# Tietokantojen perusteet - Harjoitustyö

Tuomo Artama, 014934065, tuomoart

27.2.2020

# 1 Sovelluksen ominaisuudet

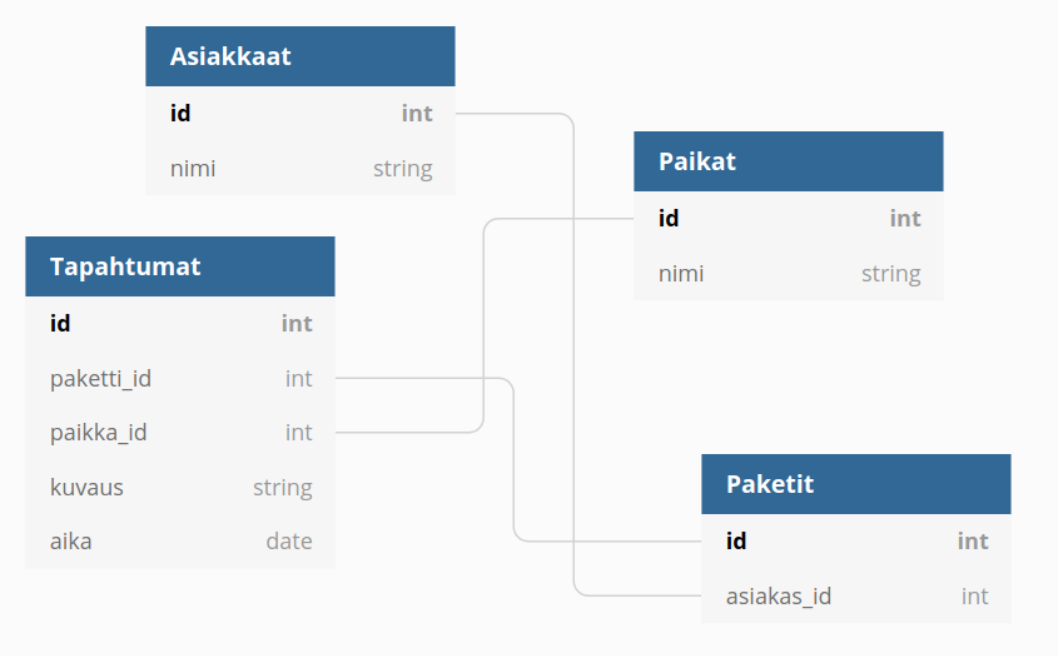
Harjoitustyön aiheena oli toteuttaa SQLite-tietokantaa käyttävä paketinseurantasovellus. Harjoitustyösovellukseen on toteutettu kaikki tehtävänannon toiminnot. Sovelluksen avulla voi luoda pakettien seurannan mahdollistavan tietokannan. Tietokantaan voi lisätä paikkoja, asiakkaita ja paketteja. Jokaisella paketilla on jokin asiakas, jolle paketti kuuluu. Tietokantaan voi myös lisätä tapahtumia, joihin jokaiseen liittyy aina paketti, paikka, kuvaus tapahtumasta sekä aika, jolloin tapahtuma on lisätty. Paikoilla ja asiakkailla on nimet ja paketilla on seurantakoodi. Kaikkien näistä tulee olla uniikkeja, eli tietokantaan ei voi lisätä kahta samannimistä asiakasta tai kahta pakettia samalla seurantakoodilla. Tietokannasta on mahdollista hakea jonkin paketin tapahtumat seurantakoodin perusteella, jonkin asiakkaan paketit asiakkaan nimen perusteella tai jonkin paikan tapahtumat tiettynä päivänä.

Jos käyttäjä antaa virheellisen syötteen, siitä kerrotaan virheviestillä. Hakutulokset tulostetaan helppolukuisena taulukkona listan sijaan. Lisäksi sovelluksella voi suorittaa tietokannalle tehokkuustestin, jonka avulla voi verrata indeksoidun ja indeksoimattoman tietokannan eroa. Sovellukseen on myös sisäänrakennettu lisätoimintoja koko tietokannan sisällön näyttämiseen ja esimerkiksi tietokannan poistamiseen, mutta ne eivät ole saatavilla tavalliselle käyttäjälle. Lisätoiminnot saa käyttöön käynnistämällä ohjelman millä tahansa ylimääräisellä parametrilla. Nämä lisätoiminnot eivät ole toteutukseltaan kovinkaan viimeisteltyjä, sillä niitä ei vaadittu tehtävänannossa.

Sovellus on rakennettu niin, että sen pääohjelmana ja käyttöliittymänä toimii harjoitus.py. Tietokantatoiminnallisuus on omana luokkanaan, ja sen apuna on virheenkäsittelyapuriluokka. Virheenkäsittely toimii sovelluksessa siten, että poikkeuksen tullessa poikkeus välitetään virheenkäsittelijälle. Käsittelijä tulkitsee, millaisesta virheestä on kyse ja palauttaa sen mukaisen koodin tietokantaoliolle. Tietokantaolio palauttaa tämän koodin eteenpäin käyttöliittymälle (pääohjelmalle), joka tulkitsee koodin ja tulostaa sen mukaisen virheviestin. Silloin, kun virheenkäsittelijä ei tunnista virhettä, se tulostaa itse olennaiset tiedot virheestä. Jos sovellukseen haluttaisiin myös muunlainen käyttölittymä, tulisi erityisten virheiden käsittely toteuttaa muutoin, esimerkiksi palauttamalla virhetiedot käyttöliittymälle, joka sitten tulostaisi ne käyttäjän nähtäväksi.

# 2 Tietokanta

## 2.1 Tietokantakaavio



### 2.2 SQL-skeema

```
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, asiakas_id INTEGER,
seurantakoodi TEXT UNIQUE);

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

CREATE TRIGGER asiakas_ei_loydy_lisatessa_paketti BEFORE INSERT ON
Paketit BEGIN SELECT CASE WHEN NEW.asiakas_id IS NULL THEN
RAISE(ABORT,'Asiakasta ei loydy!') END; END;

CREATE TRIGGER paketti_ei_loydy_lisatessa_tapahtuma BEFORE INSERT ON
Tapahtumat BEGIN SELECT CASE WHEN NEW.paikka_id IS NULL THEN
RAISE(ABORT, 'Paikkaa ei loydy!') END; END;

CREATE TRIGGER paikka_ei_loydy_lisatessa_tapahtuma BEFORE INSERT ON
Tapahtumat BEGIN SELECT CASE WHEN NEW.paketti_id IS NULL THEN
RAISE(ABORT, 'Pakettia ei loydy!') END; END;

CREATE INDEX idx_asiakkaat ON Asiakkaat (nimi);

CREATE INDEX idx_Paikat ON Paikat (nimi);

CREATE INDEX idx_paketit ON Paketit (seurantakoodi);

CREATE INDEX idx_paketit_asiakas_id ON Paketit (asiakas_id);

CREATE INDEX idx_tapahtumat_paketti_id ON Tapahtumat (paketti_id);

CREATE INDEX idx_tapahtumat_paikka_id ON Tapahtumat (paikka_id);
```

## 3 Tehokkuustesti

Sovellukseen on toteutettu tehtävänannon mukainen tehokkuustesti. Asiakkaat, paikat ja paketit on lisätty indekseillä $1 - 1000$ ja tapahtumat on lisätty satunnaisesti valituille paketeille. Tehokkuustestin kyselyt tehdään samaten satunnaisille asiakkaille ja paketeille. Alla on ohjelman tulostamat kestot:

```
Ilman indeksointia:

Paikkojen lisaamisessa kesti: 0:00:00.001494
Asiakkaiden lisaamisessa kesti: 0:00:00.001325
Pakettien lisaamisessa kesti: 0:00:00.002280
Tapahtumien lisaamisessa kesti: 0:00:04.216910
Asiakkaiden pakettien maaran hakemisessa meni: 0:00:00.084750
Pakettien tapahtumien maaran hakemisessa meni: 0:01:02.476351


Indeksoituna:

Paikkojen lisaamisessa kesti: 0:00:00.002003
Asiakkaiden lisaamisessa kesti: 0:00:00.001952
Pakettien lisaamisessa kesti: 0:00:00.005958
Tapahtumien lisaamisessa kesti: 0:00:14.779937
Asiakkaiden pakettien maaran hakemisessa meni: 0:00:00.037650
Pakettien tapahtumien maaran hakemisessa meni: 0:00:00.052183
```

Erillisillä suoirtuskerroilla tulokset olivat hyvin lähellä toisiaan. Kuten nähdään, indeksoimattomassa tietokannassa lisääminen onnistuu hieman nopeammin, mutta pakettien tapahtumien määrän hakeminen on merkittävästi hitaampaa. Indeksoidussa tietokannassa lisätessä indeksit on luotava, mikä monimutkaistaa ja hidastaa lisäysprosessia hieman, mutta niistä saatava nopeusetu hakiessa on selkeästi suurempi kuin lisäämisen hidastuminen. Käyttämässäni tietokannassa on hieman ylimääräisiäkin indeksejä kun otetaan huomioon tehtävänanto, mutta päätin jättää ne paikalleen, sillä tämäntyylisessä oikeassa tietokannassa niistä olisi hyötyä, sillä toimintoja ja erilaisia kyselyitä olisi enemmän.

Havaitsin tehokkuustestiä tehdessä myös, että yllättävän suuri merkitys paketin tapahtumien määrän hakemiseen on sillä, yhdistääkö taulut Tapahtumat ja Paketit vai käyttääkö paketin id:n hakemiseen alikyselyä. Alikysely oli nopein, ja molempien taulujen käyttö WHERE-ehdolla oli lähes yhtä nopea. LEFT JOIN -yhdistäminen sen sijaan oli moninkertaisesti hitaampaa. Kyseessä saattaa kuitenkin vain olla virhe, tai jokin käyttämäni ratkaisun erityisominaisuus, enkä tutkinut tätä asiaa sen tarkemmin.

# 4 Tiedon yksilöllisyyden varmistaminen

Toteuttamassani sovelluksessa tiedon yksilöllisyys ja sen olemassaolo on varmistettu lähes kokonaan tietokannan sisäisillä ominaisuuksilla. Asiakkaiden, paikkojen ja pakettien yksilöllisyys on varmistettu määrittämällä niitä vastavissa tauluissa sopiville sarakkeille ehto UNIQUE. Lisättäessä esimerkiksi pakettia, on halutun asiakkaan olemassaolo varmistettu ehdolla, ettei paketteihin lisätessä asiakas_id:n arvo saa olla NULL. Tällöin ohjelma ei lisää pakettia olemattomalle asiakkaalle vaan ilmoittaa käyttäjälle asiakkaan puuttumisesta. Ehto itsessään on toteutettu SQLite-laukaisimen (trigger) avulla, vaikkakin jälkeenpäin havaitsin, että helpompiakin vaihtoehtoja olisi saattanut löytyä.

Vastaavat laukaisimet vahtivat myös tapahtumien lisäämistä.

Kaikki tietokannan toiminnot tehdään yksittäisissä transaktioissa ja useimmiten vain yhdessä komennossa. Tällä varmistutaan siitä, ettei rinnakkainen tietokannan käyttäminen ole ongelma. Tiedon yksilöllisyyteen liittyvää varmistamista ei tehdä erikseen, vaan tietokanta huolehtii siitä itse, minkä seurauksena tieto säilyy yksilöllisenä vaikka useat käyttäjät yrittäisivät lisätä samaa arvoa samaan tauluun samaan aikaan. Tietokantasovellukseni ei ole kuitenkaan tehty rinnakkaiseen käyttöön, sillä jos sovelluksen tarvitsema taulu on varattuna, se ei osaa odottaa ja yrittää uudestaan sen vapautuessa vaan suoritus keskeytyy. Rinnakkainen käyttö onnistuu sovelluksessani niin kauan, kuin käyttö on vuorottaista. Aidosti samanaikainen käyttäminen johtaisi toisen käyttäjän toiminnon epäonnistumiseen.

# 5 Lähdekoodi

Lähdekoodi on saatavilla myös GitHub:ssa: `https://github.com/tuomoart/tikape-harjoitustyo`

## 5.1 Pääohjelma

```
import sqlite3
from prettytable import PrettyTable
from datetime import datetime
import tietokanta
import sys
import os

def timeFormatHelper(aika):
    aika=int(aika)
    if (aika<10):
        return "0"+str(aika)
    return str(aika)

def formatTime(aika):
    try:
        palat=aika.split(".")
        palat[1]=timeFormatHelper(palat[1])
        palat[0]=timeFormatHelper(palat[0])
        uusiAika=""
        i=len(palat)-1
        while i>=0:
            uusiAika=uusiAika+palat[i]
            if i!=0:
                uusiAika=uusiAika+"-"
            i=i-1
        return uusiAika
    except:
        return -1
```

```
def tulostaOhjeet():
    print("Valitse toiminto (1-9):")
    print("0: Lopeta")
    print("1: Luo tietokanta")
    print("2: Lisää paikka")
    print("3: Lisää asiakas")
    print("4: Lisää paketti")
    print("5: Lisää tapahtuma")
    print("6: Hae paketin tapahtumat")
    print("7: Hae asiakkaan paketit")
    print("8: Hae paikan tapahtumien määrä")
    print("9: Suorita tehokkuustesti")
    if unlocked:
        print("10: Poista tietokanta")
        print("12: Tulosta taulu 'Paikat'")
        print("13: Tulosta taulu 'Asiakkaat'")
        print("14: Tulosta taulu 'Paketit'")
        print("15: Tulosta taulu 'Tapahtumat'")

def tulostaTaulu(taulu, otsikot):
    t=PrettyTable(otsikot)
    i=0
    while i<len(taulu):
        t.add_row(taulu[i])
        i=i+1
    print(t)

def main():
    tk=tietokanta.Tietokanta("database.db")
    clear = lambda: os.system('clear')
    clear()
    while True:
        tulostaOhjeet()
        inp=input("syätä komento: ")
        print()
        if inp=="":
            clear()
            continue
        elif inp=="0":
            print("heihei")
            break
        elif inp=="1":
            tk.createdb(True)
            print("Tietokanta luotu")
        elif inp=="2":
            nimi=input("Paikan nimi: ")
            status=tk.lisaaPaikka(nimi)
            if status==0:
```

```
            print("Paikka lisätty!")
        elif status==7:
            print("Paikka on jo lisätty!")
    elif inp=="3":
        nimi=input("Asiakkaan nimi: ")
        status=tk.lisaaAsiakas(nimi)
        if status==0:
            print("Asiakas lisätty!")
        elif status==1:
            print("Asiakas on jo lisätty!")
    elif inp=="4":
        nimi=input("Asiakkaan nimi: ")
        koodi=input("Seurantakoodi: ")
        status=tk.lisaaPaketti(nimi,koodi)
        if status==0:
            print("Paketti lisätty!")
        elif status==3:
            print("Asiakasta ei läydy!")
        elif status==4:
            print("Paketti on jo olemassa!")
    elif inp=="5":
        koodi = input("Anna seurantakoodi: ")
        paikka=input("Anna paikka: ")
        kuvaus=input("Anna kuvaus: ")
        aika=datetime.now()
        status=tk.lisaaTapahtuma(koodi, paikka, kuvaus, aika)
        if status==0:
            print("Tapahtuma lisätty!")
        elif status==5:
            print("Pakettia ei läydy!")
        elif status==6:
            print("Paikkaa ei läydy!")
    elif inp=="6":
        koodi=input("Anna seurantakoodi: ")
        status=tk.haeTapahtumatKoodilla(koodi)
        if status==665:
            print("Pakettia ei läytynyt!")
        else:
            tulostaTaulu(status[0],status[1])
    elif inp=="7":
        nimi=input("Asiakkaan nimi: ")
        status=tk.haeAsiakkaanPaketit(nimi)
        if status==664:
            print("Asiakasta ei läytynyt!")
        else:
            tulostaTaulu(status[0],status[1])
    elif inp=="8":
        paikka=input("Anna paikka: ")
        while True:
```

```
                paiva=input("Anna paiva (pp.kk.vvvv): ")
                paiva=formatTime(paiva)
                if paiva==-1:
                    print("Virheellinen päivämäärän muoto!")
                    continue
                break
            status=tk.haePaikanTapahtumat(paikka,paiva)
            if status==666:
                print("Paikkaa ei läytynyt!")
            elif len(status)>1:
                tulostaTaulu(status[0],status[1])
        elif inp=="9":
            tkt=tietokanta.Tietokanta("testbase.db")
            tkt.tehokkuustesti()
        elif inp=="10" and unlocked:
            tk.deletedb()
            print("Tietokanta poistettu")
        elif inp=="12" and unlocked:
            print("Paikat:")
            tk.tulostaPaikat()
        elif inp=="13" and unlocked:
            print("Asiakkaat:")
            tk.tulostaAsiakkaat()
        elif inp=="14" and unlocked:
            print("Paketit:")
            tk.tulostaPaketit()
        elif inp=="15" and unlocked:
            print("Tapahtumat:")
            tk.tulostaTapahtumat()
        else:
            print("Virheellinen komento!")
        input("Paina enter jatkaaksesi")
        clear()


if len(sys.argv)>1:
    unlocked=True
else:
    unlocked=False

main()
```

## 5.2 Tietokanta

```
import DbExceptionHandler
import sqlite3
from prettytable import PrettyTable
```

```
from datetime import datetime
import random
import sys

class Tietokanta:
    def __init__(self, tietokanta):
        self.db=sqlite3.connect(tietokanta)
        self.db.isolation_level = None

        self.c=self.db.cursor()

        self.handler = DbExceptionHandler.DbExceptionHandler()

    def createdb(self, indeksoitu):
        queries = ["CREATE TABLE IF NOT EXISTS Asiakkaat (id INTEGER
            ↪ PRIMARY KEY, nimi TEXT UNIQUE);",
                   "CREATE TABLE IF NOT EXISTS Paikat (id INTEGER PRIMARY
                       ↪ KEY, nimi TEXT UNIQUE);",
                   "CREATE TABLE IF NOT EXISTS Paketit (id INTEGER PRIMARY
                       ↪ KEY, asiakas_id INTEGER, seurantakoodi TEXT UNIQUE
                       ↪ );",
                   "CREATE TABLE IF NOT EXISTS Tapahtumat (id INTEGER
                       ↪ PRIMARY KEY, paketti_id INTEGER, paikka_id INTEGER
                       ↪ , kuvaus TEXT, aika DATE);",
                   "CREATE TRIGGER IF NOT EXISTS
                       ↪ asiakas_ei_loydy_lisatessa_paketti BEFORE INSERT
                       ↪ ON Paketit BEGIN SELECT CASE WHEN NEW.asiakas_id
                       ↪ IS NULL THEN RAISE(ABORT,'Asiakasta ei loydy!')
                       ↪ END; END;",
                   "CREATE TRIGGER IF NOT EXISTS
                       ↪ paketti_ei_loydy_lisatessa_tapahtuma BEFORE INSERT
                       ↪  ON Tapahtumat BEGIN SELECT CASE WHEN NEW.
                       ↪ paikka_id IS NULL THEN RAISE(ABORT, 'Paikkaa ei
                       ↪ loydy!') END; END;",
                   "CREATE TRIGGER IF NOT EXISTS
                       ↪ paikka_ei_loydy_lisatessa_tapahtuma BEFORE INSERT
                       ↪ ON Tapahtumat BEGIN SELECT CASE WHEN NEW.
                       ↪ paketti_id IS NULL THEN RAISE(ABORT, 'Pakettia ei
                       ↪ loydy!') END; END;"]

        for query in queries:
            try:
                self.c.execute(query)
            except:
                self.handler.handle(sys.exc_info())

        if indeksoitu:
            self.indeksoi()
```

```
    def deletedb(self):
        self.c.execute("DROP TABLE IF EXISTS Asiakkaat;")
        self.c.execute("DROP TABLE IF EXISTS Paikat;")
        self.c.execute("DROP TABLE IF EXISTS Paketit;")
        self.c.execute("DROP TABLE IF EXISTS Tapahtumat;")

    def indeksoi(self):
        try:
            self.c.execute("CREATE INDEX idx_asiakkaat ON Asiakkaat (nimi);"
                ↪ )
            self.c.execute("CREATE INDEX idx_Paikat ON Paikat (nimi);")
            self.c.execute("CREATE INDEX idx_paketit ON Paketit (
                ↪ seurantakoodi);")
            self.c.execute("CREATE INDEX idx_paketit_asiakas_id ON Paketit (
                ↪ asiakas_id)")
            self.c.execute("CREATE INDEX idx_tapahtumat_paketti_id ON
                ↪ Tapahtumat (paketti_id)")
            self.c.execute("CREATE INDEX idx_tapahtumat_paikka_id ON
                ↪ Tapahtumat (paikka_id)")
        except:
            self.handler.handle(sys.exc_info())

    def lisaaPaikka(self, paikka):
        try:
            self.c.execute("INSERT INTO Paikat (nimi) VALUES (?);",[paikka])
        except:
            return self.handler.handle(sys.exc_info())
        return 0

    def lisaaAsiakas(self, nimi):
        try:
            self.c.execute("INSERT INTO Asiakkaat (nimi) VALUES (?);",[nimi
                ↪ ])
        except:
            return self.handler.handle(sys.exc_info())
        return 0

    def lisaaPaketti(self, asiakas, koodi):
        try:
            self.c.execute("INSERT INTO Paketit (asiakas_id, seurantakoodi)
                ↪ VALUES ((SELECT id FROM Asiakkaat WHERE nimi=?),?);", [
                ↪ asiakas,koodi])
        except:
            return self.handler.handle(sys.exc_info())
        return 0

    def lisaaTapahtuma(self, koodi, paikka, kuvaus, aika):
        try:
```

```
            self.c.execute("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'))",[koodi, paikka, kuvaus])
        except:
            return self.handler.handle(sys.exc_info())
        return 0

    def getAsiakas(self, nimi):
        self.c.execute("SELECT id FROM Asiakkaat WHERE nimi = ?",[nimi])
        tulos = self.c.fetchone()
        if tulos==None:
            return -1
        return int(tulos[0])

    def getPaketti(self, koodi):
        self.c.execute("SELECT id FROM Paketit WHERE seurantakoodi = ?",
        [koodi])
        tulos = self.c.fetchone()
        if tulos==None:
            return -1
        return int(tulos[0])

    def getPaikka(self, nimi):
        self.c.execute("SELECT id FROM Paikat WHERE nimi = ?",[nimi])
        tulos=self.c.fetchone()
        if (tulos==None):
            return -1
        return int(tulos[0])

    def haeTapahtumatKoodilla(self, koodi):
        if self.getPaketti(koodi)==-1:
            return 665
        try:
            self.c.execute("SELECT aika, Paikat.nimi, kuvaus FROM Paikat
                ↪ LEFT JOIN Tapahtumat ON Paikat.id=paikka_id WHERE
                ↪ paketti_id=(SELECT id FROM Paketit WHERE seurantakoodi=?)
                ↪ ;",[koodi])
        except:
            return self.handler.handle(sys.exc_info())
        return self.c.fetchall(),["Aika","Paikka","Kuvaus"]

    def haeAsiakkaanPaketit(self, asiakas):
        if self.getAsiakas(asiakas)==-1:
            return 664
        try:
            self.c.execute("SELECT Paketit.seurantakoodi, COUNT(TapP.id)
                ↪ FROM Paketit LEFT JOIN Tapahtumat TapP ON Paketit.id=TapP
                ↪ .paketti_id WHERE Paketit.asiakas_id=(SELECT id FROM
```

```
            ↪ Asiakkaat WHERE nimi=?) GROUP BY Paketit.seurantakoodi ;"
            ↪ ,[asiakas])
        except:
            return self.handler.handle(sys.exc_info())
        return self.c.fetchall(),["paketti", "tapahtumia"]

    def haePaikanTapahtumat(self, paikka, paiva):
        if(self.getPaikka(paikka)==-1):
            return 666
        try:
            self.c.execute("SELECT COUNT(id) FROM Tapahtumat WHERE paikka_id
            ↪ =(SELECT id FROM Paikat WHERE nimi=?) AND DATE(aika)=DATE
            ↪ (?)",[paikka,paiva])
        except:
            return self.handler.handle(sys.exc_info())
        return self.c.fetchall(),["tapahtumia"]


    def tulostaPaikat(self):
        self.c.execute("SELECT * FROM Paikat;")
        self.tulostaTaulu(self.c.fetchall(),[])

    def tulostaAsiakkaat(self):
        self.c.execute("SELECT * FROM Asiakkaat;")
        self.tulostaTaulu(self.c.fetchall(),[])

    def tulostaPaketit(self):
        self.c.execute("SELECT * FROM Paketit;")
        self.tulostaTaulu(self.c.fetchall(),[])

    def tulostaTapahtumat(self):
        self.c.execute("SELECT * FROM Tapahtumat;")
        self.tulostaTaulu(self.c.fetchall(),["id","paketti_id", "paikka_id"
            ↪ ,
        "kuvaus", "aika"])

    def tulostaTaulu(self, taulu, otsikot):
        t=PrettyTable(otsikot)
        i=0
        while i<len(taulu):
            t.add_row(taulu[i])
            i=i+1
        print(t)

    def tehokkuustesti(self):
        print()
        print("Ilman indeksointia:")
        print()
        self.tehokkuustestiSuoritus(False)
```

```
        print()
        print()
        print("Indeksoituna:")
        print()
        self.tehokkuustestiSuoritus(True)

    def tehokkuustestiSuoritus(self, indeksoitu):
        self.deletedb()

        self.createdb(indeksoitu)

        alku=datetime.now()

        paikat = []
        asiakkaat = []
        paketit = []

        i=1
        while i<=1000:
            paikat.append(["P"+str(i)])
            asiakkaat.append(["A"+str(i)])
            paketit.append([i,i])

            i+=1

        i=0

        tapahtumatq="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'))"
        tapahtumatParams=[]
        while i<1000000:
            tapahtumatParams.append([str(random.randrange(1,1001,1)),
            "P"+str(random.randrange(1,1001,1)),"tapahtuma "+str(i)])
            i+=1
        self.c.execute("BEGIN TRANSACTION;")
        alku=datetime.now()
        self.c.executemany("INSERT INTO Paikat (nimi) VALUES (?)", paikat)
        print("Paikkojen lisäämisessä kesti: " + str(datetime.now()-alku))
        alku=datetime.now()
        self.c.executemany("INSERT INTO Asiakkaat (nimi) VALUES (?)",
            ↪ asiakkaat)
        print("Asiakkaiden lisäämisessä kesti: " + str(datetime.now()-alku)
            ↪ )
        alku=datetime.now()
        self.c.executemany("INSERT INTO Paketit (asiakas_id, seurantakoodi)
            ↪  VALUES (?,?)", paketit)
```

```
        print("Pakettien lisäämisessä kesti: " + str(datetime.now()-alku))
        alku=datetime.now()
        self.c.executemany("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'))", tapahtumatParams)
        print("Tapahtumien lisäämisessä kesti: " + str(datetime.now()-alku)
            ↪ )
        self.c.execute("COMMIT;")

        i=0

        alku=datetime.now()
        while i<1000:
            self.c.execute("SELECT COUNT(id) FROM Paketit WHERE asiakas_id =
                ↪  (SELECT id FROM Asiakkaat WHERE nimi=?)",asiakkaat[i])
            i+=1
        print("Asiakkaiden pakettien määrän hakemisessa meni: "
        +str(datetime.now()-alku))

        i=0
        alku = datetime.now()
        while i<2000:
            self.c.execute("SELECT COUNT(paketti_id) FROM Tapahtumat WHERE
                ↪ paketti_id = (SELECT id FROM Paketit WHERE seurantakoodi
                ↪ = ?)", [str(i)])
            i+=2

        print("Pakettien tapahtumien määrän hakemisessa meni: "
        +str(datetime.now()-alku))
```

## 5.3 Virheenkäsittelyapuri

```
class DbExceptionHandler:


    def handle(self, exception):
        if str(exception[1])=="UNIQUE constraint failed: Asiakkaat.nimi":
            return 1
        elif str(exception[1])=="UNIQUE constraint failed: Paketit.nimi":
            return 2
        elif str(exception[1])=="Asiakasta ei loydy!":
            return 3
        elif str(exception[1])=="UNIQUE constraint failed: Paketit.
            ↪ seurantakoodi":
```

```
            return 4
        elif str(exception[1])=="Pakettia ei loydy!":
            return 5
        elif str(exception[1])=="Paikkaa ei loydy!":
            return 6
        elif str(exception[1])=="UNIQUE constraint failed: Paikat.nimi":
            return 7
        print("Tuntematon virhe:")
        print(exception)
```