**Harjoitustyön raportti, Tietokantojen perusteet**

**Nimi:** Jarno Saastamoinen

**Opiskelijanumero:** 015066246

**Mooc.fi-tunnus:** [jarno.saastamoinen@hotmail.com](mailto:jarno.saastamoinen@hotmail.com)

**Harjoitustyö GitHubissa:** <https://github.com/SJarno/HarjoitusduuniBeta>

**Teknologiat ja muut vastaavat:** Java 8, Sqlite, DB-viewer, NetBeans

**Yleisesti**

Harjoitustyön aiheeksi annettiin kurssin materiaalissa tehdä sovellus, millä on tarkoitus seurata paketin matkaa eri vaiheissa seurantakoodia käyttäen. Tarkoituksena siis on pyrkiä mallintamaan todellisen maailman tilannetta, missä esimerkiksi netistä tilataan tuote ja sen matkaa on mahdollista seurata esimerkiksi paikkakohtaisesti. Samalla tietoa tallennetaan relaatiotietokantaan, kun taas ohjelmallisuus käsitellään korkean tason ohjelmointikielellä.

Käytännössä harjoitustyöhän annettiin mahdollisuus toteuttaa ohjelma joko Javalla tai Pythonilla, itse tietokanta on toteutettu luonnollisesti materiaalissakin esitellyllä Sqlitella.

Päädyin itse toteuttamaan ohjelmallisen osuuden Javalla, sillä tämä oli itselleni tuttu ohjelmointikieli. Tietokantojen käsittely ohjelmallisesti ei itselleni ollut aiemmin kovin tuttua, joten käytin pääasiassa materiaalissa olleita esimerkkejä yhdistellen omaan aikaisempiin kokemuksiini OOP-ohjelmointitapoihin. Kuitenkin tietyissä tilanteissa jouduin netistä etsimään tietoja ja tapoja, jotta sain joko ratkaistua esiintyvän ongelman tai muuten vaan piti turvautua johonkin hyväksi havaittuun tapaan.

Annettuja tehtäviä pyrin seuraamaan melko pikkutarkkaan. Tässä raportissa viittaan satunnaisesti siteerauksien kanssa materiaalissa esiintyneihin tehtävänantoihin, kun se on tarpeellista.

Kurssin vetäjälle voisin lähettää terveisiä, että jäin itse kaipaamaan kunnollista DAO-mallia, miten tietokantoja olisi kuulunut käsitellä Javalla. Tai mikä ammattimainen tapa/tavat nyt ikinä olisi sopiva lähestyminen. Kaiken kaikkiaan tämä oli mielekäs ja mielenkiintoinen projekti ja nyt kirjoitushetkellä oikeastaan tiedän jo miten olisin voinut koodin toteuttaa huomattavasti paremmin vaikka kunnon vertailu puuttuukin.

**Java**

Ohjelmallinen osuus on koodattu Java 8 versiolla, käyttäen NetBeansia. Yritin mahdollisuuksien mukaan lähestyä olio-ohjelmoinnin näkökulmasta aihealuetta. Kuten aiemmin mainittukin, ei aiempaa kokemusta itselläni juuri ole tämän kaltaisesta ohjelmoinnin alueesta koskien tietokantoja, jouduin siis turvautumaan niihin palikoihin, mitä tarjolla on ollut.

Käytännössä ohjelma koostuu tietokannan lisäksi kolmesta Java-luokasta:

* Tietokanta.java
* Kayttoliittyma.java
* Main.java

Tietokanta -luokka on vastuussa tietokannan käsittelystä. Tietokanta luo tietokannan sekä taulut tietokantaan, lähettää kyselyt ja muut vastaavat tietokantaan. Luokka on myös vastuussa virheviesteistä, mitä käyttäjälle lähetetään liittyen sql-kyselyihin.

Kayttoliittyma -luokka on vastuussa ohjelman käyttöliittymästä nimensä mukaisesti sekä käyttäjän syötteiden käsittelystä ja niihin liittyvistä asioista. Luokassa on myös otettu huomioon mahdolliset käyttäjän antamat virhesyötteet joko osittain tai kokonaan, jos niitä ei ole otettu huomioon, on ne otettu Tietokantaluokassa huomioon.

Main -luokka, luo tarvittavat olio-luokat eli Tietokanta- ja Kayttoliittyma.java, alustaa tarvittavan polun tietokantaan sekä kutsuu Kayttoliittyma-luokan käynnistä -metodia, millä käyttäjä pääsee antamaan syötteitä ja tekemään muita tarvittavia toimenpiteitä.

Yleisesti pyrin ottamaan mahdollisimman paljon huomioon mahdollisia käyttäjän tekemiä virhesyötteitä tai muita tilanteita, missä ohjelma saattaisi kaatua tai väärää tietoa pääsisi muuten tietokantaan. Nämä ovatkin joko estettyjä jo ohjelman puolella ja käyttäjälle suunnataan mahdollisimman järkevä virheviesti, mikä kertoo mahdollisesta erheestä sekä kiinnittämään asiaan huomiota.

Käytännössä kaikki tietokantaan liittyvät metodit, jotka luo yhteyden kantaan suljetaan aina sen jälkeen, kun metodi on tehnyt tehtävänsä.

**Tietokanta**

Harjoitustyössä on käytössä kolme tietokantaa: Yksi on varsinainen ”päätietokanta”, millä toteutetaan pääosa toiminnallisuudesta ja loput kaksi on nopeustestejä varten, mihin luodaan täsmälleen samat taulut kuin päätietokantaan.

Tietokanta on toteutettu materiaalin luvun 5.1 periaatteiden mukaisesti; taulut ja sarakkeet ovat kiinteitä, muutokset tehdään käyttäjän puolelta ainoastaan riveihin sekä saman tyyppinen tieto on yhdessä taulussa. Jokaisessa sarakkeessa on yksittäinen tieto – eli listaa, tieto on yhdessä paikassa, mihin viitataan pääavaimella. Muuta tietoa ei myöskään pysty päättelemään muun sisällön perusteella.

Tietokanta on myös pyritty suunnittelemaan siten, ettei väärää tietoa pitäisi sinne joutua. Vaikkakin ohjelmallinen puoli pitää tästä pitkälti myös huolen, katsoin tarpeelliseksi tietyissä kohdissa tehdä tietokannan myös tällä perusteella.

Tehtävänantoa siteeraten tässä kohtaa; taulut ovat tehty seuraavalla tavalla:

*” Paikka on jokin kohde paketin matkan varrella, jossa paketti voidaan skannata. Jokaisella paikalla on eri nimi.*

*Asiakas on paketin tilaaja. Jokaisella asiakkaalla on eri nimi.*

*Paketilla on seurantakoodi, jolla siihen voidaan viitata. Jokaisella paketilla on eri seurantakoodi. Paketti liittyy tiettyyn asiakkaaseen.*

*Kun paketti skannataan, syntyy tapahtuma. Tapahtuma liittyy tiettyyn pakettiin ja tiettyyn paikkaan. Lisäksi tapahtumaan kuuluu kuvaus ja lisäyshetki (päivämäärä ja kellonaika).”*

Tauluja tietokannassa(issa) on neljä, kaikissa pääavain:

* Paikat: Uniikki nimi jokaisella paikalla.
* Asiakas: Uniikki nimi jokaisella asiakkaalla.
* Paketti: Uniikki seurantakoodi jokaisella paketilla, liittyy tauluun Asiakas ja asiakkaan nimeen.
* Tapahtuma: Liittyy Paketti-tauluun(viittaa seurantakoodiin), lisää järjestelmässä olevan aikaleiman todellisesta ajasta sekä lisää mahdollisen kuvauksen.

**Selostus, mitkä toiminnot harjoitustyöhön on toteutettu**

Ohjelman käynnistyessä luetellaan käyttäjälle muutama peruskomento, missä kehotetaan valitsemaan vaihtoehdoista sopivin. Valittavana on komentojen luotteleminen (L), Komentojen valitseminen numerolla väliltä 1-9 tai ohjelmasta poistuminen komennolla (X).

Jos käyttäjä antaa komennon, mitä ei ole annettu peruskomennoissa ei tapahdu mitään, vaan käyttäjälle annetaan uudestaan peruskomennot. Komennoiksi hyväksytään myös isot ja pienet kirjaimet sen takia, että käyttäminen olisi suhteellisen sujuvaa.

Komento L luettelee yhdeksän erilaista komentoa, mitkä on käytännössä harjoitustyössä vaaditut yhdeksän osaa.

Komennolla X luonnollisesti poistutaan ohjelmasta ja tietokanta suljetaan.

Syöttämällä numeron väliltä 1-9, tekevät seuraavanlaista, nämä ovat toteutettu opiskelumateriaalissa olevien ohjeita mukaillen:

1. *" Luo sovelluksen tarvitsemat taulut tyhjään tietokantaan (tätä toimintoa voidaan käyttää, kun tietokantaa ei ole vielä olemassa).*
2. *Lisää uusi paikka tietokantaan, kun annetaan paikan nimi.*
3. *Lisää uusi asiakas tietokantaan, kun annetaan asiakkaan nimi.*
4. *Lisää uusi paketti tietokantaan, kun annetaan paketin seurantakoodi ja asiakkaan nimi. Asiakkaan tulee olla valmiiksi tietokannassa.*
5. *Lisää uusi tapahtuma tietokantaan, kun annetaan paketin seurantakoodi, tapahtuman paikka sekä kuvaus. Paketin ja paikan tulee olla valmiiksi tietokannassa.*
6. *Hae kaikki paketin tapahtumat seurantakoodin perusteella.*
7. *Hae kaikki asiakkaan paketit ja niihin liittyvien tapahtumien määrä.*
8. *Hae annetusta paikasta tapahtumien määrä tiettynä päivänä.*
9. *Suorita tietokannan tehokkuustesti (tästä lisää alempana).*

Jatkossa numerolla viitataan lähdekoodissa löytyvään samaan numeroituun kohtaan, mikä löytyy aina kommenttina, yllä olevan tehtävän annon lisäksi. Jos ei toisin ole sanottu, yhteys suljetaan jokaisen kutsun jälkeen ja virheisiin on varauduttu vähintään try-catch lohkoilla.

**1.Komento – Luodaan tietokanta ja taulut**

Käyttöliittymä kutsuu Tietokantaolion metodia ”luoTaulutTietokantaan”, mikä luo tietokannan ja asianmukaiset taulut sinne. Metodin sisäinen objekti lähettää sql-kyselyn tietokantaan ja mahdolliseen virheeseen varaudutaan ja annetaan käyttäjälle virheviestin. Käytännössä jos tietokanta ja taulut ovat jo olemassa, tulee tästä käyttäjälle viesti. Onnistuessa lisätään jo aiemminkin mainitut neljä taulua: Paikat, Asiakas, Paketti ja Tapahtuma.

**2.Komento – Paikan lisääminen tietokantaan**

Käyttöliittymässä pyydetään käyttäjää syöttämään lisättävän paikan nimi ja samalla pitäen huolen että jos käyttäjä erehtyy syöttämään välin alkuun tai loppuun, trimmataan syötteestä pois ylimääräiset välit. Tämän jälkeen kutsutaan Tietokanta-objektista ”lisaaPaikka”-metodia, mikä ei ota vastaan tyhjää riviä, vaan palauttaa käyttäjän peruskomentoihin virheviestin kera, jos näin tapahtuu.

Lisäksi metodi tekee tarkistuksen tietokantaan olemassaolevasta paikasta, jos sellainen löytyy palautetaan käyttäjä peruskomentoihin asianmukaisen virheviestin kanssa.

Tietokannassa oleva taulu Paikat varmistaa, että syötetty paikka ei ota null-arvoa vastaan, eli ei voi olla tyhjä, että syötetty paikka on vähintään yhden merkkijonon pituinen sekä toista vastaavaa paikkaa ei löydy.

Jos yllä olevat ehdot täyttyvät, lisätään paikka tietokantaa, muussa tapauksessa käyttäjä palautetaan peruskomentoihin virheviestin saattelemana. Onnistuneesta lisäyksestä ja tarkistuksesta tulee käyttäjälle viesti.

**3.komento – Asiakkaan lisääminen tietokantaan**

Samoin kuin edellisessä komennossa, myös tässä käyttäjän syöte trimmataan, jotta turhat välit poistetaan alusta ja lopusta. Tämän jälkeen kutsutaan Tietokanta-objektin metodia ”lisaaNimi”. Metodi ottaa vastaan käyttäjän syötteen, mikä on asiakkaan nimi. Metodi ei ota vastaan tyhjäksi jätettyä syötettä, tekee tarkistuksen tietokantaa olemassa olevasta asiakkaan nimestä. Jos syöte on tyhjä tai asiakas on jo olemassa, palautetaan käyttäjä peruskomentoihin virheviestin kanssa.

Tietokannassa oleva taulu Asiakas varmistaa, ettei se ota vastaa null-arvoa, merkkijonon pituus tulee olla vähintään yhden merkin pituinen sekä varmistaa että paikka on uniikki, eli toista vastaavaa ei ole.

Mahdollisista virheistä tulee käyttäjälle viesti ja käyttäjä palautetaan pääkomentoihin, muussa tapauksessa onnistuneesta lisäyksestä tulee ilmoitus, jos asiakasta ei ollut tietokannasta sekä itse lisäyksestä.

**4.komento – Paketin lisääminen tietokantaan**

Neljännellä komennolla lisätään tietokantaan paketti seurantakoodin ja asiakkaan nimen perusteella. Käyttöliittymä kysyy käyttäjältä lisättävää seurantakoodia sekä asiakkaan nimeä. Turhat välit alusta ja lopuista trimmataan pois molemmista syötteistä ja Käyttöliittymä kutsuu Tietokannan metodia lisaaPaketti.

Metodissa on useampi eri tarkastus; syötetty seurantakoodi ei saa olla negatiivinen, se täytyy olla numeromuodossa eikä syöte saa olla tyhjä – käyttäjä palautetaan muussa tapauksessa. Lisäksi tarkistetaan onko seurantakoodi jo kannassa, jos on niin käyttäjä palautetaan peruskomentoihin.

Syötetyn asiakkaan nimen kanssa tarkistetaan samalla tavalla, ettei syöte ole tyhjä tai ettei asiakas ole jo valmiina kannassa, tarvittaessa käyttäjä palautetaan virheviestin kanssa peruskomentoihin.

Tietokannassa oleva taulu Paketti ei ota vastaan null-arvoa seurantakoodiksi, eikä samanlaista vastaavaa seurantakoodia ja pitää huolen, että seurantakoodi on vähintään yhden merkin pituinen. Taulusta myös viitataan asiakkaan id-numeroon toiseen tauluun, missä on asiakkaan nimet – Asiakas.

**5.komento – Tapahtuman lisääminen tietokantaan**

Tapahtuman lisääminen tietokantaan tauluun ”Tapahtuma” tapahtuu siten, että käyttöliittymä kutsuu Tietokanta-objektin metodia lisaaTapahtuma ja pyytää käyttäjältä seurantakoodin, asiakkaan nimen ja kuvauksen tapahtumasta. Molemmista syötteistä poistetaan alusta ja lopusta turhat välit käyttöliittymän puolella. Itse metodi tarkistaa, ettei paikka tai nimi ole tyhjiä, muuten käyttäjä palautetaan peruskomentoihin.

Syötetty seurantakoodi ja paikka tulee olla valmiiksi kannassa ja tästä tehdään tarkistus, jos jompaakumpaa ei löydy tietokannasta, palautetaan käyttäjä peruskomentoihin virheviestin kanssa.

Itse kuvaus saa olla tyhjä eikä rajoita syötettä mitenkään käyttäjän puolelta. Tämän lisäksi ohjelma hakee reaaliaikaisen aikaleiman omalla metodilla ja lisää sen tietokantatauluun.

Tietokannassa oleva taulu ”Tapahtuma” ei ota null-arvoja vastaan seurantakoodiin eikä paikkaan, nämä myös viittaavat tauluihin Paketti ja Paikat omilla id-numeroillaan. Kuvaus tosiaan saa olla tarvittaessa tyhjä.

**6.komento – Haetaan paketin tapahtumat**

Komennolla käyttäjältä pyydetään syöttämään seurantakoodi, ohjelma trimmaa syötteen, eli poistaa alusta ja lopusta ylimääräiset välit ja kutsuu metodia haePaketinTapahtumat, mikä ottaa parametriksi käyttäjän antaman syötteen.

Metodi tarkistaa, ettei syöte ole tyhjä, muuntaa syötteen numeroksi ja varmistaa sen olevan numeromuodossa. Molemmissa tapauksissa, jos käyttäjä on antanut vääränlaisen syötteen, palautetaan käyttäjä peruskomentoihin. Myös siinä tapauksessa, jos seurantakoodia ei kannasta löydy, palautetaan käyttäjä. Jos seurantakoodilla ei ole tapahtumia lainkaan, tulee tästä myös käyttäjälle viesti.

Muussa tapauksessa ohjelma lähettää tietokantaan kyselyn, missä haetaan tietyn paketin tapahtumat. Aikajärjestyksessä ohjelma kertoo käyttäjälle tietyn paketin aikaleiman, paikan sekä annetun kuvauksen.

Itse sql-kysely näyttää seuraavanlaiselta:

*”SELECT paivamaara, paikka, kuvaus*

*FROM Tapahtuma LEFT JOIN Paikat, Paketti*

*ON Paketti.id = seurantakoodi\_id AND paikka\_id = Paikat.id*

*AND Paketti.seurantakoodi =?*

*GROUP BY paivamaara;”*

**7.komento – Haetaan asiakkaan paketit ja tapahtumien määrä**

Käyttäjä syöttää tässä tapauksessa asiakkaan nimen, jolloin tietokannasta haetaan tiedot kaikista asiakkaan paketeista ja niihin liittyvien tapahtumien määrästä.

Käyttöliittymä trimmaa annetun syötteen ja kutsuu metodia ”haeAsiakkaanPaketitJaTapahtumat” , mikä tarkistaa ettei syöte ole tyhjä ja sen onko asiakas tietokannassa. Kummastakin tulee erillinen virhesyöte ja käyttäjä palautetaan peruskomentoihin.

Muussa tapauksessa tietokantaan lähetetään seuraavanlainen syöte:

*”SELECT Pa.seurantakoodi, COUNT(T.seurantakoodi\_id)*

*FROM Paketti Pa LEFT JOIN Tapahtuma T*

*ON Pa.id = T.seurantakoodi\_id*

*LEFT JOIN Asiakas A ON A.id = Pa.asiakas\_id WHERE A.nimi =?*

*GROUP BY Pa.seurantakoodi*

*ORDER BY COUNT(T.seurantakoodi\_id) DESC;”*

Käyttäjälle palautetaan pakettien seurantakoodi sekä niihin liittyvät tapahtumat. Jos tapahtumia ei ole tietylle seurantakoodille, mikä on asiakkaaseen sidottu, annetaan tästä määräksi luonnollisesti 0.

**8.komento – Haetaan annetusta paikasta tapahtumien määrä**

Ohjelma kysyy käyttäjältä paikan nimeä sekä päivämäärää, millä on tarkoitus hakea tiettyyn paikkaan siduttu tapahtumien määrä tietyllä päivämäärällä.

Kuten tähänkin saakka käyttöliittymä trimmaa pois ylimääräiset välilyönnit alusta. Tämän jälkeen kutsutaan Tietokanta-luokasta metodia haePaketinTapahtumatPaivana, mikä ottaa parametreiksi edellä mainitut paikan ja päivämäärän. Metodi palauttaa käyttäjän takaisin peruskomentoihin, jos syötteet ovat tyhjiä tai jos annettua paikkaa tai päivämäärää ei löydy tietokannasta.

Muussa tapauksessa tietokantaan lähetetään seuraavanlainen kysely:

"SELECT COUNT(T.id)

FROM Tapahtuma T LEFT JOIN Paikat P

ON T.paikka\_id = P.id WHERE P.paikka =?

AND T.paivamaara LIKE ?"

Kysely palauttaa käyttäjälle tiedon siitä, kuinka monta tapahtumaa löytyy paikkakohtaisesti tiettynä päivämääränä.

**9.komento – Tehokkuustesti**

Tehokkuustestejä varten käyttäjälle tarjotaan kaksi vaihtoehtoa: tehdä testi ilman indeksejä ja indeksien kanssa – käytännössä vaihtoehdot 1 ja 2.

Molempia testejä varten luodaan omat tietokannat, mitkä ovat identtisiä tauluineen ja niissä olevien ehtojen kanssa. Käyttöliittymän puolella luodaan siis kaksi uutta Tietokanta-objektia tätä varten, joissa on samat metodit.

Testien sujuvuuden ja käyttäjää ajatellen olemassa olevat taulut kuitenkin poistetaan tietokannasta ja luodaan uudet. Tämä sen takia, että tehokkuustestejä olisi sujuva ajaa peräkkäin.

Materiaalin mukaisesti testi ovat jaettu seuraavanlaisesti kuuteen osaan:

1. *Tietokantaan lisätään tuhat paikkaa nimillä P1, P2, P3, jne.*
2. *Tietokantaan lisätään tuhat asiakasta nimillä A1, A2, A3, jne.*
3. *Tietokantaan lisätään tuhat pakettia, jokaiselle jokin asiakas.*
4. *Tietokantaan lisätään miljoona tapahtumaa, jokaiselle jokin paketti.*
5. *Suoritetaan tuhat kyselyä, joista jokaisessa haetaan jonkin asiakkaan pakettien määrä.*
6. *Suoritetaan tuhat kyselyä, joista jokaisessa haetaan jonkin paketin tapahtumien määrä.*

Kohdat 1-4 tehdään yhden transaktion sisällä, jotta testeihin ei menisi liian pitkä aika. Aiemmat kokeilut osoitti, että ilman indeksien lisäämistä miljoonan tapahtuman lisäämisessä meni noin 40-50 minuuttia.

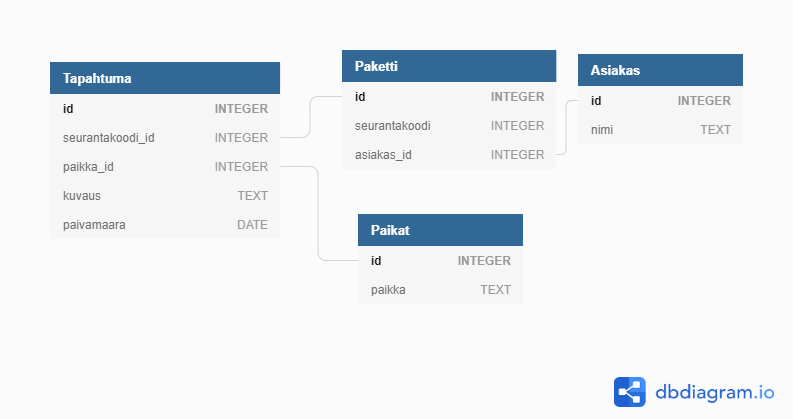
* 1.- ja 2.kohdissa paikat ja asiakkaat lisätään suoraviivaisesti for-loopilla kantaan.
* 3.kohdassa tuhannen paketin lisäämisessä arvotaan kannassa olevan asikkaan id-tunnus väliltä 1-1000.
* 4.kohdassa paketin tunnus arvotaan kannasta id-tunnuksella väliltä 1-1000.
* 5.kohdassa arvotaan asiakkaan id-tunnuksella tuhat kyselyä.
* 6.kohdassa arvotaan seurantakoodin id-tunnuksella paketin tapahtumamäärä.

Lopuksi käyttäjälle tulostetaan tehokkuustestien tulokset jokaisesta kuudesta eri kohdasta sekä paljonko kokonaisuudessa tähän meni aikaa. Ajanotto on toteutettu käyttämällä Javan tarjoamaa metodia System.nanoTime(), jonka avustuksella lasketaan kulunut aika sekunteina.

Molemmat tehokkuustestit toimivat muuten samalla tavalla, mutta indeksien kanssa tehtävä tehokkuustesti 2 suoritetaan siten, että kohtia 5 ja 6 ajatellen on taulujen luonnin yhteydessä lisätty indeksit, joiden perusteella haut toimivat jonkin verran nopeammin.

**Tietokantakaavio ja SQL-skeema**

Tietokantakaavio menee harjoitustyössä seuraavalla tavalla:

****

Sql-skeema taas näyttää tältä:

CREATE TABLE Paikat (id INTEGER PRIMARY KEY, paikka TEXT UNIQUE NOT NULL CHECK (LENGTH(paikka) > 0));

CREATE TABLE Asiakas (id INTEGER PRIMARY KEY,nimi TEXT UNIQUE NOT NULL CHECK (LENGTH(nimi) > 0));

CREATE TABLE Paketti (id INTEGER PRIMARY KEY,seurantakoodi INTEGER UNIQUE NOT NULL CHECK (LENGTH(seurantakoodi) > 0),asi

akas\_id INTEGER NOT NULL REFERENCES Asiakas);

CREATE TABLE Tapahtuma (id INTEGER PRIMARY KEY,seurantakoodi\_id INTEGER NOT NULL REFERENCES Paketti,paikka\_id INTEGER NO

T NULL REFERENCES Paikat,kuvaus TEXT, paivamaara DATE);

**Tehokkuustestin tulokset kahdessa tapauksessa: ilman indeksejä ja indeksien lisäämisen jälkeen**

Useamman testiajon jälkeen ajat pysyvät melko samanlaisina, ainakin jos sekunneista puhutaan. Joten en tehnyt tuloksia varten mitenkään laajempaa keskiarvon laskemista usealla otannalla.

**Testi 1 – tehokkuustesti ilman indeksejä:**

Testiin yksi kului aikaa: 0.008304 sekuntia

Testiin kaksi kului aikaa: 0.0042832 sekuntia

Testiin kolme kului aikaa: 0.0038198 sekuntia

Testiin neljä kului aikaa: 2.9607771 sekuntia

Testiin viisi kului aikaa: 0.1608534 sekuntia

Testiin kuusi kului aikaa: 80.2029918sekuntia

Aikaa kului yhteensä: 83.4284598 sekuntia

**Testi 2 – tehokkuustesti indeksien kanssa:**

Testiin yksi kului aikaa: 0.002625101 sekuntia

Testiin kaksi kului aikaa: 0.002359299 sekuntia

Testiin kolme kului aikaa: 0.002378701 sekuntia

Testiin neljä kului aikaa: 11.8022639 sekuntia

Testiin viisi kului aikaa: 0.0881053 sekuntia

Testiin kuusi kului aikaa: 0.1779347 sekuntia

Aikaa kului yhteensä: 12.235067201 sekuntia

Erot ovat olemassa, mutta pääasiassa ne ovat melko pieniä, pois lukien testi kuusi, missä on jo merkittävä ero.

**Selostus, miten on varmistettu, että jokaisella paikalla ja asiakkaalla on eri nimi ja jokaisella paketilla on eri seurantakoodi (entä jos on useita samanaikaisia käyttäjiä?)**

Kuten aiemmasta selostuksesta on varmasti käynytkin ilmi, tietokannassa itsessään on varmistettu, että jokaisella paikalla, asiakkaalla on eri nimi sekä seurantakoodi on yksilöllinen. Käytännössä tämä on toteutettu taulujen luonnin yhteydessä käyttämällä ehtoa *UNIQUE*. Tämän lisäksi ohjelma myös tekee kyselyitä tietokantaan, missä varmistetaan, onko jokin asia jo lisätty sinne.

Mitä samanaikaisiin käyttäjiin tulee, Sqlite on lähtökohtaisesti tason 4 transaktion omaava. Tarkoittaa sitä, että jos sovelluksella on samanaikaisia käyttäjiä, ei samanaikaiset transaktiot onnistu.

Omien kokeilujen perusteella tietoa voidaan lisätä kyllä vaikka useammalla käyttäjällä olisi sovellus auki. Tosin tämä vaatii ohjelmalta sen, että yhteys suljetaan esimerkiksi jonkin metodin yhteydessä, kun metodi on tehnyt tarvittavan toiminnallisuuden. Sqlite lukitsee tietokannan sille käyttäjälle, jolla on yhteys siis auki, tänä aikana ei voida lisäyksiä tehdä, mutta kyselyiden tekeminen, missä haetaan tietoa, pitäisi onnistua.

Tein testejä sekä DB-viewer:lla sekä sqliten komentorivitulkilla samaan aikaan, kun oma ohjelma oli toiminnassa. Tässä yhteydessä tietokanta toimii muun muassa niin, että jos olen metodin avustuksella syöttämässä paikkaa, eli ollaan siinä vaiheessa, ettei yhteyttä vielä ole tehty, pystytään lisäämään saman niminen paikka tietokantaan, minkä jälkeen omassa ohjelmassa paikkaa ei voida enää lisätä.

Pieni huomio DB-viewerista: Ilmeisesti yhteyttä ei suljeta, jos tehdään yksikin kysely tai mikä tahansa toimenpide tietokantaan, vaan yhteys suljetaan vasta, kun sovellus suljetaan tai tietokanta suljetaan sovelluksessa. Sqliten komentorivitulkki ei tällä tavalla sentään toimi, vaan jokainen kysely tai muu vastaava tietokantaan on yksittäinen tapahtuma.

**Toteutetun sovelluksen lähdekoodi**

**Main.java**

import java.util.Scanner;

/\*

\*

\*/

/\*\*

\*

\* @author Jarno Saastamoinen

\*/

public class Main {

public static void main(String[] args) {

String polku = "jdbc:sqlite:harjoitus.db";

Scanner scanner = new Scanner(System.in);

Tietokanta tietokanta = new Tietokanta(polku);

Kayttoliittyma kayttis = new Kayttoliittyma(scanner, tietokanta);

kayttis.kaynnista();

}

}

**Kayttoliittyma.java**

import java.util.Scanner;

public class Kayttoliittyma {

private Scanner lukija;

private Tietokanta tietokanta;

public Kayttoliittyma(Scanner lukija, Tietokanta tietokanta) {

this.lukija = lukija;

this.tietokanta = tietokanta;

}

public void kaynnista() {

System.out.println("===Harjoitustyö===\n");

OUTER:

while (true) {

System.out.println("Valitse vaihtoehdoista: \n"

+ "Luettele komennot (L)\n"

+ "Anna komento (1-9)\n"

+ "Poistu ohjelmasta (X)\n");

System.out.print("Komento: ");

String komento = lukija.nextLine().trim().toUpperCase();

switch (komento) {

case "X":

break OUTER;

case "L":

System.out.println(luettele());

break;

default:

kutsuTietokantaa(komento);

System.out.println("");

break;

}

}

}

public String luettele() {

return "Komennot \n"

+ "1. Luo tietokanta ja taulut\n"

+ "2. Uusi paikka\n"

+ "3. Uusi asiakas\n"

+ "4. Uusi paketti\n"

+ "5. Uusi tapahtuma\n"

+ "6. Hae tapahtumat \n"

+ "7. Hae asiakkaan paketit ja tapahtumat\n"

+ "8. Hae paikan tapahtumamäärät\n"

+ "9. Tehokkuustesti\n";

}

public void kutsuTietokantaa(String komento) {

switch (komento) {

case "1":

tietokanta.luoTaulutTietokantaan();

break;

case "2":

System.out.print("Anna paikan nimi: ");

String paikka = lukija.nextLine().trim();

tietokanta.lisaaPaikka(paikka);

break;

case "3":

System.out.print("Anna asiakkaan nimi: ");

String asiakas = lukija.nextLine().trim();

tietokanta.lisaaAsiakas(asiakas);

break;

case "4":

System.out.print("Anna paketin seurantakoodi: ");

String koodi = lukija.nextLine();

System.out.print("Anna asiakkaan nimi: ");

String asiakasId = lukija.nextLine().trim();

tietokanta.lisaaPaketti(koodi, asiakasId);

break;

case "5":

System.out.print("Anna seurantakoodi: ");

String tapahtumaKoodi = lukija.nextLine().trim();

System.out.print("Anna paikka: ");

String tapahtumaPaikka = lukija.nextLine().trim();

System.out.print("Anna kuvaus: ");

String tapahtumaKuvaus = lukija.nextLine();

tietokanta.lisaaTapahtuma(tapahtumaKoodi, tapahtumaPaikka, tapahtumaKuvaus);

break;

case "6":

System.out.print("Anna paketin seurantakoodi: ");

String seurantaKoodi = lukija.nextLine().trim();

tietokanta.haePaketinTapahtumat(seurantaKoodi);

break;

case "7":

System.out.print("Anna asiakkaan nimi: ");

String asiakasNimi = lukija.nextLine().trim();

tietokanta.haeAsiakkaanPaketitJaTapahtumat(asiakasNimi);

break;

case "8":

System.out.print("Anna paikan nimi: ");

String paikanNimi = lukija.nextLine().trim();

System.out.print("Anna päivämäärä(pp.kk.vvvv): ");

String paivamaara = lukija.nextLine().trim();

tietokanta.haePaketinTapahtumatPaivana(paikanNimi, paivamaara);

break;

case "9":

System.out.println("Tehokkuustesti\n"

+ "(1) Testi ilman indeksiä\n"

+ "(2) Testi indeksin kanssa\n");

System.out.print("Syötä numero: ");

String testinumero = lukija.nextLine();

if (testinumero.trim().equals("1")) {

String polku = "jdbc:sqlite:testi\_ilman\_indeksia.db";

Tietokanta testi1 = new Tietokanta(polku);

testi1.poistaTaulutKannasta();

testi1.luoTaulutTietokantaan();

testi1.tehokkuustestiIlmanIndeksia();

} else if (testinumero.trim().equals("2")) {

String polku = "jdbc:sqlite:testi\_indekseilla.db";

Tietokanta testi2 = new Tietokanta(polku);

testi2.poistaTaulutKannasta();

testi2.luoTaulutTietokantaan();

testi2.tehokkuustestiIndeksilla();

}

break;

default:

break;

}

}

}

**Tietokanta.java**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Random;

public class Tietokanta {

private final String tietokannanPolku;

public Tietokanta(String polku) {

this.tietokannanPolku = polku;

}

public Connection luoYhteysJaTietokanta() {

Connection yhteys = null;

try {

yhteys = DriverManager.getConnection(tietokannanPolku);

} catch (SQLException e) {

System.out.println("Virhe yhteydessä: " + e.getMessage());

}

return yhteys;

}

public void luoTaulutTietokantaan() {

/\*1. Luo sovelluksen tarvitsemat taulut tyhjään tietokantaan

(tätä toimintoa voidaan käyttää, kun tietokantaa ei ole vielä olemassa).\*/

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

if (yhteys != null) {

Statement s = yhteys.createStatement();

s.execute("PRAGMA foreign\_keys = ON;");

s.execute("CREATE TABLE Paikat (id INTEGER PRIMARY KEY, "

+ "paikka TEXT UNIQUE NOT NULL CHECK (LENGTH(paikka) > 0));");

s.execute("CREATE TABLE Asiakas (id INTEGER PRIMARY KEY,"

+ "nimi TEXT UNIQUE NOT NULL CHECK (LENGTH(nimi) > 0));");

s.execute("CREATE TABLE Paketti (id INTEGER PRIMARY KEY,"

+ "seurantakoodi INTEGER UNIQUE NOT NULL CHECK (LENGTH(seurantakoodi) > 0),"

+ "asiakas\_id INTEGER NOT NULL REFERENCES Asiakas);");

s.execute("CREATE TABLE Tapahtuma (id INTEGER PRIMARY KEY,"

+ "seurantakoodi\_id INTEGER NOT NULL REFERENCES Paketti,"

+ "paikka\_id INTEGER NOT NULL REFERENCES Paikat,"

+ "kuvaus TEXT, paivamaara DATE);");

}

System.out.println("Tietokanta luotu");

} catch (SQLException e) {

System.out.println("Tietokanta on jo olemassa");

System.out.println("Virhe taulujen luomisessa: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public void lisaaPaikka(String paikka) {

/\*2. Lisää uusi paikka tietokantaan, kun annetaan paikan nimi.\*/

if (paikka.isEmpty()) {

System.out.println("Tyhjää paikkaa ei saa syöttää");

return;

}

int paikkaKannassa = this.haePaikkaId(paikka);

if (paikkaKannassa != -1) {

System.out.println("Paikka on jo olemassa");

return;

}

PreparedStatement stm = null;

String lisaa = "INSERT INTO Paikat (paikka) VALUES(?)";

Connection yhteys = null;

yhteys = this.luoYhteysJaTietokanta();

try {

stm = yhteys.prepareStatement(lisaa);

stm.setString(1, paikka);

stm.execute();

System.out.println("Paikka lisätty");

} catch (SQLException e) {

System.out.println("Virhe paikan lisäämisessä: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public void lisaaAsiakas(String asiakas) {

/\*3. Lisää uusi asiakas tietokantaan, kun annetaan asiakkaan nimi.\*/

if (asiakas.isEmpty()) {

System.out.println("Tyhjää riviä ei saa syöttää.");

return;

}

if (this.haeAsiakasId(asiakas) != -1) {

System.out.println("Asiakas on jo olemassa.");

return;

}

String lisaa = "INSERT INTO Asiakas (nimi) VALUES(?)";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement stm = yhteys.prepareStatement(lisaa);

stm.setString(1, asiakas);

stm.execute();

System.out.println("Asiakas lisätty");

} catch (SQLException e) {

System.out.println("Virhe asiakkaan lisäämisessä: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public void lisaaPaketti(String seurantakoodi, String asiakas) {

/\*4.Lisää uusi paketti tietokantaan, kun annetaan paketin seurantakoodi

ja asiakkaan nimi. \*/

int muunnosLuvuksi = 0;

try {

muunnosLuvuksi = Integer.valueOf(seurantakoodi);

} catch (NumberFormatException e) {

System.out.println("Syöte tulee olla numeromuodossa");

return;

}

if (muunnosLuvuksi < 1) {

System.out.println("Numero ei saa olla negatiivinen.");

return;

}

int koodiKannassa = this.haeSeurantakoodi(seurantakoodi);

if (koodiKannassa != -1) {

System.out.println("Seurantakoodi on jo tietokannassa");

return;

}

int haettavaId = 0;

if (!asiakas.isEmpty()) {

String nimiKannassa = haeAsiakasNimi(asiakas);

if (asiakas.equals(nimiKannassa)) {

haettavaId = this.haeAsiakasId(asiakas);

} else {

System.out.println("Asiakasta ei löydy");

return;

}

} else {

System.out.println("Asiakkaan nimi ei saa olla tyhjä");

return;

}

String lisaa = "INSERT INTO Paketti (seurantakoodi, asiakas\_id) VALUES (?,?)";

Connection yhteys = null;

yhteys = this.luoYhteysJaTietokanta();

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement stm = yhteys.prepareStatement(lisaa);

stm.setInt(1, muunnosLuvuksi);

stm.setInt(2, haettavaId);

stm.execute();

System.out.println("Paketti lisätty");

} catch (SQLException e) {

System.out.println("Seurantakoodi virheellinen:" + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public void lisaaTapahtuma(String seurantakoodi, String paikka, String kuvaus) {

/\*5. Lisää uusi tapahtuma tietokantaan, kun annetaan paketin seurantakoodi,

tapahtuman paikka sekä kuvaus.

Paketin ja paikan tulee olla valmiiksi tietokannassa.\*/

if (seurantakoodi.isEmpty()) {

System.out.println("Seurantakoodi ei saa olla tyhjä");

return;

}

if (paikka.isEmpty()) {

System.out.println("Paikka ei saa olla tyhjä");

return;

}

int haettavaKoodi = 0;

try {

haettavaKoodi = Integer.valueOf(seurantakoodi);

} catch (NumberFormatException e) {

System.out.println("Seurantakoodi täytyy olla numeromuodossa.");

return;

}

int koodiKannassa = this.haeSeurantakoodi(seurantakoodi);

if (koodiKannassa == -1) {

System.out.println("Tarkista seurantakoodi");

return;

}

int paikkaKannassa = this.haePaikkaId(paikka);

if (paikkaKannassa == -1) {

System.out.println("Tarkista paikka");

return;

}

String lisaa = "INSERT INTO Tapahtuma (seurantakoodi\_id, paikka\_id, kuvaus, paivamaara)

VALUES (?,?,?,?)";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setInt(1, koodiKannassa);

kysely.setInt(2, paikkaKannassa);

kysely.setString(3, kuvaus);

kysely.setString(4, haeAika());

kysely.execute();

System.out.println("Tapahtuman lisäys onnistui.");

} catch (SQLException e) {

System.out.println("Virhe tapahtuman lisäämisessä: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public void haePaketinTapahtumat(String koodi) {

/\*6. Hae kaikki paketin tapahtumat seurantakoodin perusteella.\*/

if (koodi.isEmpty()) {

System.out.println("Seurantakoodi ei saa olla tyhjä");

return;

}

int haettavaKoodi = 0;

try {

haettavaKoodi = Integer.valueOf(koodi);

} catch (NumberFormatException e) {

System.out.println("Seurantakoodi täytyy olla numeromuodossa");

return;

}

int koodiKannassa = this.haeSeurantakoodi(koodi);

if (koodiKannassa == -1) {

System.out.println("Tarkista seurantakoodi");

return;

}

String lisaa

= "SELECT paivamaara, paikka, kuvaus \n"

+ "FROM Tapahtuma LEFT JOIN Paikat, Paketti \n"

+ "ON Paketti.id = seurantakoodi\_id AND paikka\_id = Paikat.id\n"

+ "AND Paketti.seurantakoodi =?\n"

+ "GROUP BY paivamaara;";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setInt(1, haettavaKoodi);

ResultSet tulos = kysely.executeQuery();

if (tulos.getString("paivamaara").equals(null)) {

}

while (tulos.next()) {

System.out.println(tulos.getString("paivamaara") + ", "

+ tulos.getString("paikka") + ", "

+ tulos.getString("kuvaus"));

}

} catch (SQLException ex) {

System.out.println("Ei tapahtumia");

System.out.println("Virhe tapahtumien hakemisessa: " + ex.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa");

}

}

}

}

public void haeAsiakkaanPaketitJaTapahtumat(String asiakas) {

/\*7. Hae kaikki asiakkaan paketit ja niihin liittyvien tapahtumien määrä.\*/

if (asiakas.isEmpty()) {

System.out.println("Asiakkaan nimi ei saa olla tyhjä");

return;

}

int nimiKannassa = this.haeAsiakasId(asiakas);

if (nimiKannassa == -1) {

System.out.println("Tarkista asiakkaan nimi");

return;

}

String lisaa

= "SELECT Pa.seurantakoodi, COUNT(T.seurantakoodi\_id)\n"

+ "FROM Paketti Pa LEFT JOIN Tapahtuma T\n"

+ "ON Pa.id = T.seurantakoodi\_id\n"

+ "LEFT JOIN Asiakas A ON A.id = Pa.asiakas\_id WHERE A.nimi =?\n"

+ "GROUP BY Pa.seurantakoodi\n"

+ "ORDER BY COUNT(T.seurantakoodi\_id) DESC;";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setString(1, asiakas);

ResultSet tulos = kysely.executeQuery();

if (tulos.getString("seurantakoodi").equals(null)) {

}

while (tulos.next()) {

System.out.println(tulos.getInt("seurantakoodi")

+ ", " + tulos.getInt("COUNT(T.seurantakoodi\_id)") + " tapahtumaa");

}

} catch (SQLException e) {

System.out.println("Ei tapahtumia");

System.out.println("Virhe: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa");

}

}

}

}

public void haePaketinTapahtumatPaivana(String paikka, String paivamaara) {

/\*8. Hae annetusta paikasta tapahtumien määrä tiettynä päivänä.\*/

if (paikka.isEmpty()) {

System.out.println("Paikka ei saa olla tyhjä");

}

if (paivamaara.isEmpty()) {

System.out.println("Päivämäärä ei saa olla tyhjä");

return;

}

int paikkaKannassa = this.haePaikkaId(paikka);

if (paikkaKannassa == -1) {

return;

}

int paivamaaraKannassa = this.haePaivamaaraId(paivamaara);

if (paivamaaraKannassa == -1) {

return;

}

Connection yhteys = null;

String lisaa = "SELECT COUNT(T.id)\n"

+ "FROM Tapahtuma T LEFT JOIN Paikat P\n"

+ "ON T.paikka\_id = P.id WHERE P.paikka =?\n"

+ "AND T.paivamaara LIKE ?";

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setString(1, paikka);

kysely.setString(2, paivamaara + "%");

ResultSet tulos = kysely.executeQuery();

while (tulos.next()) {

System.out.println("Tapahtumien määrä: " + tulos.getInt("COUNT(T.id)"));

}

} catch (SQLException e) {

System.out.println("Virhe jossain: " + e.getMessage());

}

}

public void tehokkuustestiIlmanIndeksia() {

/\*9.1 Suorita tietokannan tehokkuustesti ilmman indeksejä.\*/

PreparedStatement stm = null;

String lisaaPaikka = "INSERT INTO Paikat (paikka) VALUES(?)";

String lisaaAsiakas = "INSERT INTO Asiakas (nimi) VALUES(?)";

String lisaaPaketti = "INSERT INTO Paketti (seurantakoodi, asiakas\_id) VALUES (?,?)";

String lisaaTapahtuma = "INSERT INTO Tapahtuma (seurantakoodi\_id, paikka\_id, kuvaus, paivamaara) VALUES (?,?,?,?)";

String haePakettienMaara = "SELECT COUNT(seurantakoodi) FROM Paketti WHERE asiakas\_id = (?)";

String haeTapahtumienMaara = "SELECT COUNT(seurantakoodi\_id) FROM Tapahtuma WHERE seurantakoodi\_id = (?)";

Connection yhteys = null;

yhteys = this.luoYhteysJaTietokanta();

try {

//lisätään paikat

long tehokkuustestiAlku = System.nanoTime();

Statement s = yhteys.createStatement();

s.execute("BEGIN TRANSACTION");

stm = yhteys.prepareStatement(lisaaPaikka);

for (int i = 1; i <= 1000; i++) {

stm.setString(1, "P" + i);

stm.execute();

}

System.out.println("Paikat lisätty");

long tehokkuustestiYksiLoppu = System.nanoTime();

//lisätaan asiakkaat

long tehokkuustestiKaksiAlku = System.nanoTime();

stm = yhteys.prepareStatement(lisaaAsiakas);

for (int i = 1; i < 1000; i++) {

stm.setString(1, "A" + i);

stm.execute();

}

System.out.println("Asiakkaat lisätty");

long tehokkuustestiKaksiLoppu = System.nanoTime();

//lisätään paketit

long tehokkuustestiKolmeAlku = System.nanoTime();

stm = yhteys.prepareStatement(lisaaPaketti);

for (int i = 1; i <= 1000; i++) {

int arvoAsiakas = this.arvo(1000);

stm.setInt(1, i);

stm.setInt(2, arvoAsiakas);

stm.execute();

}

System.out.println("Paketit lisätty");

long tehokkuustestiKolmeLoppu = System.nanoTime();

//lisätäään miljoona tapahtumaa

long tehokkuustestiNeljaAlku = System.nanoTime();

stm = yhteys.prepareStatement(lisaaTapahtuma);

for (int i = 1; i <= 1000000; i++) {

//stm.setInt(1, 1);

int arvoSeurantakoodi = this.arvo(1000);

stm.setInt(1, arvoSeurantakoodi);

stm.setInt(2, 1);

stm.setString(3, "Testi" + i);

stm.setString(4, this.haeAika());

stm.execute();

}

System.out.println("Tapahtumat lisätty");

long tehokkuustestiNeljaLoppu = System.nanoTime();

s.execute("COMMIT");

long tehokkuustestiViisiAlku = System.nanoTime();

//Suoritetaan tuhat kyselyä, joista jokaisessa haetaan jonkin asiakkaan pakettien määrä.

stm = yhteys.prepareStatement(haePakettienMaara);

for (int i = 1; i <= 1000; i++) {

int arvottuLuku = this.arvo(1000);

stm.setInt(1, arvottuLuku);

ResultSet tulos = stm.executeQuery();

while (tulos.next()) {

System.out.println("Pakettien määrä: " + tulos.getInt("COUNT(seurantakoodi)"));

}

}

long tehokkuustestiViisiLoppu = System.nanoTime();

long tehokkuustestiKuusiAlku = System.nanoTime();

//Suoritetaan tuhat kyselyä, joista jokaisessa haetaan jonkin paketin tapahtumien määrä.

stm = yhteys.prepareStatement(haeTapahtumienMaara);

for (int i = 1; i <= 1000; i++) {

int arvottuLuku = this.arvo(1000);

stm.setInt(1, arvottuLuku);

ResultSet tulos = stm.executeQuery();

while (tulos.next()) {

System.out.println("Tapahtumien määrä: " + tulos.getInt("COUNT(seurantakoodi\_id)"));

}

}

long tehokkuustestiKuusiLoppu = System.nanoTime();

long tehokkuustestiLoppu = System.nanoTime();

System.out.println("Testiin yksi kului aikaa: " + (tehokkuustestiYksiLoppu - tehokkuustestiAlku) / 1e9 + " sekuntia");

System.out.println("Testiin kaksi kului aikaa: " + (tehokkuustestiKaksiLoppu - tehokkuustestiKaksiAlku) / 1e9 + " sekuntia");

System.out.println("Testiin kolme kului aikaa: " + (tehokkuustestiKolmeLoppu - tehokkuustestiKolmeAlku) / 1e9 + " sekuntia");

System.out.println("Testiin neljä kului aikaa: " + (tehokkuustestiNeljaLoppu - tehokkuustestiNeljaAlku) / 1e9 + " sekuntia");

System.out.println("Testiin viisi kului aikaa: " + (tehokkuustestiViisiLoppu - tehokkuustestiViisiAlku) / 1e9 + " sekuntia");

System.out.println("Testiin kuusi kului aikaa: " + (tehokkuustestiKuusiLoppu - tehokkuustestiKuusiAlku) / 1e9 + " sekuntia");

System.out.println("Aikaa kului yhteensä: " + (tehokkuustestiLoppu - tehokkuustestiAlku) / 1e9 + " sekuntia");

} catch (SQLException e) {

System.out.println("Virhe: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public void tehokkuustestiIndeksilla() {

//9.2 suorita tehokkustesti indekseillä

PreparedStatement stm = null;

String lisaaPaikka = "INSERT INTO Paikat (paikka) VALUES(?)";

String lisaaAsiakas = "INSERT INTO Asiakas (nimi) VALUES(?)";

String lisaaPaketti = "INSERT INTO Paketti (seurantakoodi, asiakas\_id) VALUES (?,?)";

String lisaaTapahtuma = "INSERT INTO Tapahtuma (seurantakoodi\_id, paikka\_id, kuvaus, paivamaara) VALUES (?,?,?,?)";

String haePakettienMaara = "SELECT COUNT(seurantakoodi) FROM Paketti WHERE asiakas\_id = (?)";

String haeTapahtumienMaara = "SELECT COUNT(seurantakoodi\_id) FROM Tapahtuma WHERE seurantakoodi\_id = (?)";

String pakettiIndeksi = "CREATE INDEX idx\_seurantakoodi ON Paketti (asiakas\_id)";

String tapahtumaIndeksi = "CREATE INDEX idx\_seurantakoodi\_id ON Tapahtuma (seurantakoodi\_id)";

Connection yhteys = null;

yhteys = this.luoYhteysJaTietokanta();

try {

//lisätään indeksitä

stm = yhteys.prepareStatement(pakettiIndeksi);

stm.execute();

stm = yhteys.prepareStatement(tapahtumaIndeksi);

stm.execute();

//lisätään paikat

long tehokkuustestiAlku = System.nanoTime();

Statement s = yhteys.createStatement();

s.execute("BEGIN TRANSACTION");

stm = yhteys.prepareStatement(lisaaPaikka);

for (int i = 1; i <= 1000; i++) {

stm.setString(1, "P" + i);

stm.execute();

}

System.out.println("Paikat lisätty");

long tehokkuustestiYksiLoppu = System.nanoTime();

//lisätaan asiakkaat

long tehokkuustestiKaksiAlku = System.nanoTime();

stm = yhteys.prepareStatement(lisaaAsiakas);

for (int i = 1; i < 1000; i++) {

stm.setString(1, "A" + i);

stm.execute();

}

System.out.println("Asiakkaat lisätty");

long tehokkuustestiKaksiLoppu = System.nanoTime();

//lisätään paketit

long tehokkuustestiKolmeAlku = System.nanoTime();

stm = yhteys.prepareStatement(lisaaPaketti);

for (int i = 1; i <= 1000; i++) {

int arvoAsiakasId = this.arvo(1000);

stm.setInt(1, i);

stm.setInt(2, arvoAsiakasId);

stm.execute();

}

System.out.println("Paketit lisätty");

long tehokkuustestiKolmeLoppu = System.nanoTime();

//lisätäään miljoona tapahtumaa

long tehokkuustestiNeljaAlku = System.nanoTime();

stm = yhteys.prepareStatement(lisaaTapahtuma);

for (int i = 1; i <= 1000000; i++) {

int arvoSeurantakoodi = this.arvo(1000);

//stm.setInt(1, 1);

stm.setInt(1, arvoSeurantakoodi);

stm.setInt(2, 1);

stm.setString(3, "Testi" + i);

stm.setString(4, this.haeAika());

stm.execute();

}

System.out.println("Tapahtumat lisätty");

long tehokkuustestiNeljaLoppu = System.nanoTime();

s.execute("COMMIT");

long tehokkuustestiViisiAlku = System.nanoTime();

//Suoritetaan tuhat kyselyä, joista jokaisessa haetaan jonkin asiakkaan pakettien määrä.

stm = yhteys.prepareStatement(haePakettienMaara);

for (int i = 1; i <= 1000; i++) {

int arvottuLuku = this.arvo(1000);

stm.setInt(1, arvottuLuku);

ResultSet tulos = stm.executeQuery();

while (tulos.next()) {

System.out.println("Pakettien määrä: " + tulos.getInt("COUNT(seurantakoodi)"));

}

}

long tehokkuustestiViisiLoppu = System.nanoTime();

long tehokkuustestiKuusiAlku = System.nanoTime();

//Suoritetaan tuhat kyselyä, joista jokaisessa haetaan jonkin paketin tapahtumien määrä.

stm = yhteys.prepareStatement(haeTapahtumienMaara);

for (int i = 1; i <= 1000; i++) {

int arvottuLuku = this.arvo(1000);

stm.setInt(1, arvottuLuku);

ResultSet tulos = stm.executeQuery();

while (tulos.next()) {

System.out.println("Tapahtumien määrä: " + tulos.getInt("COUNT(seurantakoodi\_id)"));

}

}

long tehokkuustestiKuusiLoppu = System.nanoTime();

long tehokkuustestiLoppu = System.nanoTime();

System.out.println("Testiin yksi kului aikaa: " + (tehokkuustestiYksiLoppu - tehokkuustestiAlku) / 1e9 + " sekuntia");

System.out.println("Testiin kaksi kului aikaa: " + (tehokkuustestiKaksiLoppu - tehokkuustestiKaksiAlku) / 1e9 + " sekuntia");

System.out.println("Testiin kolme kului aikaa: " + (tehokkuustestiKolmeLoppu - tehokkuustestiKolmeAlku) / 1e9 + " sekuntia");

System.out.println("Testiin neljä kului aikaa: " + (tehokkuustestiNeljaLoppu - tehokkuustestiNeljaAlku) / 1e9 + " sekuntia");

System.out.println("Testiin viisi kului aikaa: " + (tehokkuustestiViisiLoppu - tehokkuustestiViisiAlku) / 1e9 + " sekuntia");

System.out.println("Testiin kuusi kului aikaa: " + (tehokkuustestiKuusiLoppu - tehokkuustestiKuusiAlku) / 1e9 + " sekuntia");

System.out.println("Aikaa kului yhteensä: " + (tehokkuustestiLoppu - tehokkuustestiAlku) / 1e9 + " sekuntia");

} catch (SQLException e) {

System.out.println("Virhe: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

}

public String haeAsiakasNimi(String nimi) {

String haettuNimi = "";

String lisaa = "SELECT nimi FROM Asiakas WHERE nimi=?";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setString(1, nimi);

ResultSet tulos = kysely.executeQuery();

haettuNimi = tulos.getString("nimi");

} catch (SQLException e) {

System.out.println("Hae nimellä virhe: " + e.getMessage());

} finally {

try {

if (yhteys != null) {

yhteys.close();

}

} catch (SQLException ex) {

System.out.println(ex.getMessage());

}

}

return haettuNimi;

}

public int haeAsiakasId(String nimi) {

int asiakasId = -1;

String lisaa = "SELECT id FROM Asiakas WHERE nimi=?";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setString(1, nimi);

ResultSet tulos = kysely.executeQuery();

asiakasId = tulos.getInt("id");

} catch (SQLException e) {

System.out.println("Asiakasta ei löytynyt tietokannasta");

} finally {

try {

if (yhteys != null) {

yhteys.close();

}

} catch (SQLException ex) {

System.out.println(ex.getMessage());

}

}

return asiakasId;

}

public int haePaikkaId(String paikka) {

int palautaId = -1;

String lisaa = "SELECT id FROM Paikat WHERE paikka=?";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setString(1, paikka);

ResultSet tulos = kysely.executeQuery();

palautaId = tulos.getInt("id");

} catch (SQLException e) {

System.out.println("Paikkaa ei löytynyt tietokannasta.");

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println(ex.getMessage());

}

}

}

return palautaId;

}

public int haeSeurantakoodi(String koodi) {

int koodiId = -1;

int haettavaKoodi = 0;

try {

haettavaKoodi = Integer.valueOf(koodi);

} catch (NumberFormatException e) {

System.out.println("Virhe: " + e.getMessage());

}

String lisaa = "SELECT id FROM Paketti WHERE seurantakoodi=?";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setInt(1, haettavaKoodi);

ResultSet tulos = kysely.executeQuery();

koodiId = tulos.getInt("id");

} catch (SQLException e) {

System.out.println("Seurantakoodia ei löytynyt");

//System.out.println("Virhe seurantaKoodi: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

return koodiId;

}

public int haePaivamaaraId(String paivamaara) {

int paikkaId = -1;

String lisaa = "SELECT id FROM Tapahtuma WHERE paivamaara LIKE ?";

Connection yhteys = null;

try {

yhteys = this.luoYhteysJaTietokanta();

PreparedStatement kysely = yhteys.prepareStatement(lisaa);

kysely.setString(1, paivamaara + "%");

ResultSet tulos = kysely.executeQuery();

paikkaId = tulos.getInt("id");

} catch (SQLException e) {

System.out.println("Päivämäärää ei löytynyt.");

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemisessa: " + ex.getMessage());

}

}

}

return paikkaId;

}

public String haeAika() {

LocalDateTime aika = LocalDateTime.now();

DateTimeFormatter muuta = DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss");

String aikanyt = aika.format(muuta);

return aikanyt;

}

public void poistaTaulutKannasta() {

Connection yhteys = null;

String poistaAsiakas = "DROP TABLE IF EXISTS Asiakas;";

String poistaPaikat = "DROP TABLE IF EXISTS Paikat;";

String poistaPaketti = "DROP TABLE IF EXISTS Paketti;";

String poistaTapahtuma = "DROP TABLE IF EXISTS Tapahtuma;";

try {

yhteys = this.luoYhteysJaTietokanta();

Statement stm = yhteys.createStatement();

stm.execute(poistaAsiakas);

stm.execute(poistaPaikat);

stm.execute(poistaPaketti);

stm.execute(poistaTapahtuma);

} catch (SQLException e) {

System.out.println("Virhe taulua poistaessa: " + e.getMessage());

} finally {

if (yhteys != null) {

try {

yhteys.close();

} catch (SQLException ex) {

System.out.println("Virhe yhteyden sulkemmisessa: " + ex.getMessage());

}

}

}

}

public int arvo(int luku) {

Random numero = new Random();

return numero.nextInt(luku) + 1;

}

}