Modul Datenbanken

Vorlesung 6

Datenbank Implementierung mit Beziehungen

IFI Wintersemester 2016/17

by Renzo Kottmann

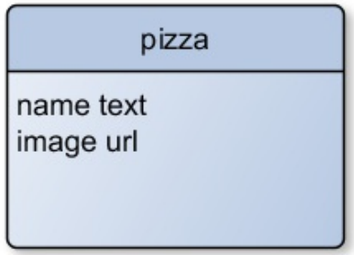
Beim letzten Mal besprochen

- Constraints auf einzelnen Tabellen

Beim letzten Mal NICHT fertig besprochen

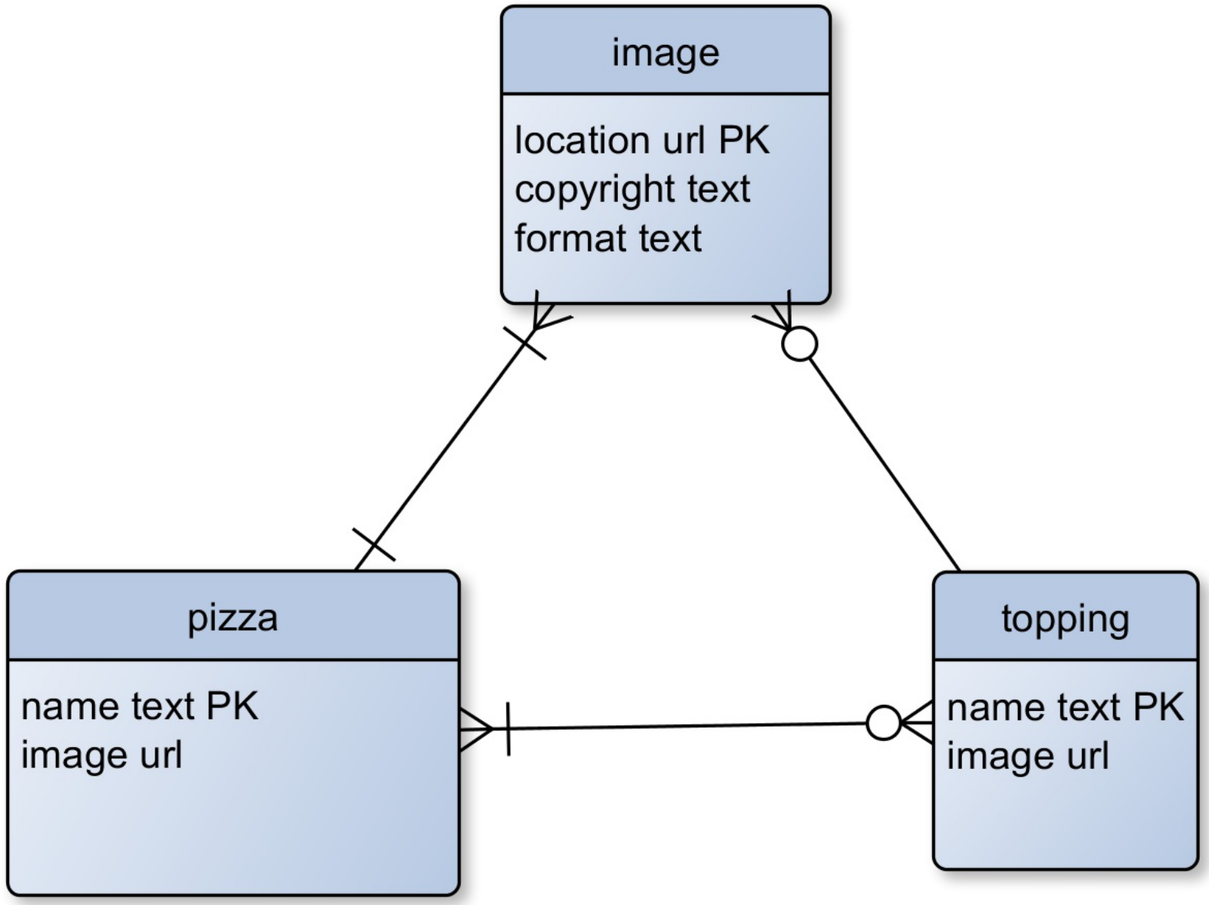
- SQL Implementierung von Entitaets-Beziehungen

Welche Beziehungen?



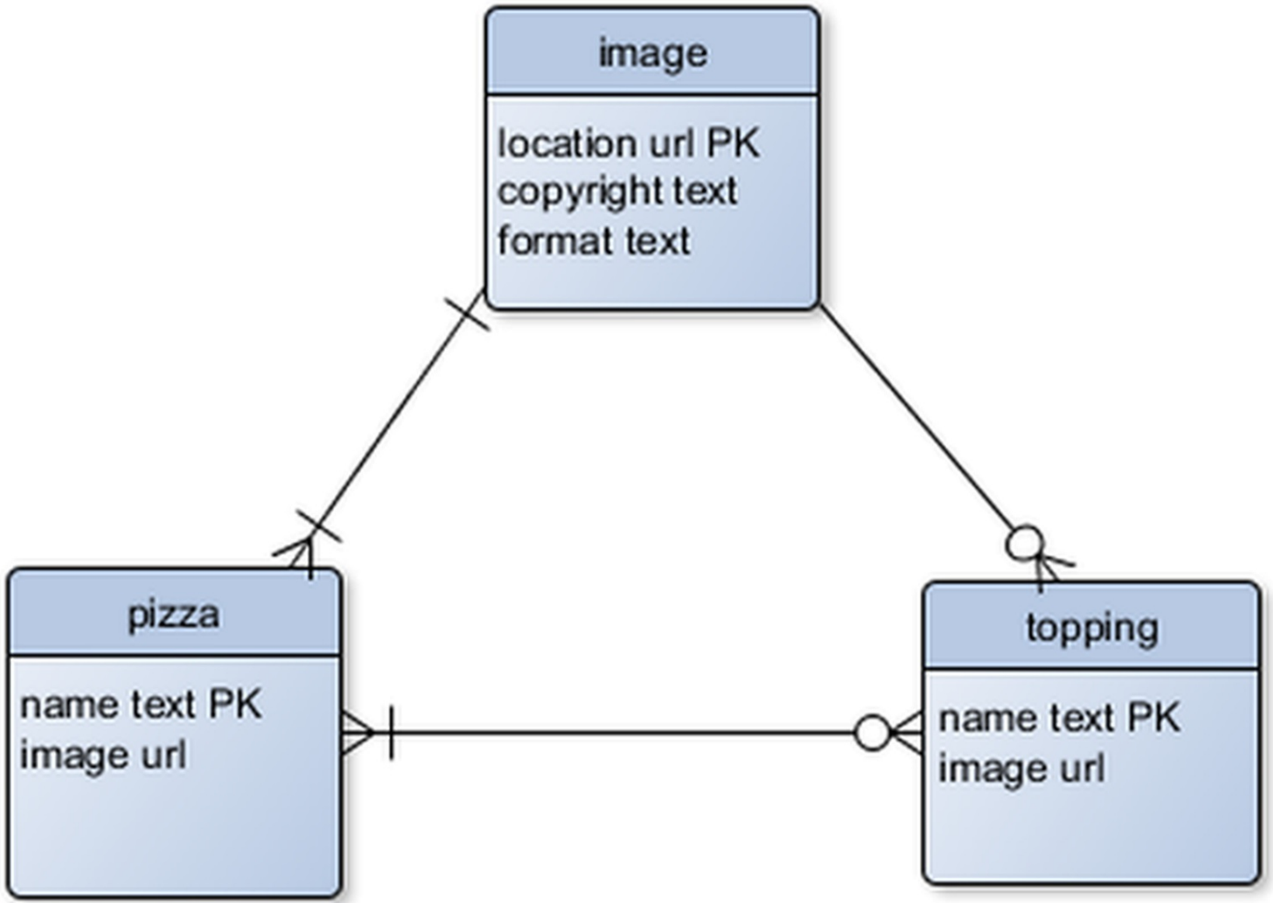
Welche Beziehungen?

Dieses ERD entsprach nicht der Implementierung aus Vorlesung 5



Welche Beziehungen?

Korrigiert!



Umsetzung der Beziehungen

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

Foreign Keys (1 to many)

- Stellt Verknuepfungen zwischen Relationen/Tabellen her
  - Die Abhaengige Relation referenziert die Quell-Relation

```
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
);
```

Foreign Keys (1 to many)

- Werte-basiert
  - Werte der Quell-Relation muessen in abhaengige Relation eingetragen werden

```
INSERT INTO image
 (location, copyright, type)
VALUES
 ('file://here', 'Renzo', 'png');

INSERT INTO pizza
 (name,img)
VALUES
 ('Salami', 'file://here');
```

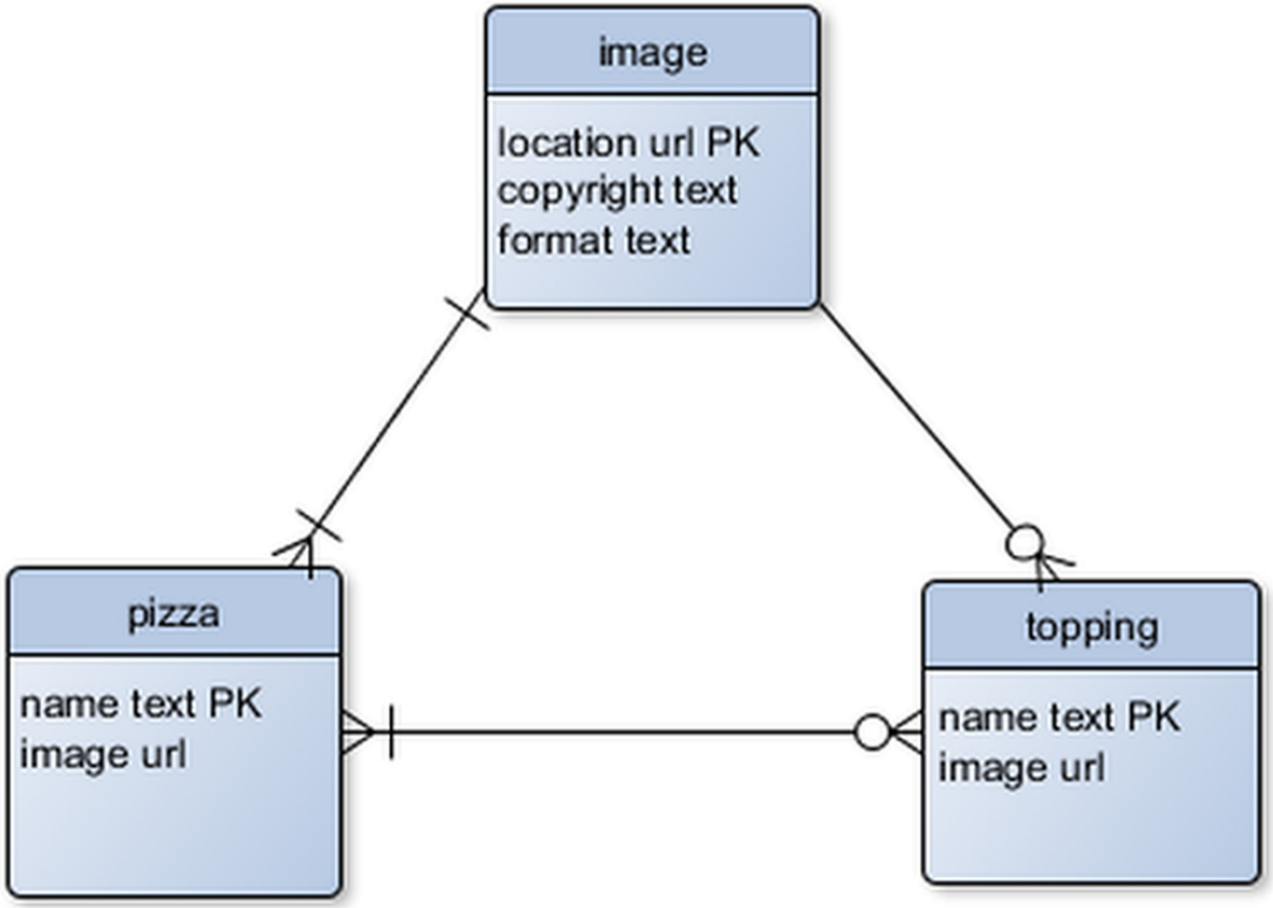
```
CREATE TABLE image (
  location text PRIMARY KEY,
  name text
    NOT NULL
    DEFAULT 'placeholder',
  copyright text
    NOT NULL DEFAULT 'unknown',
  type text
    NOT NULL DEFAULT 'unknown'
);
CREATE TABLE pizza (
  name text
    check (name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
    DEFAULT 'placeholder'
    REFERENCES image (location)
    -- referenz auf PK von image
);
```

Foreign Keys (1 to many)

- Bei Einfuegung einer Zeile in abhaengiger Relation:
  - Garantiert existenz des Quell-Eintrags
- Bei Loeschung einer Quell-Zeile
  - Garantiert, dass Quell-Zeile nur geloescht werden kann, wenn es keinen Eintrag in abhaengiger Relation gibt

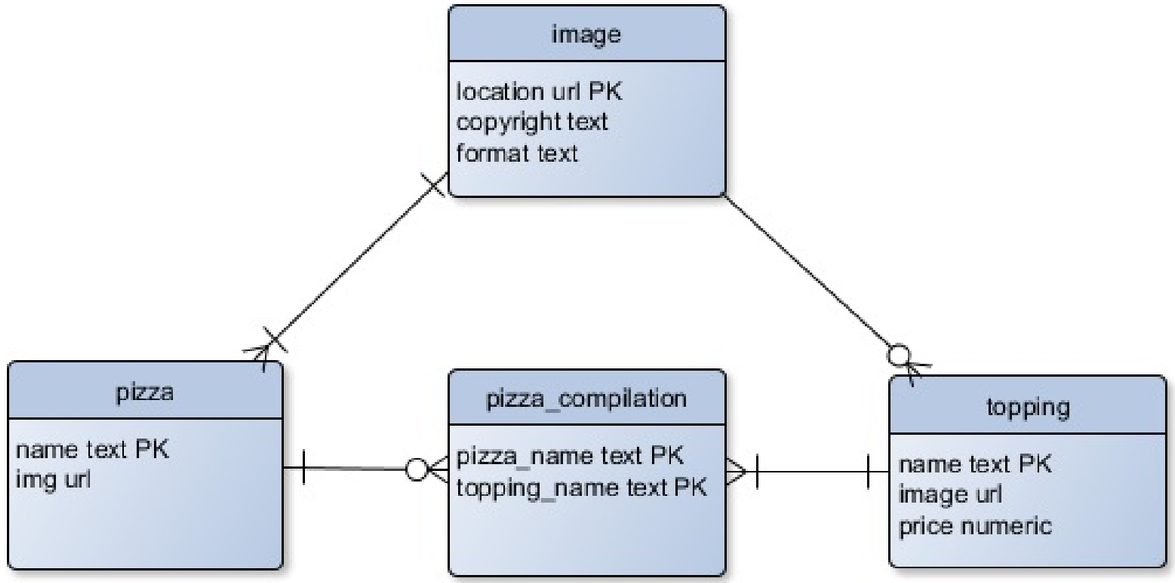
```
CREATE TABLE image (
  location text PRIMARY KEY,
  name text
    NOT NULL
    DEFAULT 'placeholder',
  copyright text
    NOT NULL DEFAULT 'unknown',
  type text
    NOT NULL DEFAULT 'unknown'
);
CREATE TABLE pizza (
  name text
    check (name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
    DEFAULT 'placeholder'
    REFERENCES image (location)
    -- referenz auf PK von image
);
```

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)
);
```

Vertiefung Wertebereiche (Domaenen)

Arten von Daten

1. Zahlen (integer, numeric, double precision...)
2. Free Text (text, char)
3. Enumeration
4. Code-List
5. Komplexe Zusamensetzungen 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

Implementierung mit SQL Commands

1. `check` constraint:

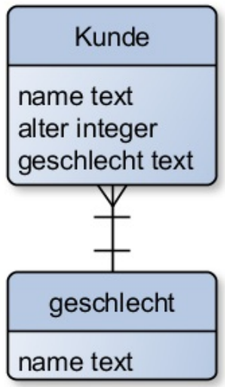
   ```
   gender text check (gender in ['maennlich','weiblich'])
   ```

2. Enumerated Types:

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

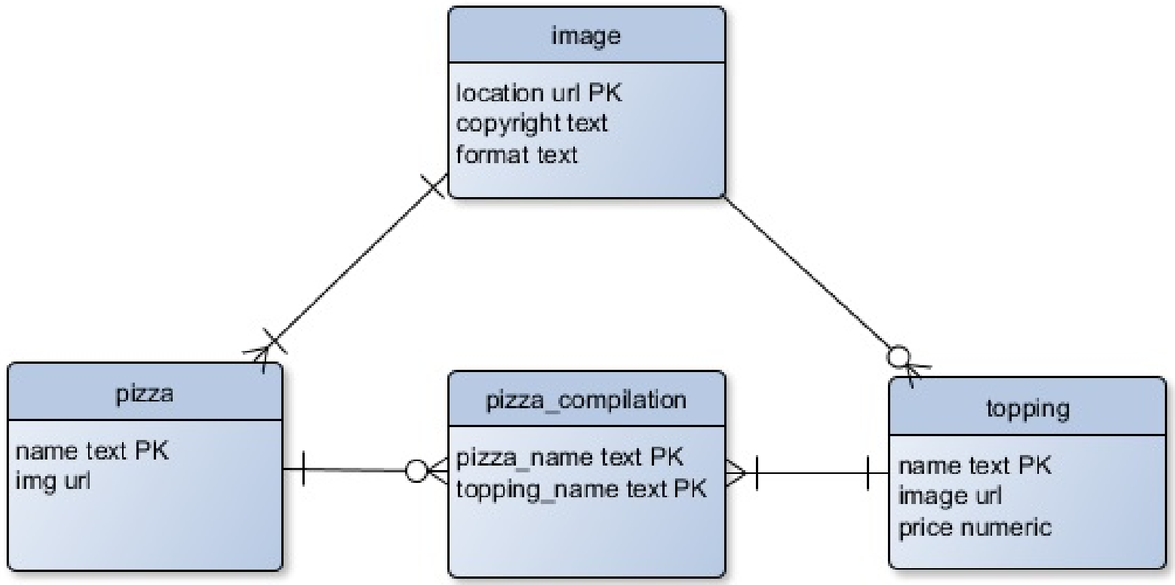
Implementierung mit Hilfe einer Tabelle

Beispiel Kunde




```sql
CREATE TABLE geschlecht (
  name text PRIMARY KEY
);
INSERT INTO geschlecht
  VALUES ('maennlich'),('weiblich');
CREATE TABLE kunde (
  name text PRIMARY KEY,
  alter integer,
  geschlecht text
    REFERENCES geschlecht(name)
);
INSERT INTO kunde (name, alter, geschlecht)
   VALUES ('renzo','100','maennlich');
```

Aktuelles ER Diagram



SQL Kommando: CREATE DOMAIN

- Definition von eigenen Datentypen basierend of existierenden

```
CREATE DOMAIN url AS text CHECK (
  VALUE ~ '^file|http'
);
```

Von Anfragen zu Abfragen

Anfragen

- Gruende fuer relationale Datenbanken:
  1. Persistente, sichere und strukturierte Datenspeicherung
  2. Effiziente Anfragen! Z.b. wieviel Umsatz hat die Pizzeria am 15.10.2015 gemacht?

Anfragen

- Gruende fuer relationale Datenbanken:
  1. Persistente, sichere und strukturierte Datenspeicherung
  2. Effiziente Anfragen! Z.b. wieviel Umsatz hat die Pizzeria am 15.10.2015 gemacht?
- SQL hat nur einen Befehl dafuer: **SELECT**

Anatomy von SELECT

```
SELECT *          -- welche Spalten sollen wie angezeigt werden
  FROM tabelle    -- Daten welcher Tabelle
 WHERE true       -- Selektionesbedingungen: nur Daten, die Kriterium entsprechen
```

Anatomy von SELECT

```
SELECT *         -- welche Spalten sollen wie angezeigt werden
  FROM tabelle   -- Daten welcher Tabelle
 WHERE true      -- Selektionesbedingungen: nur Daten, die Kriterium entsprechen
```

Kann gelesen werden als:

```
 Zeige mir alle Spalten der Tabelle "tabelle" an und davon alle Zeilen.
```

Anatomy von SELECT

```
SELECT *         -- welche Spalten sollen wie angezeigt werden
  FROM tabelle   -- Daten welcher Tabelle
 WHERE true      -- Selektionesbedingungen: nur Daten, die Kriterium entsprechen
```

Kann gelesen werden als:

```
Zeige mir alle Spalten der Tabelle "tabelle" an und davon alle Zeilen.
```

Datenbank interpretiert das in der Reihenfolge `FROM`, `WHERE`, `'*'` (Spalten)

```
Hole aus der Tabelle "tabelle" alle Zeilen die der Bedingung 'true' entsprechen
und zeige davon alle Spalten an.
```

Konkretes SELECT

```sql
SELECT *         -- * (asterisk) heisst alle spalten, wie sie sind
  FROM "order"; -- Daten der Tabelle mit dem Namen "order"
```

Boolsche `WHERE` Bedingung kann weggelassen werden, wenn man alle Zeilen will.

Referenzen:

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