# Modul Datenbanken

# Vorlesung 5

# Datenbank Implementierung mit Integritätsbedingungen

**IFI Wintersemester 2016/17**

by Renzo Kottmann

# Beim letzten Mal besprochen

- Structured Query Language
- Datenbanken anlegen
- Implementienrung mit 1. Tabelle (SQL `CREATE TABLE`)
- Daten einfuegen mit SQL `INSERT`

# Wiederholungsfragen:

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. Was sind Primary Keys und wie implementiert man diese
   - Wieso braucht man Primary Keys ueberhaupt?

# Integritätsbedingungen

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

Ziatat 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

- so strikt wie moeglich

```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
);
```

# Neues Datenbank Projekt

## Pizza Lieferservice Spezifikation

# Erstes ER Diagram

# Massnahmen zur Gestaltung der Datenintegritaet

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

# 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
  - der Standwert wird eingetragen, falls kein Wert angegeben wurde

```
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"
```

- `INSERT INTO PIZZA (name) VALUES ('salami');` fuehrt zu einem Eintrag mit:

| name | img |
|---|---|
| salami | placeholder |

# Check Constraint

- Ein Wert in ein oder meheren Spalten muss einer boolean Funktion entsprechen
  - muss `TRUE` ergeben

```
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 muessen 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.
```

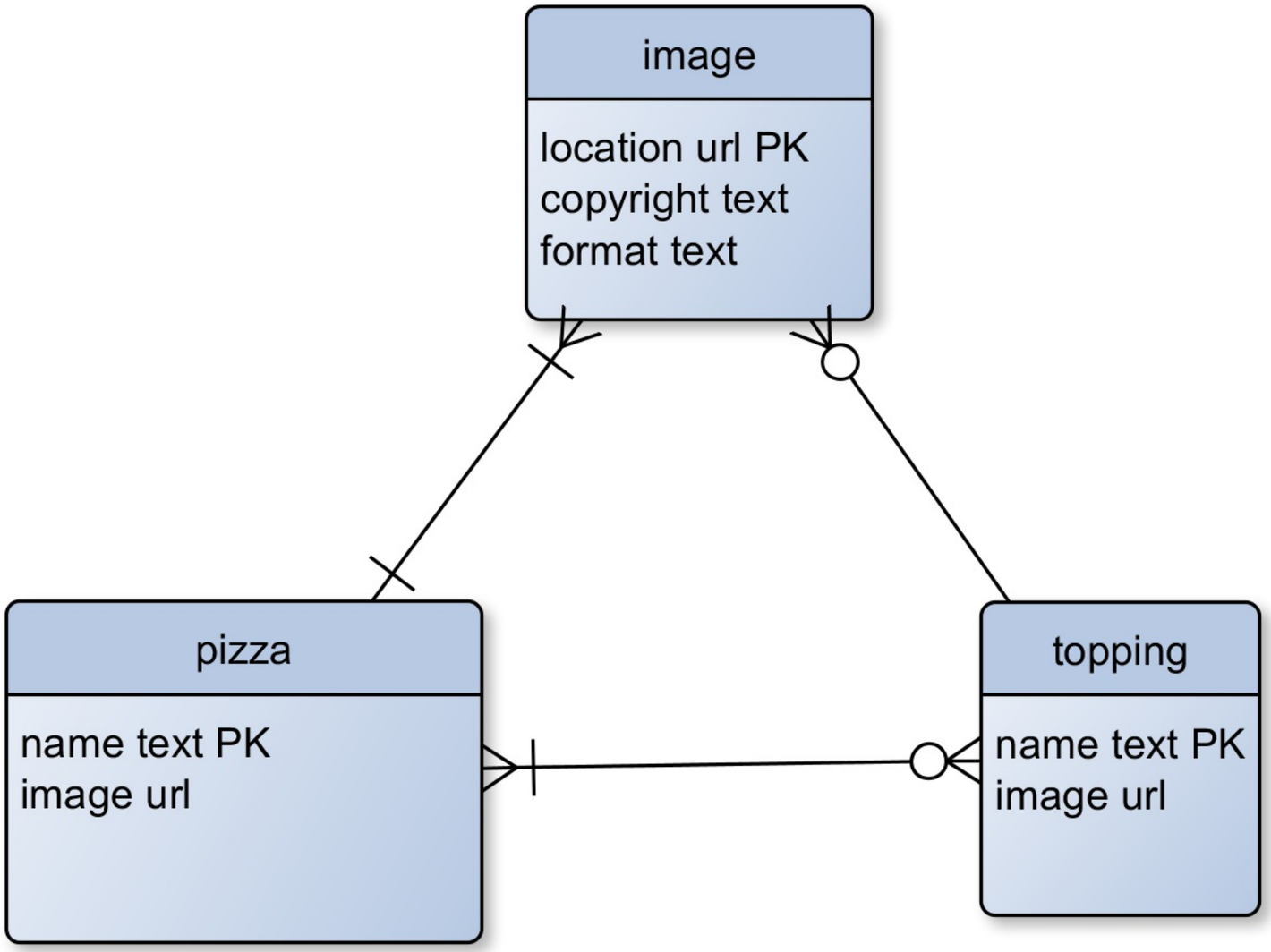
# Logisch gesehen: Primary Key

- 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?

# Welche Beziehungen?



# 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');
```

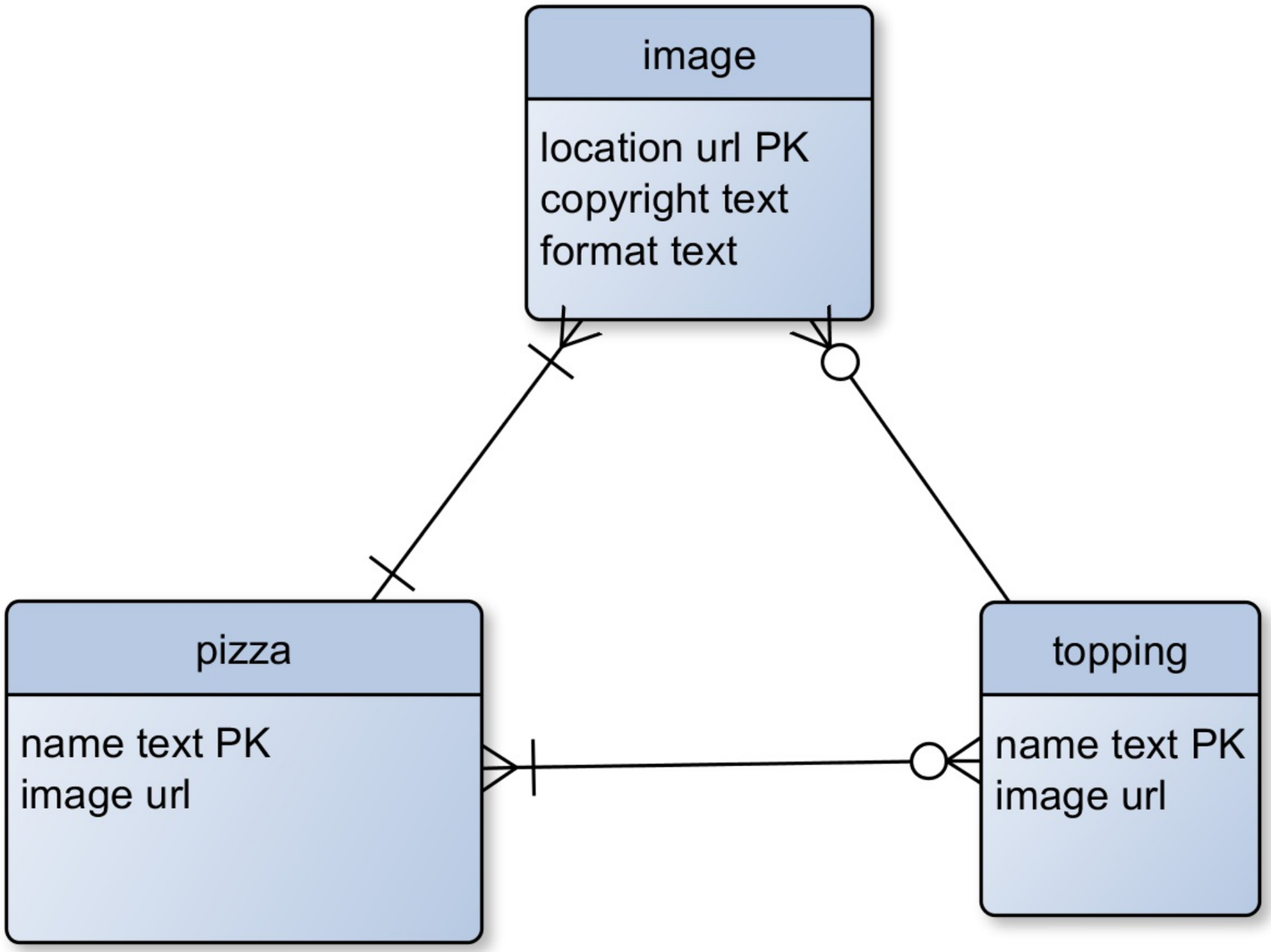
```
CREATE TABLE image (
  location text PRIMARY KEY,
  name text
    NOT NULL
    DEFAULT 'placeholder',
  copyright text
    NOT NULL DEFAULT 'unknown',
  type text
    NOT NULL DEFAULT 'unknown'
);
CREATE TABLE pizza (
  name text
    check (name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
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

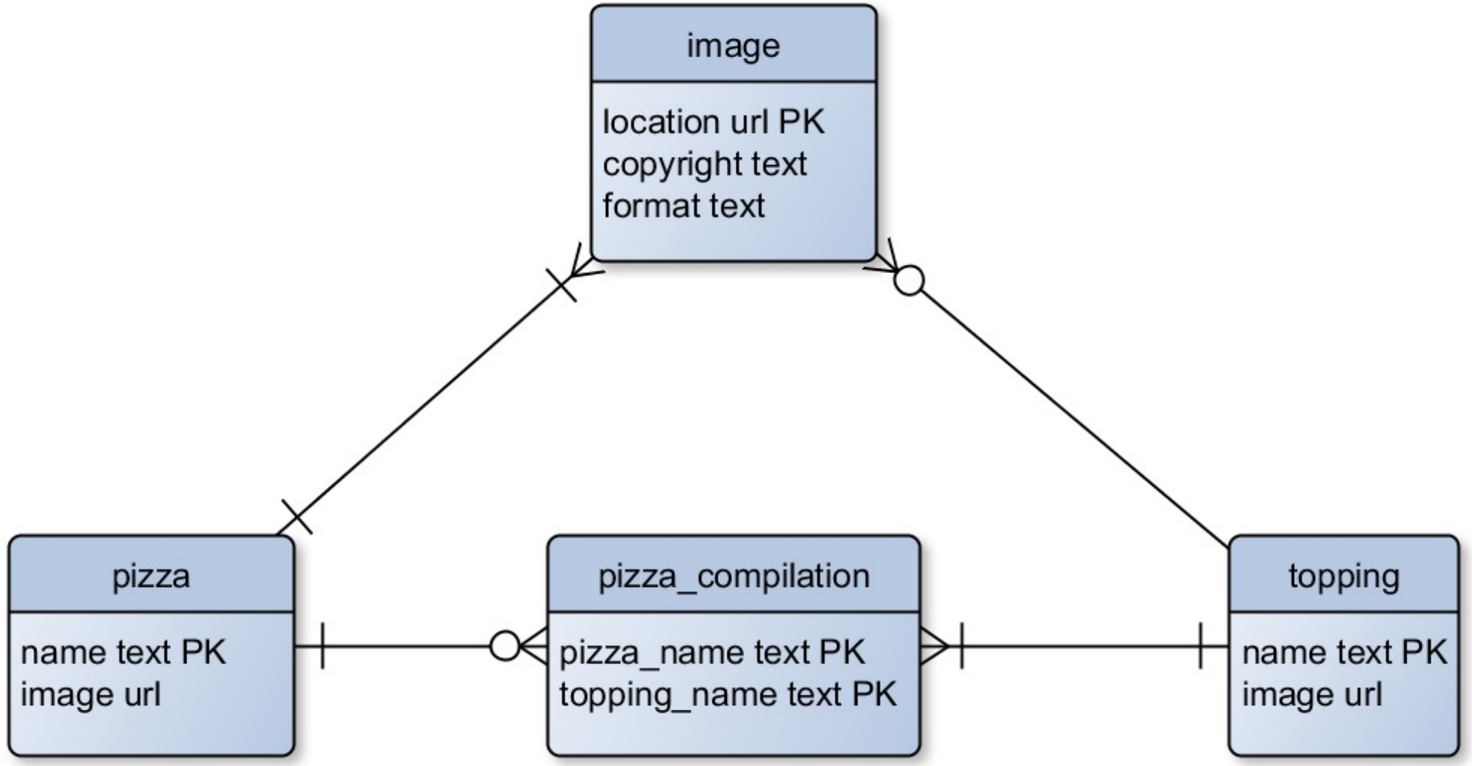
```
CREATE TABLE image (
  location text PRIMARY KEY,
  name text
    NOT NULL
    DEFAULT 'placeholder',
  copyright text
    NOT NULL DEFAULT 'unknown',
  type text
    NOT NULL DEFAULT 'unknown'
);
CREATE TABLE pizza (
  name text
    check (name != ''::text)
    PRIMARY KEY,
  img text NOT NULL
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

Schoene Woche noch :)

# Material fuer Vorlesung 6

folgende slides nur falls ueberhaupt Zeit

# Referenzen:


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