25.6.2020

Tietokantojen perusteet kesä 2020 – Harjoitustyö.

Tehtävä 1

Sovelluskoodin kirjoittamiseen on kulunut aikaa yhteensä 2h 9min. ajatustyöta en ole laskenut mukaan.

Vaikka koodi näyttää päällisin puolin copy-pastelta, en kuitenkaan tyytynyt copy-pastetukseen, vaan kirjoitin koodin manuaalisesti, koska se on hauskempaa ja samalla jää koodaus paremmin mieleen.

Koodi on kirjoitettu Vim-editorilla, jossa SpaceVim-IDE lisäosa Linux Ubuntu Studio käyttöjärjestelmässä.

Mitä toimintoja on toteutettu?

Ensimmäisessä tehtävässä on toteutettu kaikki esimerkissä vaaditut toiminnot.

Sovelluksen lähdekoodi:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;

/*
 *
 * @author  Erkki Pokkinen
 * @version 1
 * @since   2020-05-08
 * @valmis  2020-06-05
 * @Teht1   01h 11min 30sek
 * @Teht2       30min 52sek
 * @Teht3       15min 29sek
 * @Teht4       11min  9sek
 * ------------------------
 *           2h  9min
 *
 */

public class Harjoitustyo {
    public static void main(String[] args) throws SQLException {
```

```
Scanner lukija = new Scanner(System.in);

Connection yhteys = DriverManager.getConnection("jdbc:sqlite:kurssit.db");

while(true) {

    System.out.print("Valitse toiminto: ");
    int toiminto = Integer.valueOf(lukija.nextLine());

    // 5 Lopetus
    if(toiminto == 5) {
        break;
    // 1 Kysely: Annetun vuoden pisteet.
    }if(toiminto == 1) {

        System.out.print("Anna vuosi: ");
        //int vuosi = Integer.valueOf(lukija.nextLine());
        String vuosi = lukija.nextLine();

        String sqlLause =
        "SELECT SUM(arvosana) AS Pisteet \n"+
        "FROM Suoritukset \n"+
        "WHERE strftime('%Y', paivays) = ? \n";

        PreparedStatement p = yhteys.prepareStatement(sqlLause);
        p.setString(1, vuosi);

        try {
            ResultSet r = p.executeQuery();
            if (r.next()) {
                if(r.getInt("Pisteet") >0) {
                    System.out.println("Opintopisteiden määrä: "
                        +r.getInt("Pisteet"));
                }else{
                    System.out.println("Vuosilukua ei löytynyt");
                }
            }
            r.close();
            p.close();
        }catch(SQLException e) {
            System.out.println("Taulun haku epäonnistui: "+e.getMessage());
        }
    // 2 Hae opiskelijan kaikki kurssit
    }if(toiminto == 2) {
       System.out.print("Anna opiskelijan nimi: ");
       String opiskelija = lukija.nextLine();

       String sqlLause2 =
       "SELECT Kurssit.nimi, Kurssit.laajuus, \n"+
       "Suoritukset.paivays, Suoritukset.arvosana \n"+
       "FROM Kurssit, Suoritukset, Opiskelijat \n"+
```

```
   "WHERE Suoritukset.opiskelija_id = Opiskelijat.id \n"+
   "AND Suoritukset.kurssi_id = Kurssit.id \n"+
   "AND Opiskelijat.nimi = ? \n";

   PreparedStatement p2 = yhteys.prepareStatement(sqlLause2);
   p2.setString(1, opiskelija);

   try{
      ResultSet r2 = p2.executeQuery();
      if(r2.next()) {//Opiskelija löytyy
          // Otsikko
          System.out.println("kurssi\t\t"+"op\t"+"päiväys\t\t"+"arvosana");

          do{
              System.out.println(r2.getString("nimi")+"\t\t"+
              r2.getInt("laajuus")+"\t"+
              r2.getString("paivays")+"\t"+
              r2.getInt("arvosana"));
          }while(r2.next());
      }else{// Jos ei löydy opiskelijaa
          System.out.println("Opiskelijaa ei löytynyt");
      }
      r2.close();
      p2.close();
   }catch(SQLException e2) {
      System.out.println(""+e2.getMessage());
   }

// 3 - Kurssin nimi
}if(toiminto == 3) {
    System.out.print("Anna kurssin nimi: ");
    String kurssi = lukija.nextLine();

    String sqlLause3 =
    "SELECT AVG(arvosana) AS keskiarvo \n"+
    "FROM Suoritukset, Kurssit \n"+
    "WHERE Suoritukset.kurssi_id = Kurssit.id \n"+
    "AND Kurssit.nimi \n"+
    "LIKE ? \n";

    PreparedStatement p3 = yhteys.prepareStatement(sqlLause3);
    p3.setString(1, kurssi);

    try
    {
        ResultSet r3 = p3.executeQuery();
        if(r3.next()) {
            if(r3.getDouble("keskiarvo") > 0) {
                System.out.println(r3.getDouble("keskiarvo"));
            }else{
                System.out.println("Kurssia ei löytynyt");
            }
```

```
                }
                r3.close();
                p3.close();
            }catch(SQLException e3) {
                System.out.print(e3.getMessage());
            }
        // 4 - Top <numero> opettajien annetut pisteet
        }if(toiminto == 4) {
            System.out.print("Anna opettajien määrä: ");
            int topOpe= Integer.valueOf(lukija.nextLine());

            String sqlLause4 =
            "SELECT Opettajat.nimi, SUM(arvosana) op \n"+
            "FROM Opettajat, Kurssit, Suoritukset \n"+
            "WHERE Suoritukset.kurssi_id = Kurssit.id \n"+
            "AND Kurssit.opettaja_id=Opettajat.id \n"+
            "GROUP BY Opettajat.id  \n"+
            "ORDER BY op DESC \n"+
            "LIMIT ? \n";

            PreparedStatement p4 = yhteys.prepareStatement(sqlLause4);
            p4.setInt(1, topOpe);

            try
            {
                ResultSet r4 = p4.executeQuery();
                if(r4.next()) {
                    // Otsikko
                    System.out.println("opettaja\t\t"+"op");
                    do
                    {
                        // Välilyönnit, jotta tulostuu kauniimmalta
                        System.out.println(r4.getString("nimi")+
                            "     \t"+r4.getInt("op"));
                    }while(r4.next());
                }else{
                    System.out.println("Sarakkeita ei löydy");
                }
                r4.close();
                p4.close();
            }catch(SQLException e4) {
                System.out.println(e4.getMessage());
            }
        }
    }
    yhteys.close();
  }
}
```

Tehtävä 2

En ole ennen suunnitellut näin mittavaa tietokantaa. Tämä harjoitustyö 2 antoi hyvää kokemusta tietokantojen suunnittelusta. Tämän tietokannan suunnittelu kesti minulla noin 12pv.

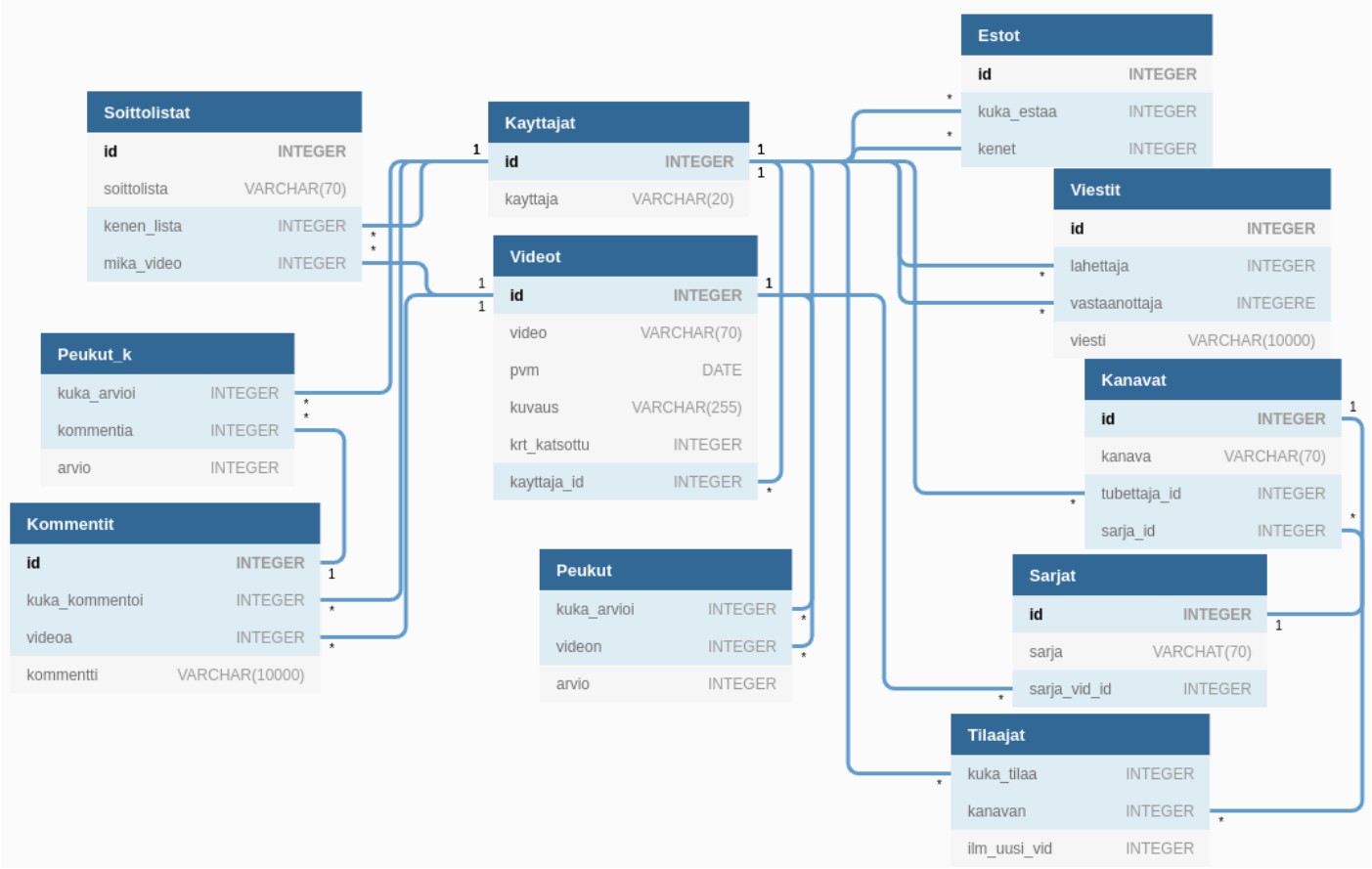
Olen käyttänyt sarakkeissa kuvaavia sarakkeiden nimiä Esim. **Estot**-taulun sarakkeet: **kuka_estaa**, **kenet**, jotka molemmat viittaavat kohtaan **Kayttaja.id** kuvastavat mielestäni hyvin mitä halutaan tehdä.

Päädyin tälläiseen lähestymistapaan, koska se on mielestäni selkeämpää kuin että tauluissa lukisi vain *kayttaja_id*, tai *video_id*. Olen toki myös näin muutamassa taulussa tehnyt, mutta mielestäni ne muutamat taulut tarvitsivat *kayttaja_id*-tyylisen nimeämisen.

Olen mielestäni toteuttanut Luvun 5 oppimaani suunnitteluperiaatteita ja luvun 6.1 tekniikoita tiedon oikeellisuuteen. Olen lopputulokseen tyytyväinen ja pidän tietokantaa selkeänä.

Taulujen id-kenttiä ei tarvitse, lukuunottamatta viittauksia, erikseen luoda Id-numerot syntyvät automaattisesti MySQL/MariaDB tutun auto_increment -tyypin mukaan. SQLitessa riittää, kun tyypiksi laitaa INTEGER ja PRIMARY KEY.

Tietokantakaavio:



SQL-skeema:

```
CREATE TABLE Kayttajat (id INTEGER PRIMARY KEY, kayttaja VARCHAR(20));
CREATE TABLE Videot (id INTEGER PRIMARY KEY, video VARCHAR(70), pvm DATE, kuvaus
VARCHAR(255), krt_katsottu INTEGER, kayttaja_id INTEGER REFERENCES Kayttajat(id));
CREATE TABLE Peukut (kuka_arvioi INTEGER REFERENCES Kayttajat(id), videon INTEGER
REFERENCES Videot(id), arvio INTEGER, UNIQUE(kuka_arvioi, videon), CHECK(arvio >=0 AND
arvio <= 1));
CREATE TABLE Kommentit (id INTEGER PRIMARY KEY, kuka_kommentoi INTEGER REFERENCES
Kayttajat(id), videoa INTEGER REFERENCES Videot(id), kommentti VARCHAR(10000));
CREATE TABLE Peukut_k (kuka_arvioi INTEGER REFERENCES Kayttajat(id), kommentia INTEGER
REFERENCES Kommentit(id), arvio INTEGER, UNIQUE(kuka_arvioi, kommentia), CHECK(arvio
>=0 AND arvio <= 1));
CREATE TABLE Kanavat (id INTEGER PRIMARY KEY, kanava VARCHAR(70), tubettaja_id INTEGER
REFERENCES Kayttajat(id), sarja_id INTEGER REFERENCES Sarjat(id));
CREATE TABLE Sarjat (id INTEGER PRIMARY KEY, sarja VARCHAT(70), sarja_vid_id INTEGER
REFERENCES Videot(id));
CREATE TABLE Tilaajat (kuka_tilaa INTEGER REFERENCES Kayttajat(id), kanavan INTEGER
DEFAULT 0 REFERENCES Kanavat(id), ilm_uusi_vid INTEGER, UNIQUE(kuka_tilaa, kanavan),
CHECK(ilm_uusi_vid >=0 AND ilm_uusi_vid <= 1));
CREATE TABLE Soittolistat (id INTEGER PRIMARY KEY, soittolista VARCHAR(70), kenen_lista
INTEGER REFERENCES Kayttajat(id), mika_video INTEGER REFERENCES Videot(id));
CREATE TABLE Viestit (id INTEGER PRIMARY KEY, lahettaja INTEGER REFERENCES
Kayttajat(id), vastaanottaja INTEGERE REFERENCES Kayttajat(id), viesti VARCHAR(10000));
CREATE TABLE Estot (id INTEGER PRIMARY KEY, kuka_estaa INTEGER REFERENCES
Kayttajat(id), kenet INTEGER REFERENCES Kayttajat(id), UNIQUE(kuka_estaa, kenet),
CHECK(kuka_estaa != kenet));
```

Vaatimukset

1. Käyttäjä voi etsiä videoita antamalla sanan, joka esiintyy videon nimessä tai kuvauksessa.

Kayttajat -taulun **kayttaja** -kenttään tulee käyttäjän nimi
Videot -tauluun **video** -kenttään tulee videon nimi
pvm -kenttään tulee päivämäärä muodossa ‘YYYY-MM-DD’
kuvaus -kenttään tulee videon kuvaus ja
kayttaja_id -kenttään tulee **Kayttajat** -taulun **id**, riippuen kenen käyttäjän video on kyseessä. Kun nämä kohdat on täytetty voi vaatimusten mukaisesti hakea videoita nimen tai kuvauksen perusteella.

Käyttöliittymässä koodaisin niin, että **pvm** tulisi automaattisesti silloisen päivämäärän mukaan, milloin video lisätään sivulle.

6. osa 1 / 2. Videoissa näkyy katsojien määrä

Lisäksi käyttöliittymässä koodaisin niin, että **krt_katsottu** -kentän arvoa nostetaan yhdellä aina, kun joku on katsonut videon.

2. Käyttäjä voi arvioida videon (peukku ylös tai alas) ja videon yhteydessä näkyy yhteenveto käyttäjien arvioista. Sama käyttäjä voi antaa vain yhden arvion videolle.

Peukut -taulun **kuka_arvioi** -kenttään tulee **Kayttajat.id,** riippuen kuka arvioi videota. Kuka arvioi videota on tässä tehtävässä lisänä, jota ei ole vaatimuksessa. Näin toteutettuna voi kontrolloida, että käyttäjä voi antaa vain yhden arvion videosta.

Käyttöliittymän puolella koodaisin kuitenkin niin, että arvioyhteys olisi anonyymina.

videon -kenttään tulee arvioidun videon **Videot.id**

arvio -kenttään tulee arvio numeroilla 0 tai 1.
Numero 0 tarkoittaa peukkua alas ja numero 1 tarkoittaa peukku ylös.
Sama käyttäjä voi antaa vain yhden arvion samasta videosta,
lisäksi **arvio** -kenttään pystyy lisämään ainoastaan numerot 0 tai 1.

3. Videon alla näkyy kommentteja käyttäjiltä. Myös näissä voi antaa arvion samaan tapaan kuin videossa (peukku ylös tai alas, vain yksi arvio samalta käyttäjältä).

Kommentit -taulun **kuka_kommentoi** -kenttään tulee **Kayttajat.id**, riippuen kuka kommentoi videota
videoa -kenttään tulee kommentoitavan videon **Videot.id**
kommentti -kenttään lisätään itse viesti mitä haluaa videolle kommentoida. Näin saadan vaatimusten kommentit aikaiseksi.

Peukuta kommentteja

Peukut_k -taulu toimii samoilla periaatteilla, kuin **Peukut** -taulu se vaan kontrolloi kommentien peukutusta.

Peukut_k -taulun **kuka_arvioi** -kenttään tulee kommentin arviojan **Kayttajat.id**
Peukut_k -taulun **kommentia** -kenttään tulee kommentojan **Kayttaja.id**
arvio -kenttän tulee numerot 0 tai 1, jolloin 0 tarkoittaa peukkua alas ja 1 tarkoittaa peukkua ylös.

Taulu on myöskin rakennettu niin, että sama käyttäjä voi antaa vain yhden arvion samasta videosta. Lisäksi **arvio** -kentässä voi syöttää ainoastaan numerot 0 tai 1.

Kyselyillä SQLitessa voi nähdä kuka arvioi kommentit, tällä tavoin voi kontrolloida, että käyttäjä voi antaa vaan yhden arvion. Käyttöliittymän koodaisin niin, että arviot olisivat anonyymejä.

4. Käyttäjä voi perustaa oman kanavan ja julkaista siellä videoita. Kanavan sisällä videoita voi luokitella sarjoihin.

Kanavat -taulun **kanava** -kenttään lisätään kanavan nimi
tubettaja_id -kenttään lisätään kanavan omistajan **Kayttaja.id**
sarja_id -kenttään lisätään **Sarjat** -taulun **id** -kenttä, riippuen mihiin sarjaan se kuuluu.

Sarjat -taulun **sarja** -kenttään tulee sarjan nimi
sarja_vid_id kenttään tulee **Videot.id** -tunnus.

Näillä kohdat täytettynä pystyy perustamaan kanavan ja kanavassa voi luoda mahdollisia sarjoja.

5. Käyttäjä voi tilata toisen käyttäjän kanavan, jolloin hän saa tietoa uusista videoista.

Tilaajat -taulun **kuka_tilaa** -kenttään tulee tilaajan **Kayttaja.id**
kanavan -kenttään tulee kanavan **Kanava.id**, jota tilataan
ilm_uusi_vid -kenttään tulee tieto saako käyttäjä tiedon uusista videoista, vai ei
i**lm_uusi_vid** -kenttään syötetään numerot 0 tai 1, muiden lukujen syöttö on estetty. 0 tarkoittaa, että tilaaja ei saa tietoa uusista videoista ja
1 tarkoittaa, että tilaaja saa tiedon uusista videoista.

6. osa 2 / 2. kanavissa näkyy tilaajien määrä.

Käyttöliittymässä näkyy Kanavat-sivulla tilaajien määrä. Tilausten määrän saa kyselyllä **Tilaukset** -taulusta.

7. Käyttäjä voi luoda soittolistoja, joihin voi valita videoita eri kanavista. Soittolistan videoilla on tietty järjestys.

Soittolistat -taulun **soittolista** -kenttään tulee soittolistan nimi
kenen_lista -kenttään tulee listan luonneen käyttäjän **Kayttaja.id**
mika_video -kenttään tulee **Videot.id**
Soittolistan järjestys riippuu siitä missä järjestyksessä **mika_video** -kentässä esintyy videoiden tunnukset.

8. Käyttäjä voi lähettää viestin toiselle käyttäjälle sekä estää toista käyttäjää lähettämästä viestejä hänelle.

Viestit -taulussa **lahettaja** -kenttään tulee viestin lähettäjän **Kayttajat.id**
vastaanottaja -kenttään tulee viestin vastaanottajan **Kayttaja.id**
viesti -kenttään tulee kirjoitettu viesti, jota lähettäjä haluaa lähettää vastaanottajalle.

Esto

Estot -taulussa **kuka_estaa** -kenttään tulee **Kayttaja.id** -kuka haluaa estää jonkin käyttäjän lähettämästä hänelle viestejä.
kenet -kenttään tulee estetyn käyttäjän **Kayttaja.id**
Sama käyttäjä ei voi estää samaa käyttäjää uudestaan, jota se ei ole jo kerran estänyt, eikä käyttäjä voi estää itseään.

Tehtävä 3

Tehokkuuslaskentan Harjoitustyö 3 meni koodamiseen noin 1h 18min ja ajatustyöhöön noin 8h 34min. Oli jännä huomata miten paljon indeksit auttaa hakua ja myös se aikaero mihin kohtaan ne laitetaan.

Tehokkuuslaskennan ajat ja koot:

Testi 1: 71.101476724 s. Tiedoston koko: 41480192 tavua. 39 Mt
Testi 2: 5.524452821 s. Tiedoston koko: 53878784 tavua. 51 Mt
Testi 3: 4.739072219 s. Tiedoston koko: 52506624 tavua. 50 Mt

Tehokkuuslaskenta-ohjelman lähdekoodi:

```
/*
 * Tehokkuuslaskenta
 *
 * Testi 1.
 *    Luo Elokuvat taulu
 *    Saman transaktin sisällä: Lisää tauluun randomisti miljoona (1e6)
 *          elokuvaa ja vuosia, vuosilta 1900-2000.
 *    Ohjelma suorittaa 1000 kertaa kyselyn, jossa haetaan elokuvan määrä
 *          vuoden perusteella. Vuodet haetaan randomisti.
 *    Ilmoita lopuksi kulunut aika ja  tietokantatiedoston koko.
 *
 *
 * Testi 2.
 *    Luo Elokuvat taulu ja lisää  indeksi vuosi-sarakkeseen
 *    Saman transaktin sisällä: Lisää tauluun randomisti miljoona (1e6)
 *          elokuvaa ja vuosia, vuosilta 1900-2000.
 *    Luo indeksi vuosi-sarakkeseen
 *    Ohjelma suorittaa 1000 kertaa kyselyn, jossa haetaan elokuvan määrä
 *          vuoden perusteella. Vuodet haetaan randomisti.
 *    Ilmoita lopuksi kulunut aika ja  tietokantatiedoston koko.
 *
 * Testi 3.
 *    Luo Elokuvat taulu ja lisää  indeksi id-sarakkeseen
 *    Saman transaktin sisällä: Lisää tauluun randomisti miljoona (1e6)
 *          elokuvaa ja vuosia, vuosilta 1900-2000.
 *    Ohjelma suorittaa 1000 kertaa kyselyn, jossa haetaan elokuvan määrä
 *          vuoden perusteella. Vuodet haetaan randomisti.
 *    Ilmoita lopuksi kulunut aika ja  tietokantatiedoston koko.
 *
 */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Random;
import java.io.File;

/*
 * @author  Erkki Pokkinen
 * @version 1
 * @since   2020-06-20
 * @valmis  2020-06-21
 * @Harj3     1h 18min
 * @Ajatustyö 8h 34min
 *
 */

public class Tehokkuuslaskenta {
    public static void main(String[] args) throws SQLException {
```

```
try
{
    // Testi 1 alkaa
    String tiedosto = "elokuvat.db";
    Connection yhteys = DriverManager.getConnection("jdbc:sqlite:"+tiedosto);
    Statement stmnt = yhteys.createStatement();

  //System.out.println("Testi 1 alkaa");
    long kello_alku1 = System.nanoTime();

    String luo_kanta =
"CREATE TABLE Elokuvat(id INTERGER PRIMARY KEY, nimi TEXT, vuosi INTEGER(4))";

    stmnt.execute(luo_kanta);

    // Lisää dataa
    stmnt.execute("BEGIN TRANSACTION");

    String lisaa_riveja =
    "INSERT INTO Elokuvat(nimi, vuosi) VALUES(?, ?)";

    PreparedStatement ps1 = yhteys.prepareStatement(lisaa_riveja);
    for(int i = 1; i < 1e6+1; i++) {
        ps1.setString(1, "Babylon 5 osa"+i);
        ps1.setInt(2, satunnaisnro(1900, 2000));
        ps1.executeUpdate();
    }
    stmnt.execute("COMMIT");

    // Tee haku 1000-kertaa
    String luo_haku1 =
    "SELECT COUNT(*) tulos FROM Elokuvat WHERE vuosi = ?";
    PreparedStatement haku1 = yhteys.prepareStatement(luo_haku1);

    for(int j = 1; j < 1000+1; j++) {
        int satu = satunnaisnro(1900, 2000);
        haku1.setInt(1, satu);
        ResultSet rs1 = haku1.executeQuery();
        rs1.next();
      //System.out.print("Vuosi: "+satu+ ". ");
      //System.out.println("Elokuvia: "+rs1.getInt("tulos")+" kpl.");
    }

    long kello_loppu1 = System.nanoTime();
    System.out.print("Testi 1: " +(kello_loppu1 - kello_alku1)/1e9+ " sek. ");
    File tsto1 = new File(tiedosto);
    long tston_koko1 = tsto1.length();
    long tston_koko1m = tsto1.length()/1048576;
    System.out.print("Tiedoston koko: "+tston_koko1+ " tavua. ");
    System.out.println(tston_koko1m+ " Mt");
```

```
    stmnt.execute("DELETE FROM Elokuvat");
    tsto1.delete();
    ps1.close();
    stmnt.close();
    yhteys.close();
} catch(Exception e) {
    System.out.println(e.getMessage());
}
// Testi 1 Loppuu


//Testi 2 alkaa
try
{
    String tiedosto2 = "elokuvat.db";
    Connection yhteys2 = DriverManager.getConnection("jdbc:sqlite:"+tiedosto2);
    Statement stmnt2 = yhteys2.createStatement();

  //System.out.println("Testi 2 alkaa");
    long kello_alku2 = System.nanoTime();

    String luo_kanta2 =
"CREATE TABLE Elokuvat(id INTERGER PRIMARY KEY, nimi TEXT, vuosi INTEGER(4))";

    stmnt2.execute(luo_kanta2);

    String luo_indeksi2 =
"CREATE INDEX idx_vuosi ON Elokuvat (vuosi)";

    stmnt2.execute(luo_indeksi2);

    // Lisää dataa
    stmnt2.execute("BEGIN TRANSACTION");

    String lisaa_riveja2 =
    "INSERT INTO Elokuvat(nimi, vuosi) VALUES(?, ?)";

    PreparedStatement ps2 = yhteys2.prepareStatement(lisaa_riveja2);
    for(int i = 1; i <=1e6; i++) {
        ps2.setString(1, "Babylon 5 osa"+i);
        ps2.setInt(2, satunnaisnro(1900, 2000));
        ps2.executeUpdate();
    }
    stmnt2.execute("COMMIT");

    // Tee haku 1000-kertaa
    String luo_haku2 =
    "SELECT COUNT(*) tulos FROM Elokuvat WHERE vuosi = ?";
    PreparedStatement haku2 = yhteys2.prepareStatement(luo_haku2);

    for(int j = 1; j <= 1000; j++) {
        int satu2 = satunnaisnro(1900, 2000);
```

```
        haku2.setInt(1, satu2);
        ResultSet rs2 = haku2.executeQuery();
        rs2.next();
      //System.out.print("Vuosi: "+satu2+ ". ");
      //System.out.println("Elokuvia: "+rs2.getInt("tulos")+" kpl.");
    }

    long kello_loppu2 = System.nanoTime();
    System.out.print("Testi 2: " +(kello_loppu2 - kello_alku2)/1e9+ " sek. ");
    File tsto2 = new File(tiedosto2);
    long tston_koko2 = tsto2.length();
    long tston_koko2m = tsto2.length()/1048576;
    System.out.print("Tiedoston koko: "+tston_koko2+ " tavua. ");
    System.out.println(tston_koko2m+ " Mt");

    stmnt2.execute("DELETE FROM Elokuvat");
    tsto2.delete();
    ps2.close();
    stmnt2.close();
    yhteys2.close();
} catch (Exception e2) {
    System.out.println(e2.getMessage());
}
// Testi 2 Loppuu


// Testi 3 Alkaa
try
{
    String tiedosto3 = "elokuvat.db";
    Connection yhteys3 = DriverManager.getConnection("jdbc:sqlite:"+tiedosto3);
    Statement stmnt3 = yhteys3.createStatement();

  //System.out.println("Testi 3 alkaa");
    long kello_alku3 = System.nanoTime();

    String luo_kanta3 =
"CREATE TABLE Elokuvat(id INTERGER PRIMARY KEY, nimi TEXT, vuosi INTEGER(4))";

    stmnt3.execute(luo_kanta3);


    // Lisää dataa
    stmnt3.execute("BEGIN TRANSACTION");

    String lisaa_riveja3 =
    "INSERT INTO Elokuvat(nimi, vuosi) VALUES(?, ?)";

    PreparedStatement ps3 = yhteys3.prepareStatement(lisaa_riveja3);
    for(int i = 1; i <=1e6; i++) {
        ps3.setString(1, "Babylon 5 osa"+i);
        ps3.setInt(2, satunnaisnro(1900, 2000));
```

```
            ps3.executeUpdate();
        }
        stmnt3.execute("COMMIT");

        String luo_indeksi3 =
      "CREATE INDEX idx_vuosi ON Elokuvat (vuosi)";

        stmnt3.execute(luo_indeksi3);

        // Tee haku 1000-kertaa
        String luo_haku3 =
        "SELECT COUNT(*) tulos FROM Elokuvat WHERE vuosi = ?";
        PreparedStatement haku3 = yhteys3.prepareStatement(luo_haku3);

        for(int j = 1; j <= 1000; j++) {
            int satu3 = satunnaisnro(1900, 2000);
            haku3.setInt(1, satu3);
            ResultSet rs3 = haku3.executeQuery();
            rs3.next();
        }

        long kello_loppu3 = System.nanoTime();
        System.out.print("Testi 3: " +(kello_loppu3 - kello_alku3)/1e9+ " sek. ");
        File tsto3 = new File(tiedosto3);
        long tston_koko3 = tsto3.length();
        long tston_koko3m = tsto3.length()/1048576;
        System.out.print("Tiedoston koko: "+tston_koko3+ " tavua. ");
        System.out.println(tston_koko3m+ " Mt");

        stmnt3.execute("DELETE FROM Elokuvat");
        tsto3.delete();
        ps3.close();
        stmnt3.close();
        yhteys3.close();
      } catch(Exception e3) {
        System.out.println(e3.getMessage());
      }
    }
    // Testi 3 Loppuu

    // Metodi
    public static int satunnaisnro(int min, int max) {
        Random rnd = new Random();

        int rand_nro = rnd.nextInt((max - min) + 1) +min;

        return rand_nro;
    }
}
```

Tehtävä 4

Normalisointi, eli lyhyesti kannan optimointia, tai kannan järkeistymistä, jolloin poistetaan turhat toistot.

Ymmärrän tämän niin, että ei ohjelmoinnissakaan ole hyvä olla toistoa, pahoittelen kun tämän raportin koodeissani on toistoa ja Tehtävä 3:ssa myös ihan copy-pastea omasta koodista, mutta haluan toki kehittyä ja tehdä tulevaisuudessa koodini, jossa on vähemmän toistoa, tai tostoa ollenkaan, miksei sitten tietokannoissakin voisi käyttää samaa periaatetta. Nerokasta!

Normalisoinnin ensimmäinen vaihe, tunnetaan myös lyhenteenä **1NF,** täyttyy mikäli tietokannassa täytyvät nämä seuraavat säännöt:

1. Sarakkeet eivät saa sisältää listoja. Yhdessä sarakkeessa saa olla vain yksi tieto, ei esimerkiksi monta numeroa pilkulla erotettuna.

2. Sarakkeissa ei saa olla toistuvia ryhmiä. Esimerkiksi jos sarakkeessa on **puhelinnro1** ja **puhelinnro2** jne. Tämä rikkoo sääntöä 2, joten puhelinnumeroille on järkevämpää tehdä oma taulu.

3. Saman sarakkeen arvot pitää olla samantyyppisiä. Esimerkiksi sarakkeessa **nimi**, joka kuuluu TEXT, VARCHAR-tyyppiseen tyyppiin, saa olla vain nimi, joka on TEXT, VARCHAR-tyyppinen tieto. Tässä kohdassa ei saa lukea muuta tyyppiä, tai tietoa esim. Osoite, puhelinumero, hinta (double-tyyppinen), tai kellonaika (date-tyyppinen), vaan pelkästään nimi.

4. Jokaisen sarakkeen nimi pitää olla uniikki. Samassa taulussa ei saa olla täsmälleen samannimisiä sarakkeita. Ei voi olla kahta **nimi** -saraketta, vaan: **etunimi**, **sukunimi**.

5. Sarakkeiden järjestyksellä ei saa olla tietokannan toimintaan vaikutusta. Eli jos taulussa muutetaan sarakkeiden järjestystä, kysely antaa silti saman tuloksen.

6. Tietokannan taulussa ei saa olla täsmälleen samanlaista riviä. Pohdin tätä kohtaa kauan ja ymmärtäisin tämän lopuksi niin, että rivillä ei saisi olla toisteista tietoa, jossa ovat kaikki kohdat täsmälleen samoja. Esim id=1, nimi=Keltainen kolli, hinta=200 ja riveiltä ei saisi löytyä täsmälleen samaa tietoa. Ilmeisesti taulu pitäisi rakentaa niin, ettei sinne ole mahdollista tehdä toistoa, vaikka UNIQUE, tai CHECK-lauseella, jolla voidaan määritellä, että jotkut kohdat pitää olla uniikkeja.

7. Rivien järjestyksellä ei saa olla vaikutusta tietokannan toimintaan. Tietokannan pitää antaa saman kyselyn tulos, vaikka rivit muutettaisiin jälkeenpäin.

Toinen normalisoinnin vaihe, tunnetaan myös lyhenteenä **2NF**.

Toinen normalisoinnin vaihe on vahvasti kytköksissä sarakkeiden välisistä riippuvaisuuksista, eli käsite: funktionaalinen riippuvuus. Mikä vaan sarake **B** on funktiollisesti riippuvainen sarakkeesta **A**, koska **A** tunnistaa sarakkeen **B** uniikiksi. Jos sarake **B** on sarake **bandinnimi** ja sarake **A** on

taulun **BandiID**, voidaan sarake **BandiID**:n avulla yksilöidä bändinnimi, jolloin sarake: **bandinnimi** on riippuvainen **BandiID**:sta, koska esim. **Earthgrave** -niminen bändi, jonka **BandiID** on numero **1** on Suomesta, Vantaalta, kun taas samaniminen bändi, jonka **BandID** numero **2** on Saksasta, Trierista. **BandID** ei ole funktionaalisesti riippuvainen sarakkeesta: **bandinimi** kun samannimisiä bändejä voi olla monia, eli **A** ei ole riippuvainen **B**:stä, vaan **B** on funktionaalisesti riippuvainen **A**:sta.

2NF Pääperiaate

Jos taulussa, jokin muu sarake kuin **A,** ei ole funktionaalisesti rippuvainen sarakkeesta **A** eli perusaivaimesta (primary key). Tällöin on syytä tehdä näille funktionaalisesti riippumattomille sarakkeille oma taulu. Tämä sääntö on normalisoinnin toisen vaiheen (2NF) pääperiaate. Tämän säännön täytettyään 2NF on toteutunut.

Esimerkki

Juomat -taulu:

PanimoID	JuomaNro	Pvm	Panimo	Juoma	Maara
5060	1	24.6.2020	Hartwall	Pepsi	12
2455	2	20.6.2020	Olvi	Olvi Cola	10

Juomat-taulussa kaikki muut sarakkeet ovat riippuvaisia perusaivaimesta, paitsi sarake **Juoma** on riippuvainen vain sarakkeesta: **JuomaNro**. Tämä ei täytä 2NF:n perusperiaatetta, joten näille funktionaalisesti riippumattomille sarakkeille (**JuomaNro**, **Juoma**) on luotava oma taulu.

Normalisoinnin kolmas vaihe, eli 3NF, tai BCNF

Normalisoinnin kolmas vaihe on lyhenneltään 3NF, mutta yleisesti tästä vaiheesta käytetään säännöiltään tiukempaa kolmatta vaihetta, koska se on täsmällisempi. Sen lyhenne on BCNF, joka on lyhenne sanoista: Boyce-Codd Normal Mode, joka tulee sunnittelijoiden sukunimistä, jotka olivat: Raymond F. Boyce ja Edgar F. Codd. Joskus BCNF tunnetaan myös nimellä 3.5NF, tai Vahva 3NF. Normalisoinnin kolmas vaihe on yleensä relaation viimeinen vaihe, sillä vaihe 4 ja 5 ovat harvinaisempia.

Tärkeimmät periaatteet:

Jotta tietokanta olisi normalisoinnin kolmannessa vaiheessa (3NF, BCNF) täytyy sen ensiksi olla normalisoinnin toisessa vaiheessa (2NF).

Transitiivisen riippuvuuden poistaminen, eli poista sarakkeet, jotka eivät ole riippuvaisia taulun **id**:stä (primary key).

Esimerkki:

Taulussa **juomakori** on sarakkeet **id, juoman_nimi, juomien_maara**, **korityyppi**. Kuvitellaan, että on olemassa juomia ja montako juomaa. Korit ovat eri suuruisia ja joihinkin koreihin voi mahtua eri maksimimääriä juomia. Kun korityyppi vaihtuu, vaihtuu myös juomien maksimimäärä.

Juomakori

id	juoman_nimi	juomien_maara	korityyppi
1	Pepsi	12	P12
2	Pepsi	20	L20

Sarake **juomien_maara** on kytköksissä **id** -pääavaimeen vain sarakkeen: **juomian_nimi** kautta. Sarakkeeseen: **juomien_maara** tietuetta ei voi lisätä tauluun ennen, kun **juomien_nimi** on lisätty tietue, eli jokin juoma. Myöskin **korityyppi** on kytköksissä **id** -pääavaimeen sarakkeen **juomien_maara** kautta. Tätä yhteyttä kutsutaan transaktiiviseksi riippuvuudeksi.

Normaalimuodon kolmannessa vaiheessa on ratkaisu tähän pulmaan, luomalla erillinen taulu.

Normalisoinnin tavallinen kolmas vaihe (3NF) muuttaisi taulun seuraavanlaiseksi:

```
Juomat                          Maarat
| id | juoman_nimi | maara_id |   | id | juomien_maara | korityyppi |
|----|-------------|----------|   |----|---------------|------------|
| 1  | Pepsi       | 1        |   | 1  | 12            | P12        |
| 2  | Pepsi       | 2        |   | 2  | 20            | L20        |
```

BCNF

Katso mikä on vasemmaisin sarake, mikäli vasemmaisin sarake on taulun yliavain, eli pääavain. On taulu BCNF-normalisoitu.

```
Autot
| id | Auton_nimi | malli |
|----|------------|-------|
| 1  | Toyota     | Verso |
```

Autot -taulussa sarakkeen **malli**n vasemmanpuolisin sarake on **Auton_nimi**, sillä **malli** on kytköksissä **Auton_nimi** -sarakkeen kanssa. Sarake on siis riippuvainen sarakkeesta **Auton_nimi**, joka ei ole taulun vasemmanpuolisin sarake **id**. Toisin sanoen ei ole olemassa autoa nimeltään Verso, vaan se auto on Toyota Verso. Tämä ei siis täytä BCNF -sääntöä, jolloin taulu täytyy katkaista kahtia.

Lopputulos näyttäisi tältä:

```
Autot                Mallit
| id | Auton_nimi |   | id | malli |
|----|------------|   |----|-------|
| 1  | Toyota     |   | 1  | Verso |

Automallit
| autot_id | mallit_id |
|----------|-----------|
| 1        | 1         |
```

Normalisoinnin neljäs vaihe, eli 4NF

Neljäs (4NF) ja viides (5NF) vaihe ovat harvinaisempia normalisoinnin vaiheita, näistä vaiheista ei ole materiaalia yhtä paljon, kuin ensimmäisestä kolmesta vaiheesta.

4NF periaatteet

4NF pitää olla BCNF, ennen kuin se on 4NF

4NF on sama, kuin BCNF, mutta lisäksi taulukosta pitää poistaa (tehdä eri taulu) kahden, tai useamman ominaisuuden arvoinen riippuvuus. Eli ns. Moniarvoinen riippuvuus.

Moniarvoinen riippuvuus tarkoittaa sitä, että kaksi asia ovat riippuvaisia kolmannesta asiasta, eli esimerkiksi juusto ja piimä ovat riippuvaisia maidosta, ilman maitoa ei ole piimää, eikä juustoa.

4NF -taulu on myös aina BCNF -taulu.

BCNF-taulu:

```
Maito                    Maitotuotteet
| id | tila          |    | maito_id | jaatelon_nimi | piiman_nimi  |
|----|---------------|    |----------|---------------|--------------|
| 1  | Laitilan tila |    | 1        | Aino          | Gefilius     |
```

4NF-taulu:

```
Maito                   Jaatelo                  Piima
| id | tila          |   | id | jaatelon_nimi |   | id | piiman_nimi |
|----|---------------|   |----|---------------|   |----|-------------|
| 1  | laitilan tila |   | 1  | Jäätelö kesä  |   | 1  | Gefilius    |

maitojaatelo               maitopiima
| maito_id | jaatelo_id |   | maito_id | piima_id |
|----------|------------|   |----------|----------|
| 1        | 1          |   | 1        | 1        |
```

Normalisoinnin viides vaihe. 5NF

Periaatteet:

Jotta taulu voi olla 5NF, pitää sen ensiksi olla 4NF.

Jos hävimättä tietoa voidaan esittää tauluilla, jossa on vähemmän sarakkeita. On silloin taulut tehtävä pienimmiksi. Jotta tauluja on useita, mutta tauluissa on vähemmän sarakkeita.

Kun pienenetään sarakkeita ja tehdään uusia tauluja, jos tietoa häviää silloin sarakkeiden pienentämistä uusilla taululla ei pitäisi tehdä, eli edellinen toimeenpide on silloin peruutettava.

Tavallinen tietokantojen suunnittelu verrattuna normalisointiin

Normalisointi on selvästi tiukempi tapa suunnitella tietokantoja, kuin ns. Tavallinen tietokantojen suunnittelu, mielestäni normalisointi antaa säännöillään kehittää paremman tietokannan ja ainakin normalisoinnin ensimmäisessä vaihessa (1NF) on oikeasti hyviä sääntöjä, joita kannattaa vähintäänkin noudattaa, jos jättää muut vaiheet toteuttamatta..

Lähteet:

Tietokantojen perusteet, syksy 2019
https://tietokantojen-perusteet-19.mooc.fi/osa-4/1-tietokannan-normalisointi

Youtube: Studytonight - First Normal Form (1NF) | Database Normalization | DBMS
https://youtu.be/mUtAPbb1ECM

AMKoodari: Metropolia - SQL ja relaatiotietokannat
https://vw4.viope.com/student/4356/#/theory/44103/5791

GitHub: yumoL - learningProgramming
https://github.com/yumoL/learningProgramming/blob/master/dokumentaatio/tietokankaavio.md

Oulun seudun AMK: Miika Kontio - Tietokannan kehittäminen kodinmonitorointijärjestelmän sensorimittauksille.
https://www.theseus.fi/bitstream/handle/10024/56451/Kontio_Miika.pdf

Enclypedia Metallum: Earthgrave
https://www.metal-archives.com/bands/Earthgrave/3540408377

Wikipedia: Tietokannan normalisointi
https://fi.wikipedia.org/wiki/Tietokannan_normalisointi#Lyhyt_yhteenveto_normaalimuodoista

Wikipedia: Boyce–Codd normal form
https://en.wikipedia.org/wiki/Boyce%E2%80%93Codd_normal_form

Youtube: Computer Science - Database Normalisation: Third Normal Form
https://youtu.be/_K7fcFQowy8

Haaga-Helia: Outi Virkki – Tietokantasuunnittelu
http://myy.haaga-helia.fi/~virou/TKS/mats/tksu9908_nf45.pdf

CSE-A1200 Tietokannat
http://www.cse.hut.fi/fi/opinnot/CSE-A1200/K2016/luennot/kalvot6-15032016.pdf

Microsoft - Tietokannan normalisoinnin perusteet
https://docs.microsoft.com/fi-fi/office/troubleshoot/access/database-normalization-description
Luento 4 - ER-mallin muuntaminen relaatiotietokannaksi ja normalisointi

Luento 4 - ER-mallin muuntaminen relaatiotietokannaksi ja normalisointi
http://appro.mit.jyu.fi/2000/yhteistoiminta/tietokannat/luennot/luento4/

Youtube: Studytonight - 5th Normal Form (5NF) | Join Dependency | Database Normalization
https://youtu.be/mbj3HSK28Kk