Klausur Datenbanken
Wintersemester 2017/2018
Prof. Dr. Wolfgang May
26. Februar 2018, 11-13 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.

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 FlexNow oder beim zuständigen Prüfungsamt.

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (ER-Modell)	15	12
Aufgabe 2 (Transformation in das Relationale Modell)	18	14
Aufgabe 3 (SQL und Relationale Algebra)	45	18
Aufgabe 4 (Verschiedenes)	12	5
Summe	90	49

Note:

Themenstellung: GöVB

Alle Klausuraufgaben basieren auf einem gemeinsamen "Auftrag": In der Klausur soll eine Datenbank für die Göttinger Verkehrsbetriebe entworfen werden, die deren Ressourcenplanung (Fahrer, Busse, Linien) unterstützt:

1. Für jeden Fahrer ist sein Name und sein Geburtsdatum gespeichert (es wird angenommen, dass der Name eindeutig ist).

   Ernst Meier ist am 26.2.1960 geboren. *Heinz Müller* ist am 3.10.1989 geboren.

2. Jeder Bus hat eine Nummer. Außerdem ist gespeichert, von welcher Marke der Bus ist, wieviele Sitz- und Stehplätze er hat, und ob er ein Gelenkbus ist.

   Bus Nr. 42 ist ein *Mercedes*-Gelenkbus, hat 39 Sitzplätze und 106 Stehplätze; Bus Nr. 43 ebenfalls. Bus Nr. 123 ist ein *MAN*-Bus, hat 25 Sitzplätze und 60 Stehplätze.

3. Jede Haltestelle hat einen eindeutigen Namen und eine Adresse.

   Die Haltestelle *Nikolausberg* hat die Adresse *Auf der Lieth 5*. Die Haltestelle *Bahnhof* hat die Adresse *Bahnhofsplatz*. Die Haltestelle *Zietenterrassen* hat die Adresse *Grete-Henry-Strasse 8*.

4. Für jede Linie ist ihre Nummer gespeichert, ihre Start- und Endhaltestelle, die Uhrzeit, wann morgens der erste Bus an der Starthaltestelle abfährt, ihre Umlaufdauer (d.h., wie lange es dauert, bis ein Bus wieder an der Starthaltestelle angekommen ist, und die nächste Runde beginnt) und ob sie mit Gelenkbussen befahren werden darf: Linie 21 fährt von *Nikolausberg* zu den *Zietenterrassen*, hat eine Umlaufdauer von 120 Minuten und darf mit Gelenkbussen befahren werden; der erste Bus fährt morgens um 6:25 Uhr. Linie 33 fährt vom *Holtenser Berg* zum *Klinikum*, hat eine Umlaufdauer von 60 Minuten, darf *nicht* mit Gelenkbussen befahren werden; erste Bus fährt morgens um 6:15 Uhr.

   Außerdem ist gespeichert, an welchen Haltestellen eine Linie hält, und nach wievielen Minuten nach ihrer Abfahrt sie diese erreicht. Die Starthaltestelle ist dabei mit "0 Minuten" angegeben (es ist nur jeweils eine Richtung gespeichert; es wird angenommen, dass die Rückfahrt genau umgekehrt erfolgt):

   Linie 21 fährt von *Nikolausberg* ab, hält (unter anderem) am *Faßberg* nach 5 Minuten, an der *Tammannstrasse* nach 10 Minuten, am *Bahnhof* nach 26 Minuten, und kommt an ihrer Endhaltestelle, den *Zietenterrassen*, nach 45 Minuten an. Keine Linie kommt auf ihrer Strecke mehrmals an derselben Haltestelle vorbei.

5. Es ist gespeichert, welche Fahrer welche Busse steuern dürfen; d.h. dass sie eine entsprechende Schulung absolviert haben, und wann dies war.

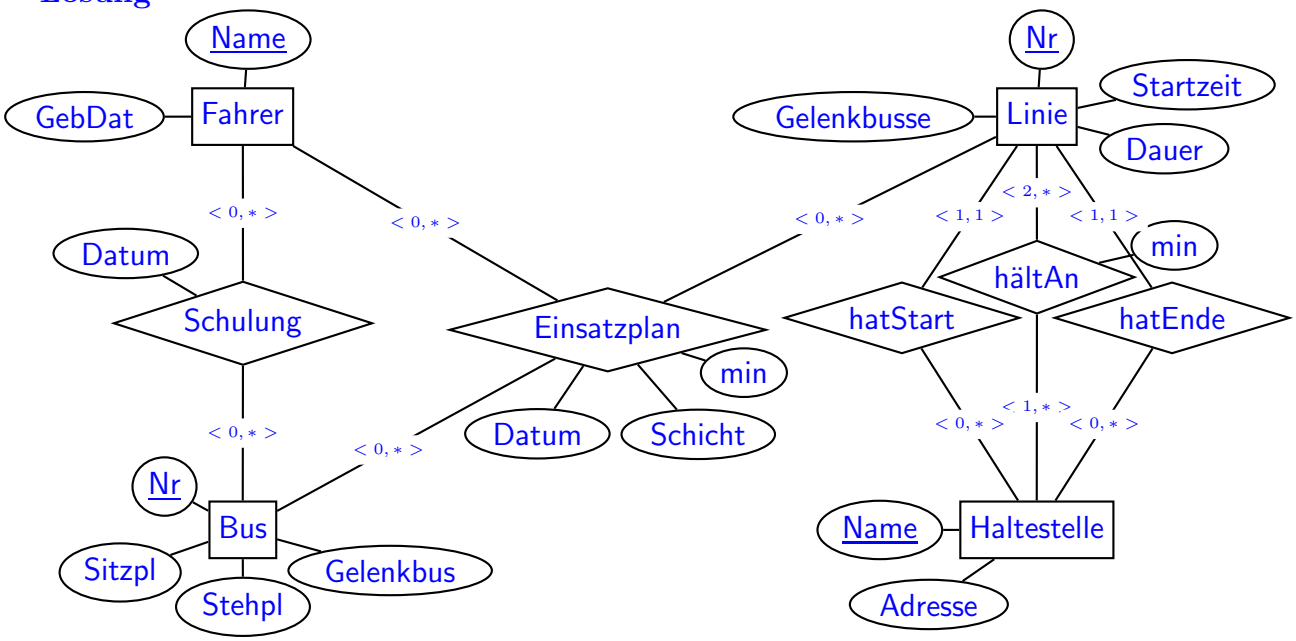
   Ernst Meier hatte Schulungen für die Busse mit den Nummern 42, 43 (jeweils am 10.3.2015) und 123 (am 24.10.2016).

6. Der Einsatzplan soll gespeichert werden: welcher Fahrer an welchem Tag mit welchem Bus welche Linie fährt, ob er für die Frühschicht oder für die Spätschicht eingeteilt ist, und wieviele Minuten nach dem ersten Bus auf dieser Linie er fährt. (Kommentar: bei einer halbstündig verkehrenden Linie mit einer Umlaufdauer von 120 Minuten sind dann z.B. pro Schicht 4 Fahrer mit 0, 30, 60, 90 Minuten Abstand eingeteilt). Ein Fahrer darf nicht am selben Tag zur Früh- und zur Spätschicht eingeteilt werden.

   Ernst Meier und *Heinz Müller* sind beide am 26.2.2018 für die Frühschicht auf der Linie 21 eingeteilt: Meier fährt morgens als erster (mit Bus Nr. 42), Müller 30 Minuten nach ihm (mit Bus Nr. 43). Am 27.2.2018 ist *Ernst Meier* für die Spätschicht auf der Linie 21 mit Bus Nr. 123 als erster eingeteilt.

Aufgabe 1 (ER-Modell [15 Punkte])
Entwickeln Sie ein ER-Modell für das Szenario. Geben Sie darin die Schlüsselattribute sowie die Beziehungskardinalitäten an.

Lösung



- Gelenkbus kann anstatt als (true/false-)Attribut auch als Subklasse von Bus modelliert werden (hat aber dann keine speziellen neuen Attribute, also nicht sehr interessant).
- Wenn man "Einsatzplan" nicht als dreistellige Beziehung, sondern als schwachen Entitätstyp modelliert, muß man an dieser Stelle schon über den (lokalen) Schlüssel und die identifizierenden Beziehungen entscheiden (ansonsten erst in Aufgabe 2; siehe Diskussion dort).

Aufgabe 2 (Transformation in das Relationale Modell [18 Punkte])

a) **Lösen Sie diesen Aufgabenteil auf dem *letzten* Blatt und trennen dieses ab (und geben es am Ende mit ab!).** Dann haben Sie dieses Blatt separat zugreifbar um später damit die Aufgaben 2b, 3 und 4 (SQL, Relationale Algebra+SQL, Diverses) zu lösen.

Geben Sie an, welche Tabellen (mit Attributen, Schlüsseln etc.) Ihre Datenbank enthält (keine SQL `CREATE TABLE`-Statements, sondern einfach grafisch). (12 P)

Markieren Sie dabei auch Schlüssel (durch unterstreichen) und Fremdschlüssel (durch überstreichen).

Geben Sie die Tabellen mit jeweils mindestens zwei Beispieltupeln (z.B. denen, die sich aus dem Aufgabentext ergeben, und weiteren erfundenen) an.

Vorgegeben wird bereits die Tabelle, in der gespeichert ist, für welche Busse die Fahrer Schulungen absolviert haben (sie wird in Aufgabe 4 wieder gebraucht):

Schulung		
Fahrer	Bus	Datum
Ernst Meier	42	10.3.2015
Ernst Meier	43	10.3.2015
Ernst Meier	123	24.10.2016
:	:	:

Lösung

Bus				
Nr	Marke	Sitzpl	Stehpl	Gelenkbus
42	Mercedes	39	106	ja
123	MAN	25	60	nein
:	:	:	:	

Fahrer	
Name	GebDat
Ernst Meier	26.2.1960
Heinz Müller	3.10.1989
:	:

Linie					
Nr	Starthaltestelle	Endhaltestelle	Startzeit	Umlaufdauer	Gelenkbusse
21	Nikolausberg	Zietenterrassen	06:25	120	ja
33	Holtenser Berg	Klinikum	06:15	60	nein
:	:	:	:	:	:

Haltestelle	
Name	Adresse
Nikolausberg	Auf der Lieth 5
Zietenterrassen	Grete-Henry-Strasse 8
:	:

hältAn		
Linie	Haltestelle	Minuten
21	Nikolausberg	0
21	Faßberg	5
21	Tammannstraße	10
21	Bahnhof	26
21	Zietenterrassen	45
:	:	:

Einsatzplan					
Datum	Fahrer	Linie	Bus	Schicht	Minuten
26.2.2018	Ernst Meier	21	42	F	0
26.2.2018	Heinz Müller	21	43	F	30
27.2.2018	Ernst Meier	21	123	S	0
:	:	:	:	:	:

- Primärschlüssel der Tabelle "hältAn": Die Kombination (Linie, Minuten) ist auch ein möglicher Primärschlüssel.
- Primärschlüssel der Tabelle "Einsatzplan":
  - (Fahrer, Datum) ist am sinnvollsten. Jeder Fahrer kann zu jedem Datum maximal einen Eintrag haben. Auf diese Weise wird die Integritätsbedingung, dass ein Fahrer nicht sowohl Früh- als auch Spätschicht am selben Tag arbeiten darf, garantiert)
  - (Datum, Linie, Schicht, Minuten) ist ebenfalls möglich und sinnvoll, und beschreibt eher die Sichtweise, dass dies die von den Kunden erwarteten Fahrten sind, zu denen jeweils Fahrer+Bus zugeordnet werden müssen.
  - Die beste Umsetzung in einem realen System wäre, eine der beiden Kombinationen als PRIMARY KEY auszuzeichnen, und die andere zusätzlich als UNIQUE.
- Wenn man es sehr genau betrachtet, sind auch Linie(Nr,Starthaltestelle) und Linie(Nr,Endhaltestell Foreign Keys auf hältAn(Linie,Haltestelle), da ein Bus nur an einer Haltestelle starten und enden kann, die auch eine Haltestelle seiner Linie ist. (Siehe dazu auch Aufgabe 4c, wo eine ähnliche Teilmengenbeziehung als Integritätsbedingung zugesichert werden kann.)

b) Geben Sie das `CREATE TABLE`-Statement für diejenige Tabelle, in der die Halte der einzelnen Linien gespeichert sind, so vollständig wie möglich an (6 P).

Lösung

```
CREATE TABLE haeltAn                    Basis 3P,
( Linie    NUMBER REFERENCES Linie(Nr),      1/2 P
  Haltestelle  VARCHAR2(40) REFERENCES Haltestelle(Name) 1/2 P
  Minuten   NUMBER NOT NULL CHECK (minuten >= 0),   1P
  PRIMARY KEY (Linie, Haltestelle),  1P
  UNIQUE (Linie, Minute) )          +1P, wenn jemand das hatte
```

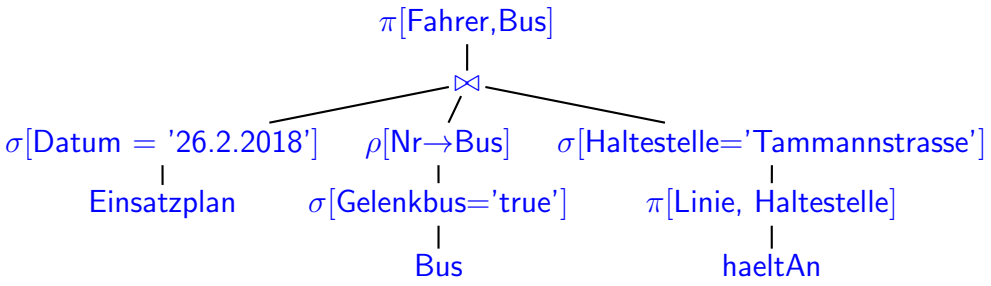
In der Aufgabenstellung wurde zugesichert, dass keine Linie zweimal an einer Haltestelle vorbeikommt; deshalb gehört "Minuten" nicht zum Schlüssel.

Aufgabe 3 (SQL und Relationale Algebra [45 Punkte])
Verwenden Sie für diese Aufgabe die von Ihnen entworfene relationale Datenbasis. Keine der Antworten soll Duplikate enthalten.

a) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die die Namen der Fahrer und die Nummern der Busse ausgeben, so dass derjenige Fahrer am 26.2.2018 mit einem Gelenkbus an der Haltestelle "Tammannstrasse" hält. (3+3P)

Lösung

```
SELECT Fahrer, Bus.Nr
FROM Einsatzplan, Bus, haeltAn
WHERE Bus = Bus.Nr
  AND Einsatzplan.Linie = haeltAn.Linie
  AND Haltestelle = 'Tammannstrasse'
  AND Bus.Gelenkbus = 'true'
  AND Datum = '26.2.2018'
```



Wichtig: in der Musterlösung haben Einsatzplan und Haltestelle jeweils "Minuten". Wenn man das so macht, muss man eine dieser Spalte vor dem Join wegprojizieren (oder wegumbenennen).

b) Geben Sie eine SQL-Anfrage an, die zu jeder Haltestelle ausgibt, wieviele Linien an dieser Haltestelle halten. (2 P)

Lösung

```
SELECT Haltestelle, count(*)
FROM haeltAn
GROUP BY Haltestelle
```

c) Geben Sie eine eine SQL-Anfrage an, die den/die Namen derjenigen Haltestelle/Haltestellen ausgibt, an der/denen die *meisten* Linien halten. (4 P)

Lösung

```
SELECT Haltestelle
FROM haeltAn
GROUP BY Haltestelle
HAVING count(*) =
  (SELECT max(count(*))
   FROM haeltAn
   GROUP BY Haltestelle)
```

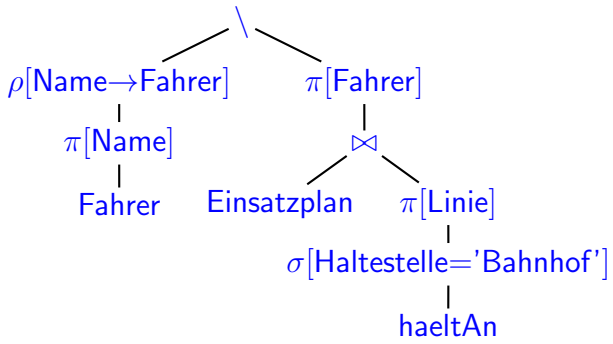
```
SELECT Haltestelle
FROM haeltAn
GROUP BY Haltestelle
HAVING count(*) >= ALL
  (SELECT count(*)
   FROM haeltAn
   GROUP BY Haltestelle)
```

d) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die die Namen aller Fahrer ausgeben, die nach ihrem Einsatzplan *nie* an der Haltestelle *Bahnhof* halten. (3+4 P)

Lösung

```
SELECT name
FROM Fahrer
WHERE NOT EXISTS
(SELECT *
 FROM Einsatzplan, haeltAn
 WHERE Fahrer = Fahrer.Name
   AND Linie = haeltAn.Linie
   AND Haltestelle = 'Bahnhof')
```

```
(SELECT name AS Fahrer
 FROM Fahrer)
 MINUS
(SELECT Fahrer
  FROM Einsatzplan, haeltAn
 WHERE Linie = haeltAn.Linie
   AND  Haltestelle = 'Bahnhof')
```

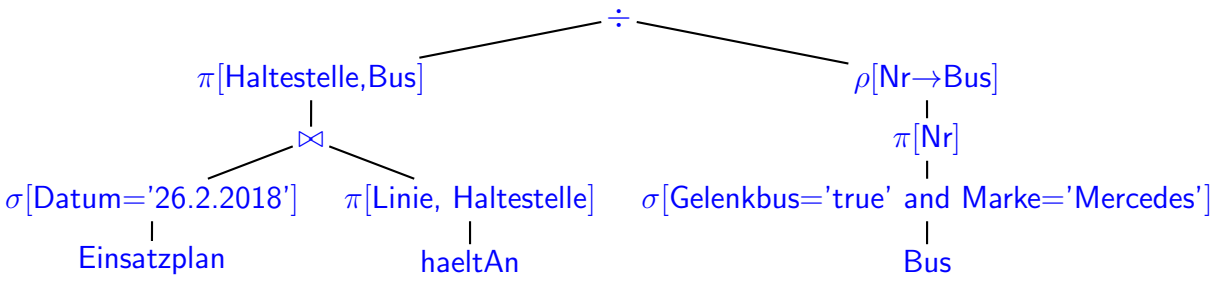


e) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck oder -Baum an, die die Namen aller Haltestellen angeben, an denen am 26.2.2018 *alle Mercedes-Gelenkbusse* mindestens einmal vorbeikommen. (5+4P)

Lösung

```
SELECT name
FROM Haltestelle hs
WHERE NOT EXISTS (SELECT *
                  FROM Bus
                  WHERE marke='Mercedes'
                    AND Gelenkbus='true'
                  AND NOT EXISTS
                     (SELECT *
```

```
FROM Einsatzplan d, haeltAn h
WHERE d.Bus = Bus.Nr
  AND d.Linie = h.Linie
  AND h.Haltestelle = hs.Name
  AND Datum = '26.2.2018'))
```



f) Geben Sie eine SQL-Anfrage an, die ergibt, um welche Uhrzeit *Heinz Müller* am 26.2.2018 zum ersten Mal mit einem Bus am Bahnhof hält, falls er Frühschicht hat. Nehmen Sie für diese Aufgabe an, dass man eine Zahl (als Minuten) einfach zu einem Zeit-Wert addieren kann. (4 P)

Lösung

```
SELECT Linie.Startzeit + Einsatzplan.minuten + haeltAn.Minuten AS zeit
FROM Einsatzplan, Linie, haeltAn
WHERE Einsatzplan.Linie = Linie.Nr
  AND Linie.Nr = haeltAn.Linie
  AND Haltestelle = 'Bahnhof'
  AND Datum = '26.2.2018'
  AND Fahrer = 'Heinz Mueller'
  AND Schicht = 'F'
```

g) Wie sieht das Ergebnis der Anfrage aus Teil (f) aus, wenn *Heinz Müller* an diesem Tag frei hat? (2 P)

Lösung In (f) sah das Ergebnis so aus:

Zeit
07:21

Wenn er frei hat, wird im Einsatzplan kein Tupel gefunden, es kann auch nichts drangejoint werden, womit das Ergebnis die entsprechende *leere Tabelle* ist:

Zeit

Analog wenn er Spätschicht hat, oder eine Linie fährt, die nicht am Bahnhof vorbeikommt; Insbesondere ist dies kein Nullwert, und keine Zahl "0".

h) Geben Sie eine SQL-Anfrage an, die für jedes Paar von Linien, die mindestens eine Haltestelle gemeinsam haben, angibt, wieviele Haltestellen sie gemeinsam haben. (5 P)

Lösung

```
SELECT h1.Linie as eine, h2.Linie as andere, count(*)
FROM haeltAn h1, haeltAn h2
WHERE h1.Haltestelle = h2.Haltestelle
  AND h1.Linie < h2.Linie    -- jedes Paar nur einmal und nicht (x,x) selbe
GROUP BY h1.Linie, h2.Linie

SELECT l1.nr, l2.nr, count(*)
FROM Linie l1, Linie l2, Haltestelle
WHERE l1.nr < l2.nr
  AND (l1, Haltestelle.name) IN (SELECT Linie, Haltestelle FROM haeltAn)
  AND (l2, Haltestelle.name) IN (SELECT Linie, Haltestelle FROM haeltAn)
GROUP BY h1.Linie, h2.Linie
```

Die Paare von Linien, die *keine* Haltestelle gemeinsam haben, muss man nicht explizit (z.B. mit "HAVING COUNT(*) $>$ 0") ausschließen, da sie aufgrund der Bedingungen garnicht auftreten.

Wenn man sie dabeihaben wollte, müßte man sie explizit dazunehmen:

```
(...) UNION
(SELECT l1.nr as eine, l2.nr as andere, 0
 FROM Linie l1, Linie l2
 WHERE NOT EXISTS
   ((SELECT Haltestelle FROM haeltAn WHERE Linie = l1.Nr)
    INTERSECT
    (SELECT Haltestelle FROM haeltAn WHERE Linie = l2.Nr)))
```

i) Ein bisschen Theorie (6 P).
Sei $R(\bar{X})$ eine Relation mit Primärschlüssel $\bar{Y} \subsetneq \bar{X}$.
Seien $\bar{V} \subsetneq \bar{X}$ und $\bar{W} \subsetneq \bar{X}$ Teilmengen der Attribute mit $\bar{V} \cup \bar{W} = \bar{X}$ und $\bar{V} \cap \bar{W} \neq \emptyset$.

i) Zeigen Sie, dass im allgemeinen **nicht** $\pi[\bar{V}](R) \bowtie \pi[\bar{W}](R) = R$ gilt, sondern meistens $\pi[\bar{V}](R) \bowtie \pi[\bar{W}](R) \supsetneq R$ ist. (3 P)

ii) Unter welchen Bedingungen (an $\bar{V}$ und $\bar{W}$) gilt garantiert $\pi[\bar{V}](R) \bowtie \pi[\bar{W}](R) = R$?
(mit Begründung) (3 P)

Lösung

i) Beispiel:

A	B	C
1	3	4
2	3	5

Betrachte $\pi[A,B](R) \bowtie \pi[B,C](R)$:

A	B
1	3
2	3

$\bowtie$

B	C
3	4
3	5

$=$

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

Derselbe Effekt trat auch auf Übungsblatt 1 bei der "nicht-verlustfreien" Zerlegung einer dreistelligen Beziehung ein; er ist allerdings nicht auf solche Fälle beschränkt.

ii) Man stellt fest, dass die Werte der Join-Attribute, d.h., $\pi[\bar{V} \cap \bar{W}](R)$, jeweils nur einmal vorkommen dürfen, damit keine zusätzlichen Tupel entstehen. D.h., $\pi[\bar{V}](R) \bowtie \pi[\bar{W}](R) = R$ kann genau dann garantiert werden, wenn die Join-Attribute alle Schlüsselattribute umfassen: $\bar{V} \cup \bar{W} \supseteq \bar{Y}$.

Dies nennt man auch "vertikale Partitionierung", zum Beispiel bei MONDIAL angewendet, um anstelle einer breiten Tabelle mit allen Daten der einzelnen Länder die Tabellen country, politics, economy, und population zu haben.
Die Tabellen mountain und mountainonisland stellen auch eine vertikale Partitionierung (aller funktionalen Eigenschaften von Bergen) dar.

Aufgabe 4 (Verschiedenes [12 Punkte])

a) *Jeden Abend* sollen die Daten des vergangenen Tages im Einsatzplan gelöscht werden. Wie macht man das am besten (SQL-Statement, und was könnte man weitergehend dazu anmerken?) (3 P)

Lösung Das SQL-Statement

```
DELETE FROM Einsatzplan
WHERE Datum = SYSDATE    -- SYSDATE gibt das aktuelle Datum
```

wird –am besten von einem zeitgesteuerten Trigger– abends ausgeführt.

b) Bus Nr. 123 bekommt als Pilotprojekt eine neue, teilautonome Fahrautomatik. *Ernst Meier*, der diesen Bus ja schon fuhr, bekam am 24.2.2018 eine neue Schulung für diesen Bus.

Jemand führt das folgende SQL-Statement aus (siehe die in Aufgabe 2b vorgegebene Tabelle "Schulung"):

```
INSERT INTO Schulung VALUES ('Ernst Meier', 123, '24.2.2018');
```

Was passiert – begründen Sie ihre Aussage? (2 P)

Lösung Das Update wird abgelehnt, da es die Schlüsselbedingung verletzt. Für jedes (Fahrer,Bus)-Paar darf nur eine Schulung eingetragen sein, und ein Tupel für (Meier,123) existiert ja schon.

Anmerkung: Man muss erst das alte Tupel löschen, oder

```
UPDATE Schulung
SET Datum='24.2.2018'
WHERE Fahrer='Ernst Meier' AND Bus='123'
```

ausführen.

c) Es darf nicht passieren, dass in den Einsatzplan eine Fahrer-Bus-Kombination eingetragen werden kann, die nicht zulässig ist (d.h., wo der Fahrer keine Schulung für diesen Bus hatte). Wie kann man das datenbankseitig *sicher verhindern*? (4 P)

Lösung Mit einer Integritätsbedingung auf der Tabelle "Einsatzplan":

```
CREATE TABLE Einsatzplan
( ...
  CONSTRAINT CheckSchulung
  FOREIGN KEY (Fahrer, Bus) REFERENCES Schulung(Fahrer, Bus)
)
```

So können dort nur Fahrer-Bus-Kombinationen eingetragen werden, die in der Tabelle "Schulung" auch eingetragen sind (dort sind diese Attribute ja Schlüssel).

Eine andere Möglichkeit ist, einen Trigger auf den Einsatzplan zu setzen, der vor der Einfügung überprüft, ob eine passende Schulung eingetragen ist (Syntax jeweils produktabhängig).

Mit einer einfachen CHECK-Bedingung ist es nicht möglich, da CHECK-Bedingungen nur *das aktuelle Tupel* betrachten dürfen (und schon gar nicht eine Anfrage an eine andere Tabelle beinhalten dürfen).

d) Während des Updates aus Aufgabenteil (b) gibt es einen Stromausfall. Der Computer läßt sich danach wieder problemlos starten, physikalisch sind alle Bauteile in Ordnung.

Skizzieren Sie, wie sich die Datenbank danach wieder (vom Admin entsprechend auszuführen) in einen regulären Zustand bringen kann. (3 P)

Lösung Es gibt zwei Möglichkeiten, beide nutzen den eingebauten Restart-Algorithmus, den der Admin entsprechend aufrufen kann:

- Im Hintergrundspeicher sind möglicherweise Teile des Updates (und vielleicht anderer nicht-committeter Transaktionen) bereits ausgeführt. Anhand des Logs kann man rückwärts alle Updates nicht-committeter Transaktionen rückgängig machen, und erhält den Zustand nach Ausführung aller committeter (und den Nutzern bestätigter Transaktionen).
- Man kann auch eine Sicherheitskopie der Datenbank (z.B. von Mitternacht) einspielen, und anhand des Logs vorwärts alle Updates committeter Transaktionen darauf wieder ausführen, und erhält ebenso den Zustand nach Ausführung aller committeter (und den Nutzern bestätigter Transaktionen).

Das (nicht committete) Update aus Aufgabenteil (b) muss man in beiden Fällen neu starten, da es ja nicht committed wurde.