Matthis Hauschild
07.05.2013

Aufgabe 1)

a) Erklären Sie das E/R-Modell

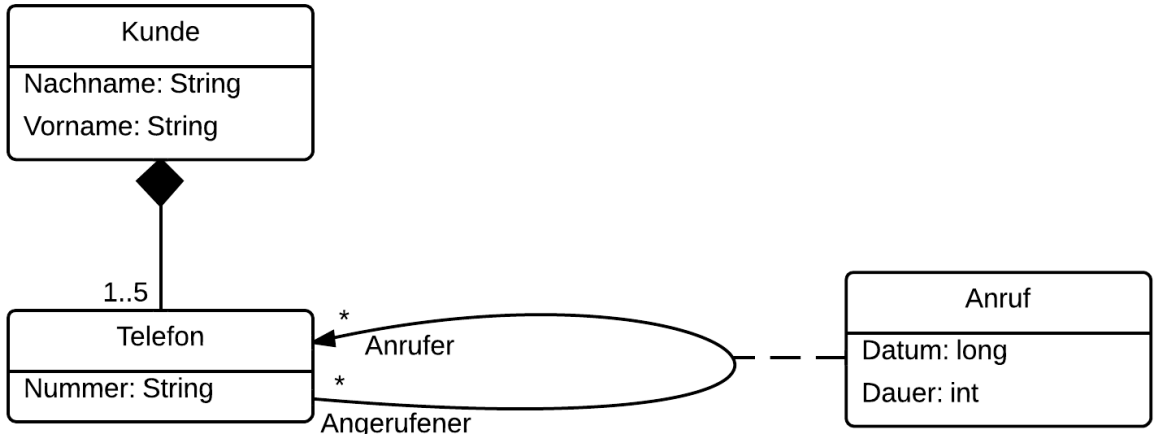

Abb. 1: ER-Diagramm erweitert

Das ER-Modell enthält zwei Entitäten *Kunde* und *Telefon* und eine Beziehung *Anruf* zwischen zwei Telefonen.
Ein Kunde hat die Attribute *Vorname* und *Nachname*, welche beide als String (varchar) realisiert sind. Ein Kunde besitzt 1 bis 5 Telefone, die jeweils eine Nummer haben, welche aus Formatierungsgründen als String abgelegt ist. Damit ist es möglich ein Plus, Slash oder auch Klammern anzugeben, wie es beispielsweise bei amerikanischen Telefonnummern üblich ist.
Die Beziehung *Anruf* besitzt die Attribute *Datum*, realisiert als long (timestamp), sodass auch die Uhrzeit direkt mit abspeichert werden kann und die *Dauer* eines Telefonats als Integer in Sekunden. Ein Anruf verweist auf zwei Telefon-Tupel.

b) E/R-Modell als Tabelle einer relationalen Datenbank

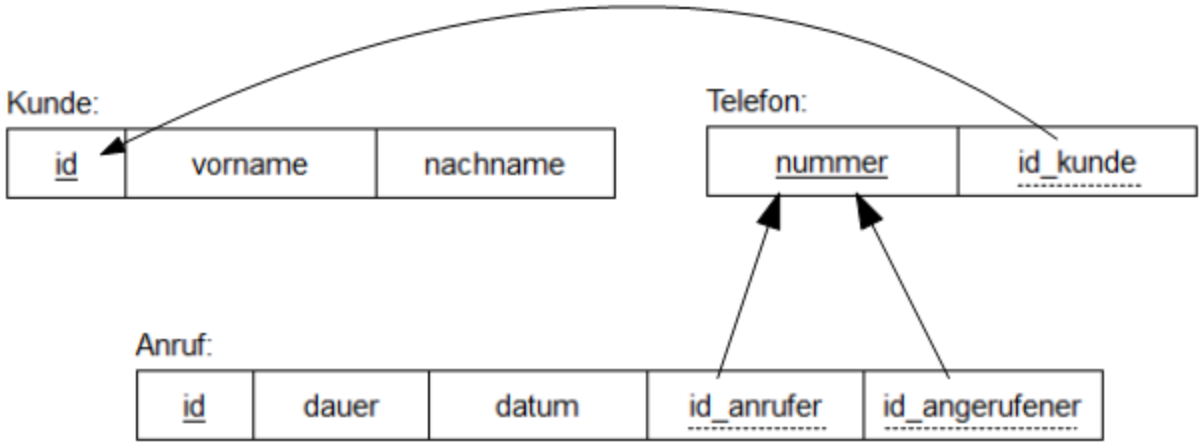

Abb. 2: Tabellenköpfe eines relationalen DB-Schemas

In a) gezeigtes E/R-Modell wurde nun in ein Datenbank-Schema übersetzt. Dabei haben die Tabellen Kunde und Anruf jeweils eine *id* bekommen, die das Flag *auto_increment* besitzt, um jeden Datensatz eindeutig zu machen. Die Tabelle Telefon benötigt das nicht, weil die Nummer bereits ein eindeutiges Attribut ist, deshalb ist dort eben jene der Primärschlüssel. Sekundärschlüssel sind gestrichelt unterstrichen und verweisen mit Pfeilen auf den Primärschlüssel (einfach unterstrichen) in einer anderen Tabelle.

c) geschätzter Sekundärspeicherbedarf

Zunächst kommt die Abschätzung der Größe mit meinen Daten, dann folgt die Abschätzung mit den Daten, die in der Aufgabenstellung gegeben sind.

Annahmen: 10.000 Kunden, 36.000.000 Anrufe, 30.000 Telefone (im Mittel)

Abschätzung mit eigenen Daten:

Tabelle Kunde:

- id: 8 Byte (64-bit System)
- vorname: varchar(100) => 100 * 1 Byte = 100 Byte (gesetz dem Fall, dass nur ASCII-Werte verwendet werden, selbst bei UTF-8)
- nachname: 100 Byte (siehe vorname)

=> 208 Byte

Tabelle Telefon:

- nummer: varchar(15) => 15 Byte (s.o.)
- id_kunde: 8 Byte

=> 23 Byte

Tabelle Anruf:

- id: 8 Byte
- dauer: int => 4 Byte
- datum: long => 8 Byte
- id_anrufer: 15 Byte
- id_angerufener: 15 Byte

=> 50 Byte

Blocklänge meiner HDD: b_s = 4096 Byte
Blockungsfaktor Kunden: $f_K = \frac{4096}{208} \approx 19$
Blockungsfaktor Telefone: $f_T = \frac{4096}{23} \approx 178$
Blockungsfaktor Anrufe: $f_A = \frac{4096}{50} \approx 81$

$b_K = \left\lceil \frac{|K|}{f_K} \right\rceil = \left\lceil \frac{10.000}{19} \right\rceil = 527$

$b_T = \left\lceil \frac{|T|}{f_T} \right\rceil = \left\lceil \frac{30.000}{178} \right\rceil = 169$

$b_A = \left\lceil \frac{|A|}{f_A} \right\rceil = \left\lceil \frac{36.000.000}{81} \right\rceil = 444.445$

$Größe = (b_K + b_T + b_A) * b_S = (527 + 169 + 444.445) * 4096\ Byte = 1.823.297.536\ Byte \approx 1,70\ GiB$

Abschätzung mit den Daten aus der Aufgabenstellung

$b_K = \left\lceil \frac{|K|}{f_K} \right\rceil = \left\lceil \frac{10.000}{50} \right\rceil = 200$

$b_T = \left\lceil \frac{|T|}{f_T} \right\rceil = \left\lceil \frac{30.000}{500} \right\rceil = 60$

$b_A = \left\lceil \frac{|A|}{f_A} \right\rceil = \left\lceil \frac{36.000.000}{200} \right\rceil = 180.000$

$Größe = (b_K + b_T + b_A) * b_S = (200 + 60 + 180.000) * 8000\ Byte = 1.442.080.000\ Byte \approx 1,34\ GiB$

d) Welche Indexe für welche Tabelle?

Annahmen & Fakten:

- Anzahl Kunden: $A_K = 10.000$
- Blockungsfaktor Kunden: $f_K = 50$
- Anzahl Blöcke für Kunden: $b_K = \left[\frac{10.000}{50}\right] = 200$
- Anzahl Telefone (im Mittel): $A_T = (1+5) / 2 * A_K = 30.000$
- Blockungsfaktor Telefone: $f_T = 500$
- Anzahl Blöcke für Telefone: $b_T = \left[\frac{30.000}{500}\right] = 60$
- Anzahl Anrufe: $A_A = 36.000.000$
- Blockungsfaktor Anrufe: $f_A = 200$
- Anzahl Blöcke für Kunden: $b_A = \left[\frac{36.000.000}{200}\right] = 180.000$
- Blockungsfaktor Index: $f_I = 100$
- Datenbankpuffer $b_M = 1000$ (Blöcke)

Anwendungsfall 1 ohne Index

Es muss durch die gesamte Tabelle “Kunde” iteriert werden, das Ergebnis wird mit der Tabelle “Telefon” gejoint, um die Telefonnummer rausfinden zu können. Es wird der Nested Loop Algorithmus betrachtet.

$$K_1 = b_K + \left[\frac{b_K}{b_M}\right] * b_T = 200 + \left[\frac{200}{1000}\right] * 60 = 260$$

Anwendungsfall 1 mit Index

Es wird auf die Kunden via Nach- und Vorname zugegriffen. Also wird ein Index über diese zwei Spalten gelegt.

$$b_{I_{N,Kunde}} = \frac{A_K}{f_I} = \frac{10.000}{100} = 100$$

Die Selektivität ist sehr hoch, weil Personen sehr selten gleich heißen und jede Person nur sehr wenige Telefone hat (nämlich im Schnitt 3 aus 30.000).

$$sel(\phi_1) = \frac{2}{10.000} * \frac{3}{30.000} = 0,00000002$$

$$K_{1I} = b_T + \left[\frac{b_T}{b_M}\right] * b_{I_{N,Kunde}} + sel(\phi_1) * A_K * A_T = 60 + \left[\frac{60}{1000}\right] * 100 + 0,00000002 * 10000 * 30000 = 166$$

Anwendungsfall 2 ohne Index

Es wird die Tabelle “Kunde” mit “Telefon” sowie “Anruf” gejoint. Es wird der Nested Loop Algorithmus verwendet. Die Kosten des Joins von Kunde und Telefon haben wir in Anwendungsfall 1 bereits berechnet.

$$K_2 = K_1 + \left[\frac{K_1}{b_M}\right] * b_A = 260 + \left[\frac{260}{1000}\right] * 180.000 = 180260$$

Anwendungsfall 2 mit Index

Es wird auf die Anruf-Tabelle zugegriffen. Dabei wird einerseits nach dem Datum (enthält auch die Uhrzeit) sowie andererseits nach der passenden Telefon-ID gesucht. Die Telefon-ID ist identisch mit der Telefonnummer, weil diese bereits eindeutig ist. Deshalb wird ein Index auf die Spalten “anrufer” und “datum” aus der Tabelle “Anruf” gelegt.

$$b_{I_{N,Anruf}} = \frac{A_A}{f_I} = \frac{36.000.000}{100} = 360.000$$

Jeder Kunde tätigt 300 Anrufe im Monat, daraus resultiert folgende Gesamtselektivität:
$sel(\phi_2) = \frac{300}{36.000.000} * sel(\phi_1) = \frac{300}{36.000.000} * \frac{6}{10.000 * 30.000}$

$$K_{2I} = K_{1I} + \left[\frac{K_{1I}}{b_M}\right] * b_{I_{N,Anruf}} + sel(\phi_2) * A_K * A_T * A_K = 166 + \left[\frac{166}{1000}\right] * 360.000 + 1800 = 361966$$

Weil der Blockungsfaktor des Indexes kleiner als der Blockungsfaktor der Anrufe ist, sind die Kosten mit Index größer.

Anwendungsfall 3 ohne Index

Bei dieser Aufgabe muss lediglich einmal über die Tabelle “Anruf” iteriert werden, also muss jeder Block genau einmal gelesen werden. Bei 1/12 der Daten, nämlich einem Monat, muss ein weiterer Zugriff zum Löschen gemacht werden.

$$K_3 = b_A + \frac{1}{12} b_A = 180.000 + \frac{1}{12} * 180.000 = 195.000$$

Anwendungsfall 3 mit Index

Es müssen die Datensätze aus einem bestimmten Monat identifiziert werden, daher wird nur ein Index über die Spalte “datum” in der Tabelle “Anruf” benötigt.

$$b_{I_{N,Anruf}} = \frac{A_A}{f_I} = \frac{36.000.000}{100} = 360.000$$

Es wird einmal durch den Index iteriert und alle betroffenen Datensätze neu geschrieben. Damit

sie neu geschrieben werden können, müssen sie vorher eingelesen werden, weil ja nur ein Teil entfernt werden soll.

$$K_{3I} = b_{I_{N,Anruf}} + 2 * \frac{1}{12} b_A = 360.000 + 2 * \frac{1}{12} * 180.000 \ = 390.000$$

Auch hier ist die Verwendung des Indexes signifikant teurer, weil der angenommene Indexblockungsfaktor nur halb so groß wie der Anrufsblockungsfaktor ist.

Anwendungsfall 4 ohne Index

Da zu Namen Anrufe gefunden werden sollen, muss über alle drei Tabellen gejoint werden. Die Kosten dafür wurden bereits in Anwendungsfall 2 ausgerechnet.

$$K_4 = K_2 = 180260$$

Anwendungsfall 4 mit Index

Es werden Anruf-Datensätze anhand des Anrufers beziehungsweise des Angerufenen gesucht, daher bekommen die Telefonnummern in der Tabelle “Anruf” einen Index, also die Spalten “anrufer” und “angerufener”.

Auch in diesem Anwendungsfall kann die Berechnung aus Anwendungsfall 2 genutzt werden, da wieder nur Indexe in der Tabelle “Anruf” gesetzt sind.

$$K_{4I} = K_{2I} = 361966$$

Aufgabe 2) db4o

b) Speicherbedarf der DB

1.808.415 kiB

c) Ausführungszeiten mit und ohne Indexe

Gerechnet mit Faktor 10

- Fall1 ohne Index: 5191 ms
- Fall1 mit Index: 729 ms
- Fall2 ohne Index: 4241975 ms
- Fall2 mit Index: 16614 ms
- Fall3 ohne Index: 2558 ms
- Fall3 mit Index: 6220 ms
- Fall4 ohne Index: 9317 ms
- Fall4 mit Index: 5208 ms

Aufgabe 3) HSQLDB

b) Speicherbedarf der DB

3.375.104 kiB

c) Ausführungszeiten mit und ohne Indexe

Gerechnet mit Faktor 10

- Fall1 ohne Index: 630 ms
- Fall1 mit Index: 498 ms
- Fall2 ohne Index: 3533641 ms
- Fall2 mit Index: 4048 ms
- Fall3 ohne Index: 3679 ms
- Fall3 mit Index: 1545 ms
- Fall4 ohne Index: 7240 ms
- Fall4 mit Index: 1827 ms

Aufgabe 4) Vergleich von Theorie und Praxis

Abweichung Speicherplatz

Der Speicherplatz der db4o-Datenbank weicht um $\frac{1.808.415\ kiB}{1.823.297.536\ Byte} \approx 1,56\ \%$ von der berechneten Größe, basierend auf meinen Daten ab. Basierend auf den in der Aufgabenstellung gegebenen Daten weicht er um $\frac{1.808.415\ kiB}{1.442.080.000\ Byte} \approx 28,41\ \%$ ab.
Der Speicherplatz der HSQL-Datenbank weicht um $\frac{3.375.104\ kiB}{1.823.297.536\ Byte} \approx 89,55\ \%$ von meinen Daten und um $\frac{3.375.104\ kiB}{1.442.080.000\ Byte} \approx 139,66\ \%$ von den Gegebenen ab.

Vergleich des Verhältnisses mit Index und ohne Index

	Fall 1	Fall 2	Fall 3	Fall 4
Kosten ohne Index	260	180.260	195.000	180.260
Kosten mit Index	166	361.966	360.000	361.966
Verbesserung	36,15 %	-100,8 %	-84,62 %	-100,8 %
db4o ohne Index	5191 ms	4241975 ms	2558 ms	9317 ms
db4o mit Index	729 ms	16614 ms	6220 ms	5208 ms
Verbesserung	85,96 %	99,61 %	-143,16%	44,10 %
HSQL ohne Index	630 ms	3533641 ms	3679 ms	7240 ms
HSQL mit Index	498 ms	4048 ms	1545 ms	1827 ms
Verbesserung	20,95%	99,89 %	58 %	74,77 %

Es ist zu erkennen, dass die gemessenen Ergebnisse teils sehr deutlich von den berechneten abweichen. Dies liegt hauptsächlich daran, dass der angenommene Indexblockungsfaktor von 100 kleiner als die Blockungsfaktoren der Tabellen Anruf und Telefon ist.
Aber auch beim Vergleich zwischen den Zeiten von db4o und HSQL sind teils signifikante Unterschiede festzustellen.
Abschließend kann man sagen, dass sich eine Aufwandsabschätzung nur dann lohnt, wenn man die Parameter des praktischen Systems sehr genau kennt, besonders Blockungsfaktoren, aber auch Cachegrößen spielen doch eine sehr wichtige Rolle. Außerdem sind die Ausführungszeiten extrem implementierungsabhängig.

```
package de.hauschil.dbprojekt.anwendungsfaelle;

import java.io.IOException;

public class Fallmanager {
    public static final int FAKTOR      = 10;
    public static final int ANZ_KUNDEN  = 100 * FAKTOR;
    public static final int ANZ_PARTNER = FAKTOR;
    public static final int ANZ_ANRUFPM = 3 * FAKTOR;
    public static final String DB4O_PATH =      "D:\\tmp\\db\
\telefongesellschaft.db4o";
    public static final String HSQL_PATH =      "D:\\tmp\\db\
\telefongesellschaft.hsql";

    private static ArrayList<DB_Controller> dbs = new ArrayList<>();
    private static long db_size;

    public static void main(String... args) throws IOException {
        dbs.add(new DB4O_Controller());
        dbs.add(new HSQL_Controller());

        for (DB_Controller db : dbs) {
            System.out.println(db);
            long timer = System.currentTimeMillis();
            setUp(db);
            System.out.println((System.currentTimeMillis() - timer) / 1000 + " seks
setuptime");
            System.out.println("DB size: " + db_size / 1024 + " kiB");

            Fall1 f1 = new Fall1(db);
            f1.run(true);
            f1.run(false);
            System.out.println(f1);

            Fall2 f2 = new Fall2(db);
            f2.run(true);
            /* Benötigt bei Faktor 10 schon über eine halbe Stunde */
            f2.run(false);
            System.out.println(f2);

            Fall4 f4 = new Fall4(db);
            f4.run(true);
            f4.run(false);
            System.out.println(f4);

            /* Fall3 als letztes, weil er Sachen löscht, deshalb muss auch neu
generiert werden */
            Fall3 f3 = new Fall3(db);
            f3.run(true);
            setUp(db);
            f3.run(false);
            System.out.println(f3);
        }
    }

    private static void setUp(DB_Controller db) throws IOException {
        Path dbPath = null;

        if (db instanceof DB4O_Controller) {
            dbPath = Paths.get(DB4O_PATH);
            Files.deleteIfExists(dbPath);
```

```
        } else if (db instanceof HSQL_Controller) {
            dbPath = Paths.get(HSQL_PATH + ".data");
            Files.deleteIfExists(dbPath);
            Files.deleteIfExists(Paths.get(HSQL_PATH + ".lck"));
            Files.deleteIfExists(Paths.get(HSQL_PATH + ".tmp"));
            Files.deleteIfExists(Paths.get(HSQL_PATH + ".log"));
            Files.deleteIfExists(Paths.get(HSQL_PATH + ".properties"));
            Files.deleteIfExists(Paths.get(HSQL_PATH + ".script"));
        } else {
            throw new RuntimeException("TODO");
        }

        db.initDBConnection();

        if (db instanceof HSQL_Controller) {
            db.createTables();
        }

        Kunde[] kunden = Kunde.generateKunden(ANZ_KUNDEN);
        System.out.println(ANZ_KUNDEN + " Kunden generiert");
        db.storeKunden(kunden);
        Anruf.generateAnrufe(kunden, ANZ_ANRUFPM, db);

        db.closeDBConncetion();
        /* Kurz warten, damit wirklich geschlossen wird */
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        db_size = Files.size(dbPath);
    }
}
```

```java
package de.hauschil.dbprojekt.anwendungsfaelle;

import de.hauschil.dbprojekt.controller.DB_Controller;
import de.hauschil.dbprojekt.controller.Index;
import de.hauschil.dbprojekt.model.Kunde;

public class Fall1 {
    private long[] anfangszeit;
    private long[] endzeit;
    private DB_Controller db;
    private Kunde[] gesuchte;

    public Fall1(DB_Controller db) {
        this.db = db;
        anfangszeit = new long[2];
        endzeit = new long[2];
        gesuchte = Kunde.generateKunden(1000);
    }

    public void run(boolean indexed) {
        db.initDBConnection(new Index[] {
            new Index(Kunde.class, "vorname", indexed),
            new Index(Kunde.class, "nachname", indexed)
        });

        anfangszeit[indexed ? 1 : 0] = System.nanoTime();
        for (int i = 0; i < 1000; i++) {
            /* liefert gesuchte Kunden zurück */
            db.getKunden(gesuchte[i % 1000].getVorname(), gesuchte[i %
1000].getNachname());
        }
        db.closeDBConncetion();

        endzeit[indexed ? 1 : 0] = System.nanoTime();
    }

    public long getTime(int which) {
        return (endzeit[which] - anfangszeit[which]) / 1000 / 1000;
    }

    @Override
    public String toString() {
        return "Fall1 ohne Index: " + getTime(0) + " ms\n" +
               "Fall1  mit Index: " + getTime(1) + " ms";
    }
}
```

```java
package de.hauschil.dbprojekt.anwendungsfaelle;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.GregorianCalendar;

import de.hauschil.dbprojekt.controller.DB_Controller;
import de.hauschil.dbprojekt.controller.Index;
import de.hauschil.dbprojekt.model.Anruf;
import de.hauschil.dbprojekt.model.Kunde;
import de.hauschil.dbprojekt.model.Telefon;

public class Fall2 {
    private long[] anfangszeit;
    private long[] endzeit;
    private DB_Controller db;

    public Fall2(DB_Controller db) {
        this.db = db;
        anfangszeit = new long[2];
        endzeit = new long[2];
    }

    public void run(boolean indexed) {
        db.initDBConnection(new Index[] {
            new Index(Anruf.class, "anrufer", indexed),
            new Index(Anruf.class, "datum", indexed)
        });
        /* null, null um keine Constraints zu haben und damit ALLE Kunden zu bekommen
*/
        ArrayList<Kunde> kunden = db.getKunden(null, null);
        ArrayList<Long> kosten = new ArrayList<>(kunden.size());

        anfangszeit[indexed ? 1 : 0] = System.nanoTime();

        for (int i = 0; i < kunden.size(); i++) {
            kosten.add(i, berechneKosten(kunden.get(i)));
        }

        db.closeDBConncetion();
        endzeit[indexed ? 1 : 0] = System.nanoTime();
    }

    public long getTime(int which) {
        return (endzeit[which] - anfangszeit[which]) / 1000 / 1000;
    }

    @Override
    public String toString() {
        return "Fall2 ohne Index: " + getTime(0) + " ms\n" +
               "Fall2  mit Index: " + getTime(1) + " ms";
    }

    public ArrayList<Anruf> getAnrufeFromDb(Kunde k, int month) {
        GregorianCalendar cal1 = new GregorianCalendar();
        cal1.set(2012, month, 1, 0, 0, 0);
        GregorianCalendar cal2 = new GregorianCalendar();
        cal2.set(2012, month, cal1.getActualMaximum(Calendar.DAY_OF_MONTH), 23, 59,
59);
        ArrayList<Anruf> list = new ArrayList<>();
```

```
        for (Telefon tel : k.getTelefone()) {
            list.addAll(db.getAnrufe(tel, null, cal1.getTimeInMillis(),
cal2.getTimeInMillis()));
        }

        return list;
    }

    private Long berechneKosten(Kunde k) {
        long kosten = 0;
        ArrayList<Anruf> anrufe = getAnrufeFromDb(k, Calendar.MAY);

        for (Anruf a : anrufe) {
            GregorianCalendar cal = new GregorianCalendar();
            cal.setTimeInMillis(a.getDatum());
            /* Am Wochenende ist Telefonieren günstiger */
            if (
                cal.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY ||
                cal.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY
            ) {
                /* Wenn beide die gleiche Vorwahl haben, gibts auch Rabatt */
                if (getVorwahl(a.getAnrufer()).equals(getVorwahl(a.getAngerufener())))
{
                    kosten += a.getDauer() / 60 * 4; /* cent/min */
                } else {
                    kosten += a.getDauer() / 60 * 9; /* cent/min */
                }
            } else {
                /* Wenn beide die gleiche Vorwahl haben, gibts auch Rabatt */
                if (getVorwahl(a.getAnrufer()).equals(getVorwahl(a.getAngerufener())))
{
                    kosten += a.getDauer() / 60 * 10; /* cent/min */
                } else {
                    kosten += a.getDauer() / 60 * 19; /* cent/min */
                }
            }
        }

        return kosten;
    }

    private static String getVorwahl(Telefon t) {
        return t.toString().split("/")[0];
    }
}
```

```java
package de.hauschil.dbprojekt.anwendungsfaelle;

import java.util.Calendar;
import java.util.GregorianCalendar;

import de.hauschil.dbprojekt.controller.DB_Controller;
import de.hauschil.dbprojekt.controller.Index;
import de.hauschil.dbprojekt.model.Anruf;

public class Fall3 {
    private long[] anfangszeit;
    private long[] endzeit;
    private DB_Controller db;

    public Fall3(DB_Controller db) {
        this.db = db;
        anfangszeit = new long[2];
        endzeit = new long[2];
    }

    public void run(boolean indexed) {
        db.initDBConnection(new Index[] {
            new Index(Anruf.class, "datum", indexed)
        });

        anfangszeit[indexed ? 1 : 0] = System.nanoTime();

        GregorianCalendar cal1 = new GregorianCalendar();
        cal1.set(2012, Calendar.JANUARY, 1, 0, 0, 0);
        GregorianCalendar cal2 = new GregorianCalendar();
        cal2.set(2012, cal1.get(Calendar.MONTH), cal1.getActualMaximum
(Calendar.DAY_OF_MONTH), 23, 59, 59);
        db.deleteAnrufe(cal1.getTimeInMillis(), cal2.getTimeInMillis());

        db.closeDBConncetion();

        endzeit[indexed ? 1 : 0] = System.nanoTime();
    }

    public long getTime(int which) {
        return (endzeit[which] - anfangszeit[which]) / 1000 / 1000;
    }

    @Override
    public String toString() {
        return "Fall3 ohne Index: " + getTime(0) + " ms\n" +
               "Fall3  mit Index: " + getTime(1) + " ms";
    }
}
```

```java
package de.hauschil.dbprojekt.anwendungsfaelle;

import java.util.ArrayList;
import java.util.HashSet;

import de.hauschil.dbprojekt.controller.DB_Controller;
import de.hauschil.dbprojekt.controller.Index;
import de.hauschil.dbprojekt.model.Anruf;
import de.hauschil.dbprojekt.model.Kunde;
import de.hauschil.dbprojekt.model.Telefon;

public class Fall4 {
    private long[] anfangszeit;
    private long[] endzeit;
    private DB_Controller db;
    /* Anzahl zu überprüfender Kontakte */
    private final int anz = 1;
    /* Kontakte, die ans Bundeskriminalamt übergeben werden können */
    private HashSet<Kunde> kontakte_anrufer = new HashSet<>();
    private HashSet<Kunde> kontakte_angerufene = new HashSet<>();

    public Fall4(DB_Controller db) {
        this.db = db;
        anfangszeit = new long[2];
        endzeit = new long[2];
    }

    public void run(boolean indexed) {
        db.initDBConnection(new Index[] {
            new Index(Anruf.class, "anrufer", indexed),
            new Index(Anruf.class, "angerufener", indexed)
        });
        /* Alle Kunden mit Telefonnummern holen, um schnell das Mapping von Nummer auf
Kunde hinzubekommen */
        ArrayList<Kunde> help = db.getKunden(null, null);
        /* beziehe zu überprüfende Kunden aus help-Liste */
        Kunde[] kunden = new Kunde[anz];
        for (int i = 0; i < anz; i++) {
            kunden[i] = help.get(i);
        }

        anfangszeit[indexed ? 1 : 0] = System.nanoTime();
        for (Kunde k : kunden) {
            for (Anruf a : getAnrufeFromDb(k, true)) {
                kontakte_angerufene.add(getKundeByNumber(help, a.getAngerufener()));
            }
            for (Anruf a : getAnrufeFromDb(k, false)) {
                kontakte_anrufer.add(getKundeByNumber(help, a.getAnrufer()));
            }
        }

        db.closeDBConncetion();
        endzeit[indexed ? 1 : 0] = System.nanoTime();
    }

    public long getTime(int which) {
        return (endzeit[which] - anfangszeit[which]) / 1000 / 1000;
    }

    @Override
    public String toString() {
```

```
        return "Fall4 ohne Index: " + getTime(0) + " ms\n" +
               "Fall4  mit Index: " + getTime(1) + " ms";
    }

    private Kunde getKundeByNumber(ArrayList<Kunde> kunden, Telefon tel) {
        for (Kunde k : kunden) {
            if (k.getTelefone().contains(tel)) {
                return k;
            }
        }
        return null;
    }

    public ArrayList<Anruf> getAnrufeFromDb(Kunde k, boolean anrufer) {
        ArrayList<Anruf> list = new ArrayList<>();
        if (anrufer) {
            for (Telefon tel : k.getTelefone()) {
                list.addAll(db.getAnrufe(tel, null, null, null));
            }
        } else {
            for (Telefon tel : k.getTelefone()) {
                list.addAll(db.getAnrufe(null, tel, null, null));
            }
        }
        return list;
    }
}
```

```java
package de.hauschil.dbprojekt.model;

import java.util.ArrayList;
import java.util.Random;

import de.hauschil.dbprojekt.controller.Helper;

public class Kunde {
    private static Random r = new Random(42);

    private String nachname;
    private String vorname;

    private ArrayList<Telefon> telefone;

    public Kunde(String vorname, String nachname, ArrayList<Telefon> tel) {
        this.nachname = nachname;
        this.vorname = vorname;
        telefone = tel;
    }

    public static Kunde[] generateKunden(int anzahl) {
        Kunde[] kunden = new Kunde[anzahl];

        /* generiere einen Kunden mit bis zu 5 Telefonen und random Vor- und
Nachnamen*/
        for (int i = 0; i < anzahl; i++) {
            kunden[i] = new Kunde(
                Helper.vornamen[r.nextInt(Helper.vornamen.length)],
                Helper.nachnamen[r.nextInt(Helper.nachnamen.length)],
                Telefon.generateTelefon(r.nextInt(5) + 1)
            );
        }

        return kunden;
    }

    public String getNachname() {
        return nachname;
    }

    public void setNachname(String nachname) {
        this.nachname = nachname;
    }

    public String getVorname() {
        return vorname;
    }

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public ArrayList<Telefon> getTelefone() {
        return telefone;
    }

    public void setTelefone(ArrayList<Telefon> telefone) {
        this.telefone = telefone;
    }
```

```
    @Override
    public String toString() {
        return vorname + " " + nachname + " " + telefone;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                + ((nachname == null) ? 0 : nachname.hashCode());
        result = prime * result
                + ((telefone == null) ? 0 : telefone.hashCode());
        result = prime * result + ((vorname == null) ? 0 : vorname.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Kunde other = (Kunde) obj;
        if (nachname == null) {
            if (other.nachname != null)
                return false;
        } else if (!nachname.equals(other.nachname))
            return false;
        if (telefone == null) {
            if (other.telefone != null)
                return false;
        } else if (!telefone.equals(other.telefone))
            return false;
        if (vorname == null) {
            if (other.vorname != null)
                return false;
        } else if (!vorname.equals(other.vorname))
            return false;
        return true;
    }
}
```

```java
package de.hauschil.dbprojekt.model;

import java.util.ArrayList;
import java.util.Random;

import de.hauschil.dbprojekt.controller.Helper;

public class Telefon {
    private static long base = 132342;
    private static Random r = new Random(42);

    private String nummer;

    public Telefon(String telnummer) {
        nummer = telnummer;
    }

    @Override
    public String toString() {
        return nummer;
    }

    public static ArrayList<Telefon> generateTelefon(int count) {
        ArrayList<Telefon> tels = new ArrayList<>(count);

        for (int i = 0; i < count; i++) {
            tels.add(new Telefon(
                Helper.vorwahlen[r.nextInt(Helper.vorwahlen.length)] + "/" + base++ +
r.nextInt(100)
            ));
        }

        return tels;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((nummer == null) ? 0 : nummer.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Telefon other = (Telefon) obj;
        if (nummer == null) {
            if (other.nummer != null)
                return false;
        } else if (!nummer.equals(other.nummer))
            return false;
        return true;
    }
}
```

```java
package de.hauschil.dbprojekt.model;

import static de.hauschil.dbprojekt.anwendungsfaelle.Fallmanager.ANZ_KUNDEN;
import static de.hauschil.dbprojekt.anwendungsfaelle.Fallmanager.ANZ_PARTNER;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Random;

import de.hauschil.dbprojekt.controller.DB_Controller;

public class Anruf {
    private static Random r = new Random(42);

    private Telefon anrufer;
    private Telefon angerufener;
    private int dauer;
    private long datum;

    public Anruf(Telefon anrufer, Telefon angerufener, long datum, int dauer) {
        this.anrufer = anrufer;
        this.angerufener = angerufener;
        this.datum = datum;
        this.dauer = dauer;
    }

    public Telefon getAnrufer() {
        return anrufer;
    }
    public void setAnrufer(Telefon anrufer) {
        this.anrufer = anrufer;
    }
    public Telefon getAngerufener() {
        return angerufener;
    }
    public void setAngerufener(Telefon angerufener) {
        this.angerufener = angerufener;
    }
    public int getDauer() {
        return dauer;
    }
    public void setDauer(int dauer) {
        this.dauer = dauer;
    }
    public long getDatum() {
        return datum;
    }
    public void setDatum(long datum) {
        this.datum = datum;
    }

    @Override
    public String toString() {
        return "Anruf [anrufer=" + anrufer + ", angerufener=" + angerufener
                + ", dauer=" + dauer + ", datum=" + datum + "]";
    }

    public static void generateAnrufe(Kunde[] kunden, int anzahlProMonat, DB_Controller
db) {
        float fortschritt = 2.0f;
        System.out.println("                      10   20   30   40   50   60   70
```

```
80   90  100");
        System.out.print("Generiere Anrufe: ");
        Anruf[] an = new Anruf[anzahlProMonat * 12];
        Kunde[] partner = new Kunde[ANZ_PARTNER];
        int next_partner;
        int next_partner_telefon;
        int next_telefon;
        /* mache fuer jeden Kunden folgendes */
        for (int i = 0; i < kunden.length; i++) {
            /* Generiere erst die Partner */
            for (int j = 0; j < ANZ_PARTNER; j++) {
                partner[j] = kunden[r.nextInt(kunden.length)];
            }
            /* jeden Monat */
            for (int month = 0; month < 12; month++) {
                /* Nun generiere die Anrufe pro Monat */
                for (int j = 0; j < anzahlProMonat; j++) {
                    next_partner = r.nextInt(partner.length);
                    next_partner_telefon = r.nextInt(partner[next_partner].getTelefone
().size());
                    next_telefon = r.nextInt(kunden[i].getTelefone().size());
                    an[month * anzahlProMonat + j] = new Anruf(
                        kunden[i].getTelefone().get(next_telefon),
                        partner[next_partner].getTelefone().get(next_partner_telefon),
                        generateDate(month),
                        r.nextInt(600)
                    );
                }
            }
            db.storeAnrufe(an);
            if (i % 100 == 99) {
                db.commit();
            }
            if (i % 1000 == 999) {
                db.closePStatement();
            }
            if (((float) i / kunden.length * 100) > fortschritt) {
                fortschritt += 2;
                System.out.print("+");
            }
        }
        db.closePStatement();
        System.out.println();
        System.out.println(anzahlProMonat * 12 * ANZ_KUNDEN + " Anrufe generiert");
    }

    /* Ein random Datum aus 2012 generieren */
    private static long generateDate(int month) {
        GregorianCalendar cal = new GregorianCalendar(2012, month, 1);
        cal.set(Calendar.DAY_OF_MONTH, r.nextInt(cal.getActualMaximum
(Calendar.DAY_OF_MONTH)));
        cal.set(Calendar.HOUR_OF_DAY, r.nextInt(24));
        cal.set(Calendar.MINUTE, r.nextInt(60));
        cal.set(Calendar.SECOND, r.nextInt(60));

        return cal.getTimeInMillis();
    }

    @Override
    public int hashCode() {
        final int prime = 31;
```

```
        int result = 1;
        result = prime * result
                + ((angerufener == null) ? 0 : angerufener.hashCode());
        result = prime * result + ((anrufer == null) ? 0 : anrufer.hashCode());
        result = prime * result + (int) (datum ^ (datum >>> 32));
        result = prime * result + dauer;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Anruf other = (Anruf) obj;
        if (angerufener == null) {
            if (other.angerufener != null)
                return false;
        } else if (!angerufener.equals(other.angerufener))
            return false;
        if (anrufer == null) {
            if (other.anrufer != null)
                return false;
        } else if (!anrufer.equals(other.anrufer))
            return false;
        if (datum != other.datum)
            return false;
        if (dauer != other.dauer)
            return false;
        return true;
    }
}
```

```java
package de.hauschil.dbprojekt.controller;

import java.util.ArrayList;

import de.hauschil.dbprojekt.model.Anruf;
import de.hauschil.dbprojekt.model.Kunde;
import de.hauschil.dbprojekt.model.Telefon;

public interface DB_Controller {
    public void initDBConnection(Index... indizes);
    public void createTables();
    public void dropTables();
    public void commit();
    public void closeDBConncetion();
    public void storeKunden(Kunde[] k);
    public void storeAnrufe(Anruf[] a);
    public void closePStatement();
    public ArrayList<Kunde> getKunden(String vorname, String nachname);
    public ArrayList<Anruf> getAnrufe(Telefon anrufer, Telefon angerufener, Long
datum1, Long datum2);
    public void deleteAnrufe(long datum1, long datum2);
    public Kunde getKundeByNumber(String number);
}
```

```java
package de.hauschil.dbprojekt.controller;

import static de.hauschil.dbprojekt.anwendungsfaelle.Fallmanager.DB4O_PATH;

import java.util.ArrayList;

import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import com.db4o.config.EmbeddedConfiguration;
import com.db4o.config.QueryEvaluationMode;
import com.db4o.query.Constraint;
import com.db4o.query.Query;

import de.hauschil.dbprojekt.model.Anruf;
import de.hauschil.dbprojekt.model.Kunde;
import de.hauschil.dbprojekt.model.Telefon;

public class DB4O_Controller implements DB_Controller {
    private static ObjectContainer db;

    @Override
    public void initDBConnection(Index... indizes) {
        EmbeddedConfiguration conf = Db4oEmbedded.newConfiguration();
        conf.file().blockSize(8);
        conf.common().queries().evaluationMode(QueryEvaluationMode.LAZY);
        for (Index i : indizes) {
            conf.common().objectClass(i.getIndexClass()).objectField(i.getIndexField
()).indexed(i.isIndexed());
        }
        db = Db4oEmbedded.openFile(conf, DB4O_PATH);
    }

    @Override
    public void commit() {
        db.commit();
    }
    @Override
    public void closeDBConncetion() {
        db.close();
    }
    @Override
    public void createTables() {
        /* wird nicht benötigt für db4o */
    }
    @Override
    public void dropTables() {
        /* wird nicht benötigt für db4o */
    }
    @Override
    public void storeKunden(Kunde[] k) {
        db.store(k);
    }
    @Override
    public void storeAnrufe(Anruf[] a) {
        db.store(a);
    }
    @Override
    public void closePStatement() {
        /* wird nicht benötigt für db4o */
    }
```

```
    @Override
    public ArrayList<Kunde> getKunden(String vorname, String nachname) {
        ArrayList<Kunde> k = new ArrayList<>();

        Query query = db.query();
        query.constrain(Kunde.class);
        if (vorname != null) {
            query.descend("vorname").constrain(vorname);
        }
        if (nachname != null) {
            query.descend("nachname").constrain(nachname);
        }

        ObjectSet<Kunde> set = query.execute();
        k.addAll(set);

        return k;
    }

    /* d1 unteres Datum, d2 oberes Datum */
    @Override
    public ArrayList<Anruf> getAnrufe(Telefon anrufer, Telefon angerufener, Long d1,
Long d2) {
        ArrayList<Anruf> list = new ArrayList<>();
        Query query = db.query();
        query.constrain(Anruf.class);
        /* Fall2 */
        if (anrufer != null && angerufener == null && d1 != null && d2 != null) {
            Constraint constraint1 = query.descend("anrufer").constrain(anrufer);
            Constraint constraint2 = query.descend("datum").constrain(d1.longValue
()).greater();
            query.descend("datum").constrain(d2.longValue()).smaller().and
(constraint2).and(constraint1);
        /* Fall3 */
        } else if (anrufer == null && angerufener == null && d1 != null && d2 != null)
{
            Constraint constraint2 = query.descend("datum").constrain(d1.longValue
()).greater();
            query.descend("datum").constrain(d2.longValue()).smaller().and
(constraint2);
        /* Fall 4 */
        } else if (anrufer != null && angerufener == null && d1 == null && d2 == null)
{
            query.descend("anrufer").constrain(anrufer);
        /* Fall 4 */
        } else if (anrufer == null && angerufener != null && d1 == null && d2 == null)
{
            query.descend("angerufener").constrain(angerufener);
        } else {
            throw new RuntimeException("TODO getAnrufe");
        }
        ObjectSet<Anruf> set = query.execute();
        list.addAll(set);

        return list;
    }

    @Override
    public void deleteAnrufe(long datum1, long datum2) {
        ArrayList<Anruf> list = getAnrufe(null, null, datum1, datum2);
```

```
        for (Anruf a : list) {
            db.delete(a);
        }
    }

    @Override
    public Kunde getKundeByNumber(String number) {
        Kunde k;

        Query query = db.query();
        query.constrain(Kunde.class);
        query.descend("telefone").descend("nummer").constrain(number);

        ObjectSet<Kunde> set = query.execute();
        k = set.get(0);

        return k;
    }

    @Override
    public String toString() {
        return "db4o-Controller!";
    }
}
```

```java
package de.hauschil.dbprojekt.controller;

import static de.hauschil.dbprojekt.anwendungsfaelle.Fallmanager.HSQL_PATH;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;

import de.hauschil.dbprojekt.model.Anruf;
import de.hauschil.dbprojekt.model.Kunde;
import de.hauschil.dbprojekt.model.Telefon;

public class HSQL_Controller implements DB_Controller {
    private Connection c = null;
    private PreparedStatement psAnrufe = null;
    private final String CON_STR ="jdbc:hsqldb:file:" + HSQL_PATH + "; shutdown=true";

    @Override
    public void initDBConnection(Index... indizes) {
        try {
            c = DriverManager.getConnection(
                CON_STR,
                "root", "password"
            );
            try (Statement st = c.createStatement()) {
                /* Spezielle Variante für diese Testcases */
                if (indizes.length == 2) {
                    /* Fall1 */
                    if (indizes[0].getIndexClass().equals(Kunde.class)) {
                        if (indizes[0].getIndexField().equals("vorname") && indizes
[1].getIndexField().equals("nachname")) {
                            createSingleIndex(st, indizes);
                            if (indizes[0].isIndexed()) {
                                st.execute("CREATE INDEX idx_name ON Kunde(vorname,
nachname)");
                            } else {
                                st.execute("DROP INDEX idx_name IF EXISTS");
                            }
                        } else {
                            throw new RuntimeException("TODO");
                        }
                    } else if (indizes[0].getIndexClass().equals(Anruf.class)) {
                        /* Fall2 */
                        if (indizes[0].getIndexField().equals("anrufer") && indizes
[1].getIndexField().equals("datum")) {
                            createSingleIndex(st, indizes);
                            if (indizes[0].isIndexed()) {
                                st.execute("CREATE INDEX idx_andat ON Anruf(id_anrufer,
datum)");
                            } else {
                                st.execute("DROP INDEX idx_andat IF EXISTS");
                            }
                        /* Fall4 */
                        } else if (indizes[0].getIndexField().equals("anrufer") &&
indizes[1].getIndexField().equals("angerufener")) {
                            createSingleIndex(st, indizes);
                            if (indizes[0].isIndexed()) {
```

```
                    st.execute("CREATE INDEX idx_anan ON Anruf(id_anrufer,
id_angerufener)");
                } else {
                    st.execute("DROP INDEX idx_anan IF EXISTS");
                }
            } else {
                throw new RuntimeException("TODO");
            }
        } else {
            throw new RuntimeException("TODO");
        }
    } else {
        /* Fall3 */
        createSingleIndex(st, indizes);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}

/* Allgemeine Methode für Einzelspalten-Index */
private void createSingleIndex(Statement st, Index... indizes) throws SQLException
{
    for (Index i : indizes) {
        String sql_start = "";
        String sql_table = "";
        String sql_end = "";

        /* index == false? Dann lösche vorhandenen Index */
        if (!i.isIndexed()) {
            sql_start = "DROP INDEX idx_" + i.getIndexField() + " IF EXISTS";
            st.execute(sql_start);
        /* Ansonsten erstelle einen, abhängig von der Klasse und den Feldern */
        } else {
            sql_start = "CREATE INDEX idx_" + i.getIndexField() + " ON ";
            sql_end = "(" + i.getIndexField() + ")";

            if (i.getIndexClass().equals(Kunde.class)) {
                sql_table = "Kunde";
            } else if (i.getIndexClass().equals(Anruf.class)) {
                sql_table = "Anruf";
                if (i.getIndexField().equals("anrufer") || i.getIndexField().equals
("angerufener")) {
                    sql_end = "(id_" + i.getIndexField() + ")";
                }
            } else {
                throw new RuntimeException("TODO");
            }

            st.execute(sql_start + sql_table + sql_end);
        }
    }
}

@Override
public void createTables() {
    try (Statement stmt = c.createStatement()) {
        /* setzte Größe für Tabellen im Ram auf 1 GiB */
        stmt.execute("SET FILES CACHE SIZE 1000000"); //kiB
    } catch (SQLException e) {
```

```
            e.printStackTrace();
        }

        try (Statement stmt = c.createStatement()) {
            /* Erzeuge Tabelle Kunde */
            stmt.execute(
                "CREATE TABLE IF NOT EXISTS Kunde (" +
                "id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY," +
                "vorname varchar(100) NOT NULL," +
                "nachname varchar(100) NOT NULL," +
                ")"
            );
            /* erzeuge Tabelle Telefon */
            /* Foreign keys auskommentiert, weil sonst automatisch Indizes angelegt
werden (eig ja gut, aber nicht für diese Aufgabe ;) ) */
            stmt.execute(
                "CREATE TABLE IF NOT EXISTS Telefon (" +
                "nummer varchar(15) PRIMARY KEY," +
                "id_kunde INTEGER NOT NULL," +
                /* auskommentiert, damit nicht automatisch ein Index erzeugt wird */
//              "FOREIGN KEY (id_kunde) REFERENCES Kunde(id)" +
                ")"
            );
            /* erzeuge Tabelle Anruf */
            stmt.execute(
                "CREATE CACHED TABLE IF NOT EXISTS Anruf (" +
                "id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY," +
                "dauer INTEGER NOT NULL," +
                "datum BIGINT NOT NULL," +
                "id_anrufer varchar(15) NOT NULL," +
                "id_angerufener varchar(15) NOT NULL," +
                /* auskommentiert, damit nicht automatisch ein Index erzeugt wird */
//              "FOREIGN KEY (id_anrufer) REFERENCES Telefon(id)," +
//              "FOREIGN KEY (id_angerufener) REFERENCES Telefon(id)" +
                ")"
            );
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }

    @Override
    public void dropTables() {
        try (Statement stmt = c.createStatement()) {
            stmt.execute("DROP TABLE IF EXISTS Anruf");
            stmt.execute("DROP TABLE IF EXISTS Telefon");
            stmt.execute("DROP TABLE IF EXISTS Kunde");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void storeKunden(Kunde[] kunden) {
        try (PreparedStatement ps = c.prepareStatement(
            "INSERT INTO Kunde (vorname, nachname) VALUES (?, ?)"
        )) {
            for (Kunde k : kunden) {
                ps.setString(1, k.getVorname());
                ps.setString(2, k.getNachname());
```

```
            ps.execute();
            try (PreparedStatement ps2 = c.prepareStatement(
                "INSERT INTO Telefon (nummer, id_kunde)" +
                "VALUES (?, (SELECT TOP 1 id FROM Kunde ORDER BY id DESC))"
            )) {
                for (Telefon t : k.getTelefone()) {
                    ps2.setString(1, t.toString());
                    ps2.addBatch();
                }
                ps2.executeBatch();
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public void storeAnrufe(Anruf[] anrufe) {
    if (psAnrufe == null) {
        try {
            c.setAutoCommit(false);
            psAnrufe = c.prepareStatement(
                "INSERT INTO Anruf (dauer, datum, id_anrufer, id_angerufener)" +
                "VALUES (?, ?, ?, ?)"
            );
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    try {
        for (Anruf a : anrufe) {
            psAnrufe.setInt(1, a.getDauer());
            psAnrufe.setLong(2, a.getDatum());
            psAnrufe.setString(3, a.getAnrufer().toString());
            psAnrufe.setString(4, a.getAngerufener().toString());
            psAnrufe.addBatch();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public void commit() {
    if (psAnrufe != null) {
        try {
            psAnrufe.executeBatch();
            c.commit();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

@Override
public void closePStatement() {
    if (psAnrufe != null) {
        try {
            psAnrufe.close();
            c.setAutoCommit(true);
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                psAnrufe = null;
            }
        }
    }

    @Override
    public void closeDBConncetion() {
        if (c == null) {
            return;
        }

        try {
            c.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public ArrayList<Kunde> getKunden(String vorname, String nachname) {
        ArrayList<Kunde> k = new ArrayList<Kunde>();
        try (Statement st = c.createStatement()) {
            ResultSet rs;
            if (vorname != null && nachname != null) {
                rs = st.executeQuery(
                    "SELECT id, vorname, nachname FROM Kunde " +
                    "WHERE vorname = '" + vorname + "' AND nachname = '" + nachname +
"' " +
                    "ORDER BY nachname ASC, vorname ASC"
                );
            } else if (vorname != null) {
                rs = st.executeQuery(
                    "SELECT id, vorname, nachname FROM Kunde " +
                    "WHERE vorname = '" + vorname + "'" +
                    "ORDER BY nachname ASC, vorname ASC"
                );
            } else if (nachname != null) {
                rs = st.executeQuery(
                    "SELECT id, vorname, nachname FROM Kunde " +
                    "WHERE nachname = '" + nachname + "' " +
                    "ORDER BY nachname ASC, vorname ASC"
                );
            /* beides null */
            } else {
                rs = st.executeQuery(
                    "SELECT id, vorname, nachname FROM Kunde " +
                    "ORDER BY nachname ASC, vorname ASC"
                );
            }

            while (rs.next()) {
                k.add(new Kunde(rs.getString(2), rs.getString(3), getTelefoneWithKID
(rs.getInt(1))));
            }
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return k;
    }

    private ArrayList<Telefon> getTelefoneWithKID(int k_id) {
        ArrayList<Telefon> t = new ArrayList<>();

        try (Statement st = c.createStatement()) {
            ResultSet rs = st.executeQuery(
                "SELECT nummer FROM Telefon WHERE id_kunde = '" + k_id + "'"
            );
            while (rs.next()) {
                t.add(new Telefon(rs.getString(1)));
            }
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return t;
    }

    /* d1 untere Grenze, d2 obere Grenze */
    @Override
    public ArrayList<Anruf> getAnrufe(Telefon anrufer, Telefon angerufener, Long d1,
Long d2) {
        ArrayList<Anruf> list = new ArrayList<>();
        /* Fall2 */
        try (Statement stmt = c.createStatement()) {
            ResultSet rs = null;
            if (anrufer != null && angerufener == null && d1 != null && d2 != null) {
                rs = stmt.executeQuery(
                    "SELECT a.id_anrufer, a.id_angerufener, dauer, datum " +
                    "FROM Anruf a " +
                    "WHERE a.id_anrufer = '" + anrufer.toString() + "' " +
                    "AND a.datum > " + d1.longValue() +" " +
                    "AND a.datum < " + d2.longValue()
                );
            /* Fall4 */
            } else if (anrufer != null && angerufener == null && d1 == null && d2 ==
null) {
                rs = stmt.executeQuery(
                        "SELECT a.id_anrufer as Anrufer,  a.id_angerufener as
Angerufener, dauer, datum " +
                        "FROM Anruf a " +
                        "WHERE a.id_anrufer = '" + anrufer.toString() + "'"
                    );
            } else if (anrufer == null && angerufener != null && d1 == null && d2 ==
null) {
                rs = stmt.executeQuery(
                        "SELECT a.id_anrufer as Anrufer, a.id_angerufener as
Angerufener, dauer, datum " +
                        "FROM Anruf a " +
                        "WHERE a.id_angerufener = '" + angerufener.toString() + "'"
                    );
            } else {
                throw new RuntimeException("TODO");
            }
            while (rs.next()) {
                if (anrufer != null) {
                    list.add(new Anruf(anrufer, new Telefon(rs.getString(2)),
```

```
rs.getLong(4), rs.getInt(3)));
                } else {
                    list.add(new Anruf(new Telefon(rs.getString(1)), angerufener,
rs.getLong(4), rs.getInt(3)));
                }
            }
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();

        }
        return list;
    }

    @Override
    public void deleteAnrufe(long datum1, long datum2) {
        try (Statement stmt = c.createStatement()) {
            stmt.execute(
                "DELETE FROM Anruf WHERE datum > " + datum1 + " AND datum < " + datum2
            );
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public Kunde getKundeByNumber(String number) {
        Kunde k = null;
        try (Statement st = c.createStatement()) {
            ResultSet rs = st.executeQuery(
                "SELECT vorname, nachname FROM Kunde k, Telefon t " +
                "WHERE k.id = t.id_kunde AND t.nummer = '" + number + "'"
            );
            rs.next();
            k = new Kunde(rs.getString(1), rs.getString(2), null);
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return k;
    }

    @Override
    public String toString() {
        return "HSQL-Controller!";
    }
}
```

```java
package de.hauschil.dbprojekt.controller;

public class Index {
    private Class<?> indexClass;
    private String indexField;
    private boolean indexed;

    public Index(Class<?> indexClass, String indexField, boolean indexed) {
        super();
        this.indexClass = indexClass;
        this.indexField = indexField;
        this.indexed = indexed;
    }

    public Class<?> getIndexClass() {
        return indexClass;
    }
    public void setIndexClass(Class<?> indexClass) {
        this.indexClass = indexClass;
    }
    public String getIndexField() {
        return indexField;
    }
    public void setIndexField(String indexField) {
        this.indexField = indexField;
    }
    public boolean isIndexed() {
        return indexed;
    }
    public void setIndexed(boolean indexed) {
        this.indexed = indexed;
    }
}
```

```java
package de.hauschil.dbprojekt.controller;

public class Helper {
    public static String[] vornamen = {
        "Agnes", "Agnieszka", "Alexandra", "Alina", "Alma", "Amelie", "Andrea",
        "Anett", "Anette", "Angela", "Angelica", "Angelika", "Anica", "Anika",
        "Anita", "Anja", "Anke", "Ann", "Anna", "Anne", "Anneliese",
        "Annemarie", "Annett", "Annette", "Anni", "Annica", "Annie", "Annika",
        "Anny", "Antje", "Antonia", "Ariane", "Astrid", "Auguste", "Ayse",
        "Bärbel", "Barbara", "Beata", "Beate", "Beatrice", "Berit", "Berta",
        "Bertha", "Bettina", "Bianca", "Bianka", "Birgit", "Birgitt", "Birte",
        "Birthe", "Brigitte", "Britta", "Caren", "Carina", "Carla", "Carmen",
        "Carola", "Carolin", "Caroline", "Catarina", "Catharina", "Cathleen",
"Cathrin",
        "Catrin", "Celina", "Charlotte", "Christa", "Christel", "Christiane",
"Christin",
        "Christina", "Christine", "Cindy", "Clara", "Claudia", "Constanze", "Cordula",
        "Corinna", "Cornelia", "Dörte", "Dörthe", "Dagmar", "Dana", "Daniela",
        "Denise", "Diana", "Dora", "Doreen", "Doris", "Dorothea", "Dorothee",
        "Edith", "Elena", "Elfriede", "Elisabeth", "Elise", "Elke", "Ella",
        "Elli", "Elly", "Elsa", "Else", "Emilia", "Emilie", "Emily",
        "Emine", "Emma", "Erika", "Erna", "Esther", "Eva", "Evelin",
        "Eveline", "Evelyn", "Evelyne", "Ewa", "Fatma", "Filiz", "Franziska",
        "Frauke", "Frida", "Frieda", "Friederike", "Gabi", "Gabriela", "Gabriele",
        "Gaby", "Gerda", "Gertrud", "Gesa", "Gisela", "Grit", "Hanna",
        "Hannah", "Hannelore", "Hatice", "Hedwig", "Heidi", "Heike", "Helena",
        "Helene", "Helga", "Henni", "Henny", "Herta", "Hertha", "Hildegard",
        "Ida", "Ilka", "Ilona", "Ilse", "Imke", "Ina", "Ines",
        "Inga", "Inge", "Ingeborg", "Ingrid", "Irene", "Irina", "Iris",
        "Irma", "Irmgard", "Isabel", "Isabell", "Isabella", "Isabelle", "Ivonne",
        "Jacqueline", "Jana", "Janet", "Janett", "Janette", "Janin", "Janina",
        "Janine", "Jaqueline", "Jasmin", "Jeanette", "Jeannette", "Jennifer", "Jenny",
        "Jessica", "Jessika", "Joanna", "Johanna", "Judith", "Julia", "Juliane",
        "Jutta", "Käte", "Käthe", "Karen", "Karin", "Karina", "Karla",
        "Karola", "Karolin", "Karoline", "Katarina", "Katarzyna", "Katharina",
"Kathi",
        "Kathie", "Kathleen", "Kathrin", "Kati", "Katie", "Katja", "Katrin",
        "Kerstin", "Kim", "Kirsten", "Kirstin", "Klara", "Klaudia", "Konstanze",
        "Kordula", "Korinna", "Kornelia", "Kristiane", "Kristin", "Kristina",
"Kristine",
        "Lara", "Larissa", "Laura", "Lea", "Leah", "Lena", "Leni",
        "Leoni", "Leonie", "Lidia", "Lieselotte", "Lili", "Lilli", "Lilly",
        "Lina", "Linda", "Lisa", "Liselotte", "Lotte", "Louisa", "Louise",
        "Luisa", "Luise", "Lydia", "Magdalena", "Maike", "Maja", "Malgorzata",
        "Mandy", "Manja", "Manuela", "Mareike", "Maren", "Margarete", "Margarethe",
        "Margot", "Margrit", "Maria", "Marianne", "Marie", "Marina", "Marion",
        "Marta", "Martha", "Martina", "Maya", "Meike", "Melanie", "Melina",
        "Melissa", "Meta", "Metha", "Mia", "Michaela", "Michelle", "Minna",
        "Miriam", "Mirja", "Mirjam", "Monika", "Monique", "Nadin", "Nadine",
        "Nadja", "Nancy", "Natalia", "Natalie", "Natascha", "Nathalie", "Neele",
        "Nele", "Nicola", "Nicole", "Nikola", "Nina", "Olga", "Pamela",
        "Patricia", "Patrizia", "Paula", "Peggy", "Petra", "Ramona", "Rebecca",
        "Rebekka", "Regina", "Renate", "Rita", "Rosemarie", "Ruth", "Sabine",
        "Sabrina", "Sandra", "Sara", "Sarah", "Saskia", "Sibylle", "Sigrid",
        "Silke", "Silvia", "Simone", "Sina", "Sinah", "Sofia", "Sofie",
        "Sonja", "Sophia", "Sophie", "Stefanie", "Steffi", "Stephanie", "Susan",
        "Susann", "Susanne", "Svantje", "Svenja", "Svetlana", "Swantje", "Swenja",
        "Swetlana", "Sybille", "Sylke", "Sylvia", "Sylwia", "Tamara", "Tania",
        "Tanja", "Tatjana", "Tina", "Ulrike", "Ursula", "Uta", "Ute",
        "Vanessa", "Vera", "Verena", "Veronica", "Veronika", "Victoria", "Viktoria",
        "Viola", "Waltraud", "Waltraut", "Wera", "Wibke", "Wiebke", "Wilhelmine",
```

```
        "Yasemin", "Yasmin", "Yvonne",

        "Adolf", "Albert", "Alexander", "Alfred", "André", "Andre", "Andreas",
        "Arthur", "Artur", "August", "Axel", "Ben", "Benjamin", "Bernd",
        "Björn", "Bruno", "Carl", "Carsten", "Christian", "Christoph", "Claus",
        "Curt", "Daniel", "David", "Dennis", "Dieter", "Dirk", "Dominic",
        "Dominik", "Elias", "Emil", "Eric", "Erich", "Erik", "Ernst",
        "Erwin", "Fabian", "Felix", "Finn", "Florian", "Frank", "Franz",
        "Friedrich", "Fritz", "Fynn", "Günter", "Günther", "Georg", "Gerd",
        "Gerhard", "Gert", "Gustav", "Hans", "Harald", "Harri", "Harry",
        "Heinrich", "Heinz", "Hellmut", "Helmut", "Helmuth", "Herbert", "Hermann",
        "Holger", "Horst", "Hugo", "Ingo", "Jörg", "Jürgen", "Jacob",
        "Jakob", "Jan", "Jannik", "Jens", "Joachim", "Johann", "Johannes",
        "Jonas", "Jonathan", "Josef", "Joseph", "Julian", "Justin", "Kai",
        "Karl", "Karl-Heinz", "Karlheinz", "Karsten", "Kay", "Kevin", "Klaus",
        "Kristian", "Kurt", "Lars", "Lennard", "Lennart", "Leon", "Lothar",
        "Louis", "Luca", "Lucas", "Ludwig", "Luis", "Luka", "Lukas",
        "Lutz", "Maik", "Manfred", "Marc", "Marcel", "Marco", "Marcus",
        "Mario", "Mark", "Marko", "Markus", "Martin", "Marvin", "Mathias",
        "Matthias", "Max", "Maximilian", "Meik", "Michael", "Mike", "Moritz",
        "Nick", "Niclas", "Nico", "Niels", "Niklas", "Niko", "Nils",
        "Noah", "Norbert", "Olaf", "Ole", "Oliver", "Oscar", "Oskar",
        "Otto", "Patrick", "Paul", "Peter", "Philip", "Philipp", "Phillipp",
        "Rainer", "Ralf", "Ralph", "Reiner", "René", "Richard", "Robert",
        "Rolf", "Rudolf", "Rudolph", "Sascha", "Sebastian", "Siegfried", "Simon",
        "Stefan", "Steffen", "Stephan", "Sven", "Swen", "Thomas", "Thorsten",
        "Tim", "Timm", "Tobias", "Tom", "Torsten", "Ulrich", "Uwe",
        "Volker", "Walter", "Walther", "Werner", "Wilhelm", "Willi", "Willy",
        "Wolfgang", "Yannic", "Yannick", "Yannik"
    };

    public static String[] nachnamen = {
        "Müller", "Schmidt", "Schneider", "Fischer", "Weber", "Meyer", "Wagner",
        "Becker", "Schulz", "Hoffmann", "Schäfer", "Bauer", "Koch", "Richter",
        "Klein", "Wolf", "Schröder", "Neumann", "Schwarz", "Braun", "Hofmann",
        "Zimmermann", "Schmitt", "Hartmann", "Krüger", "Schmid", "Werner", "Lange",
        "Schmitz", "Meier", "Krause", "Maier", "Lehmann", "Huber", "Mayer",
        "Herrmann", "Köhler", "Walter", "König", "Schulze", "Fuchs", "Kaiser",
        "Lang", "Weiß", "Peters", "Scholz", "Jung", "Möller", "Hahn",
        "Keller", "Vogel", "Schubert", "Roth", "Frank", "Friedrich", "Beck",
        "Günther", "Berger", "Winkler", "Lorenz", "Baumann", "Schuster", "Kraus",
        "Böhm", "Simon", "Franke", "Albrecht", "Winter", "Ludwig", "Martin",
        "Krämer", "Schumacher", "Vogt", "Jäger", "Stein", "Otto", "Groß",
        "Sommer", "Haas", "Graf", "Heinrich", "Seidel", "Schreiber", "Ziegler",
        "Brandt", "Kuhn", "Schulte", "Dietrich", "Kühn", "Engel", "Pohl",
        "Horn", "Sauer", "Arnold", "Thomas", "Bergmann", "Busch", "Pfeiffer",
        "Voigt", "Götz", "Seifert", "Lindner", "Ernst", "Hübner", "Kramer",
        "Franz", "Beyer", "Wolff", "Peter", "Jansen", "Kern", "Barth",
        "Wenzel", "Hermann", "Ott", "Paul", "Riedel", "Wilhelm", "Hansen",
        "Nagel", "Grimm", "Lenz", "Ritter", "Bock", "Langer", "Kaufmann",
        "Mohr", "Förster", "Zimmer", "Haase", "Lutz", "Kruse", "Jahn",
        "Schumann", "Fiedler", "Thiel", "Hoppe", "Kraft", "Michel", "Marx",
        "Fritz", "Arndt", "Eckert", "Schütz", "Walther", "Petersen", "Berg",
        "Schindler", "Kunz", "Reuter", "Sander", "Schilling", "Reinhardt", "Frey",
        "Ebert", "Böttcher", "Thiele", "Gruber", "Schramm", "Hein", "Bayer",
        "Fröhlich", "Voß", "Herzog", "Hesse", "Maurer", "Rudolph", "Nowak",
        "Geiger", "Beckmann", "Kunze", "Seitz", "Stephan", "Büttner", "Bender",
        "Gärtner", "Bachmann", "Behrens", "Scherer", "Adam", "Stahl", "Steiner",
        "Kurz", "Dietz", "Brunner", "Witt", "Moser", "Fink", "Ullrich",
        "Kirchner", "Löffler", "Heinz", "Schultz", "Ulrich", "Reichert", "Schwab",
        "Breuer", "Gerlach", "Brinkmann", "Göbel", "Blum", "Brand", "Naumann",
```

```
        "Stark", "Wirth", "Schenk", "Binder", "Körner", "Schlüter", "Rieger",
        "Urban", "Böhme", "Jakob", "Schröter", "Krebs", "Wegner", "Heller",
        "Kopp", "Link", "Wittmann", "Unger", "Reimann", "Ackermann", "Hirsch",
        "Schiller", "Döring", "May", "Bruns", "Wendt", "Wolter", "Menzel",
        "Pfeifer", "Sturm", "Buchholz", "Rose", "Meißner", "Janssen", "Bach",
        "Engelhardt", "Bischoff", "Bartsch", "John", "Kohl", "Kolb", "Münch",
        "Vetter", "Hildebrandt", "Renner", "Weiss", "Kiefer", "Rau", "Hinz",
        "Wilke", "Gebhardt", "Siebert", "Baier", "Köster", "Rohde", "Will",
        "Fricke", "Freitag", "Nickel", "Reich", "Funk", "Linke", "Keil",
        "Hennig", "Witte", "Stoll", "Schreiner", "Held", "Noack", "Brückner",
        "Decker", "Neubauer", "Westphal", "Heinze", "Baum", "Schön", "Wimmer",
        "Marquardt", "Stadler", "Schlegel", "Kremer", "Ahrens", "Hammer", "Röder",
        "Pieper", "Kirsch", "Fuhrmann", "Henning", "Krug", "Popp", "Conrad",
        "Karl", "Krieger", "Mann", "Wiedemann", "Lemke", "Erdmann", "Mertens",
        "Heß", "Esser", "Hanke", "Strauß", "Miller", "Berndt", "Konrad",
        "Jacob", "Philipp", "Metzger", "Henke", "Wiese", "Hauser", "Dittrich",
        "Albert", "Klose", "Bader", "Herbst", "Henkel", "Kröger", "Wahl",
        "Bartels", "Harms", "Fritsch", "Adler", "Großmann", "Burger", "Schrader",
        "Probst", "Martens", "Baur", "Burkhardt", "Hess", "Mayr", "Nolte",
        "Heine", "Kuhlmann", "Klaus", "Kühne", "Kluge", "Bernhardt", "Blank",
        "Hamann", "Steffen", "Brenner", "Rauch", "Reiter", "Preuß", "Jost",
        "Wild", "Hummel", "Beier", "Krauß", "Lindemann", "Herold", "Christ",
        "Niemann", "Funke", "Haupt", "Janßen", "Vollmer", "Straub", "Strobel",
        "Wiegand", "Merz", "Haag", "Holz", "Knoll", "Zander", "Rausch",
        "Bode", "Beer", "Betz", "Anders", "Wetzel", "Hartung", "Glaser",
        "Fleischer", "Rupp", "Reichel", "Lohmann", "Diehl", "Jordan", "Eder",
        "Rothe", "Weis", "Heinemann", "Dörr", "Metz", "Kroll", "Freund",
        "Wegener", "Hohmann", "Geißler", "Schüler", "Schade", "Raab", "Feldmann",
        "Zeller", "Neubert", "Rapp", "Kessler", "Heck", "Meister", "Stock",
        "Römer", "Seiler", "Altmann", "Behrendt", "Jacobs", "Mai", "Bär",
        "Wunderlich", "Schütte", "Lauer", "Benz", "Weise", "Völker", "Sonntag",
        "Bühler", "Gerber", "Kellner", "Bittner", "Schweizer", "Keßler", "Hagen",
        "Wieland", "Born", "Merkel", "Falk", "Busse", "Gross", "Eichhorn",
        "Greiner", "Moritz", "Forster", "Stumpf", "Seidl", "Scharf", "Hentschel",
        "Buck", "Voss", "Hartwig", "Heil", "Eberhardt", "Oswald", "Lechner",
        "Block", "Heim", "Steffens", "Weigel", "Pietsch", "Brandl", "Schott",
        "Gottschalk", "Bertram", "Ehlers", "Fleischmann", "Albers", "Weidner",
"Hiller",
        "Seeger", "Geyer", "Jürgens", "Baumgartner", "Mack", "Schuler", "Appel",
        "Pape", "Dorn", "Wulf", "Opitz", "Wiesner", "Hecht", "Moll",
        "Gabriel", "Auer", "Engelmann", "Singer", "Neuhaus", "Giese", "Schütze",
        "Geisler", "Ruf", "Heuer", "Noll", "Scheffler", "Sauter", "Reimer",
        "Klemm", "Schaller", "Hempel", "Kretschmer", "Runge", "Springer", "Riedl",
        "Steinbach", "Michels", "Barthel"
    };

    public static String[] vorwahlen = {
        "01511", "01512", "01514", "01515", "01516", "0160", "01609",
        "0170", "0171", "0175", "01520", "01522", "01523", "01525",
        "0162", "0172", "0173", "0174", "01521", "01529", "01573",
        "01575", "01577", "01578", "0163", "0177", "0178", "01570",
        "01579", "01590", "0176", "0179"
    };

}
```