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

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

Christoph Anneser (anneser@in.tum.de)
`http://db.in.tum.de/teaching/ss23/ei2/`

Blatt Nr. 8
Dieses Blatt wird am Montag, den 26.06.2023 besprochen.

Aufgabe 1: Hashing mit Linear Chaining

Sie finden die Lösung unter `https://gitlab.db.in.tum.de/DropTableGrades/hashset` unter dem Branch *solution*.

Aufgabe 2: Hashing mit Linear Probing

Veranschaulichen Sie Hashing mit Linear Probing.

Die Größe der Hashtabelle ist dabei jeweils $m = 13$. Führen Sie die folgenden Operationen aus:

`insert`	$16, 3, 12, 17, 29, 10, 24$
`delete`	16
`insert`	$5, 1, 15$
`delete`	10
`insert`	14
`delete`	1

Verwenden Sie die Hashfunktion

$$h(x) = 3x \bmod 13.$$

Bei dieser Aufgabe sind die Schlüssel der Elemente die Elemente selbst.

Beim **Löschen** soll die Wiederherstellung der folgenden Invariante erfolgen: *Für jedes Element e in der Hashtabelle mit Schlüssel $k(e)$, aktueller Position j und optimaler Position $i = h(k(e))$ sind alle Positionen $i, (i+1) \bmod m, (i+2) \bmod m, \ldots, j$ der Hashtabelle belegt.* Überlegen Sie sich eine effiziente Strategie, die diese Invariante widerherstellt. Bei dieser Aufgabe soll keine dynamische Größenanpassung der Hashtabelle stattfinden.

1. Operation: `insert`(16) mit opt. Position: 9

0	1	2	3	4	5	6	7	8	9	10	11	12
									16			

2. Operation: `insert`(3) mit opt. Position: 9

0	1	2	3	4	5	6	7	8	9	10	11	12
									16	3		

3. Operation: `insert`(12) mit opt. Position: 10

0	1	2	3	4	5	6	7	8	9	10	11	12
									16	3	12	

4. Operation: `insert`(17) mit opt. Position: 12

0	1	2	3	4	5	6	7	8	9	10	11	12
									16	3	12	17

5. Operation: `insert`(29) mit opt. Position: 9

0	1	2	3	4	5	6	7	8	9	10	11	12
29									16	3	12	17

6. Operation: `insert`(10) mit opt. Position: 4

0	1	2	3	4	5	6	7	8	9	10	11	12
29				10					16	3	12	17

7. Operation: `insert`(24) mit opt. Position: 7

0	1	2	3	4	5	6	7	8	9	10	11	12
29				10			24		16	3	12	17

8. Operation: `delete`(16) mit opt. Position: 9

0	1	2	3	4	5	6	7	8	9	10	11	12
				10			24		3	12	29	17

9. Operation: `insert`(5) mit opt. Position: 2

0	1	2	3	4	5	6	7	8	9	10	11	12
		5		10			24		3	12	29	17

10. Operation: `insert`(1) mit opt. Position: 3

0	1	2	3	4	5	6	7	8	9	10	11	12
		5	1	10			24		3	12	29	17

11. Operation: `insert`(15) mit opt. Position: 6

0	1	2	3	4	5	6	7	8	9	10	11	12
		5	1	10		15	24		3	12	29	17

12. Operation: `delete`(10) mit opt. Position: 4

0	1	2	3	4	5	6	7	8	9	10	11	12
		5	1			15	24		3	12	29	17

13. Operation: `insert`(14) mit opt. Position: 3

0	1	2	3	4	5	6	7	8	9	10	11	12
		5	1	14		15	24		3	12	29	17

14. Operation: `delete`(1) mit opt. Position: 3

0	1	2	3	4	5	6	7	8	9	10	11	12
		5	14			15	24		3	12	29	17

Aufgabe 3: Hashing mit Chaining

Veranschaulichen Sie Hashing mit Chaining. Die Größe m der Hash-Tabelle ist in den folgenden Beispielen jeweils die Primzahl 11. Die folgenden Operationen sollen nacheinander ausgeführt werden.

`insert`	$3, 11, 9, 7, 14, 56, 4, 12, 15, 8, 1$
`delete`	56
`insert`	25

Der Einfachheit halber sollen die Schlüssel der Elemente die Elemente selbst sein.

Verwenden Sie zunächst die Hashfunktion

$$g(x) = 5x \mod m.$$

$k(e)$	1	3	4	7	8	9	11	12	14	15	25	56
$g(k(e))$	5	4	9	2	7	1	0	5	4	9	4	5

1. Operation: `insert`(3):

0	1	2	3	4	5	6	7	8	9	10
				3						

2. Operation: `insert`(11):

0	1	2	3	4	5	6	7	8	9	10
11				3						

3. Operation: `insert`(9):

0	1	2	3	4	5	6	7	8	9	10
11	9			3						

4. Operation: `insert`(7):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3						

5. Operation: `insert`(14):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3 14						

6. Operation: `insert`(56):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3 14	56					

7. Operation: `insert`(4):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3 14	56				4	

8. Operation: `insert`(12):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3 14	56 12				4	

9. Operation: `insert`(15):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3	56				4	
				14	12				15	

10. Operation: `insert`(8):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3	56		8		4	
				14	12				15	

11. Operation: `insert`(1):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3	56		8		4	
				14	12				15	
					1					

12. Operation: `delete`(56):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3	12		8		4	
				14	1				15	

13. Operation: `insert`(25):

0	1	2	3	4	5	6	7	8	9	10
11	9	7		3	12		8		4	
				14	1				15	
				25						

Aufgabe 4: Relationale Modellierung

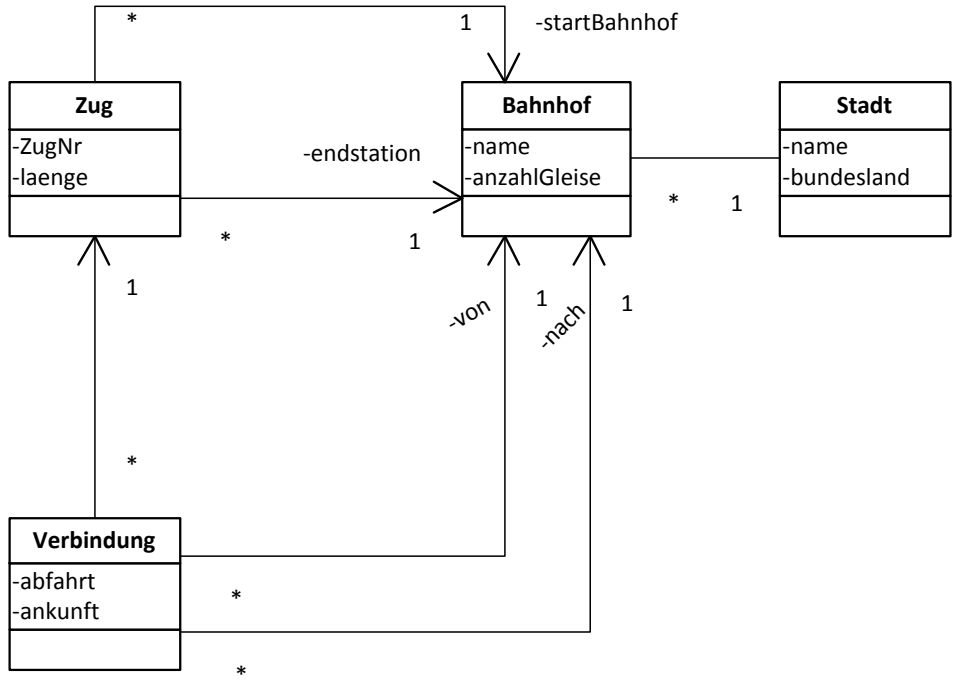

Abbildung 1: UML Modell von Zugverbindungen

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.

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:

Züge	:	{[ZugNr: integer, Laenge: integer]}	(1)
Bahnhöfe	:	{[Name: string, AnzahlGleise: integer]}	(2)
Städte	:	{[Name: string, Bundesland: string]}	(3)
Verbindung	:	{[VerbindungsNr: integer, Abfahrt: date, Ankunft: date]}	(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]}