

# Einführung in Datenbanken

---

## Kapitel 3: Das Relationale Modell

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2021/22

<http://www.informatik.uni-halle.de/~brass/db21/>

# Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Begriffe „Tupel“, „Attribut“, „Relation“ erklären.
- Die in dieser Vorlesung verwendete Kurz-Notation für relationale DB-Schemata  $R(\underline{A}, B)$  lesen und schreiben.
- Nullwerte erklären. Vor-/Nachteile nennen.
- Das Konzept eines Schlüssels erklären.  
Gegebene Tabellen auf Einhaltung von Schlüsseln prüfen.  
Sinnvolle Schlüssel für eine Anwendung wählen.
- Das Konzept eines Fremdschlüssels erklären.  
Die Fremdschlüssel-Bedingung für einen Zustand prüfen.
- Einfache **CREATE TABLE** Anweisungen schreiben.

# Inhalt

- 1 Beispiel-Datenbank
- 2 Relationale DB-Schemata
- 3 Nullwerte
- 4 Schlüssel
- 5 Fremdschlüssel
- 6 CHECK-Constraints

# Beispiel-Datenbank (1)

## STUDENTEN

<u>SID</u>	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

## AUFGABEN

<u>ATYP</u>	<u>ANR</u>	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

## BEWERTUNGEN

<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

# Beispiel-Datenbank (2)

- **STUDENTEN**: enthält eine Zeile für jeden Studenten.
  - **SID**: „Studenten-ID“ (eindeutige Nummer).
  - **VORNAME**, **NACHNAME**: Vor- und Nachname.
  - **EMAIL**: Email-Adresse (kann NULL sein).
- **AUFGABEN**: enthält eine Zeile für jede Aufgabe.
  - **ATYP**: Typ/Kategorie der Aufgabe.  
Z.B. 'H': Hausaufgabe, 'Z': Zwischenklausur, 'E': Endklausur.
  - **ANR**: Aufgabennummer (innerhalb des Typs).
  - **THEMA**: Thema der Aufgabe.
  - **MAXPT**: Maximale/volle Punktzahl der Aufgabe.

# Beispiel-Datenbank (3)

- **BEWERTUNGEN**: enthält eine Zeile für jede abgegebene Lösung zu einer Aufgabe.
  - **SID**: Student, der die Lösung abgegeben hat.

Dies referenziert eine Zeile in **STUDENTEN**.
  - **ATYP, ANR**: Identifikation der Aufgabe.

Zusammen identifiziert dies eine Zeile in **AUFGABEN**.
  - **PUNKTE**: Punkte, die der Student für die Lösung bekommen hat.
  - Falls es keinen Eintrag für einen Studenten und eine Aufgabe gibt: Aufgabe nicht abgegeben.

# Inhalt

- 1 Beispiel-Datenbank
- 2 Relationale DB-Schemata**
- 3 Nullwerte
- 4 Schlüssel
- 5 Fremdschlüssel
- 6 CHECK-Constraints

# Relationale DB-Schemata

- Ein Relationenschema (Schema einer Relation) definiert
  - eine Folge  $A_1, \dots, A_n$  von Spalten-/Attributnamen,  
Diese müssen untereinander verschieden sein, also  $A_i \neq A_j$  für  $i \neq j$ .
  - für jede Spalte  $A_i$  einen Datentyp  $D_i$ .  
Das DBMS legt eine feste Auswahl von möglichen Datentypen fest.

Ein Relationenschema kann geschrieben werden als

$$(A_1: D_1, \dots, A_n: D_n).$$

- Ein relationales Datenbank-Schema definiert:
  - eine endliche Menge  $R_1, \dots, R_m$  von Relationennamen,  
„Relation“ und „Tabelle“ werden synonym gebraucht.
  - für jede Relation  $R_i$  ein Relationenschema  $sch(R_i)$ ,
  - eine Menge  $\mathcal{C}$  von Integritätsbedingungen (s.u.).



# Schemata: Notation (1)

- Betrachten Sie die Beispiel-Tabelle:

AUFGABEN			
ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

- Eine Art, ein Schema präzise zu definieren, ist über eine SQL-Anweisung:

```
CREATE TABLE AUFGABEN(ATYP CHAR(1),  
                       ANR  NUMERIC(2),  
                       THEMA VARCHAR(40),  
                       MAXPT NUMERIC(2))
```

Hier ist AUFGABEN der Relationenname, ATYP ein Attributname, und CHAR(1) der zugehörige Datentyp.

# Schemata: Notation (2)

- Obwohl letztendlich ein **CREATE TABLE**-Statement für das DBMS benötigt wird, gibt es andere Notationen, um das Schema zu dokumentieren.
- Bei der Diskussion der DB-Struktur sind die Datentypen der Spalten oft nicht wichtig.
- Eine kurze Notation ist der Tabellename, gefolgt von der Liste der Spaltennamen in Klammern:

**AUFGABEN(ATYP, ANR, THEMA, MAXPT)**

- Wenn nötig, werden die Datentypen hinzugefügt:

**AUFGABEN(ATYP: CHAR(1), ...)**

# Tupel (1)

- Ein  $n$ -Tupel ist eine Folge von  $n$  Werten.

Man kann auch nur „Tupel“ statt  $n$ -Tupel sagen, wenn das  $n$  nicht wichtig ist oder vom Kontext her klar ist. Tupel werden verwendet, um Tabellenzeilen zu formalisieren, dann ist  $n$  die Anzahl der Spalten.

- Z.B. sind XY-Koordinaten Paare  $(X, Y)$  von reellen Zahlen. Paare sind Tupel der Länge 2 („2-Tupel“).

3-Tupel werden auch Tripel genannt und 4-Tupel Quadrupel.

- Das kartesische Produkt  $\times$  erstellt Mengen von Tupeln, z.B.:

$$\mathbb{R} \times \mathbb{R} := \{(X, Y) \mid X \in \mathbb{R}, Y \in \mathbb{R}\}.$$

# Tupel (2)

- Sei  $\mathcal{I}_D[D_i]$  der Wertebereich des Datentyps  $D_i$ .

$\mathcal{I}_D$  ist eine Interpretation der mehrsortigen Prädikatenlogik, siehe Kapitel 7.

Z.B.  $\mathcal{I}_D[\text{NUMERIC}(1)] = \{-9, -8, \dots, 0, 1, \dots, 9\}$ .

- Ein Tupel  $t$  für das Relationen-Schema

$$(A_1 : D_1, \dots, A_n : D_n)$$

ist eine Folge  $(d_1, \dots, d_n)$  von  $n$  Werten, so dass  $d_i \in \mathcal{I}_D[D_i]$ .

D.h.  $t \in \mathcal{I}_D[D_1] \times \dots \times \mathcal{I}_D[D_n]$ .

- Gegeben sei ein solches Tupel. Wir schreiben  $t.A_i$  für den Wert  $d_i$  in der Spalte  $A_i$ .

Alternative Notation:  $t[A_i]$ .

- Z.B. ist eine Zeile in der Beispieltabelle „AUFGABEN“ das Tupel  $(\text{'H'}, 1, \text{'ER'}, 10)$ .

# DB-Zustände (1)

Sei ein DB-Schema  $(\{R_1, \dots, R_m\}, sch, \mathcal{C})$  gegeben.

- Ein DB-Zustand  $\mathcal{I}$  für dieses Schema definiert für jede Relation  $R_i$  eine endliche Menge von Tupeln für das Relationen-Schema  $sch(R_i)$ .
- D.h. wenn  $sch(R_i) = (A_{i,1} : D_{i,1}, \dots, A_{i,n_i} : D_{i,n_i})$ , dann

$$\mathcal{I}[R_i] \subseteq \mathcal{I}_{\mathcal{D}}[D_{i,1}] \times \dots \times \mathcal{I}_{\mathcal{D}}[D_{i,n_i}].$$

- D.h. ein DB-Zustand interpretiert die Symbole im Schema, er bildet Relationen-Namen auf Relationen ab.

Es ist auch eine Interpretation der Prädikatenlogik (Erweiterung von  $\mathcal{I}_{\mathcal{D}}$ ).

- Außerdem muss ein DB-Zustand alle Integritätsbedingungen in  $\mathcal{C}$  erfüllen (Formeln der Prädikatenlogik, s.u.).

# DB-Zustände (2)

Relationen sind **Mengen** von Tupeln. Daher

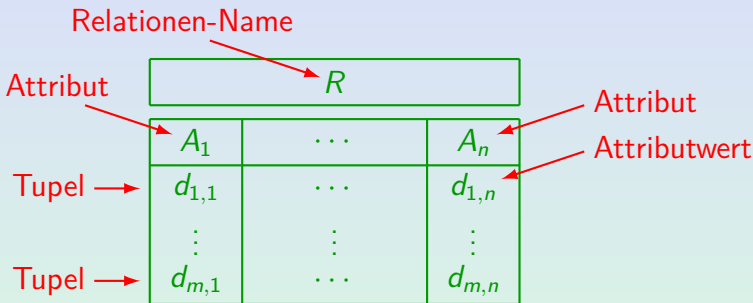
- ist die Reihenfolge der Tupel nicht definiert.
  - Die Darstellung in einer Tabelle ist etwas irreführend.  
Es gibt keine erste, zweite, usw. Zeile.

Die Speicherverwaltung legt fest, wo eine neue Zeile eingefügt wird (dabei wird z.B. der Platz von gelöschten Zeilen wiederverwendet).

- Relationen können bei der Ausgabe sortiert werden.
- gibt es keine Tupel-Duplikate.
  - Viele derzeitige Systeme erlauben doppelte Tupel, solange kein Schlüssel definiert ist (später).

Also wäre eine Formalisierung als Multimengen korrekt.

# Zusammenfassung (1)



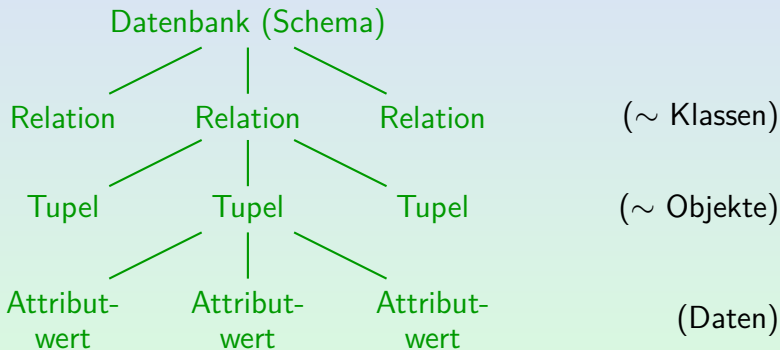
Synonyme: Relation und Tabelle.

Tupel, Zeile und Record.

Attribut, Spalte, Feld.

Attributwert, Spaltenwert, Tabelleneintrag.

# Zusammenfassung (2)





# Inhalt

- 1 Beispiel-Datenbank
- 2 Relationale DB-Schemata
- 3 Nullwerte**
- 4 Schlüssel
- 5 Fremdschlüssel
- 6 CHECK-Constraints

# Nullwerte

- Das relationale Modell erlaubt fehlende Attributwerte, d.h. **Tabelleneinträge können leer sein.**
- Formal wird die Menge der möglichen Attributwerte durch einen neuen Wert „Null“ erweitert.

- Wenn  $R$  das Schema  $(A_1: D_1, \dots, A_n: D_n)$  hat, dann

$$\mathcal{I}[R] \subseteq \left( \mathcal{I}_D[D_1] \cup \{null\} \right) \times \dots \times \left( \mathcal{I}_D[D_n] \cup \{null\} \right).$$

- **„Null“ ist nicht die Zahl 0 oder der leere String!**  
Es ist von allen Werten des Datentyps verschieden.

In Oracle ist es der leere String, aber das widerspricht dem SQL-Standard. Normalerweise sollte man für String-wertige Spalten entweder den leeren String oder den Nullwert ausschließen. Bei anderen Datentypen ist der Nullwert von den normalen Werten deutlich zu unterscheiden.

# Nullwerte ausschließen (1)

- Da Nullwerte zu Komplikationen führen, kann für jedes Attribut festgelegt werden, ob Nullwerte erlaubt sind.

Nullwerte werden in SQL mit einer dreiwertigen Logik behandelt. Wir sind aber mit einer zweiwertigen Logik vertrauter. Wenn man den Wert einer Datenbank-Spalte z.B. in eine `int`-Variable in einem Java-Programm laden will, ist ein Problem, dass es dort keinen Nullwert gibt.

- Man sollte bewusst darüber nachdenken, wo Nullwerte gebraucht werden, und wo nicht:
  - Viele Spalten als „not null“ zu deklarieren, vereinfacht Programme und verringert Überraschungen.
  - Die Flexibilität geht jedoch verloren: Nutzer werden gezwungen, für alle „not null“-Attribute Werte einzutragen.

Notfalls auch sinnlose Werte.

## Nullwerte ausschließen (2)

- In SQL schreibt man **NOT NULL** hinter den Datentyp für ein Attribut, das nicht Null sein kann.

Dies ist genau genommen eine Integritätsbedingung, aber man kann es auch als Teil des Datentyps ansehen. Die genaue Syntax der „CREATE TABLE“ Anweisung wird ausführlich erst in der Vorlesung „Datenbank-Programmierung“ behandelt.

- Z.B. kann **EMAIL** in **STUDENTEN** Null sein:

```
CREATE TABLE STUDENTEN(  
    SID          NUMERIC(3)  NOT NULL,  
    VORNAME     VARCHAR(20) NOT NULL,  
    NACHNAME    VARCHAR(20) NOT NULL,  
    EMAIL       VARCHAR(80) )
```

# Nullwerte ausschließen (3)

- In SQL sind Nullwerte als Default erlaubt und man muss explizit „**NOT NULL**“ verlangen.
- Oft können nur wenige Spalten Nullwerte haben.
- Daher ist es besser, in der vereinfachten Notation umgekehrt optionale Attribute zu markieren:

`STUDENTEN(SID, VORNAME, NACHNAME, EMAILo)`

- In dieser Notation werden Attribute, die Nullwerte enthalten können, mit einem kleinen „o“ (optional) im Exponenten markiert.

Dies ist nicht Teil des Spaltennamens.

- Alternative (wenn nur ASCII Zeichen möglich): „**EMAIL?**“.

# Inhalt

- 1 Beispiel-Datenbank
- 2 Relationale DB-Schemata
- 3 Nullwerte
- 4 Schlüssel**
- 5 Fremdschlüssel
- 6 CHECK-Constraints

# Eindeutige Identifikation (1)

- Ein Schlüssel einer Relation  $R$  ist eine Spalte  $A$ , die die Tupel/Zeilen in  $R$  eindeutig identifiziert.
- Wenn z.B.  $SID$  als Schlüssel von  $STUDENTEN$  deklariert wurde, ist dieser DB-Zustand verboten:

STUDENTEN			
<u>SID</u>	VORNAME	NACHNAME	EMAIL
⇒ 101	Lisa	Weiss	...
⇒ 101	Michael	Grau	...
103	Daniel	Sommer	NULL
104	Iris	Winter	...

- Schlüssel sind eine Art von Integritätsbedingungen.  
Bedingungen, die jeder Datenbank-Zustand erfüllen muss.

# Eindeutige Identifikation (2)

- Wurde **SID** als Schlüssel von **STUDENTEN** deklariert, akzeptiert das DBMS keine Einfügung einer Zeile mit dem gleichen Wert für **SID** wie eine existierende Zeile.

Entsprechend kann man auch durch Änderung eines Tabelleneintrags keinen DB-Zustand erreichen, der den Schlüssel (oder eine andere Integritätsbedingung) verletzen würde.

- Schlüssel werden als Teil des Datenbank-Schemas deklariert, und müssen dann für alle Datenbank-Zustände gelten.
- Obwohl im obigen DB-Zustand (mit nur 4 Studenten) der Nachname (**NACHNAME**) eindeutig ist, würde dies allgemein zu einschränkend sein.

Z.B. wäre das zukünftige Einfügen von „Nina Weiss“ unmöglich. Bei der Entscheidung über Schlüssel muss man an alle möglichen Zustände denken.



# Eindeutige Identifikation (3)

- Ein Schlüssel kann auch aus mehreren Attributen bestehen („**zusammengesetzter Schlüssel**“).

Wenn  $A$  und  $B$  zusammen einen Schlüssel bilden, ist es verboten, dass es zwei Zeilen  $t$  und  $u$  gibt, die in beiden Attributen übereinstimmen (d.h.  $t.A = u.A$  und  $t.B = u.B$ ). Zwei Zeilen können in einem Attribut übereinstimmen, aber nicht in beiden.

- Der Schlüssel „**VORNAME, NACHNAME**“ ist hier erfüllt:

STUDENTEN			
<u>SID</u>	<u>VORNAME</u>	<u>NACHNAME</u>	EMAIL
101	Lisa	Weiss	...
102	Michael	Weiss	...
103	Michael	Grau	...

# Mehrere Schlüssel

- Eine Relation kann mehr als einen Schlüssel haben.
- Z.B. ist **SID** ein Schlüssel von **STUDENTEN** und „**VORNAME, NACHNAME**“ ggf. ein weiterer Schlüssel.
- Ein Schlüssel wird zum „**Primärschlüssel**“ ernannt.

Der Primärschlüssel sollte möglichst aus einem einzigen kurzen Attribut bestehen, das möglichst nie verändert wird (durch Updates).

Der Primärschlüssel wird in anderen Tabellen verwendet, die sich auf Zeilen dieser Tabelle beziehen. In manchen Systemen ist Zugriff über Primärschlüssel besonders schnell. Ansonsten ist die Wahl des Primärschlüssels egal.

- Die anderen sind „**Alternativ-/Sekundär-Schlüssel**“.

SQL verwendet das Schlüsselwort **UNIQUE** für alternative Schlüssel.

# Schlüssel: Minimalität

- Jeder DB-Zustand  $\mathcal{I}$ ,
  - der den Schlüssel „NACHNAME“ erfüllt,
  - erfüllt auch den Schlüssel „VORNAME, NACHNAME“.
- Beispiel: Ist  $SID$  allein schon eindeutig, so ist erst recht die Kombination mit  $NACHNAME$  zusammen eindeutig.

Gibt es keine zwei Zeilen, die in der  $SID$  übereinstimmen, so gibt es auch nicht zwei Zeilen, die in  $SID$  und  $NACHNAME$  übereinstimmen. Allgemein macht das Hinzufügen von Attributen Schlüssel weniger einschränkend (d.h. führt zu einer Obermenge von Zuständen). Wenn man die logischen Formeln betrachtet, die den Schlüsseln entsprechen, ist die Formel für den Schlüssel  $\{SID, NACHNAME\}$  eine logische Folgerung aus der Formel für  $\{SID\}$ .
- Man wird daher nie zwei Schlüssel deklarieren, so dass einer eine Obermenge von Attributen des anderen ist.

# Schlüssel: Notation (1)

- Die Primärschlüssel-Attribute werden oft markiert, indem man sie unterstreicht:

$$R(\underline{A_1 : D_1}, \dots, \underline{A_k : D_k}, A_{k+1} : D_{k+1}, \dots, A_n : D_n).$$

Meist ordnet man die Attribute so, dass der Primärschlüssel am Anfang steht.

STUDENTEN			
<u>SID</u>	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

- Weitere Schlüssel muss man als Text dazu schreiben.
- Wenn man ASCII schreiben muss: # vor dem Attribut statt unterstreichen: `AUFGABEN(#ATYP, #ANR, THEMA, MAXPT)`

# Schlüssel: Notation (2)

- In SQL können Schlüssel folgendermaßen definiert werden:

```
CREATE TABLE STUDENTEN(  
    SID          NUMERIC(3)  NOT NULL,  
    VORNAME     VARCHAR(20) NOT NULL,  
    NACHNAME    VARCHAR(20) NOT NULL,  
    EMAIL       VARCHAR(80),  
    PRIMARY KEY(SID),  
    UNIQUE(VORNAME, NACHNAME))
```

Die genaue Syntax der CREATE TABLE Anweisung und mehr Details zur Integritätsüberwachung sind ein Kapitel der Fortsetzungs-Vorlesung „Datenbank-Programmierung“.

# Inhalt

- 1 Beispiel-Datenbank
- 2 Relationale DB-Schemata
- 3 Nullwerte
- 4 Schlüssel
- 5 Fremdschlüssel**
- 6 CHECK-Constraints

# Fremdschlüssel (1)

- Das relationale Modell hat keine expliziten Verweise (Zeiger) oder Beziehungen zwischen Tupeln.
- Schlüsselattributwerte identifizieren ein Tupel.

Sie sind „logische Adressen“ der Tupel.

- Um sich in einer Relation  $S$  auf Tupel von  $R$  zu beziehen, fügt man den Primärschlüssel von  $R$  zu den Attributen von  $S$  hinzu.

Solche Attributwerte sind „logische Zeiger“ auf Tupel in  $R$ .

- Z.B. hat die Tabelle **BEWERTUNGEN** das Attribut **SID**, welches Primärschlüsselwerte von **STUDENTEN** enthält.

# Fremdschlüssel (2)

**SID** in **BEWERTUNGEN** ist ein Fremdschlüssel, der **STUDENTEN** referenziert:

STUDENTEN			
<u>SID</u>	VORNAME	NACHNAME	...
101	Lisa	Weiss	...
102	Michael	Grau	...
103	Daniel	Sommer	...
104	Iris	Winter	...

BEWERTUNGEN			
<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	PUNKTE
101	H	1	10
101	H	2	8
102	H	1	9
102	H	2	9
103	H	1	5
105	H	1	7

**?**  
**Fehler**

Die hier benötigte Bedingung ist, dass jeder **SID**-Wert in **BEWERTUNGEN** auch in **STUDENTEN** auftaucht.



# Fremdschlüssel (3)

- Die Fremdschlüsselbedingung

„**BEWERTUNGEN.SID**  $\rightarrow$  **STUDENTEN**“

fordert, dass die Menge der Werte in der Spalte **SID** der Tabelle **BEWERTUNGEN** immer eine Teilmenge der Primärschlüsselwerte in **STUDENTEN** ist.

Somit ist die Menge der SID-Werte in **STUDENTEN** eine Art „dynamischer Wertebereich“ für SID in **BEWERTUNGEN**.

- In Logik: Für alle Tupel  $t \in \mathcal{I}[\text{BEWERTUNGEN}]$  gibt es ein Tupel  $u \in \mathcal{I}[\text{STUDENTEN}]$  mit  $t.\text{SID} = u.\text{SID}$ .

Würde die Fremdschlüssel-Spalte SID in **BEWERTUNGEN** Nullwerte erlauben, so wären die Tupel mit einem Nullwert von dieser Bedingung ausgenommen.

- Es ist nicht verlangt, dass die Spalten in beiden Tabellen gleich heißen.

# Fremdschlüssel (4)

- Die Tabelle **BEWERTUNGEN** enthält noch einen Fremdschlüssel, der die gelöste Aufgabe referenziert.
- Aufgaben werden durch eine Kategorie und eine Nummer (**ATYP** und **ANR**) identifiziert:

BEWERTUNGEN			
SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
⋮	⋮	⋮	⋮

AUFGABEN			
ATYP	ANR	...	MAXPT
H	1	...	10
H	2	...	10
Z	1	...	14

# Fremdschlüssel (5)

- Eine Tabelle mit zusammengesetztem Schlüssel (wie **AUFGABEN**) muss mit einem Fremdschlüssel referenziert werden, der die gleiche Spaltenanzahl hat.
- Zusammengehörige Spalten müssen den gleichen Datentyp haben. Es ist nicht nötig, dass sie den gleichen Namen haben.

Die erste Spalte des Fremdschlüssels wird mit der ersten Spalte des Schlüssels verglichen, u.s.w. (Identifikation über die Position).

- Im Beispiel erfordert der Fremdschlüssel, dass jede Kombination von **ATYP** und **ANR**, die in **BEWERTUNGEN** vorkommt, auch in **AUFGABEN** existiert.
- Man kann nur Schlüssel referenzieren, nicht beliebige Spalten.

# Fremdschlüssel: Notation (1)

- In der Attributlisten-Notation können Fremdschlüssel durch einen Pfeil und den Namen der referenzierten Tabelle markiert werden. Bei zusammengesetzten Fremdschlüsseln braucht man Klammern:

```
BEWERTUNGEN(SID → STUDENTEN,  
              (ATYP, ANR) → AUFGABEN, PUNKTE)  
STUDENTEN(SID, VORNAME, NACHNAME, EMAIL)  
AUFGABEN(ATYP, ANR, THEMA, MAXPT)
```

In ASCII schreibt man den Pfeil natürlich ->.

- Da normalerweise nur Primärschlüssel referenziert werden, ist es nicht nötig, die zugehörigen Attribute der referenzierten Tabelle anzugeben.

# Fremdschlüssel: Notation (2)

- In SQL können Fremdschlüssel wie folgt deklariert werden:

```
CREATE TABLE BEWERTUNGEN(  
    SID      NUMERIC(3)    NOT NULL,  
    ATYP     CHAR(1)      NOT NULL,  
    ANR      NUMERIC(2)    NOT NULL,  
    PUNKTE   NUMERIC(4,1) NOT NULL,  
    PRIMARY KEY(SID, ATYP, ANR),  
    FOREIGN KEY(SID)  
        REFERENCES STUDENTEN,  
    FOREIGN KEY(ATYP, ANR)  
        REFERENCES AUFGABEN)
```

Es ist möglich, die Namen der referenzierten Spalten anzugeben, z.B. „FOREIGN KEY(SID) REFERENCES STUDENTEN(SID)“. Dies ist nur nötig, wenn man einen Alternativschlüssel referenziert (bei MySQL aber immer).

# Noch ein Beispiel:

- Im obigen Beispiel sind die Fremdschlüsselattribute auch Teil des Schlüssels. Das muss nicht so sein.
- Beispiel (Beziehung Vorlesung-Dozent):

DOZENT(KUERZEL, VORNAME, NACHNAME, EMAIL)

VORLESUNG(VNR, TITEL, LP, GEHALTEN\_VON → DOZENT)

DOZENT			
<u>KUERZEL</u>	VORNAME	NACHNAME	EMAIL
sb	Stefan	Brass	brass@...

VORLESUNG			
<u>VNR</u>	TITEL	LP	GEHALTEN_VON
1	Einführung in DBen	5	sb
2	DB-Entwurf	5	sb

# Bitte merken:

- Fremdschlüssel sind selbst keine Schlüssel!

Die Attribute eines Fremdschlüssels können Teil eines Schlüssels sein, aber das ist eher die Ausnahme. Die Fremdschlüssel-Bedingung hat nichts mit einer Schlüsselbedingung zu tun. Für manche Autoren ist jedoch jedes Attribut, das Tupel identifiziert (nicht unbedingt in der gleichen Tabelle), ein Schlüssel. Dann wären Fremdschlüssel Schlüssel, aber normale Schlüssel brauchen dann immer einen Zusatz („Primär-/Alternativ-“).

- Nur Schlüssel einer Relation können referenziert werden, keine beliebigen Attribute.
- Besteht der Schlüssel der referenzierten Relation aus zwei Attributen, muss der Fremdschlüssel auch aus zwei Attributen bestehen.

# Inhalt

- 1 Beispiel-Datenbank
- 2 Relationale DB-Schemata
- 3 Nullwerte
- 4 Schlüssel
- 5 Fremdschlüssel
- 6 CHECK-Constraints**



# CHECK-Constraints (1)

- Man kann Bedingungen angeben, die alle Daten erfüllen müssen:

```
CREATE TABLE STUDENTEN(  
    SID          NUMERIC(3)  NOT NULL,  
    VORNAME     VARCHAR(20) NOT NULL,  
    NACHNAME    VARCHAR(20) NOT NULL,  
    EMAIL       VARCHAR(80),  
    PRIMARY KEY(SID),  
    UNIQUE(VORNAME, NACHNAME),  
    CHECK(SID > 0))
```

- So ist sichergestellt, dass alle SIDs positiv sind.

# CHECK-Constraints (2)

- Auch möglich wäre:

```
CHECK(SID BETWEEN 101 AND 999).
```

Es ist günstig, disjunkte Intervalle für unterschiedliche „Typen“ zu verwenden. Wenn man z.B. eine Methode schreibt, die Parameter ANR und SID hat, und man vertauscht beim Aufruf die beiden Werte, wird man beim Testen zumindest bemerken, dass das Ergebnis leer ist.

- Die möglichen Bedingungen sind die gleichen wie in SQL-Anfragen unter WHERE, allerdings ohne Unteranfragen.

Die Bedingung kann mehrere Spalten der Tabelle verwenden, und kann mit AND und OR zusammengesetzt sein, aber sie darf sich nur auf die eine gerade eingefügte oder modifizierte Zeile beziehen. So ist eine effiziente Integritätsprüfung möglich: Alle vorher schon eingefügten Zeilen wurden ja früher schon geprüft. Man kann dann wie bei der vollständigen Induktion schließen, dass die Bedingung für alle Zeilen der Tabelle gilt.