Modul Datenbanken

Vorlesung 8

Von Anfragen zu Abfragen

IFI Wintersemester 2016/17

by Renzo Kottmann

Beim letzten Mal besprochen

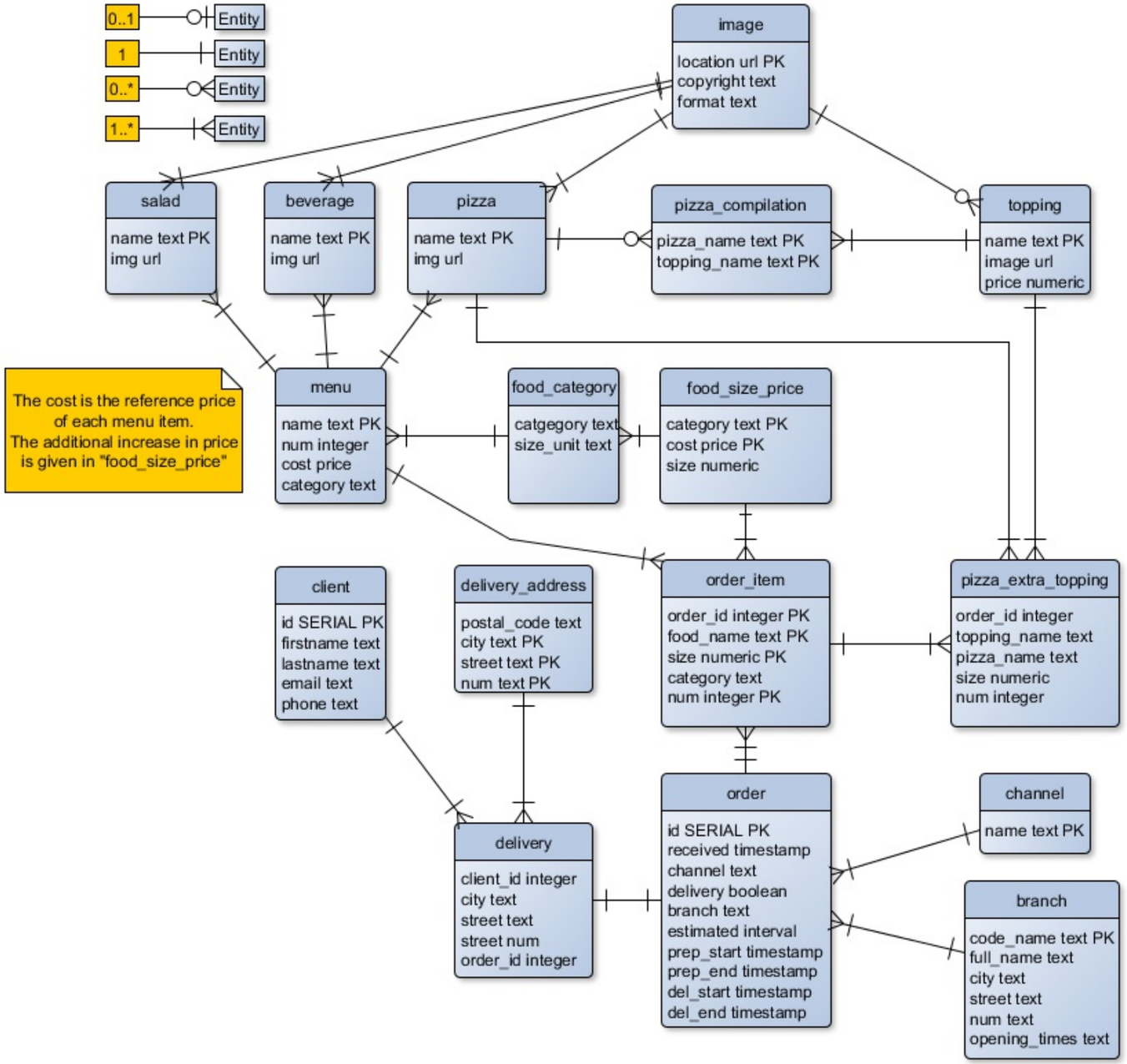
- SQL Implementierung von m-n Entitaets-Beziehungen
- Erweiterung des Datenbankmodells um Bestellungen
  - Schoene Gruppenarbeitsloesung
  - Musterloesung

Beim letzten Mal NICHT fertig besprochen

- SQL Abfragen mit SELECT

Beim letzten GAR NICHT besprochen

- Vertiefung Wertebereiche


0..1
Entity
1
Entity
0..*
Entity
1..*
Entity
image
location url PK
copyright text
format text
salad
name text PK
img url
beverage
name text PK
img url
pizza
name text PK
img url
pizza_compilation
pizza_name text PK
topping_name text PK
topping
name text PK
image url
price numeric
The cost is the reference price of each menu item.
The additional increase in price is given in "food_size_price"
menu
name text PK
num integer
cost price
category text
food_category
catgegory text
size_unit text
food_size_price
category text PK
cost price PK
size numeric
client
id SERIAL PK
firstname text
lastname text
email text
phone text
delivery_address
postal_code text
city text PK
street text PK
num text PK
order_item
order_id integer PK
food_name text PK
size numeric PK
category text
num integer PK
pizza_extra_topping
order_id integer
topping_name text
pizza_name text
size numeric
num integer
order
id SERIAL PK
received timestamp
channel text
delivery boolean
branch text
estimated interval
prep_start timestamp
prep_end timestamp
del_start timestamp
del_end timestamp
channel
name text PK
delivery
client_id integer
city text
street text
street num
order_id integer
branch
code_name text PK
full_name text
city text
street text
num text
opening_times text

Aufgaben:

1. Verschaffe Dir erneut einen Ueberblick ueber das ER-Diagram

2. Erzeuge eine Datenbank mit folgendem SQL Code:

   Download-Link

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}`

   Zukuenftige Veraenderbarkeit manchmal nur schwer vorauszusehen!

2. **Code-List**: Liste von Werten, die in Zukunft haeufig und zu jeder Zeit geaendert wird

Enumeration

Implementierung mit:

1. `check` constraint:

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

2. Enumerated Types:

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

Enumeration

Implementierung mit: `CREATE DOMAIN`

- Definition von eigenen Datentypen basierend of existierenden

```
CREATE DOMAIN gender AS text CHECK (
  VALUE ~ '^maennlich|weiblich$'
);
```

SQL Kommando: `CREATE DOMAIN`

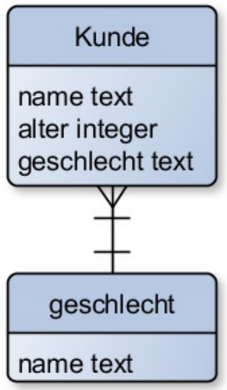
- Allgemein verwendet fuer wiederverwendbare
  - `CHECK` constraints
  - DEFAULTS
  - und `NOT NULL`
- Weiteres Beispiel:

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

Code-List

Implementierung mit Hilfe einer Tabelle

Beispiel Kunde




```
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');
```

Von Anfragen zu Abfragen

Revisited:

Gruende fuer relationale Datenbanken:

1. Persistente, sichere und strukturierte Datenspeicherung
2. Effiziente Anfragen!

   z.B.: Wieviel Umsatz machte die Pizzeria am 15.10.2015?

Revisited:

Gruende fuer relationale Datenbanken:

1. Persistente, sichere und strukturierte Datenspeicherung
2. Effiziente Anfragen!

   z.B.: Wieviel Umsatz machte die Pizzeria am 15.10.2015?

- SQL hat nur einen einzigen 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

```
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.

Von Anfragen zu Abfragen

ANFRAGE:

Zeig mir alle Preise pro Groesse!

ANFRAGE:

Zeig mir alle Preise pro Groesse!

ABFRAGE:

```
SELECT *
  FROM food_size_price;
```

Wie teuer ist ein Getränk der Grösse 0.3L ?

```
SELECT *
  FROM food_size_price
 WHERE size = 0.3;
```

Wie teuer ist ein Getränk der Grösse 0.3L ?

Explizite Nennung der gewuenschten Spalten

```
SELECT category, cost, size
  FROM food_size_price
 WHERE size = 0.3;
```

Wie teuer ist ein Getränk der Grösse 0.3L ?

Umbennung der gewuenschten Spalten und Tabelle

```
SELECT category AS art, cost AS preis, size as groesse
  FROM food_size_price
 WHERE size = 0.3;
```

Welche Getraenke sind groesser als 0.3L ?

```
SELECT category AS art, cost AS preis, size as groesse
  FROM food_size_price
 WHERE size > 0.3;
```

Welche Getraenke sind groesser als 0.3L ?

Besssere, da praezisere Abfrage:

```
SELECT category AS art, cost AS preis, size as groesse
  FROM food_size_price
 WHERE size > 0.3
       AND
       category = 'beverage';
```

Zeig mir alle Pizza-Groessen!

```
SELECT category AS art, cost AS preis, size as groesse
  FROM food_size_price
 WHERE category = 'Pizza';
```

Zeige mir alle Getraenke-Groessen mit Masseinheit!

```
SELECT size || 'L' AS groesse_einheit
  FROM food_size_price
 WHERE category = 'beverage';
```

Zeig mir das Menu

```
SELECT *
  FROM menu;
```

Zeig mir das Menu geordnet nach Nummer!

```
SELECT *
  FROM menu
  ORDER BY num;
```

Zeig mir das Menu geordnet nach Nummer!

```
SELECT *
  FROM menu
  ORDER BY num;
```

und umgedrehte Reihenfolger

```
SELECT *
  FROM menu
  ORDER BY num DESC;
```

Wieviele Eintraege hat das Menu?

```
SELECT count(*)        -- Aggregat Funktion: Zaehle Zeilen
  FROM menu
  ORDER BY num DESC;
```

Aggregat Funktionen berechnen einen einzigen Wert aus einer Menge von Werten.

Es gibt einige Funktionen u.a. auch `min`, `max` und `sum`.

Abfragen ueber mehere Tabellen

Revisited: 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.
```

Revisited: 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.
```

Es wird immer eine und nur eine Tabelle durch `SELECT` erzeugt, daher ist das technisch praeziser:

```
Erzeuge und zeig mir eine virtuelle Tabelle, die folgender Anweisung enpricht:
Zeige alle Spalten der Tabelle "tabelle" an und davon alle Zeilen.
```

Beispiel-Tabellen

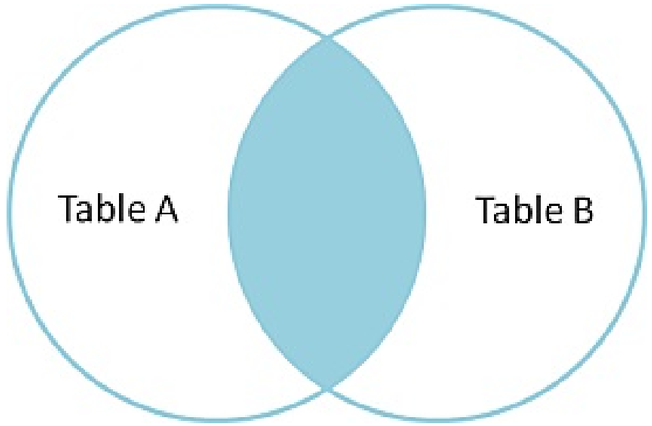
```
TabelleA                TabelleB

id | name                 id | name
---+----------            ---+------------
1  | Pirate               1  | Rutabaga
2  | Monkey               2  | Pirate
3  | Ninja                3  | Darth Vader
4  | Spaghetti            4  | Ninja
```

INNER JOIN

```
SELECT *
  FROM TabelleA AS a
       INNER JOIN
       TabelleB as b
    ON a.name = b.name
```

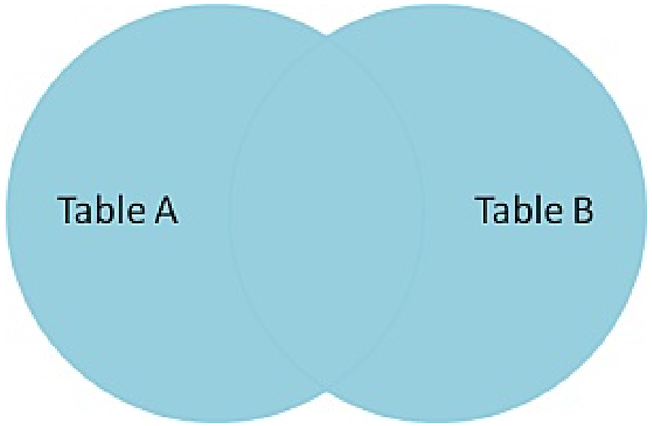
```
id  name       id   name
--  ----       --   ----
1   Pirate     2    Pirate
3   Ninja      4    Ninja
```



FULL OUTER JOIN

```
SELECT *
  FROM TabelleA as a
       FULL OUTER JOIN
       TabelleB as b
    ON a.name = b.name
```

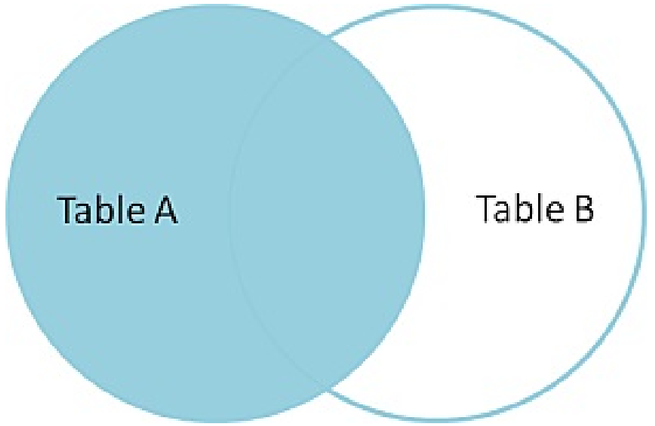
```
id    name       id    name
--    ----       --    ----
1     Pirate     2     Pirate
2     Monkey     null  null
3     Ninja      4     Ninja
4     Spaghetti  null  null
null  null       1     Rutabaga
null  null       3     Darth Vader
```



LEFT OUTER JOIN

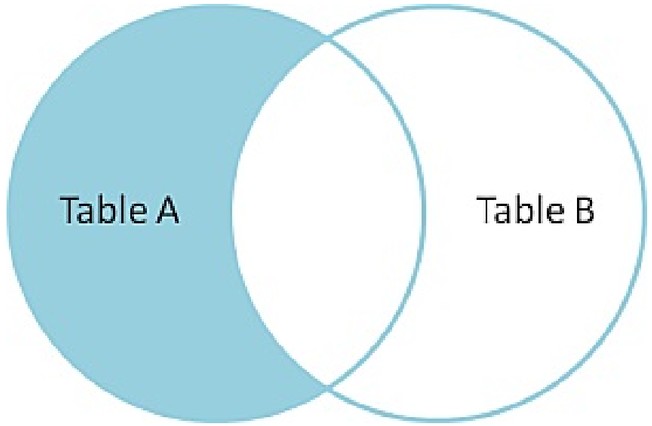
```sql
SELECT *
  FROM TabelleA AS a
       LEFT OUTER JOIN
       TabelleB AS b
    ON a.name = b.name
```

```
id  name       id    name
--  ----       --    ----
1   Pirate     2     Pirate
2   Monkey     null  null
3   Ninja      4     Ninja
4   Spaghetti  null  null
```



LEFT OUTER JOIN: nur Zeilen von TabelleA

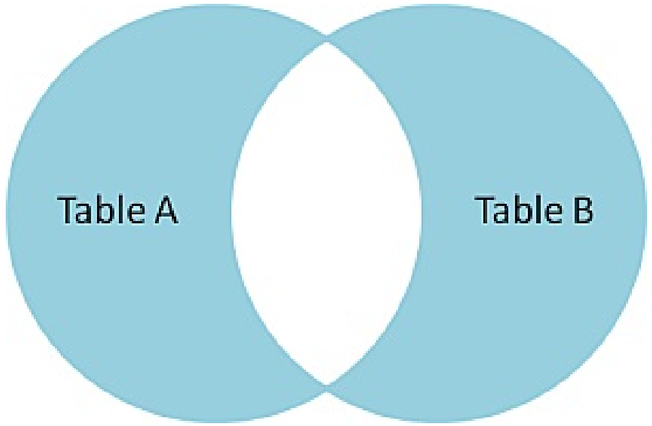
```
SELECT *
  FROM TabelleA AS a
       LEFT OUTER JOIN
       TabelleB AS b
    ON a.name = b.name
 WHERE b.id IS null
```




```
id  name       id    name
--  ----       --    ----
2   Monkey     null  null
4   Spaghetti  null  null
```

FULL OUTER JOIN: nur exklusive Zeilen

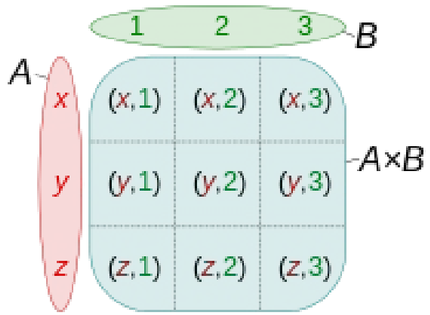
```
SELECT *
  FROM TabelleA AS a
       FULL OUTER JOIN
       TabelleB AS b
    ON a.name = b.name
 WHERE a.id IS null
       OR
       b.id IS null
```



id	name	id	name
2	Monkey	null	null
4	Spaghetti	null	null
null	null	1	Rutabaga
null	null	3	Darth Vader

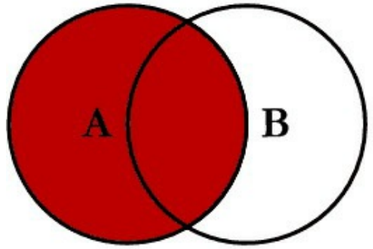
CROSS JOIN: Erzeugt kartesiches Produkt

```
SELECT *
  FROM TabelleA
       CROSS JOIN
       TabelleB
```

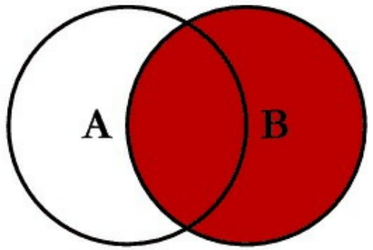


In einfachen Worten: Verbindet jede Zeile der TabelleA mit jeder Zeile der TabelleB

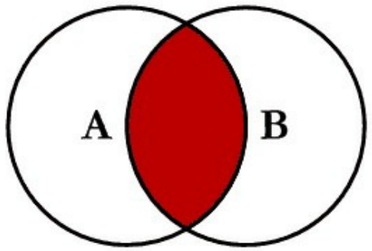
SQL JOINS



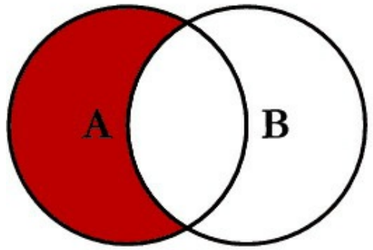

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



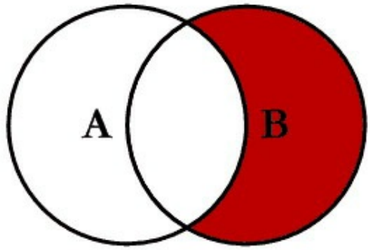

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



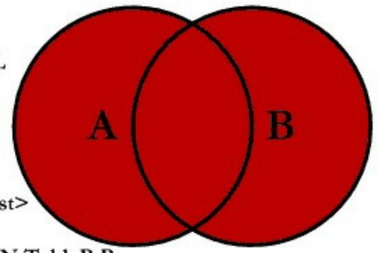

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



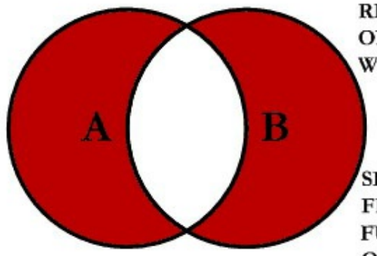

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Diskussion zu Visualisierung von SQL-joins

- Original Artikel mit Venn-Overview
- Diskussion und alternative Darstellung
- Kategorisierung nach JOIN-Typen

Referenzen:

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