

Übung zur Vorlesung Datenbanksysteme im WS05/06

Richard Kuntschke (Richard.Kuntschke@in.tum.de)

Martin Wimmer (Martin.Wimmer@in.tum.de)

Lösung von Blatt Nr. 14

Aufgabe 1

Modellieren Sie ein Zugauskunftssystem, in dem die wichtigsten Züge (z.B. die Intercity- und Eurocity-Züge) repräsentiert werden. Aus dem System sollen die Start- und Zielbahnhöfe und die durch den Zug verbundenen Bahnhöfe einschließlich Ankunfts- und Abfahrtszeiten ersichtlich sein. Geben Sie die Funktionalitäten der Beziehungstypen an.

Lösung

Eine mögliche Modellierung ist in Abbildung 1 dargestellt. Folgende Eigenschaften werden hier umgesetzt:

- Jeder Bahnhof liegt in einer Stadt. Eine Stadt kann aber wiederum mehrere Bahnhöfe haben.
- Jeder Zug hat einen ausgewiesenen Start- und Zielbahnhof, d.h. fährt auf einer festen Route.
- Auf dieser Route können mehrere Zwischenstationen auftreten, dies wird über die dreistellige Relation *verbindet* realisiert. Die Ankunfts- und Abfahrtszeiten stellen Attribute der Relation dar.

Wir wollen die Relation *verbindet* noch genauer betrachten, da sie uns auch in späteren Aufgaben noch beschäftigen wird. Die Beziehung wurde so modelliert, dass die Multiplizität von *Von-* und *Nach-*Bahnhöfen jeweils 1 und die Funktionalität von *Züge* N ist. Die Rollen *von* und *nach* werden, wie wir später noch genauer zeigen, über *VonBahnhof* und *NachBahnhof* realisiert. Damit sind folgende Konsistenzbedingungen modelliert:

$$\begin{aligned} \text{Züge} \times \text{VonBahnhof} &\rightarrow \text{NachBahnhof} \\ \text{Züge} \times \text{NachBahnhof} &\rightarrow \text{VonBahnhof} \end{aligned}$$

Jedem Zug wird eine eindeutige Identifikationsnummer zugeordnet, abhängig von der Verbindung. Beispielsweise gibt es einen ICE mit der Nummer 1518, der München und Kiel verbindet. Fährt dieser Zug die Strecke wieder zurück, erhält er eine andere Zugnummer zugewiesen. Einträge in die *verbindet*-Relation sehen z.B. wie folgt aus¹:

¹Auf eine vollständige Datumsangabe in den Spalten *Ankunft* und *Abfahrt* wurde zugunsten einer besseren Übersichtlichkeit verzichtet.

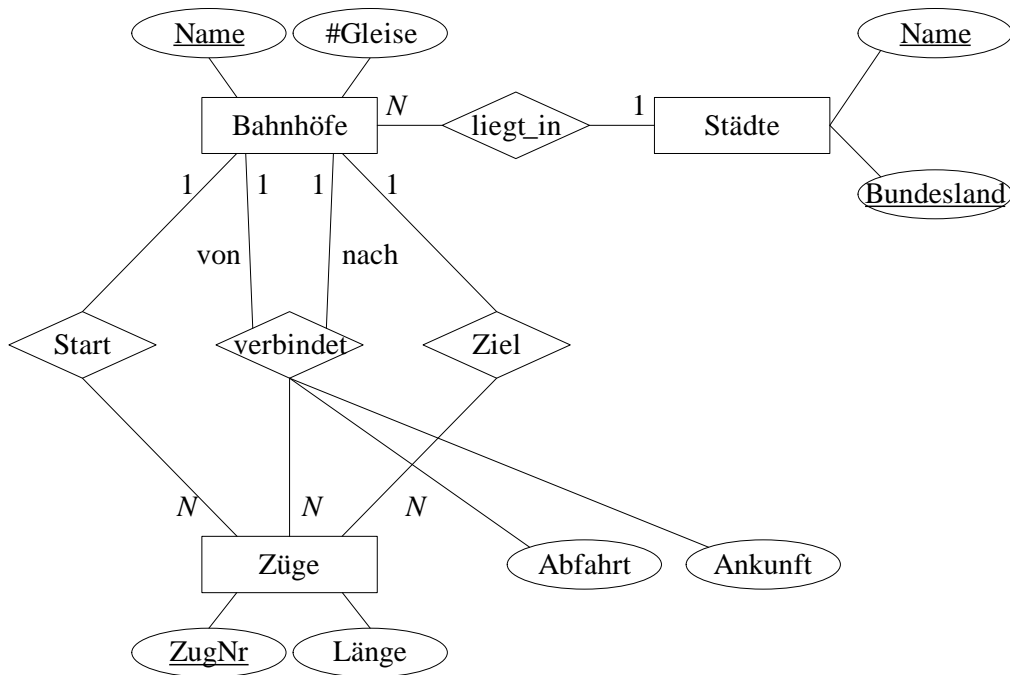


Abbildung 1: Modellierung eines Zugauskunftssystems

verbindet				
ZugNr	VonBahnhof	NachBahnhof	Abfahrt	Ankunft
ICE 1518	München Hbf	Augsburg Hbf	10:47	11:23
ICE 1518	Augsburg Hbf	Nürnberg Hbf	11:25	11:28
ICE 1518	Nürnberg Hbf	⋮	⋮	⋮
ICE 1518	Neumünster	Kiel Hbf	20:32	20:49
RE 11215	Bordesholm	Neumünster	10:32	10:41
RE 11215	⋮	⋮	⋮	⋮
RE 11215	⋮	Hamburg Hbf	11:34	11:37

Aufgabe 2

Übertragen Sie das ER-Modell aus Aufgabe 1 in ein relationales Schema. Verfeinern Sie das relationale Schema soweit möglich durch Eliminierung von Relationen.

Lösung

Erstellen des relationalen Schemas

Die initiale Überführung ergibt folgende Relationen für die Entitytypen:

$$\text{Städte} : \{[\text{Name} : \text{string}, \text{Bundesland} : \text{string}]\} \quad (1)$$

$$\text{Bahnhöfe} : \{[\text{Name} : \text{string}, \text{\#Gleise} : \text{integer}]\} \quad (2)$$

$$\text{Züge} : \{[\text{ZugNr} : \text{integer}, \text{Länge} : \text{integer}]\} \quad (3)$$

Für die Beziehungstypen werden folgende Relationen erstellt:

$$\text{liegt_in} : \{[\underline{\text{BName}} : \text{string}, \text{SName} : \text{string}, \text{Bundesland} : \text{string}]\} \quad (4)$$

$$\text{Start} : \{[\underline{\text{ZugNr}} : \text{integer}, \text{BName} : \text{string}]\} \quad (5)$$

$$\text{Ziel} : \{[\underline{\text{ZugNr}} : \text{integer}, \text{BName} : \text{string}]\} \quad (6)$$

$$\text{verbindet} : \{[\underline{\text{VonBahnhof}} : \text{string}, \text{NachBahnhof} : \text{string}, \underline{\text{ZugNr}} : \text{integer}, \text{Abfahrt} : \text{date}, \text{Ankunft} : \text{date}]\} \quad (7)$$

Als Nächstes wird das relationale Schema verfeinert, indem Relationen zusammengefasst werden.

Dabei werden Relationen für binäre Beziehungstypen mit Relationen für Entitytypen zusammengefasst, falls diese gleiche Schlüssel besitzen und es sich dabei um 1:N, N:1 oder 1:1 Beziehungen handelt.

So kann Relation (4) in (2) aufgenommen werden. (5) wird mit (3) zusammengefasst. Auch die *Ziel*-Relation (6) wird mit der *Züge*-Relation (3) zusammengefasst, d.h.

$$(4) \mapsto (2), (5) \mapsto (3), (6) \mapsto (3)$$

Damit ergibt sich folgendes Schema:

$$\begin{aligned} \text{Städte} & : \{[\underline{\text{Name}} : \text{string}, \underline{\text{Bundesland}} : \text{string}]\} \\ \text{Bahnhöfe} & : \{[\underline{\text{Name}} : \text{string}, \# \text{Gleise} : \text{integer}, \text{SName} : \text{string}, \text{Bundesland} : \text{string}]\} \\ \text{Züge} & : \{[\underline{\text{ZugNr}} : \text{integer}, \text{Länge} : \text{integer}, \text{StartBahnhof} : \text{string}, \text{ZielBahnhof} : \text{string}]\} \\ \text{verbindet} & : \{[\underline{\text{VonBahnhof}} : \text{string}, \text{NachBahnhof} : \text{string}, \underline{\text{ZugNr}} : \text{integer}, \text{Abfahrt} : \text{date}, \text{Ankunft} : \text{date}]\} \end{aligned}$$

Im vorliegenden Fall ist die Zugnummer eindeutig für eine Verbindung. Ein ICE, der die Städte München (*StartBahnhof*) und Berlin (*ZielBahnhof*) verbindet, hat somit eine eindeutige Zugnummer für diese Verbindung, die über mehrere Zwischenbahnhöfe erfolgen kann. Fährt der Zug zurück, erhält er eine andere Nummer zugewiesen. Dadurch sind die Kombinationen (*ZugNr*, *VonBahnhof*) und (*ZugNr*, *NachBahnhof*) zwei mögliche Schlüssel für die Relation *verbindet*.

Aufgabe 3

Formulieren Sie folgende Anfragen in SQL:

- Suchen Sie unter Verwendung von **any** die Professoren heraus, die Vorlesungen halten. Finden Sie mindestens zwei weitere alternative äquivalente Formulierungen dieser Anfrage.
- Ermitteln Sie für die einzelnen Vorlesungen die Durchfallquote als die Anzahl der durchgefallenen Prüflinge relativ zur Anzahl der für diese Vorlesung angetretenen Prüflinge. Als Variation der obigen Anfrage ermitteln Sie die Durchfallquote bei den einzelnen Professoren.

Anfrage 1

Lösung

Formulierung unter Verwendung von any

```
select PersNr, Name
from Professoren
where PersNr = any (select distinct gelesenVon
                    from Vorlesungen);
```

Zwei alternative Formulierungen

```
-- unter Verwendung von exists
select PersNr, Name
from Professoren
where exists (select *
              from Vorlesungen
              where gelesenVon = PersNr);

-- unter Verwendung von distinct
select distinct PersNr, Name
from Professoren, Vorlesungen
where gelesenVon = PersNr;
```

Anfrage 2

Lösung

Durchfallquote pro Vorlesung:

Um diese Anfrage zu beantworten, erstellen wir zuerst zwei Views, *durchgefallene* und *alle*. Mit der ersten Sicht ermitteln wir für jede Vorlesung die Anzahl der Studenten, die eine Prüfung nicht bestanden haben. Der Sonderfall, dass kein Student in einer Prüfung über die jeweilige Vorlesung durchgefallen ist, muss dabei auch berücksichtigt werden. Mit der zweiten View fragen wir ab, wie viele Studenten eine Prüfung über jede Vorlesung abgelegt haben. Der Sonderfall, dass kein Student sich über eine bestimmte Vorlesung hat prüfen lassen, muss hier nicht berücksichtigt werden. Dies erfolgt in der anschließenden Anfrage.

```
create view durchgefallene as
(select VorlNr, count(Note) as Anzahl
 from prüfen
 where Note > 4.0
 group by VorlNr)
union
(select VorlNr, 0 as Anzahl
 from Vorlesungen
 where VorlNr not in (select VorlNr
                      from prüfen
                      where Note > 4.0));

create view alle as
select VorlNr, count(*) as Anzahl
from prüfen
group by VorlNr;
```

Basierend auf diesen Views lässt sich die Aufgabe damit wie folgt beantworten:

```

(select a.VorlNr, float(d.Anzahl)/float(a.Anzahl) as Grad
 from alle a, durchgefallene d
 where a.VorlNr = d.VorlNr)
union
(select VorlNr, 0 as Grad
 from Vorlesungen
 where VorlNr not in (select VorlNr
                      from prüfen));

```

Wie zuvor angeführt, wird der Sonderfall, dass keine Studenten eine bestimmte Vorlesung prüfen ließen, durch die zweite Ergebnismenge behandelt.

Eine alternative, in gewisser Weise elegantere, da kürzere Formulierung sieht wie folgt aus:

```

create view Vorl_AnzDurch_AnzAlle as
select VorlNr, count(*) as AnzAlle,
       sum(case when Note > 4.0 then 1 else 0 end) as
       AnzDurch
from   prüfen
group by VorlNr;

(select VorlNr, float(AnzDurch)/float(AnzAlle) as Grad
 from Vorl_AnzDurch_AnzAlle)
union
(select VorlNr, 0 as Grad
 from Vorlesungen
 where VorlNr not in (select VorlNr from prüfen));

```

Durchfallquote pro Professor:

Auch für diese Anfrage erstellen wir uns zuerst zwei temporäre Views. *durchgefalleneProProf* bestimmt je Professor die Anzahl der Studenten, die Prüfungen bei ihm/ihr nicht bestanden haben, wohingegen *alleProProf* für jeden Professor die Anzahl abgenommener Prüfungen zählt (erneut werden die Ereignisse mit leerem Ergebnis hier noch nicht betrachtet).

```

create view durchgefalleneProProf as
(select PersNr, count(Note) as Anzahl
 from   prüfen
 where Note > 4.0
 group by PersNr)
union
(select PersNr, 0 as Anzahl
 from   Professoren
 where PersNr not in
       (select PersNr from prüfen where Note > 4.0));

create view alleProProf as
select PersNr, count(*) as Anzahl
 from   prüfen
 group by PersNr;

```

Analog sieht die Anfrage dann wie folgt aus:

```

(select a.PersNr, float(d.Anzahl)/float(a.Anzahl) as Grad
 from alleProProf a, durchgefalleneProProf d
 where a.PersNr = d.PersNr)

```

```

union
(select PersNr, 0 as Grad
 from Professoren
 where PersNr not in (select PersNr from prüfen));

```

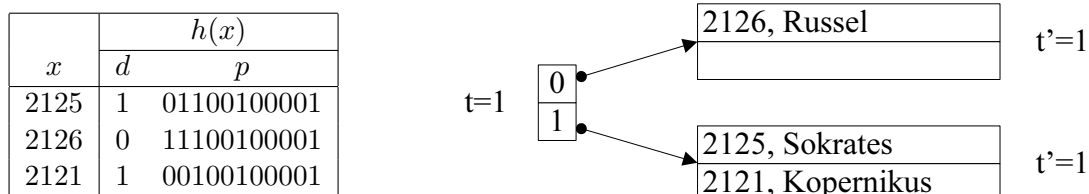
Auch hier kann wieder eine entsprechend kürzere Lösung wie im vorhergehenden Fall erstellt werden.

Aufgabe 4

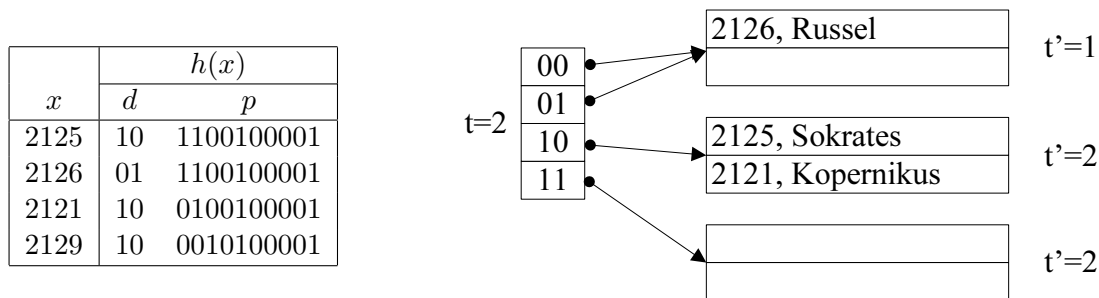
Gegeben sei eine erweiterbare Hashtabelle mit Bucketgröße 2. Fügen Sie die Tupel für Sokrates, Russel, Kopernikus und Descartes in diese Hashtabelle ein. Verwenden Sie als Hashfunktion die inverse binäre Darstellung der Personalnummer und nehmen Sie an, dass (anders als in der bekannten Ausprägung) die Personalnummer von Kopernikus 2121 ist.

Lösung

Als Hashfunktion h wird die inverse binäre Darstellung der Personalnummer verwendet. Für Kopernikus mit Personalnummer 2121 ergibt sich damit $h(2121) = 100100100001$. Folgende Abbildung zeigt die erweiterbare Hashtabelle, wenn Kopernikus, Sokrates (2125) und Russel (2126) bereits eingefügt wurden:

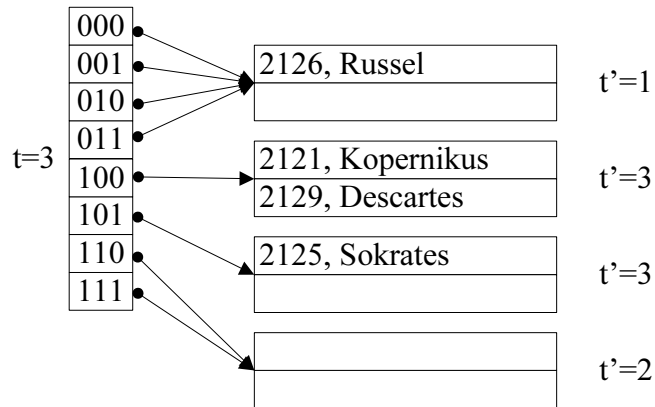


Möchte man nun, wie im ursprünglichen Beispiel, Descartes (2129) in die Tabelle mit aufnehmen, so erfolgt zuerst eine Aufspaltung des Verzeichnisses. Wie man folgender Tabelle entnehmen kann, unterscheiden sich die Hashwerte für Sokrates, Kopernikus und Descartes aber nicht bezüglich der ersten beiden Binärzahlen.



Deshalb muss das Verzeichnis erneut verdoppelt werden. Es ergibt sich, nach dem Einfügen von Descartes, dann folgende Hashtabelle:

x	$h(x)$	
	d	p
2125	101	100100001
2126	011	100100001
2121	100	100100001
2129	100	010100001



Aufgabe 5

Demonstrieren Sie anhand eines Beispiels, dass man die Strategien *force* und \neg *steal* nicht kombinieren kann, wenn parallele Transaktionen gleichzeitig Änderungen an Datenobjekten innerhalb einer Seite durchführen. Betrachten Sie dazu z.B. die in Abbildung 2 dargestellte Seitenbelegung, bei der die Seite P_A die beiden Datensätze A und D enthält. Entwerfen Sie eine verzahnte Ausführung zweier Transaktionen, bei der eine Kombination aus *force* und \neg *steal* ausgeschlossen ist.

Lösung

Die Datenobjekte A und D liegen auf einer Seite P_A auf dem Hintergrundspeicher. Die Pufferseiten-Ersetzungsstrategien sind wie folgt definiert:

- steal*: Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung, falls neue Seiten eingelagert werden müssen.
- \neg *steal*: Seiten, die Änderungen von noch aktiven Transaktionen enthalten, dürfen nicht ausgelagert werden.
- force*: Beim **commit** einer Transaktion werden **alle** von ihr modifizierte Seiten in die Datenbasis übertragen.
- \neg *force*: Von einer Transaktion modifizierte Seiten müssen nach dem **commit** der Transaktion nicht notwendigerweise in die Datenbasis eingebracht werden.

Wir zeigen im Folgenden einen Beispielschedule zweier Transaktionen T_1 und T_2 , der eine Kombination der Strategien *force* und \neg *steal* unmöglich macht.

Beispielschedule

Schritt	T_1	T_2
1.	BOT	
2.		BOT
3.	read(A)	
4.		read(D)
5.		write(D)
6.	write(A)	
7.	commit	\vdots

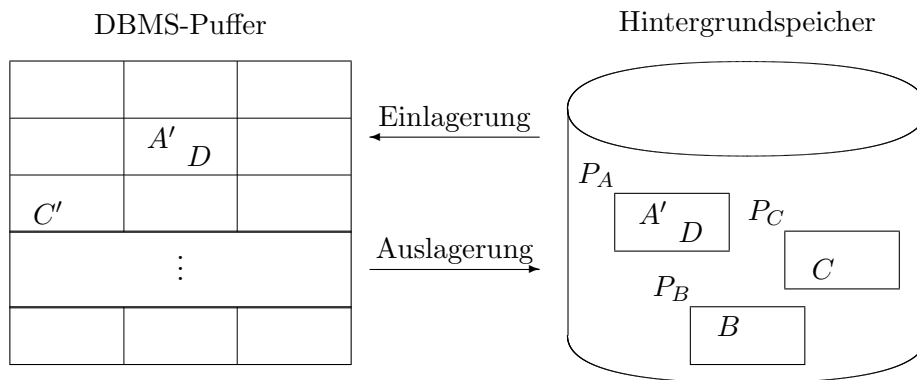


Abbildung 2: Schematische Darstellung der (zweistufigen) Speicherhierarchie

Wegen der *force* Strategie beim Einbringen von Änderungen und der \neg *steal* Strategie bei der Seitenersetzung entsteht bei Schritt 7 ein Konflikt:

- Wegen *force* müssten die Änderungen der Transaktion T_1 auf der Seite P_A auf den Hintergrundspeicher propagiert werden.
- Wegen \neg *steal* dürfen aber die Änderungen der noch aktiven Transaktion T_2 auf der Seite P_A nicht auf den Hintergrundspeicher gelangen.

Selbst eine Verzögerung des **commit** von T_1 würde das Problem nicht lösen, wenn T_2 als Nächstes versucht, eine Sperre auf A (die ja noch von T_1 gehalten wird) zu erlangen.

Aufgabe 6

In Abbildung 3 ist die verzahnte Ausführung der beiden Transaktionen T_1 und T_2 und das zugehörige *Log* auf der Basis logischer Protokollierung gezeigt. Wie sähe das *Log* bei physischer Protokollierung aus, wenn die Datenobjekte A , B und C die Initialwerte 1000, 2000 und 3000 hätten?

Lösung

Verwendet man anstelle der logischen Protokollierung die physische Log-Protokollierung, so müssen die After- und Before-Image-Einträge angepasst werden. Diese spiegeln den tatsächlichen Zustand der Datensätze vor bzw. nach der Ausführung der Operationen wider. Das *Log* sieht dann wie folgt aus:

```

[#1,  $T_1$ , BOT, 0]
[#2,  $T_2$ , BOT, 0]
[#3,  $T_1$ ,  $P_A$ ,  $A = 950$ ,  $A = 1000$ , #1]
[#4,  $T_2$ ,  $P_C$ ,  $C = 3100$ ,  $C = 3000$ , #2]
[#5,  $T_1$ ,  $P_B$ ,  $B = 2050$ ,  $B = 2000$ , #3]
[#6,  $T_1$ , commit, #5]
[#7,  $T_2$ ,  $P_A$ ,  $A = 850$ ,  $A = 950$ , #4]
[#8,  $T_2$ , commit, #7]

```

Schritt	T_1	T_2	Log
			[LSN,TA,PageID,Redo,Undo,PrevLSN]
1.	BOT		[#1, T_1 , BOT , 0]
2.	$r(A, a_1)$		
3.		BOT	[#2, T_2 , BOT , 0]
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		[#3, T_1 , P_A , $A-=50$, $A+=50$, #1]
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	[#4, T_2 , P_C , $C+=100$, $C-=100$, #2]
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		[#5, T_1 , P_B , $B+=50$, $B-=50$, #3]
12.	commit		[#6, T_1 , commit , #5]
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	[#7, T_2 , P_A , $A-=100$, $A+=100$, #4]
16.		commit	[#8, T_2 , commit , #7]

Abbildung 3: Verzahnte Ausführung zweier Transaktionen und das erstellte Log

Zur Wiederholung: Bei physischer Protokollierung steht im Log zuerst das After- und dann das Before-Image, in obigem Beispiel also:

$$[#3, T_1, P_A, \underbrace{A = 950}_{\text{After-Image}}, \underbrace{A = 1000}_{\text{Before-Image}}, #1]$$