

# Datenbankbasierte Web-Anwendungen

## Datenbank Schema Erstellung mit Structured Query Language (SQL)

Medieninformatik SoSe 2017

Renzo Kottmann

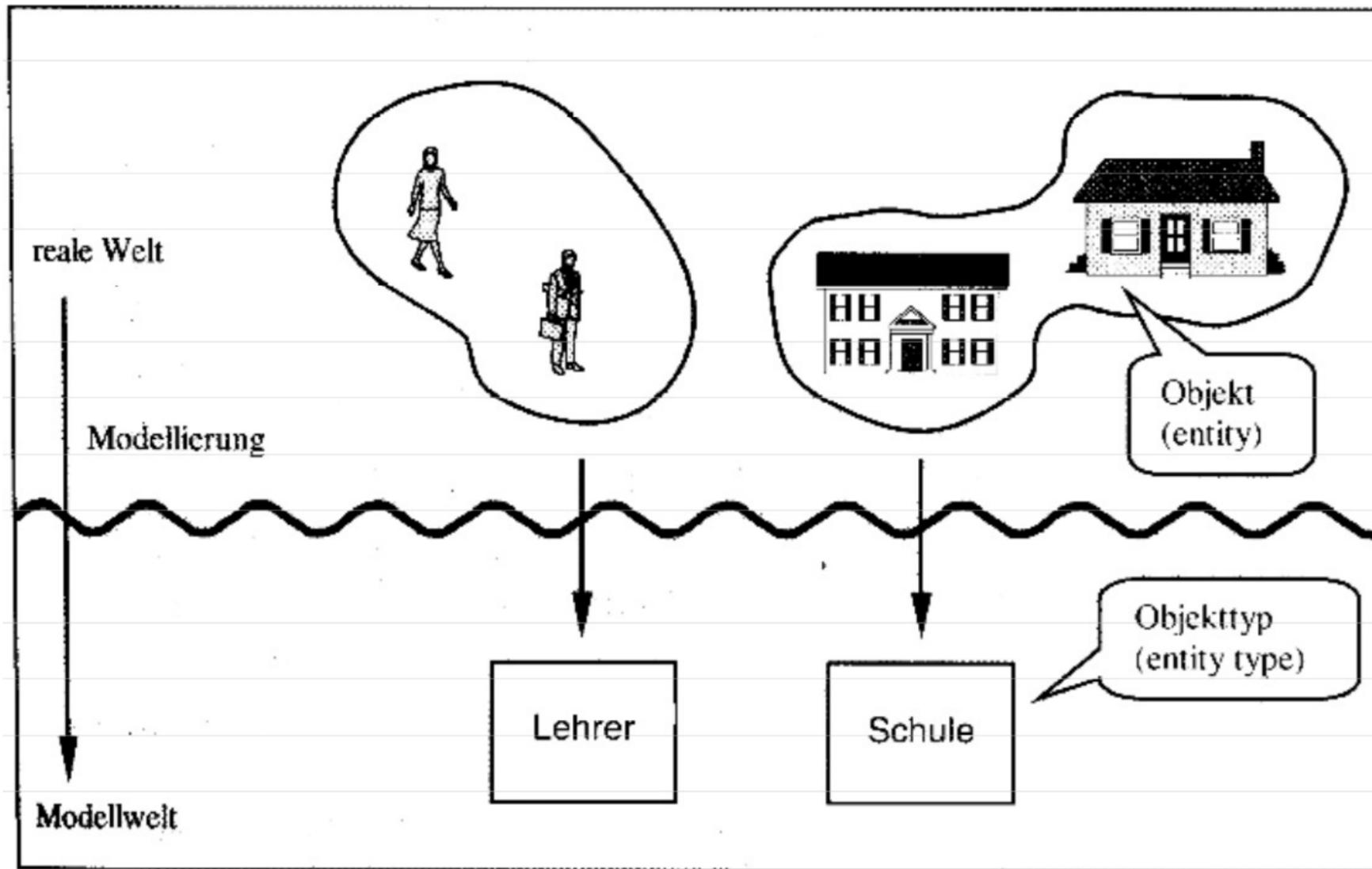


This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).

# Mini-Welt

Da wir wissen was wir wollen, können wir folgende Fragen beantworten:

- Welchen Ausschnitt der realen Welt brauchen wir?
- Welche Aspekte müssen wir Berücksichtigen?



# Entity Relationship Modellierung (ERM)

- Semantischer Datenbankentwurf
  - unabhängig von konkreten Datenbank-spezifischen Modellen
- Graphisch
- Idee simpel
  - Leider sehr viele inkonsistente Varianten

# Entity (Entität)

- Ein Entity-Relationship-Model (ERM) geht von Entitäten ( $\approx$  Objekten) aus.  
"Eine Entität ist eine eigenständige Einheit, die im Rahmen des betrachteten Modells **eindeutig identifiziert** werden kann."
- Ein Entitätstyp wird durch Attribute genauer beschrieben und stellt somit eine abstrakte Beschreibung oder Charakterisierung von Entitäten da.
- Beispiel:  
Lehrer = Entitätstyp  
Renzo  $\approx$  Entität (ein spezieller Lehrender)

# Attribute

- Eigenschaften von Entitäten werden durch Attribute beschrieben
- Attribute haben einen Namen und eine Domain (= Bestimmung der Wertmenge).

# Keys (Schlüssel)

Da die Definition einer Entität beinhaltet, dass diese zumindest im Rahmen eines Modells eindeutig identifiziert werden kann, braucht jeder Entitätstyp eine Menge von Attributen als Schlüssel.

Die Auswahl eines oder mehrerer Attribute als Schlüssel legt fest, dass es keine zwei Entitäten eines Entitätstyp geben kann die identische Attributwerte haben.

- Wichtige Eigenschaften:
  - Eindeutigkeit
  - Zuteilbarkeit

# Relationship (Beziehung)

Verschiedene Entitäten können zueinander in Beziehung gesetzt werden.

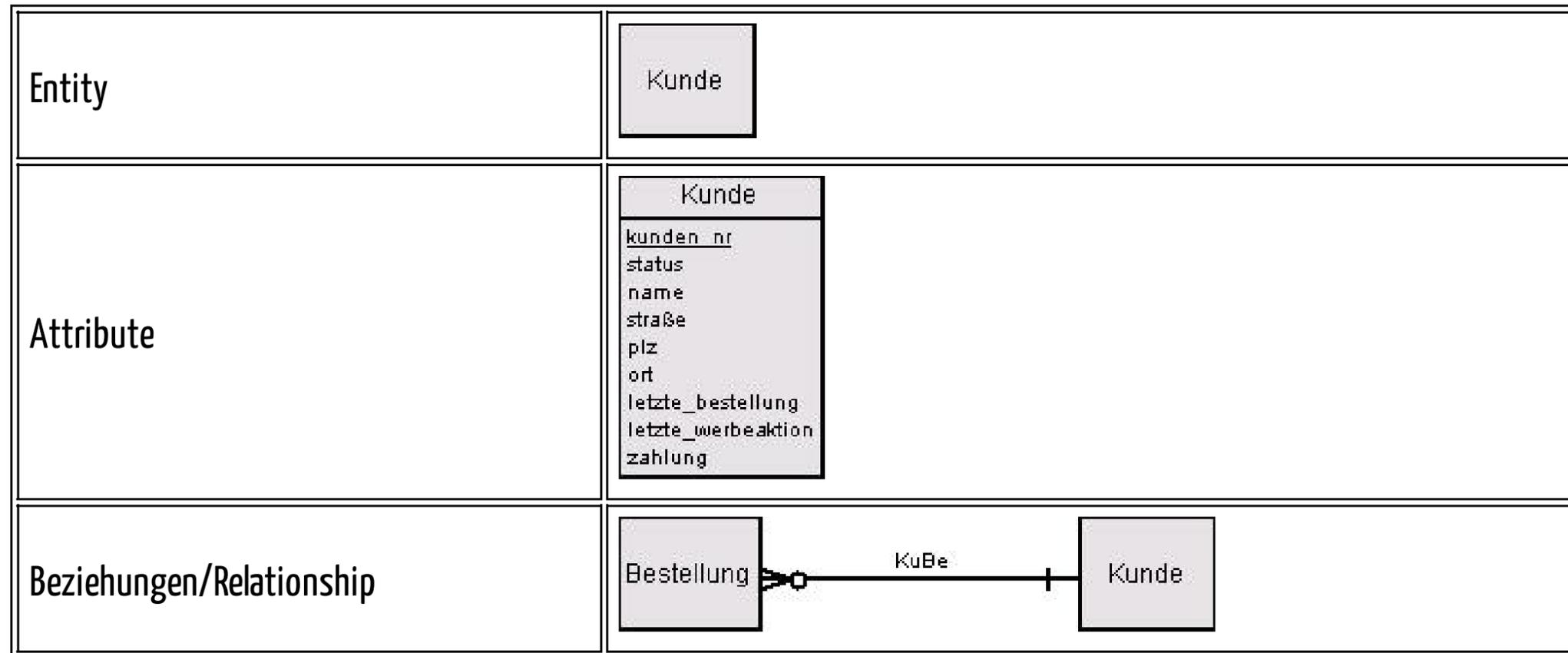
- In jeder Beziehung haben Entitäten gewisse Rollen
- Beziehungen können Eigenschaften (Attribute) haben
- Beziehungen haben Kardinalitäten

# Notationen

Es gibt verschiedene Formen ERM zu notieren (textuell und/oder graphisch):

- [Chen Notation](#)
- [Crow Foot's](#)
- [Unified Modelling Language \(UML\)](#)

# Notation



# Von ERM zur ersten Implementation

# Structured Query Language (SQL)

SQL ist eine Datenbanksprache

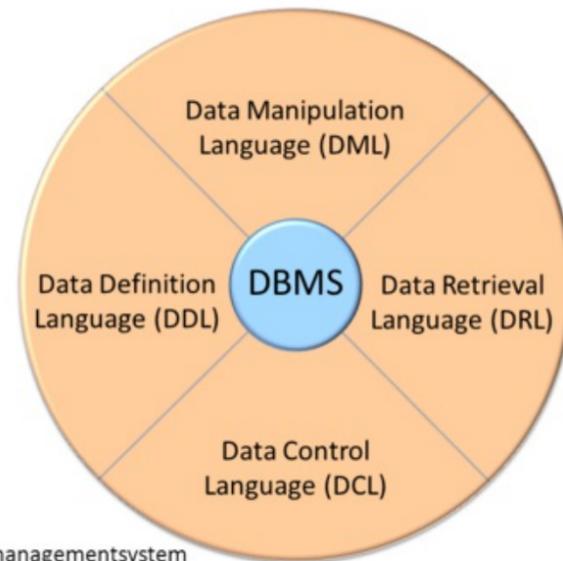
1. zur Definition von Datenstrukturen/Modellen
2. zum Bearbeiten (Einfügen, Verändern, Löschen)
3. zum Abfragen von darauf basierenden Datenbeständen
4. zur Rechtevergabe

# SQL Eigenschaften

- basiert auf relationaler Algebra
- an English angelehnt
- Deklarativ und funktional
- Fast alle Datenbanken verstehen SQL
- Standardisiert
  - PostgreSQL hat einer der besten Umsetzungen

# SQL Überblick

## Structured Query Language (SQL)



DBMS =  
Datenbankmanagementsystem

- **DDL = Data Definition Language:**  
Definition des Datenbankschemas
- **DML = Data Manipulation Language:**  
Ändern, Einfügen, Löschen und lesender Zugriff
- **DRL = Data Retrieval Language**  
Nicht standardisierte Bezeichnung des SELECT aus DML
- **DCL = Data Control Language:**  
Rechteverwaltung und Transaktionskontrolle

# SQL in PostgreSQL

- Scheinbar viele SQL Kommandos
  - [Sehr guter Überblick in der PostgreSQL Dokumentation](#)
- Die meisten sind **DDL** Kommandos
  - CREATE, DROP oder ALTER
    - Jeweils ein Eintrag pro Datenbank-Objekt
    - Folgen dem selben Syntax Schema

# Datenbank Anlegen

- CREATE DATABASE
  - [Dokumentation](#)

# Praktische Anmerkungen

## psql

- Kommandozeile = Command Line Interface (CLI)

## PgAdminIII oder 4

- Graphische Oberfläche = Graphical User Interface (GUI)

# DDL: Create Table

Teilnehmerin
Vorname
Nachname
Matrikel Nummer
email
Semester

```
CREATE TABLE teilnehmer (  
  --Spaltenname Datentyp,  
  vorname text,  
  nachname text,  
  matrikel_nr integer,  
  email text,  
  semester integer  
);
```

[s. DDL-Basics](#) und [CREATE TABLE](#)

# Integritätsbedingungen

"Integritätsbedingungen beschreiben Annahmen, die über die Daten getroffen werden, beispielsweise ein bestimmter Datentyp, ein Wertebereich oder eine Abhängigkeitsbeziehung zwischen zwei Objekten."

[Zitat Wikipedia: Integritätsbedingung](#)

# Constraints

Mit Constraints (deutsch „Einschränkung“) werden in diversen Programmiersprachen Bedingungen definiert, die zwingend vom Wert einer Variablen erfüllt werden müssen, damit der Wert ins System übernommen werden kann. In Datenbanksystemen finden Constraints rege Anwendung um den Wertebereich (Domain) von Attributen einzuschränken und Werte auf deren Zulässigkeit zu überprüfen.

[modifiziert von Wikipedia: Constraint](#)

# Datentypen sind erste Constraints

- PostgreSQL stellt [viele Datentypen zur Verfügung](#)
- Auch eigene Datentypen können definiert werden
- Verwende Datentypen so strikt wie möglich

```
CREATE TABLE teilnehmer (  
  --Spaltenname Datentyp,  
  vorname text,  
  nachname text,  
  matrikel_nr integer PRIMARY KEY,  
  email text,  
  semester integer  
);
```

# Weitere Constraints für Datenintegrität

- Primary Keys
- NULL or NOT NULL Constraints
- DEFAULT VALUES
- CHECK Constraints
- Unique Constraint

# Primary Keys

- Die Eindeutigkeit jedes Eintrags wird durch den PRIMARY KEY Ausdruck sichergestellt

```
CREATE TABLE teilnehmer (  
  -- Spaltenname Datentyp,  
  vorname text,  
  nachname text,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text,  
  semester integer  
);
```

# NULL or NOT NULL Constraints

- Implizit ist jedes Attribut einer Tabelle NULL
  - d.h. kann leer sein
- Nicht bei PRIMARY KEYS
- oder Schlüsselwort NOT NULL

```
CREATE TABLE teilnehmer (  
  -- Spaltenname Datentyp,  
  vorname text,  
  nachname text,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text NOT NULL,  
  semester integer  
);
```

Die Verwendung von `NOT NULL` implementiert hier die Anforderung: "Zu jedem Teilnehmer muss es eine E-mail geben."

**Für welche Attribute ist NOT NULL noch sinnvoll?**

# DML: Daten Einfügen

Teilnehmerin
Vorname
Nachname
Matrikel Nummer
email
Semester

```
INSERT INTO teilnehmer  
  (vorname, nachname, matrikel_nr, email, semester)  
VALUES  
  ('renzo', 'kottmann', 007, 'renzo@007.bond', 100);
```

s. [INSERT Dokumentation](#) und [Kommando Referenz](#)

# Weitere Massnahmen zur Gestaltung der Datenintegrität

- DEFAULT VALUES
- CHECK Constraints
- Unique Constraint

# Default Values Constraint

- Für jedes Attribut kann man einen Standard-Wert festlegen
- der Standardwert wird dann eingetragen, wenn
  - kein Wert angegeben wurde
  - oder explizit auf DEFAULT gesetzt wurde

```
CREATE TABLE teilnehmer (  
  -- Spaltenname Datentyp,  
  vorname text,  
  nachname text,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text NOT NULL,  
  semester integer DEFAULT 4  
);
```

Die Verwendung von `DEFAULT`  
implementiert hier die Anforderung:  
"Bei jedem Teilnehmer ist das Standard S

# Default Values Constraint

```
INSERT INTO teilnehmer (vorname, nachname, matrikel_nr, email)
VALUES ('renzo', 'kottmann', 007, 'renzo@007.bond');
```

führt zu einem Eintrag mit:

vorname	nachname	matrikel_nr	email	semester
renzo	kottmann	7	renzo@007.bond	4

# Check Constraint

- Ein Wert in ein oder mehreren Spalten muss einer boolschen Funktion entsprechen
  - es muss TRUE ergeben

```
CREATE TABLE teilnehmer (  
  -- Spaltenname Datentyp,  
  vorname text CHECK ( vorname != '' ),  
  nachname text,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text NOT NULL CHECK ( email ~ '  
  semester integer DEFAULT 4  
);
```

# Unique Constraint

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

```
CREATE TABLE teilnehmer (  
  -- Spaltenname Datentyp,  
  vorname text CHECK ( vorname != '' ),  
  nachname text,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text NOT NULL CHECK ( email ~ '@' ),  
  semester integer DEFAULT 4  
);  
Jeder Teilnehmer muss eine andere E-Mail
```

# 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 (  
  -- Spaltenname Datentyp,  
  vorname text CHECK ( vorname != '' ),  
  nachname text,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text NOT NULL CHECK ( email ~ '  
  semester integer DEFAULT 4  
);  
Jeder Teilnehmer muss eine andere E-Mail
```

# 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 (  
  -- Spaltenname Datentyp,  
  vorname person_name CHECK ( vorname != '' ),  
  nachname person_name,  
  -- Simpler Primary Key  
  matrikel_nr integer PRIMARY KEY,  
  email text NOT NULL CHECK ( email ~ '.*@.*' ) UNIQUE,  
  semester integer DEFAULT 4  
);
```

# Weiterführende Fragen:

1. Wie ändert sich das ERM und die implementierung wenn folgende Anforderung hinzukommt:
  - Die Datenbank soll für alle vergangenen und zukünftigen Datenbankkurse informationen speichern können

**Danke für die Zusammenarbeit**