Modul Datenbanken

Vorlesung 4

Vom Datenbankentwurf zur Implementierung II

IFI Wintersemester 2016/17

by Renzo Kottmann

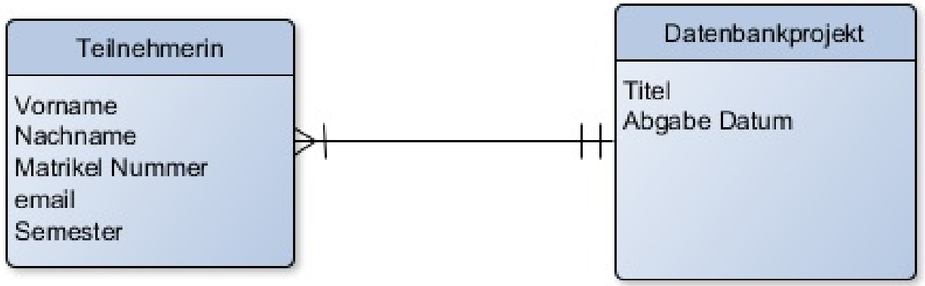
Beim letzten Mal besprochen

- Entity Relationship Modellierung

Beim letzten Mal nicht besprochen

- SQL Ueberblick
- Erstellung einer Tabelle/Relation

ERM Teilnehmerinnen



Structured Query Language (SQL)

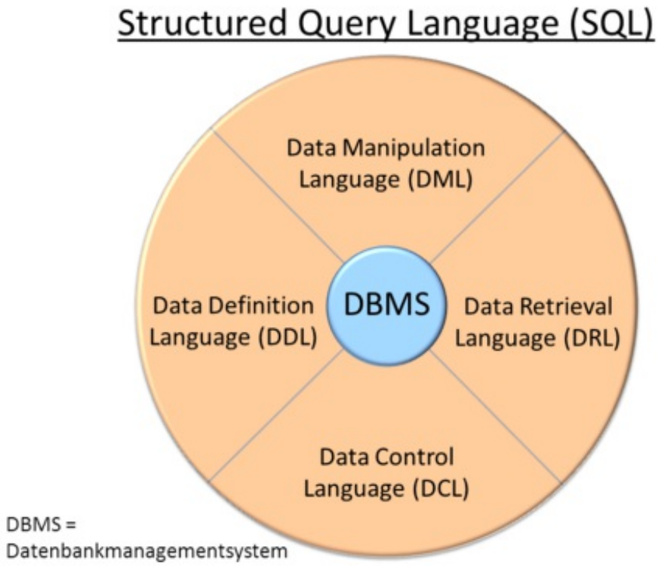
SQL ist eine Datenbanksprache

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



- DML = Data Manipluation Language: Ändern, Einfügen, Löschen und lesender Zugriff
- DDL = Data Defintion Language: Definition des Datenbankschemas
- DCL = Data Control Language: Rechteverwaltung und Transaktionskontrolle

SQL in PostgreSQL

- Scheinbar viele SQL Kommandos
  - Sehr guter Ueberblick 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

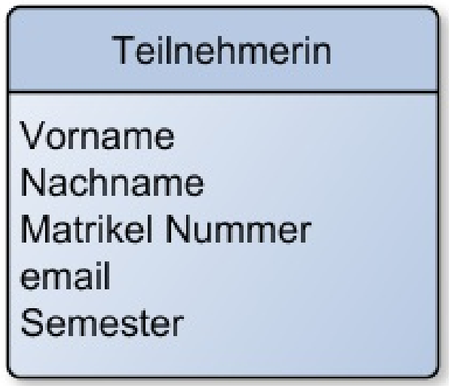
psql

- Kommandozeile = Command Line Interface (CLI)

PgAdminIII oder 4

- Graphische Oberflaeche = Graphical User Interface (GUI)

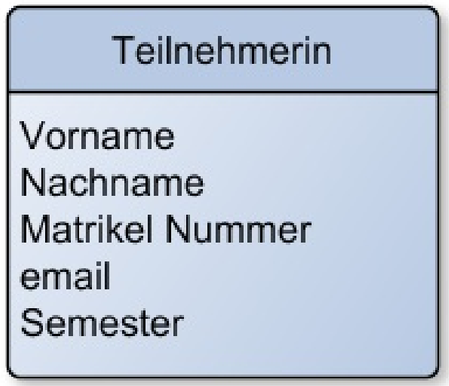
DDL: Create Table




```sql
CREATE TABLE teilnehmer (
  --Spalten Name dann Datentyp,
  vorname text,
  nachname text,
  matrikel_nr integer,
  email text,
  semester integer
);
```

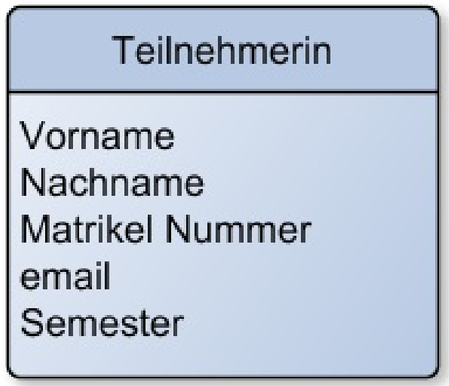
s. Table Basics und CREATE TABLE Dokumentation

DDL: Create Table Primary Key



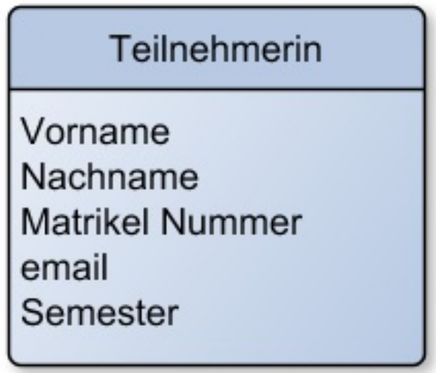

```sql
CREATE TABLE teilnehmer (
  --Spalten Name dann Datentyp,
  vorname text,
  nachname text,
  matrikel_nr integer,
  email text,
  semester integer
);
```

DDL: Create Table Primary Key




```sql
CREATE TABLE teilnehmer (
--Spalten Name Datentyp,
  vorname text,
  nachname text,
  -- Simpler (nicht bester Primary Key)
  matrikel_nr integer PRIMARY KEY,
  email text,
  semester integer
);
```

DML: Daten Einfügen




```
INSERT INTO teilnehmer
  (vorname, nachname, matrikel_nr, email, semester)
VALUES
  ('renzo','kottmann',007,'renzo@007.bond', 0);
```

s. Inserting Data und INSERT Kommando

Teilnehmerinnen Datenbank

- Erarbeitetes Ergebnis dieser Vorlesung/Uebung

Weiterfuehrende Fragen:

1. Welche weiteren SQL-Befehle für Datenmodell-Management (DDL) gibt es noch?

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

3. Welche Datentypen gibt es schon in PostgreSQL?

4. Kann man eigene Datentypen definieren?
   - Wenn ja, welche Möglichkeiten gibt es?

Schoene Woche noch :)

Material fuer Vorlesung 5

folgende slides nur falls ueberhaupt Zeit

Pizza Lieferservice Spezifikation

Erstes ER Diagram

Umsetzung der einzelnen Entitaeten

- Rohes SQL file mit ersten Testdaten

Massnahmen zur Gestaltung der Datenintegritaet

- Datentypen
- Primary Keys
- NULL or NOT NULL Constraints

Datentypen

- PostgreSQL stellt viele Datentypen zur Verfuegung
- Auch eigene Dataentypen koennen definiert werden

```
CREATE TABLE pizza (
  name text PRIMARY KEY,
  img text
);
```

Primary Keys

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

```
CREATE TABLE pizza (
  name text PRIMARY KEY
  -- name kann es nur einmal geben,
  img text
);
```

NULL or NOT NULL Constraints

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

```
CREATE TABLE pizza (
  name text PRIMARY KEY
  -- name kann es nur einmal geben,
  img text NOT NULL
  -- es muss einen Eintrag
  -- fuer image geben
);

Die Verwendung von `NOT NULL`
implementiert hier die Anforderung:
"Zu jeder Pizza muss es ein Bild geben."
```

Weitere Massnahmen zur Gestaltung der Datenintegritaet

- DEFAULT VALUES
- CHECK Constraints
- Unique Constraint

Default Constraint

- Fuer jedes Attribut kann man einen Standard-Wert festlegen
- D.h. der Standwert wird eingetragen, falls nicht explizit ein anderer Wert angegeben wurde
- `INSERT INTO PIZZA (name) VALUES ('salami');` fuehrt zu einem Eintrag mit

name	**img**
salami	placeholder

```
CREATE TABLE pizza (
  name text PRIMARY KEY
  -- name kann es nur einmal geben,
  img text NOT NULL
    DEFAULT 'placeholder'
  -- es muss einen Eintrag
  -- fuer image geben
);

Die Verwendung von `DEFAULT`
implementiert hier die Anforderung:
"Zu jeder Pizza muss es ein Bild geben,
zumindest ein Platzhalter Bild"
```

Check Constraint

- Ein Wert in ein oder meherer Spalten muss einer boolean funktion entsprechen

```
CREATE TABLE pizza (
  name text
    check ( name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
    DEFAULT 'placeholder'
    REFERENCES image (location)
);
```

Unique Constraint

- Alle Werte ein oder meherer Spalten muss eindeutig sein
- Damit werden weitere Schluessel implementiert

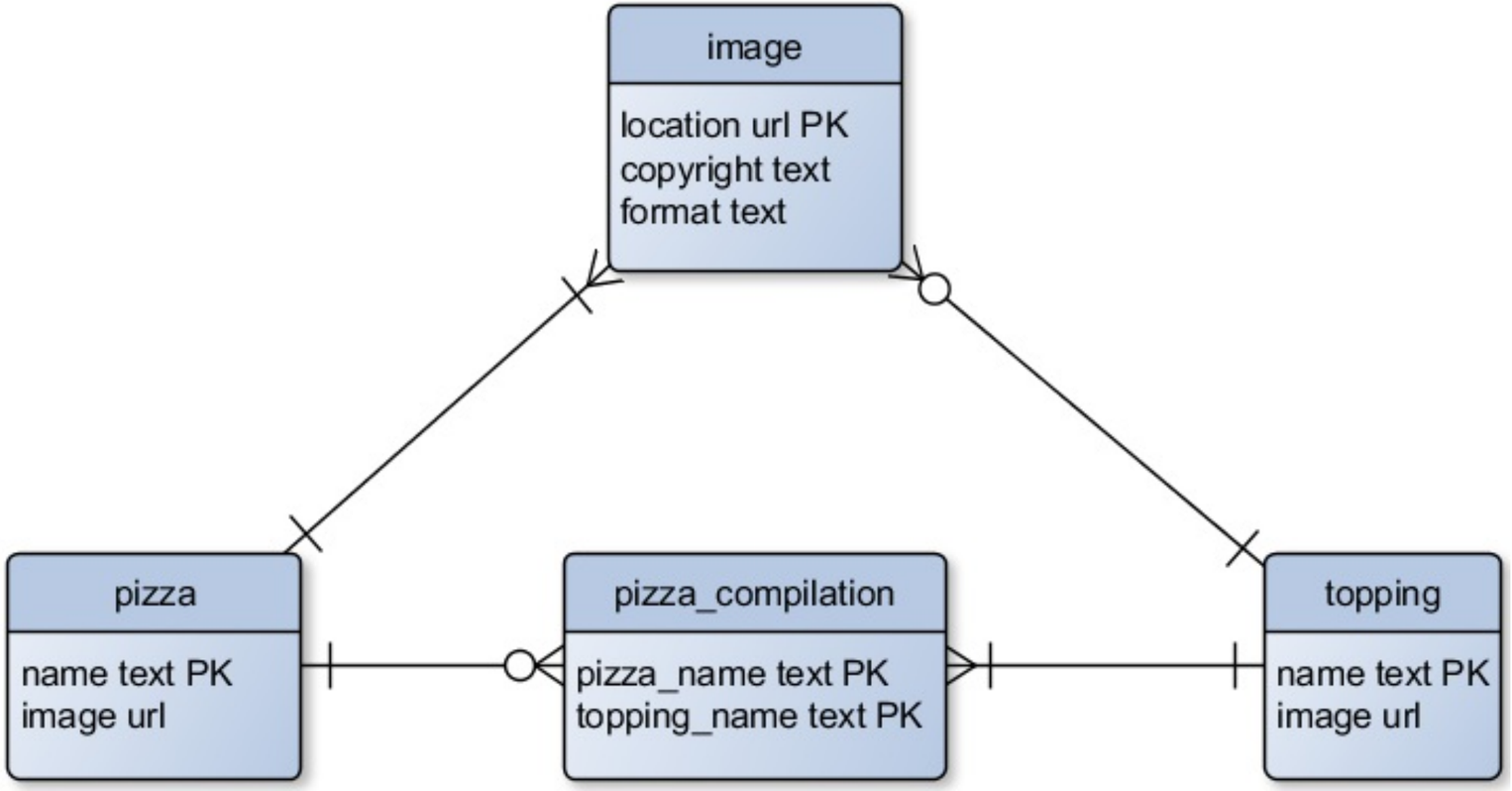
```
CREATE TABLE pizza (
  name text
    check ( name != ''::text)
    PRIMARY KEY,
  img text NOT NULL UNIQUE
    DEFAULT 'placeholder'
    REFERENCES image (location)
);
Jede Pizza muss ein anderes Bild haben.
```

Logisch gesehen: Primary Key

- Ein `PRIMARY KEY` ist nichts anderes als ein `UNIQUE NOT NULL`
- D.h. es kann mehere Schluessel geben, aber nur einer wird als `PRIMARY KEY` gewaehlt

```
CREATE TABLE pizza (
  name text
    check ( name != ''::text)
    PRIMARY KEY,
  img text NOT NULL UNIQUE
    DEFAULT 'placeholder'
    REFERENCES image (location)
);
Jede Pizza muss ein anderes Bild haben.
```

Welche Beziehungen?



Umsetzung der Beziehungen

- Durch Foreign Keys (Fremdschluessel)
- Legen die genauen Bedingung der Beziehung fest
  - Wichtige Frage: Was identifiziert die Beziehung?!?

Foreign Keys (1 to many)

- Stellt Verknuepfungen zwischen Relationen/Tabellen her
  - Dies geschieht ueber Werte
  - Die Abhaengige Relation referenziert die Quell-Relation
  - Garantiert existenz des Quell-Eintrags

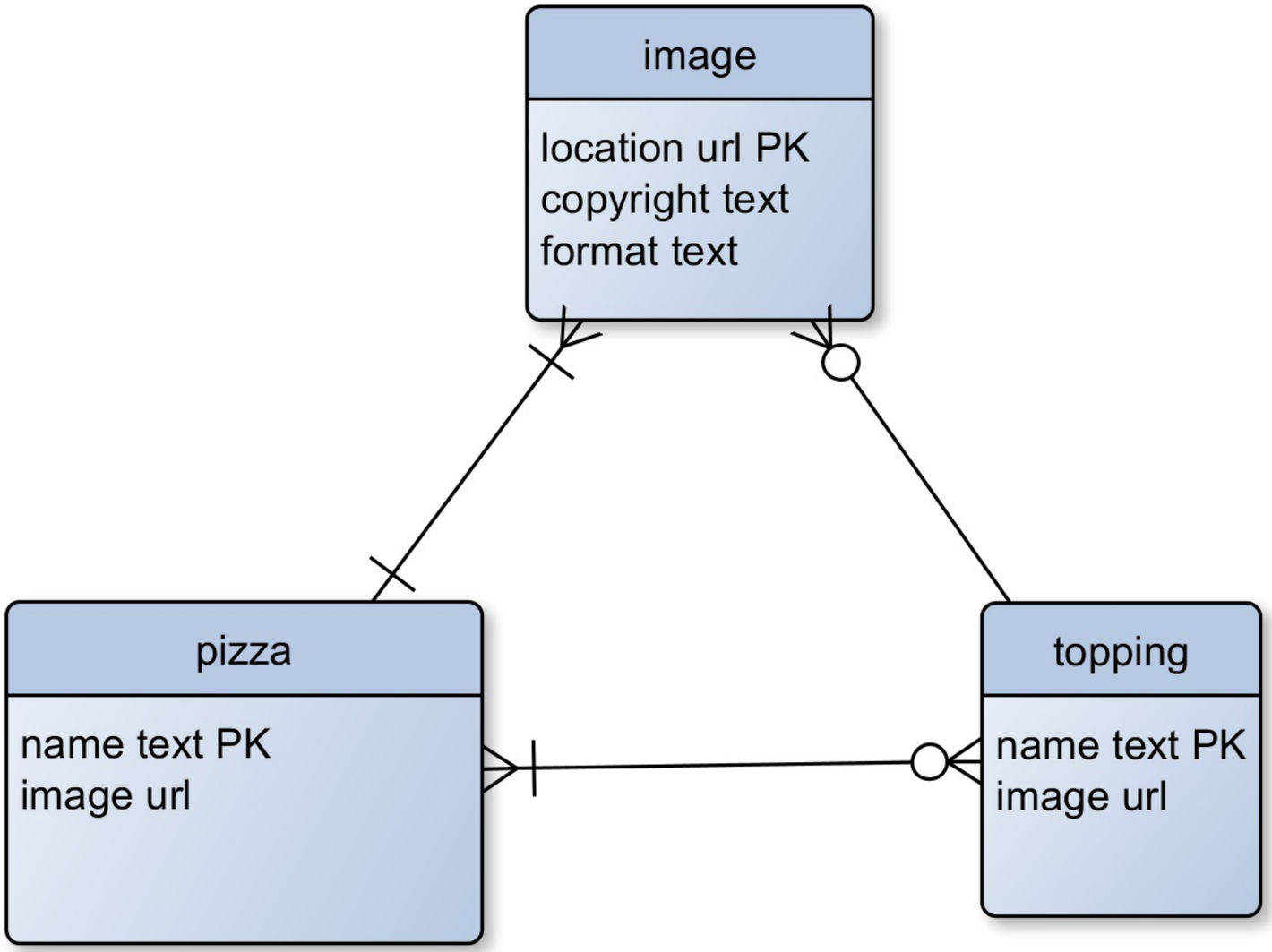
```
CREATE TABLE image (
  location text PRIMARY KEY,
  name text
    NOT NULL
    DEFAULT 'placeholder',
  copyright text
    NOT NULL DEFAULT 'unknown',
  type text
    NOT NULL DEFAULT 'unknown'
);
CREATE TABLE pizza (
  name text
    check (name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
    DEFAULT 'placeholder'
    REFERENCES image (location)
    -- referenz auf PK von image
);
INSERT INTO
  image (location, copyright, type)
  VALUES
  ('file://here', 'Renzo Kottmann', 'png');
INSERT INTO pizza (name,img)
  VALUES
  ('Salami', 'file://here');
```

Foreign Keys (1 to many)

- Garantiert existenz des Quell-Eintrags
  - Werte der Quell-Relation muessen in Abhaengige Relation eingetragen werden

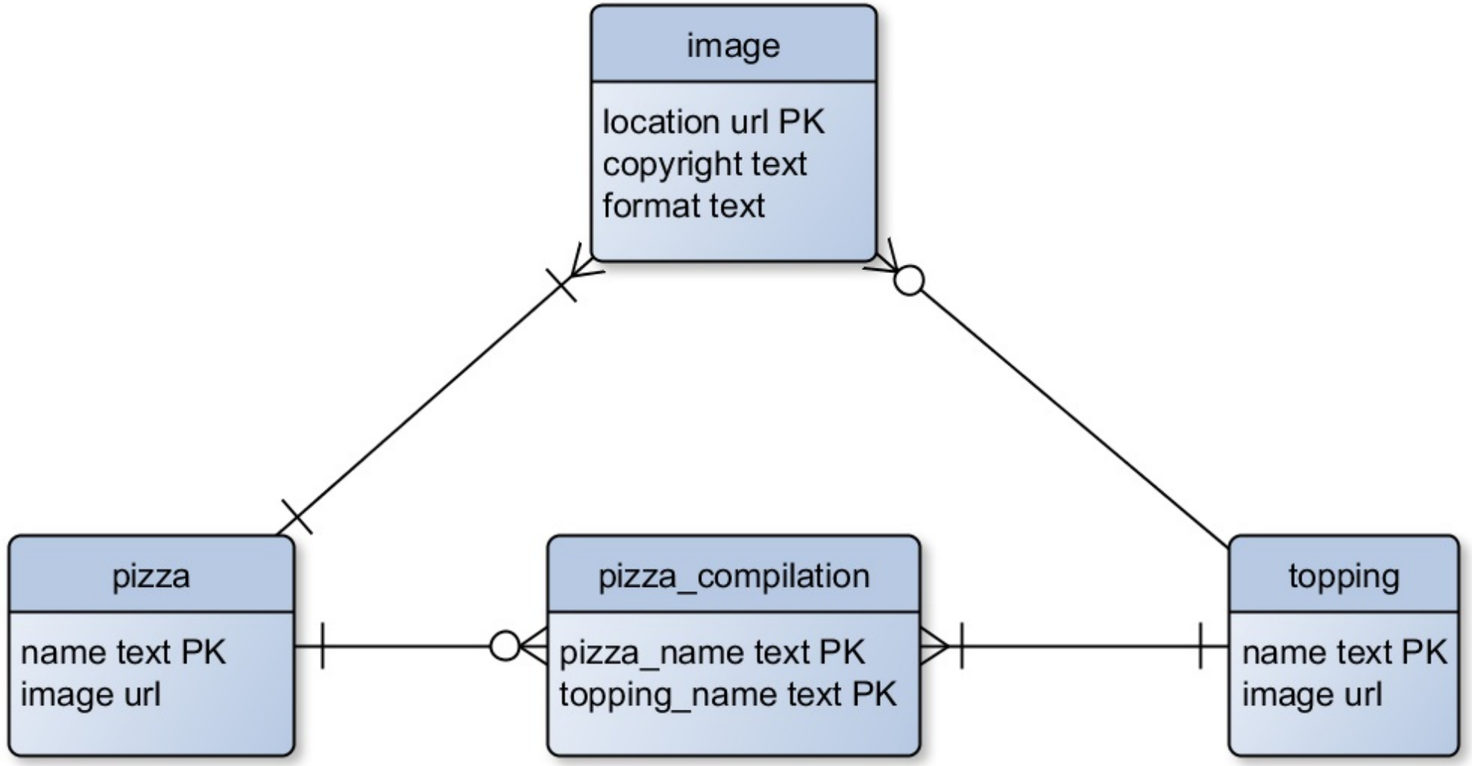
```
CREATE TABLE image (
  location text PRIMARY KEY,
  name text
    NOT NULL
    DEFAULT 'placeholder',
  copyright text
    NOT NULL DEFAULT 'unknown',
  type text
    NOT NULL DEFAULT 'unknown'
);
CREATE TABLE pizza (
  name text
    check ( name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
    DEFAULT 'placeholder'
    REFERENCES image (location)
    -- referenz auf PK von image
);
INSERT INTO
  image (location, copyright, type)
  VALUES
  ('file://here', 'Renzo Kottmann', 'png');
INSERT INTO pizza (name,img)
  VALUES
  ('Salami', 'file://here');
```

Foreign Keys (many to many)



Foreign Keys (many to many)

- Implementierung durch neue "Beziehungs"-Relation



Foreign Keys (many to many)

- Neue Tabelle, die auf die beiden existierenden referenziert
  - Primary Key der neuen Tabelle ist Kombination der PKs der existierenden Tabellen

```
CREATE TABLE pizza (
  name text
    check ( name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
    DEFAULT 'placeholder'
    REFERENCES image (location)
    -- referenz auf PK von image
);
CREATE TABLE topping (
  name text PRIMARY KEY,
  img text
);
CREATE TABLE pizza_compilation (
  pizza_name text
    references pizza(name),
  topping_name text
    references topping(name),
  PRIMARY KEY (pizza_name, topping_name)
);
```

Referenzen:

- M. Unterstein and G. Matthiessen, Relationale Datenbanken und SQL in Theorie und Praxis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.