

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS08/09

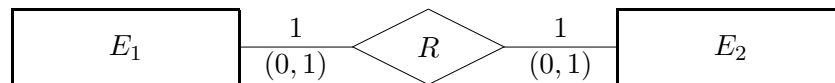
Martina Albutiu (albutiu@in.tum.de)

<http://www-db.in.tum.de/teaching/ws0809/dbsys/exercises/>

Lösung von Blatt Nr. 14

Aufgabe 1

Eine 1:1-Beziehung der Art



kann man sowohl durch Übernahme des Primärschlüssels von E_2 (als Fremdschlüssel) in E_1 als auch umgekehrt modellieren. Wenn die Beziehung aber nur für wenige Elemente von E_1 definiert ist, enthält die Relation viele Tupel mit Null-Werten für diesen Fremdschlüssel.

Geben Sie Beispiele aus der realen Welt, wo dies der Fall ist und man die Beziehungen deshalb besser in E_2 repräsentiert.

Geben Sie Beispiele, wo es sowohl für E_1 als auch für E_2 viele Elemente gibt, die die Beziehung R nicht „eingehen“. Diskutieren Sie für diesen Fall die Vor- und Nachteile einer separaten Repräsentation der Beziehung als eigenständige Relation.

Mögliche Umsetzungen ins relationale Modell:

- Übernahme des Schlüssels von E_2 als Fremdschlüssel in E_1 :

$$E_1 : \{[\underline{\text{Primärschlüssel von } E_1 : D_1}, \dots, \text{Fremdschlüssel von } E_2 : D_2]\}$$
$$E_2 : \{[\underline{\text{Primärschlüssel von } E_2 : D_2}, \dots]\}$$

- Übernahme des Schlüssels von E_1 als Fremdschlüssel in E_2 (symmetrisch zur ersten Möglichkeit):

$$E_1 : \{[\underline{\text{Primärschlüssel von } E_1 : D_1}, \dots]\}$$
$$E_2 : \{[\underline{\text{Primärschlüssel von } E_2 : D_2}, \dots, \text{Fremdschlüssel von } E_1 : D_1]\}$$

- Modellierung der Beziehung als separate Relation:

$$E_1 : \{[\underline{\text{Primärschlüssel von } E_1 : D_1}, \dots]\}$$
$$E_2 : \{[\underline{\text{Primärschlüssel von } E_2 : D_2}, \dots]\}$$
$$R : \{[\underline{\text{Fremdschlüssel von } E_1 : D_1}, \text{Fremdschlüssel von } E_2 : D_2]\}$$

bzw.

$$R : \{[\text{Fremdschlüssel von } E_1 : D_1, \underline{\text{Fremdschlüssel von } E_2 : D_2}]\}$$

Erstes Beispiel

Modellierung der Ministerpräsidenten der Bundesländer:



Eine relativ schlechte Umsetzung ist:

Bundesland : {[Name : string, ...]}

Person : {[SozVersNr : string, Name : string, MinisterpräsidentVon : string, ...]}

Diese hat den Nachteil vieler Nullwerte für das Attribut *MinisterpräsidentVon*.

Eine bessere Umsetzung, bei der Nullwerte vermieden werden, ist:

Bundesland : {[Name : string, ..., Ministerpräsident : string]}

Person : {[SozVersNr : string, Name : string, ...]}

Zweites Beispiel

Modellierung von Eheschließungen:



Möglichkeiten, dies in das relationale Modell überzuführen, sind:

Männer : {[Name : string, ..., verheiratetMit : string]}

Frauen : {[Name : string, ...]}

oder

Männer : {[Name : string, ...]}

Frauen : {[Name : string, ..., verheiratetMit : string]}

Zudem kann der Beziehungstyp auch als eigene Relation im relationalen Modell realisiert werden:

Männer : {[Name : string, ...]}

Frauen : {[Name : string, ...]}

verheiratetMit : {[FName : string, MName : string]} oder

verheiratetMit : {[FName : string, MName : string]}

Hinweis: Man muss für die Relation *verheiratetMit* tatsächlich beide Schlüsselkandidaten anwenden, um die 1:1-Beziehung auszudrücken. Wählt man z.B. nur *FName* als Schlüsselkandidat, so ist es möglich, dass ein Mann mit mehreren Frauen verheiratet ist. Die Konsistenzbedingung wäre damit verletzt.

Damit ist nicht gemeint, dass $\{FName, MName\}$ den Schlüssel bildet – denn damit würde man eine allgemeine N:M-Beziehung modellieren. Vielmehr müssen beide Schlüsselinformationen in der Datenbank getrennt realisiert werden. Man wählt z.B. *FName* als *primary key* und setzt bzgl. *MName* das Constraint **unique**.

Vergleicht man die dritte Realisierung mit den beiden vorangegangenen, so ergeben sich folgende Vorteile:

- keine Nullwerte bei Personen, die nicht verheiratet sind,
- schnelleres Durchsuchen der Beziehung (mit Indexunterstützung),
- die Suche nach Ehepartnern wird in beide Richtungen gleich gut unterstützt.

Abhängig von der tatsächlichen Ausprägung kann sowohl die eine, wie auch die andere Realisierung eine Speicherplatzersparnis bewirken, so dass dies weder als Vorteil noch als Nachteil gewertet wird.

Aufgabe 2

- (a) Ermitteln Sie den Bekanntheitsgrad der Professoren unter den Studenten, wobei wir annehmen, dass Studenten die Professoren nur durch Vorlesungen oder Prüfungen kennenlernen.

Eine mögliche Lösung der Aufgabe ist:

```
create view kenntprof as
  select h.MatrNr, v.gelesenVon as PersNr
  from hören h, Vorlesungen v
  where v.VorlNr = h.VorlNr
  union
  select p.MatrNr, p.PersNr
  from prüfen p;

select k.PersNr, p.Name,
  cast(count(*) as decimal) /
    cast(alle.AnzStud as decimal) as grad
from kenntprof k, Professoren p,
  (select count(*) as AnzStud from Studenten) alle
where k.PersNr = p.PersNr
group by k.PersNr, p.Name, alle.AnzStud
order by grad desc;
```

Zu beachten ist, dass Studenten einen Professor / eine Professorin nur einmal kennen können. Das heißt, auch wenn sie bei ihm / ihr eine Vorlesung hören und bei ihm / ihr geprüft wurden, kennen sie ihn / sie trotzdem nur einmal. Duplikate werden bei Verwendung von **union** (im Gegensatz zu **union all**) bereits eliminiert.

SQL bietet mit **distinct** oder auch **unique** weitere Möglichkeiten, gleiche Einträge zu verhindern. Angenommen, *kenntprof* wäre nicht duplikatfrei, könnte man die Anfrage wie folgt formulieren:

```
select k.PersNr, p.Name, count(unique MatrNr) as grad
from kenntprof k, Professoren p
where k.PersNr = p.PersNr
```

```

group by k.PersNr , p.Name
order by grad desc;

```

- (b) Ermitteln Sie für die einzelnen Vorlesungen die Durchfallquote als die Anzahl der durchgefallenen Prüflinge relativ zur Anzahl der für diese Vorlesung angetretenen Prüflinge.

Als Variation der obigen Anfrage ermitteln Sie die Durchfallquote bei den einzelnen Professoren.

Durchfallquote pro Vorlesung

Um diese Anfrage zu beantworten, erstellen wir zuerst zwei Views, *durchgefallene* und *alle*. Mit der ersten Sicht ermitteln wir für jede Vorlesung die Anzahl der Studenten, die eine Prüfung nicht bestanden haben. Der Sonderfall, dass kein Student in einer Prüfung über die jeweilige Vorlesung durchgefallen ist, muss dabei auch berücksichtigt werden. Mit der zweiten View fragen wir ab, wie viele Studenten eine Prüfung über jede Vorlesung abgelegt haben. Der Sonderfall, dass kein Student sich über eine bestimmte Vorlesung hat prüfen lassen, muss hier nicht berücksichtigt werden. Dies erfolgt in der anschließenden Anfrage.

```

create view durchgefallene as
(select VorlNr, count(Note) as Anzahl
 from prüfen
 where Note > 4.0
 group by VorlNr)
union
(select VorlNr, 0 as Anzahl
 from Vorlesungen
 where VorlNr not in (select VorlNr
                      from prüfen
                      where Note > 4.0));

create view alle as
select VorlNr, count(*) as Anzahl
 from prüfen
 group by VorlNr;

```

Basierend auf diesen Views lässt sich die Aufgabe damit wie folgt beantworten:

```

(select a.VorlNr, float(d.Anzahl)/float(a.Anzahl) as Grad
 from alle a, durchgefallene d
 where a.VorlNr = d.VorlNr)
union
(select VorlNr, 0 as Grad
 from Vorlesungen
 where VorlNr not in (select VorlNr
                      from prüfen));

```

Wie zuvor angeführt, wird der Sonderfall, dass keine Studenten eine bestimmte Vorlesung prüfen ließen, durch die zweite Ergebnismenge behandelt.

Eine alternative, in gewisser Weise elegantere, da kürzere Formulierung sieht wie folgt aus:

```

create view Vorl_AnzDurch_AnzAlle as
  select VorlNr, count(*) as AnzAlle,
         sum(case when Note > 4.0 then 1 else 0 end) as AnzDurch
  from   prüfen
  group by VorlNr;

(select VorlNr, float(AnzDurch)/float(AnzAlle) as Grad
  from Vorl_AnzDurch_AnzAlle)
union
(select VorlNr, 0 as Grad
  from Vorlesungen
  where VorlNr not in (select VorlNr from prüfen));

```

Durchfallquote pro Professor

Auch für diese Anfrage erstellen wir uns zuerst zwei temporäre Views. *durchgefalleneProProf* bestimmt je Professor die Anzahl der Studenten, die Prüfungen bei ihm/ihr nicht bestanden haben, wohingegen *alleProProf* für jeden Professor die Anzahl abgenommener Prüfungen zählt (erneut werden die Ereignisse mit leerem Ergebnis hier noch nicht betrachtet).

```

create view durchgefalleneProProf as
  (select PersNr, count(Note) as Anzahl
   from   prüfen
   where  Note > 4.0
   group by PersNr)
union
  (select PersNr, 0 as Anzahl
   from   Professoren
   where  PersNr not in
         (select PersNr from prüfen where Note > 4.0));

create view alleProProf as
  select PersNr, count(*) as Anzahl
  from   prüfen
  group by PersNr;

```

Analog sieht die Anfrage dann wie folgt aus:

```

(select a.PersNr, float(d.Anzahl)/float(a.Anzahl) as Grad
  from alleProProf a, durchgefalleneProProf d
  where a.PersNr = d.PersNr)
union
  (select PersNr, 0 as Grad
   from Professoren
   where PersNr not in (select PersNr from prüfen));

```

Auch hier kann wieder eine entsprechend kürzere Lösung wie im vorhergehenden Fall erstellt werden.

Aufgabe 3

Beschreiben Sie die Auswirkungen der folgenden Operationen wenn die Beispielausprägung aus Abbildung 1 mit dem Schema aus Abbildung 2 vorgegeben sind:

- **delete from** Vorlesungen **where** Titel = 'Ethik';
- **insert into** prüfen **values** (24002, 5001, 2138, 2.0);
- **insert into** prüfen **values** (28106, 5001, 2127, 4.3);
- **drop table** Studenten;
- Löschen der Vorlesung Ethik:
 - Auf diese Vorlesung (*VorlNr* 5041) existieren Referenzen in den Tabellen *hören*, *voraussetzen* und *prüfen*.
 - Referenzen in *hören* und *voraussetzen* sind als **on delete cascade** definiert, d.h., bei Löschen der Vorlesung werden die entsprechenden Tupel in diesen Relationen gelöscht.
 - In *prüfen* ist *VorlNr* ein Fremdschlüssel auf *Vorlesungen*. Es ist kein spezielles Verhalten definiert. Das Default-Verhalten der Datenbanksysteme verhindert jedoch das Entstehen ungültiger Referenzen (sog. *dangling references*) und blockiert die Löschoperation.
Dieses Verhalten kann man mit dem Constraint **on delete no action** auch explizit erzwingen.
- Erstes Einfügen eines prüfen-Tupels:
Das Einfügen scheitert, da es keinen Professor mit PersNr 2138 gibt, die Tabelle *prüfen* jedoch mit dem Constraint **references** Professoren **on delete ...** spezifiziert ist. Bei **insert**-Anweisungen wird also sichergestellt, dass der Eintrag existiert.
- Zweites Einfügen eines prüfen-Tupels:
prüfen hat den Primärschlüssel (*MatrNr*, *VorlNr*). Da bereits ein Tupel mit (28106, 5001) als Primärschlüsselausprägung in *prüfen* existiert, scheitert auch diese **insert**-Anweisung.
- Löschen der Tabelle Studenten:
Der **drop table**-Befehl scheitert, da in den Tabellendefinitionen von *prüfen* und *hören* **references**-Constraints definiert wurden, die ein Löschen verhindern. Dies ist auch dann der Fall, wenn die Constraints mit **on delete cascade** oder **on delete set null** definiert wurden – auch dann, wenn die Tabelle *Studenten* leer ist.
Man könnte also alle Studenten in der Tabelle *Studenten* löschen; nicht jedoch die Tabelle selbst.

Aufgabe 4

Gegeben sei das folgende Schema:

- Familie: {[Opa, Oma, Vater, Mutter, Kind]}

Hierbei sei vereinfachend vorausgesetzt, dass Personen eindeutig durch ihren Vornamen identifiziert werden. Für ein Tupel [Theo, Martha, Herbert, Maria, Else] soll gelten, dass Theo und Martha entweder die Eltern von Herbert oder von Maria sind – die Großeltern werden also immer als Paar gespeichert, ohne dass ersichtlich ist, ob es die Großeltern väterlicher- oder mütterlicherseits sind. Wir gehen weiterhin davon aus, dass zu einem Kind immer beide Elternteile und beide Großeltern-Paare (also sowohl mütterlicherseits als auch väterlicherseits) bekannt sind.

Professoren				Studenten		
PersNr	Name	Rang	Raum	MatrNr	Name	Semester
2125	Sokrates	C4	226	24002	Xenokrates	18
2126	Russel	C4	232	25403	Jonas	12
2127	Kopernikus	C3	310	26120	Fichte	10
2133	Popper	C3	52	26830	Aristoxenos	8
2134	Augustinus	C3	309	27550	Schopenhauer	6
2136	Curie	C4	36	28106	Carnap	3
2137	Kant	C4	7	29120	Theophrastos	2
				29555	Feuerbach	2

Vorlesungen				voraussetzen	
VorlNr	Titel	SWS	gelesenVon	Vorgänger	Nachfolger
5001	Grundzüge	4	2137	5001	5041
5041	Ethik	4	2125	5001	5043
5043	Erkenntnistheorie	3	2126	5001	5049
5049	Mäeutik	2	2125	5041	5216
4052	Logik	4	2125	5043	5052
5052	Wissenschaftstheorie	3	2126	5041	5052
5216	Bioethik	2	2126	5052	5259
5259	Der Wiener Kreis	2	2133		
5022	Glaube und Wissen	2	2134		
4630	Die 3 Kritiken	4	2137		

hören		Assistenten			
MatrNr	VorlNr	PersNr	Name	Fachgebiet	Boss
26120	5001	3002	Platon	Ideenlehre	2125
27550	5001	3003	Aristoteles	Syllogistik	2125
27550	4052	3004	Wittgenstein	Sprachtheorie	2126
28106	5041	3005	Rhetikus	Planetenbewegung	2127
28106	5052	3006	Newton	Keplersche Gesetze	2127
28106	5216	3007	Spinoza	Gott und Natur	2134
28106	5259				
29120	5001				
29120	5041				
29120	5049				
29555	5022				
25403	5022				
29555	5001				

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Abbildung 1: Beispielausprägung unserer Universitäts-Datenbank

- Bestimmen Sie alle FDs.
- Bestimmen Sie den/die Kandidatenschlüssel der Relation *Familie*.
- Überführen Sie das Schema unter Verwendung des Synthesealgorithmus in die dritte Normalform.

```

create table Studenten
  ( MatrNr      integer primary key,
    Name        varchar(30) not null,
    Semester    integer check (Semester between 1 and 13));

create table Professoren
  ( PersNr      integer primary key,
    Name        varchar(30) not null,
    Rang        character(2) check (Rang in ('C2', 'C3', 'C4')),
    Raum        integer unique);

create table Assistenten
  ( PersNr      integer primary key,
    Name        varchar(30) not null,
    Fachgebiet  varchar(30),
    Boss        integer,
    foreign key (Boss) references Professoren on delete set null);

create table Vorlesungen
  ( VorlNr      integer primary key,
    Titel       varchar(30),
    SWS         integer,
    gelesenVon  integer references Professoren on delete set null);

create table hören
  ( MatrNr      integer references Studenten on delete cascade,
    VorlNr      integer references Vorlesungen on delete cascade,
    primary key (MatrNr, VorlNr));

create table voraussetzen
  ( Vorgänger   integer references Vorlesungen on delete cascade,
    Nachfolger   integer references Vorlesungen on delete cascade,
    primary key (Vorgänger, Nachfolger));

create table prüfen
  ( MatrNr      integer references Studenten on delete cascade,
    VorlNr      integer references Vorlesungen,
    PersNr      integer references Professoren on delete set null,
    Note        numeric(2,1) check (Note between 0.7 and 5.0),
    primary key (MatrNr, VorlNr));

```

Abbildung 2: Das vollständige Universitätsschema mit Integritätsbedingungen

- Überführen Sie das Schema unter Verwendung des Dekompositionsalgorithmus in die BCNF.

Beispielausprägung

Als Beispiel betrachten wir den Stammbaum der griechischen Götter und Helden. Ein kleiner Ausschnitt davon ist in Abbildung 3 gezeigt und in nachfolgender Tabelle skizziert:

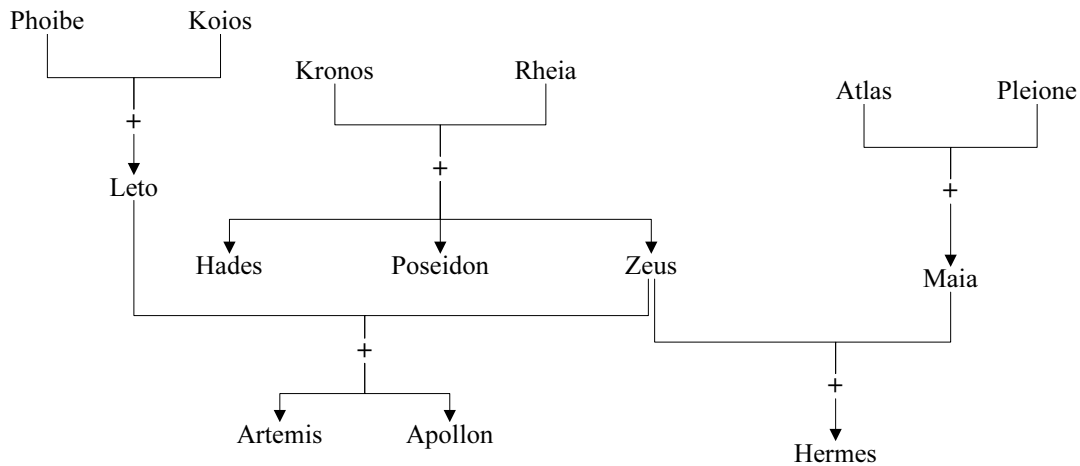


Abbildung 3: Ausschnitt aus dem Stammbaum der griechischen Götter und Helden

Familie				
Opa	Oma	Vater	Mutter	Kind
⋮	⋮	⋮	⋮	⋮
Kronos	Rheia	Zeus	Leto	Artemis
Kronos	Rheia	Zeus	Leto	Apollon
Koios	Phoibe	Zeus	Leto	Artemis
Koios	Phoibe	Zeus	Leto	Apollon
Kronos	Rheia	Zeus	Maia	Hermes
Atlas	Pleione	Zeus	Maia	Hermes
⋮	⋮	⋮	⋮	⋮

Bereits diese kleine Ausprägung zeigt, dass das Schema *Familie* nicht unsere Qualitätsanforderungen für einen guten Datenbankentwurf erfüllt, da Information (z.B. über die Großeltern) redundant abgespeichert wird.

Geltende funktionale Abhängigkeiten

Folgende funktionale Abhängigkeiten drücken bekannte Beziehungen zwischen Familienmitgliedern aus:

$$\{\text{Kind}\} \rightarrow \{\text{Mutter, Vater}\} \quad (1)$$

$$\{\text{Kind, Oma}\} \rightarrow \{\text{Opa}\} \quad (2)$$

$$\{\text{Kind, Opa}\} \rightarrow \{\text{Oma}\} \quad (3)$$

FD (1) drückt aus, dass Kinder eindeutig ihre Eltern bestimmen. FD (2) und FD (3) sagen aus, dass Kinder und ein Großelternanteil (väterlicherseits oder mütterlicherseits) den zweiten Großelternanteil eindeutig festlegen.

Ferner sind die folgenden funktionalen Abhängigkeiten gültig:

$$\{\text{Mutter, Vater, Opa}\} \rightarrow \{\text{Oma}\} \quad (4)$$

$$\{\text{Mutter, Vater, Oma}\} \rightarrow \{\text{Opa}\} \quad (5)$$

FD (4) und FD (5) sagen aus, dass Geschwister, die dieselben Eltern haben (also keine Halbgeschwister sind), auch dieselben Großeltern haben müssen.

Bestimmen der Kandidatenschlüssel

$\{Kind, Oma\}$ und $\{Kind, Opa\}$ sind Kandidatenschlüssel. Für beide ist die Attributhülle jeweils die komplette Familie. Diese Schlüsselkandidaten können nicht weiter verkleinert werden, da die Attributhülle von $Kind$ nur $\{Kind, Mutter, Vater\}$ und die Attributhülle von Oma nur $\{Oma\}$ ist. Entsprechendes gilt für die Attributhülle von Opa , die $\{Opa\}$ ist.

Überführen des Schemas in die 3. Normalform:

1. Kanonische Überdeckung

1. *Linksreduktion:* Die FD (1) kann nicht weiter linksreduziert werden.

Betrachte FD (2):

- Ist Kind überflüssig?
 $Opa \notin AttrHülle(F, \{Oma\})$
- Ist Oma überflüssig?
 $Opa \notin AttrHülle(F, \{Kind\})$.

Betrachte FD (3):

- Ist Kind überflüssig?
 $Oma \notin AttrHülle(F, \{Opa\})$
- Ist Opa überflüssig?
 $Oma \notin AttrHülle(F, \{Kind\})$.

Betrachte FD (4) und (5): Diese FDs lassen sich auch nicht linksreduzieren.

Das bisherige Zwischenergebnis ist also unverändert.

2. *Rechtsreduktion:*

Betrachte FD (1):

- Ist Mutter überflüssig?
 $Mutter \notin AttrHülle(F - FD(1) \cup (\{Kind\} \rightarrow \{Vater\}), \{Kind\})$
- Ist Vater überflüssig?
 $Vater \notin AttrHülle(F - FD(1) \cup (\{Kind\} \rightarrow \{Mutter\}), \{Kind\})$

Betrachte FD (2):

- Ist Opa überflüssig?
 $Opa \in AttrHülle(F - FD(2) \cup (\{Kind, Oma\} \rightarrow \emptyset), \{Kind, Oma\})$,
da $Kind \rightarrow \{Mutter, Vater\}, \{Mutter, Vater, Oma\} \rightarrow Opa$
Damit erhält man für FD (2): $\{Kind, Oma\} \rightarrow \emptyset$.

Betrachte FD (3):

- Ist Oma überflüssig?
 $Oma \in AttrHülle(F - FD(3) \cup (\{Kind, Opa\} \rightarrow \emptyset), \{Kind, Opa\})$,
da $\{Kind\} \rightarrow \{Mutter, Vater\}, \{Mutter, Vater, Opa\} \rightarrow \{Oma\}$
Damit erhält man für FD (3): $\{Kind, Opa\} \rightarrow \emptyset$.

Betrachte FD (4) und FD (5): Diese FDs sind nicht weiter rechtsreduzierbar.

Bisheriges Zwischenergebnis:

$$\begin{aligned}
 \{\text{Kind}\} &\rightarrow \{\text{Mutter, Vater}\} \\
 \{\text{Kind, Oma}\} &\rightarrow \emptyset \\
 \{\text{Kind, Opa}\} &\rightarrow \emptyset \\
 \{\text{Mutter, Vater, Opa}\} &\rightarrow \{\text{Oma}\} \\
 \{\text{Mutter, Vater, Oma}\} &\rightarrow \{\text{Opa}\}
 \end{aligned}$$

3. *Entferne die FDs der Form $\alpha \rightarrow \emptyset$.*

FDs (2) und (3) werden eliminiert.

Bisheriges Zwischenergebnis:

$$\begin{aligned}
 \{\text{Kind}\} &\rightarrow \{\text{Mutter, Vater}\} \\
 \{\text{Mutter, Vater, Opa}\} &\rightarrow \{\text{Oma}\} \\
 \{\text{Mutter, Vater, Oma}\} &\rightarrow \{\text{Opa}\}
 \end{aligned}$$

4. *Fasse mittels der Vereinigungsregel FDs der Form $\alpha \rightarrow \beta_1, \dots, \alpha \rightarrow \beta_n$ zusammen.*

Es werden keine FDs vereinigt, da es keine zwei FDs mit gleicher linker Seite gibt.

F_c ist eine kanonische Überdeckung zur Ausgangsmenge F .

2. Erstellung der Relationenschemata aus der kanonischen Überdeckung

$$\begin{aligned}
 \text{aus (1)} &\Rightarrow R_1 : \{\{\underline{\text{Kind}}, \text{Mutter, Vater}\}\} \\
 \text{aus (4)} &\Rightarrow R_2 : \{\{\underline{\text{Mutter, Vater, Opa}}, \text{Oma}\}\} \\
 \text{aus (5)} &\Rightarrow R_3 : \{\{\underline{\text{Mutter, Vater, Oma}}, \text{Opa}\}\}
 \end{aligned}$$

3. Erstellung eines weiteren Relationenschemas, das den Schlüssel der ursprünglichen Relation enthält, falls kein solches existiert

Wir erstellen ein weiteres Schema $R_4 : \{\{\underline{\text{Kind}}, \text{Oma}\}\}$.

4. Eliminierung der Schemata, die in anderen Relationenschemata enthalten sind.

Das Relationenschema R_3 ist im Relationenschema R_2 enthalten und kann somit eliminiert werden.

Das Ergebnis ist wie folgt:

$$\begin{aligned}
 \text{aus (1)} &\Rightarrow R_1 : \{\{\underline{\text{Kind}}, \text{Mutter, Vater}\}\} \\
 \text{aus (4)} &\Rightarrow R_2 : \{\{\underline{\text{Mutter, Vater, Opa}}, \text{Oma}\}\} \\
 \text{aus Schlüssel} &\Rightarrow R_4 : \{\{\underline{\text{Kind}}, \text{Oma}\}\}
 \end{aligned}$$

Die Zerlegung in 3NF ist verlustlos und abhängigkeiterhaltend.

Überführen des Schemas in die BCNF:

Wir betrachten das ursprüngliche Relationenschema *Familie* und die darauf geltenden FDs (ohne Erzeugung der kanonischen Überdeckung).

Betrachte FD (1):

- Für die FD gilt: sie ist nicht trivial, rechte und linke Seite sind disjunkt und {Kind} ist nicht Superschlüssel.
- Die ursprüngliche Relation wird in die folgenden zwei neuen Relationen aufgespaltet:

$$R_1 : \{[\underline{\text{Kind}}, \text{Mutter}, \text{Vater}]\}$$

$$R_2 : \{[\underline{\text{Kind}}, \text{Oma}, \text{Opa}]\}$$

Betrachte FD (2): {Kind, Oma} ist ein Superschlüssel.

Betrachte FD (3): {Kind, Opa} ist ein Superschlüssel.

Betrachte FD (4) und (5): Diese FDs können keiner der Relationen mehr zugeordnet werden.

Das Endergebnis ist also:

$$R_1 : \{[\underline{\text{Kind}}, \text{Mutter}, \text{Vater}]\}$$

$$R_2 : \{[\underline{\text{Kind}}, \text{Oma}, \text{Opa}]\}$$

Die Zerlegung war nicht abhängigkeiterhaltend.