Harjoitustyön raportti

Iira Häkkinen, 014923869, hiira

27.02.2020

1 Tiivistelmä

Tässä raportissa tarkastellaan Tietokantojen perusteet -kurssin harjotustyönä toteuttamani pakettiseurantasovellusta. Raportissa käydään läpi ohjelman toiminnallisuus, siihen liittyvän tietokannan rakenne, tarkastellaan ohjelman tehokkuustestiä sekä selostetaan, kuinka on varmistuttu siitä, että tietokantaan liittyvät tiedot, kuten asiakkaiden ja pakettien nimet, ovat yksilöllisiä.

2 Harjoitustyön toiminnot

Pakettiseurantaohjelmaan on toteutettu ohjeistuksen vaatimat toiminnot käyttäjän komentoriville syöttäminä numerokomentoina 1-9. Lisäksi valittiin, että komennolla 0 ohjelman suoritus lopetetaan. Numerokomennot 1-9 ovat listattuna alla ja niiden toimintaaa selostetaan hieman listauksen jälkeen.

1 - luo taulut tietokantaan
2 - lisää uusi paikka tietokantaan
3 - lisää uusi asiakas teitokantaan
4 - lisää uusi paketti tietokantaan
5 - lisää uusi tapahtuma tietokantaan
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

Komento 1 luo luo taulut Asiakkaat, Paikat, Paketit ja Tapahtumat tietokantaan jos niitä ei siellä vielä ole. Komennot 2-5 lisäävät uuden paikan (komento 2), asiakkaan (komento 3), paketin (komento 4) tai tapahtuman (komento 5) tietokantaan. Uuden paikan ja asiakkaan lisäämisessä varmistetaan, ettei vastaavalla nimellä löydy jo aisakasta tai paikkaa tietokannasta. Myöskin paketin lisäämisessä varmistetaan, ettei annettu seurantakoodi ole jo käytössä, jonka lisäksi tarkastetaan, että paketille annettu asiakas löytyy tietokannasta. Vastaavalla tavalla Tapahtumaa lisättäessä annetun paketin ja paikan on löydyttävä tietokannasta.

Komennoilla 6-8 suoritetaan erilaisia hakuja tietokannasta. Komento 6 hakee kaikki tietyn paketin tapahtumat, kun käyttäjä antaa syötteenä tietokannasta löytyvän paketin seurantakoodin. Komennolla 7 haetaan syötteenä saadun, tietokannasta löytyvän asiakkaan paketit, sekä tiedon siitä, kuinka monta tapahtumaa kuhunkin pakettiin liittyy. Komento 8 tulostaa kaikkien tapahtumien määrän jossain tietyssä paikassa tiettynä päivänä, kun käyttäjä syöttää paikan ja päivämäärän, edellyttäen, että annettu paikka löytyy tietokannasta.

Komento 9 suorittaa tehokkuustestin kahdella eri tavalla: ilman indeksointia ja indekseillä. Molemmissa tapauksissa tietokantaan lisätään tuhat paikkaa, asiakasta ja pakettia sekä miljoona tapahtumaa. Tämän jälkeen haetaan tuhannen asiakkaan paketit ja tuhannen paketin tapahtumat. Jokaiseen vaiheeseen kuluva aika mitataan erikseen ja tulostetaan. Tehokkuustesti käyttää omaa tietokaantaansa (tehokkuustestikanta.db), jotta testin suoritus ei sotkeutuisi muun ohjelman suoritukseen.

Lisäksi ohjelma sisältää lähinnä testauksen helpottamiseksi lisäkomennot 10-11, jotka eivät tulostu ohjelman suoritusessa valikkoon, vaan komennot ja niiden toiminnot on tiedettävä ohjelmakoodin perusteella. Komento 10 tulostaa tietokannan sisällön ja komennolla 11 poistetaan tietokanta ja sen sisältö.

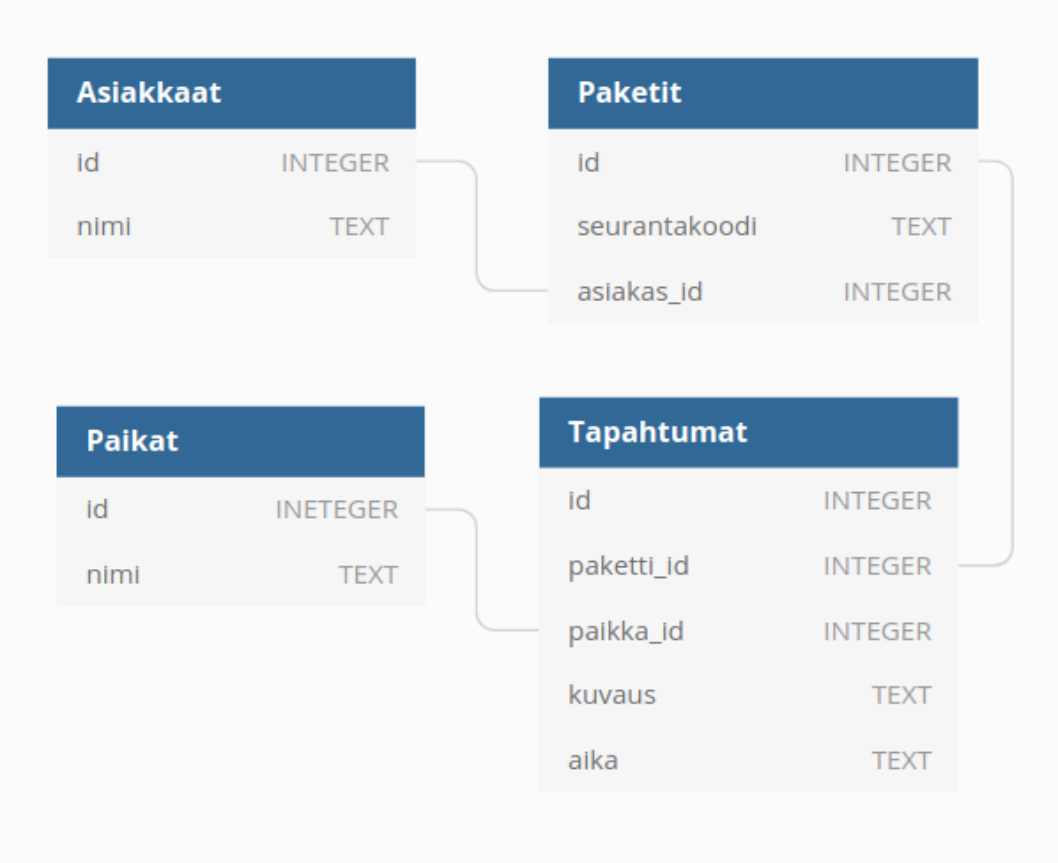
Mikäli käyttäjä yrittää syöttää jonkin muun komennon, kun mitä yllä on listattu, tulostuu viesti *"Virheellinen komento, anna numero 0-9"* ja käyttäjää pyydetään antamaan uusi komento. Yleisestikin ohjelman suorituksessa on varauduttu virhetilanteisiin

tulostamalla aina virheen sattuessa jokin selkeä itse kirjoitettu virheviesti ja SQL:n oma virheilmoitus virhekoodeineen. Virheenkäsittelyä olisi voitu vielä parantaa siirtämällä kaikki tulostukset ja virheenkäsittely omaan luokkaansa, jotta tietokantatoimintoja käsittelevä luokka (*DAO.java*) olisi mahdollisimman selkeä ja ajaisi vain ja ainoastaan oman tarkoituksensa.

3 Tietokantakaavio ja SQL-skeema

3.1 Tietokantakaavio

Tietokanta sisältää taulut Paikat, Asiakkaat, Paketit ja Tapahtumat. Sekä taulussa Paikat että Asiakkaat on sarakkeet id ja nimi. Taulussa Paketit on sarakkeet id, seurantakoodi ja asiakas_id. Tapahtumat-taulussa sarakkeet ovat id, paketti_id, paikka_id, kuvaus ja aika.



3.2 SQL-skeema

SQL-skeemassa on eriteltynä tietokannan luominen. Skeemasta voidaan nähdä, että asiakkaan nimellä, paikan nimellä ja paketin seurantakoodilla on määre UNIQUE, jonka avulla SQLite varmistaa, ettei näissä sarakkeissa esiinny sama arvo useampaan kertaan.

```
CREATE TABLE Asiakkaat (id INTEGER PRIMARY KEY, nimi TEXT UNIQUE);

CREATE TABLE Paikat (id INTEGER PRIMARY KEY, nimi TEXT UNIQUE);

CREATE TABLE Paketit (id INTEGER PRIMARY KEY, seurantakoodi TEXT UNIQUE,
    ↪ asiakas_id INTEGER);

CREATE TABLE Tapahtumat (id INTEGER PRIMARY KEY, paketti_id INTEGER,
    ↪ paikka_id INTEGER, kuvaus TEXT, aika TEXT);
```

4 Tehokkuustestin tulokset

Tehokkuustestin tulosten listaamista raporttia varten tehokkuustesti suoritettiin 3 kertaa, minkä jälkeen tuloksista otettiin keskiarvot. Tulosten keskiarvot ilman indeksejä olivat:

Paikkojen (1000 kpl) lisäämiseen kului: 0.010795 s
Asiakkaiden (1000 kpl) lisäämiseen kului: 0.00462 s
Pakettien (1000 kpl) lisäämiseen kului: 0.00644 s
Miljoonan tapahtumaan lisäämiseen kului: 7.58048 s
Tuhannen asiakkaan pakettien hakemiseen kului: 0.10667 s
Tuhannen paketin tapahtumien määrän hakeminen: 156.43491 s

Indeksien kanssa keskiarvot olivat:

Paikkojen (1000 kpl) lisäämiseen kului: 0.01103 s
Asiakkaiden (1000 kpl) lisäämiseen kului: 0.00875 s
Pakettien (1000 kpl) lisäämiseen kului: 0.00915 s
Miljoonan tapahtumaan lisäämiseen kului: 24.18028 s
Tuhannen asiakkaan pakettien hakemiseen kului: 0.03941 s
Tuhannen paketin tapahtumien määrän hakeminen: 0.12375 s

Tuloksista voidaan havaita selkeästi, että indeksien kanssa hakujen suorittaminen tietokantaan on huomattavasti nopeampaa, sillä hakua suorittaessa ei tarvitse käydä kaikkia taulun rivejä läpi, koska voidaan hyödyntää tietoa indeksistä. Havaittavissa on myös, että indekseillä miljoonan tapahtuman lisäämiseen kuluu hieman pidempään kuin ilman indeksejä, sillä indeksien luomiseen kuluu aikaa. Kuitenkin indekseillä hakujen suorittamiseen saatava nopeusetu on niin huomattava, että indeksien käyttäminen on hyödyllistä.

Erityisesti tehokkuustestin SQL-kyselyjä toteutettaessa huomattiin, että kyselyn rakenteellakin oli vaikutusta toteutukseen kuluneeseen aikaan. Pakettien tapahtumia haettaessa havaittiin, että alikyselyjen käyttäminen toimi kaikista kokeilluista vaihtoehdoista nopeiten. Syytä siihen, että miksi näin on ei saatu selville. Syyn selvittäminen vaatisi parempaa perehtymistä SQLite:n sisäiseen toimintaan.

5 Tiedon yksilöllisyyden varmistaminen

Tietokannassa on sallittua olla vain yksi samanniminen paikka ja asiakas, sekä jokaista seurantakoodia voi vastata vain yksi paketti. Tämä on varmistettu jokaisen asiakkaan, paketin ja paikan luomisen yhteydessä niin, että ennen uuden arvon lisäämistä tietokantaan tehdään haku kyseisellä nimellä tai seurantakoodilla, ja mikäli haulla on tuloksia (tarkemmin ottaen yksi tulos, sillä tietokannasta ei voi löytyä kuin yksi rivi annetulla nimellä/seurantakoodilla), lähetetään käyttäjälle virheviesti ja lisäysmetodin suoritus lopetetaan. Kuten jo SQL-skeeman yhteydessä mainittiin, myöskin tietokannan luonnin yhteydessä on jo näitä kolmea saraketta määreen UNIQUE avulla rajattu siten, että kussakin sarakkeessa saa esiintyä jokin tietty arvo vain kerran. Aiemmin mainittu manuaalinen tarkastus lisäämisen kanssa samassa transaktiossa on toteutettu lähinnä virheen tulostamista varten.

Toteutuksessa on huomioitu tilanne, jossa käyttäjät 1 ja 2 ovat molemmat samaan aikaan lisäämässä esimerkiksi asiakasta nimeltä 'Pertti'. Tilanteessa molemmat ehtivät hakemaan tietokannasta tiedon siitä, että Pertti-nimistä henkilöä ei kannassa vielä ole, jolloin molemmat pääsevät lisäämään asiakkaan tietokantaan. Tietokannassa olisi tällöi kaksi asiakasta nimellä 'Pertti'. Tämä ei ole suotavaa, joten ohjelmakoodissa lisättävän asiakkaan nimen, paikan nimen ja paketin seurantakoodin olemassaolon tarkastus ja luominen on pakotettu saman transaktion sisälle, jolloin tietokantayhteys on varattu näiden kahden toiminnon suorittamiseen koko ajaksi, eikä vahinkoa pääse tapahtumaan.

Tietokantasovelluksen käyttö onnistuu rinnakkain niin kauan, kunnes yritetään tehdä muutoksia tauluun, joka on jo jollain toisella käyttäjällä varattuna. Tällöin törmätään virheilmoitukseen ja ohjelman suoritus katkeaa. Mikäli tästä haluttaisiin päästä vielä eroon, tulisi sovellus tehdä siten, että se osaa odottaa vuoroaan päästä lukemaan tai muokkaamaan taulua.

6 Sovelluksen lähdekoodi

Pääohjelma - Main.java

```
package harjoitustyo;

import java.sql.*;
import java.util.Scanner;


 // @author hiira

public class Main {
    private static Scanner lukija = new Scanner(System.in);
    private static DAO kanta = new DAO("htkanta.db");

    private static void lisaaPaikka() {
```

```
        System.out.print("Anna paikan nimi: ");
        String paikanNimi = lukija.nextLine();
        kanta.lisaaPaikka(paikanNimi);
    }

    private static void lisaaAsiakas() {
        System.out.print("Anna asiakkaan nimi: ");
        String asiakkaanNimi = lukija.nextLine();
        kanta.lisaaAsiakas(asiakkaanNimi);
    }

    private static void lisaaPaketti() {
        System.out.print("Anna lisättävän paketin seurantakoodi: ");
        String seurantakoodi = lukija.nextLine();
        System.out.print("Anna asiakkaan nimi: ");
        String nimi = lukija.nextLine();
        kanta.lisaaPaketti(seurantakoodi,nimi);
    }

    private static void lisaaTapahtuma() {
        System.out.print("Anna paketin seurantakoodi: ");
        String koodi = lukija.nextLine();
        System.out.print("Anna tapahtuman paikka: ");
        String paikka = lukija.nextLine();
        System.out.print("Anna tapahtuman kuvaus: ");
        String kuvaus = lukija.nextLine();
        kanta.lisaaTapahtuma(koodi,paikka,kuvaus);
    }

    private static void haePaketinTapahtumat() {
        System.out.println("Paketin tapahtumien haku.");
        System.out.print("Anna paketin seurantakoodi: ");
        String koodi = lukija.nextLine();
        kanta.haePaketinTapahtumat(koodi);
    }

    private static void haeAsiakkaanPaketit() {
        System.out.println("Asiakkaan pakettien haku.");
        System.out.print("Anna asiakkaan nimi: ");
        String asiakas = lukija.nextLine();
        kanta.haeAsiakkaanPaketit(asiakas);
    }

    private static void haePaikanTapahtumat() {
        System.out.println("Paikan tapahtumien haku tiettynä päivänä.");
        System.out.print("Anna paikan nimi: ");
        String paikka = lukija.nextLine();
        System.out.print("Anna päivämäärä (d.m.yyyy): ");
        String paiva = lukija.nextLine();
```

```
        kanta.haePaikanTapahtumat(paikka,paiva);
    }

    public static void main(String[] args) {
            //valintavalikko
            System.out.println("PAKETTISEURANTA");
            System.out.println("");
            System.out.println("1 - luo taulut tietokantaan");
            System.out.println("2 - lisää uusi paikka tietokantaan");
            System.out.println("3 - lisää uusi asiakas tietokantaan");
            System.out.println("4 - lisää uusi paketti tietokantaan");
            System.out.println("5 - lisää uusi tapahtuma tietokantaan");
            System.out.println("6 - hae kaikki paketin tapahtumat
                ↪ seurantakoodin perusteella");
            System.out.println("7 - hae kaikki asiakkaan paketit ja niihin
                ↪ liittyvien tapahtumien määrä");
            System.out.println("8 - hae annetusta paikasta tapahtumien määrä
                ↪  tiettynä päivänä");
            System.out.println("9 - suorita tietokannan tehokkuustesti");
            System.out.println("0 - lopeta ohjelman suoritus");

        while (true) {

            System.out.println("");
            System.out.print("Valitse toiminto: ");
            String komento = lukija.nextLine();
            //int komento = Integer.valueOf(syote);

            System.out.println("");

            switch (komento) {
                case "1":
                    kanta.luoTaulut();
                    break;
                case "2":
                    lisaaPaikka();
                    break;
                case "3":
                    lisaaAsiakas();
                    break;
                case "4":
                    lisaaPaketti();
                    break;
                case "5":
                    lisaaTapahtuma();
                    break;
                case "6":
                    haePaketinTapahtumat();
                    break;
```

```
                case "7":
                    haeAsiakkaanPaketit();
                    break;
                case "8":
                    haePaikanTapahtumat();
                    break;
                case "9":
                    DAO tehokanta = new DAO("tehokkuustestikanta.db");
                    tehokanta.suoritaTehokkuustesti();
                    break;
                case "0":
                    System.exit(0);
                case "10":
                    kanta.tulostaTietokanta();
                    break;
                case "11":
                    kanta.poistaTaulut();
                    break;
                default:
                    System.out.println("Virheellinen komento, anna numero
                        ↪ 0-9");
                    break;
            }
        }
    }
}
```

Tietokantatoiminnot - DAO.java

```
package harjoitustyo;

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.format.DateTimeFormatter;
import java.time.LocalDateTime;
import java.util.Random;


// @author hiira

public class DAO {
    final String nimi;

    //konstruktori
    public DAO(String tietokannanNimi) {
        this.nimi=tietokannanNimi;
    }

    //metodi, joka luo taulut tietokantaan
    public void luoTaulut() {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            Statement s = db.createStatement();
            s.execute("CREATE TABLE IF NOT EXISTS Asiakkaat (id INTEGER
                ↪ PRIMARY KEY, nimi TEXT UNIQUE)");
            s.execute("CREATE TABLE IF NOT EXISTS Paikat (id INTEGER PRIMARY
                ↪  KEY, nimi TEXT UNIQUE)");
            s.execute("CREATE TABLE IF NOT EXISTS Paketit (id INTEGER
                ↪ PRIMARY KEY, seurantakoodi TEXT UNIQUE, asiakas_id
                ↪ INTEGER)");
            s.execute("CREATE TABLE IF NOT EXISTS Tapahtumat (id INTEGER
                ↪ PRIMARY KEY, paketti_id INTEGER, paikka_id INTEGER,
                ↪ kuvaus TEXT, aika TEXT)");

        } catch (SQLException e) {
            System.err.println("Tietokantataulujen luonnissa tapahtui virhe.
                ↪ ");
            getErrorMessages(e);
        }
    }
```

```
public void poistaTaulut() {
    try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
        ↪ this.nimi)) {
        Statement s = db.createStatement();
        s.execute("DROP TABLE IF EXISTS Asiakkaat");
        s.execute("DROP TABLE IF EXISTS Paikat");
        s.execute("DROP TABLE IF EXISTS Paketit");
        s.execute("DROP TABLE IF EXISTS Tapahtumat");
    } catch (SQLException e) {
        System.out.println("Taulujen poistossa tapahtui virhe.");
        getErrorMessages(e);
    }
}

public void lisaaPaikka(String paikka) {
    try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
        ↪ this.nimi)) {
         //aloitetaan transaktio, jotta vältytään
             samanaikaisuusogelmalta tarkistuksen ja lisäämisen välissä
        Statement st = db.createStatement();
        st.execute("BEGIN TRANSACTION");

        //varmitetaan ensin, että paikkaa ei ole jo olemassa
        PreparedStatement s = db.prepareStatement("SELECT * FROM Paikat
            ↪ WHERE nimi=?");
        s.setString(1, paikka);
        ResultSet resultset = s.executeQuery();

        if(resultset.next()) {
            System.out.println("Paikka on olemassa jo.");
            st.execute("COMMIT");
            return;
        }

        //lisätään uusi paikka tauluun Paikat
        s = db.prepareStatement("INSERT INTO Paikat (nimi) VALUES (?)");
        s.setString(1, paikka);
        s.executeUpdate();

        //lopetetaan transaktio
        st.execute("COMMIT");

        System.out.println("Paikka " + paikka + " lisätty.");

    } catch (SQLException e) {
        System.err.println("Paikan lisääminen ei onnistu.");
        getErrorMessages(e);
    }
```

```
    }

    public void lisaaAsiakas(String nimi) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            //aloitetaan transaktio, jotta tarkastuksessa vältytään
                samanaikaisuusongelmalta
            Statement st = db.createStatement();
            st.execute("BEGIN TRANSACTION");

            //varmistetaan ensin, ettei kyseisen nimistä asiakasta ole jo
                olemassa
            PreparedStatement pre = db.prepareStatement("SELECT * FROM
                ↪ Asiakkaat WHERE nimi=?");
            pre.setString(1, nimi);
            ResultSet rs = pre.executeQuery();

            if (rs.next()) {
                System.out.println("VIRHE: Asiakas on jo olemassa.");
                st.execute("COMMIT");
                return;
            }

            //lisätään uusi asiakas tauluun Asiakaat
            pre=db.prepareStatement("INSERT INTO Asiakkaat (nimi) VALUES (?)
                ↪ ");
            pre.setString(1, nimi);
            pre.executeUpdate();

            //lopetetaan transaktio
            st.execute("COMMIT");

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

        } catch (SQLException e) {
            System.out.println("Asiakasta ei saatu lisättyä.");
            getErrorMessages(e);
        }
    }

    public void lisaaPaketti(String koodi, String nimi) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            //aloitetaan transaktio
            Statement st = db.createStatement();
            st.execute("BEGIN TRANSACTION");

            //Tarkastetaan, että seurantakoodia ei ole jo olemassa
            PreparedStatement s = db.prepareStatement("SELECT * FROM Paketit
                ↪  WHERE seurantakoodi=?");
```

```
            s.setString(1,koodi);

            ResultSet rs = s.executeQuery();
            if (rs.next()) {
                System.out.println("VIRHE: annetulla seurantakoodilla on jo
                    ↪ paketti järjestelmässä.");
                st.execute("COMMIT");
                return;
            }
            //Tarkastetaan, että asiakas on jo tietokannassa
            if(haeAsiakkaanId(nimi)==-1) {
                System.out.println("VIRHE: Asiakasta ei löydy tietokannasta.
                    ↪ ");
                st.execute("COMMIT");
                return;
            }

            //lisätään uusi paketti tietokantaan
            s=db.prepareStatement("INSERT INTO Paketit (seurantakoodi,
                ↪ asiakas_id) VALUES (?,?)");
            s.setString(1,koodi);
            s.setInt(2,haeAsiakkaanId(nimi));
            s.executeUpdate();

            //lopetetaan transaktio
            st.execute("COMMIT");

            System.out.println("Paketti lisättiin seurantakoodilla "+koodi+"
                ↪ .");

        } catch (SQLException e) {
            System.out.println("Pakettia ei saada lisättyä.");
            getErrorMessages(e);
        }
    }

    public void lisaaTapahtuma(String koodi, String paikka, String kuvaus)
        ↪ {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {

            //tarkastetaan, että paketti löytyy tietokannasta
            if(haePaketinId(koodi)==-1) {
                System.out.println("VIRHE: Pakettia ei löydy tietokannasta."
                    ↪ );
                return;
            }

            //tarkastetaan, että paikka löytyy tietokannasta
```

```
            if(haePaikanId(paikka)==-1) {
                System.out.println("VIRHE: Paikkaa ei löydy tietokannasta.")
                    ↪ ;
                return;
            }

            //lisätään tapahtuma tietokantaan
            PreparedStatement s=db.prepareStatement("INSERT INTO Tapahtumat
                ↪ (paketti_id,paikka_id,kuvaus,aika) VALUES (?,?,?,?)");
            s.setInt(1, this.haePaketinId(koodi));
            s.setInt(2, this.haePaikanId(paikka));
            s.setString(3,kuvaus);
            s.setString(4,this.haeAika());

            s.executeUpdate();

            System.out.println("Uusi tapahtuma lisättiin.");

        } catch (SQLException e) {
            System.out.println("Tapahtuman lisääminen ei onnistunut.");
            getErrorMessages(e);
        }
    }

    public void haePaketinTapahtumat(String koodi) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            //tarkastetaan, että annettua koodia vastaa jokin paketti
                tietokannassa
            PreparedStatement s = db.prepareStatement("SELECT * FROM Paketit
                ↪  WHERE seurantakoodi=?");
            s.setString(1,koodi);
            ResultSet rs = s.executeQuery();
            if(!rs.next()) {
                System.out.println("Annettua seurantakoodia vastaavaa
                    ↪ pakettia ei löydy tietokannasta.");
                return;
            }

            //haetaan paketin tapahtumat
            s=db.prepareStatement("SELECT T.aika, PA.nimi, T.kuvaus "
                    + "FROM Tapahtumat T LEFT JOIN Paketit P ON T.paketti_id
                        ↪ =P.id "
                    + "LEFT JOIN Paikat PA ON T.paikka_id=PA.id "
                    + "WHERE P.seurantakoodi=?");
            s.setString(1, koodi);
            rs=s.executeQuery();
            while (rs.next()) {
                System.out.println(rs.getString("aika")+", "+rs.getString("
                    ↪ nimi")+", "+rs.getString("kuvaus"));
```

```
            }

        } catch (SQLException e) {
            System.out.println("Paketin tapahtumien haussa tapahtui virhe.")
                ↪ ;
            getErrorMessages(e);
        }
    }

    public void haeAsiakkaanPaketit(String asiakas) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {

            //Tarkastetaan, että annettu asiakas on olemassa
            if (haeAsiakkaanId(asiakas)==-1) {
                System.out.println("VIRHE: asiakasta ei löydy tietokannasta.
                    ↪ ");
                return;
            }
            //haetaan Asiakkaan paketit ja tapahtumien määrä
            PreparedStatement s = db.prepareStatement("SELECT P.
                ↪ seurantakoodi sk, COUNT(T.id) maara "
                    + "FROM Paketit P"
                    + " LEFT JOIN Tapahtumat T ON T.paketti_id=P.id "
                    + "WHERE P.asiakas_id=(SELECT A.id FROM Asiakkaat A
                        ↪ WHERE A.nimi=?)"
                    + " GROUP BY P.seurantakoodi");
            s.setString(1,asiakas);

            //tulostetaan paketit ja tapahtumien määrä
            ResultSet rs = s.executeQuery();
            while (rs.next()) {
                System.out.println(rs.getString("sk")+", "+rs.getInt("maara"
                    ↪ )+" tapahtumaa");
            }

        } catch (SQLException e) {
            System.out.println("Asiakkaan pakettien haussa tapahtui virhe.")
                ↪ ;
            getErrorMessages(e);
        }
    }

    public void haePaikanTapahtumat(String paikka, String paiva) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            //tarkastetaan onko paikka olemassa
            if (haePaikanId(paikka)==-1) {
```

```
            System.out.println("VIRHE: annettua paikkaa ei löydy
                ↪ tietokannasta.");
            return;
        }

        //haetaan Paikan tapahtumat annettuna päivänä
        PreparedStatement s = db.prepareStatement("SELECT COUNT(T.kuvaus
            ↪ ) m FROM Tapahtumat T "
                + "WHERE T.paikka_id=(SELECT id FROM Paikat WHERE nimi
                    ↪ =?) "
                + "AND DATE(T.aika)=DATE(?)");
        s.setString(1,paikka);
        s.setString(2,formatoiPaivamaara(paiva));

        //tulostetaan
        ResultSet rs = s.executeQuery();
        while (rs.next()) {
            System.out.println("Paikan "+paikka+" tapahtumien määrä
                ↪ annettuna päivämääränä: " + rs.getInt("m"));
        }

    } catch (SQLException e) {
        System.out.println("Virhe Paikan tapahtumien haussa tiettynä pä
            ↪ ivänä.");
        getErrorMessages(e);
    }

}

public void suoritaTehokkuustesti() {
    System.out.println("TEHOKKUUSTESTI");
    System.out.println("");
    System.out.println("Ilman indeksejä:");
    tehokkuustesti(false);
    System.out.println("");
    System.out.println("Indeksien kanssa:");
    tehokkuustesti(true);
}


public void tehokkuustesti(boolean onkoIndeksointi) {

    try (Connection db = DriverManager.getConnection("jdbc:sqlite:
        ↪ tehokkuustestikanta.db")) {

        //luodaan taulut tietokantaan, poistetaan jos jo olemassa
        poistaTaulut();
        luoTaulut();
```

```
//indekointi
if (onkoIndeksointi) {
    indeksoi();
}


//aloitetaan transaktio ja alkioiden lisääminen
Statement s = db.createStatement();
s.execute("BEGIN TRANSACTION");


//Paikat
long alku = System.nanoTime();
PreparedStatement p = db.prepareStatement("INSERT INTO Paikat (
    ↪ nimi) VALUES (?)");
for (int i=1; i<1000 ; i++) {
    p.setString(1,"P"+i);
    p.executeUpdate();
}
long loppu = System.nanoTime();

System.out.println("Paikkojen (1000 kpl) lisäämiseen kului: " +
    ↪ ((loppu-alku)/1e9) +" s");


//Asiakkaat
alku=System.nanoTime();
p=db.prepareStatement("INSERT INTO Asiakkaat (nimi) VALUES (?)")
    ↪ ;
for (int i=1 ; i<=1000 ; i++) {
    p.setString(1,"A"+i);
    p.executeUpdate();
}
loppu=System.nanoTime();

System.out.println("Asiakkaiden (1000 kpl) lisäämiseen kului: "
    ↪ +((loppu-alku)/1e9)+" s");


//Paketit
alku=System.nanoTime();
p=db.prepareStatement("INSERT INTO Paketit (seurantakoodi,
    ↪ asiakas_id) VALUES (?,(SELECT id FROM Asiakkaat WHERE
    ↪ nimi=?))");
for (int i=1 ; i<=1000 ; i++) {
    p.setString(1,"k"+i);
    p.setString(2, "A"+i);
    p.executeUpdate();
}
```

```
loppu=System.nanoTime();

System.out.println("Pakettien (1000 kpl) lisäämiseen kului: "+((
    ↪ loppu-alku)/1e9)+" s");


//Tapahtumat
alku=System.nanoTime();
Random random = new Random();
int luku;

p=db.prepareStatement("INSERT INTO Tapahtumat (paketti_id,
    ↪ paikka_id, kuvaus, aika) "
        + "VALUES ((SELECT id FROM Paketit WHERE seurantakoodi
            ↪ =?), "
        + "(SELECT id FROM Paikat WHERE nimi=?), ?, DATETIME('
            ↪ now','localtime'))");

for(int i=1 ; i<=1000000 ; i++) {
    luku=1+random.nextInt(1001);
    p.setString(1,"k"+luku);
    p.setString(2,"P"+luku);
    p.setString(3,"tapahtuma "+luku);
    p.executeUpdate();
}
loppu=System.nanoTime();
System.out.println("Miljoonan tapahtumaan lisäämiseen kului: "
    ↪ +((loppu-alku)/1e9)+" s");

//lopetetaan transaktio
s.execute("COMMIT");

//Tuhat kyselyä, haetaan jonkin asiakkaan pakettien määrä
alku=System.nanoTime();
for (int i=1 ; i<=1000 ; i++) {
    p=db.prepareStatement("SELECT COUNT(seurantakoodi) FROM
        ↪ Paketit WHERE asiakas_id=(SELECT id FROM Asiakkaat
        ↪ WHERE nimi=?)");
    p.setString(1, "A"+i);
    p.executeQuery();
}
loppu=System.nanoTime();
System.out.println("Tuhannen asiakkaan pakettien hakemiseen
    ↪ kului: "+((loppu-alku)/1e9)+" s");


//Tuhannen paketin tapahtumien määrän hakemiseen kuluva aika:
alku=System.nanoTime();
```

```
            p=db.prepareStatement("SELECT COUNT(id) maara FROM Tapahtumat
                ↪ WHERE paketti_id=(SELECT id FROM Paketit WHERE
                ↪ seurantakoodi=?)");

            for (int i=1 ; i<=1000 ; i++) {
                p.setString(1,"k"+i);
                p.executeQuery();
            }
            loppu=System.nanoTime();
            System.out.println("Tuhannen paketin tapahtumien määrän
                ↪ hakemiseen kului: "+((loppu-alku)/1e9)+" s");



        } catch (SQLException e) {
            getErrorMessages(e);
        }

    }

    public void indeksoi() {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            Statement s = db.createStatement();
            s.execute("CREATE INDEX idx_paikka ON Paikat (nimi);");
            s.execute("CREATE INDEX idx_asiakas ON Asiakkaat (nimi);");
            s.execute("CREATE INDEX idx_paketti ON Paketit (seurantakoodi);"
                ↪ );
            s.execute("CREATE INDEX idx_paketit_asiakas_id ON Paketit (
                ↪ asiakas_id);");
            s.execute("CREATE INDEX idx_tapahtumat_paketti_id ON Tapahtumat
                ↪ (paketti_id);");
            s.execute("CREATE INDEX idx_tapahtumat_paikka_id ON Tapahtumat (
                ↪ paikka_id);");
        } catch (SQLException e) {
            System.out.println("Virhe indeksien luomisessa.");
            getErrorMessages(e);
        }
    }

    //APUMETODIT:

    public int haeAsiakkaanId(String nimi) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            PreparedStatement s = db.prepareStatement("SELECT id FROM
                ↪ Asiakkaat WHERE nimi=?");
            s.setString(1,nimi);
```

```
            ResultSet rs = s.executeQuery();
            int id=rs.getInt("id");
            return id;
        } catch (SQLException e) {
            return -1;
        }
    }

    public int haePaketinId(String koodi) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            PreparedStatement s = db.prepareStatement("SELECT id FROM
                ↪ Paketit WHERE seurantakoodi=?");
            s.setString(1,koodi);
            ResultSet rs = s.executeQuery();
            int id=rs.getInt("id");
            return id;
        } catch (SQLException e) {
            return -1;
        }
    }

    public int haePaikanId(String nimi) {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
            PreparedStatement s = db.prepareStatement("SELECT id FROM Paikat
                ↪  WHERE nimi=?");
            s.setString(1,nimi);
            ResultSet rs = s.executeQuery();
            int id=rs.getInt("id");
            return id;
        } catch (SQLException e) {
            return -1;
        }
    }

    public String haeAika() {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:
            ↪ mm:ss");
        LocalDateTime now = LocalDateTime.now();
        return dtf.format(now);
    }

    //metodi formatoiPaiva muuttaa muotoa d.m.yyyy olevan päivän muotoon
        yyyy-MM-dd
    public String formatoiPaivamaara(String paivamaara) {

        String[] palat = paivamaara.split("\\.");

        if (palat.length != 3) {
```

```
            System.out.println("Tarkista päivämäärä.");
        }

        palat[0]=lisaaNolla(palat[0]);
        palat[1]=lisaaNolla(palat[1]);

        String formatoituPaiva = palat[2]+"-"+palat[1]+"-"+palat[0];

        return formatoituPaiva;
    }

    //lisätään nollat pienten päivien ja kuukausien eteen mikäli käyttäjä
        ei niitä syöttänyt
    public String lisaaNolla(String aikapala) {
        int x = Integer.valueOf(aikapala);  //vaikka käyttäjä olisi
            syöttänyt 01, muuttuu se tässä vaiheessa muotoon 1
        String uusiAikapala = null;
        if (x<10) {
            uusiAikapala="0"+x;
            return uusiAikapala;
        }
        return aikapala;
    }


    public void tulostaTietokanta() {
        try (Connection db = DriverManager.getConnection("jdbc:sqlite:"+
            ↪ this.nimi)) {
        Statement s = db.createStatement();
        //Asiakkaat
        ResultSet r = s.executeQuery("SELECT * FROM Asiakkaat");
        System.out.println("ASIAKKAAT");
        while (r.next()) {
            System.out.println(r.getInt("id")+" "+r.getString("nimi"));
        }
        System.out.println("");
        //Paikat
        r=s.executeQuery("SELECT * FROM Paikat");
        System.out.println("PAIKAT");
        while (r.next()) {
            System.out.println(r.getInt("id")+" "+r.getString("nimi"));
        }
        System.out.println("");
        //Paketit
        r=s.executeQuery("SELECT * FROM Paketit");
        System.out.println("PAKETIT");
        while (r.next()) {
            System.out.println(r.getInt("id")+" "+r.getString("seurantakoodi
                ↪ ")+" "+r.getString("asiakas_id"));
        }
```

```
        System.out.println("");
        //Tapahtumat
        r=s.executeQuery("SELECT * FROM Tapahtumat");
        System.out.println("TAPAHTUMAT");
        while (r.next()) {
            System.out.println(r.getInt("id")+" "+r.getString("paketti_id")+
                ↪ " "+r.getString("paikka_id")+" "+r.getString("kuvaus")+"
                ↪ "+r.getString("aika"));
        }
        System.out.println("");

        } catch (SQLException e) {
            System.out.println("Ongelma taulujen tulostuksessa.");
            getErrorMessages(e);
        }
    }

    //erillinen metodi virheviesteille
    public void getErrorMessages(SQLException e) {
        do {
            System.err.println("Viesti: " + e.getMessage());
            System.err.println("Virhekoodi: " + e.getErrorCode());
            System.err.println("SQL-tilakoodi: " + e.getSQLState());
        } while (e.getNextException() != null);
    }
}
```