Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr 2018

66116

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Datenbanksysteme, Softwaretechnologie**

Anzahl der gestellten Themen (Aufgaben): **2**

Anzahl der Druckseiten dieser Vorlage: **21**

Bitte wenden!

Thema Nr. 1

Teilaufgabe 1

1. Grundlagen

Beantworten Sie die folgenden Fragen und begründen oder erläutern Sie Ihre Antwort in jeweils ein bis zwei Sätzen.

a) Was versteht man unter Selektivität?

b) Worin unterscheidet sich SQL von Programmiersprachen wie Java, C, C++, . . . und welche Vorteile ergeben sich daraus?

c) Wieso ist es wichtig, dass eine Transaktion atomar ausgeführt wird?

d) Welches Problem könnte auftreten, wenn Transaktionen nicht isoliert ablaufen würden?

e) Ist es richtig, dass die Ausführung von Anfragen durch die Verwendung von Sichten beschleunigt wird, weil dadurch weniger Daten von der Festplatte gelesen werden müssen?

f) Kann auf jeder Sicht eine SQL-Insert-Anweisung ausgeführt werden?

g) Kann die Performance von Anfragen durch Redundanzen im Datenbestand verbessert werden?

2. ER-Modellierung

Im Folgenden finden Sie die Beschreibung der Verwaltung eines Verkehrsunternehmens. Erstellen Sie zu dieser Beschreibung ein erweitertes ER-Diagramm. Kennzeichnen Sie die Primärschlüssel durch passendes Unterstreichen und geben Sie die Kardinalitäten in Chen-Notation (= Funktionalitäten) an. Kennzeichnen Sie auch die totale Teilnahme (= Existenzabhängigkeit, Partizipität) von Entitytypen.

Das Unternehmen betreibt Fahrzeuge. Diese sind durch eine Fahrgestellnummer eindeutig identifizierbar. Weiterhin haben Sie eine maximale Fahrgastzahl. Jedes Fahrzeug ist entweder ein Bus, eine Straßenbahn oder eine U-Bahn. Nur Busse haben eine Treibstoffart.

Weiterhin werden Mitarbeiter verwaltet. Diese verfügen über eine eindeutige Personalnummer, einen Nachnamen und einen Vornamen.

Zu Fahrzeugen werden Inspektionen verwaltet. Eine Inspektion hat ein für das jeweilige Fahrzeug eindeutiges Inspektionsdatum und wird von ein oder mehreren Mitarbeitern durchgeführt.

Das Verkehrsunternehmen bedient verschiedene Haltestellen, die alle einen eindeutigen Namen haben. Außerdem wird zu ihnen gespeichert, ob sie ein Wartehäuschen besitzen.

Fortsetzung nächste Seite!

Es gibt verschiedene Linien. Eine Linie hat eine eindeutige Nummer und beinhaltet ein oder mehrere Haltestellen. Es werden auch die Position einer Haltestelle auf einer Linie gespeichert sowie die Fahrzeit von der vorhergehenden Haltestelle auf der Linie. Pro Linie wird jede Haltestelle maximal einmal angefahren. Jede Linie wird entweder von genau einem Bus, einer Straßenbahn oder einer U-Bahn bedient.

Mitarbeiter können in einer Schicht mit einem bestimmten Fahrzeug auf einer bestimmten Linie fahren. Eine Schicht ist eindeutig gekennzeichnet durch ihren Startzeitpunkt und hat außerdem einen Endzeitpunkt. In jeder Schicht führt genau ein Mitarbeiter die Verkehrsaufsicht.

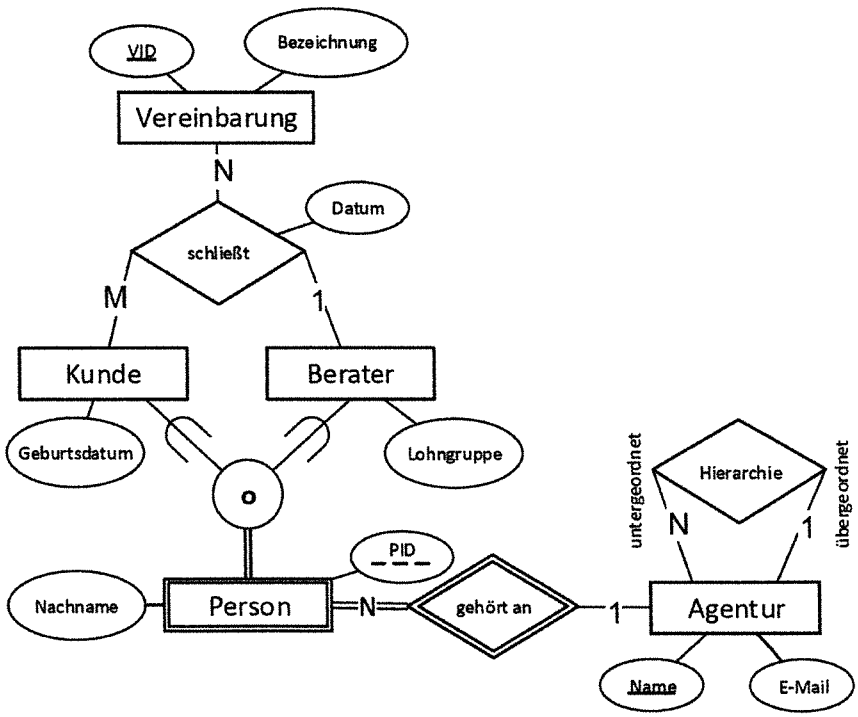
3. Relationaler Entwurf

Entwerfen Sie zum untenstehenden ER-Diagramm ein Relationenschema in dritter Normalform (3 NF) mit möglichst wenig Relationen.

Verwenden Sie dabei folgende Notation: Primärschlüssel werden durch Unterstreichen gekennzeichnet, Fremdschlüssel durch die Nennung der Relation, auf die sie verweisen, in eckigen Klammern hinter dem Fremdschlüsselattribut. Attribute zusammengesetzter Fremdschlüssel werden durch runde Klammern als zusammengehörig markiert.

Beispiel:

```
Relation1(Primärschlüssel, Attribut1, Attribut2,
Fremdschlüsselattribut1[Relation1],
(Fremdschlüssel2_Attribut1, Fremdschlüssel2_Attribut2)[Relation2])
```



Fortsetzung nächste Seite!

4. SQL

Gegeben sind folgende Relationen aus einer Bibliotheksverwaltung:

```
Buch(ISBN, Titel, ID[Autor], Erscheinungsjahr)
Autor(ID, Nachname, Vorname, Geburtsdatum)
Referenz(Referenzierendes_Buch[Buch], Referenziertes_Buch[Buch])
```

Verwenden Sie im Folgenden nur Standard-SQL und keine produktspezifischen Erweiterungen. Sie dürfen bei Bedarf Views anlegen. Geben Sie einen Datensatz nicht mehrfach aus.

a) Schreiben Sie eine SQL-Anweisung, die die Tabelle Buch (inklusive Integritätsbedingungen) anlegt. Beim Löschen eines Autors sollen auch all dessen Bücher gelöscht werden. Verwenden Sie sinnvolle Datentypen.

b) Schreiben Sie einen Trigger, der beim Einfügen neuer Autoren den Nachnamen auf die Zeichenkette „NN" setzt, falls dieser NULL ist.

c) Schreiben Sie eine SQL-Anfrage, die die Titel aller Bücher von Autoren listet, die vor dem 1.1.1990 geboren wurden.

d) Schreiben Sie eine SQL-Anfrage, die zu einem Buch die Anzahl der Bücher auflistet, in denen es referenziert wird. Geben Sie nur Bücher aus, die in weniger als fünf Büchern referenziert wurden. Ausgegeben werden sollen der Titel des Buchs und die Anzahl der Referenzen, absteigend sortiert nach der Anzahl an Referenzen. Achten Sie darauf, dass auch Bücher, die nicht referenziert werden, ausgegeben werden.

e) Ermitteln Sie alle Autoren, die sich hauptsächlich selbst referenzieren. Schreiben Sie dazu eine SQL-Anfrage, die ID und Nachname aller Autoren liefert, in deren Büchern die Referenzen im Durchschnitt zu mindestens 90% auf eigene Bücher des Autors verweisen.

f) Schreiben Sie eine SQL-Anweisung, die alle Bücher löscht, die weder andere Bücher referenzieren noch selbst referenziert werden.

g) Nennen Sie die beiden SQL-Befehle zur Vergabe und zum Entzug von Rechten.

5. Relationale Algebra

Geben Sie an ob der folgende Zusammenhang in der relationalen Algebra gilt und beweisen Sie Ihre Aussage.

$$\pi_A(R - S) = \pi_A(R) - \pi_A(S)$$

mit:
R, S: Relationen mit gleichen Attributnamen und Attributtypen (= Domänen)
A: Teilmenge der Menge der Attribute von R oder S

Umgangssprachlich: Dürfen Projektion und Mengendifferenz vertauscht werden?

Fortsetzung nächste Seite!

6. Normalformen

a) Erläutern Sie in zwei bis drei Sätzen die beiden grundlegenden Eigenschaften, die bei der Aufspaltung eines Relationenschemas gelten müssen.

b) Gegeben ist die Relation Verbräuche (Gebäudenummer, Straße, Hausnummer, PLZ, Jahr, Wasserverbrauch, Stromverbrauch) mit den beiden Schlüsselkandidaten (Gebäudenummer, Jahr) und (Straße, Hausnummer, PLZ, Jahr). Die Attribute Wasserverbrauch und Stromverbrauch enthalten die entsprechenden Verbrauchswerte für ein bestimmtes Gebäude in einem bestimmten Jahr. Alle Attributwerte sind atomar. Geben Sie die höchste Normalform an, die die Relation Verbräuche erfüllt. Zeigen Sie, dass alle Bedingungen für diese Normalform erfüllt sind und dass mindestens eine Bedingung der nächsthöheren Normalform verletzt ist. Beziehen Sie sich bei der Begründung auf die gegebene Relation und nennen Sie nicht nur die allgemeinen Definitionen der Normalformen.

7. Optimierung

a) Erläutern Sie den Unterschied zwischen logischer (= algebraischer) und physischer Optimierung in drei bis vier Sätzen.

b) Übertragen Sie folgendes SQL-Statement in einen *nicht optimierten* algebraischen Ausdruck (= Anfragegraph, Operatorgraph).

```
select Kund.Vname, Kund.Nname
from Kund, Rechng
where Kund.ID = Rechng.Kund
and Rechng.Sum > 500;
```

c) Nennen Sie zwei Möglichkeiten, den algebraischen Ausdruck aus der vorhergehenden Teilaufgabe logisch (= algebraisch) zu optimieren. Beziehen Sie sich auf konkrete Stellen und Operatoren des von Ihnen aufgestellten algebraischen Ausdrucks.

Fortsetzung nächste Seite!

Teilaufgabe 2

Aufgabe 1: „Formale Spezifikation Abstrakter Datentypen (ADT)"

Hinweis: Sie dürfen in dieser Aufgabe alternativ auch andere *formale* Schreibweisen wie zum Beispiel UML/OCL oder die Z-Notation verwenden.

WICHTIG: In dieser Aufgabe stehen **keine** anderen Datentypen (also **kein int**, **String** usw.), **keine** anderen Konstanten (auch **nicht** 0, 1 usw.) und **keine** anderen Operationen (also **kein** <, +, ==, ≠ usw.) zur Verfügung: Sie dürfen ausschließlich die vorgegebenen oder zu ergänzenden **adt**s mit deren **ops** verwenden.

Gegeben seien die folgenden Gerüste der ADTs *Nat* (Repräsentation natürlicher Zahlen $\mathbb{N}_0^+$) und *NatStack* (Stapel mit *Nat*-Elementen):

adt *Nat*
sorts *Nat*
ops

NaN:		→ *Nat*	*// „Not a Nat" – vergleichbar mit Double.NaN*
zero:		→ *Nat*	*// entspricht der natürlichen Zahl „0"*
succ:	*Nat*	→ *Nat*	*// Nachfolger „(x + 1)" der Nat-Zahl x*
add:	*Nat* × *Nat*	→ *Nat*	*// Addition im Nat-Raum*
sub:	*Nat* × *Nat*	→ *Nat*	*// Subtraktion im Nat-Raum (z.B. „42 − 666 = 0")*
mul:	*Nat* × *Nat*	→ *Nat*	*// Multiplikation im Nat-Raum*

axs

succ(NaN) = NaN *// alle Operationen mit NaN ergeben stets wieder NaN*
// weitere Axiome aus Platzgründen weggelassen

end *Nat*

---

adt *NatStack*
sorts *NatStack, Nat*
ops

empty:		→ *NatStack*	*// erzeugt einen neuen leeren Stapel*
push:	*NatStack* × *Nat*	→ *NatStack*	*// legt ein Nat **auf** den Stapel*
peek:	*NatStack*	→ *Nat*	*// gibt bei leerem Stapel NaN und bei // nicht-leerem Stapel das „**oberste**" Nat zurück*
pop:	*NatStack*	→ *NatStack*	*// gibt bei leerem Stapel empty und bei nicht-leerem // Stapel den Stapel ohne das „**oberste**" Nat zurück*

axs

// Axiome aus Platzgründen weggelassen

end *NatStack*

---

a) Ergänzen Sie den ADT *Nat* um die Axiome des Vergleichsoperators *eq*, der genau dann *succ(zero)* zurück gibt, wenn die Operanden die gleiche *Nat*-Zahl darstellen – sonst *zero*, d.h. der Rückgabewert ist boolesch codiert: false ≡ 0 ≡ *zero* bzw. true ≡ 1 ≡ *succ(zero)*. Beachten Sie, dass alle Operationen *NaN* zurückgeben müssen, falls mind. einer der Operanden *NaN* ist. *Nicht vergessen:* Sie dürfen **keine** Vergleichsoperatoren wie ==, ≠ usw. verwenden – auch **nicht** auf der „rechten Seite" der Axiome (in „falls/sonst"-Bedingungen).

---

ops

eq: *Nat* × *Nat* → *Nat* *// succ(zero), falls die Operanden gleich sind; zero sonst*

Fortsetzung nächste Seite!

b) Ergänzen Sie den ADT *NatStack* um die Axiome der Operation *stackMul* gemäß Kommentar. Beachten Sie auch hier alle bisher genannten Einschränkungen.

ops

stackMul: NatStack $\rightarrow$ *Nat* // *multipliziert alle auf dem Stapel liegenden Nat-Zahlen*

c) Ergänzen Sie den ADT *NatStack* um die Axiome der Operation *stackPairOp* gemäß Kommentar bzw. Beispiel. Die Zahlen im Stack werden paarweise von „oben nach unten" betrachtet und genau dann durch ihr Produkt ersetzt, wenn sie gleich sind ($\rightarrow$ Quadratzahl) – andernfalls durch ihre Summe. Sie dürfen annehmen, dass die Anzahl der Elemente im Stack beim Aufruf der Operation geradzahlig ist. Alle Einschränkungen gelten weiterhin.

ops

stackPairOp: NatStack $\rightarrow$ *NatStack* // *multipliziert gleiche bzw. addiert ungleiche*
// *benachbarte Zahlen auf dem Stack und*
// *ersetzt sie mit dem Ergebnis der Operation*

$$Beispiel: \begin{Bmatrix} \oplus & 47_{(Nat)} \\ \oplus & 11_{(Nat)} \\ \otimes & 11_{(Nat)} \\ \otimes & 11_{(Nat)} \\ \oplus & 11_{(Nat)} \\ \oplus & 655_{(Nat)} \end{Bmatrix} \xrightarrow{stackPairOp} \begin{Bmatrix} \oplus & 58_{(Nat)} \\ \otimes & 121_{(Nat)} \\ \oplus & 666_{(Nat)} \end{Bmatrix}$$

Zur Vereinfachung wurden die *Nat*-Zahlen hier mit Ziffern ausgedrückt,
d.h. $3_{(Nat)}$ entspräche *succ(succ(succ(zero)))*

d) Ergänzen Sie den ADT *NatStack* um die Axiome der folgenden Operationen, so dass der *NatStack* wie eine *Deque* von beiden Seiten bearbeitet werden kann. Führen Sie dazu die Operationen *put/get/poll* rekursiv und ausschließlich auf *empty/push/peek/pop* zurück:

ops

put: *Nat* $\times$ *NatStack* $\rightarrow$ *NatStack* // *fügt ein Nat ganz* ***unten*** *in den Stapel ein*
get: *NatStack* $\rightarrow$ *Nat* // *gibt bei leerem Stapel NaN und bei*
// *nicht-leerem Stapel das „****unterste****" Nat zurück*
poll: *NatStack* $\rightarrow$ *NatStack* // *gibt bei leerem Stapel empty und bei nicht-leerem*
// *Stapel den Stapel ohne das „****unterste****" Nat zurück*

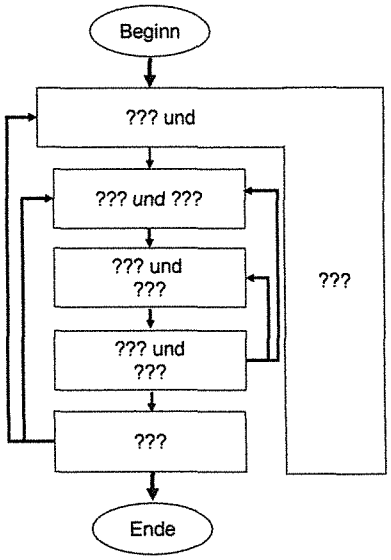
Fortsetzung nächste Seite!

Aufgabe 2: „Prozesse und Rollen“

a) Die Zertifizierungsstelle für Softwaretester ISTQB (International Software Testing Qualifications Board) definiert den Prozess des formalen Reviews (Inspektion) mit sechs Hauptphasen, während der Standard IEEE-1028 etwas genauer unterteilt und acht (auf den ISTQB-Prozess leicht abbildbare) Schritte vorsieht. Nennen und beschreiben Sie kurz jede einzelne Phase bei einem der beiden Standards.

b) Der Standard IEEE-1028-2008 definiert für ein Technisches Review vier Hauptrollen. Nennen Sie den jeweiligen Personenkreis zusammen mit deren jeweils wichtigsten Aufgaben.

c) Das ISTQB definiert den sogenannten „fundamentalen Testprozess“ und stellt seinen Ablauf wie folgt dar:



Übernehmen Sie die Skizze, ergänzen Sie die fehlenden Hauptaktivitäten („???“) und beschreiben Sie jede dieser Aktivitäten kurz.

Fortsetzung nächste Seite!

Aufgabe 3: „Entscheidungstabellentest“

Zu den funktionalen Testverfahren gehört der sog. Entscheidungstabellentest. Dabei werden die Informationen aus der Spezifikation systematisch in einer Tabelle erfasst, woraus sich die einzelnen Testfälle direkt ablesen lassen. Die Tabelle hat dabei folgende Grundstruktur:

Bedingungen	Regeln
Aktionen	Aktionsanzeiger

a) Skizzieren Sie kurz die vier Quadranten der Entscheidungstabelle.

b) Extrahieren Sie aus folgender Spezifikation die zugehörige Entscheidungstabelle für den Test.

„Das Warenwirtschaftssystem BitShop ist für kleine Online-Shops mit genau einem Produkt gedacht. Es verwaltet automatisch den aktuellen Lagerbestand des Artikels, bietet eine Web-Oberfläche für Bestellungen durch Kunden des Shops und generiert bei Bedarf Berichte für den Betreiber oder Bestellungen an Lieferanten. Dabei gelten folgende Geschäftsregeln:

- *Die über die Web-Oberfläche eingegebene Bestellmenge muss > 0 sein, andernfalls wird eine Fehlermeldung ausgegeben und die Bestellung storniert.*
- *Es sind keine Teillieferungen zugelassen. Falls der Lagerbestand für die aktuelle Bestellung zu niedrig ist, wird die Bestellung ebenfalls zurückgewiesen.*
- *Sinkt der Lagerbestand bei einer Bestellung unter ein vorgegebenes Minimum, dann wird automatisch eine Nachbestellung an den Lieferanten erstellt.“*

c) Geben Sie exemplarisch vier logische Testfälle an, die Sie aus Ihrer Tabelle abgeleitet haben.

Thema Nr. 2

Teilaufgabe 1

Aufgabe 1: Allgemeine Fragen

a) Nennen Sie die 4 Bestandteile des ACID-Prinzips und erläutern Sie diese kurz.

b) Was versteht man unter einer Relation? (Es genügt eine Definition als Antwort.)

c) Gegeben seien zwei Relationen *R(A,B,C)* und *S(C,D,E)*. Die Relation *R* enthalte 20 Tupel und die Relation S enthalte 10 Tupel. Sei p ein beliebiges Selektionsprädikat.
Gegeben seien außerdem folgende Anfragen:

Q_1:

```
SELECT *
FROM R,S
WHERE p;
```

Q_2:

```
SELECT DISTINCT A
FROM R,S;
```

- Wieviele Ergebnistupel liefert die Anfrage Q_1 mindestens?
- Wieviele Ergebnistupel liefert die Anfrage Q_1 höchstens?
- Wieviele Ergebnistupel liefert die Anfrage Q_2 mindestens?
- Wieviele Ergebnistupel liefert die Anfrage Q_2 höchstens?

d) Nennen Sie drei mögliche Anomalien, die bei einem unkontrollierten Mehrbenutzerbetrieb auftreten können.

e) Welche Anomalie tritt bzgl. welcher Variable in dem folgenden Schedule der Transaktionen T_1 und T_2 auf?

$$S_1 = (r_1(x), r_2(y), w_2(x), r_1(z), r_1(x), w_2(y), w_1(z))$$

f) Gegeben seien die Transaktionen T_1, T_2 und T_3. Wie viele mögliche verschiedene **serielle** Schedules gibt es?

g) Was unterscheidet eine materialisierte View von einer gewöhnlichen View?

h) Geben Sie ein Beispiel für eine transitive Abhängigkeit.

i) Geben Sie ein Beispiel für eine reflexive Abhängigkeit.

Fortsetzung nächste Seite!

Aufgabe 2: Relationales Modell

Das Fremdenverkehrsamt will sich einen besseren Überblick über Zirkusse verschaffen. In einer Datenbank sollen dazu die Zirkusse, die angebotenen Vorstellungen, die einzelnen Darbietungen in einer Vorstellung sowie die zugehörigen Dompteure und Tiere verwaltet werden.

Ein Zirkus wird eindeutig durch seinen Namen gekennzeichnet und hat einen Besitzer. Vorstellungen haben eine eindeutige VorstellungsID und ein Datum. Darbietungen haben neben der eindeutigen ProgrammNr eine Uhrzeit. Ein Dompteur hat eine eindeutige AngestelltenNr sowie einen Künstlernamen. Tiere sind eindeutig durch eine TierNr bestimmt und haben außerdem eine Bezeichnung der Tierart.

Vorstellungen werden von genau einem Zirkus angeboten. Ein Zirkus bietet mehrere Vorstellungen an und stellt mehrere Dompteure an. Ein Dompteur ist genau bei einem bestimmten Zirkus angestellt. Eine Darbietung findet in einer bestimmten Vorstellung statt, während sich eine Vorstellung aus mehreren Darbietungen zusammensetzt. Des Weiteren trainiert ein Dompteur mehrere Tiere, ein Tier kann allerdings auch von mehreren Dompteuren trainiert werden. In einer Darbietung tritt genau ein Dompteur mit mindestens einem Tier auf. Ein Dompteur kann in mehreren Darbietungen auftreten.

Erstellen Sie ein Entity-Relationship-Diagramm für obige Datenbank.

Aufgabe 3: Relationale Algebra

Gegeben seien folgende Relationen:

X

A	B	C	D
1	4	6	3
6	2	5	1
4	5	6	3
1	4	5	2
1	4	6	2
3	3	5	2
4	3	1	2
4	5	6	2

Y

C	D	E
5	2	3
6	2	1
6	3	1

Z

D	F
6	2
5	6
1	3

Fortsetzung nächste Seite!

Geben Sie die Ergebnisrelationen folgender Ausdrücke der relationalen Algebra als Tabellen an.

$$\sigma_{B>3}(X) \underset{C=F}{\bowtie} \sigma_{F>4}(\pi_F(Z))$$

$$\pi_{C,D}(X) - (\pi_C(Y) \times \pi_D(Z))$$

$$\pi_{A,C}(X) \div \pi_C(Y)$$

Aufgabe 4: Tupelkalkül

Gegeben sei das folgende Datenbank-Schema, das für die Speicherung der Daten einer Universität entworfen wurde, zusammen mit einem Teil seiner Ausprägung. Die Primärschlüssel-Attribute sind jeweils unterstrichen.
Die Relation *Dozent* enthält allgemeine Daten zu den Dozentinnen und Dozenten.
Dozentinnen und Dozenten halten Vorlesungen, die in der Relation *Vorlesung* abgespeichert sind.
Wir gehen davon aus, dass es zu jeder Vorlesung genau einen Dozenten (und nicht mehrere) gibt.
Zusätzlich wird in der Relation *Vorlesung* das Datum gespeichert, an dem die Klausur stattfindet.
In der Relation Student werden die Daten der teilnehmenden Studierenden verwaltet,
während die Relation *besucht* Auskunft darüber gibt, welche Vorlesung von welchen Studierenden besucht wird.

Dozent

DNR	DVorname	DNachname	DTitel
1	Christina	Müller	Prof. Dr.
2	Sabine	Schulz	Prof. Dr.
3	Herbert	Schmidt	PD Dr.
4	Adelheit	Berger	PD Dr.
...	...	...	...

Vorlesung

VNR	VTitel	Klausurtermin	Dozent
1	Datenbanksysteme I	29.01.2018	3
2	Einführung in die Programmierung	30.01.2018	2
3	Index- und Speicherstrukturen	29.01.2018	1
4	Knowledge Discovery in Databases I	05.02.2018	4
5	Softwareentwicklungspraktikum	22.02.2018	3
6	Datenbankpraktikum	11.02.2018	4
...	...	...	...

Student

Matrikelnummer	SVorname	SNachname	Semesterzahl
10000000	Frederik	Frühling	3
10000001	Valerie	Winter	12
...	...	...	...

besucht

Student	Vorlesung
10000000	3
...	...

Fortsetzung nächste Seite!

Dozent (DNR, DVorname, DNachname, DTitel)
Vorlesung (VNR, VTitel, Klausurtermin, Dozent)
Student (Matrikelnummer, SVorname, SNachname, Semesterzahl)
besucht (Student, Vorlesung)

Formulieren Sie die folgenden Anfragen im Tupelkalkül.
Datumsvergleiche können Sie mit $>$, $\geq$, $<$, $\leq$ oder $=$ angeben:

a) Geben Sie die Vornamen aller Studierenden aus, die die Vorlesung „Datenbanksysteme I" besuchen oder besucht haben.

b) Geben Sie die Matrikelnummern der Studierenden an, die keine Vorlesung mit einem Klausurtermin nach dem 31.12.2017 besuchen oder besucht haben.

c) Geben Sie die Matrikelnummern der Studierenden aus, die alle Vorlesungen von Prof. Dr. Schulz hören oder gehört haben.

Aufgabe 5: Anfragen

Gegeben sei das folgende Datenbankschema. Die Attribute Preis und Menge seien ganzzahlig, alle weiteren Attribute seien Zeichenketten. Im folgenden Schema sind Primärschlüssel unterstrichen und Fremdschlüssel überstrichen, wobei die von Fremdschlüsseln referenzierte Relation in eckigen Klammern nach dem Fremdschlüsselattribut angegeben ist.

```
Lieferant(LNr, LName, LStadt)
Ware(WNr, Bezeichnung, Preis)
Kunde(KNr, KName, KStadt)
Auftrag(LNr[Lieferant], WNr[Ware], KNr[Kunde], Menge)
```

a) Geben Sie die Anweisung in **SQL-DDL** an, die notwendig ist, um die Relation 'Auftrag' zu erzeugen. Achten Sie dabei auf Fremdschlüsselbeziehungen.

b) Formulieren Sie die folgende Anfrage in **SQL**:
Gesucht sind die Nummern aller Waren, die Lieferant Huber nach Hamburg liefert.

c) Formulieren Sie die folgende Anfrage in **SQL**: Gesucht ist eine Liste aller belieferten Städte mit ihrem jeweiligen gesamten Lieferungsvolumen.

Fortsetzung nächste Seite!

Aufgabe 6 Normalisierung

Gegeben sei das Relationenschema $R(A,B,C,D,E,F)$, sowie die Menge der zugehörigen funktionalen Abhängigkeiten F:

- $C \rightarrow B$
- $B \rightarrow A$
- $CE \rightarrow D$
- $E \rightarrow F$
- $CE \rightarrow F$
- $C \rightarrow A$

a) Bestimmen Sie den Schlüsselkandidaten der Relation R und begründen Sie, warum es keine weiteren Schlüsselkandidaten gibt.

b) Überführen Sie das Relationenschema R mit Hilfe des Synthesealgorithmus in die dritte Normalform. Führen Sie hierfür jeden der vier Schritte durch und kennzeichnen Sie Stellen, bei denen nichts zu tun ist.

Fortsetzung nächste Seite!

Teilaufgabe 2

Aufgabe 1: (Objektorientierte Programme und Reverse Engineering)

Gegeben sei das folgende Java-Programm:

```
class M {
   private boolean b;
   private F f;
   private A a;
   public void m() {
      f = new F();
      a = new A(f);
      b = true;
   }
}


class A {
   private R r;
   public A(I i) {
      r = i.createX();
   }
}


interface I {
   public X createX();
}


class F implements I {
   public X createX() {
      return new X(0,0);
   }
}


abstract class R {
   protected int v;
}


class X extends R {
   private int v, w;
   public X(int v, int w) {
      this.v = v;
      this.w = w;
   }
}
```

Fortsetzung nächste Seite!

a) Das Subtypprinzip der objektorientierten Programmierung wird in obigem Programmcode zweimal ausgenutzt. Erläutern Sie wo und wie dies geschieht.

b) Zeichnen Sie ein UML-Klassendiagramm, das die statische Struktur des obigen Programms modelliert. Instanzvariablen mit einem Klassentyp sollen durch gerichtete Assoziationen mit Rollennamen und Multiplizität am gerichteten Assoziationsende modelliert werden. Alle aus dem Programmcode ersichtlichen statischen Informationen (insbesondere Interfaces, abstrakte Klassen, Zugriffsrechte, benutzerdefinierte Konstruktoren und Methoden) sollen in dem Klassendiagramm abgebildet werden.

c) Es wird angenommen, dass ein Objekt der Klasse `M` existiert, für das die Methode `m()` aufgerufen wird. Geben Sie ein Instanzendiagramm (Objektdiagramm) an, das alle **nach** der Ausführung der Methode `m` existierenden Objekte und deren Verbindungen (Links) zeigt.

d) Wie in Teil c) werde angenommen, dass ein Objekt der Klasse `M` existiert, für das die Methode `m()` aufgerufen wird. Diese Situation wird in Abb. 1 dargestellt. Zeichnen Sie ein Sequenzdiagramm, das Abb. 1 so ergänzt, dass alle auf den Aufruf der Methode `m()` folgenden Objekterzeugungen und Interaktionen gemäß der im Programmcode angegebenen Konstruktor- und Methodenrümpfe dargestellt werden. Aktivierungsphasen von Objekten sind durch längliche Rechtecke deutlich zu machen.

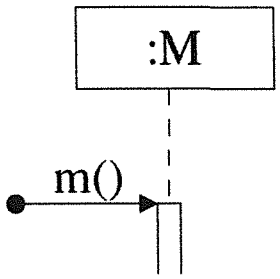


Abbildung 1: Beginn eines Sequenzdiagramms

Aufgabe 2: (Anwendung des Observer-Patterns)

Es soll eine (kleine) Anwendung entwickelt werden, in der ein Zähler in 1-er Schritten von 5000 bis 0 herunterzählt. Der Zähler soll als Objekt der Klasse `Countdown` realisiert werden, die in UML-Notation in Abb. 2 dargestellt ist. Das Attribut `value` soll den aktuellen Zählerstand speichern, der mit dem Konstruktor zu initialisieren ist. Die Methode `getValue` soll den aktuellen Zählerstand liefern und die Methode `countdown` soll den Zähler von 5000 bis 0 herunterzählen.

Der jeweilige Zählerstand soll von einem Objekt der in Abb. 2 angegebenen Klasse `Display` am Bildschirm ausgegeben werden. Bei der Konstruktion eines `Display`-Objekts soll es mit einem `Countdown`-Objekt verbunden werden, indem dessen Referenz unter `myCountdown` abgespeichert wird. Die Methode `update` soll den aktuellen Zählerstand vom `Countdown`- Objekt holen und mit `System.out.println` am Bildschirm ausgeben. Dies soll zu Beginn des Zählprozesses und nach jeder Änderung des Zählerstands erfolgen.

Fortsetzung nächste Seite!

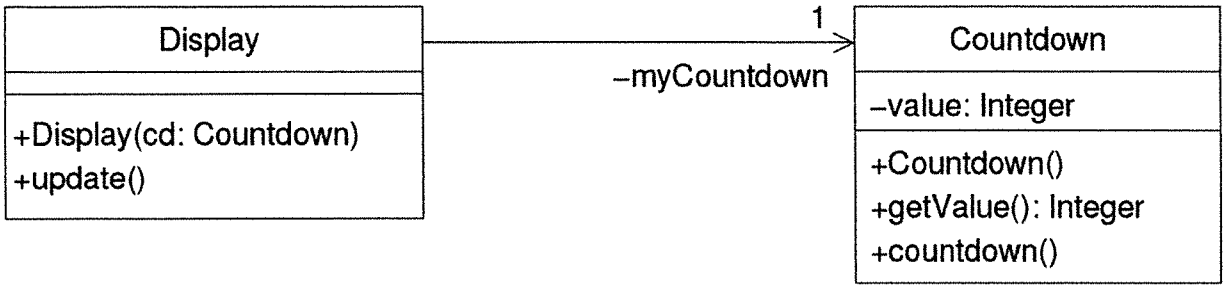


Abbildung 2: `Display` und `Countdown`

Damit das `Display`-Objekt über Zählerstände des `Countdown`-Objekts informiert wird, soll das Observer-Pattern angewendet werden. Abb. 3 zeigt die im Observer-Pattern vorkommenden abstrakten Klassen. (Kursivschreibweise bedeutet abstrakte Klasse bzw. abstrakte Methode.)

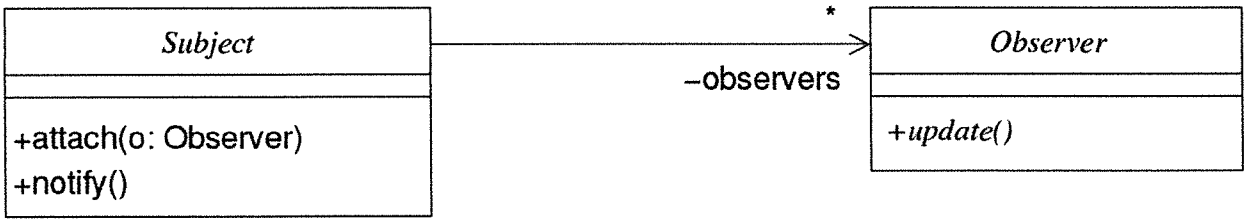


Abbildung 3: *Subject* und *Observer*

a) Welche Wirkung haben die Methoden `attach` und `notify` gemäß der Idee des Observer-Patterns?

b) Welche der beiden Klassen `Display` und `Countdown` aus Abb. 2 spielt die Rolle eines *Subject* und welche die Rolle eines *Observer*?

c) Erstellen Sie ein Klassendiagramm, das die in Abb. 2 und Abb. 3 gegebenen Diagramme in geeigneter Weise, d.h. entsprechend der Idee des Observer-Patterns, zusammenfügt. Es reicht die Klassen und deren Beziehungen anzugeben. Eine nochmalige Nennung der Attribute und Methoden ist nicht notwendig.

Fortsetzung nächste Seite!

d) Unsere Anwendung soll nun in einer objektorientierten Programmiersprache Ihrer Wahl (z.B. Java oder C++) implementiert werden. Dabei soll von folgenden Annahmen ausgegangen werden:

- Das Programm wird mit einer `main`-Methode gestartet, die folgenden Rumpf hat:

```
{Countdown cd = new Countdown();
new Display(cd);
cd.countdown();}
```

- Die beiden Klassen *Subject* und *Observer* sind bereits gemäß der Idee des Observer-Patterns implementiert.

Geben Sie auf dieser Grundlage eine Implementierung der beiden Klassen `Display` und `Countdown` an, so dass das gewünschte Verhalten, d.h. Anzeige der Zählerstände und Herunterzählen des Zählers, realisiert wird. Die Methoden der Klassen *Subject* und *Observer* sind dabei auf geeignete Weise zu verwenden bzw. zu implementieren. Geben Sie die verwendete Programmierprache an.

Aufgabe 3: (Entwicklung von Zustandsdiagrammen)

Ein Benutzer einer Online-Plattform für Flugreservierungen möchte für eine gegebene Flugnummer einen Flug buchen. Das Sequenzdiagramm in Abb. 4 spezifiziert, welche Interaktionen dazu in einem Hauptablauf durchgeführt werden. An der Interaktion sind Objekte der Klassen `Flugreservierung`, `Flug` und `Printer` beteiligt. Zunächst wird geprüft, ob für die gegebene Flugnummer noch Plätze frei sind. Ist dies der Fall, wird die Buchung getätigt und das Flugticket ausgedruckt. Das Sequenzdiagramm in Abb. 5 spezifiziert einen Ausnahmeablauf, in dem eine Fugbuchung mangels freier Plätze nicht möglich ist. Das Sequenzdiagramm in Abb. 6 spezifiziert einen zweiten Ausnahmeablauf, in dem zwar zunächst freie Plätze vorhanden sind, jedoch in der Zwischenzeit, bis der Benutzer die Buchung vornehmen will, der Flug ausgebucht ist.

Es ist **ein** Zustandsdiagramm für die Klasse `Flugreservierung` zu entwickeln. Das Zustandsdiagramm soll das mögliche Verhalten eines Objekts der Klasse Flugreservierung zeigen, indem es die Lebenslinien des `Flugreservierung`-Objekts aus allen drei Abläufen geeignet integriert. Auf folgende Punkte ist zu achten:

- Alle Abläufe sollen beliebig wiederholt werden können.
- Ereignisse sind von Aktionen zu unterscheiden und es muss klar sein, welche Aktionen als Reaktion auf welche Ereignisse durchgeführt werden.
- Aktionen sind hier immer Meldungen an den Benutzer oder Sendeaktionen oder (sequentiell) zusammengesetzte Aktionen. Das Senden einer Nachricht `n()` an ein Objekt `o:K` einer Klasse `K` ist im Zustandsdiagramm mit `o.n()` zu notieren. Die sequentielle Zusammensetzung zweier Aktionen `a1` und `a2` ist mit `a1;a2` zu notieren.
- Das Zustandsdiagramm hat einen Anfangszustand.

Fortsetzung nächste Seite!

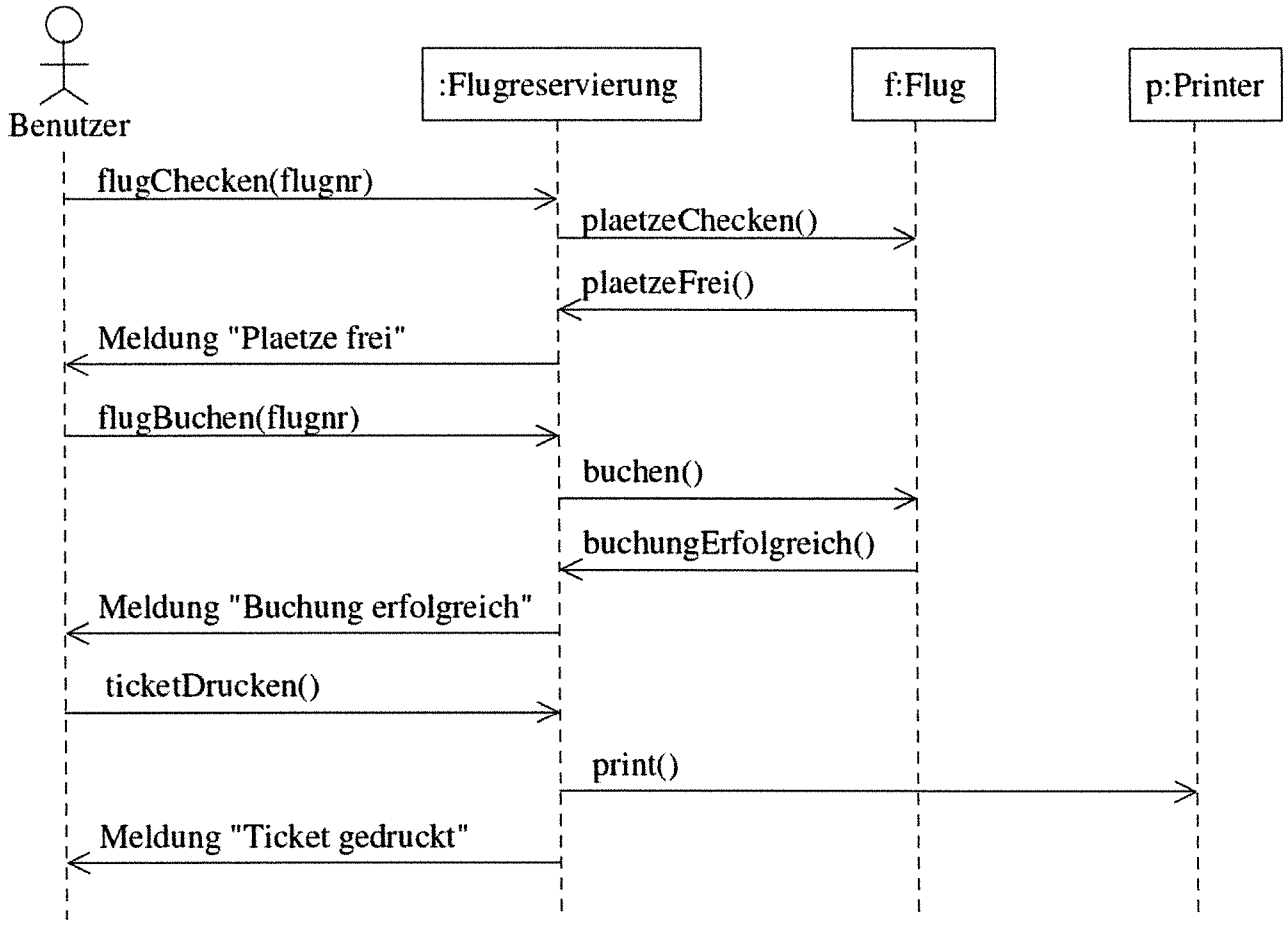


Abbildung 4: Hauptablauf

Fortsetzung nächste Seite!

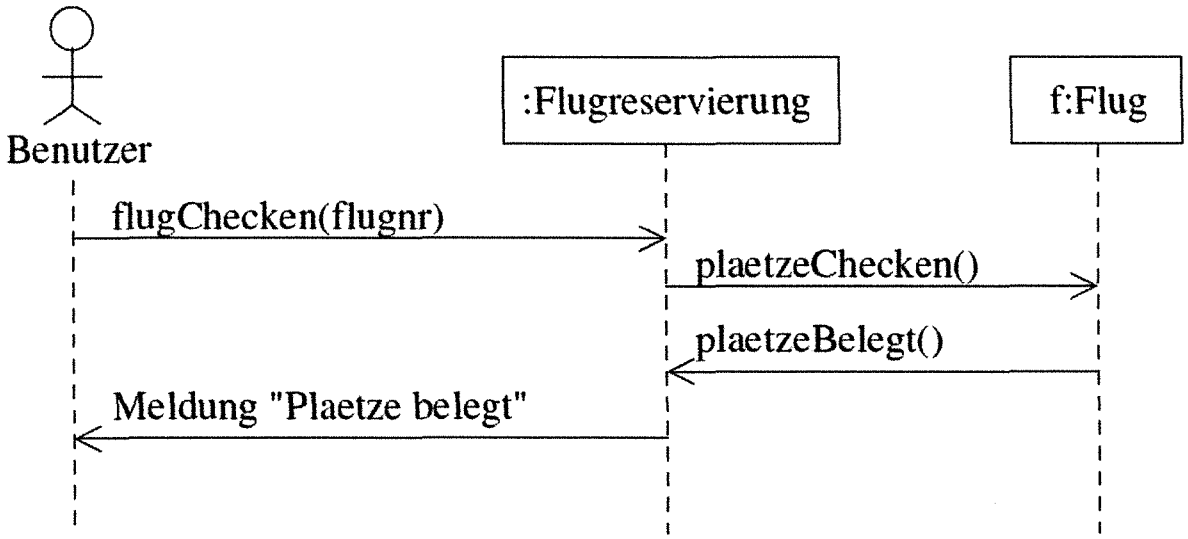


Abbildung 5: Ausnahmeablauf 1

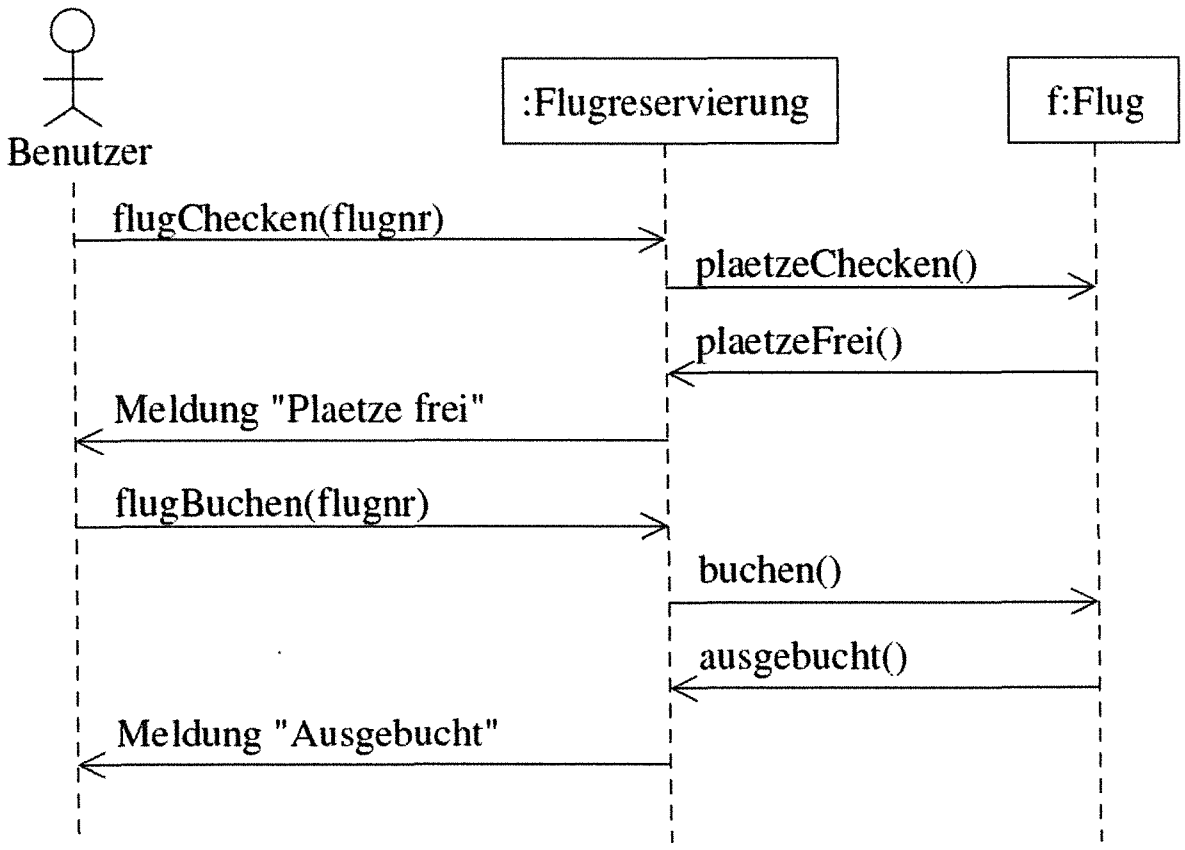


Abbildung 6: Ausnahmeablauf 2

Fortsetzung nächste Seite!

Aufgabe 4: (Verifikation funktionaler Programme)

Gegeben sei der folgende polymorphe Datentyp *'a boolexp* zur Darstellung Boolescher Ausdrücke über einer beliebigen Menge *'a* von Booleschen Variablen. Boolesche Ausdrücke werden hier gebildet aus Booleschen Variablen sowie aus Negation und Konjunktion.

datatype *'a boolexp* = *Var* **of** *'a* | *Not* **of** *'a boolexp* | *And* **of** *'a boolexp* * *'a boolexp*

Zum Beispiel wird der Boolesche Ausdruck : $\neg(x \wedge \neg y) \wedge \neg(\neg x \wedge y)$ dargestellt durch:

And(Not(And(x, Not(y))), Not(And(Not(x), y)))

Die Tiefe eines Ausdrucks wird durch die folgende Funktion *depth* (durch Pattern-Matching gemäß des strukturellen Aufbaus eines Ausdrucks) definiert. Sie berechnet die Anzahl der *Not*- und *And*-Konstruktoren auf dem längsten Ast des Ausdrucks.

fun *depth(Var(x))* = 0
| *depth(Not(B))* = *depth(B)* + 1
| *depth(And(B1,B2))* = *max(depth(B1), depth(B2))* + 1

Die Breite eines Ausdrucks wird durch die folgende Funktion *width* definiert. Sie berechnet die Anzahl der in einem Ausdruck vorkommenden Variablen.

fun *width (Var(x))* = 1
| *width (Not(B))* = *width(B)*
| *width (And(B1,B2))* = *width(B1)* + *width(B2)*

Es wird behauptet, dass für alle Ausdrücke *B* des Typs *'a boolexp* Folgendes gilt:

$$width(B) \leq 2^{depth(B)}$$

Beweisen Sie diese Behauptung durch strukturelle Induktion gemäß des Aufbaus des Ausdrucks *B*. Geben Sie zunächst an, welche Fälle für die Gestalt von *B* unterschieden werden müssen und welcher Fall der Basisfall der Induktion ist. Für jeden Nicht-Basisfall ist im Beweis die Induktionsvoraussetzung zu nennen und genau anzugeben, wo diese verwendet wird.