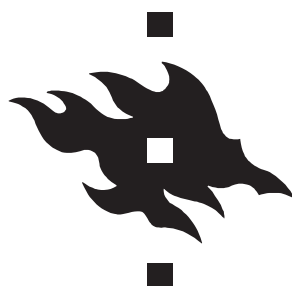


Tietokantojen perusteet, kevät 2020

Harjoitustyön raportti

Ville Manninen ^{*†}

20. helmikuuta 2020



HELSINGIN YLIOPISTO

^{*}mooc.fi: vw.manninen@gmail.com

[†]opiskelijanumero: 014922187

Sisältö

1	Esittely	3
2	Kaaviot ja Skeemat	5
3	Tehokkuustesti	6
4	Tietokannan eheys	7
5	Lähdekoodi	8

1 Esittely

Harjoitustyö on moniluokkainen komentoriviohjelma jonka avulla voidaan tallentaa asiakkaita, paikkoja, paketteja ja niihin liittyviä tapahtumia tietokantaan. Tietokantaan voidaan suorittaa ennakoon määriteltäviä kyselyitä, kyselyiden tulokset tulostetaan käyttäjän komentoriville. Ohjelmaan tallennetut tiedot jäävät muistiin tietokantaan ohjelman sulkemisen jälkeen.

Ohjelman ominaisuudet

Ohjelmalla on kirjoitushetkellä seuraavat ominaisuudet.

- Ohjelman tietokannan luominen kun tietokanta puuttuu tai on puutteellinen.
- Asiakkaiden lisääminen tietokantaan.
- Paikkojen lisääminen tietokantaan.
- Pakettien lisääminen tietokantaan.
- Tapahtumien lisääminen tietokantaan.
- Pakettien tapahtumien hakeminen.
- Asiakkaan pakettien hakeminen ja yksittäisten pakettien tapahtumien lukumäärä.
- Tapahtumien määrän hakeminen halutusta paikasta päivämäärällä.

Komentorivi komennot

Ohjelmalle annetaan syöttämällä komentoriville komentoa kuvaava numero(0-9).

- 1** lisää uuden asiakkaan tietokantaan.
- 2** lisää uuden paikan tietokantaan.
- 3** lisää uuden paketin tietokantaan.
- 4** lisää uuden tapahtuman tietokantaan.
- 5** tulostaa paketin tapahtumat tietokannasta.
- 6** tulostaa asiakkaan paketit ja niihin liittyvien tapahtumien määrän tietokannasta.
- 7** tulostaa tapahtumien määrän haku paikasta tietyllä päivämäärällä tietokannasta.
- 8** suorittaa tehokkuustestin ja tulostaa testin tulokset käyttäjälle.
- 9** luo tietokannan.
- 0** Sulkee ohjelman.

2 Kaaviot ja Skeemat

Tietokantakaavio

SQL-skeema

3 Tehokkuustesti

Ohjelmaan sisältyy automaattinen tehokkuustesti jonka avulla voidaan testata tietokannan tehokkuutta. Tehokkuustesti suorittaa ennalta määriteltyjä muutoksia tietokantaan ja ilmoittaa niihin käytetyn ajan. Testi koostuu kuudesta eri vaiheesta.

Testin vaiheissa yhdestä neljään tietokantaan lisätään tuhat asiakasta, paikkaa ja pakettia. Seuraavaksi tietokantaan lisätään miljoona tapahtumaa käyttäen satunnaisesti tietokantaan lisättyjä arvoja. Viimeisissä vaiheissa viisi ja kuusi testi suorittaa kyselyitä joissa haetaan tuhannen asiakkaan pakettien määrä, sekä tuhannen paketin tapahtumien määrä.

Ilman indeksejä

Ohjelmaa testatessa ilman lisättyjä hakemistorakenteita tuhannen arvon lisääminen tapahtui alle sekunnissa, ja miljoonaa arvoa lisätessä aikaa kului 10-12 sekuntia.

Kyselyitä suorittaessa tehokkuuteen vaikutti merkittävästi haettavan taulukon koko. Haettaessa asiakkaan pakettien määrää tuhannen arvon taulukosta aikaa kului alle sekunti, Kun haettiin paketin tapahtumien määrää miljoonan arvon taulukosta aikaa kului noin minuutti.

Indeksien lisäämisen jälkeen

Seuraavaksi ohjelman tehokkuustestiin lisättiin hakemistorakenne jonka avulla tapahtumat taulukolle luotiin indksi sarakkeen arvoille 'paketti.id'. Lisäyksen jälkeen tapahtumien haku paketin arvolla toimii huomattavasti tehokkaammin, lisäämisen jälkeen paketin tapahtumien määrän hakeminen, johon ennen kului aikaa yli minuutin, tapahtui alle sekunnissa.

Huomattavaa oli myös se missä kohdassa testiä indeksi lisättiin, kun indeksi lisättiin ennen miljoonan tapahtuman lisäämistä mainittujen tapahtumien luomiseen kului tuplasti enemmän aikaa.

4 Tietokannan eheys

Ohjelman tietokannan eheyttä pidetään yllä käyttämällä SQLiten sekä Javan tarjoamia ominaisuuksia. Tietokantaan ei ole mahdollista luoda samannimisiä asiakkaita tai paikkoja, eikä tietokantaan luoduille paketeille saa antaa samaa seurantakoodia. Samannimisiä syötteitä valvotaan SQLiten rajoitteella UNIQUE, joka varmistaa ettei samaan tauluun lisätä samannimisiä arvoja. Lisäksi SQLiten rajoitteella NOT NULL valvotaan että sarakkeisiin ei lisätä tyhjiä arvoja, tätä käytetään hyväksi esimerkiksi kun lisätään arvoja jotka viittaavat muihin sarakkeisiin, alikysely palauttaa arvon NULL jos arvoa johon viitataan ei löydy tietokannasta.

Java-ohjelma tarkistaa käyttäjän antamat syötteet ja ilmoittaa syyn jos annettu syöte on puutteellinen, esimerkiksi jos käyttäjän syöte on tyhjä. Tietokantaan lisättäessä paketteja tai tapahtumia jotka viittaavat muualle tallennettuun tietoon, sovellus varmistaa että annetut viitteet löytyvät tietokannasta.

5 Lähdekoodi

<https://github.com/Viltska/sql-app-training>.

Main.java

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

public class Main {

    public static void main(String[] args) throws SQLException {
        Scanner lukija = new Scanner(System.in);
        String databaseName = "tikape.db";
        System.out.println("Current database: " + databaseName);
        DatabaseManager manager = new DatabaseManager(databaseName);

        //Komentorivi ohjelma
        System.out.println("Komennot:");
        System.out.println("");
        System.out.println("1: Lisää asiakas ");
        System.out.println("2: Lisää paikka ");
        System.out.println("3: Lisää paketti");
        System.out.println("4: Lisää tapahtuma");
        System.out.println("5: Hae paketin tapahtumat");
        System.out.println("6: Hae asiakkaan paketit");
        System.out.println("7: Hae paikan tapahtumien määrä
            päivämäärällä");
        System.out.println("8: Tehokkuus testi");
        System.out.println("9: Luo tietokannan ja/tai puuttuvat
            taulukot");
        System.out.println("0: Lopettaa ohjelman");
        System.out.println("");

        while (true) {
            System.out.print("Syötä komento (0-9): ");
            String komento = lukija.nextLine();
            if (komento.equals("0")) {
                manager.tikapePrint();
                System.out.println("Ohjelma suljetaan.");
                break;
            }
            if (komento.equals("1")) {
                System.out.print("Syötä nimi: ");
                String nimi = lukija.nextLine();
                manager.uusiAsiakas(nimi);
            }
        }
    }
}
```



```

if (komento.equals("2")) {
    System.out.print("Syötä paikannimi: ");
    String paikka = lukija.nextLine();
    manager.uusiPaikka(paikka);
}

if (komento.equals("3")) {
    System.out.print("Syötä paketin asiakas: ");
    String asiakas = lukija.nextLine();
    System.out.print("Syötä paketin seurantakoodi: ");
    String koodi = lukija.nextLine();

    manager.uusiPaketti(asiakas, koodi);
}
if (komento.equals("4")) {
    System.out.print("Syötä tapahtuman paikka: ");
    String paikka = lukija.nextLine();
    System.out.print("Syötä paketin seurantakoodi: ");
    String seurantaKoodi = lukija.nextLine();
    System.out.print("Syötä tapahtuman kuvaus: ");
    String kuvaus = lukija.nextLine();
    manager.uusiTapahtuma(paikka, seurantaKoodi, kuvaus);
}
if (komento.equals("5")) {
    System.out.print("Syötä paketin seurantakoodi: ");
    String seurantaKoodi = lukija.nextLine();
    manager.haePaketinTapahtumat(seurantaKoodi);
}
if (komento.equals("6")) {
    System.out.print("Syötä asiakkaan nimi: ");
    String asiakas = lukija.nextLine();
    manager.haeAsiakkaanPaketit(asiakas);
}
if (komento.equals("7")) {
    System.out.println("Syötä päivämäärä muodossa
        'YYYY-MM-DD'");
    System.out.print("Päivämäärä: ");
    String pvm = lukija.nextLine();
    System.out.print("Syötä paikannimi: ");
    String paikannimi = lukija.nextLine();
    manager.haePaikanTapahtumatPaivamaaralla(pvm,
        paikannimi);
}
if (komento.equals("8")) {
    System.out.print("Tallennetaanko testin aikana tehdyt
        muutokset taulukkoon? (Y/N): ");
    String yesno = lukija.nextLine();
}

```

```

        boolean poistetaan = false;
        if (yesno.equals("N") || yesno.equals("n")) {
            poistetaan = true;
        }
        manager.tehokkuusTesti(poistetaan);
    }

    if (komento.equals("9")) {
        System.out.println("Tarkistetaan tietokantaa..");
        manager.createTables();
    }
}
}
}

```

DatabaseManager.java

```

import java.io.File;
import java.sql.Connection;
import java.sql.*;
import java.util.Random;

public class DatabaseManager {

    // Taulukko luokat
    private final Asiakkaat asiakkaat;
    private final Paikat paikat;
    private final Paketit paketit;
    private final Tapahtumat tapahtumat;
    // SQL
    private final String connectionName;
    private Connection db;

    public DatabaseManager(String database) throws SQLException {
        this.connectionName = "jdbc:sqlite:" + database;
        this.db = DriverManager.getConnection(connectionName);

        this.asiakkaat = new Asiakkaat(this.db);
        this.paikat = new Paikat(this.db);
        this.paketit = new Paketit(this.db);
        this.tapahtumat = new Tapahtumat(this.db);
    }

    public void uusiAsiakas(String nimi) throws SQLException {
        if (!nimi.isEmpty()) {
            this.asiakkaat.uusiAsiakas(nimi);
        } else {
            System.out.println("Asiakkaan nimi ei saa olla tyhjä");
        }
    }
}

```

```

    }
}

public void uusiPaikka(String nimi) throws SQLException {
    if (!nimi.isEmpty()) {
        this.paikat.uusiPaikka(nimi);
    } else {
        System.out.println("Paikannimi ei saa olla tyhjä");
    }
}

public void uusiPaketti(String asiakas, String seurantakoodi)
    throws SQLException {
    if (!asiakas.isEmpty() && !seurantakoodi.isEmpty()) {
        paketit.uusiPaketti(asiakas, seurantakoodi);
    } else {
        System.out.println("Syöte ei saa olla tyhjä");
    }
}

public void uusiTapahtuma(String paikka, String seurantaKoodi,
    String kuvaus) throws SQLException {
    if (!paikka.isEmpty() && !seurantaKoodi.isEmpty() &&
        !kuvaus.isEmpty()) {
        tapahtumat.uusiTapahtuma(paikka, seurantaKoodi, kuvaus);
    } else {
        System.out.println("Syöteet eivät saa olla tyhjä");
    }
}

public void haePaketinTapahtumat(String seurantaKoodi) throws
    SQLException {
    if (!seurantaKoodi.isEmpty()) {
        try {
            System.out.println("Paketin (" + seurantaKoodi + ")
                tapahtumat:");
            paketit.haePaketinTapahtumat(seurantaKoodi);
        } catch (SQLException e) {
            System.out.println(e);
        }
    } else {
        System.out.println("Syöte ei saa olla tyhjä");
    }
}

public void haeAsiakkaanPaketit(String asiakas) throws
    SQLException {
    if (!asiakas.isEmpty()) {

```

```

        try {
            asiakkaat.haeAsiakkaanPaketit(asiakas);
        } catch (SQLException e) {
            System.out.println(e);
        }
    } else {
        System.out.println("Syöte ei saa olla tyhjä");
    }
}

public void haePaikanTapahtumatPaivamaaralla(String paivamaara,
String paikannimi) throws SQLException {
    String regex =
        "^((19|20)\\d\\d)-(0?[1-9]|1[012])-(0?[1-9]|[12][0-9]|3[01])$";
    int paikka_id = paikat.getPaikkaID(paikannimi);
    if (paivamaara.matches(regex)) {
        if (paikka_id != -1) {
            try {
                System.out.println("Haetaan paikan (" +
                    paikannimi + ") tapahtumat, pvm(" +
                    paivamaara + ").");
                tapahtumat.haeTapahtumatPaikasta(paivamaara,
                    paikannimi);

            } catch (SQLException e) {
                System.out.println(e);
            }

        } else {
            System.out.println("Paikkaa ei löytynyt");
        }
    } else {
        System.out.println("Tarkista että päivämäärä on oikein
            ja muodossa 'YYYY-MM-DD'");
    }
}

public void tehokkuusTesti(boolean poistetaan) throws
    SQLException {
    //Luodaan testi tietokanta 'tehokkuus.db'
    this.db =
        DriverManager.getConnection("jdbc:sqlite:tehokkuus.db");
    createTables();
    Statement s = db.createStatement();
    db.setAutoCommit(false);
    Random rand = new Random(1137);

    //Tehokkuus testi

```

```

System.out.println("");
System.out.println("Tehokkuustesti: ");
System.out.println("");
try {

    // 1. Asiakkaiden lisäys
    long t1 = System.nanoTime();
    for (int i = 1; i < 1001; i++) {
        PreparedStatement p = db.prepareStatement("INSERT
            INTO Asiakkaat (nimi) VALUES (?");
        p.setString(1, ("A" + i));
        p.execute();
    }
    long t2 = System.nanoTime();
    System.out.println("Lisätty 1000 asiakasta ajassa: " +
        (t2 - t1) / 1e9 + "s.");

    // 2. Paikkojen lisäys
    for (int i = 1; i < 1001; i++) {
        PreparedStatement p = db.prepareStatement("INSERT
            INTO Paikat (paikannimi) VALUES (?");
        p.setString(1, ("P" + i));
        p.execute();
    }
    long t3 = System.nanoTime();
    System.out.println("Lisätty 1000 paikkaa ajassa: " + (t3
        - t2) / 1e9 + "s.");

    // 3. Pakettien lisäys
    for (int i = 1; i < 1001; i++) {
        PreparedStatement p = db.prepareStatement("INSERT
            INTO Paketit (asiakas_id,seurantakoodi) VALUES
            ((SELECT id FROM Asiakkaat WHERE nimi = ?),?");
        p.setString(1, ("A" + i));
        p.setString(2, ("koodi" + i));
        p.execute();
    }
    long t4 = System.nanoTime();
    System.out.println("Lisätty 1000 pakettia ajassa: " +
        (t4 - t3) / 1e9 + "s.");

    // 4. Tapahtumien lisäys
    for (int i = 1; i < 1000001; i++) {
        PreparedStatement p = db.prepareStatement("INSERT
            INTO Tapahtumat
            (paikka_id,paketti_id,date,kuvaus) VALUES
            (?, ?,datetime(), 'testi')");
        int j = rand.nextInt(1000) + 1;
    }
}

```

```

        p.setInt(1, j);
        p.setInt(2, j);
        p.execute();

    }
    long t5 = System.nanoTime();
    System.out.println("Lisätty 1000 000 tapahtumaa ja
        pakettia ajassa: " + (t5 - t4) / 1e9 + "s.");

    // 5. Asiakkaan pakettien määrä
    for (int i = 1; i < 1001; i++) {
        PreparedStatement p = db.prepareStatement("SELECT
            COUNT(*) FROM Paketit, Asiakkaat WHERE
            Paketit.asiakas_id = Asiakkaat.id AND
            Asiakkaat.nimi = ?");
        p.setString(1, ("A" + i));
        ResultSet r = p.executeQuery();
    }
    long t6 = System.nanoTime();
    System.out.println("Haettu 1000 asiakkaan pakettien
        määrä ajassa: " + (t6 - t5) / 1e9 + "s.");
    s.execute("CREATE INDEX idx_paketti ON Tapahtumat
        (paketti_id)");

    // 6. Paketin tapahtumien määrä
    for (int i = 1; i < 1001; i++) {
        int j = rand.nextInt(1000) + 1;
        PreparedStatement p = db.prepareStatement("SELECT
            COUNT(*) FROM Tapahtumat, Paketit WHERE
            Tapahtumat.paketti_id = Paketit.id AND Paketit.id
            = ?");
        p.setInt(1, j);
        ResultSet r = p.executeQuery();
    }
    long t7 = System.nanoTime();
    System.out.println("Haettu 1000 paketin tapahtumien
        määrä ajassa: " + (t7 - t6) / 1e9 + "s.");

} catch (SQLException e) {
    System.out.println(e);
}
if (!poistetaan) {
    db.commit();
} else {
    db.rollback();
}

// Sulkee yhteyden tehokkuustesti-tietokantaan ja ottaa
// yhteyden alkuperäiseen tietokantaan

```

```

        this.db.close();
        this.db = DriverManager.getConnection(connectionName);
        db.setAutoCommit(true);
    }

    public void createTables() throws SQLException {
        Statement s = db.createStatement();
        try {
            s.execute("CREATE TABLE Asiakkaat (id INTEGER PRIMARY
                KEY, nimi TEXT NOT NULL UNIQUE)");
            System.out.println("Luotu taulukko 'Asiakkaat'");
        } catch (SQLException e) {
            System.out.println("Löytyi taulukko 'Asiakkaat'");
        }
        try {
            s.execute("CREATE TABLE Paikat (id INTEGER PRIMARY KEY,
                paikannimi TEXT NOT NULL UNIQUE)");
            System.out.println("Luotu taulukko 'Paikat'");
        } catch (SQLException e) {
            System.out.println("Löytyi taulukko 'Paikat'");
        }
        try {
            s.execute("CREATE TABLE Paketit (id INTEGER PRIMARY KEY,
                asiakas_id INTEGER NOT NULL, seurantakoodi TEXT NOT
                NULL UNIQUE)");
            System.out.println("Luotu taulukko 'Paketit'");
        } catch (SQLException e) {
            System.out.println("Löytyi taulukko 'Paketit'");
        }
        try {
            s.execute("CREATE TABLE Tapahtumat (id INTEGER PRIMARY
                KEY, paikka_id INTEGER NOT NULL, paketti_id INTEGER
                NOT NULL, date DATETIME, kuvaus TEXT NOT NULL)");
            System.out.println("Luotu taulukko 'Tapahtumat'");
        } catch (SQLException e) {
            System.out.println("Löytyi taulukko 'Tapahtumat'");
        }
        System.out.println("Tietokanta valmis.");
    }

    public void tikapePrint() {
        System.out.println("-----");
        System.out.println("      ");
        System.out.println("      ");
    }

```



```

        ResultSet r = p.executeQuery();

        if (r.next()) {
            return r.getInt("id");
        } else {
            return -1;
        }
    } catch (SQLException e) {
        System.out.println("Ongelma metodissa 'Asiakas.getID'");
        System.out.println(e);
    }
    return -1;
}

public void haeAsiakkaanPaketit(String asiakas) throws
SQLException {
    try {
        PreparedStatement p = db.prepareStatement("SELECT
            seurantakoodi, COUNT(Tapahtumat.paketti_id) FROM
            Paketit\n"
            + "LEFT JOIN Tapahtumat ON Tapahtumat.paketti_id
            = Paketit.id\n"
            + "LEFT JOIN Asiakkaat ON Asiakkaat.id =
            Paketit.asiakas_id\n"
            + "GROUP BY Paketit.seurantakoodi HAVING
            Paketit.asiakas_id = (SELECT id FROM
            Asiakkaat WHERE nimi = ?)");
        p.setString(1, asiakas);
        ResultSet r = p.executeQuery();

        if (r.next() == false) {
            System.out.println("Asiakkaalla ei ollut paketteja");
        } else {
            System.out.println("Asiakkaan paketit");
            System.out.println("-----");
            do {
                System.out.println("Paketti (" +
                    r.getString("seurantakoodi") + ")");
                System.out.println("Tapahtumia(" +
                    r.getString("COUNT(Tapahtumat.paketti_id)") +
                    ")");
                System.out.println("-----");
            } while (r.next());
        }
    } catch (SQLException e) {
        System.out.println(e);
    }
}

```

```

    }

    }

}

```

Paikat.java

```

import java.sql.*;

public class Paikat {

    private final Connection db;

    public Paikat(Connection db) throws SQLException {
        this.db = db;
    }

    public void uusiPaikka(String paikannimi) throws SQLException {
        // Annettu paikka ei saa löytyä tietokannasta
        try {
            PreparedStatement p = db.prepareStatement("INSERT INTO
                Paikat (paikannimi) VALUES (?)");
            p.setString(1, paikannimi);
            p.executeUpdate();
            System.out.println("Paikka lisätty");
        } catch (SQLException e) {
            System.out.println("Ongelma metodissa
                'Paikat.uusiPaikka'");
            System.out.println(e);
        }
    }

    public int getPaikkaID(String paikka) throws SQLException {
        //Palauttaa arvon -1 jos paikkaa ei löydy tietokannasta
        try {
            PreparedStatement p = db.prepareStatement("SELECT id
                FROM Paikat WHERE paikannimi=?");
            p.setString(1, paikka);

            ResultSet r = p.executeQuery();

            if (r.next()) {
                return r.getInt("id");
            } else {
                return -1;
            }
        }
    }
}

```

```

    }
} catch (SQLException e) {
    System.out.println("Ongelma metodissa
        'Paikat.getPaikkaID'");
    System.out.println(e);
}
return -1;
}
}

```

Paketit.java

```

import java.sql.*;

public class Paketit {

    private final Connection db;

    public Paketit(Connection db) throws SQLException {
        this.db = db;
    }

    public void uusiPaketti(String asiakas, String seurantakoodi)
        throws SQLException {
        try {
            PreparedStatement p = db.prepareStatement("INSERT INTO
                Paketit (asiakas_id,seurantakoodi) VALUES ((SELECT
                id FROM Asiakkaat WHERE nimi = ?),?)");
            p.setString(1, asiakas);
            p.setString(2, seurantakoodi);
            p.executeUpdate();
            System.out.println("Paketti lisätty");
        } catch (SQLException e) {
            System.out.println("ongelma metodissa
                'Paketit.uusiPaketti'");
            System.out.println(e);
        }
    }

    public int getID(String seurantaKoodi) throws SQLException {
        try {
            PreparedStatement p = db.prepareStatement("SELECT id
                FROM Paketit WHERE seurantakoodi = ?");
            p.setString(1, seurantaKoodi);

            ResultSet r = p.executeQuery();

```

```

        if (r.next()) {
            return r.getInt("id");
        } else {
            return -1;
        }
    } catch (SQLException e) {
        System.out.println("Ongelma metodissa 'Paketit.getID'");
        System.out.println(e);
    }
    return -1;
}

public void haePaketinTapahtumat(String seurantaKoodi) throws
SQLException {
    int kertoja = 1;
    try {
        PreparedStatement p = db.prepareStatement("SELECT date,
            paikannimi, kuvaus FROM Tapahtumat \n"
            + "JOIN Paketit ON Paketit.id =
            Tapahtumat.paketti_id\n"
            + "JOIN Paikat ON Tapahtumat.paikka_id =
            Paikat.id\n"
            + "WHERE Paketit.id = (SELECT id FROM Paketit
            WHERE seurantakoodi = ?)");
        p.setString(1, seurantaKoodi);
        ResultSet r = p.executeQuery();

        if (r.next() == false) {
            System.out.println("Paketilla ei ollut tapahtumia");
        } else {
            System.out.println("-----");
            do {
                System.out.println("(" + kertoja + ")");
                System.out.println("TAPAHTUMA (" +
                    r.getString("date") + ")");
                System.out.println("Sijainti: " +
                    r.getString("paikannimi"));
                System.out.println("Kuvaus: " +
                    r.getString("kuvaus"));
                System.out.println("-----");
                kertoja++;
            } while (r.next());
            System.out.println("Tapahtumia yhteensä: " + (kertoja
                - 1));
        }
    } catch (SQLException e) {

```

```

        System.out.println(e);
    }

}

public int paketinTapahtumienMaara(int paketti_id) throws
SQLException {
    try {
        PreparedStatement p = db.prepareStatement("SELECT
            COUNT(*) FROM Tapahtumat, Paikat, Paketit WHERE
            Tapahtumat.paikka_id = Paikat.id AND
            Tapahtumat.paketti_id = Paketit.id AND Paketit.id =
            ?");
        p.setInt(1, paketti_id);
        ResultSet r = p.executeQuery();

        while (r.next()) {
            return 0;
        }

    } catch (SQLException e) {
        System.out.println(e);
    }

    return -1;
}
}

```

Tapahtumat.java

```

import java.sql.*;

public class Tapahtumat {

    private final Connection db;

    public Tapahtumat(Connection db) throws SQLException {
        this.db = db;
    }

    public void uusiTapahtuma(String paikannimi, String
        seurantaKoodi, String kuvaus) throws SQLException {
        try {
            PreparedStatement p = db.prepareStatement("INSERT INTO
                Tapahtumat (paikka_id,paketti_id,date,kuvaus) VALUES
                ((SELECT id FROM Paikat WHERE paikannimi =

```

```

        ?),(SELECT id FROM Paketit WHERE seurantakoodi =
        ?),datetime(),?)");
        p.setString(1, paikannimi);
        p.setString(2, seurantaKoodi);
        p.setString(3, kuvaus);
        p.executeUpdate();
        System.out.println("Tapahtuma lisätty");

    } catch (SQLException e) {
        System.out.println(e);
    }
}

public void haeTapahtumatPaikasta(String paivamaara, String
    paikannimi) throws SQLException {
    try {
        PreparedStatement p = db.prepareStatement("SELECT
            COUNT(Tapahtumat.id) AS Tapahtumia FROM Tapahtumat,
            Paikat WHERE Paikat.id = Tapahtumat.paikka_id AND
            DATE(Tapahtumat.date) = ? AND Paikat.paikannimi =
            ?");
        p.setString(1, paivamaara);
        p.setString(2, paikannimi);
        ResultSet r = p.executeQuery();

        if (r.next() == false) {
            System.out.println("Paikassa ei ollut tapahtumia
                annetulla päivämäärällä");
        } else {
            do {
                System.out.println("Tapahtumia: " +
                    r.getString("Tapahtumia"));
            } while (r.next());
        }

    } catch (SQLException e) {
        System.out.println(e);
    }
}
}

```