Klausur zu Unix-Datenbanken I

SS 1999

7. Juli 1999

Name ______________________________

Vorname ______________________________

Matrikel-Nr. ______________________________

Studiengang ______________________________

Aufgabe		Punkte	Bewertung
1		12	
2	a)	6	
	b)	6	
3		12	
4		6	
5		6	
6	a)	4	
	b)	6	
	c)	4	
7		8	
Summe		**70**	

☞ Es sind mindestens 35 Punkte zu erreichen!
☞ Hilfsmittel: alles ausser Hilfe durch Dritte und ausser elektr. Geräten
☞ Bearbeitungszeit: 120 Min.

Aufgabe 1

Der im Folgenden beschriebene Sachverhalt ist in ein ER-Diagramm gemäß Vorlesung (Entity-Typen, Beziehungen, Schlüssel, Komplexitätsgrade, 1NF, 2NF, 3NF) umzusetzen; Ziel ist die Entwicklung einer Datenbank im Auftrag Ihrer Fachschaft, die Ihre Prüfungsplanung (Haupt-Diplom) erleichtern soll.
Sollten Ihnen irgendwelche Angaben fehlen, so legen Sie Annahmen für das Diagramm zugrunde (die Sie natürlich mit angeben müssen) - Fragen an die Aufsicht sind nicht zweckmäßig und auch nicht zulässig.

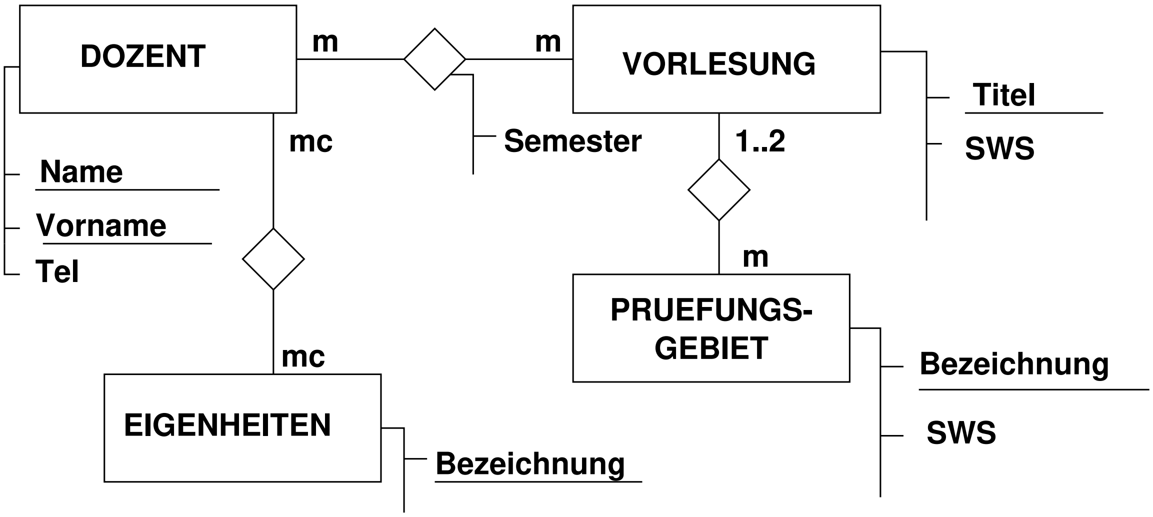
a) In der Prüfungsordnung Ihres Faches werden Prüfungsgebiete über eine eindeutige Bezeichnung definiert; zu jedem ist der Prüfungsumfang in Semesterwochenstunden festgelegt.

b) Vorlesungen werden durch ihren Titel definiert; der Umfang in Semesterwochenstunden bleibt - egal wann wer die Vorlesung liest - immer gleich. Die Inhalte einer bestimmten Vorlesung können sich ändern, von Semester zu Semester, von Dozent zu Dozent. In Ihrer Planung wollen Sie sich aber sowohl auf den Dozenten wie auch auf das Semester, in der dieser diese Vorlesung gehalten hat, festlegen.

c) Vorlesungen werden immer von genau einem Dozenten gelesen. Dozenten sind eindeutig über Name und Vorname identifiziert; zusätzlich wollen Sie zum Dozenten immer noch seine (einzige) Telefonnummer wissen.

d) Eine Vorlesung kann maximal zwei verschiedenen Prüfungsgebieten zugeordnet werden. Die Zuordnung ist ‘zentral’ festgelegt.

e) Manche Dozenten zeigen in den Prüfungen gewisse ‘Eigenheiten’, die Fachschaft weiß, welche das sind.



Aufgabe 2

Zugrundeliege folgende Tabelle:

Vorlesungen	**Id**	**Titel**	**SWS**

Dabei ist **Id** ein künstlich eingeführter Schlüssel (ursprünglich durchlaufende Numerierung von 1 in den natürlichen Zahlen aufwärts, durch Löschen von Einträgen inzwischen etwas lückenhaft). Der **Titel** ist im zugrundeliegenden ER-Diagramm als der Schlüssel definiert. **SWS** gibt die Semesterwochenstunden der Vorlesung an, was im Folgenden aber ohne Bedeutung ist.

Die Tupel dieser Tabelle stehen in einer Datei namens **VL**, die in keiner Weise sortiert ist; der Trenner ist der senkrechte Strich (|), die Struktur ist wie in der Tabelle festgelegt, die Attributzeile ist nicht in der Datei **VL** enthalten.

Beispiel: `cat VL` liefert
1|UDB1|6
7|UDB2|6
3|Algebra|6
4|Topologie|4
11|Zahlentheorie|2
5|Numerik|7

(*a*) Zum Einfügen einer neuen Vorlesung (über ein hier nicht zu erstellenden Bash-Skript) brauchen Sie die bisher größte vergebene **ID**. Diese soll einer Variablen **HighestId** direkt zugewiesen werden. Es dürfen nur allgemein existierende Kommandos wie *grep* &Co. verwendet werden; die Zuweisung muß in einer Kommandozeile erfolgen. Sollten Sie von Kommandos benötigte Optionen nicht parat haben, nehmen Sie eine Optionsbezeichnung Ihrer Wahl, erklären bitte aber, was Sie mit dieser erwarten.

```
HighestId=`cut -d'|' -f1 VL | sort -n | tail -1`
```

(*b*) Die Lücken in der Numerierung wollen Sie nun schließen, und zwar so, daß entsprechend der Sortierung nach **Titel** die Numerierung aufsteigend ist. Geben Sie hierzu ein Bash-Skript an, das die neu numerierten Datensätze in eine Datei namens **VL.new** schreibt.

```
i=1
cut -d'|' -f2,3 VL | sort | while read line
do
     echo  $i'|'$line
     i=`expr $i + 1`
done > VL.new
```

Aufgabe 3

Für das folgende liege folgende Tabellenstruktur zugrunde:

Prof_Vorlesung	**ProfId**	**VorlesungsId**	**Semester**

Die Tupel seien in dieser Struktur mit dem Doppelpunkt als Trenner in der Datei **ProfVorl** abgelegt, die Attribute sind nicht eingetragen. Wie viele andere sehen auch wir dem Jahr 2000 gelassen entgegen, obwohl wir die Jahresangaben in der Spalte **Semester** auch nur zweistellig führen (als z.B. SS99, WS 99/00). Wir haben uns nämlich für die sog. Schwellwert-Technik entschieden: Jahresangaben kleiner als 50 werden in das 21. Jahrhundert (also 20XX), solche größer oder gleich 50 ins 20. Jahrhundert transponiert (also 19XX).

Beispiel: `cat ProfVorl` liefert
10:101:WS99/00
10:102:WS87/88
10:103:SS05
12:101:SS17
12:104:WS22/23

Bevor aber das Jahr 2050 naht, bekommen Sie die Aufgabe, alle Jahresangaben korrekt zur Schwellwert-Interpretation in 4-stellige Jahreszahlen umzusetzen - und dies in gekonnter Manier mit der **tcl-Shell**, egal wie, aber es soll ein ausführbares Programm sein!

Beispiel: WS99/00 wird zu WS1999/2000

```
#!/usr/local/bin/tclsh

set f [open ProfVorl r]

while { [ gets $f line] >= 0 } {
     regexp {^([^:]*):([^:]*):([WS]*)([0-9]*)([/]?)(.*)$} $line match \
          f1 f2 f3 f4 f5 f6

     if { $f4 < 50  } {
          set f4 [expr 2000 + $f4]
     } else {
          set f4 [ expr 1900 + $f4]
     }
     if { [ string length "$f6"] > 0 } {
          if { $f6 < 50  } {
               set f6 [expr 2000 + $f6]
          } else {
               set f6 [ expr 1900 + $f6]
     }    }
     puts stderr "$f1:$f2:$f3$f4$f5$f6"
}
```

Aufgabe 4

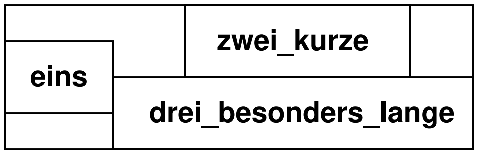
Gegeben sei folgendes tcl/Tk-Skript:

```
#!/bin/sh
# the next line restarts using wish \
exec wsh "$0" "$@"

# Hauptfenster "." auf schwarz setzen:
 . config -bg black

label .eins -bg white -text eins
pack .eins -side left
label .zwei -bg white -text zwei_kurze
pack .zwei -side top
label .drei -bg white -text drei_besonders_lange
pack .drei -side top
```

Geben Sie in einer Skizze an, wie das erzeugte Window (ohne Rahmen des Window Manager) aussieht!



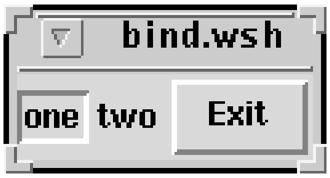
Aufgabe 5

Gegeben sei folgendes kleines tcl/Tk-Skript:

```
#!/bin/sh
# the next line restarts using wish \
exec wsh "$0" "$@"

label .one -text one
label .two -text two
button .exit -text "Exit" -command exit
pack .one .two .exit -side left
bind .one  <Control-x> { pack forget %W }
bind all <Enter> { focus .one }
```

Dieses erzeugt folgende Oberfläche:



Welche der folgenden Aussagen sind richtig, welche falsch (bitte Zutreffendes klar ankreuzen)? Es ist immer von der Start-Darstellung auszugehen! Falsche Antworten führen zu Punktabzug!

	richtig	falsch
Mauszeiger auf ‘Exit’, Eingabe von <ctrl-x> => Anwendung terminiert		X
Mauszeiger auf ‘Exit’, Eingabe von <ctrl-x> => Element ‘one’ verschwindet	X	
Mauszeiger auf ‘Exit’, Eingabe von <ctrl-x> => Elemente ‘one’, ‘two’ und ‘Exit’ verschwinden		X
Mauszeiger auf ‘two’, Eingabe von <ctrl-x> => Element ‘one’ verschwindet	X	
Mauszeiger auf Titelleiste ‘bind.wsh’, Eingabe von <ctrl-x> => Anwendung terminiert		X
Mauszeiger auf Titelleiste ‘bind.wsh’, Eingabe von <ctrl-x> => Elemente ‘one’, ‘two’ und ‘Exit’ verschwinden		X

Aufgabe 6

Gegeben sei folgendes Bash-Skript auf der thales (Inhalt der ausführbaren Datei *script*):

```
#!/usr/local/bin/bash

i=0
while read line; do
   i=`expr $i + 1`
done < xyz
echo $i

i=0
cat xyz | while read line; do
   i=`expr $i + 1`
done
echo $i
```

Die im Skript bearbeitete Datei xyz habe folgenden Inhalt:

```
Zeile1
Zeile2
Zeile3
```

(*a*) Das Programm erzeugt folgende Ausgabe:

```
3
0
```

Erklären Sie, wie es zu den unterschiedlichen Werten kommt.

Die erste Ausgabe (3) wird in der ausführenden Shell (kein neuer Prozess für die Schleife) erzeugt. Die zweite Ausgabe (0) wird auch von der ausführenden Shell erzeugt, die Schleife läuft wegen der Pipe in einem neuen Prozess, der Wert von i, der ausgegeben wird, ist nicht der, der in der Schleife erzeugt wird.

(*b*) Skizzieren Sie ein C-Programm, um zu erklären, wie die Bash den Programmteil

```
while read line; do
   i=`expr $i + 1`
done < xyz
```

möglicherweise realisiert.

Hinweis: Schreiben Sie **kein** komplettes C-Programm.

Ihre Aufgabe ist es zu demonstrieren, welchen Unterschied der Zusatz **< xyz** am Ende der While-Schleife ausmacht und wie dieser Zusatz implementiert werden kann, um das in Teil a) beobachtete Verhalten zu erzielen.

Die C-Implementierung der While-Schleife sei gegeben, deuten Sie sie als Kommentar an: /* Hier kommt die While-Schleife...*/

```
int fd_stin;
char * file = "xyz";

fd_stdin = dup(0);
close(0);
if (open(file, O_RONLY) == -1)
     return -1;

/* While-Schleife */

close(0);
dup(fd_stdin);
close(fd_stdin);
```

(*c*) **Erweitern** Sie folgendes Bash-Skript so, daß die Variable *i* in der letzten Anweisung (*echo $i*), den Wert enthält, den die Variable i innerhalb der while-Schleife zuletzt hatte.

```
i=0
cat xyz | while read line; do
   i=`expr $i + 1`
done
echo $i
```

```
i=0
i=`cat xyz | { while read line; do; i=`expr $i + 1`; done  ; echo $i }`
echo $i
```

Aufgabe 7

Schreiben Sie ein Bash-Skript, das feststellt, ob in einer Tabelle (Datei *file* mit Doppelpunkt als Trenner) eine Spalte leer ist, also keinerlei Zeichen enthält. Trenner der Tupel ist *newline*.

Die Zahl der Spalten der Tabelle kann im Skript über eine Variablenzuweisung definiert werden (`maxSp=50`), die Spalten gelten als von 1 an durchnumeriert; die Datei *file* kann im Skript fest 'verdrahtet' werden.

Welche Spalte überprüft werden soll, wird über das Argument des Skriptes definiert (also genau ein Argument). Ob diese Zahl im zulässigen Bereich [1..maxSp] liegt, muß nicht überprüft werden. Bei falscher Argumentzahl (0 oder mehr als 1) muß die übliche Usage-Meldung auf stderr ausgegeben werden. Das Skript soll auf stdout entweder *Spalte XXX ist in mindestens einem Tupel leer* oder *Spalte XXX ist in keinem Tupel leer* ausgeben.

Also: auf stdout darf nur eine der obigen Ausgaben, auf stderr allenfalls die Usage-Meldung erscheinen.

Hinweis: **egrep** kann prima suchen!

```
#!/usr/local/bin/bash

if [ $# -ne 1 ]
then
     echo "Usage: $0 zahl" >&2
     exit 1
fi
maxSp=50
zahl=$1

i=1
muster='^'
spalte='[^:]*:'
while [ $i -lt $zahl ]
do
muster=$muster$spalte
i=`expr $i + 1`
done
if [ $zahl -lt $maxSp ]
then
     muster=$muster':'
else
     muster=$muster'$'
fi
echo $muster
if egrep "$muster" file >/dev/null
then
     echo "Spalte $zahl ist in mindestens einem Tupel leer"
else
     echo "Spalte $zahl ist in keinem Tupel leer"
fi
exit 0
```