Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

University of Munich
Institute for Computer Science
Database and Information Systems

Prof. Dr. S. Conrad

Vorlesung Einführung in die Informatik:
Systeme und Anwendungen im SS 2001

KLAUSUR

Name:	Vorname:	Matrikelnummer:

Studienrichtung:

Einwilligungserklärung

Hiermit stimme ich der Veröffentlichung meines Ergebnisses bei dieser Klausur unter Verwendung meiner Matrikelnummer im Internet zu.

München, den 20. Juli 2001 _____

Organisatorisches:

Zur Klausur sind nicht-technische Hilfsmittel zugelassen,
d.h. handschriftliche Notizen, Kopien, Bücher und Vergleichbares.

Die Benutzung jeglicher technischer Hilfsmittel (Computer, Handies, etc.)
wird als Täuschungsversuch gewertet (dies bedeutet „nicht bestanden“).

Aufgabe	1	2	3	4	5	6	7	Σ
Max. Pkte	22	14	22	20	26	36	4	144
Punkte								

Note

1 Allgemeine Fragen (22 Punkte)

Kreuzen Sie an, ob folgende Aussagen richtig oder falsch sind.

Bewertung: 1 Punkt pro richtige Antwort, –1 Punkt pro falsche Antwort.
(Pro Teilaufgabe 1.1–4 können minimal 0 Punkte erworben werden.)

Tipp: Statt wildem Raten lieber keine Antwort geben.

1.1 Rechner, Betriebssystem

richtig | FALSCH — Eine Maschinensprache ist vom Typ RISC, wenn sie Befehle enthält, die nur von erfahrenen Programmierern benutzt werden sollen.

richtig | FALSCH — Der Wert des Program Counter sagt, wie viele Programme augenblicklich ausgeführt werden.

RICHTIG | falsch — Ein Multitasking Betriebssystem ist auch dann sinnvoll, wenn an einem Rechner immer nur ein Benutzer gleichzeitig arbeitet.

richtig | FALSCH — Unter einem Register eines Rechners versteht man eine Liste von Verwaltungseinheiten des Betriebssystems, also z.B. eine Liste von Prozessen, von Druckaufträgen ...

richtig | FALSCH — Der Prozeßkontrollblock dient dazu es dem Benutzer zu ermöglichen seine Prozesse zu überwachen

richtig | FALSCH — Prozesse haben 2 Arten von Zuständen: beschreibbare und unbeschreibbare.

1.2 Verklemmungen ...

RICHTIG | falsch — Wenn sich 2 Prozesse nicht verklemmen können, so ist damit nicht sichergestellt, daß keiner der beiden verhungern kann.

RICHTIG | falsch — Wenn 2 Prozesse sich verklemmen können, dann muß dies nicht zwingend eintreten.

richtig | FALSCH — Die einfachste Möglichkeit eine Verklemmung aufzulösen ist so lange zu warten, bis dies durch den Dispatcher im Verlauf des weiteren Rechenbetriebs von selbst erfolgt.

RICHTIG | falsch — Wenn ein Prozeß in den Zustand `blocked` gerät, bedeutet dies nicht, daß zwingend über kurz oder lang eine Verklemmung eintreten wird.

richtig | FALSCH — Solange noch ein Prozeß rechenwillig ist — einer `ready` und alle anderen `blocked` — kann noch keine Verklemmung vorliegen.

1.3 Semaphore

RICHTIG | falsch — Ein Prozeß kann sich nicht zum gleichen Zeitpunkt in der Warteschlange mehrerer Semaphoren befinden.

richtig | FALSCH — Der Wert (`value`) eines binären Semaphors wird immer mit 0 initialisiert.

richtig | FALSCH — In der Warteschlange (`queue`) eines binären Semaphors kann sich immer nur höchstens 1 Prozeß befinden.

RICHTIG | falsch — Falls der Wert (`value`) eines Zählsemaphors negativ ist, also echt kleiner 0, so ist seine Warteschlange (`queue`) nicht leer.

richtig | FALSCH — Falls die Warteschlange (`queue`) eines Zählsemaphors leer ist, so ist sein Wert (`value`) echt größer 0.

RICHTIG | falsch — Befindet sich ein Prozeß in der Warteschlange (`queue`) eines Zählsemaphors, so kann er nicht gleichzeitig rechenwillig (`ready`) sein.

1.4 SQL

RICHTIG | falsch — Ein Relationenschema kann mehr als einen Fremdschlüssel enthalten.

richtig | FALSCH — Der Wertebereich eines Attributs muß mit einer Domain-Anweisung definiert werden.

richtig | FALSCH — Bei einer mit `create table ...` definierten Tabelle können zwar noch die Einträge verändert werden, nicht jedoch ihre Form, d.h. das zugrunde liegende Relationenschema.

RICHTIG | falsch — Durch geschachtelte SQL Anfragen kann man die Bildung großer Joins vermeiden.

richtig | FALSCH — Durch die Angabe einer `order by` Anweisung werden Einträge in der Datenbank sortiert, so daß sie zukünftig in sortierter Folge vorliegen.

2 HTML Aufgabe (14 Punkte)

Zwischen den folgenden horizontalen Linien befindet sich zuerst ein Text, wie er vom Browser Netscape 4.7 angezeigt wird. Anschließend ist ein HTML-Quelltext gegeben.

Allerdings passen der angezeigte Text und der HTML-Quelltext nicht ganz zusammen.

Ändern Sie den HTML-Quelltext so ab, daß durch ihn von Netscape 4.7 der angezeigte Text erzeugt wird.

Leider ist mir kein vernünftiger Text eingefallen.

Findet Ihr das
1. traurig 2. nicht so schlimm 3. verwerflich ?

1	2	3
	123	

Das war's auch schon: ◇ ◇

```
<html>
<head>
<title></title>
</head>
<body>

Leider ist mir <BOLD>kein</BOLD> vernünftiger Text eingefallen.
Findet Ihr das <BR>
<ol>
<li>1.  traurig <li>2.  nicht so schlimm <li>3.  verwerflich ?
</ol>
<table border="0">
<tr><td>1<td>2<td>3
<tr><td rowspan="3" align="center">123
</table>
Das war's auch schon:<br>
<input type="radio" name="FEHLER" checked value="x">
<input type="radio" name="FEHLER" value="y">

</body>
</html>
```

LÖSUNG: Aufgabe 2

Fehler:

1. `<BOLD>...</BOLD>` um 'kein' streichen
2. `<P>` nach '...vernünftiger Text eingefallen.' ergänzen.
3. HTML-tags `<ol> <li>...</ol>` streichen, nur Text stehen lassen
4. `colspan` statt `rowspan` in Tabelle
5. z.B. `<form action="mailto:myself" method="post">` ergänzen.
6. `<br>` nach 'Das war's auch schon:' streichen
7. `checked` streichen

Korrektur des Quelltexts: (Ergänzungen in [ ])

```
<html>
<head>
<title></title>
</head>
<body>

Leider ist mir <BOLD>kein</BOLD> vernünftiger Text eingefallen.[<P>]
Findet Ihr das <BR>
<ol>
<li>1.  traurig <li>2.  nicht so schlimm <li>3.  verwerflich ?
</ol>
<table border="0">
<tr><td>1<td>2<td>3
<tr><td [colspan]="3" align="center">123
</table>
Das war's auch schon:<br>
[<form action="mailto:myself" method="post">]
<input type="radio" name="FEHLER" checked value="x">
<input type="radio" name="FEHLER" value="y">

</body>
</html>
```

---

3 Relationenalgebra (22 Punkte)

Seien A, B, C, D, E, F verschiedene Attribute.
Sei R_1 eine Relation über den Attributen A, B und C und sei R_2 eine Relation über den Attributen C, D, E und F.
Dies ergibt anschaulich die folgenden Schemata : R_1 :

A	B	C
⋮	⋮	⋮

R_2 :

C	D	E	F
⋮	⋮	⋮	⋮

Gelten die folgenden Gleichheiten von Ausdrücken der Relationenalgebra?

Bewertung: 2 Punkt pro richtige Antwort, –2 Punkt pro falsche Antwort.
(Pro Teilaufgabe 3.1–2 können minimal 0 Punkte erworben werden.)

3.1 Teilaufgabe

RICHTIG falsch $\pi_{C,F}(\pi_{E,F,C,D}(R_2)) = \pi_{C,F}(R_2)$

RICHTIG falsch $\pi_{A,C}(R_1) \bowtie \pi_{C,D,E}(R_2) = \pi_{A,C,D,E}(R_1 \bowtie R_2)$

richtig FALSCH $\pi_{A,B}(R_1) \bowtie \pi_{C,D,F}(R_2) = \pi_{A,B,C,D,F}(R_1 \bowtie R_2)$

RICHTIG falsch $\pi_C(R_1) \bowtie \pi_C(R_2) = \pi_C(R_1 \bowtie R_2)$

RICHTIG falsch $\pi_C\Big(\pi_C(R_1) \bowtie \pi_C(R_2)\Big) = \pi_C(R_1 \bowtie R_2)$

richtig FALSCH $\pi_A(R_1) = \pi_A(R_1 \bowtie R_2)$

3.2 Teilaufgabe

RICHTIG falsch $\pi_C(R_1 \bowtie R_2) = \pi_C(R_1) \cap \pi_C(R_2)$

RICHTIG falsch $\pi_C(R_1) \bowtie \pi_C(R_2) = \pi_C(R_1) \cap \pi_C(R_2)$

richtig FALSCH $\pi_A(R_1) \bowtie \pi_F(R_2) = \pi_A(R_1) \cup \pi_F(R_2)$

richtig FALSCH $\pi_A(R_1) \bowtie \pi_A(\beta_{A \leftarrow F}(R_2)) = \pi_{A,F}(R_1 \bowtie R_2)$

RICHTIG falsch $\pi_{C,F}(R_2) \div \pi_C(R_1) =$
$\pi_F(\pi_{C,F}(R_2)) - \pi_F\Big(\Big(\pi_F(\pi_{C,F}(R_2)) \bowtie \pi_C(R_1)\Big) - \pi_{C,F}(R_2)\Big)$

4 Normalisierung (20 Punkte)

Durch folgende Definitionen wird eine Datenbank `klausurDB` erzeugt.
(identisch zu Aufgabe 5, nicht identisch mit der Datenbank in den Übungen.)

```
create table kunde
  ( kund_nr char(6), kund_name char(30), adresse char(30), ort char(12) );
create table personal
  ( persnr int, nachname char(20), vorname char(12),
    einsatz char(12), plz_p int, vorgesetzt int, gehalt int );
create table verkauf
  ( auftr_nr int, art_nr int, bestelldat char(8), persnr int, kund_nr char(6) );
create table ausgang
  ( auftr_nr int, art_nr int, menge int );
create table inventar
  ( art_nr int, plz_i int, art_bez char(30), lagerbest int, lagerort char(12) );
```

Die folgende Übersicht enthält für jede Tabelle der Datenbank `klausurDB` einen Schlüssel.

Tabelle	Schlüssel
kunde	kund_nr
personal	persnr
verkauf	auftr_nr
ausgang	auftr_nr, art_nr
inventar	art_nr, plz_i

Es gelten darüberhinaus die folgenden funktionalen Abhängigkeiten:

`plz_p` $\longrightarrow$ `einsatz` d.h., `plz_p` ist die Postleitzahl des `einsatz`-Ortes
`plz_i` $\longrightarrow$ `lagerort` d.h., `plz_i` ist die Postleitzahl von `lagerort`
`art_nr` $\longrightarrow$ `art_bez`

Es sei zudem angenommen, daß alle Relationen der Datenbank `klausurDB` bereits in erster Normalform sind.

4.1 Teilaufgabe

Bestimmen Sie für jede Relation der Datenbank `klausurDB`, ob sie in zweiter Normalform (2NF) ist und wenn ja, ob sie auch schon in dritter Normalform (3NF) ist.
Begründen Sie kurz Ihre Entscheidung.

Tabelle	2NF		3NF	
kunde	RICHTIG	falsch	RICHTIG	falsch
personal	RICHTIG	falsch	richtig	FALSCH
verkauf	RICHTIG	falsch	RICHTIG	falsch
ausgang	RICHTIG	falsch	RICHTIG	falsch
inventar	richtig	FALSCH	richtig	FALSCH

4.2 Teilaufgabe

Falls eine Relation nicht in zweiter oder in dritter Normalform ist, dann sind dafür Relationen in dritter Normalform anzugeben.
Geben Sie für die neu erstellten Tabellen jeweils einen Schlüssel an.

- – `personal*` : Schlüssel: `persnr`

  ```
  create table personal*
     ( persnr int, nachname char(20), vorname char(12),
       plz_p int, vorgesetzt int, gehalt int );
  ```

  – `plz_einsatz` : Schlüssel: `plz_p`

  ```
  create table plz_einsatz
      ( einsatz char(12), plz_p int );
  ```

- – `inventar*` : Schlüssel: `art_nr, plz_i`

  ```
  create table inventar*
     ( art_nr int, lagerbest int, plz_i int );
  ```

  – `plz_lagerort` : Schlüssel: `plz_i`

  ```
  create table plz_lagerort
     ( plz_i int,  lagerort char(12) ) ;
  ```

  – `nr_bez` : Schlüssel: `art_nr`

  ```
  create table nr_bez
     ( art_nr int, art_bez char(30) ) ;
  ```

5 Datenbankabfragen (26 Punkte)

Durch folgende Definitionen wird eine Datenbank `klausurDB` erzeugt.
(identisch zu Aufgabe 4, nicht identisch mit der Datenbank in den Übungen.)

```
create table kunde
  ( kund_nr char(6), kund_name char(30), adresse char(30), ort char(12) );
create table personal
  ( persnr int, nachname char(20), vorname char(12),
    einsatz char(12), plz_p int, vorgesetzt int, gehalt int );
create table verkauf
  ( auftr_nr int, art_nr int, bestelldat char(8), persnr int, kund_nr char(6) );
create table ausgang
  ( auftr_nr int, art_nr int, menge int );
create table inventar
  ( art_nr int, plz_i int, art_bez char(30), lagerbest int, lagerort char(12) );
```

Formulieren Sie die folgenden Anfragen in SQL und Relationenalgebra (RA).

1. Finden Sie den Nachnamen aller Angestellten, die einen Artikel mit der Bezeichnung 'SCHRANKWAND' verkauft haben.
   (Machen Sie eine geschachtelte SQL-Anfrage und selektieren Sie beim RA-Ausdruck so weit wie möglich vor den Join-Operationen.)

   SQL:

   ```
   select nachname
   from personal
   where persnr in
   (select persnr
   from verkauf
   where art_nr in
   (select art_nr
   from inventar
   where
   art_bez = 'SCHRANKWAND'
   ))
   ```

   RA:

   $\pi_{nachname}(personal \bowtie$
   $\pi_{persnr}(verkauf \bowtie$
   $\pi_{art\_nr}(\sigma_{art\_bez='SCHRANKWAND'}(inventar)))))$

2. Finden Sie Nachname und Vorname aller Angestellten deren Vorgesetzte(r) mindestens DM 5000 verdient.
(Machen Sie eine geschachtelte SQL-Anfrage und selektieren Sie so früh wie möglich.)

SQL:

```
select nachname, vorname
from personal
where vorgesetzt in
(select persnr from personal
where gehalt >= 5000)
```

RA:

$\pi_{nachname,vorname}(\beta_{Vp \leftarrow vorgesetzt}(personal) \bowtie$
$\beta_{Vp \leftarrow persnr, Vn \leftarrow nachname, Vv \leftarrow vorname, Ve \leftarrow einsatz, Vplz \leftarrow plz\_p, Vc \leftarrow vorgesetzt, Vg \leftarrow gehalt}$
$(\sigma_{gehalt \geq 5000}(personal)))$

3. Finden Sie die Personalnummern aller Angestellten, die weniger als einer ihrer Untergebenen verdienen.

SQL:

```
select vorg.persnr
from personal ang, personal vorg
where ang.vorgesetzt = vorg.persnr
and vorg.gehalt < ang.gehalt
```

RA:

$\pi_{Vp}(\sigma_{gehalt > Vg}(\beta_{Vp \leftarrow vorgesetzt}(personal) \bowtie$
$\beta_{Vp \leftarrow persnr, Vn \leftarrow nachname, Vv \leftarrow vorname, Ve \leftarrow einsatz, Vplz \leftarrow plz\_p, Vc \leftarrow vorgesetzt, Vg \leftarrow gehalt}(personal)))$

6 Synchronisation (36 Punkte)

Gegeben sind die beiden unten angegebenen Programme `put` und `get` sowie ein gemeinsamer Speicherbereich, welcher (unter anderem) einen Nachrichtenpuffer `p` sowie die Variable `anz` enthält.

Die Programme `put` und `get` werden von anderen Programmen aufgerufen und legen dann eine Nachricht `w` in den Puffer `p` — Aufruf `put(w)` — bzw. lesen eine Nachricht `w` aus dem Puffer `p` — Aufruf `get(w)`.

Die Nachrichten bestehen jeweils nur aus einem Buchstaben (`char`). Der Wert der Variablen `anz` sagt aus, wie viele ungelesene Nachrichten sich in Puffer `p` befinden. (Am Anfang ist der Nachrichtenpuffer also leer.)

Damit sich die beiden Programme `put` und `get` beim Schreiben und Lesen von Nachrichten nicht in die Quere kommen, wird wechselseitiger Ausschluß durch den Semaphor `wA` gewährleistet.

```
VAR wA : binarysemaphor;
wA.value = 1; wA.queue = [] ;
VAR p :  array[1..n] of char;
VAR anz :  Integer;
anz := 0;
```

```
put(w) :
  wait(frei);
  wait(wA);

  p[anz] := w;

  anz := anz + 1;

  signal(wA);
  signal(belegt);
END;
```

```
get(w):
  wait(belegt);
  wait(wA);

  w := p[anz];

  anz := anz - 1;

  signal(wA);
  signal(frei);
END;
```

6.1 Teilaufgabe

Wir nehmen an, daß beide Programme `put` und `get` aufgerufen worden sind und daß zugehörige (gleichnamige) Prozesse erzeugt worden sind, jedoch noch keiner der beiden Prozesse bisher den Prozessor bekommen hat. Ein Ablaufprotokoll würde also wie folgt initialisiert sein (0. Zeile bzw. 0. Schritt der Abarbeitung).

	put	get	wA	anz	'ready' (or 'running')
0.			(1,[])	0	put, get

Erstellen Sie alle möglichen Ablaufprotokolle über die nächsten 2 Schritte:

1.	wait(wA)		(0,[])	0	put, get
2.		wait(wA)	(0,[get])	0	put
1.		wait(wA)	(0,[])	0	put, get
2.	wait(wA)		(0,[put])	0	get
1.	wait(wA)		(0,[])	0	put, get
2.	p[anz] := w		(0,[])	0	put, get
1.		wait(wA)	(0,[])	0	put, get
2.		w := p[anz]	(0,[])	0	put, get

6.2 Teilaufgabe

Obige Programme leiden unter dem folgenden Pufferproblem: Zum einen kann aus dem Nachrichtenpuffer `p` auch dann gelesen werden, wenn keine Nachricht vorhanden ist (`anz=0`), und zum anderen kann eine Nachricht hineingeschrieben werden, wenn er schon voll ist (`anz`$\geq$`n`).

Lösen Sie dieses Pufferproblem durch die Einführung zweier Zählsemaphore.

Ergänzen Sie hierzu (an den unterstrichenen Stellen) die untenstehende Deklaration und Initialisierung — ähnlich der Deklaration und Initialisierung des Semaphors `wA` oben — der 2 Zählsemaphoren `frei` und `belegt`.

```
VAR frei :  semaphor;
frei.count = [n]; frei.queue = [ [] ] ;
VAR belegt :  semaphor;
belegt.count = [0]; belegt.queue = [ [] ] ;
```

Ändern Sie, unter Verwendung der beiden Semaphore `frei` und `belegt`, die obenstehenden Programme `put` und `get` so ab, daß das Pufferproblem gelöst ist. (Schreiben Sie dabei Ihre Änderungen direkt in obenstehende Programme.)

7 UNIX Aufgabe (4 Punkte)

Gegeben seien die folgenden beiden Dateien `DAT1` und `DAT2` welche jede aus 4 Zeilen bestehen, wobei in jeder Zeile nur 1 Buchstabe steht.

`DAT1`:

D
E
B
A

`DAT2`:

A
D
C
A

Durch das Unix Kommando

```
cat DAT1 DAT2 | uniq | sort > DAT3
```

wird eine Datei DAT3 erzeugt. Geben Sie diese an.

`DAT3`:

LÖSUNG: Aufgabe 7

```
: > cat DAT1
D
E
B
A
: > cat DAT2
A
D
C
A
: > cat DAT1 DAT2
D
E
B
A
A
D
C
A
: > cat DAT1 DAT2 | uniq
D
E
B
A
D
C
A
: > cat DAT1 DAT2 | uniq | sort
A
A
B
C
D
D
E
```