Elias Keski-Nisula
015084600
elias@keski-nisula.fi

Toteutetut toiminnot:

Tietokantojen perusteet 2020-kevät kurssin harjoitustyön tehtävänä oli toteuttaa sovellus, joka hakee ja lisää tietoa tietokantaan. Sovelluksen tomintaa voi kuvailla pelkistettynä pakettien seurantasovelluksella, jossa on asiakkaita, paikkoja, paketteja ja tapahtumia. Sovellukseen luoduilla toiminnoilla ja komennoilla tätä tietokantaa voi hallita.

Sovellus on ohjelmoitu java-kielellä, ja sovellus otta jdbc(java database connectivity)-rajapinnan avulla yhteyttä tietokantaan. Olen valinnut sovelluksen ratkaisutavaksi rakenteen, jossa on itse pääluokka Main (eli käyttöliittymä) ja luokka DatabaseHandler. Jälkimmäiseen luokkaan on toteutettu kaikki toiminnot jotka liittyvät tietokannan hallintaan, ja oliolle annetaan parametreinä yhteys tietokantaan (Connection) ja jonkinlainen lukija (Scanner). Käyttöliittymä kutsuu DatabaseHandler-luokan avulla metodeja, jotka tekevät varsinaisen työn.

Käyttöliittymän valintamekanismi on toteutettu while-silmukan ja switch-lausekkeen avulla. Switch-lauseke vertaa käyttäjän syötettä komentoihin jotka ovat 1-9 (ja -1). Tämän perusteella haluttua metodia kutsutaan. Komento -1 sulkee sovelluksen.

Seuraavat toiminnot ovat toteutettu metodeina DatabaseHandler-luokkaan:

1. Luo tietokannan ja asettaa halutut taulut tietokantaan
2. Lisää paikan (pyytää käyttäjältä syötettä)
3. Lisää asiakkaan (pyytää käyttäjältä syötettä)
4. Lisää paketin (pytää käyttäjältä syötettä; ensin seurantanumero ja tämän jälkeen asiakkaan nimeä)
5. Lisää tapahtuman (pytää käyttäjältä syötettä; ensin seurantanumeroa, tämän jälkeen paikkaa, ja lopuksi tapahtuman kuvausta). Tapahtuman ajankohta asettuu automaattisesti.
6. Listaa tapahtumat paketin seurantanumeron perusteella (pyytää käyttäjältä syötettä)
7. Listaa halutun asiakkaan paketit ja näiden tapahtumat (pyytää käyttäjältä syötettä)
8. Listaa tapahtumat halutussa paikassa halutun päivämäärän perusteella (pyytää käyttäjältä syötettä järjestyksessä paikka-ajankohta)
9. Suorittaa tietokannan tehokkuustestin

Komentojen järjestys on tärkeä. Paketin luonnissa annetun käyttäjän pitää olla kirjattuna tietokantaan. Samaten tapahtumaa lisätessä paketin seurantanumeron ja tapahtumapaikan tiedot jotka annetaan kuulu olla jo tietokannassa. Jollei nämä vaatimukset täyty, ohjelma ilmoittaa virheestä ja pyytää käyttäjältä uutta syötettä. Listaustoimintojen virhetilanteissa, esimerkiksi jos annetulla seurantanumerolla ei löydy pakettia, taikka asiakasta ei ole olemassa, ohjelma ilmoittaa myös tästä.

Näitten tehtävänantoon kuuluvien toimintojen lisäksi sovellukseen on myös toteutettu:

- Komento -1 sulkee tietokantasovelluksen
- Komento ”help” listaa ja kuvailee komennot

Tehokkuustesti ei käytä päätietokantaa, vaan luo uuden tietokannan sekä indeksittömälle ja indeksilliselle testille. Kun tehokkuustesti on päättynyt, tietokannoista pyyhitään tieto, jotta testit voidaan suorittaa uudestaan. Kun tehokkuustestit ovat valmiita, yhteys alkuperäiseen tietokantaan avataan uudelleen.

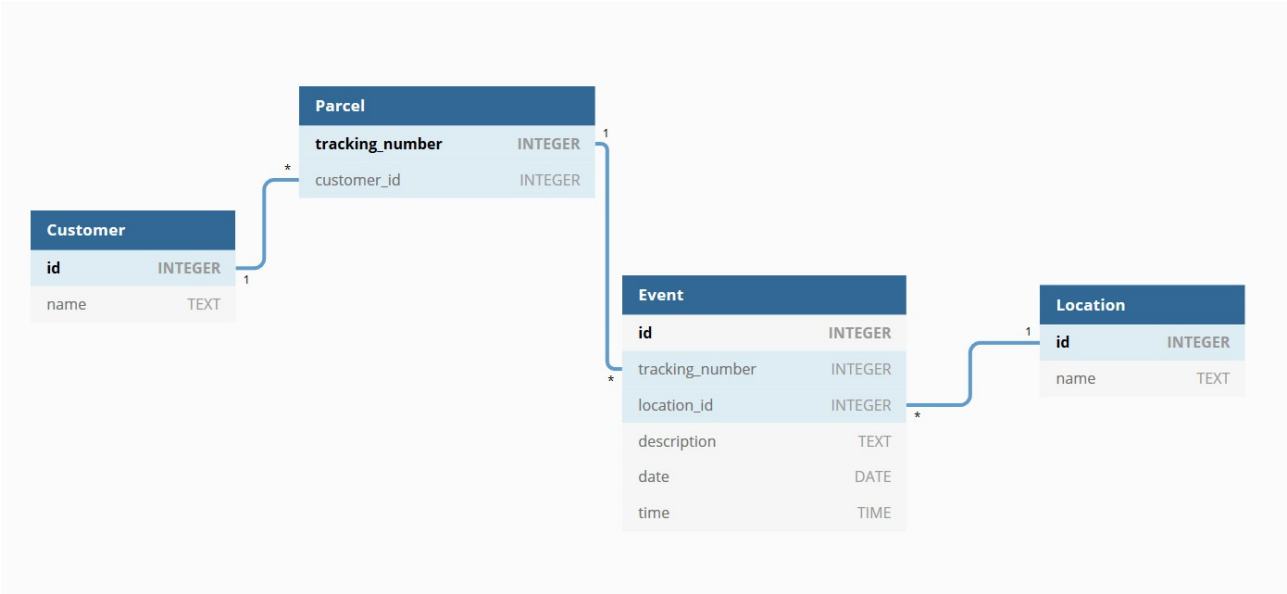
Testi tulostaa selkeästi vaaditut tiedot, ja ilmoittaa onko kyseessä indeksitön vai indeksillinen testi. Molemmat testit suoritetaan automaattisesti peräkkäin.

SQL-skeema ja tietokantakaavio

Tietokannan sql-skeema on seuraava:

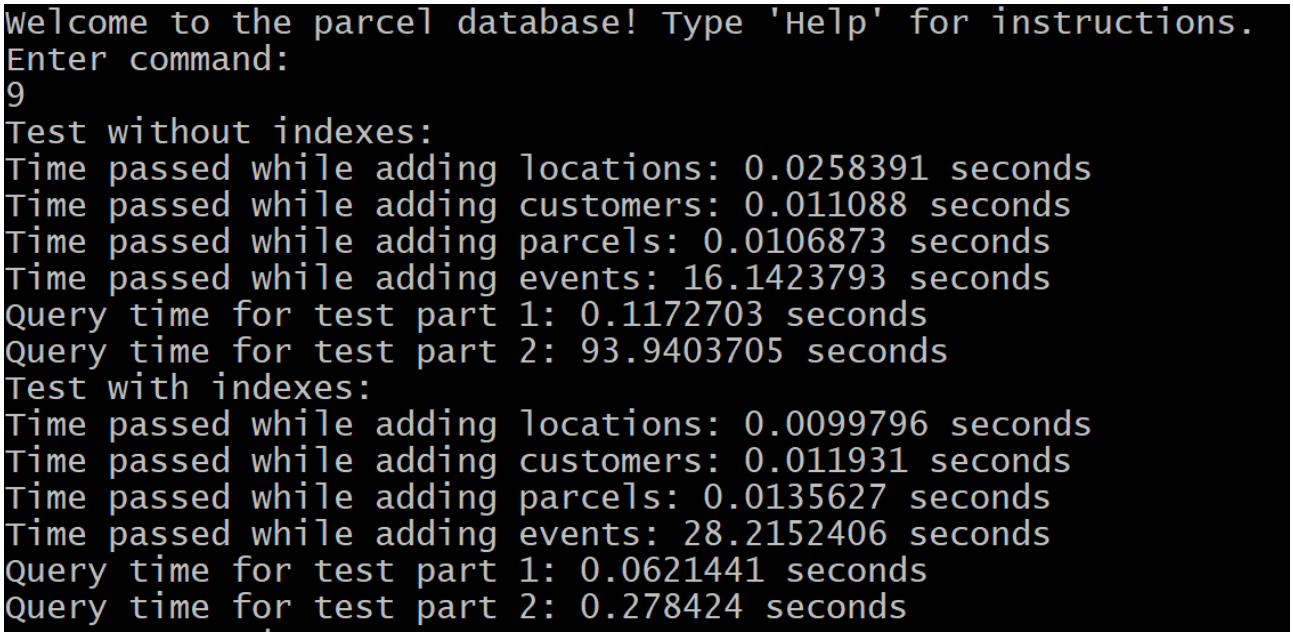
CREATE TABLE Customer (id INTEGER PRIMARY KEY, name TEXT UNIQUE NOT NULL);
CREATE TABLE Parcel (tracking_number INTEGER UNIQUE PRIMARY KEY, customer_id INTEGER, FOREIGN KEY(customer_id) REFERENCES Customer(id));
CREATE TABLE Event (id INTEGER PRIMARY KEY, tracking_number INTEGER, location_id INTEGER, description TEXT, date DATE, time TIME, FOREIGN KEY(tracking_number) REFERENCES Parcel(tracking_number), FOREIGN KEY(location_id) REFERENCES Location(id));
CREATE TABLE Location(id INTEGER PRIMARY KEY, name TEXT UNIQUE NOT NULL);

Vastaava tietokantakaavio:



Tehokkuustestin tulokset

Kuvankaappaus ohjelman tehokkuustestin tulosteesta:



Tulosteen ensimmäisen osan otsikon ’Test without indexes’ alla on testin tulokset ilman indeksiä, otsikon ’Test with indexes’ alla on testin tulokset indeksin kanssa.

Indeksöity tietokanta on huomattavan paljon nopea varsinkin testin viimeisessä osassa jossa suoritetaan 1000 kyselyä joissa haetaan paketin tapahtumien määrä. Ilman indeksejä kyseinen kysely kestää 93 sekuntia, kun taas indeksien kanssa kysely kestää noin 0,28 sekuntia. Ensimmäinen kysely, jossa haetaan tuhannen asiakkaan pakettien määrän, kestää myös puolet lyhyemmän ajan (0.117s / 0.062s).

Tapahtumien lisääminen puolestaan hidastui; syynä luultavasti indeksien luonti tapahtumille, jota ei suoriteta ilman indeksiä.

Tiedon eheys tietokannassa

Tiedon eheyttä tietokannassa ylläpidetään usealla eri tavalla. SQL-skeemaa tarkasteltaessa huomaa, että tietyille arvoille on lisätty NOT NULL- ja UNIQUE-ehdot. Asiakkaita tai paikkoja ei saa olla kahta samalla nimellä, ja UNIQUE-ehto pitää tästä huolen. Myös seurantanumero (joka on lisäksi Parcel-taulukon pääavain) on myös UNIQUE.

Jokainen päivitys ja kysely on myös luotu transaktioksi. Connection-luokan setAutoCommit(false)-metodilla kyselyiden automaattinen commitaaminen estetään. Tällöin muutokset ja lisäykset suoritetaan ainoastaan connectionname.commit()-metodin avulla. Jos kyselyn aikana tapahtuu virhe, try-catch lausekkeen catch-osiossa on toiminto

connectionname.rollback(), joka estää mahdolliset tiedon lisäyksen ja päivityksen virheet. Jokaisen metodin lopuksi automaattinen commit laitetaan päälle.

Jos commit ei onnistu, tai joku muu yllättävä virhe tapahtuu, sovellus ilmoittaa siitä metodin genericErrorMessage()-avulla, ja ohjaa käyttäjää tarkistamaan syötteen.

Sovelluksen lähdekoodi:

Sovellus löytyy myös github-repositoriosta https://github.com/ekeskini/TiKaPe20

Pääohjelma:

```
package main.java;

import java.sql.*;
import java.util.*;
import main.java.DatabaseHandler;

public class Main {

	public static void main(String[] args) throws SQLException {
		//Setting up required objects and classes
		Connection parceldb = DriverManager.getConnection("jdbc:sqlite:parceldatabase.db");
		Scanner scanner = new Scanner(System.in);
		DatabaseHandler dbhandler = new DatabaseHandler(parceldb, scanner);

		//A variable for checking if the user has finished
		boolean finished = false;
		System.out.println("Welcome to the parcel database! Type 'Help' for instructions.");
		//Loop functionality
		while (finished == false) {
			System.out.println("Enter command:");
			String input = scanner.nextLine();
			if (dbhandler.checkIfValidInt(input) == false) {
				if (input.trim().toLowerCase().equals("help")) {
					System.out.println(printHelp());
					continue;
				}
				System.out.println("Not a valid command. Enter a number between 0-9, -
1 to end the program.\n'Help' lists the functions of the commands.");
				continue;
			}
			Integer i = Integer.valueOf(input);
			switch(i) {
				case 1:
					dbhandler.createDatabase();
					break;
				case 2:
					dbhandler.addLocation();
					break;
				case 3:
					dbhandler.addCustomer();
					break;
				case 4:
					dbhandler.addParcel();
					break;
				case 5:
					dbhandler.addEvent();
					break;
				case 6:
					dbhandler.getEvents();
					break;
				case 7:
					dbhandler.getParcels();
					break;
				case 8:
					dbhandler.getEventsInLocation();
```

```
                                        break;
                                case 9:
                                        //Opening a new connection to another database for testing (as
not to affect the "proper" database)
                                        parceldb.close();

                                        //Test database without indexes
                                        System.out.println("Test without indexes:");
                                        Connection testdb =
DriverManager.getConnection("jdbc:sqlite:testdatabase.db");
                                        DatabaseHandler testdbhandler = new DatabaseHandler(testdb,
scanner);
                                        testdbhandler.createDatabase();
                                        testdbhandler.efficiencytest();
                                        testdbhandler.clearDatabase();
                                        testdb.close();

                                        //Test database with indexes
                                        System.out.println("Test with indexes:");
                                        Connection testdbindex =
DriverManager.getConnection("jdbc:sqlite:testdatabasewindex.db");
                                        DatabaseHandler testdbhandlerindex = new
DatabaseHandler(testdbindex, scanner);
                                        testdbhandlerindex.createDatabaseWIndexes();
                                        testdbhandlerindex.efficiencytest();
                                        testdbhandlerindex.clearDatabase();
                                        testdbindex.close();

                                        //Opening the connection the the original "proper" database
                                        parceldb =
DriverManager.getConnection("jdbc:sqlite:parceldatabase.db");
                                        dbhandler = new DatabaseHandler(parceldb, scanner);
                                        break;
                                case -1:
                                        finished = true;
                                        break;
                        }
                }
                System.out.println("Thank you for using the parcel database!");
        }
        public static String printHelp() {
                return "List of commands:"
                                + "\n1: Create database with appropriate tables if it does not exist"
                                + "\n2: Add a location"
                                + "\n3: Add a customer"
                                + "\n4: Add a parcel"
                                + "\n5: Add an event for a parcel"
                                + "\n6: List the events concerning a parcel"
                                + "\n7: List all parcels belonging to a certain customer"
                                + "\n8: List all events in a certain location"
                                + "\n9: Execute an efficiency test for the database"
                                + "\n-1: Close the database."
                                + "\nNote that when opening the database manager for the first time, \
n"
                                + "the command 1 has to be run.";
        }
}
```

DatabaseHandler-luokka:

```
package main.java;

import java.sql.*;
import java.sql.Date;
import java.time.Instant;
import java.util.*;

public class DatabaseHandler {
        private Connection dbconnection;
        private Scanner scanner;

        //Constructor which injects the database and the scanner to the class
        public DatabaseHandler(Connection connection, Scanner injectedscanner) {
                dbconnection = connection;
                scanner = injectedscanner;
        }

        //Procedure for command 2: Adding a location
```

```
	public void addLocation() throws SQLException{
		try {
			//Creating a transaction by turning autocommit mode off
			dbconnection.setAutoCommit(false);
			System.out.println("Enter location name:");
			String name = scanner.nextLine();

			//Preparing a parametrised statement
			PreparedStatement pstatement = dbconnection.prepareStatement("INSERT INTO
Location(name) VALUES (?);");
			pstatement.setString(1, name);

			//Executing statement and preparing for possible errors
			try {
				pstatement.executeUpdate();
				//Commiting the changes
				dbconnection.commit();
				System.out.println("Location added");
			}
			//Error message
			catch (SQLException e) {
				System.out.println("A location with the given name already exists.");

			}
		} catch (SQLException ex) {
			System.out.println(genericErrorMessage());
			dbconnection.rollback();
		}
		dbconnection.setAutoCommit(true);
	}

	//Procedure for command 3: adding a customer
	public void addCustomer() throws SQLException{
		try {
			//Creating a transaction by turning autocommit mode off
			dbconnection.setAutoCommit(false);
			System.out.println("Enter customer name:");
			String name = scanner.nextLine();

			//Preparing a parametrised statement
			PreparedStatement pstatement = dbconnection.prepareStatement("INSERT INTO
Customer(name) Values (?)");
			pstatement.setString(1, name);

			try {
				pstatement.executeUpdate();
				//Commiting the transaction
				dbconnection.commit();
				System.out.println("Customer added");
			}
			catch (SQLException e) {
				System.out.println("A customer with the given name already exists");

			}
		} catch (SQLException e) {
			System.out.println(genericErrorMessage());
			dbconnection.rollback();
		}
		dbconnection.setAutoCommit(true);
	}

	//Procedure for command 4: adding a parcel
	public void addParcel() throws SQLException {
		try {
			//Creating a transaction by turning autocommit mode off
			dbconnection.setAutoCommit(false);
			//Inputs by user
			System.out.println("Enter parcel tracking number:");
			String tn = scanner.nextLine();
			//Checking if the input is valid
			if (checkIfValidIntWMessage(tn) == false) {
				return;
			}
			Integer tracking_number = Integer.valueOf(tn);
			System.out.println("Enter customer name for parcel:");
			String customer_name = scanner.nextLine();

			//Preparing a variable for later use (the customer id is extracted from a
query later)
```

```
                        Integer customer_id = -1;

                        //Checking if customer exists
                        //Preparing a parametrised query where the customer name given by the user is
a parameter
                        PreparedStatement controlstatement = dbconnection.prepareStatement("SELECT
id, COUNT(*) as count FROM Customer WHERE Customer.name = (?)");
                        controlstatement.setString(1, customer_name);

                        try {
                                ResultSet rs = controlstatement.executeQuery();
                                if (rs.getInt("count") != 1) {
                                        System.out.println("No customer with the given name exists.
Please create a new customer or enter an existing customer name");
                                        //Returning to the main menu in cases where the query returns
more or less than one customer
                                        return;
                                }
                                //Selecting the customer id for adding the parcel
                                customer_id = rs.getInt("id");
                        }
                        catch (SQLException e) {
                                System.out.println("Problems with adding the parcel. Please check your
input and try again");
                        }
                        //Preparing the statement for adding a parcel
                        PreparedStatement pstatement = dbconnection.prepareStatement("INSERT INTO
Parcel (tracking_number, customer_id) VALUES (?,?)");
                        pstatement.setInt(1, tracking_number);
                        pstatement.setInt(2, customer_id);

                        try {
                                pstatement.executeUpdate();
                                dbconnection.commit();
                                System.out.println("Parcel added");
                        }
                        catch (SQLException e) {
                                System.out.println("Such an parcel already exists");
                        }
                } catch (SQLException ex) {
                        System.out.println(genericErrorMessage());
                        dbconnection.rollback();
                }
                dbconnection.setAutoCommit(true);
        }

        //Procedure for command 5: adding an event
        public void addEvent() throws SQLException{
                try {
                        dbconnection.setAutoCommit(false);
                        System.out.println("Enter parcel tracking number:");
                        String tn = scanner.nextLine();
                        if (checkIfValidIntWMessage(tn) == false) {
                                return;
                        }
                        Integer tracking_number = Integer.valueOf(tn);
                        System.out.println("Enter location:");
                        String location = scanner.nextLine();
                        System.out.println("Enter description of event:");
                        String description = scanner.nextLine();

                        //Preparing variable for later use (the actual value is extracted from a
query later)
                        Integer location_id = -1;

                        //Preparing queries for checking the database for existing location and
tracking number
                        PreparedStatement controlstatement1 = dbconnection.prepareStatement("SELECT
id, COUNT(*) as count FROM Location WHERE name = (?)");
                        PreparedStatement controlstatement2 = dbconnection.prepareStatement("SELECT
tracking_number, COUNT(*) as count FROM Parcel WHERE tracking_number = (?)");

                        //Executing queries
                        controlstatement1.setString(1, location);
                        controlstatement2.setInt(1, tracking_number);

                        //Comparing queries to expected return values
                        //Gives an error message and returns to the main menu if the parcel/location
does not exist
```

```
                        try {
                                ResultSet rs1 = controlstatement1.executeQuery();
                                ResultSet rs2 = controlstatement2.executeQuery();
                                if (rs1.getInt("count") != 1) {
                                        System.out.println("No location with the given name exists.
Please add a new location or enter a valid location name");
                                        return;
                                }
                                if (rs2.getInt("count") != 1) {
                                        System.out.println("No parcel with the given tracking number
exists. Please add a new parcel with the wanted parcel tracking number or enter a valid tracking
number");
                                        return;
                                }
                                location_id = rs1.getInt("id");
                        } catch (SQLException e){
                                System.out.println("Problems with adding the event. Please check your
input and try again");
                        }
                        Date date = Date.valueOf(java.time.LocalDate.now());
                        Time time = Time.valueOf(java.time.LocalTime.now());

                        //Preparing statement for adding a parcel
                        PreparedStatement pstatement = dbconnection.prepareStatement("INSERT INTO
Event (tracking_number, location_id, description, date, time) VALUES (?, ?, ?, ?, ?)");
                        pstatement.setInt(1, tracking_number);
                        pstatement.setInt(2, location_id);
                        pstatement.setString(3, description);
                        pstatement.setDate(4, date);
                        pstatement.setTime(5, time);

                        try {
                                pstatement.executeUpdate();
                                System.out.println("Event added");
                                dbconnection.commit();
                        } catch (SQLException e) {
                                System.out.println(genericErrorMessage());
                        }
                } catch (SQLException ex) {
                        System.out.println(genericErrorMessage());
                        dbconnection.rollback();
                }

                dbconnection.setAutoCommit(true);
        }

        //Procedure for command 6: listing events of a certain parcel
        public void getEvents() throws SQLException{
                dbconnection.setAutoCommit(false);
                try {
                        System.out.println("Enter parcel tracking number:");
                        String tn = scanner.nextLine();
                        if (checkIfValidIntWMessage(tn) == false) {
                                return;
                        }
                        Integer tracking_number = Integer.valueOf(tn);

                        //Preparing a statement for the query
                        PreparedStatement pstatement = dbconnection.prepareStatement("SELECT
Event.description as eventdescription, Location.name as locationname, "
                                        + "Event.date AS d, Event.time AS t FROM Event LEFT JOIN
Location "
                                        + "ON Event.location_id = Location.id WHERE
Event.tracking_number = ?");
                        pstatement.setInt(1, tracking_number);

                        try {
                                //Executing the query
                                ResultSet rs = pstatement.executeQuery();
                                dbconnection.commit();
                                System.out.println("List of events for parcel " + tracking_number +
":");
                                //Listing results
                                while (rs.next()) {

                                        System.out.println("Time: " + rs.getDate("d") + " " +
rs.getTime("t") + ", location: " + rs.getString("locationname") + ", description: " +
rs.getString("eventdescription"));
                                }
```

```
                        } catch (SQLException e) {
                                System.out.println(genericErrorMessage());
                        }
                } catch (SQLException e) {
                        System.out.println(genericErrorMessage());
                }
                dbconnection.setAutoCommit(true);
        }

        //Procedure for command 7: listing parcels and the count of events per parcel for given
customer
        public void getParcels() throws SQLException{
                dbconnection.setAutoCommit(false);
                try {
                        System.out.println("Enter customer name:");
                        String customername = scanner.nextLine();
                        //Preparing variable for later use (correct value extracted in query later)
                        Integer customerid = -1;

                        //Preparing statement for checking if named customer exists
                        PreparedStatement controlstatement = dbconnection.prepareStatement("SELECT
id, COUNT(*) AS count FROM Customer WHERE name = (?)");
                        controlstatement.setString(1, customername);

                        //Checking for the existence of named customer
                        try {
                                ResultSet rs = controlstatement.executeQuery();

                                //If there is no customer or there (for some reason) are more
customers than 1 with the same name
                                if (rs.getInt("count") != 1) {
                                        System.out.println("No customer with the given name exists.
Please check your input and try again");
                                        return;
                                }
                                //Extracting customer id
                                customerid = rs.getInt("id");
                        } catch (SQLException e){
                                System.out.println(genericErrorMessage());
                        }

                        //Preparing statement for wanted query
                        PreparedStatement pstatement = dbconnection.prepareStatement("SELECT
Parcel.tracking_number AS tn, COALESCE(COUNT(Event.id), 0) AS count FROM Parcel "
                                        + "LEFT JOIN Event ON Event.tracking_number =
Parcel.tracking_number "
                                        + "WHERE Parcel.customer_id = (?) "
                                        + "GROUP BY tn "
                                        + "ORDER BY tn");

                        pstatement.setInt(1, customerid);

                        //Executing query and listing parcels + amount of events concerning each
parcel
                        System.out.println("Parcels for customer " + customername + ":");
                        try {
                                ResultSet rs2 = pstatement.executeQuery();
                                dbconnection.commit();
                                while(rs2.next()) {

                                        System.out.println(rs2.getInt("tn") + ", number of events: " +
rs2.getInt("count"));
                                }
                        } catch (SQLException e) {
                                System.out.println(genericErrorMessage());
                        }
                } catch (SQLException e) {
                        System.out.println(genericErrorMessage());
                }
                dbconnection.setAutoCommit(true);
        }

        //Procedure for command 8: listing events in a specific location on a certain date
        public void getEventsInLocation() throws SQLException{
                dbconnection.setAutoCommit(false);
                try {
                        System.out.println("Enter location name:");
                        String locationname = scanner.nextLine();
```

```
                        //Setting up a date-type variable
                        Date d = Date.valueOf("0000-01-01");

                        System.out.println("Enter date (format YYYY-MM-DD), with the current date as
input 'today':");
                        //Setting the date-type object to the wanted date
                        while (true) {

                                String date = scanner.nextLine();

                                if (date.equals("today")) {
                                        d = Date.valueOf(java.time.LocalDate.now());
                                        break;
                                }
                                //Checking if input is of correct format
                                try {
                                        //If it is, change the value of the variable d to the wanted
date
                                        d = Date.valueOf(date);
                                } catch (Exception e) {
                                        System.out.println(genericErrorMessage());
                                        continue;
                                }
                                break;
                        }

                        //Preparing variable for later use
                        Integer locationid = -1;

                        //Preparing statement for extracting the location id based on the location
name
                        PreparedStatement controlstatement = dbconnection.prepareStatement("SELECT
id, COUNT(*) AS count FROM Location WHERE name = ?");
                        controlstatement.setString(1, locationname);

                        try {
                                ResultSet rs = controlstatement.executeQuery();

                                if(rs.getInt("count") != 1){
                                        System.out.println("No location with the given name exists.
Please check your input and try again");
                                        return;
                                }
                                locationid = rs.getInt("id");
                        } catch (SQLException e) {
                                System.out.println(genericErrorMessage());
                        }

                        //Preparing the main query, using the location id and date as parameters
                        PreparedStatement pstatement = dbconnection.prepareStatement("SELECT COUNT(*)
AS count FROM Event WHERE location_id = ? AND date = ?");
                        pstatement.setInt(1, locationid);
                        pstatement.setDate(2, d);

                        try {
                                ResultSet rs = pstatement.executeQuery();
                                dbconnection.commit();
                                while(rs.next()) {
                                        System.out.println("Events in location " + locationname + " on
date " + d.toString() + ": " + rs.getInt("count"));
                                }
                        } catch (SQLException e) {
                                System.out.println(genericErrorMessage());
                        }
                } catch (SQLException e) {
                        System.out.println(genericErrorMessage());
                }
                dbconnection.setAutoCommit(true);
        }

        //Procedure 9: efficiency test
        public void efficiencytest() throws SQLException{
                //Setting up required variables;
                Random rnd = new Random();
                Date date = Date.valueOf(java.time.LocalDate.now());
                Time time = Time.valueOf(java.time.LocalTime.now());
                try {
                        //Creating a transaction
                        dbconnection.setAutoCommit(false);
```

```
                        long locstart = System.nanoTime();
                        for (int i = 0; i <= 1000; i++) {
                                PreparedStatement stmt = dbconnection.prepareStatement("INSERT INTO
Location(name) VALUES (?)");
                                stmt.setString(1, "P" + i);
                                stmt.executeUpdate();
                        }
                        long locend = System.nanoTime();

                        System.out.println("Time passed while adding locations: " + (locend -
locstart)/1e9 + " seconds");

                        long custstart = System.nanoTime();
                        for (int i = 0; i <= 1000; i++) {
                                PreparedStatement stmt = dbconnection.prepareStatement("INSERT INTO
Customer(name) VALUES (?)");
                                stmt.setString(1, "A" + i);
                                stmt.executeUpdate();
                        }
                        long custend = System.nanoTime();

                        System.out.println("Time passed while adding customers: " + (custend -
custstart)/1e9 + " seconds");

                        long parcstart = System.nanoTime();
                        for (int i = 1; i <= 1001; i++) {
                                PreparedStatement stmt = dbconnection.prepareStatement("INSERT INTO
Parcel(tracking_number, customer_id) VALUES (?, ?)");
                                stmt.setInt(1, i);
                                //Adds the parcel to a random customer
                                stmt.setInt(2, rnd.nextInt(1001));
                                stmt.executeUpdate();
                        }
                        long parcend = System.nanoTime();

                        System.out.println("Time passed while adding parcels: " + (parcend -
parcstart)/1e9 + " seconds");

                        long eventstart = System.nanoTime();
                        for (int i = 0; i <= 1000000; i++) {
                                PreparedStatement stmt = dbconnection.prepareStatement("INSERT INTO
Event(tracking_number, location_id, description, date, time) VALUES (?, ?, ?, ?, ?)");
                                //Chooses a random parcel
                                stmt.setInt(1, rnd.nextInt(1001));
                                //Chooses a random location
                                stmt.setInt(2, rnd.nextInt(1001));
                                //Adds a generic description
                                stmt.setString(3, "Scan");
                                //Adds the current date and time
                                stmt.setDate(4, date);
                                stmt.setTime(5, time);
                                stmt.executeUpdate();
                        }
                        long eventend = System.nanoTime();

                        System.out.println("Time passed while adding events: " + (eventend -
eventstart)/1e9 + " seconds");
                        dbconnection.commit();
                } catch (SQLException e) {
                        dbconnection.rollback();
                }
                try {
                        long time1 = System.nanoTime();
                        for (int i = 0; i <= 1000; i++) {
                                PreparedStatement stmt = dbconnection.prepareStatement("SELECT
Customer.name AS name, COUNT(Parcel.tracking_number) AS count FROM Customer "
                                                + "LEFT JOIN Parcel ON Customer.id = Parcel.customer_id
WHERE Customer.name = (?) GROUP BY Customer.name");
                                stmt.setString(1, "A" + i);
                                stmt.executeQuery();

                        }
                        long time2 = System.nanoTime();
                        dbconnection.commit();
                        System.out.println("Query time for test part 1: " + (time2 - time1)/1e9 + "
seconds");
                } catch (SQLException e) {
```

```
                        dbconnection.rollback();
                }

                try {
                        long time1 = System.nanoTime();
                        for (int i = 0; i <= 1000; i++) {
                                PreparedStatement stmt = dbconnection.prepareStatement("SELECT
Parcel.tracking_number, COUNT(Event.id) FROM Event "
                                                + "JOIN Parcel ON Event.tracking_number =
Parcel.tracking_number WHERE Parcel.tracking_number = (?) GROUP BY Parcel.tracking_number");
                                stmt.setInt(1, i);

                                stmt.executeQuery();
                        }
                        long time2 = System.nanoTime();
                        dbconnection.commit();
                        System.out.println("Query time for test part 2: " + (time2 - time1)/1e9 + "
seconds");
                } catch (SQLException e) {
                        dbconnection.rollback();
                }

        }

        //Procedure 1: Creating database and the required empty tables
        //Database connection as a parameter for the Procedure
        public void createDatabase() throws SQLException{
                Statement s = dbconnection.createStatement();

                try {
                        //Creating the necessary tables
                        s.execute("CREATE TABLE Customer (id INTEGER PRIMARY KEY, name TEXT UNIQUE
NOT NULL)");
                        s.execute("CREATE TABLE Parcel (tracking_number INTEGER UNIQUE PRIMARY KEY,
customer_id INTEGER, "
                                        + "FOREIGN KEY(customer_id) REFERENCES Customer(id))");
                        s.execute("CREATE TABLE Event (id INTEGER PRIMARY KEY, tracking_number
INTEGER, location_id INTEGER, "
                                        + "description TEXT, date DATE, time TIME, FOREIGN
KEY(tracking_number) REFERENCES Parcel(tracking_number), "
                                        + "FOREIGN KEY(location_id) REFERENCES Location(id))");
                        s.execute("CREATE TABLE Location(id INTEGER PRIMARY KEY, name TEXT UNIQUE NOT
NULL)");
                }
                //catch statement for when the user executes the command "1" more than once
                catch (SQLException e) {


                }
        }

        //Creating database with indexes
        public void createDatabaseWIndexes() throws SQLException{
                Statement s = dbconnection.createStatement();

                try {
                        //Creating the necessary tables
                        s.execute("CREATE TABLE Customer (id INTEGER PRIMARY KEY, name TEXT UNIQUE
NOT NULL)");
                        s.execute("CREATE TABLE Parcel (tracking_number INTEGER UNIQUE PRIMARY KEY,
customer_id INTEGER, "
                                        + "FOREIGN KEY(customer_id) REFERENCES Customer(id))");
                        s.execute("CREATE TABLE Event (id INTEGER PRIMARY KEY, tracking_number
INTEGER, location_id INTEGER, "
                                        + "description TEXT, date DATE, time TIME, FOREIGN
KEY(tracking_number) REFERENCES Parcel(tracking_number), "
                                        + "FOREIGN KEY(location_id) REFERENCES Location(id))");
                        s.execute("CREATE TABLE Location(id INTEGER PRIMARY KEY, name TEXT UNIQUE NOT
NULL)");
                }
                //catch statement for when the user executes the command "1" more than once
                catch (SQLException e) {
                }
                //Creating indexes if they do not exist (if they exist, no changes to the database
are made)
                try {
                        dbconnection.setAutoCommit(false);

                        s.execute("CREATE INDEX idx_customername ON Customer (name)");
```

```
                s.execute("CREATE INDEX idx_parcelcustomerid ON Parcel (customer_id)");
                s.execute("CREATE INDEX idx_eventn ON Event (tracking_number)");

                dbconnection.commit();
            } catch (SQLException e) {
                dbconnection.rollback();
            }
            dbconnection.setAutoCommit(true);
        }
        //Clearing database for possibility to test in the future
        public void clearDatabase() throws SQLException{
            try {
                dbconnection.setAutoCommit(false);
                Statement s = dbconnection.createStatement();

                s.execute("DELETE FROM Parcel");
                s.execute("DELETE FROM Customer");
                s.execute("DELETE FROM Event");
                s.execute("DELETE FROM Location");

                dbconnection.commit();

            } catch (SQLException e) {
                dbconnection.rollback();
            }

            //Clearing possible indexes
            try {
                Statement s = dbconnection.createStatement();

                s.execute("DROP INDEX idx_customername");
                s.execute("DROP INDEX idx_parcelcustomerid");
                s.execute("DROP INDEX idx_eventn");

                dbconnection.commit();
            } catch (SQLException e) {

            }
            dbconnection.setAutoCommit(true);
        }
        //Checking if input is a valid integer (with an error message)
        public boolean checkIfValidIntWMessage(String i) {
            try {
                Integer.valueOf(i);

            } catch (Exception e) {
                System.out.println("Not an integer. Please check your input");
                return false;
            }
            return true;
        }
        //Checking if input is a valid integer (without an error message)
        public boolean checkIfValidInt(String i) {
            try {
                Integer.valueOf(i);

            } catch (Exception e) {
                return false;
            }
            return true;
        }

        //Generic error message for uncommon situations
        public String genericErrorMessage() {
            return "ERROR: An error occured. Please check your input and try again";
        }

}
```