Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

Fach Informatik

# Frühjahr 2020

66116

Datenbanksysteme / Softwaretechnologie (vertieft)

Aufgabenstellungen mit Lösungsvorschlägen

# Aufgabenübersicht

# Thema Nr. 1

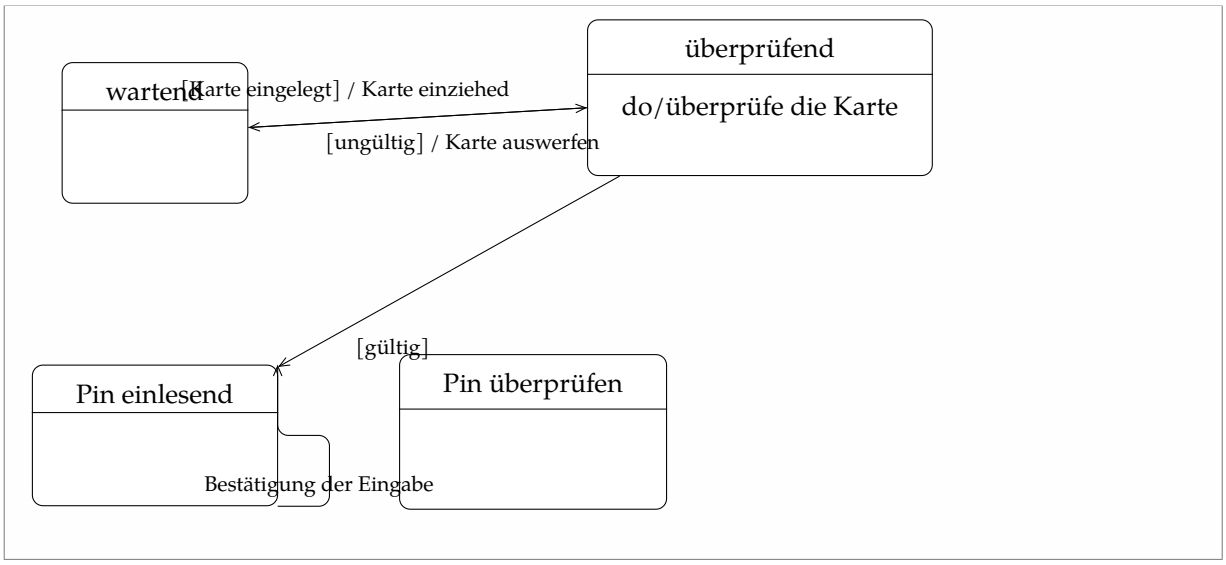
## Teilaufgabe Nr. 1

### Aufgabe 1 [Bankautomat]

(a) Basisfunktion

Erstellen Sie ein Zustandsdiagramm für einen Bankautomat, welcher den im Folgenden beschriebenen Authentifizierungsvorgang von Bankkunden realisiert. Modellieren Sie dazu soweit nötig sowohl Zustände und Transitionsbedingungen als auch die Aktionen der Zustände.

Der Bankautomat startet im Grundzustand und wartet auf das Einlegen einer Bankkarte. Wird eine Karte eingelegt, wird diese eingezogen und der Automat startet die Überprüfung der Bankkarte. Ist die Karte ungültig, wird die Karte ausgeworfen und der Automat wechselt in den Grundzustand. Ist die Karte gültig, kann die vierstellige PIN eingelesen werden. Nach der Bestätigung der Eingabe wird diese überprüft. Ist die PIN gültig, so stoppt der Automat und zeigt eine erfolgreiche Authentifizierung an. Ist die PIN ungültig, zeigt der Automat einen Fehler an und erlaubt eine erneute Eingabe der PIN.




(b) Erweiterung

Der Bankautomaten aus Aufgabe a) soll nun so verändert werden, dass ein Bankkunde nach dem ersten fehlerhaften Eingeben der PIN die PIN erneut eingeben muss. Bei erneuter Falscheingabe, wird eine dritte Eingabe möglich. Bei der dritten Falscheingabe der PIN wird die Karte vom Automaten eingezogen und der Automat geht wieder in den Ausgangszustand über.

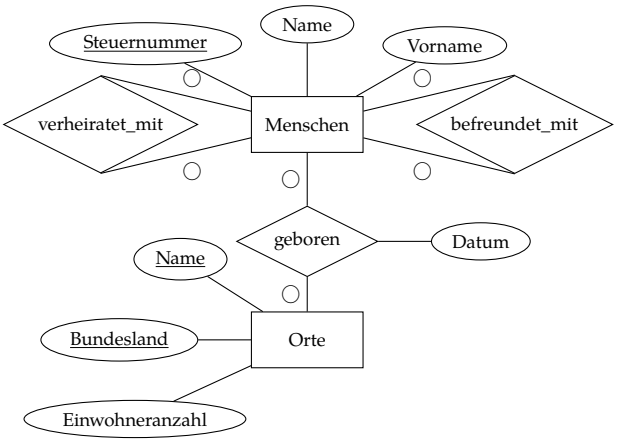
Hinweis: Für diese Aufgabe dürfen Sie Ihr Zustandsdiagramm aus a) weiter verwenden, wenn Sie eindeutig, z. B. durch den Einsatz von Farben kennzeichnen, was nur zur Aufgabe a) gehört und was Abänderungen des Zustandsdiagramms aus a) sind. Sie können, falls Sie einen neuen Automaten zeichnen, Zustände und Übergänge, die inhaltsgleich zur Lösung des Aufgabenteils a) sind mit einem "W" markieren, statt sie zu beschriften. In diesem Fall wird der Text aus der Lösung zu Aufgabenteil a) an dieser Stelle wiederholt gedacht.

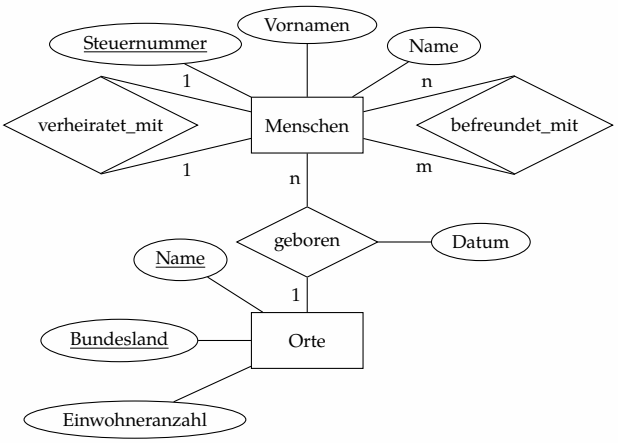
## Teilaufgabe Nr. 2

### Aufgabe 1 [Einwohnermeldeamt]

Gegeben sei folgendes ER-Diagramm:

(a) Übernehmen Sie das ER-Diagramm auf Ihre Bearbeitung und ergänzen Sie die Funktionalitätsangaben im Diagramm.




(b) Übersetzen Sie das ER-Diagramm in ein relationales Schema. - Datentypen müssen nicht angegeben werden.

```
Menschen(Steuernummer, Name, Vorname)

Orte(Name, Bundesland, Einwohneranzahl)

verheiratet_mit(Mensch[Menschen], Ehepartner[Menschen])

befreundet_mit(Mensch[Menschen], Freund[Menschen])

geboren(Datum, Steuernummer, Geburtsort[Orte], Geburtsbundesland[Orte])
```

(c) Verfeinern Sie das Schema aus Teilaufgabe b) indem Sie die Relationen zusammenfassen.

```
Menschen(Steuernummer, Name, Vorname, Ehepartner[Menschen], Geburtsdatum, Geburtsort[Orte], Geburtsbundes
land[Orte])

Orte(Name, Bundesland, Einwohneranzahl)

befreundet_mit(Mensch[Menschen], Freund[Menschen])
```

(d) Geben Sie sinnvolle SQL Datentypen für Ihr verfeinertes Schema an.

```sql
CREATE TABLE Menschen (
  Steuernummer BIGINT PRIMARY KEY,
  Name VARCHAR(30),
  Vorname VARCHAR(30),
  Ehepartner BIGINT REFERENCES Steuernummer,
  Geburtsdatum DATE,
  Geburtsort VARCHAR(30) REFERENCES Orte(Name),
  Geburtsbundesland VARCHAR(30) REFERENCES Orte(Bundesland)
);

CREATE TABLE Orte (
  Name VARCHAR(30),
  Bundesland VARCHAR(30),
  Einwohneranzahl INTEGER,
  PRIMARY KEY (Name, Bundesland)
);

CREATE TABLE befreundet_mit (
  Mensch BIGINT REFERENCES Mensch(Steuernummer),
  Freund BIGINT REFERENCES Mensch(Steuernummer),
  PRIMARY KEY (Mensch, Freund)
);
```

### Aufgabe 2 [Sekretäre]

#### Relation „Sekretäre"

| PersNr | Name | Boss | Raum |
|---|---|---|---|
| 4000 | Freud | 2125 | 225 |
| 4000 | | | 225 |
| 4020 | Röntgen | 2163 | 6 |
| 4020 | Röntgen | | 26 |
| 4030 | Galileo | 2127 | |
| | Freud | 2137 | 80 |

Gegeben sei oben stehenden (lückenhafte) Relationenausprägung **Sekretäre** sowie die folgenden funktionalen Abhängigkeiten:

$$\mathrm{FA} = \left\{ \begin{array}{r} \{\, \mathit{PersNr} \,\} \rightarrow \{\, \mathit{Name} \,\}, \\ \{\, \mathit{PersNr}, \mathit{Boss} \,\} \rightarrow \{\, \mathit{Raum} \,\}, \end{array} \right\}$$

Geben Sie für alle leeren Zellen Werte an, so dass keine funktionalen Abhängigkeiten verletzt werden. (Hinweis: Es gibt mehrere richtige Antworten.)

| PersNr | Name | Boss | Raum |
|---|---|---|---|
| 4000 | Freud | 2125 | 225 |
| 4000 | *Freud* | *2143*[a] | 225 |
| 4020 | Röntgen | 2163 | 6 |
| 4020 | Röntgen | *2163*[b] | 26 |
| 4030 | Galileo | 2127 | *27*[c] |
| *4000* | Freud | 2137 | 80 |

[a] Muss eine andere Boss-ID sein, sonst gäbe es zwei identische Zeilen.

[b] anderer Boss
[c] anderer Raum

### Aufgabe 3 [Universitätsschema]

Gegeben sei ein Universitätsschema.

(a) Finden Sie alle Studierenden, die keine Vorlesung hören. Formulieren Sie die Anfrage im Tupelkalkül.

$\{s \in \text{Studierende} \wedge h \in \text{hören} \mid \neg\exists \text{s.MatrNr} = \text{h.MatrNr}\}$

(b) Geben Sie einen Ausdruck an, der die Relation ¬`hören` erzeugt. Diese enthält für jeden Studierenden und jede Vorlesung, die der Studierende **nicht** hört, einen Eintrag mit Matrikelnummer und Vorlesungsnummer. Formulieren Sie die Anfrage in **relationaler Algebra**.

$\rho_{\neg\text{hören}}\Big((\pi_{\text{MatrNr}}(\text{Studierende}) \times \pi_{\text{VorlNr}}(\text{Vorlesungen})) - \text{hören}\Big)$

(c) Finden Sie alle Studierenden, die **keine** Vorlesung hören. Formulieren Sie die Anfrage in **relationaler Algebra**.

$\pi_{\text{MatrNr}}(\text{Studierende}) - \pi_{\text{MatrNr}}(\text{hören})$

### Aufgabe 4 [Relation A-F]

Gegeben sei die Relation

$$R(A, B, C, D, E, F)$$

mit den FDs

$$\text{FA} = \Big\{ \begin{aligned} \{A\} &\to \{B, C, F\}, \\ \{B\} &\to \{A, B, F\}, \\ \{C, D\} &\to \{E, F\}, \end{aligned} \Big\}$$

(a) Geben Sie alle Kandidatenschlüssel an.

- $\{A, D\}$
- $\{B, D\}$

(b) Überführen Sie die Relation mittels Synthesealgorithmus in die 3. NF. Geben Sie alle Relationen in der 3. NF an und **unterstreichen Sie in jeder einen Kandidatenschlüssel.** — Falls Sie Zwischenschritte notieren, machen Sie das Endergebnis **klar kenntlich.**

(i) **Kanonische Überdeckung**
*— Die kanonische Überdeckung - also die kleinst mögliche noch äquivalente Menge von funktionalen Abhängigkeiten kann in vier Schritten erreicht werden.*

i. **Linksreduktion**
*— Führe für jede funktionale Anhängigkeit $\alpha \to \beta \in F$ die Linksreduktion durch, überprüfe also für alle $A \in \alpha$, ob $A$ überflüssig ist, d. h. ob $\beta \subseteq \mathit{AttrHülle}(F, \alpha - A)$.*

**$\{C, D\} \to \{E, F\}$**
$\{E, F\} \notin \text{AttrHülle}(F, \{C, D \setminus D\}) = \{C\}$
$\{E, F\} \notin \text{AttrHülle}(F, \{C, D \setminus C\}) = \{D\}$

$$\text{FA} = \Big\{ \begin{array}{r} \{A\} \to \{B, C, F\}, \\ \{B\} \to \{A, B, F\}, \\ \{C, D\} \to \{E, F\}, \end{array} \Big\}$$

ii. **Rechtsreduktion**

*— Führe für jede (verbliebene) funktionale Abhängigkeit $\alpha \to \beta$ die Rechtsreduktion durch, überprüfe also für alle $B \in \beta$, ob $B \in AttrHülle(F - (\alpha \to \beta) \cup (\alpha \to (\beta - B)), \alpha)$ gilt. In diesem Fall ist $B$ auf der rechten Seite überflüssig und kann eleminiert werden, ðα → β wird durch α → (β − B) ersetzt. ———*

**F**

$F \in \text{AttrHülle}(F \setminus \{A\} \to \{B, C, F\} \cup \{A\} \to \{B, C\}, \{A\}) = \{A, B, C, \boldsymbol{F}\}$

$$\text{FA} = \Big\{ \begin{array}{r} \{A\} \to \{B, C\}, \\ \{B\} \to \{A, B, F\}, \\ \{C, D\} \to \{E, F\}, \end{array} \Big\}$$

$F \notin \text{AttrHülle}(F \setminus \{B\} \to \{A, B, F\} \cup \{B\} \to \{A, B\}, \{B\}) = \{A, B, C\}$
$F \notin \text{AttrHülle}(F \setminus \{C, D\} \to \{E, F\} \cup \{C, D\} \to \{E\}, \{C, D\}) = \{C, D, E\}$

**B**

$B \notin \text{AttrHülle}(F \setminus \{A\} \to \{B, C\} \cup \{A\} \to \{C\}, \{A\}) = \{A, C\}$
$B \in \text{AttrHülle}(F \setminus \{B\} \to \{A, B, F\} \cup \{B\} \to \{A, F\}, \{B\}) = \{A, \boldsymbol{B}, F\}$

$$\text{FA} = \Big\{ \begin{array}{r} \{A\} \to \{B, C\}, \\ \{B\} \to \{A, F\}, \\ \{C, D\} \to \{E, F\}, \end{array} \Big\}$$

iii. **Löschen leerer Klauseln**

*— Entferne die funktionalen Abhängigkeiten der Form $\alpha \to \varnothing$, die im 2. Schritt möglicherweise entstanden sind. ———*

$\varnothing$ Nichts zu tun

iv. **Vereinigung**

*— Fasse mittels der Vereinigungsregel funktionale Abhängigkeiten der Form $\alpha \to \beta_1, \ldots, \alpha \to \beta_n$, so dass $\alpha \to \beta_1 \cup \cdots \cup \beta_n$ verbleibt. ———*

$\varnothing$ Nichts zu tun

(ii) **Relationsschemata formen**

*— Erzeuge für jede funktionale Abhängigkeit $\alpha \to \beta \in F_c$ ein Relationenschema $\mathcal{R}_\alpha := \alpha \cup \beta$. ———*

$R_1(\underline{A, B}, C)$
$R_2(\underline{A, B}, F)$
$R_3(\underline{C, D}, E, F)$

(iii) **Schlüssel hinzufügen**

*— Falls eines der in Schritt 2. erzeugten Schemata $R_\alpha$ einen Schlüsselkandidaten von $\mathcal{R}$ bezüglich $F_c$ enthält, sind wir fertig, sonst wähle einen Schlüsselkandidaten $\mathcal{K} \subseteq \mathcal{R}$ aus und definiere folgendes zusätzliche Schema: $\mathcal{R}_\mathcal{K} := \mathcal{K}$ und $\mathcal{F}_\mathcal{K} := \varnothing$ ———*

$R_1(\underline{A, B}, C)$
$R_2(\underline{A, B}, F)$
$R_3(\underline{C, D}, E, F)$
$R_4(\underline{A, D})$

(iv) **Entfernung überflüssiger Teilschemata**

— *Eliminiere diejenigen Schemata $R_\alpha$, die in einem anderen Relationenschema $R_{\alpha'}$ enthalten sind, d. h. $R_\alpha \subseteq R_{\alpha'}$.* ————

∅ Nichts zu tun

### Aufgabe 5 [Schlüssel]

Gegeben sei die Relation $R(A,B,C)$

(a) Schreiben Sie eine SQL-Anfrage, mit der sich zeigen lässt, ob das Paar $A$, $B$ ein Superschlüssel der Relation $R$ ist. Beschreiben Sie ggf. textuell - falls nicht eindeutig ersichtlich - wie das Ergebnis Ihrer Anfrage interpretiert werden muss, um zu erkennen ob $A$, $B$ ein Superschlüssel ist.

Diese Anfrage darf keine Ergebnisse liefern, dann ist das Paar $A$, $B$ ein Superschlüssel.

```sql
SELECT *
FROM R
GROUP BY A, B
HAVING COUNT(*) > 1;
```

(b) Erläutern Sie den Unterschied zwischen einem Superschlüssel und einem Kandidatenschlüssel. Tipp: Was muss gelten, damit $A$, $B$ ein Kandidatenschlüssel ist und nicht nur ein Superschlüssel?

Ein Superschlüssel ist ein Attribut oder eine Attributkombination, von der *alle Attribute* einer Relation funktional *abhängen*.

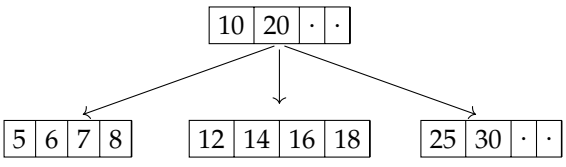
Ein Kandidatenschlüssel ist ein *minimaler* Superschlüssel. Keine Teilmenge dieses Superschlüssels ist ebenfalls Superschlüssels.

(c) Sei $A$, $B$ der Kandidatenschlüssel für die Relation $R$. Geben Sie eine minimale Ausprägung der Relation $R$ an, die diese Eigenschaft erfüllt.

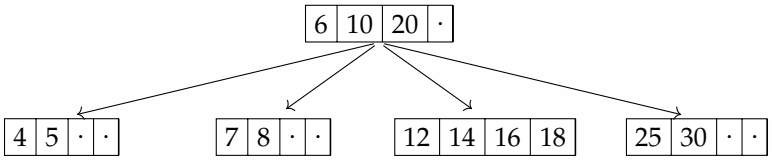
| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 1 | 4 |
| 1 | 1 | 5 |
| 2 | 2 | 5 |

### Aufgabe 6 [Physische Datenstrukturen)]

Fügen Sie die Zahl 4 in den folgenden B-Baum ein.




Zeichnen Sie den vollständigen, resultierenden Baum.



### Aufgabe 7 [Zehnkampf]

Gegeben sei die Relation *Zehnkampf*, welche die Ergebnisse eines Zehnkampfwettkampfes verwaltet. Eine beispielhafte Ausprägung ist in nachfolgender Tabelle gegeben.

**Hinweise:** Jeder Athlet kann in jeder Disziplin maximal ein Ergebnis erzielen. Außerdem können Sie davon ausgehen, dass jeder Name eindeutig ist.

| Name | Disziplin | Leistung | Einheit | Punkte |
|---|---|---|---|---|
| John | 100m | 10.21 | Sekunden | 845 |
| Peter | Hochsprung | 213 | Zentimeter | 812 |
| Peter | 100m | 10.10 | Sekunden | 920 |
| Hans | 100m | 10.21 | Sekunden | 845 |
| Hans | 400m | 44.12 | Sekunden | 910 |
| ... | ... | ... | ... | ... |

```sql
CREATE TABLE Zehnkampf (
  Name VARCHAR(30),
  Disziplin VARCHAR(30),
  Leistung FLOAT,
  Einheit VARCHAR(30),
  Punkte INTEGER,
  PRIMARY KEY(Name, Disziplin, Leistung)
);

INSERT INTO Zehnkampf VALUES
  ('John', '100m', 10.21, 'Sekunden', 845),
  ('Peter', 'Hochsprung', 213, 'Zentimeter', 812),
  ('Peter', '100m', 10.10, 'Sekunden', 920),
  ('Hans', '100m', 10.21, 'Sekunden', 845),
  ('Hans', '400m', 44.12, 'Sekunden', 910);
```

(a) Bestimmen Sie alle funktionale Abhängigkeiten, die **sinnvollerweise** in der Relation Zehnkampf gelten.

$$\text{FA} = \Big\{ \begin{aligned} \{\, \textit{Disziplin} \,\} &\rightarrow \{\, \textit{Einheit} \,\}, \\ \{\, \textit{Disziplin, Leistung} \,\} &\rightarrow \{\, \textit{Punkte} \,\}, \\ \{\, \textit{Name, Disziplin} \,\} &\rightarrow \{\, \textit{Leistung} \,\}, \end{aligned} \Big\}$$

(b) Normalisieren Sie die Relation Zehnkampf unter Beachtung der von Ihnen identifzierten funktionalen Abhängigkeiten. Unterstreichen Sie alle Schlüssel des resultierenden Schemas.

$R_1$ : {[ Disziplin, Einheit ]}
$R_2$ : {[ Disziplin, Leistung, Punkte ]}
$R_3$ : {[ Name, Disziplin, Leistung ]}

(c) Bestimmen Sie in SQL den Athleten (oder bei Punktgleichheit, die Athleten), der in der Summe am meisten Punkte in allen Disziplinen erzielt hat. Benutzen Sie dazu die noch nicht normalisierte Ausgangsrelation *Zehnkampf*.

```sql
CREATE VIEW GesamtPunkte AS
  SELECT Name, SUM(Punkte) As Punkte
  FROM Zehnkampf
  GROUP BY Name;

```

```sql
SELECT g1.Name, g1.Punkte, COUNT(*) AS Rang
FROM GesamtPunkte g1, GesamtPunkte g2
WHERE g1.Punkte <= g2.Punkte
GROUP BY g1.Name, g1.Punkte
HAVING COUNT(*) = 1;
```

```
 name | punkte | rang
------+--------+------
 Hans |   1755 |    1
(1 row)
```

### Aufgabe 8 [Universitätssschema]

Gegeben sei das Universitätssschema. Formulieren Sie folgende Anfragen in SQL-92:

```sql
CREATE TABLE Studierende (
  MatrNr INTEGER PRIMARY KEY,
  Name VARCHAR(15),
  Semester INTEGER
);

CREATE TABLE Professoren (
  PersNr INTEGER PRIMARY KEY,
  Name VARCHAR(30),
  Rang VARCHAR(30),
  Raum INTEGER
);

CREATE TABLE Assistenten (
  PersNr INTEGER PRIMARY KEY,
  Name VARCHAR(20),
  Fachgebiet VARCHAR(30),
  Boss INTEGER,
  FOREIGN KEY (Boss) REFERENCES Professoren(PersNr)
);

CREATE TABLE Vorlesungen (
  VorlNr INTEGER PRIMARY KEY,
  Titel VARCHAR(30),
  SWS INTEGER,
  gelesenVon INTEGER,
  FOREIGN KEY (gelesenVon) REFERENCES Professoren(PersNr)
);

CREATE TABLE hören (
  MatrNr INTEGER,
  VorlNr INTEGER,
  PRIMARY KEY(MatrNr, VorlNr),
  FOREIGN KEY (MatrNr) REFERENCES Studierende(MatrNr),
  FOREIGN KEY (VorlNr) REFERENCES Vorlesungen(VorlNr)
);

CREATE TABLE prüfen (
  MatrNr INTEGER,
  VorlNr INTEGER,
  PersNr INTEGER,
  Note INTEGER,
  PRIMARY KEY(MatrNr, VorlNr, PersNr),
  FOREIGN KEY (MatrNr) REFERENCES Studierende(MatrNr),
  FOREIGN KEY (VorlNr) REFERENCES Vorlesungen(VorlNr),
  FOREIGN KEY (PersNr) REFERENCES Professoren(PersNr)
);

CREATE TABLE voraussetzen (
  Vorgänger INTEGER,
  Nachfolger INTEGER,
  PRIMARY KEY(Vorgänger, Nachfolger),
  FOREIGN KEY (Vorgänger) REFERENCES Vorlesungen(VorlNr),
  FOREIGN KEY (Nachfolger) REFERENCES Vorlesungen(VorlNr)
```

```
);

INSERT INTO Studierende
  (MatrNr, Name, Semester)
VALUES
  (24002, 'Xenokrates', 18),
  (25403, 'Jonas', 12),
  (26120, 'Fichte', 10),
  (26830, 'Aristoxenos', 8),
  (27550, 'Schopenhauer', 6),
  (28106, 'Carnap', 3),
  (29120, 'Theophrastos', 2),
  (29555, 'Feuerbach', 2);

INSERT INTO Professoren
  (PersNr, Name, Rang, Raum)
VALUES
  (2125, 'Sokrates', 'C4', 226),
  (2126, 'Russel', 'C4', 226),
  (2127, 'Kopernikus', 'C3', 226),
  (2133, 'Popper', 'C3', 226),
  (2134, 'Augustinus', 'C3', 226),
  (2136, 'Curie', 'C4', 226),
  (2137, 'Kant', 'C4', 226);

INSERT INTO Assistenten
  (PersNr, Name, Fachgebiet, Boss)
VALUES
  (3002, 'Platon', 'Ideenlehre', 2125),
  (3003, 'Aristoteles', 'Syllogistik', 2125),
  (3004, 'Wittgenstein', 'Sprachtheorie', 2126),
  (3005, 'Rhetikus', 'Planetenbewegung', 2127),
  (3006, 'Newton', 'Kaplersche Gesetze', 2127),
  (3007, 'Spinosa', 'Gott und Natur', 2134);

INSERT INTO Vorlesungen
  (VorlNr, Titel, SWS, gelesenVon)
VALUES
  (4052, 'Logik', 4, 2125),
  (4630, 'Die 3 Kritiken', 4, 2137),
  (5001, 'Grundzüge', 4, 2137),
  (5022, 'Glaube und Wissen', 2, 2134),
  (5041, 'Ethik', 4, 2125),
  (5043, 'Erkenntnisstheorie', 3, 2126),
  (5049, 'Mäeutik', 2, 2125),
  (5052, 'Wissenschaftstheorie', 3, 2126),
  (5216, 'Bioethik', 2, 2126),
  (5259, 'Der Wiener Kreis', 2, 2133);

INSERT INTO hören
  (MatrNr, VorlNr)
VALUES
  (25403, 5022),
  (26120, 5001),
  (27550, 4052),
  (27550, 5001),
  (28106, 5041),
  (28106, 5052),
  (28106, 5216),
  (28106, 5259),
  (29120, 5001),
  (29120, 5041),
  (29120, 5049),
  (29555, 5001),
  (29555, 5022),
  (28106, 5001),
  (28106, 5022);

INSERT INTO prüfen
  (MatrNr, VorlNr, PersNr, Note)
VALUES
```

```
    (28106, 5001, 2126, 1),
    (25403, 5041, 2125, 2),
    (27550, 4630, 2137, 2),
    (25403, 4630, 2137, 5);

INSERT INTO voraussetzen
  (Vorgänger, Nachfolger)
VALUES
  (5001, 5041),
  (5001, 5043),
  (5001, 5049),
  (5041, 5216),
  (5043, 5052),
  (5041, 5052),
  (5052, 5259);
```

(a) Welche Vorlesungen liest der Boss des Assistenten *Platon* (nur Vorlesungsnummer und Titel ausgeben)?

```
SELECT v.VorlNr, v.Titel
FROM Vorlesungen v, Assistenten a
WHERE a.Boss = v.gelesenVon AND a.Name = 'Platon';
```

```
 vorlnr |  titel
--------+---------
   4052 | Logik
   5041 | Ethik
   5049 | Mäeutik
(3 rows)
```

(b) Welche Studierende haben sich schon in mindestens einer direkten Voraussetzung von *Wissenschaftstheorie* prüfen lassen?

Wissenschaftstheorie (5052) → Erkenntnistheorie (5043) Ethik (5041) → Jonas (25403)

```
SELECT s.Name
FROM Vorlesungen l, voraussetzen a, prüfen p, Studierende s
WHERE
  l.Titel = 'Wissenschaftstheorie' AND
  l.VorlNr = a.Nachfolger AND
  a.Vorgänger = p.VorlNr AND
  p.MatrNr = s.MatrNr;
```

```
 name
-------
 Jonas
(1 row)
```

(c) Wie viele Studierende hören *Ethik*?

```
SELECT COUNT(*)
FROM Vorlesungen v, hören h
WHERE
  v.Titel = 'Ethik' AND
  v.VorlNr = h.VorlNr;
```

```
 count
-------
     2
(1 row)
```

(d) Welche Studierende sind im gleichen Semester? — Geben Sie Paare von Studierenden aus.

Achten Sie darauf, dass ein/e Studierende/r mit sich selbst kein Paar bildet. — Achten Sie auch darauf, dass kein Paar doppelt ausgeben wird: wenn das Paar *StudentA*, *StudentB* im Ergebnis enthalten ist, soll nicht auch noch das Paar *StudentB*, *StudentA* ausgegeben werden.

```
SELECT s1.Name, s2.Name
FROM Studierende s1, Studierende s2
WHERE
  s1.Semester = s2.Semester AND
  s1.MatrNr < s2.MatrNr;
```

```
     name     |   name
--------------+-----------
 Theophrastos | Feuerbach
(1 row)
```

(e) In welchen Fächern ist die Durchschnittsnote schlechter als 2? Geben Sie die Vorlesungsnummer und den Titel aus.

```
SELECT v.VorlNr, v.Titel
FROM prüfen p, Vorlesungen v
WHERE p.VorlNr = v.VorlNr
GROUP BY v.VorlNr, v.Titel
HAVING AVG(p.Note) >  2;
```

```
 vorlnr |     titel
--------+----------------
   4630 | Die 3 Kritiken
(1 row)
```

(f) Finden Sie alle Paare von Studierenden (*MatrNr* duplikatfrei ausgeben), die mindestens zwei Vorlesungen gemeinsam hören.

```
SELECT h1.MatrNr, h2.MatrNr
FROM hören h1, hören h2
WHERE
  h1.VorlNr = h2.VorlNr AND
  h1.MatrNr < h2.MatrNr
GROUP BY h1.MatrNr, h2.MatrNr
HAVING COUNT(*) > 1;
```

```
 matrnr | matrnr
--------+--------
  28106 |  29120
  28106 |  29555
(2 rows)
```

# Thema Nr. 2

## Teilaufgabe Nr. 1

### Aufgabe 4 [Kartenschalter]

Erstellen Sie ein UML-Sequenzdiagramm zur Abbildung des folgenden Szenarios:

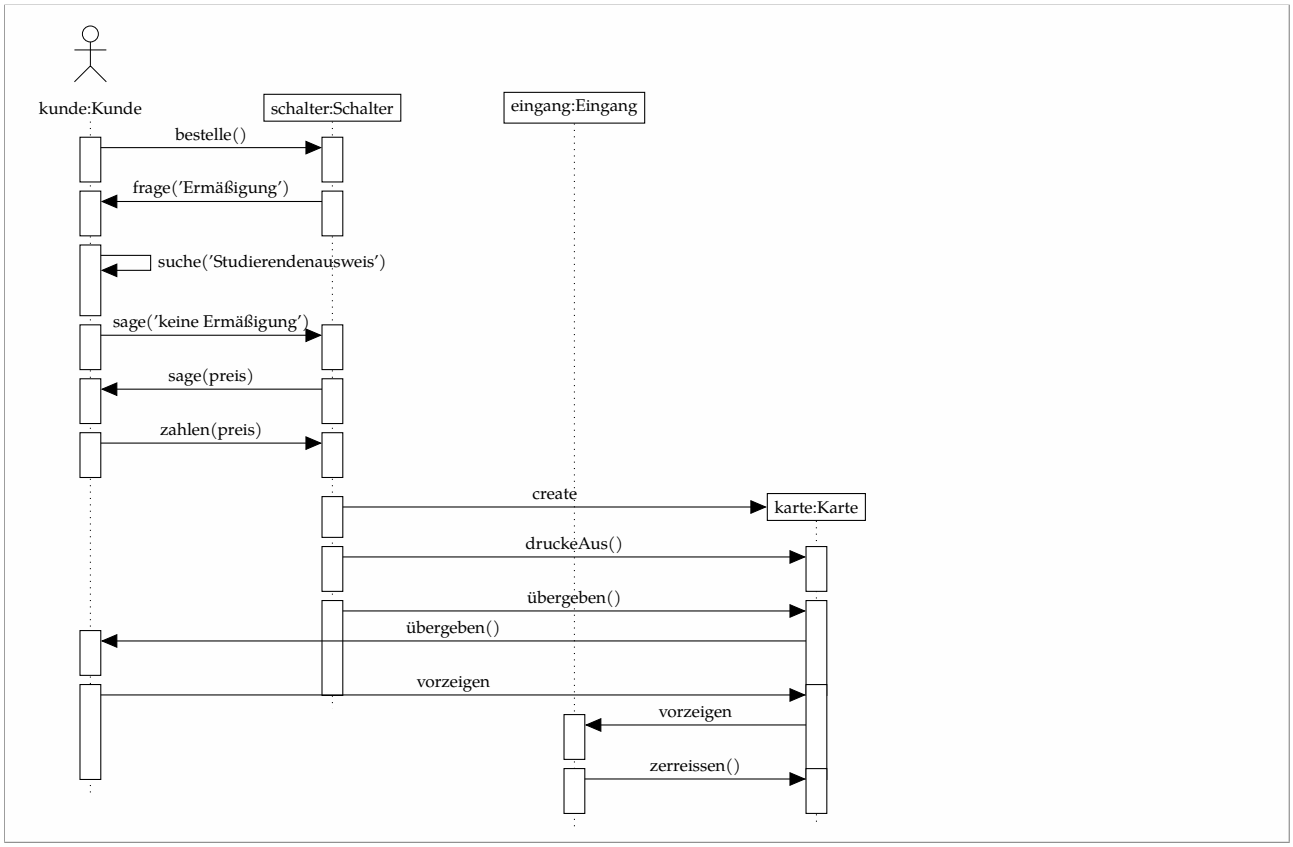
(a) Ein *Kunde* bestellt Karten an einem *Schalter*. Daraufhin wird er vom Schalter gefragt, ob er eine Ermäßigung nachweisen kann.

(b) Der *Kunde* sucht, leider erfolglos, seinen Studierendenausweis und sagt dem *Schalter*, dass er keinen Ermäßigungsgrund vorweisen kann.

(c) Der *Schalter* sagt dem Kunden den Preis der *Karten* und der Kunde gibt dem *Schalter* das notwendige Geld.

(d) Der *Schalter* erstellt die *Karten*, druckt diese aus und übergibt sie dem Kunden.

(e) Dieser geht mit den *Karten* zum *Eingang*, worauf die *Karten* zum Nachweis des Eintritts zerrissen werden.




### Aufgabe 5 [Object2D]

(a) Implementieren Sie ein Programm in einer objektorientierten Programmiersprache, z. B. Java, für das folgende UML-Klassendiagramm.

Die `shift`-Methode soll die `x`-Postion eines Objektes um `xShift` verändern und die `y`-Position um `yShift`. Die `draw`-Methode soll die Werte der Attribute der Klasse auf der Konsole ausgeben (- dies kann in Java mit `System.out.println ("...")` erfolgen).

```java
interface Drawable {

  public void draw();

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/fruehjahr/object2d/Drawable.java

```java
abstract class Object2D implements Drawable {
  public abstract void shift(int xShift, int yShift);
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/fruehjahr/object2d/Object2D.java

```java
public class Point extends Object2D {
  int xPos;
  int yPos;

  public Point(int x, int y) {
    xPos = x;
    yPos = y;
  }

  public void shift(int xShift, int yShift) {
    xPos += xShift;
    yPos += yShift;
  }

  public void draw() {
    System.out.println(String.format("xPos: %s, yPos: %s", xPos, yPos));
  }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/fruehjahr/object2d/Point.java

```java
public class Square extends Object2D {
  Point topLeft;
  Point bottomRight;

  public Square(int top, int left, int bottom, int right) {
    topLeft = new Point(left, top);
    bottomRight = new Point(right, bottom);
  }

  public void shift(int xShift, int yShift) {
    topLeft.shift(xShift, yShift);
    bottomRight.shift(xShift, yShift);
  }

  public void draw() {
    topLeft.draw();
    bottomRight.draw();
  }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/fruehjahr/object2d/Square.java

(b) Schreiben Sie eine Methode, die ein zweidimensionales Array aus ganzen Zahlen (Datentyp `int`) als Parameter bekommt und ein eindimensionales Array (bestehend aus ganzen Zahlen (Datentyp `int`)) zurückgibt, dessen Elemente jeweils der Summe der Einträge in der entsprechenden Zeile des zweidimensionalen Arrays entsprechen.

Achtung: Die Zeilen des zweidimensionalen Arrays können unterschiedlich lang sein.

Zur Vereinfachung sei die Signatur der Methode gegeben: `public int[] computeSum(int[][] input)`

```java
public class ComputeSum {
  public static int[] computeSum(int[][] input) {
    int[] output = new int[input.length];
    for (int i = 0; i < input.length; i++) {
      int[] numbers = input[i];
      int sum = 0;
      for (int j = 0; j < numbers.length; j++) {
        sum += numbers[j];
      }
      output[i] = sum;
    }
    return output;
  }

  public static void main(String[] args) {
    int[][] input = new int[3][];
    input[0] = new int[] { 1, 2, 3 };
    input[1] = new int[] { 4, 5 };
    input[2] = new int[] { 6 };
    int[] output = computeSum(input);
    for (int i = 0; i < output.length; i++) {
      System.out.println(output[i]);
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/fruehjahr/ComputeSum.java`

(c) Implementieren Sie eine einfach verkettete Liste in einer Klasse List (z. B. in Java), in der in jedem Listenelement ein String gespeichert wird. Die Klasse soll folgende Methoden bereitstellen:

- `void addFirst (String element)` : Diese Methode fügt ein Element am Anfang einer Liste ein.
- `void addLast (String element)` : Diese Methode hängt ein Element an das Ende der Liste an.
- `boolean exists(String element)` : Diese Methode gibt true zurück, wenn die Liste ein Element mit dem Inhalt element beinhaltet, andernfalls gibt sie false zurück.

Hinweis: Zwei `String`-Objekte können mittels der Funktion `equals(..)` verglichen werden.

```java
public class List {

  Element head;

  class Element {
    String value;
    Element next;

    public Element(String value) {
      this.value = value;
    }
  }

  public List() {
    head = null;
  }

  void addFirst(String element) {
    Element newElement = new Element(element);
    if (head != null) {
      newElement.next = head;
    }
    head = newElement;
  }

  void addLast(String element) {
    Element nextElement = head;
    Element lastElement = head;
    while (nextElement != null) {
```

```
32          lastElement = nextElement;
33          nextElement = nextElement.next;
34        }
35        if (lastElement != null) {
36          lastElement.next = new Element(element);
37        } else {
38          head = new Element(element);
39        }
40      }
41
42      boolean exists(String element) {
43        Element nextElement = head;
44        while (nextElement != null) {
45          if (nextElement.value.equals(element)) {
46            return true;
47          }
48          nextElement = nextElement.next;
49        }
50        return false;
51      }
52
53      public static void main(String[] args) {
54        List list = new List();
55        list.addLast("two");
56        list.addFirst("one");
57        list.addLast("three");
58
59        System.out.println(list.exists("one"));
60        System.out.println(list.exists("four"));
61        Element nextElement = list.head;
62        while (nextElement != null) {
63          System.out.println(nextElement.value);
64          nextElement = nextElement.next;
65        }
66      }
67    }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/fruehjahr/List.java

## Teilaufgabe Nr. 2

### Aufgabe 1 [Wetterdienst]

Hinweis: Bei Wahl dieser Aufgabe wird Wissen über das erweiterte Entity- Relationship-Modell (beispielsweise schwache Entity-Typen, Vererbung) sowie die Verfeinerung eines relationalen Sche- mas vorausgesetzt.

Gegeben seien folgende Informationen:

- Ein Wetterdienst - identifiziert durch einen eindeutigen Namen - betreibt mehrere Wetterstationen, die mit einer eindeutigen Nummer je Wetterdienst identifiziert werden können. Jede Wetterstation hat zudem mehrere Messgeräte, die wiederum pro Wetterstation einen eindeutigen Code besitzen und zudem eine Betriebsdauer.
- Ein Wetterdienst hat eine Adresse.
- Bei den Messgeräten wird unter anderem zwischen manuellen und digitalen Messgeräten unterschieden. Dabei kann ein Messgerät immer nur zu einer Kategorie gehören.
- Meteorologen sind Mitarbeiter eines Wetterdienstes, haben einen Namen und werden über eine Personalnummer identifiziert. Zudem soll gespeichert werden, in welcher Wetterstation welcher Mitarbeiter zu welchem Zeitpunkt arbeitet.
- Meteorologen können für eine Menge an Messgeräten verantwortlich sein. Manuelle Messgeräte werden von Meteorologen abgelesen.
- Ein Wettermoderator präsentiert das vorhergesagte Wetter eines Wetterdienstes für einen Fernsehsender.

- Für einen Wettermoderator wird der eindeutige Name und die Größe gespeichert. Ein Fernsehsender wird ebenfalls über den eindeutigen Namen identifiziert.

(a) Erstellen Sie für das oben gegebene Szenario ein geeignetes ER-Diagramm.

Verwenden Sie dabei - wenn angebracht - das Prinzip der Spezialisierung. Kennzeichnen Sie die Primärschlüssel der Entity-Typen, totale Teilnahmen (existenzabhängige Beziehungen) und schwache Entity-Typen.

Zeichnen Sie die Funktionalitäten der Relationship-Typen in das Diagramm ein.

(b) Überführen Sie das in Teilaufgabe a) erstellte ER-Modell in ein verfeinertes relationales Schema. Kennzeichnen Sie die Schlüssel durch Unterstreichen. Datentypen müssen nicht angegeben werden. Die einzelnen Schritte müssen angegeben werden.

> 1. Schritt: Starke Entity-Typen einfügen Wetterdienst Name, Adresse Meteorologe Personalnummer, Name Wettermoderator Name, Größe Fernsehsender Name 2. Schritt: Schwache Entity-Typen einfügen Wetterstation Name [Wetterdienst], Nummer Messgeraet Name [Wetterdienst], Nummer [Wetterstation], Code, Betriebsdauer 3. Schritt: Is-A-Beziehung einfügen ManuellesMessgeraet Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] DigitalesMessgeraet Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] 4. Schritt: Beziehungen einfügen Betreibt Name [Wetterdienst], Nummer Hat Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] ArbeitetIn Personalnummer [Meteorologe], Name [Wetterdienst], Nummer [Wetterstation], Zeitpunkt IstMitarbeiterVon Personalnummer [Meteorologe], Name [Wetterdienst] IstVerantwortlichFür Personalnummer [Meteorologe], Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] LiestAb Personalnummer [Meteorologe], Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] PraesentiertWetter ModeratorenName [Wettermoderator], FernsehsenderName [Fernsehsender], WetterdienstName [Wetterdienst] (Bei PraesentiertWetter sind nur zwei Felder Teil des Primärschlüssels, da es sich um eine 1 : 1 : n- Beziehung handelt.) 5. Schritt: Verfeinerung Wetterdienst Name, Adresse Wetterstation Name [Wetterdienst], Nummer [Wetterstation] Messgeraet Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet], Betriebsdauer, VerantwortlicherMitarbeiterPersonalnummer [Meteorologe] ManuellesMessgeraet Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] DigitalesMessgeraet Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] Meteorologe Personalnummer, Name, WetterdienstName [Wetterdienst] ArbeitetIn Personalnummer [Meteorologe], Name [Wetterdienst], Nummer [Wetterstation], Zeitpunkt LiestAb Personalnummer [Meteorologe], Name [Wetterdienst], Nummer [Wetterstation], Code [Messgeraet] Wettermoderator Name, Größe Fernsehsender Name PraesentiertWetter ModeratorenName [Wettermoderator], FernsehsenderName [Fernsehsender], WetterdienstName [Wetterdienst]

**Aufgabe 2 [Mode-Kollektionen]**

Gegeben sei der folgende Ausschnitt eines Schemas für die Verwaltung von Kollektionen:
Die Tabelle *Promi* beschreibt Promis über ihren eindeutigen Namen, ihr Alter und ihren Wohnort. Kollektion enthält Informationen über *Kollektionen*, nämlich deren eindeutigen Namen, das Jahr und die Saison. Die Tabelle *promotet* verwaltet über Referenzen, welcher Promi welche Kollektion promotet. *Kleidungsstück* speichert die IDs von Kleidungsstücken zusammen mit dem Hauptbestandteil und einer Referenz auf die zugehörige Kollektion. Die Tabelle *hat_getragen* verwaltet über Referenzen, welcher Promi welches Kleidungsstück an welchem Datum getragen hat.

Beachten Sie bei der Formulierung der SQL-Anweisungen, dass die Ergebnisrelationen keine Duplikate enthalten dürfen. Sie dürfen geeignete Views definieren.

```
1  CREATE TABLE Promi (
2    Name VARCHAR(255) PRIMARY KEY,
3    Alter INTEGER,
4    Wohnort VARCHAR(255)
5  );
6
7  CREATE TABLE Kollektion (
```

```
  Name VARCHAR(255) PRIMARY KEY,
  Jahr INTEGER,
  Saison VARCHAR(255)
);

CREATE TABLE Kleidungsstueck (
  ID INTEGER PRIMARY KEY,
  Hauptbestandteil VARCHAR(255),
  gehoert_zu VARCHAR(255) REFERENCES Kollektion(Name)
);

CREATE TABLE hat_getragen (
  PromiName VARCHAR(255) REFERENCES Promi(Name),
  KleidungsstueckID INTEGER REFERENCES Kleidungsstueck(ID),
  Datum DATE,
  PRIMARY KEY(PromiName, KleidungsstueckID, Datum)
);

CREATE TABLE promotet (
  PromiName VARCHAR(255),
  KollektionName VARCHAR(255),
  PRIMARY KEY(PromiName, KollektionName)
);

INSERT INTO Promi
  (Name, Alter, Wohnort)
VALUES
  ('Till Schweiger', 52, 'Dortmund'),
  ('Lena Meyer-Landrut', 30, 'Hannover');

INSERT INTO Kollektion VALUES
  ('Gerry Weber', 2020, 'Sommer'),
  ('H & M', 2020, 'Sommer');

INSERT INTO promotet
  (PromiName, KollektionName)
VALUES
  ('Till Schweiger', 'Gerry Weber'),
  ('Lena Meyer-Landrut', 'H & M');

INSERT INTO Kleidungsstueck
  (ID, Hauptbestandteil, gehoert_zu)
VALUES
  (1, 'Hose', 'Gerry Weber');

INSERT INTO hat_getragen
  (PromiName, KleidungsstueckID, Datum)
VALUES
  ('Till Schweiger', 1, '2021-08-03');
```

(a) Schreiben Sie SQL-Anweisungen, welche die Tabelle `hat_getragen` inklusive aller benötigten Fremdschlüsselconstraints anlegt. Erläutern Sie kurz, warum die Spalte `Datum` Teil des Primärschlüssels ist.

```
CREATE TABLE IF NOT EXISTS hat_getragen (
  PromiName VARCHAR(255) REFERENCES Promi(Name),
  KleidungsstueckID INTEGER REFERENCES Kleidungsstueck(ID),
  Datum DATE,
  PRIMARY KEY(PromiName, KleidungsstueckID, Datum)
);
```

Das Datenbanksystem achtet selbst darauf, dass Felder des Primärschlüssels nicht NULL sind. Deshalb muss man NOT NULL bei keinem der drei Felder angeben.

(b) Schreiben Sie eine SQL-Anweisung, welche die Namen der Promis ausgibt, die eine Sommer-Kollektion promoten (Saison ist „Sommer“).

```sql
SELECT p.PromiName
FROM promotet p, Kollektion k
WHERE
  k.Saison = 'Sommer' AND
  p.KollektionName = k.Name;
```

```
      prominame
--------------------
 Till Schweiger
 Lena Meyer-Landrut
(2 rows)
```

(c) Schreiben Sie eine SQL-Anweisung, die die Namen aller Promis und der Kollektionen bestimmt, welche der Promi zwar promotet, aber daraus noch kein Kleidungsstück getragen hat.

Lösung mit NOT IN:

```sql
SELECT * FROM promotet p
WHERE p.KollektionName NOT IN (
  SELECT k.Name FROM Kollektion k
  INNER JOIN Kleidungsstueck s ON s.gehoert_zu = K.Name
  INNER JOIN hat_getragen t ON t.KleidungsstueckID = s.ID
  WHERE t.PromiName = p.PromiName
);
```

Mit EXCEPT:

```sql
(
  SELECT p.PromiName, p.KollektionName FROM promotet p
)
EXCEPT
(
  SELECT p.PromiName, p.KollektionName FROM promotet p
  INNER JOIN Kollektion k ON p.KollektionName = k.Name
  INNER JOIN Kleidungsstueck s ON s.gehoert_zu = k.Name
  INNER JOIN hat_getragen t ON t.KleidungsstueckID = s.ID
  WHERE t.PromiName = p.PromiName
);
```

(d) Bestimmen Sie für die folgenden SQL-Anweisungen die minimale und maximale Anzahl an Tupeln im Ergebnis. Beziehen Sie sich dabei auf die Größe der einzelnen Tabellen.

Verwenden Sie für die Lösung folgende Notation: – Promi – beschreibt die Größe der Tabelle Promi.

(i)

```sql
SELECT k.Name
FROM Kollektion k, Kleidungsstueck kl
WHERE k.Name = kl.gehoert_zu and k.Jahr = 2018
GROUP BY k.Name
HAVING COUNT(kl.Hauptbestandteil) > 10;
```

– Kollektion – beschreibt die Anzahl der Tupel in der Tabelle Kollektion. Die minimale Anzahl an Tupeln im Ergebnis ist 0, da es sein kann, dass keine Kollektion den Anforderungen `k.Jahr = 2018` oder `HAVING COUNT(...)` genügt. Die maximale Anzahl an Tupeln im Ergebnis ist – Kollektion –, da nur Namen aus Kollektion ausgewählt werden.

(ii)

```sql
SELECT DISTINCT k.Jahr
FROM Kollektion k
WHERE k.Name IN (
  SELECT pr.KollektionName
  FROM Promi p, promotet pr
  WHERE p.Alter < 30 AND pr.PromiName = p.Name
);
```

Die minimale Anzahl an Tupeln im Ergebnis ist 0, da es sein kann, dass keine Kollektion promotet wird. Die maximale Anzahl ist das Minimum von – Kollektion – und 30, da die Promis, die Kollektionen beworben haben, die älter als 30 Jahre sind, selbst mindestens 30 Jahre alt sein müssen.

Zugrundeliegende Annahmen: Kollektionen werden nur im Erscheinungsjahr von Promis beworben; Neugeborene, die Kollektionen bewerben, werden ggf. Promis.

(e) Beschreiben Sie den Effekt der folgenden SQL-Anfrage in natürlicher Sprache

```sql
SELECT pr.KollektionName
FROM promotet pr, Promi p
WHERE pr.PromiName = p.Name
GROUP BY pr.KollektionName
HAVING COUNT (*) IN (
  SELECT MAX(anzahl)
  FROM (
    SELECT k.Name, COUNT(*) AS anzahl
    FROM Kollektion k, promotet pr
    WHERE k.Name = pr.KollektionName
    GROUP BY k.Name
  ) as tmp
);
```

Die Abfrage gibt die Namen aller Kollektionen aus, bei denen die Anzahl der bewerbenden Promis am größten ist.

(f) Formulieren Sie folgende SQL-Anfrage in relationaler Algebra. Die Lösung kann in Baum- oder in Term-Schreibweise angegeben werden, wobei eine Schreibweise genügt.

```sql
SELECT p.Wohnort
FROM Promi p, promotet pr, Kollektion k
WHERE
  p.Name = pr.PromiName AND
  k.Name pr.KollektionName AND
  k.Jahr 2018;
```

(i) Konvertieren Sie zunächst die gegebene SQL-Anfrage in die zugehörige Anfrage in relationaler Algebra nach Standard-Algorithmus.

$$\pi_{\text{Promi.Wohnort}}\Big(\sigma_{\text{Promi.Name}=\text{promotet.PromiName}\wedge\text{Kollektion.Name}=\text{promotet.KollektionName}\wedge\text{Kollektion.Jahr}=2018}(\text{Promi}\times\text{promotet}\times\text{Kollektion})\Big)$$

(ii) Führen Sie anschließend eine relationale Optimierung durch. Beschreiben und begründen Sie dabei kurz jeden durchgeführten Schritt.

1. Schritt: Um die kartesischen Produkte in Joins zu verwandeln, muss die Selektion in drei Selektionen zerlegt werden.

$$\pi_{\text{Promi.Wohnort}}($$
$$\sigma_{\text{Promi.Name=promotet.PromiName}}($$
$$\sigma_{\text{Kollektion.Name=promotet.KollektionName}}($$
$$\sigma_{\text{Kollektion.Jahr=2018}}(\text{Promi} \times (\text{promotet} \times$$
$$\text{Kollektion}$$
$$)$$
$$)$$
$$)$$
$$)$$

2. Schritt: Man kann die Selektion nach dem Jahr ausführen, bevor die kartesischen Produkte ausgeführt werden. Dadurch werden von vornherein nur die Kollektionen betrachtet, die in dem entsprechenden Jahr aufgetreten sind.

$$\pi_{\text{Promi.Wohnort}}(\sigma Promi.Name = promotet.PromiName(\sigma Kollektion.Name = promotet.KollektionName(Promi \times$$

3. Schritt: Die zweitinnerste Selektion kann direkt nach dem innersten kartesischen Produkt ausgeführt werden; dadurch wird das Gesamtprodukt nicht so groß. Man benötigt also weniger Rechenzeit und Speicher.

$$\pi_{\text{Promi.Wohnort}}(\sigma Promi.Name = promotet.PromiName(Promi \times \sigma Kollektion.Name = promotet.KollektionName$$

4. Schritt: Beide Selektionen in Verbindung mit dem jeweiligen kartesischen Produkt können in einen Join verwandelt werden. So wird nicht zuerst das gesamte Produkt berechnet und danach die passenden Eintraege ausgewaehlt, sondern gleich nur die zusammengehörigen Eintraege kombiniert. Auch dies spart Rechenzeit und Speicher.

$$\pi_{\text{Promi.Wohnort}}(Promi \bowtie Promi.Name = promotet.PromiName(promotet \bowtie Kollektion.Name = promotet.Kollek$$

**Aufgabe 3**

Gegeben sei folgendes relationales Schema R in erster Normalform:

$$\text{R:[A,B,C,D,E,F]}$$

Für $R$ gelte folgende Menge FD funktionaler Abhängigkeiten:

$$\text{FA} = \Big\{ \begin{array}{r} \{A\} \to \{F\}, \\ \{C, E, F\} \to \{A, B\}, \\ \{A, E\} \to \{B\}, \\ \{B, C\} \to \{D\}, \\ \{A, F\} \to \{C\}, \end{array} \Big\}$$

(a) Bestimmen Sie alle Kandidatenschlüssel/Schlüsselkandidaten von $R$ mit FD. Begründen Sie Ihre Antwort. Begründen Sie zudem, warum es keine weiteren Kandidatenschlüssel/Schlüsselkandidaten gibt.

*Hinweis: Die Angabe von Attributmengen, die keine Kandidatenschlüssel sind, führt zu Abzügen.*

E muss in allen Superschlüsseln enthalten sein, denn es steht nicht auf der rechten Seite von FD (*).

D kann in keinem Schlüsselkandidaten vorkommen, denn es steht nur auf der rechten Seite von FD (**).

E allein ist kein Schlüsselkandidat (***).

AE führt über FD zu B, A zu F, AF zu C und BC zu D, also ist AE ein Superschlüssel und damit wegen (*) und (***) ein Schlüsselkandidat. Wegen (*) enthält jeder Superschlüssel, der A enthält, AE. Also ist kein weiterer Superschlüssel, der A enthält, ein Schlüsselkandidat (****).

BE, CE und EF sind keine Superschlüssel, also auch keine Schlüsselkandidaten.

BCE ist kein Superschlüssel, da A und F nicht erreicht werden können.

BEF ist kein Superschlüssel, da A, D und F nicht erreicht werden können.

CEF führt über FD zu AB, BC führt dann zu D, also ist CEF ein Superschlüssel. Wegen (*), (**) und weil CE und EF keine Superschlüssel sind, ist CEF ein Schlüsselkandidat.

Das waren alle dreielementigen Buchstabenkombinationen, die (*), (**) und (****) genügen. Vierelementig ist nur BCEF und das enthält CEF, ist also kein Schlüsselkandidat.

Die einzigen Schlüsselkandidaten sind folglich AE und CEF.

(b) Prüfen Sie, ob R mit FD in 2NF bzw. 3NF ist.

R mit FD ist nicht in 2NF, denn bei Wahl des Schlüsselkandidaten AE hängt F von A, also nur einem Teil des Schlüssels, ab. Also ist AE $\rightarrow$ F nicht voll funktional. Damit ist R mit FD auch nicht in 3NF, denn 3NF $\subseteq$ 2NF.

(c) Bestimmen Sie mit folgenden Schritten eine kanonische Überdeckung FDc von FD. Begründen Sie jede Ihrer Entscheidungen:

(i) Führen Sie eine Linksreduktion von FD durch. Geben Sie die Menge funktionaler Abhängigkeiten nach der Linksreduktion an (FD;).

$$\mathrm{FA} = \Big\{ \begin{array}{r} \{A\} \rightarrow \{F\}, \\ \{C, E, F\} \rightarrow \{A, B\}, \\ \{A, E\} \rightarrow \{B\}, \\ \{B, C\} \rightarrow \{D\}, \\ \{A\} \rightarrow \{C\}, \end{array} \Big\}$$

(ii) Führen Sie eine Rechtsreduktion des Ergebnisses der Linksreduktion (FD;) durch. Geben Sie die Menge funktionaler Abhängigkeiten nach der Rechtsreduktion an (FD).

$$\mathrm{FA} = \Big\{ \begin{array}{r} \{A\} \rightarrow \{F\}, \\ \{C, E, F\} \rightarrow \{A\}, \\ \{A, E\} \rightarrow \{B\}, \\ \{B, C\} \rightarrow \{D\}, \\ \{A\} \rightarrow \{C\}, \end{array} \Big\}$$

(iii) Bestimmen Sie eine kanonische Überdeckung FD. von FD auf Basis des Ergebnisses der Rechtsreduktion (FD).

$$\text{FA} = \left\{ \begin{array}{r} \{A\} \to \{F, C\}, \\ \{C, E, F\} \to \{A\}, \\ \{A, E\} \to \{B\}, \\ \{B, C\} \to \{D\}, \end{array} \right\}$$

(d) Zerlegen Sie R mit FDc mithilfe des Synthesealgorithmus in 3NF. Geben Sie zudem alle funktionalen Abhängigkeiten der erzeugten Relationenschemata an.

(e) Prüfen Sie für alle Relationen der Zerlegung aus 4., ob sie jeweils in BCNF sind.

**Aufgabe 4**

(a) Betrachten Sie den folgenden Schedule S:

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | $r_2(z)$ | |
| | | $w_3(y)$ |
| | $r_2(x)$ | |
| $w_1(x)$ | | |
| | $w_2(x)$ | |
| | | $r_3(z)$ |
| | | $c_3$ |
| | $w_2(z)$ | |
| $w_1(y)$ | | |
| $c_1$ | | |
| | $c_2$ | |

Geben Sie den Ausgabeschedule (einschließlich der Operationen zur Sperranforderung und -freigabe) im rigorosen Zweiphasen-Sperrprotokoll für den obigen Eingabeschedule S an.

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | $\text{rlock}_2(z)$ | |
| | $r_2(z)$ | |
| | | $\text{xlock}_3(y)$ |
| | | $w_3(y)$ |
| | $\text{rlock}_2(x)$ | |
| | $r_2(x)$ | |
| | $\text{xlock}_2(x)$ | |
| | $w_2(x)$ | |
| | | $\text{rlock}_3(z)$ |
| | | $r_3(z)$ |
| | | $c_3$ |
| | | $\text{unlock}_2(y,z)$ |
| | $\text{xlock}_2(z)$ | |
| | $w_2(z)$ | |
| | $c_2$ | |
| | $\text{unlock}_2(x,z)$ | |
| $\text{xlock}_1(x)$ | | |
| $w_1(x)$ | | |
| $\text{xlock}_1(y)$ | | |
| $w_1(y)$ | | |
| $c_1$ | | |
| $\text{unlock}_2(x,z)$ | | |

(b) Beschreiben Sie den Unterschied zwischen dem herkömmlichen Zweiphasen-Sperrprotokoll (2PL) und dem rigorosen Zweiphasen-Sperrprotokoll. Warum wird in der Praxis häufiger das rigorose Zweiphasen-Sperrprotokoll verwendet?

Bei beiden Protokollen fordert die Transaktion erst alle Sperren an (Anforderungsphase) und gibt sie später frei (Freigabephase).

- Beim strengen oder rigorosen 2PL werden die Sperren dann angefordert, wenn sie benötigt werden, beim konservativen 2PL werden alle Sperren zu Beginn gemeinsam angefordert.
- Beim strengen oder rigorosen 2PL werden die Lesesperren bis zum Commit, die Schreibsperren sogar bis nach dem Commit gehalten. Beim konservativen 2PL werden dagegen die Sperren freigegeben, wenn sie nicht mehr benötigt werden.

Das rigorose 2PL wird der Praxis häufiger verwendet, weil dabei nicht zu Beginn der Transaktion bekannt sein muss, welche Sperren benötigt werden, und durch die schrittweise Anforderung der Sperren unter Umständen ein höheres Maß an Parallelität erreicht werden kann.