there are no active transactions or locks on the table. You can use the DBCC CHECKIDENT statement to verify the current identity of the table before deleting the row. For example:

```
DBCC CHECKIDENT ('asiakkaat');
```

Also, be aware that this will delete all related records in other tables (e.g., kohteet, sopimukset) where the foreign key references the deleted client's ID. If you want to preserve these relationships, you may need to use more complex query or transactions.

Please test this query in a safe environment before running it on your production database.

e5. `SELECT * FROM asiakkaat;`

This will return all columns (*) for all rows in the asiakkaat table. If you want to fetch only specific columns, replace * with the column names you're interested in (e.g., `select asiatas, maanammi, puhelin from asiakkaat;`).

e6. `SELECT * FROM asiakkaat WHERE maa = 'Suomi';`

e7.1. To sort the Asiakkaat table in ascending order, you can use the following SQL query:

```
SELECT * FROM dbo.asiakkaat ORDER BY asiakas ASC;
```

This will return all columns (*) from the Asiakkaat table and sort the result set by the asiakas column in ascending order.

Llama 3.3

e4. To delete a record from the asiakkaat table, you can use the following SQL query:

```
DELETE FROM asiakkaat WHERE asiakasnumero = 101;
```

This will delete the row where asiakasnumero (which is the primary key) equals 101. Please note that this will also cascade to other tables if there are foreign keys referencing asiakkaat, but based on the provided context, it seems like only sopimukset has a foreign key referencing asiakkaat. If you want to avoid cascading deletes, you may need to adjust the query or handle the related records in sopimukset separately.

Also, please be aware that deleting data is permanent and should be done with caution. It's always a good idea to back up your database before making significant changes like this.

e5. `SELECT * FROM asiakkaat;`

e6. `SELECT * FROM asiakkaat WHERE maa = 'Suomi';`

e7.1 To sort the Asiakkaat table in ascending order, you can use the following SQL query:

```
SELECT * FROM asiakkaat ORDER BY asiakasnumero ASC;
```

This will return all columns (*) from the asiakkaat table sorted by the asiakasnumero column in ascending order. If you want to sort by a different column, replace asiakasnumero with your desired column name.

Llama 3.2

e7.2. To sort the Asiakkaat table in descending order, you can use the following SQL query:

```
SELECT * FROM dbo.asiakkaat ORDER BY asiakas
DESC;
```

This will return all columns (*) from the Asiakkaat table and sort the result set by the asiakas column in descending order.

e8. Unfortunately, I don't have access to real-time customer data or specific information about individual customers. However, I can suggest some general insights.

Phone numbers typically follow international standards, and the format usually includes country codes, area codes, and local numbers. The last digit of a phone number is often referred to as the "line number" or "suffix."

In most countries, including the United States, the UK, Australia, and others, phone numbers do not have a fixed sequence for their last digit (1 in your case). Instead, each country's telephone numbering plan dictates the possible values for the last digit. For example, in the US, mobile phone numbers often end with 1-9, while landline numbers can end with any digit from 0 to 9. In some countries like the UK, Australia, and others, the last digit of a phone number is more restricted.

Without specific data on your customer base or industry, it's challenging to provide an exact count of customers with phone numbers ending in 1. If you have access to your customer database or want to know more about phone number patterns in a particular country or region, I'd be happy to help with further research or guidance.

Llama 3.3

e7.2. To sort the Asiakkaat table in descending order, you can use the following SQL query:

```
SELECT * FROM asiakkaat ORDER BY asiakasnumero
DESC;
```

This will return all columns (*) from the asiakkaat table sorted by the asiakasnumero column in descending order. If you want to sort by a different column, replace asiakasnumero with your desired column name.

e8. To find out how many customers have a phone number ending with the number 1, I would need access to your customer database. However, assuming I have that access, I can tell you that I would use a query or filter to search for phone numbers that end with '1'.

For example, if this were a SQL database, the query might look something like: `SELECT COUNT(*) FROM customers WHERE phone_number LIKE '%1';`

This query would return the count of rows (i.e., customers) in your database where the phone number ends with '1'. Without actual access to the data or more specifics about how it's stored and managed, I can only provide this general approach.

Llama 3.2

e9. To achieve this, you can use a SQL query that joins the asiakkaat table with the sopimukset table based on the foreign key relationship between asiakasnumero and sopimusnumero. You can then select the asiakas column from the asiakkaat table and the puhelin column from the same table.

Here is an example query:

```
SELECT a.asiakas, a.puhelin FROM asiakkaat a INNER JOIN sopimukset s ON a.asiakasnumero = s.asiakasnumero;
```

This will return a list of customers who have a sopimus, along with their corresponding puhelin numbers.

e10. You can achieve this using SQL with a subquery or join, assuming you are using a database management system that supports these features.

Here's an example query:

```
SELECT * FROM customers WHERE maan = 'Ruotsi' ORDER BY puhelin;
```

This will return all rows from the customers table where maan is 'Ruotsi', sorted by the puhelin field in ascending order.

e11. The issue here is that the maa column in the asiakkaat table has a maximum length of 50 characters, and you're trying to update it with a string that's longer than that.

Try updating it like this:

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

If the maa column can actually hold a longer string (maybe due to a typo in the documentation?), you'll need to adjust the MAX LENGTH value accordingly in your database schema.

Llama 3.3

e9. To achieve this, you can use the following SQL query:

```
SELECT a.asiakas, a.puhelin FROM asiakkaat a JOIN sopimukset s ON a.asiakasnumero = s.asiakasnumero;
```

This query joins the asiakkaat table with the sopimukset table on the asiakasnumero column, which is common to both tables. The result will be a list of customers (asiakas) who have at least one sopimus (contract) and their corresponding phone numbers (puhelin).

Note: If a customer has multiple contracts, they will appear in the results multiple times, once for each contract. If you want to remove duplicates, you can use the DISTINCT keyword:

```
SELECT DISTINCT a.asiakas, a.puhelin FROM asiakkaat a JOIN sopimukset s ON a.asiakasnumero = s.asiakasnumero;
```

e10. `SELECT * FROM customers WHERE maa = 'Ruotsi' ORDER BY puhelin;`

e11. The issue with your SQL query is a simple typo. You have written "UDPATE" instead of "UPDATE". The correct query should be:

```
UPDATE asiakkaat SET maa = 'Norja' WHERE asiakasnumero = 99;
```

