Klausur Datenbanken
Wintersemester 2005/2006
Prof. Dr. Wolfgang May
14. Februar 2006, 14-16 Uhr
Bearbeitungszeit: 90 Minuten

Vorname:

Nachname:

Matrikelnummer:

Bei der Klausur sind **keine Hilfsmittel** (Skripten, Taschenrechner, etc.) erlaubt. Handies müssen ausgeschaltet sein. Papier wird gestellt. Benutzen Sie nur die **ausgeteilten**, zusammengehefteten **Blätter** für Ihre Antworten. Schreiben Sie mit blauem/schwarzem Kugelschreiber, Füller, etc.; Bleistift ist nicht erlaubt.

Auf dem letzten Blatt finden Sie eine Datenbasis, die in den Aufgaben 2 und 3 verwendet wird. Trennen Sie es ggf. zur Bearbeitung der Aufgaben ab.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

☐ meine Note soll mit Matrikelnummer so bald wie möglich auf der Vorlesungs-Webseite veröffentlicht werden.

☐ meine Note soll nicht veröffentlicht werden; ich erfahre sie dann aus Munopag (bzw. CLZ: beim Prüfungsamt dort zu erfragen).

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (ER-Modell, Relationales Modell)	15	10
Aufgabe 2 (SQL und Relationale Algebra)	28	19
Aufgabe 3 (Anfragen Allgemein)	21	9
Aufgabe 4 (Transaktionen)	26	10
Summe	90	48

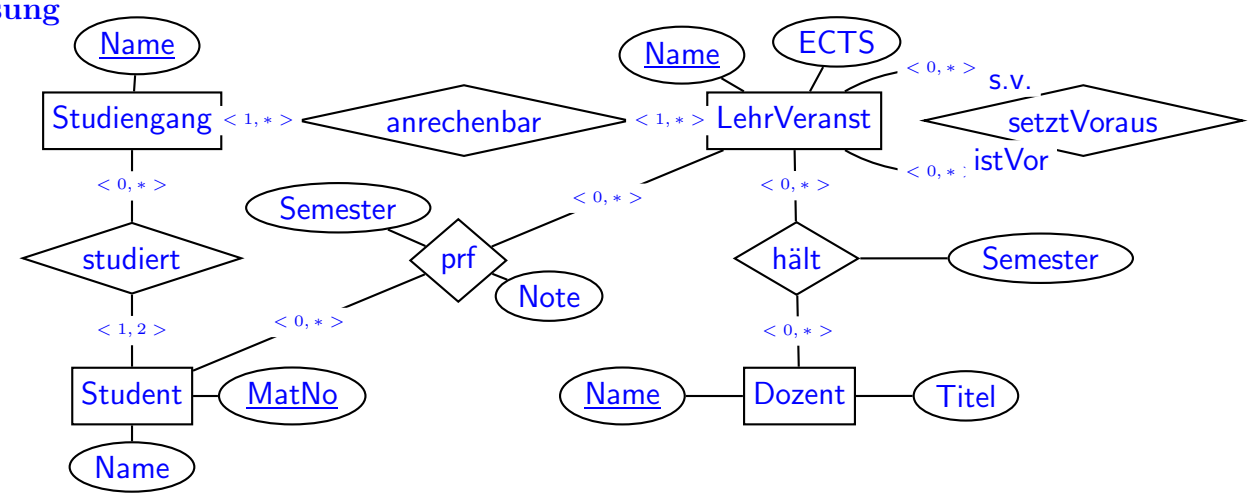
Note:

Aufgabe 1 (ER-Modell [15 Punkte])

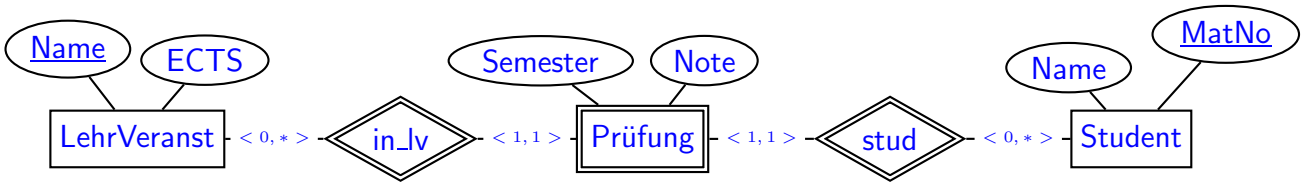
Geben Sie ein ER-Modell für den folgenden Sachverhalt an (die für die SQL-Aufgaben angegebene Datenbasis ist ein Teil dieses Szenarios). Geben Sie Schlüssel und Kardinalitäten sinnvoll an.

- Es gibt Studiengänge. Jeder Studiengang hat einen Namen (z.B. "Informatik", "Mathematik", ...).
- Studierende: Jeder Student hat einen Namen und eine Matrikelnummer, studiert in mindestens einem, maximal zwei Studiengängen.
- Dozenten: Jeder Dozent hat einen Namen und einen Titel (z.B. "Prof." oder "Dr.").
- Lehrveranstaltungen (LV): Jede LV hat einen Namen und ist in mindestens einem Studiengang anrechenbar. Jeder LV ist eine bestimmte Anzahl von ECTS-Kreditpunkten (entsprechend dem Aufwand) zugeordnet.
- Das Vorlesungsverzeichnis (VV) gibt an, welcher Dozent in welchem Semester welche Vorlesung(en) hält (hier gehen wir von einem VV aus, das Daten über viele Semester enthalten kann).
- Teilnahmevoraussetzungen geben an, welche LVs man erfolgreich absolviert haben muss, um an einer anderen LV teilzunehmen.
- Studierende, die in einem Semester an einer Prüfung zu einer LV teilgenommen haben, bekommen jeweils eine Note. (Hinweis: Sie können für diese Aufgabe davon ausgehen, dass Prüfungen auch in Semestern stattfinden, in denen die Vorlesung nicht gehalten wurde.)

Lösung



Bewertung u.a.:
fehlende Teilnahmevoraussetzungen -1.5P, fehlende Rollen -0.5P.
Es gab einige Lösungen, die anstelle der Relationen "prf" und "hält" Entity-Types verwenden wollten. Das geht, allerdings muss man es richtig machen (hier am Beispiel von "prf"), indem man *weak entity type* nimmt, der sich durch die teilnehmenden anderen Entities identifiziert (jeweils mit Kardinalität $< 1, 1 >$):


Name
ECTS
LehrVeranst
< 0, ∗ >
in_lv
< 1, 1 >
Semester
Note
Prüfung
< 1, 1 >
stud
< 0, ∗ >
Name
MatNo
Student

Aufgabe 2 (SQL und Relationale Algebra [28 Punkte])
Verwenden Sie für Aufgaben 2 und 3 die Datenbasis, die auf dem letzten Blatt der Klausur angegeben ist.

1. Geben Sie ein **SQL-Statement** an, das das folgende tut:
   Ausgabe aller Paare (Student(Name), Lehrveranstaltung), so dass der Student eine Prüfung über die LV nicht bestanden hat (Noten 1 bis 4.0 sind bestanden, 5.0 ist nicht bestanden). (3P)

   Lösung

   ```
   SELECT distinct name, lv
   FROM stud, prf
   WHERE stud.matno = prf.matno
     AND note > 4;
   ```

   (fehlendes distinct -0.5P)

2. Geben Sie einen **Ausdruck (oder Baum) der relationalen Algebra** an, der (1) beantwortet. (3P)

   Lösung π[name,lv](σ[note>4](stud $\bowtie$ prf))

3. Geben Sie ein **SQL-Statement** an, das das folgende tut:
   "Alle Paare (Student(Name), Dozent), so dass der Student eine Prüfung bei diesem Dozenten bestanden hat". (3P)

   Lösung

   ```
   SELECT distinct stud.name, vv.dozent
   FROM stud, prf, vv
   WHERE stud.matno = prf.matno
     AND prf.lv = vv.lv
     AND prf.semester = vv.semester;
     AND note <= 4;
   ```

   (fehlendes distinct -0.5P)

4. Geben Sie ein **SQL-Statement** an, das das folgende tut:
   "Die Namen aller Studierenden, die mindestens 180 ECTS-Punkte erfolgreich erbracht haben". (6 P)

   Lösung

   ```
   SELECT name,
   FROM stud, prf, lv
   WHERE stud.matno = prf.matno
     AND prf.lv = lv.name
     AND note <= 4
   ```

```
GROUP BY matno, stud.name
 // man braucht beides:
 // matno wichtig, da es der key ist (group by name wuerde die Daten
 //       zweier Studierender mit demselben Namen zusammenrechnen)
 // name wichtig, sonst nicht im select erlaubt!
HAVING sum(lv.ects) >= 180

SELECT name
FROM stud
WHERE 180 <=
  (SELECT SUM(lv.ects)
   FROM prf, lv
   WHERE prf.matno = stud.matno
     AND prf.lv = lv.name
     AND note <= 4)
```

Die Anfrage geht also auch mit einer korrelierten Subquery anstelle GROUP BY und HAVING zu beantworten. Dies ist allerdings nur in Fällen möglich, wo man das Ergebnis der Subquery nur zum Vergleichen benötigt und es nicht im Ergebnis ausgeben will.

5. Geben Sie einen **SQL-Ausdruck** an, der die Namen aller Studierenden zurückgibt, die "Datenbanken" noch nicht erfolgreich absolviert haben (5P)

Lösung
Hierbei handelt es sich sowohl um diejenigen Studenten, die "Datenbanken" noch nie geschrieben haben, als auch um diejenigen, die es zwar versucht, aber immer durchgefallen sind (Ein Lösungsweg wäre also auch, diese Mengen zu berechnen und zu vereinigen). Einfacher kann man aber sagen: alle, die "Datenbanken" bisher nicht bestanden haben:

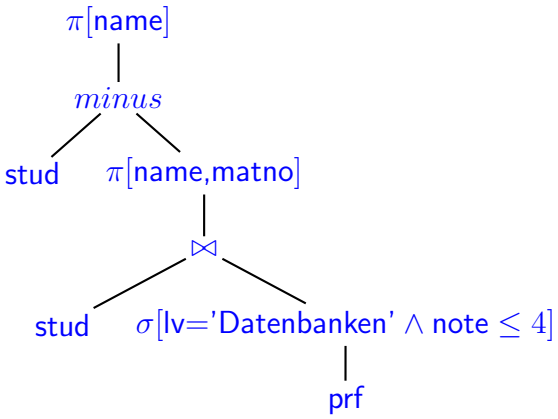
```
SELECT name
FROM stud
WHERE NOT EXISTS
 ( SELECT *
   FROM prf
   WHERE stud.matno = prf.matno
     AND prf.lv = 'Datenbanken'
     AND note <= 4 );
```

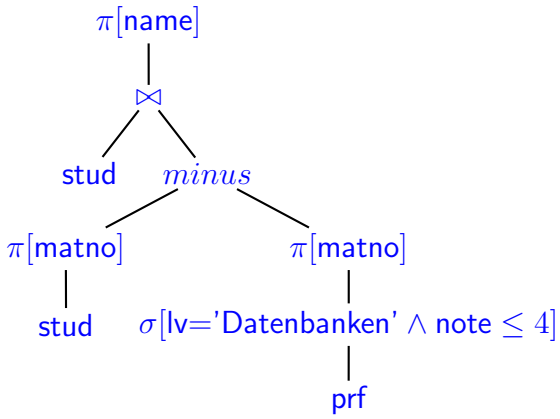
```
SELECT name
FROM stud
WHERE matno NOT IN
 (SELECT matno FROM prf
  WHERE prf.lv = 'Datenbanken'
    AND note <= 4 );
```

... und die SQL-Äquivalente der untenstehenden Bäume:

6. Geben Sie einen **Ausdruck (oder Baum) der relationalen Algebra** an, der (5) beantwortet. (3P)

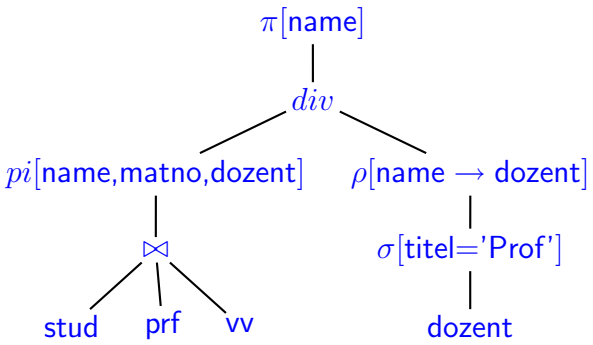
Lösung





7. Geben Sie eine SQL-Anfrage *oder* einen Algebra-Ausdruck an (d.h. entscheiden Sie selber, welche Anfragesprache Sie bevorzugen), der folgende Anfrage beantwortet: "Namen aller Studierenden, die bei jedem Professor zu mindestens einer Veranstaltung eine Prüfung abgelegt haben" (5P).

Lösung Offensichtlich eine relationale Division:



Hinweis: Das Ergebnis der div sind Tupel der Form (name,matno).
Als SQL-Anfrage ist es deutlich komplizierter (mit doppelt geschachteltem Minus und/oder NOT EXISTS).

Aufgabe 3 (Anfragen Allgemein [21 Punkte])
Geben Sie für die folgenden Anfragen jeweils an, ob Sie in SQL oder der relationalen Algebra lösbar sind. Falls ja, geben Sie einen SQL- oder Algebraausdruck an. Falls nein, begründen Sie, warum es nicht möglich ist.

a) "Die Menge aller Veranstaltungen, die ein Studierender erfolgreich absolviert haben muss, um am *Datenbankpraktikum SQL* teilnehmen zu dürfen". (3P)

Lösung Dies ist die *Transitive Hülle* der *setztVoraus*-Relation. Diese läßt sich in SQL und der relationalen Algebra nicht ausdrücken.

b) "Die Menge aller Studierenden die eine Prüfung zu einer Veranstaltung abgelegt haben, ohne die Voraussetzungen erfüllt zu haben". (6P)

Lösung Dies ist möglich, da es genügt, jedes einzelne Paar von Veranstaltungen zu überprüfen. Daraus ergibt sich dann die Erfüllung der transitiven Hülle automatisch.

```
SELECT name
FROM stud, prf, vorauss
WHERE stud.matnr = prf.matnr
  AND prf.lv = vorauss.setztvoraus
  AND NOT EXISTS
      (SELECT *
       FROM prf as prf2
       WHERE stud.matnr = prf2.matnr
         AND prf2.lv=vorauss.wirdvorausgesetzt
         AND prf2.semester < prf.semester
         AND note <= 4)
```

```
SELECT name
FROM stud, prf, vorauss
WHERE stud.matnr = prf.matnr
  AND prf.lv = vorauss.setztvoraus
  AND vorauss.wirdVorausgesetzt NOT IN
    (SELECT lv
       FROM prf2
       WHERE stud.matnr = prf2.matnr
         AND prf2.semester < prf.semester
         AND note <= 4)
```

Als Algebra-Baum/Ausdruck ist die Anfrage deutlich komplizierter (es ist keine einfache Division, sondern eine "korrelierte").

c) Die Tabelle `prf` enthält alle Prüfungsdaten aller Studierenden, und ist damit ziemlich groß. Welche Möglichkeiten gibt es, die *Speicherung der Daten innerhalb der Datenbank* zu optimieren um

- das Drucken eines Zeugnisses für einen einzelnen Studenten, und
- das Erstellen einer Notenstatistik über die "Informatik I" der letzten 10 Jahre

zu unterstützen (6P)?

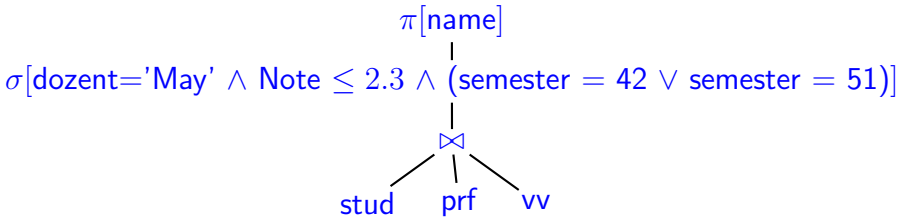
Lösung Es bieten sich Suchindexe und optimierte Speicherung an.
Zeugnisdruck: Baumindex oder Hash auf prf.matno. (2P)
Notenstatistik: Baumindex oder Bitmap auf prf.lv bzw. auf das Paar (prf.lv,prf.semester) (2P)
Bei matno bietet sich ein Hash-Index an, da man diese nur selten sequentiell nacheinander durchgehen wird. Für (lv,semester) bringt ein Hash wenig, da keine gleichmäßige Verteilung erreicht wird. Ein Bitmap-Index kann hier auch sinnvoll eingesetzt werden.
Man kann die Tabelle geordnet/geclustert ablegen (z.B. nach Matrikelnummern, da dieses Merkmal häufiger benötigt wird, als eine Notenstatistik). (2P)

Hingegen ist das Aufspalten der Tabelle auf mehrere Tabellen *kein* geeignetes Mittel! (damit müssten viele Anfragen viel umständlicher gestellt werden)

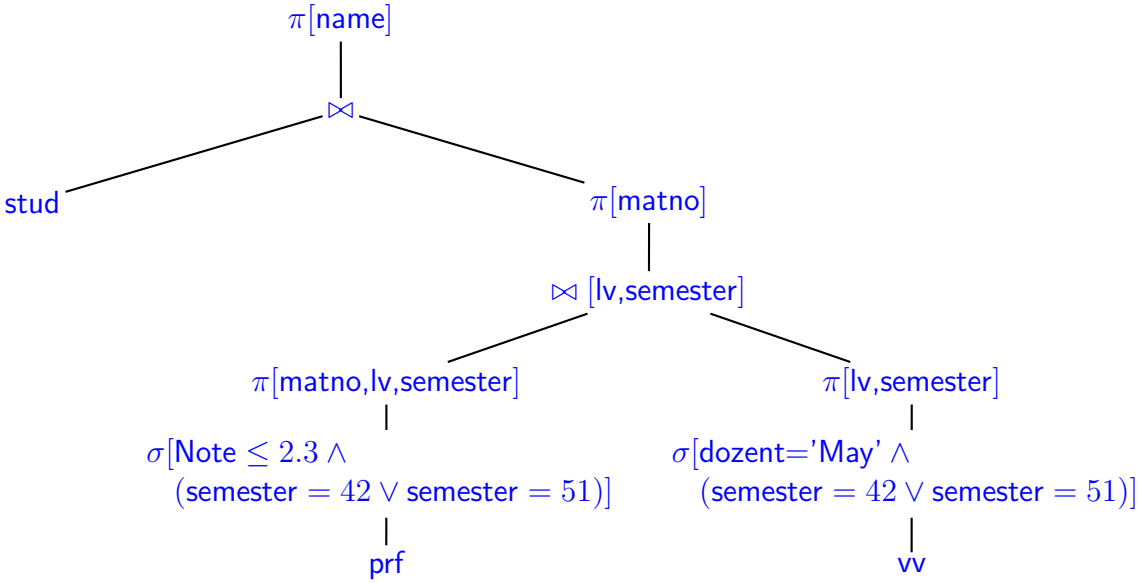
d) Anfrageoptimierung: Prof. May möchte eine Liste mit den Namen aller Studierenden, die im Jahr 2005 eine Prüfung bei ihm mit mindestens "gut" bestanden haben.

Geben Sie die Anfrage als Algebra-Ausdruck oder Algebra-Baum an, und optimieren Sie diese(n) soweit wie möglich. (6P)

Lösung



Selektionen soweit möglich nach unten, Projektionen ebenfalls so früh wie möglich und Zwischenergebnis möglichst klein:



Aufgabe 4 (Transaktionen [26 Punkte])
Gegeben sind die folgenden Transaktionen:

T_1:	T_2:
RA	RD
A := A-10	D := D-20
WA	WD
RC	RA
RB	A=A+20
B=B+10	WA
WB	

Betrachten Sie den folgenden Schedule S:

T_1	T_2
R_1A	
	R_2D
	$D := D - 20$
	W_2D
	R_2A
	$A := A + 20$
	W_2A
$A := A - 10$	
W_1A	
R_1C	
R_1B	
$B = B + 10$	
W_1B	

a) Gehen Sie von den Anfangswerten A=100, B=200, C=true und D=150 aus und geben Sie die Werte von A,B,C,D nach Ablauf des Schedules S an. (4P)

b) Geben Sie den Dependency-Graphen von S an (berücksichtigen Sie dabei auch die Transaktionen T_0 und T_∞ wie in der Vorlesung). (4P)

c) Geben Sie den Conflict-Graphen von S an. (4P)

d) Geben Sie zu jedem der Aufgabenteile a)-c) an, welche Hinweise es gibt, dass der Schedule nicht serialisierbar ist. Geben Sie an, ob diese Feststellungen eine sichere Aussage ermöglichen, oder nur Hinweise sind. (6P)

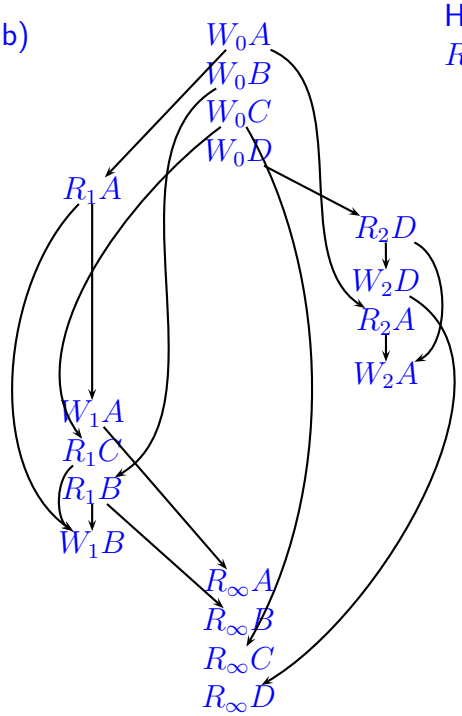
e) Ergänzen Sie beide Transaktionen mit LOCK-Aktionen (z.B. LOCK A und UNLOCK A; keine Unterscheidung zwischen Lese- und Schreib-Locks), so dass damit garantiert wird, dass nur noch serialisierbare Schedules ausgeführt werden können. Beschreiben Sie *kurz* die von Ihnen gewählte Strategie. (5P)

f) Geben Sie einen Schedule mit diesem Locking an, der mit der ersten Aktion von T_1 beginnt, und dann -soweit wie möglich- T_2 vorrangig bearbeitet. (3P)

Lösung

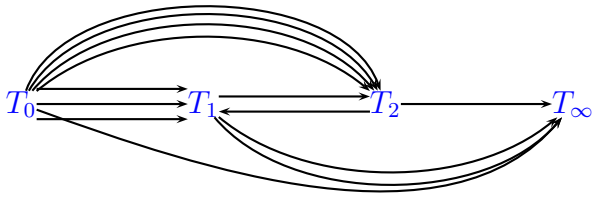
a) A=90, B=210, C=true, D=130.

b)



Häufig wurden die Kanten $W_0C \rightarrow R_\infty C$, $R_1C \rightarrow W_1B$ oder $W_0A \rightarrow R_2A$ vergessen.

c)



Man hat einen RW-Konflikt $R_1A \rightarrow W_2A$ und einen WR-Konflikt $W_2A \rightarrow W_1A$.

d) In (a) ist die Summe vorher 450, nachher 430, obwohl man sieht, dass keine der Transaktionen etwas verschwinden läßt. Dies zeigt sicher, dass etwas nicht stimmen kann.
Außerdem stellt man beim durchlaufen bereits fest, dass der bei W_2A geschriebene Wert überschrieben wird, ohne gelesen zu werden ("Lost Update")
In (b) sieht man dann auch im Graphen, dass der Wert von W_2A nie gelesen wird (keine Datenflusskante beginnt dort). Dies ist nur ein Hinweis, dass evtl. etwas nicht stimmt, da es auch in seriellen Schedules auftreten kann, wenn eine andere Transaktion später ein blind write ausführt.
Der C-Graph in (c) ist zyklisch. Dies genügt zuerst einmal, um festzustellen, dass S nicht C-serialisierbar ist. Da es keine blind writes gibt, ist dies gleichbedeutend damit, dass S nicht serialisierbar ist.

e) 2-Phasen Locking: Grundsätzlich darf eine Transaktion ein Datenitem nur lesen/schreiben, wenn sie ein LOCK darauf besitzt. Beim 2-Phasen Locking teilt sich die Transaktion in eine erste Phase, während der LOCKs angefordert werden, und eine zweite, in der sie wieder freigegeben werden. Aktionen siehe (g), wobei es viele verschiedene korrekte Möglichkeiten gibt.

f)

LA		
R_1A		
	// T_2 kann jetzt laufen, erst beim LA muss es warten	
	LD	
	R_2D	
	$D := D - 20$	
	W_2D	
$A := A - 10$		
W_1A		
LC, LB	// T_1 muss B und C locken, bevor UA	
UA	// jetzt kann T_2 A haben	
	LA	
	UD	
	R_2A	
	$A := A + 20$	
	W_2A	
	UA	
R_1C		
UC		
R_1B		
$B = B + 10$		
W_1B		
UB		

Die folgende Datenbasis wird in den Aufgaben 2 und 3 verwendet.

Betrachten Sie die folgende Datenbasis, die ein Prüfungsverwaltungssystem beschreibt: Studierende haben eine Matrikel-Nummer und einen Namen.
Semester werden mit 21 = Sommersemester 2002, 22 = Wintersemester 2002/03, 31 = Sommersemester 2003 bezeichnet. Auf diese Weise entspricht die "Kleiner-Relation" der zeitlichen Abfolge ($41 < 42 < 51$, Sommersemester 2004 früher als Wintersemester 2004/05 früher als Sommersemester 2005).

Stud

MatNo	Name
0900	Werner Brösel
4242	Karl Napf
:	:

Doz

Name	Titel
Hogrefe	Prof
Dix	Prof
Richter	Prof
May	Prof
Ebner	Dr
Behrends	DiplInf
:	:

LehrVeranst (LV)

Name	ECTS
Informatik I	9
Informatik II	9
Informatik II	9
Programmierprakt	9
Datenbanken	6
SQL-Prakt	9
:	:

VorlVerz (VV)

LV	Dozent	Semester
Informatik I	May	22
Informatik I	May	32
Informatik II	Fu	31
Informatik II	Ebner	41
Programmierprakt	Werner	41
Datenbanken	May	42
Datenbanken	May	52
SQL-Prakt	Behrends	51
:	:	:

setztVoraus

SetztVoraus	wirdVorausgesetzt
Informatik II	Informatik I
Datenbanken	Informatik II
SQL-Prakt	Datenbanken
SQL-Prakt	Programmierprakt
:	:

Prf

MatNo	LV	Semester	Note
0900	Informatik I	22	4.0
0900	Informatik II	31	3.3
0900	Programmierprakt	31	4.0
0900	Datenbanken	32	5.0
0900	Datenbanken	42	4.0
0900	SQL-Prakt	51	2.3
:	:	:	:

Hinweis: Für Aufgaben 2 und 3 werden Prüfungen jeweils nur in Semestern betrachtet, wo die Vorlesung auch gehalten wurde, um eine eindeutige Zuordnung zu einem Dozenten zu haben.