TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Alexander van Renen (renen@in.tum.de)
http://db.in.tum.de/teaching/ss20/ei2/

Lösungen zu Blatt 8

Tool zum Üben der relationalen Algebra: https://tools.db.in.tum.de/ira/.
SQL-Schnittstelle: http://hyper-db.de/interface.html.

Aufgabe 1: Hashing mit Linear Chaining

Implementieren Sie in Java eine Hashtabelle die "linear chaining" zur Kollisionsbehandlung verwendet: Jede Position in der Hashtabelle speichert eine Liste von Elementen mit diesem Hashwert. Bonus: Implementieren Sie einen Mechanismus um die Hashtabelle wachsen zu lassen.

Lösung 1

Die folgende Klasse implementiert eine Hashtabelle mit linear chaining.

```java
import java.util.ArrayList;
import java.util.List;

public class MyHashTable<T> {
    private static final float maxLoadFactor = 0.75f;
    private int numBuckets; // initial number of buckets; usually a power of 2
    private int size = 0; // number of elements currently saved in hashtable
    private ArrayList<ArrayList<T>> buckets;

    MyHashTable() {
        this(2);
    }

    MyHashTable(int numBuckets) {
        this.numBuckets = numBuckets;
        this.buckets = new ArrayList<>(numBuckets); // (optional) specify initial capacity for no reallocations
        for (int i = 0; i < numBuckets; i++) { // fill array with empty lists
            this.buckets.add(new ArrayList<>());
        }
    }

    public void add(T value) {
        // Add to appropriate bucket
```

```
24          int hash = calculateBucketOffset(value);
25          List<T> bucket = buckets.get(hash);
26          for (T key : bucket) {
27              if (key == value) { // Avoid duplicates
28                  return;
29              }
30          }
31          bucket.add(value);
32          size++;
33
34          // Rehash ?
35          if ((1.0f * size) / numBuckets >= maxLoadFactor) {
36              rehash(numBuckets * 2);
37          }
38      }
39
40      public void remove(T value) {
41          int hash = calculateBucketOffset(value);
42          List<T> bucket = buckets.get(hash);
43
44          for (int i = 0; i < bucket.size(); i++) {
45              if (bucket.get(i) == value) {
46                  size--;
47                  bucket.remove(i);
48                  return;
49              }
50          }
51      }
52
53      private int calculateBucketOffset(T value) {
54          return Math.abs(value.hashCode() % numBuckets); // save hash,
                absolute value for negative hashCode
55      }
56
57      private void rehash(int newNumBuckets) {
58          MyHashTable<T> largerHashTable = new MyHashTable<>(
               newNumBuckets);
59          for (List<T> bucket : buckets) {
60              for (T value : bucket) {
61                  largerHashTable.add(value);
62              }
63          }
64          numBuckets = newNumBuckets;
65          buckets = largerHashTable.buckets;
66      }
67  }
```

Aufgabe 2: Relationale Modellierung

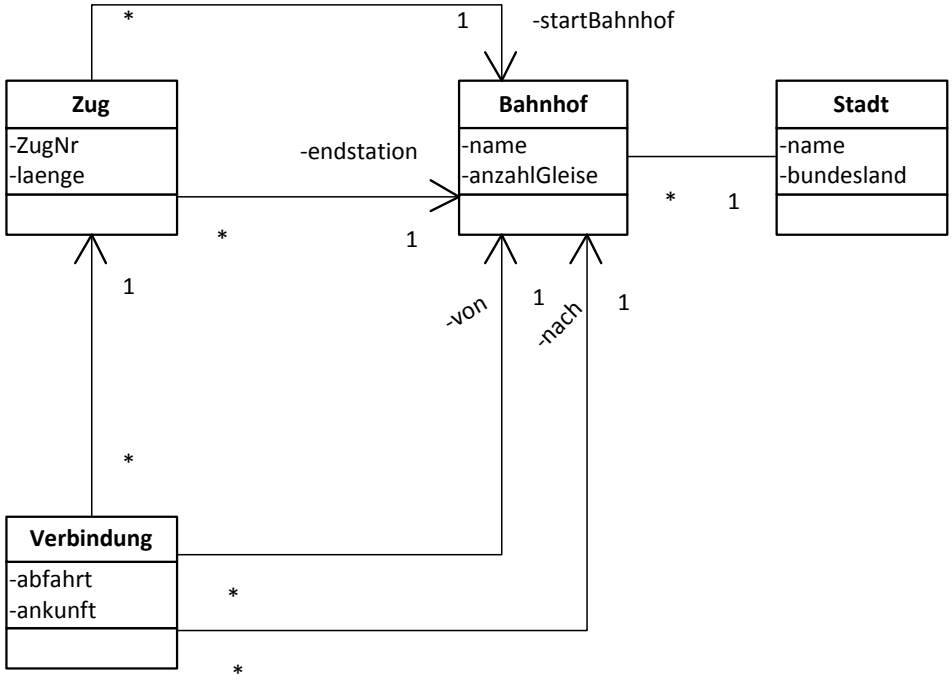

Abbildung 1: UML Modell von Zugverbindungen

Setzen Sie das UML-Modell von Zugverbindungen aus Abbildung 1 in ein relationales Modell um. Wandeln Sie dazu zunächst die Klassen mit ihren Attributen in Relationen um. Anschließend beachten Sie noch die Beziehungen, die Sie entweder mit zusätzlichen Attributen (Spalten) in den bestehenden Relationen umsetzen können oder aber mit zusätzlichen Relationen. Zuletzt überlegen Sie sich, welche Attribute jeweils einen Primärschlüssel für die Relationen darstellen und unterstreichen diese.

Lösung 2

Zuerst modellieren wir die Klassen als Relationen und bestimmen passende Schlüssel. Im Fall der Verbindung müssen wir einen künstlichen Schlüssel einführen, da nicht einmal die Kombination aus Abfahrt und Ankunft eindeutig ist:

$$\textbf{Züge} : \{[\underline{\text{ZugNr: integer}}, \text{Laenge: integer}]\} \quad (1)$$

$$\textbf{Bahnhöfe} : \{[\underline{\text{Name: string}}, \text{AnzahlGleise: integer}]\} \quad (2)$$

$$\textbf{Städte} : \{[\underline{\text{Name: string, Bundesland: string}}]\} \quad (3)$$

$$\textbf{Verbindung} : \{[\underline{\text{VerbindungsNr: integer}}, \text{Abfahrt: date, Ankunft: date}]\} \quad (4)$$

Nun kümmern wir uns um die Beziehungen, die wir in einem ersten Schritt alle als eigenständige Relationen modellieren:

liegtIn	:	{[Bahnhof: string, Stadt: string, Bundesland: string]}	(5)
startetIn	:	{[ZugNr: integer, Bahnhof: string]}	(6)
endetIn	:	{[ZugNr: integer, Bahnhof: string]}	(7)
verbindetVon	:	{[VerbindungsNr: integer, Bahnhof: string]}	(8)
verbindetNach	:	{[VerbindungsNr: integer, Bahnhof: string]}	(9)
verbindetMit	:	{[VerbindungsNr: integer, Zug: integer]}	(10)

Als letztes verfeinern wir das relationale Schema, indem wir Relationen zusammenfassen. Dabei werden Relationen für Beziehungen mit Relationen für Klassen zusammengefasst, falls diese den gleichen Schlüssel haben und es sich dabei um eine 1:N, N:1 oder 1:1 Beziehung handelt.

So kann Relation (5) in (2) aufgenommen werden. (6) und (7) werden mit (1) zusammengefasst. Und (8), (9) und (10) werden alle mit (4) zusammengefasst. Das ergibt folgendes Schema:

Züge	:	{[ZugNr: integer, Laenge: integer, StartBahnhof: string, ZielBahnhof: string]}
Bahnhöfe	:	{[Name: string, AnzahlGleise: integer, Stadt: string, Bundesland: string]}
Städte	:	{[Name: string, Bundesland: string]}
Verbindung	:	{[VerbindungsNr: integer, Abfahrt: date, Ankunft: date, VonBahnhof: string, NachBahnhof: string, ZugNr: integer]}

Aufgabe 3: Relationenalgebra

Formulieren Sie die folgenden Anfragen auf dem Universitätsschema in Relationenalgebra.

Lösung 3

(a) Geben Sie alle *Vorlesungen* an, die der *Student* Xenokrates gehört hat.

$$R := \Pi_{\text{VorlNr, Titel}}(\text{Vorlesungen} \bowtie (\text{hören} \bowtie (\sigma_{\text{Name='Xenokrates'}}(\text{Studenten}))))$$

(b) Geben Sie die Titel der direkten Voraussetzungen für die *Vorlesung* Wissenschaftstheorie an.

$$R := \Pi_{\text{v2.Titel}}(\rho_{\text{v2}}(\text{Vorlesungen}) \bowtie_{\text{v2.VorlNr=Vorgänger}} (\text{voraussetzen} \bowtie_{\text{v1.VorlNr=Nachfolger}} (\sigma_{\text{v1.Titel='Wissenschaftstheorie'}}(\rho_{\text{v1}}(\text{Vorlesungen})))))$$

Aufgabe 4: SQL

Formulieren Sie folgende Anfragen auf dem Universitätsschema in SQL.

Lösung 4

(a) Finden Sie die *Studenten*, die Sokrates aus *Vorlesung(en)* kennen.

```
select s.Name, s.MatrNr
from Studenten s, hoeren h, Vorlesungen v, Professoren p
where s.MatrNr = h.MatrNr
and h.VorlNr = v.VorlNr
and v.gelesenVon = p.PersNr
and p.Name = 'Sokrates';
```

`DISTINCT` wäre nett, um Duplikate zu unterdrücken ist aber nicht explizit in der Aufgabe gefordert.

(b) Finden Sie die *Assistenten* von *Professoren*, die den Studenten Fichte unterrichtet haben – z.B. als potentielle Betreuer seiner Diplomarbeit.

```
select a.Name, a.PersNr
from Assistenten a, Vorlesungen v, hoeren h, Studenten s
where a.Boss = v.gelesenVon
and v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
and s.Name = 'Fichte';
```

(c) Geben Sie die Namen der *Professoren* an, die Xenokrates aus *Vorlesungen* kennt.

```
select p.PersNr, p.Name
from Professoren p, hoeren h, Vorlesungen v, Studenten s
where p.PersNr = v.gelesenVon
and v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
and s.Name = 'Xenokrates';
```

(d) Welche *Vorlesungen* werden von *Studenten* im Grundstudium (1.-4. Semester) gehört? Geben Sie die Titel dieser *Vorlesungen* an.

```
select v.Titel
from Vorlesungen v, hoeren h, Studenten s
where v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
and s.Semester between 1 and 4;
```