

# Datenbankbasierte Web-Anwendungen

## Datenbank Schema: Eigene Datentypen und Umsetzung von ERM Beziehungen

Medieninformatik SoSe 2017

Renzo Kottmann



This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).

# Zusammenfassung der vorherigen Vorlesung

## Constraints

- Datentypen
- Primary Keys
- NULL or NOT NULL Constraints
- DEFAULT VALUES
- CHECK Constraints

Nicht besprochen:

- Unique Constraint

# Unique Constraint

- Alle Werte einer- oder mehrerer Spalte(n) müssen eindeutig sein
- Damit werden weitere Schlüssel implementiert

```
CREATE TABLE teilnehmer (  
  vorname text  
    CHECK ( vorname != ' ' ),  
  nachname text,  
  matrikel_nr integer  
    PRIMARY KEY,  
  email text  
    NOT NULL  
    CHECK ( email ~ '.*@.*' )  
    UNIQUE,  
  semester integer DEFAULT 4  
);  
Jeder Teilnehmer muss eine  
andere E-Mail haben.
```

# Logisch gesehen: Primary Key vs. Unique

- Ein PRIMARY KEY ist nichts anderes als ein UNIQUE NOT NULL
- D.h. es kann mehrere Schlüssel geben, aber nur einer wird als PRIMARY KEY gewählt

```
CREATE TABLE teilnehmer (  
  vorname text  
    CHECK ( vorname != ' ' ),  
  nachname text,  
  matrikel_nr integer PRIMARY KEY,  
  email text  
    NOT NULL  
    CHECK ( email ~ '.*@.*' )  
    UNIQUE,  
  semester integer DEFAULT 4  
);  
Jeder Teilnehmer muss eine  
andere E-Mail haben.
```

# SQL Kommando: CREATE DOMAIN

- Definition von eigenen Datentypen basierend of existierenden

```
CREATE DOMAIN person_name AS text CHECK (  
    VALUE ~ '^[A-Z][a-z]*'  
);
```

```
CREATE TABLE teilnehmer (  
    vorname person_name  
        CHECK ( vorname != '' ),  
    nachname person_name,  
    matrikel_nr integer PRIMARY KEY,  
    email text  
        NOT NULL  
        CHECK ( email ~ '.*@.*' )  
        UNIQUE,  
    semester integer DEFAULT 4  
);  
Jeder Name faengt mit einem  
Gross-Buchstaben an.
```

# Vertiefung Einschränkung von Wertebereichen

## Arten von Daten

1. Zahlen (integer, numeric, double precision...)
2. Free Text (text, char)
3. Enumeration
4. Code-List
5. Komplexe Zusammensetzungen der oberen "einfachen Arten"  
Z.B. Telefonnummern, Zeitstempel (Datum + Uhrzeit)...

# Enumeration & Code List

1. **Enumeration:** Liste von Werten, die in Zukunft wenig bis gar nicht geaendert wird  
z.B. Geschlecht = {maennlich, weiblich}
2. **Code-List:** Liste von Werten, die in Zukunft haeufig und zu jeder Zeit geaendert wird

# Implementierungsvarianten

1. check constraint:

```
gender text check (gender in ['maennlich','weiblich'])
```

2. Enumerated Types:

```
CREATE TYPE gender AS ENUM ('maennlich','weiblich');
```



# Implementierung mit Hilfe einer Tabelle

```
CREATE TABLE geschlecht (  
    name text PRIMARY KEY  
);  
INSERT INTO geschlecht  
VALUES ('maennlich'),('weiblich');  
  
CREATE TABLE teilnehmer (  
    vorname text  
        CHECK ( vorname != '' ),  
    nachname text,  
    matrikel_nr integer  
        PRIMARY KEY,  
    email text  
        NOT NULL  
        CHECK ( email ~ '.*@.*' )  
        UNIQUE,  
    semester integer DEFAULT 4,  
    geschlecht text  
        REFERENCES geschlecht(name)  
);  
  
INSERT INTO teilnehmer (geschlecht,vorname, nachname, matrikel_nr, email)  
VALUES ('maennlich','renzo','kottmann',007,'renzo@007.bond');
```

# Implementierung durch Foreign Key

A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table.

We say this maintains the referential integrity between two related tables.

```
CREATE TABLE geschlecht (  
    name text PRIMARY KEY  
);  
  
CREATE TABLE teilnehmer (  
    vorname text  
        CHECK ( vorname != ' ' ),  
    nachname text,  
    matrikel_nr integer  
        PRIMARY KEY,  
    email text  
        NOT NULL  
        CHECK ( email ~ '.*@.*' )  
        UNIQUE,  
    semester integer DEFAULT 4,  
    geschlecht text  
        REFERENCES geschlecht(name)  
);
```

# Implementierung durch Foreign Key

- Kann man erst Tabelle teilnehmer und dann geschlecht anlegen?
- Was passiert, wenn in teilnehmer geschlecht null ist?
- Was passiert, wenn in geschlecht die Zeile maennlich geloescht wird?

```
CREATE TABLE geschlecht (  
  name text PRIMARY KEY  
);  
  
CREATE TABLE teilnehmer (  
  vorname text  
    CHECK ( vorname != ' ' ),  
  nachname text,  
  matrikel_nr integer  
    PRIMARY KEY,  
  email text  
    NOT NULL  
    CHECK ( email ~ '.*@.*' )  
    UNIQUE,  
  semester integer DEFAULT 4,  
  geschlecht text  
    REFERENCES geschlecht(name)  
);
```

# Foreign Key Verhalten bei Datenaenderung

- Was soll mit Eintraegen in teilnehmer passieren wenn ein FOREIGN KEY (also ein Eintrag in geschlecht geaendert wird?)
  - Verhindern oder alle Eintraege in teilnehmer auch aendern?

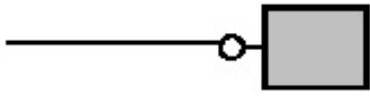
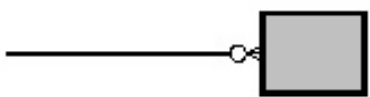
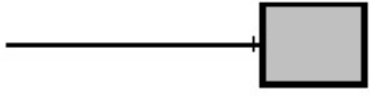
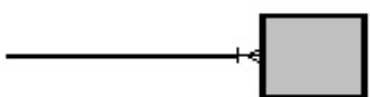
```
CREATE TABLE geschlecht (  
  name text PRIMARY KEY  
);  
  
CREATE TABLE teilnehmer (  
  vorname text  
    CHECK ( vorname != '' ),  
  nachname text,  
  matrikel_nr integer  
    PRIMARY KEY,  
  email text  
    NOT NULL  
    CHECK ( email ~ '.*@.*' )  
    UNIQUE,  
  semester integer DEFAULT 4,  
  geschlecht text  
    REFERENCES geschlecht(name)  
);
```

# Relationships (Beziehungen) revisited

Verschiedene Entitäten können zueinander in Beziehung gesetzt werden.

- In jeder Beziehung haben Entitäten gewisse Rollen
- Beziehungen können Eigenschaften (Attribute) haben
- Beziehungen haben Kardinalitäten

# Notationen

(1)	(2)	(3)	(4)
0..1		1	c
0..*		N	mc
1..1		1	1
1..*		N	m

1. UML
2. Crow Foot
3. Darstellung nach Chen-Notationen
4. Darstellung nach [Zehnder](#)  
aus [Matthiesen et al.](#)

**Whiteboard**

**ERD und Beziehungen von Teilnehmer  
Datenbank**

# Agile Aenderung

Wie ändert sich das ERM und die implementierung wenn folgende Anforderng hinzukommt:

- Die Datenbank soll für alle vergangenen und zukünftigen Datenbankkurse informationen speichern können



**Danke für die Zusammenarbeit**