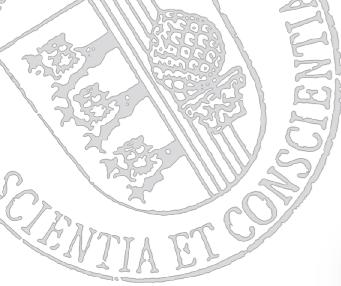
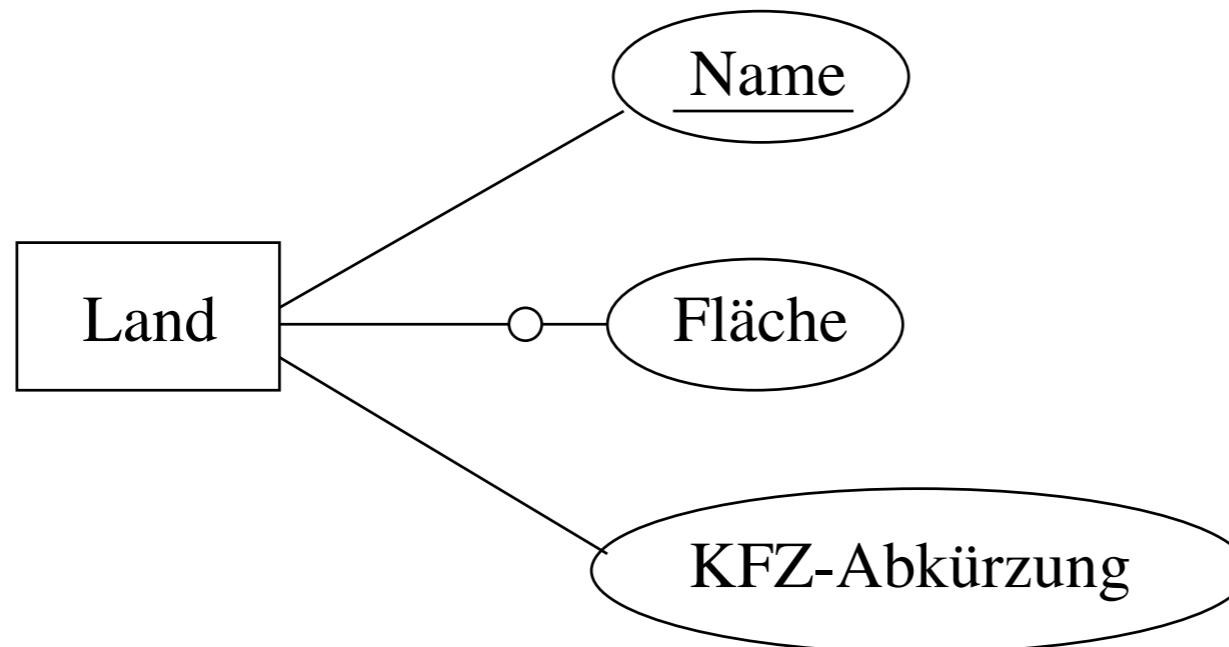


4.3 Transformation von ER nach SQL



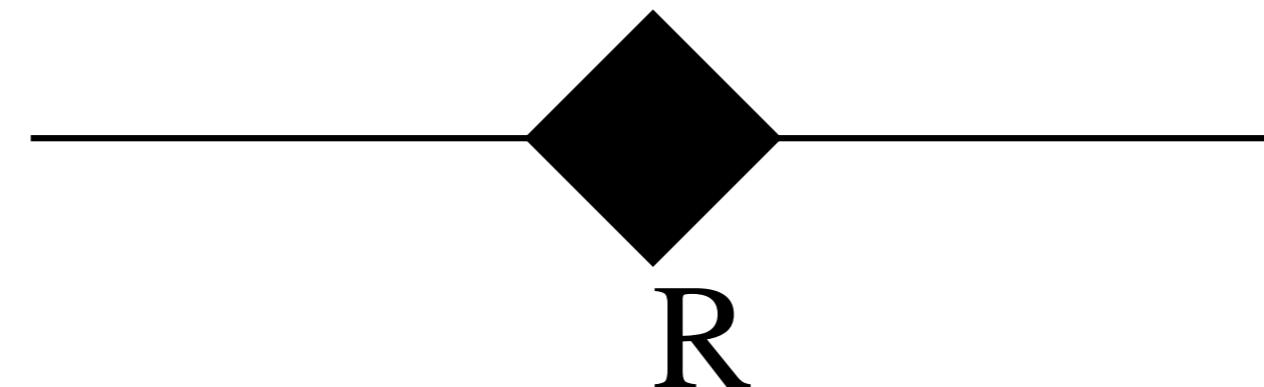
- Aus jedem *Entity-Typ* $E(A_1 : \text{dom}(A_1), \dots, A_n : \text{dom}(A_n))$ wird eine Relation $R(A_1 : \text{dom}(A_1), \dots, A_n : \text{dom}(A_n))$
- Falls dabei ein Relationship-Typ $E \text{ } isa \text{ } F$ vorliegt, kann man die Attributvererbung durch Hinzunahme aller Schlüsselattribute von F berücksichtigen
- Eine **Relationship R** zwischen den Entity-Typen E_1, \dots, E_n wird dargestellt durch ein **Relationenschema R**, dessen Attribute aus allen Schlüsselattributen der E_i bestehen. Gleiche Attribute werden dabei durch Umbenennung in R eindeutig gemacht. Falls R eigene Attribute besitzt, nimmt man diese hinzu.



Schlüssel und Optionalitäten berücksichtigen!

```
CREATE TABLE Land (
    Name          VARCHAR(20) PRIMARY KEY,
    Fläche        INTEGER,
    KFZ-Abkürzung VARCHAR(4) NOT NULL UNIQUE
);
```

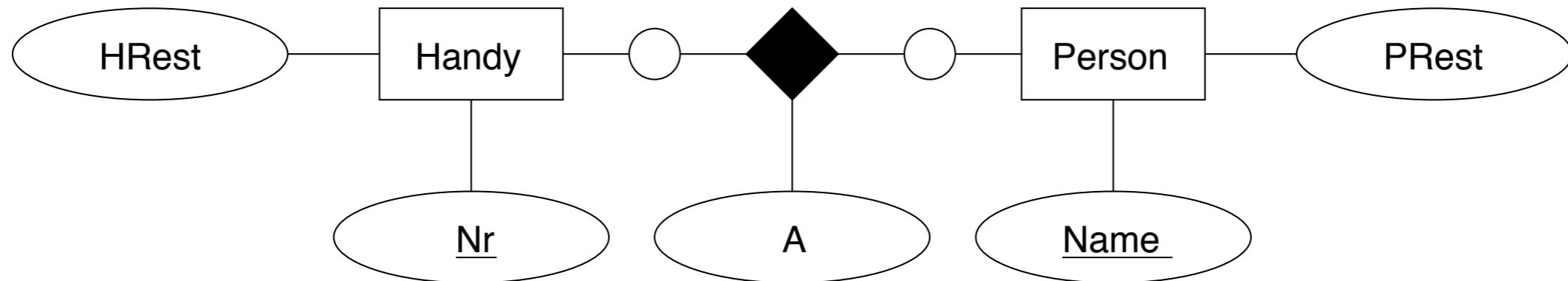
n:m Relationships



Relationships: n:m

allgemeiner Fall, beide Teilnahmen optional

Eine Person besitzt mehrere Handys, ein Handy hat mehrere Besitzer



```
CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL);
```

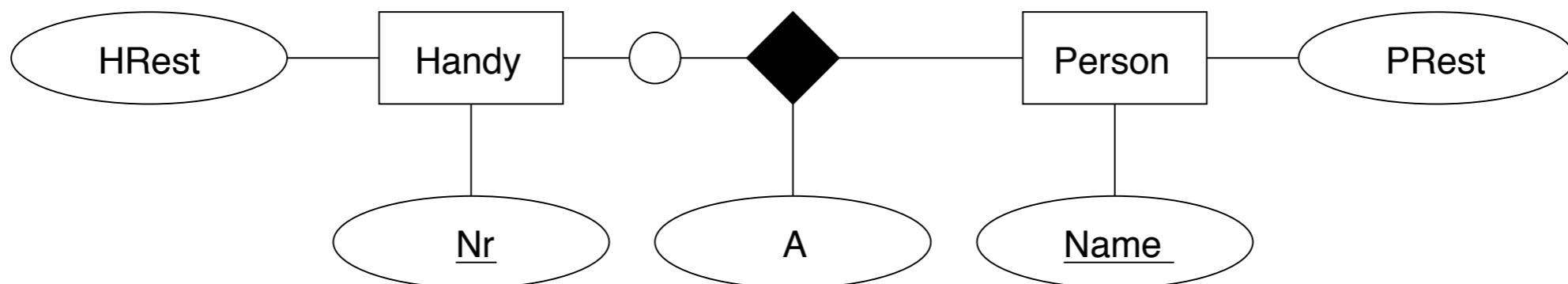
```
CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL);
```

Besitzt eine der Entity-Tabellen neben den Schlüsselattributen keine weiteren Attribute, so kann sie mit der Relationshipstabelle zusammengefaßt werden (häufig bei mehrwertigen Attributten).

```
CREATE TABLE R (
    HNr ... NOT NULL REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL,
    PRIMARY KEY (HNr, PName)
);
```

HNr	PName
1234	Endres
1234	Mandl
5678	Endres

**Ein Handy hat immer mind. einen Besitzer,
eine Person aber nicht unbedingt ein Handy**



```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL);
  
```

```

CREATE TABLE Handy (
    Nr ... PRIMARY KEY REFERENCES R(HNr),
    HRest ... NOT NULL);
  
```

```

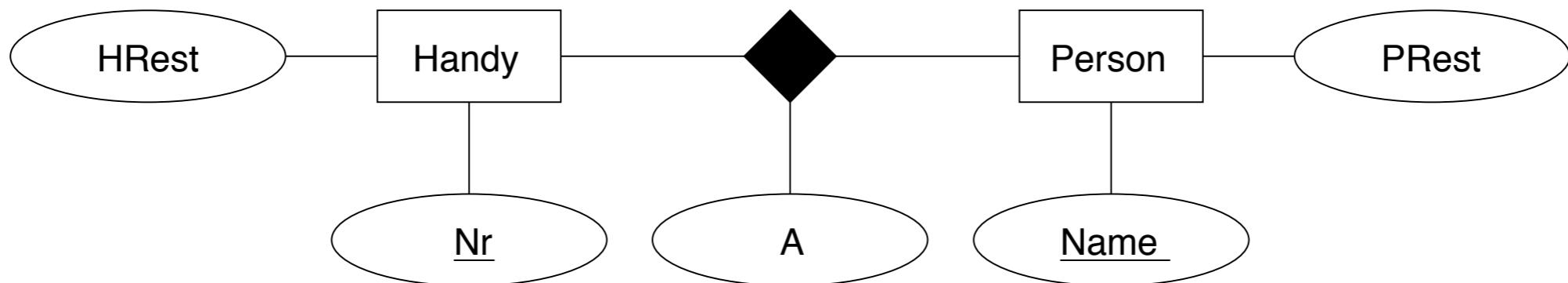
CREATE TABLE R (
    HNr ... NOT NULL REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL,
    PRIMARY KEY (HNr, PName)
  
```

HNr	PName
1234	Endres
1234	Mandl
5678	Endres

Relationships: n:m

keine Optionalität

**Ein Handy hat immer mind. einen Besitzer,
eine Person immer mind. ein Handy**

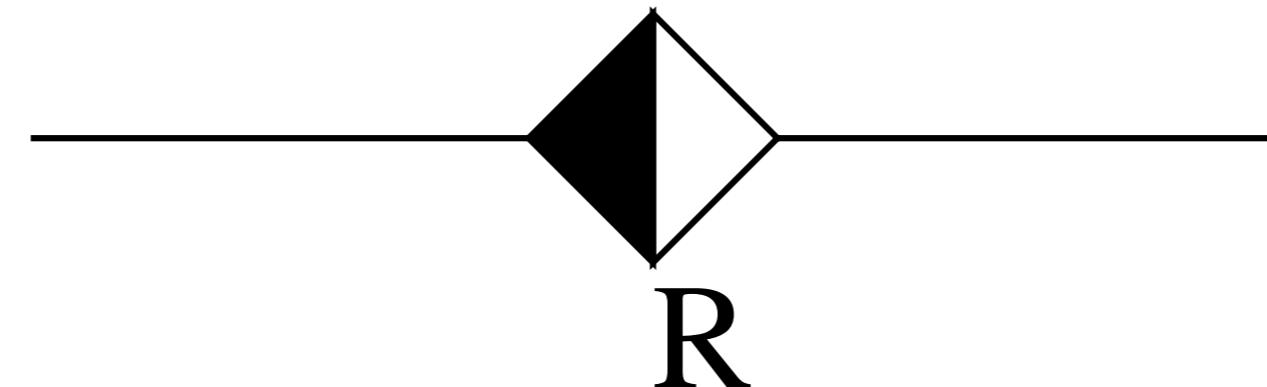


```
CREATE TABLE Person (
    Name ... PRIMARY KEY REFERENCES R(PName),
    PRest ... NOT NULL);
```

```
CREATE TABLE Handy (
    Nr ... PRIMARY KEY REFERENCES R(HNr),
    HRest ... NOT NULL );
```

```
CREATE TABLE R (
    HNr ... NOT NULL REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL,
    PRIMARY KEY (HNr, PName)
);
```

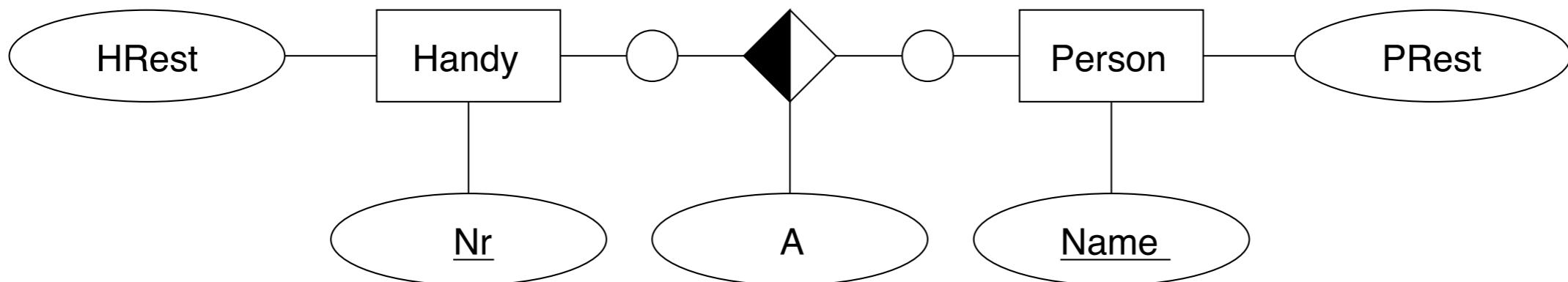
n:1 Relationships



Relationships: n:1

beide Teilnahmen optional

Ein Handy braucht keinen Besitzer,
eine Person kann (muss aber nicht) mehrere Handys haben.



```
CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL);
```

```
CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL);
```

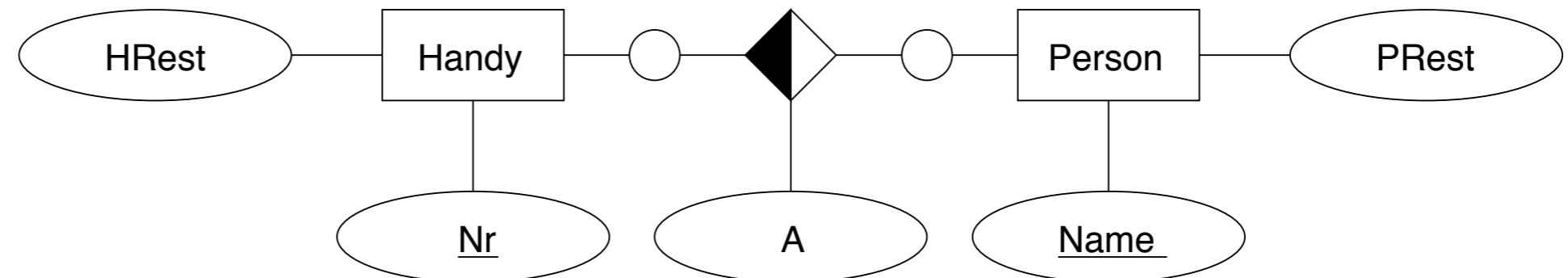
```
CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL
);
```

HNr	PName
1234	Endres
4444	Mandl
5678	Endres

Relationships: n:1

beide Teilnahmen optional

Tabelle **R** kann mit **Handy** zusammengefasst werden



```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL);
    
```

```

CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL,
    A ...,
    PName ... REFERENCES Person(Name), -- NULL erlaubt
    CHECK ((PName IS NULL AND A IS NULL) OR
           (PName IS NOT NULL AND A IS NOT NULL))
);
    
```

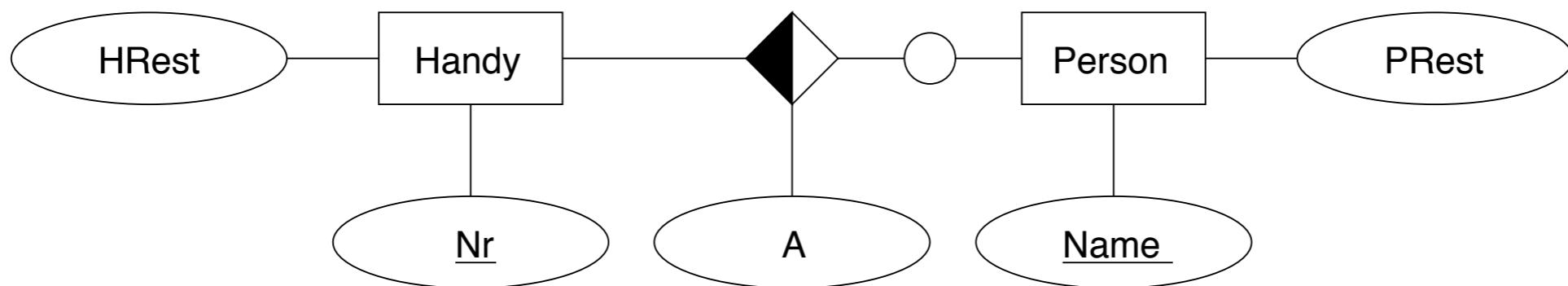
```

CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL
);
    
```

Relationships: n:1

eine Teilnahme optional (1)

**Manche Handys haben keinen Besitzer,
aber alle Personen haben mindestens ein Handy**



```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL
    CHECK ((SELECT COUNT(*)
            FROM R
            WHERE PName = Name) > 0)
);
    
```

```

CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL
);
    
```

```

CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL
);
    
```

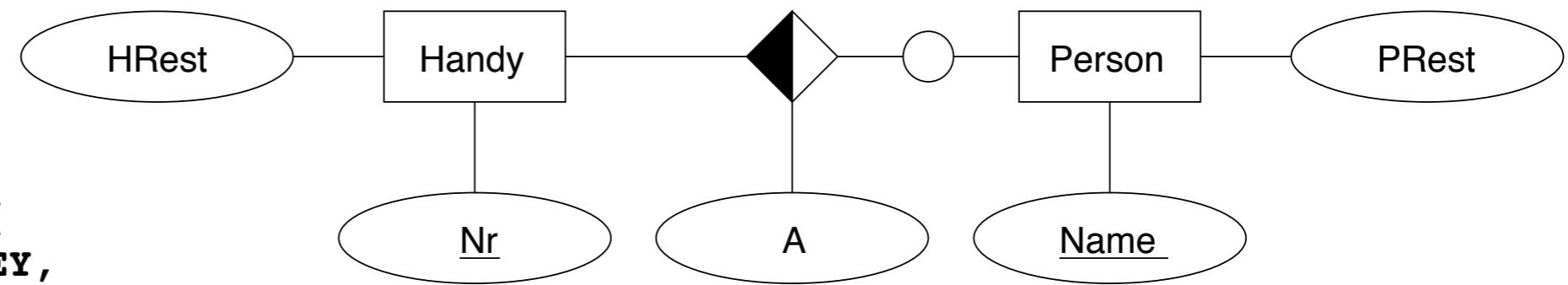
Anmerkung: Kaum eine DB beherrscht Subqueries in CHECK-Klauseln.

Wie kann dieses Problem gelöst werden?

Relationships: n:1

eine Teilnahme optional (1)

Tabelle **R** kann mit Tabelle **Handy** zusammengefasst werden.



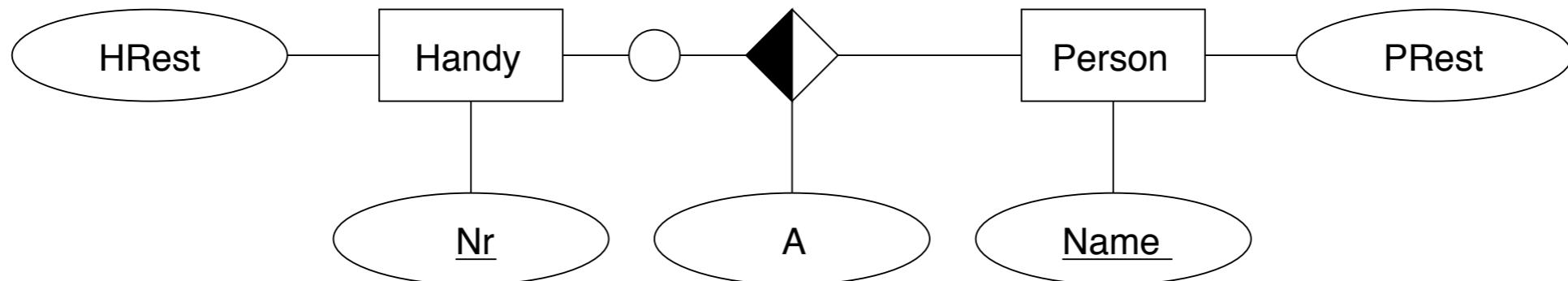
```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL,
    CHECK ((SELECT COUNT(*)
            FROM Handy
            WHERE PName = Name) > 0) );
    
```

```

CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL,
    A ... , -- NULL erlaubt
    PName ... REFERENCES Person(Name), -- NULL erlaubt
    CHECK ((PName IS NULL AND A IS NULL) OR
           (PName IS NOT NULL AND A IS NOT NULL))
);
    
```

**Manche Personen haben keine Mobiltelefone.
Alle Handys haben einen Besitzer.**



```

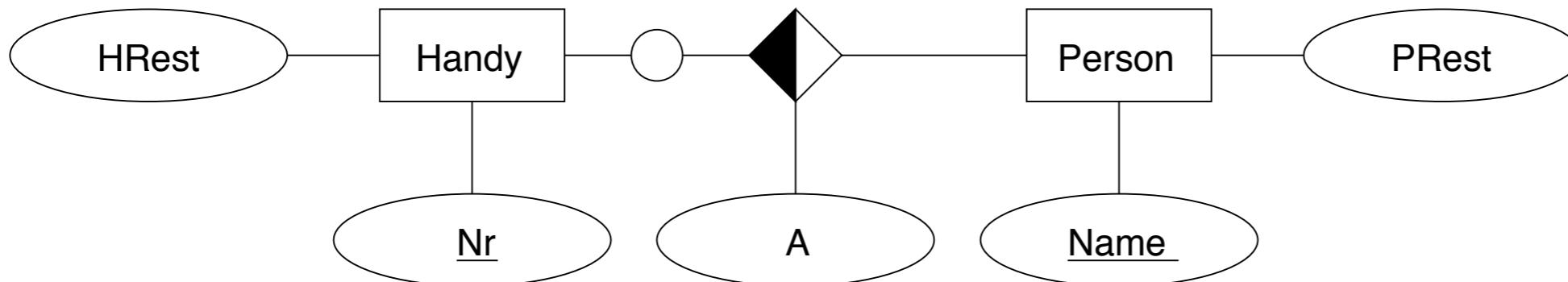
CREATE TABLE Person (
  Name ... PRIMARY KEY,
  PRest ... NOT NULL
);

CREATE TABLE Handy (
  Nr ... PRIMARY KEY REFERENCES R(HNr),
  HRest ... NOT NULL
);
  
```

```

CREATE TABLE R (
  HNr ... PRIMARY KEY REFERENCES Handy(Nr),
  PName ... NOT NULL REFERENCES Person(Name),
  A ... NOT NULL
);
  
```

- Tabelle **R** kann mit Tabelle **Handy** zusammengefasst werden.
- Dadurch entstehen auch keine zyklischen Fremdschlüsselbeziehungen mehr.



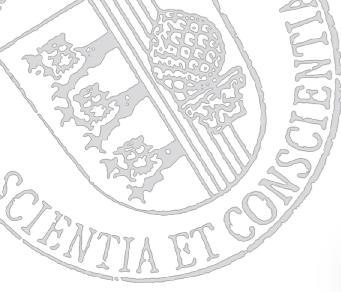
```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL
);

CREATE TABLE Handy (
    Nr ... NOT NULL PRIMARY KEY,
    HRest ... NOT NULL
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL
);
    
```

```

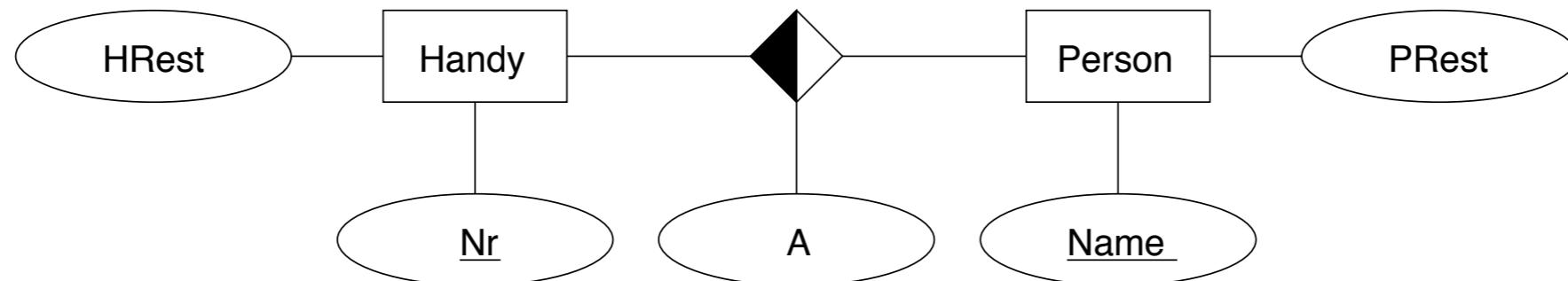
CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL
);
    
```



Relationships: n:1

keine Optionalität

Eine Person hat ein oder mehrere Handys, jedes Handy hat einen Besitzer



```
CREATE TABLE Person (
    Name    ... PRIMARY KEY,
    PRest  ... NOT NULL
    CHECK ((SELECT COUNT(*)
            FROM R
            WHERE PName = Name) > 0
    );
)
```

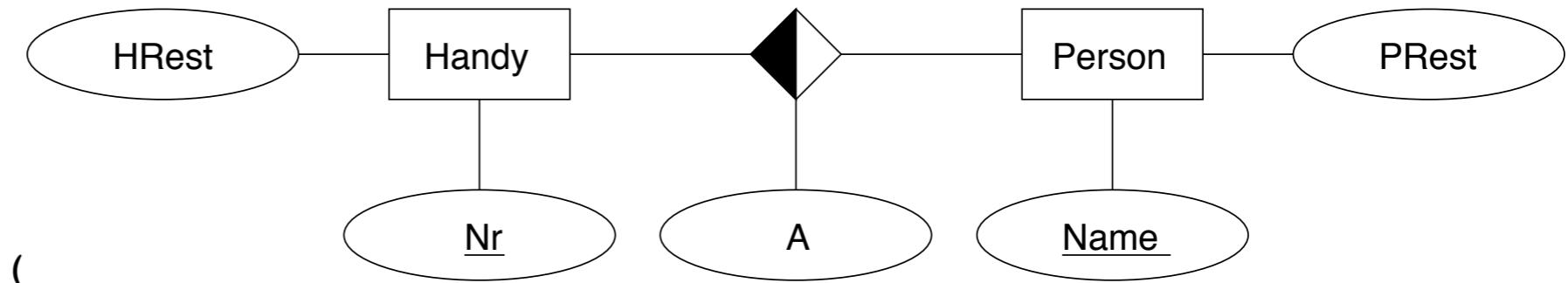
```
CREATE TABLE R (
    HNr      ... PRIMARY KEY REFERENCES Handy(Nr),
    PName   ... NOT NULL REFERENCES Person(Name),
    A        ... NOT NULL
);
```

```
CREATE TABLE Handy (
    Nr      ... PRIMARY KEY REFERENCES R(HNr)
    HRest  ... NOT NULL
);
```

Relationships: n:1

keine Optionalität

Tabelle **R** kann mit Tabelle **Handy** zusammengefasst werden.



```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL
    CHECK ((SELECT COUNT(*) 
            FROM Handy
            WHERE PName = Name) > 0)
);
    
```

```

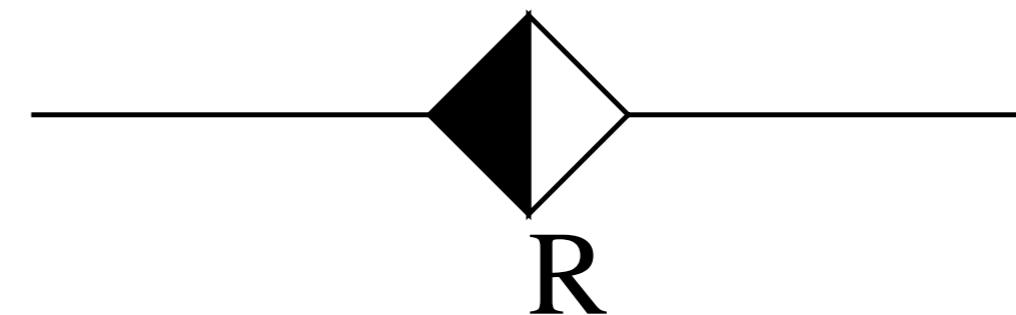
CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL
    A ... NOT NULL,
    PName ... NOT NULL REFERENCES Person(Name)
);
    
```

```

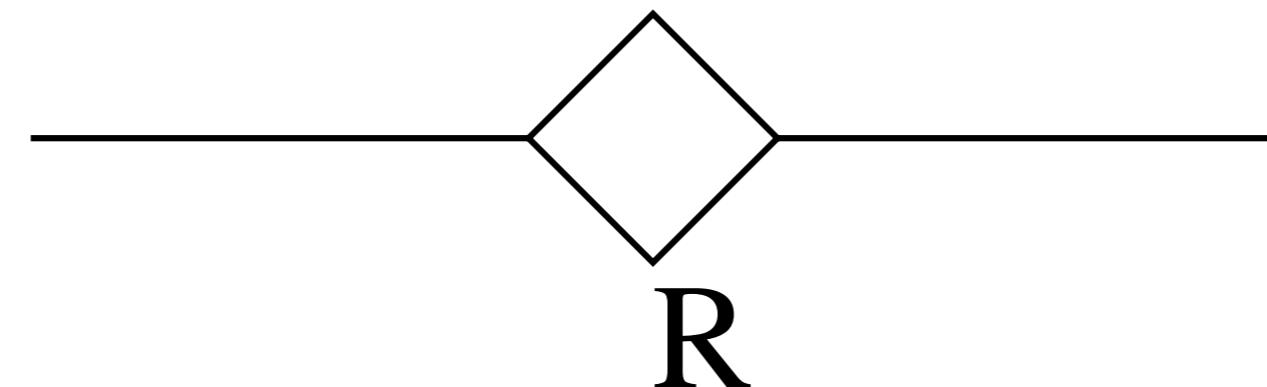
CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL REFERENCES Person(Name),
    A ... NOT NULL
);
    
```

Zusammenfassung

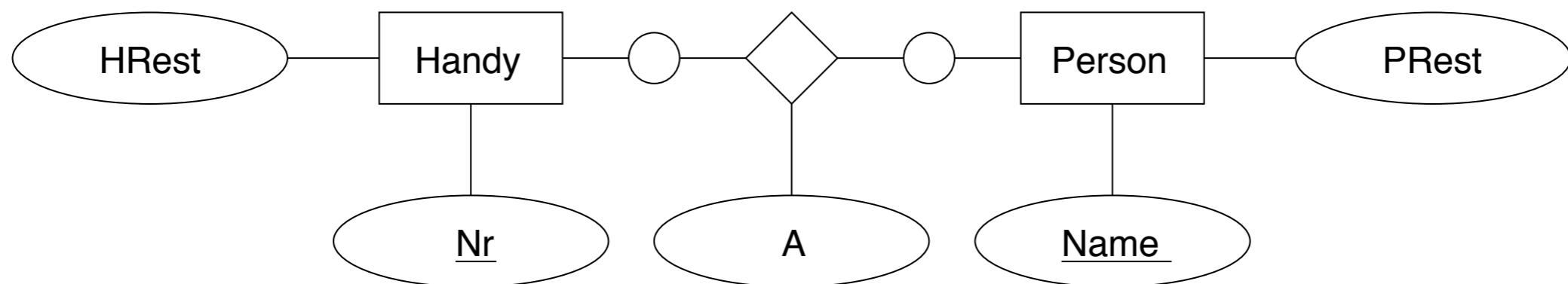
- 1:n–Beziehungen lassen sich wie n:m–Beziehungen immer mit einer Relationstabelle darstellen.
- Man kann die Relationstabelle mit der Entity-Tabelle der n-Seite zusammenfassen.
- Bei Optionalitäten kann durch dann auftretende **NULL**-Werte aber auch Speicher verschwendet werden.
- Entities, die nicht in der Relation vorkommen, enthalten dann jedoch **NULL**-Einträge für Fremdschlüssel und Attribute der Relation.



1:1 Relationships



Ein Handy braucht keinen Besitzer, eine Person kein Handy.



```

CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL);
  
```

```

CREATE TABLE Handy (
    Nr ... PRIMARY KEY,
    HRest ... NOT NULL);
  
```

```

CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL UNIQUE REFERENCES Person(Name),
    A ... NOT NULL );
  
```

Anmerkung: In der Relationstabelle ist es egal, welcher Fremdschlüssel als **PRIMARY KEY** und welcher als **UNIQUE** angegeben wird.

- Eine Tabelle.
- Da beide Entitäten **NUL** sein können, muss ein künstlicher Primärschlüssel eingeführt werden

```
CREATE TABLE PersonHandy (
    ID      ... PRIMARY KEY,
    PName   ... UNIQUE, -- NULL erlaubt
    PRest   ...,        -- NULL erlaubt
    HNr    ... UNIQUE, -- NULL erlaubt
    HRest   ...,        -- NULL erlaubt
    A      ...,        -- NULL erlaubt
    ...
    ...
```

...

```

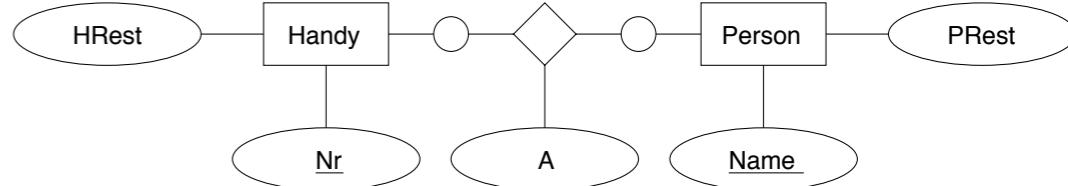
CHECK (
    -- Person konsistent:
    -- "keine Person" erlauben
    ((PName IS NULL AND PRest IS NULL) OR
     -- Person erlauben
     (PName IS NOT NULL AND PRest IS NOT NULL))

AND
    -- Handy konsistent:
    -- "kein Handy" erlauben
    ((HNr IS NULL AND HRest IS NULL) OR
     -- Handy erlauben
     (HNr IS NOT NULL AND HRest IS NOT NULL))

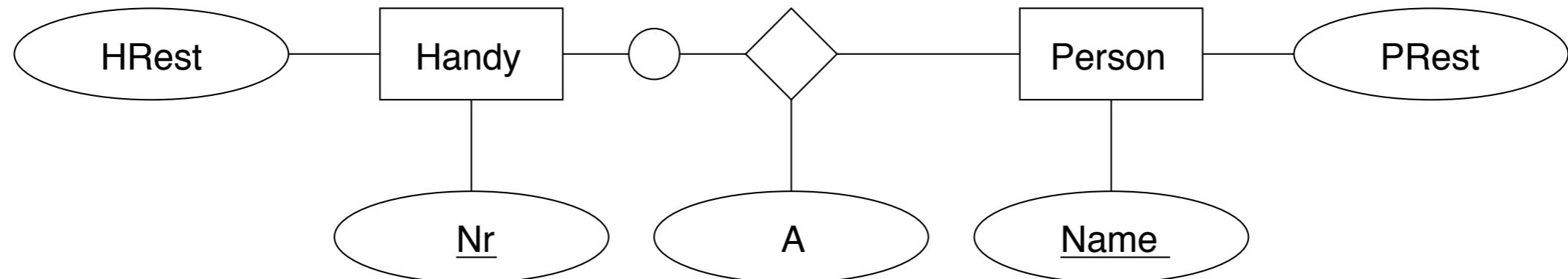
AND
    -- Relation konsistent
    -- "alles vorhanden" erlauben:
    (NOT (A IS NULL OR PName IS NULL OR HNr IS NULL) OR
     -- kein Relationsattribut bedingt mindestens
     -- einen nicht existierenden Relationspartner
     (A IS NULL AND (PName IS NULL OR HNr IS NULL)))

AND
    -- "nichts vorhanden" verbieten:
    (NOT (PName IS NULL AND HNr IS NULL AND A IS NULL))
)
```

);



Manche Personen haben kein Handy. Ein Handy hat genau einen Besitzer.

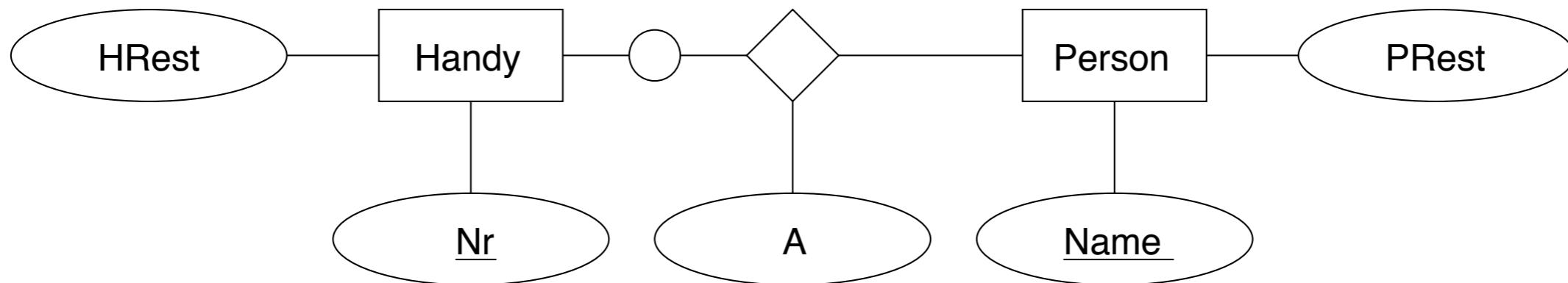


```
CREATE TABLE Person (
    Name ... PRIMARY KEY,
    PRest ... NOT NULL);
```

```
CREATE TABLE Handy (
    Nr ... PRIMARY KEY REFERENCES R(HNr),
    HRest ... NOT NULL);
```

```
CREATE TABLE R (
    HNr ... PRIMARY KEY REFERENCES Handy(Nr),
    PName ... NOT NULL UNIQUE REFERENCES Person(Name),
    A ... NOT NULL );
```

Darstellung mit nur einer Tabelle



```

CREATE TABLE PersonHandy (
    PName ... PRIMARY KEY,
    PRest ... NOT NULL,
    HNr ... UNIQUE, -- NULL erlaubt
    HRest ... , -- NULL erlaubt
    A ... -- NULL erlaubt
)
  
```

```

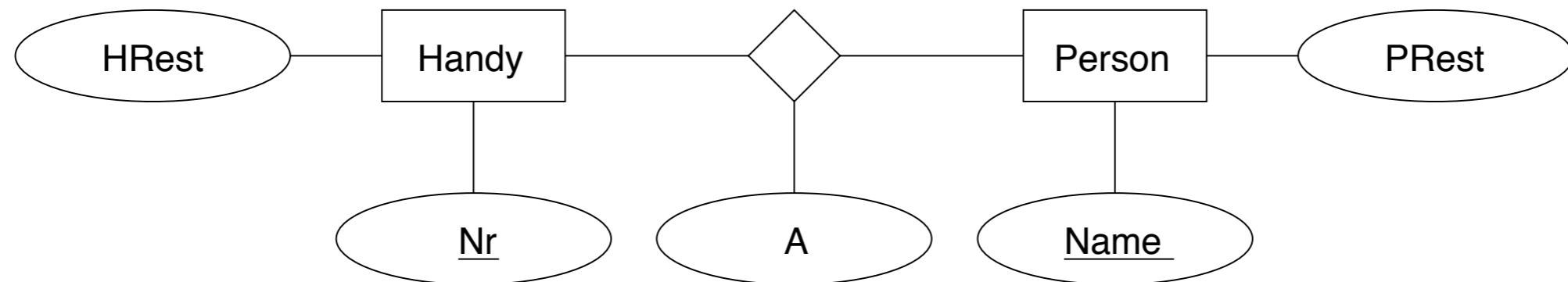
CHECK ( -- "kein Handy" erlauben:
    (HNr IS NULL AND HRest IS NULL AND A IS NULL) OR
    -- "alles vorhanden" erlauben:
    NOT (HNr IS NULL OR HRest IS NULL OR A IS NULL)
)
)
);
  
```

Anmerkung: Wenn ein Handy vorhanden ist, muss es auch an der Relation beteiligt sein. Das heißt, wenn **HNr** ungleich **NULL** ist, muss das auch für **A** gelten.

Relationships: 1:1

keine Optionalität

Jede Person hat genau ein Handy, jedes Handy genau einen Besitzer



```
CREATE TABLE Person (
  Name ... PRIMARY KEY REFERENCES R(PName),
  PRest ... NOT NULL);
```

```
CREATE TABLE Handy (
  Nr ... PRIMARY KEY REFERENCES R(HNr),
  HRest ... NOT NULL);
```

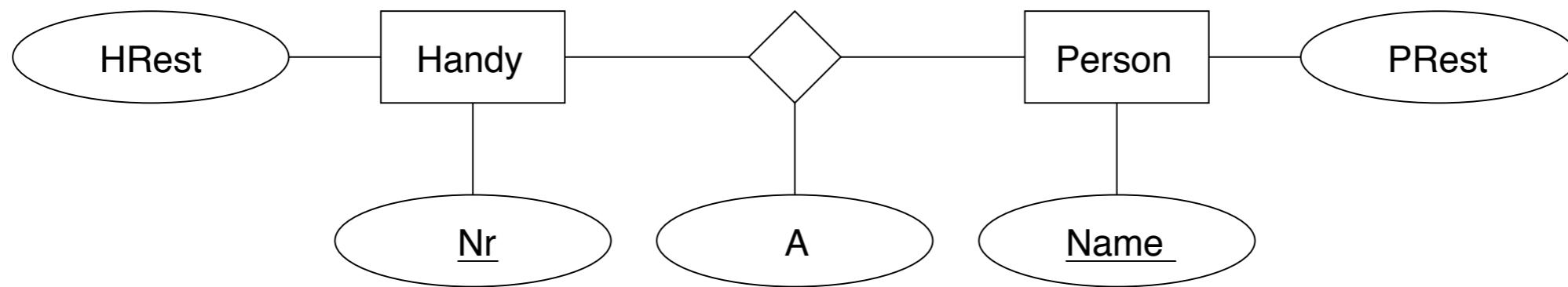
```
CREATE TABLE R (
  HNr ... PRIMARY KEY REFERENCES Handy(Nr),
  PName ... NOT NULL UNIQUE REFERENCES Person(Name),
  A ... NOT NULL );
```

Wie kann man alle Tabellen zu einer Tabelle zusammenfassen ?

Relationships: 1:1

keine Optionalität

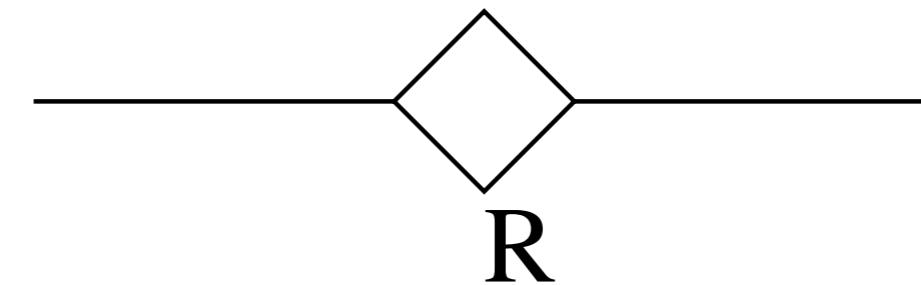
Darstellung mit nur einer Tabelle:

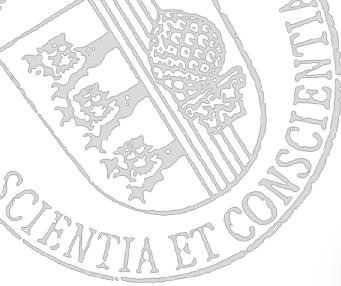


```
CREATE TABLE PersonHandy (
    PName ... PRIMARY KEY,
    PRest ... NOT NULL,
    HNr ... NOT NULL UNIQUE,
    HRest ... NOT NULL,
    A ... NOT NULL
);
```

Zusammenfassung

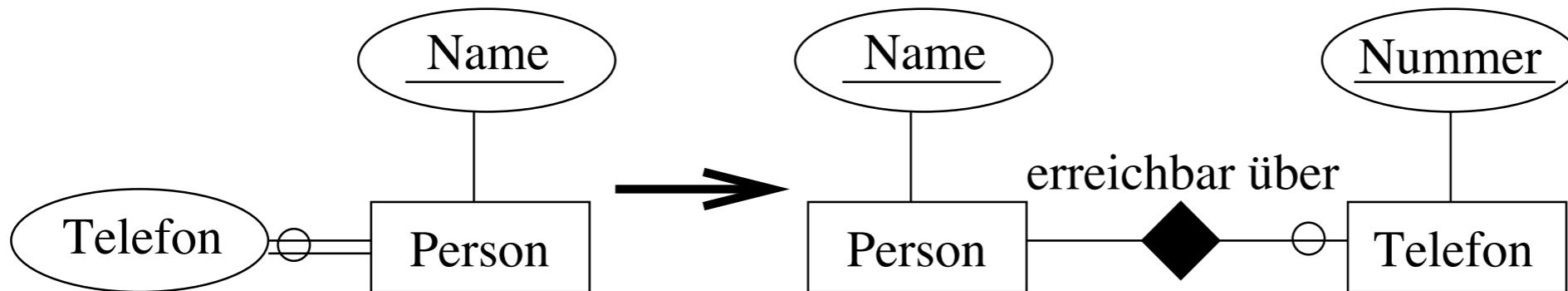
- Wie 1:n–Beziehungen lassen sich auch 1:1–Beziehungen mit oder ohne Relationstabelle darstellen.
- Bei einer Darstellung nur mit Entity-Tabelle können Relationsattribute und Fremdschlüssel wahlweise einer der beiden Entity-Tabellen zugeordnet werden.
- 1:1 Beziehungen lassen sich außerdem in einer einzigen Tabelle zusammenfassen.
- Das bietet sich besonders bei Relationen ohne Optionalitäten an, weil dann keine **NULL**-Einträge für Relationsattribute und Entitäten ohne Relationspartner vorkommen.





Mehrwertige Attribute, Schwache Entitäten und ISA-Beziehungen

- Mehrwertige Attribute sind nur Abkürzungen für eigene Entitäten

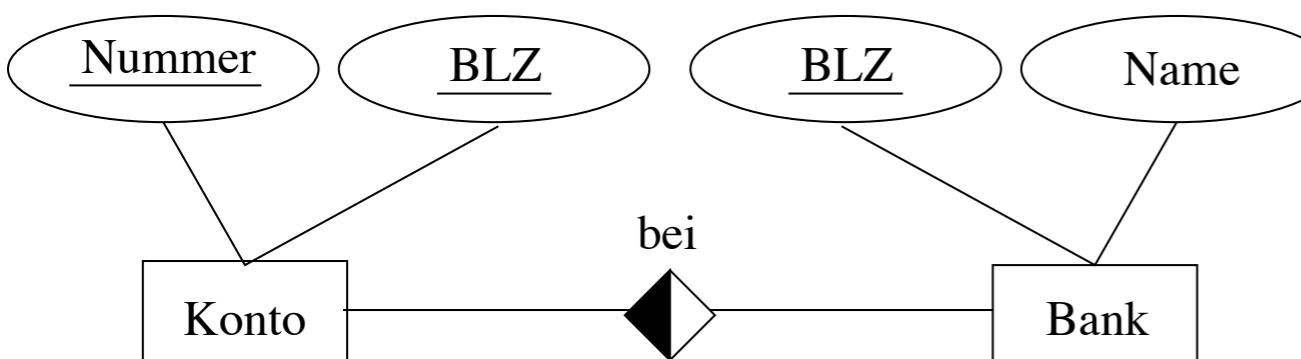
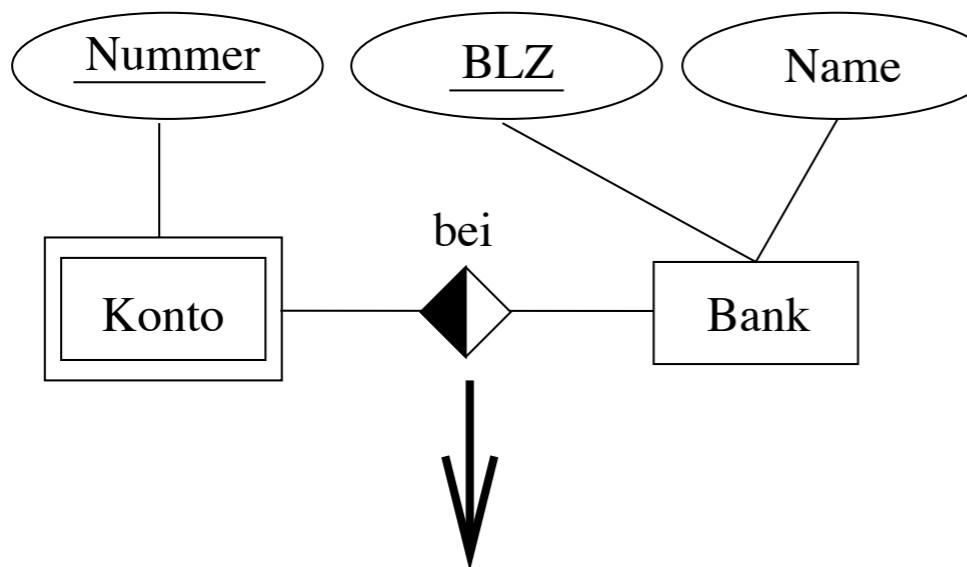


```
CREATE TABLE Person (
    Name CHAR(30) PRIMARY KEY
);

CREATE TABLE erreichbar_ueber_Telefon (
    Nummer CHAR(15) NOT NULL,
    Name CHAR(30) NOT NULL REFERENCES Person,
    PRIMARY KEY(Nummer, Name)
);
```

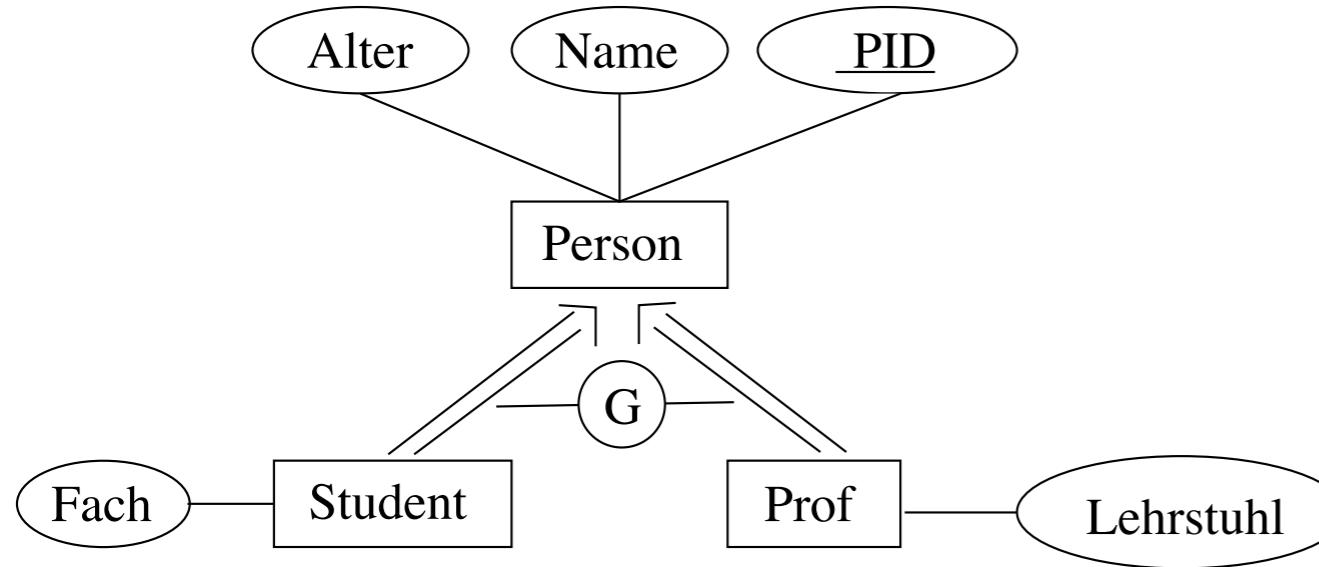
**Warum gibt es keine
Tabelle Telefon ?**

- **Schwache (abhängige) Entitäten** benötigen zur eindeutigen Identifikation ihrer Tupel die Schlüsselattribute einer anderen Entität.
- Schwache Entitäten können nicht ohne starke Entitäten existieren



```
CREATE TABLE Bank (
    BLZ INTEGER PRIMARY KEY,
    Name CHAR(100) NOT NULL,
    CHECK ((SELECT COUNT(*)
            FROM Konto
           WHERE Konto.BLZ = BLZ) > 0);
);
```

```
CREATE TABLE Konto (
    Nummer INTEGER NOT NULL,
    BLZ INTEGER NOT NULL REFERENCES Bank,
    PRIMARY KEY(Nummer, BLZ)
);
```



Schlüssel von Superentitäten werden als **Fremdschlüssel** übernommen.

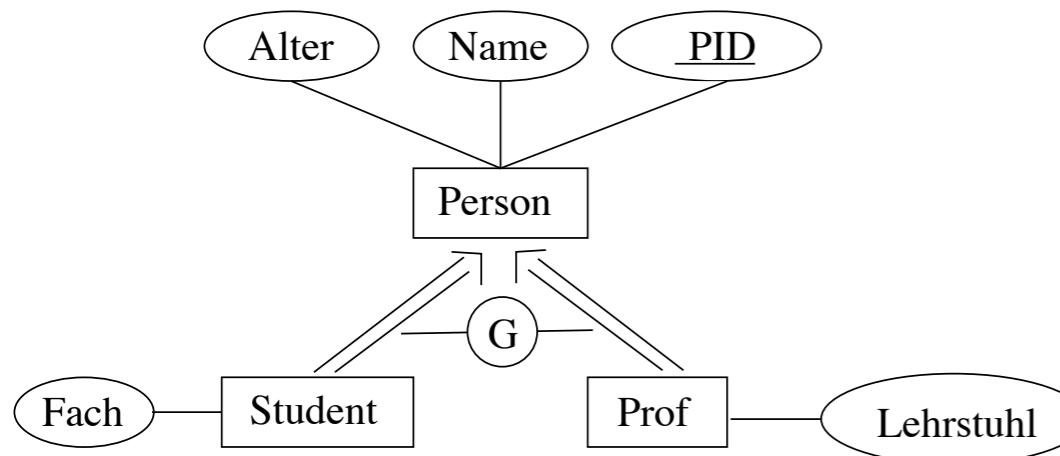
Dadurch wird sichergestellt, dass alle Studenten / Profs auch Personen sind.

```
CREATE TABLE Person (
    PID INTEGER PRIMARY KEY,
    Name CHAR(100) NOT NULL,
    Alter INTEGER NOT NULL
);

CREATE TABLE Student (
    PID INTEGER PRIMARY KEY REFERENCES Person,
    Fach CHAR(100) NOT NULL
);

CREATE TABLE Prof (
    PID INTEGER PRIMARY KEY REFERENCES Person,
    Lehrstuhl CHAR(100) NOT NULL
);
```

Frage:
**Wie kann die Disjunktheit
 (Generalisierung G)
 sichergestellt werden?**



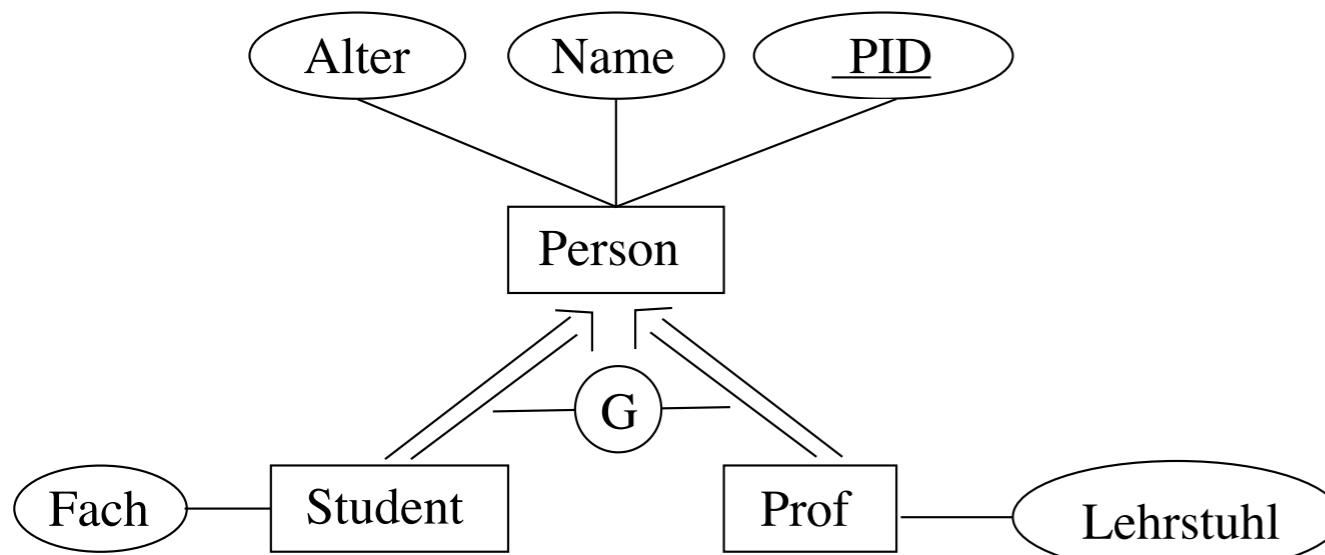
```

CREATE TABLE Person (
    PID INTEGER PRIMARY KEY,
    PTyp CHAR(4) NOT NULL
      CHECK (PTyp IN ('Stud', 'Prof')),
    Name CHAR(100) NOT NULL,
    Alter INTEGER NOT NULL,
    -- wegen Fremdschlüssel von Student/Prof
    UNIQUE(PID, PTyp) );

CREATE TABLE Student (
    PID INTEGER PRIMARY KEY,
    PTyp CHAR(4) NOT NULL CHECK (PTyp IN ('Stud')),
    Fach CHAR(100) NOT NULL,
    FOREIGN KEY(PID, PTyp) REFERENCES Person(PID, PTyp)
);

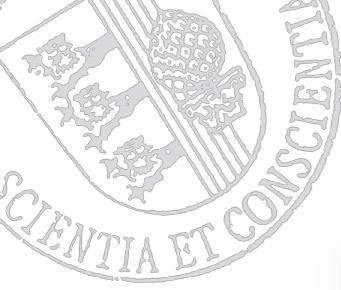
CREATE TABLE Prof (
    PID INTEGER PRIMARY KEY,
    PTyp CHAR(4) NOT NULL CHECK (PTyp IN ('Prof')),
    Lehrstuhl CHAR(100) NOT NULL,
    FOREIGN KEY(PID, PTyp) REFERENCES Person(PID, PTyp)
);
  
```

Ist noch die vollständige Überdeckung gewünscht (keine Person außer Studenten oder Profs), so muss Person angepasst werden.



```
CREATE TABLE Person (
    PID INTEGER PRIMARY KEY
    CHECK (PID IN
        ((SELECT PID FROM Student)
        UNION
        (SELECT PID FROM Prof))),
    PTyp CHAR(4) NOT NULL
    CHECK (PTyp IN ('Stud', 'Prof')),
    Name CHAR(100) NOT NULL,
    Alter INTEGER NOT NULL,
    -- wegen Fremdschlüssel von Student/Prof
    UNIQUE(PID, PTyp)
);
```

Anmerkung: Die Auswertung für das Einfügen von Daten muss verzögert werden (Stichwort: DEFERRED).



Mögliche Klausuraufgaben