

# Informatik 10 - Datenbanken (Teil 1)

aktualisiert: 20. Okt. 2025

## Stunde 1+2

Wdh: Klassen und Objekte  
Objektkarten Memory

## Stunde 3+4

Wdh: Von der Klasse zur Tabelle  
Wdh: Aufbau von (relationalen) Datenbanken  
SQL Spickzettel  
Übung: SQL Island

## Stunde 5+6

SQL Puzzle  
Wdh: SQL Basics

## Stunde 7+8

Tabellenbeziehungen

Tabellenbeziehungen: Fremdschlüssel

## Stunde 9+10

Tabellenbeziehungen im Klassendiagramm  
Kardinalitäten



Klassendiagramm Flugverspätung  
SQL: Tabellen verbinden  
Kreuzprodukt / Join

## Stunde 11+12

Join Beispiel



SQL mit Kreuzprodukt und Join

# Outline

Stunde 1+2

Stunde 3+4

Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12

## Wdh: Klassen und Objekte

Objekte repräsentieren Gegenstände in einem Computerprogramm. Klassen sind der Bauplan, der festlegt, welche Eigenschaften ( Attribute ) und Fähigkeiten ( Methoden ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

Klassenkarte

```
Person
String hobby
int alter
boolean hatHaustier
String peinlichesErlebnis
void atmen()
```

spitze Ecken

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

Objektkarte

p1 : Person

```
hobby = "Klettern"
alter = 23
hatHaustier = false
peinlichesErlebnis =
"..."
```

runde Ecken

## Objektkarten Memory

- Erstelle auf einem Blatt eine Objektkarte der Klasse Person zu dir selbst. → 3x falten
- Gib deine Objektkarte bei der Lehrkraft ab. → Objektkarten werden gemischt
- Ziehe eine Objektkarte und versuche, das zugehörige Objekt zu finden.
  - Frage deine:n Gegenüber dafür, ob die Attributwerte auf deiner gezogenen Karte auf sie/ihn zutreffen.
  - Ihr darf euch nicht gegenseitig die Objektkarten zeigen!
  - Wer gefunden wurde, gibt seine aktuelle Objektkarte weiter und setzt sich.
  - Der/Die Finder:in sammelt alle gefundenen Objekte.



repräsentieren **Gegenstände** in einem Computerprogramm.  
der festlegt, welche **Eigenschaften** ( ) und **Fähigkeiten** ( )  
Objektart gespeichert werden sollen. Man stellt sie dar mit:

sind der **Bauplan**,  
) einer bestimmten

**Klassenkarte**

**Objektkarte**





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm.  
der festlegt, welche **Eigenschaften** ( ) und **Fähigkeiten** ( )  
Objektart gespeichert werden sollen. Man stellt sie dar mit:

sind der **Bauplan**,  
) einer bestimmten

**Klassenkarte**

**Objektkarte**





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( ) und **Fähigkeiten** ( ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

**Klassenkarte**

**Objektkarte**





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

**Klassenkarte**

**Objektkarte**





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

**Klassenkarte**

**Objektkarte**





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

## Klassenkarte

Person

String hobby

int alter

boolean hatHaustier

String peinlichesEr-

lebnis

void atmen()

## Objektkarte





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

## Klassenkarte

Person

String hobby

int alter

boolean hatHaustier

String peinlichesEr-

lebnis

void atmen()

## Objektkarte





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

## Klassenkarte

Person

String hobby

int alter

boolean hatHaustier

String peinlichesEr-

lebnis

void atmen()

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

## Objektkarte





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

**Klassenkarte**

Person
String hobby
int alter
boolean hatHaustier
String peinlichesErlebnis
void atmen()

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

**Objektkarte**

p1 : Person
hobby = "Klettern"
alter = 23
hatHaustier = false
peinlichesErlebnis =
"..."





**Objekte** repräsentieren **Gegenstände** in einem Computerprogramm. **Klassen** sind der **Bauplan**, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:

**Klassenkarte**

```
Person
String hobby
int alter
boolean hatHaustier
String peinlichesEr-
lebnis
void atmen()
```

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

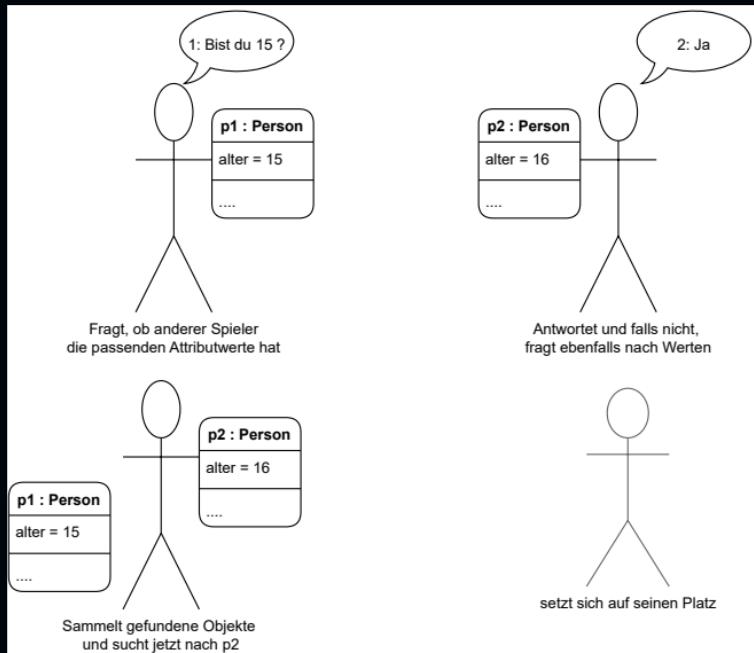
**Objektkarte**

```
p1 : Person
hobby = "Klettern"
alter = 23
hatHaustier = false
peinlichesErlebnis =
    ...
    
```

# Objektkarten Memory



- Erstelle auf einem Blatt eine Objektkarte der Klasse Person zu dir selbst. → **3x falten**
- Gib deine Objektkarte bei der Lehrkraft ab. → Objektkarten werden gemischt.
- Ziehe eine Objektkarte und versuche, das zugehörige Objekt zu finden.
  - Frage deine:n Gegenüber dafür, ob die Attributwerte auf deiner gezogenen Karte auf sie/ihn zutreffen.
  - Ihr dürft euch nicht gegenseitig die Objektkarten zeigen!
  - Wer gefunden wurde, gibt seine aktuelle Objektkarte weiter und setzt sich.
  - Der/Die Finder:in sammelt alle gefundenen Objekte.



# Outline

Stunde 1+2

Stunde 3+4

Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12

## Wdh: Klassen und Objekte

**Objekte** repräsentieren Gegenstände in einem Computerprogramm. **Klassen** sind der Bauplan, der festlegt, welche **Eigenschaften** ( **Attribute** ) und **Fähigkeiten** ( **Methoden** ) einer bestimmten Objektart gespeichert werden sollen. Man stellt sie dar mit:



## Objektkarten Memory

- Erstelle auf einem Blatt eine Objektkarte der Klasse Person zu dir selbst. → 3x falten
- Gib deine Objektkarte bei der Lehrkraft ab. → Objektkarten werden gemischt
- Ziehe eine Objektkarte und versuche, das zugehörige Objekt zu finden.
  - Frage deinen: Gegenüber dafür, ob die Attributwerte auf deiner gezogenen Karte auf sie/ihm treffen.
  - Ihr dürft euch nicht gegenseitig die Objektkarten zeigen!
  - Wer gefunden wurde, gibt seine aktuelle Objektkarte weiter und setzt sich.
  - Der/Die Finder:in sammelt alle gefundenen Objekte.

## Wdh: Von der Klasse zur Tabelle

Zeichnet zu zweit eine Tabelle, der man alle Objekte der Klasse Person sammeln kann.

- Zeichnet eine Tabelle mit den Attributen der Klasse Person.
- Zeichnet eine Tabelle mit den Attributen der Klasse Person.
- Ordnet die folgenden Begriffe den Teilen der Tabelle zu. Achte auf: Nicht alle Begriffe passen und manches hat mehrere Bedeutungen! Datentyp: Tabelle, Zeile, Spalte, Objekt, Parameter, Attribut, Spezialisierung, Konkretisierung, Abstraktion, Spezialisierung, Konkretisierung, Abstraktion

Lösung:  
Nicht verwendete Begriffe: Parameter, Methode, Board, Datentyp  
Viele Wörter mit gleicher Bedeutung verweisen, v.a. in Programmen ein konkreter Name oder ein Access.

## Wdh: Aufbau von (relationalen) Datenbanken

Datenbanken speichern Datensätze in Tabellen. Die Spaltenüberschriften repräsentieren **Attributnamen**. Ein Datensatz besteht aus den Werten für alle Attributnamen. Ein Objekt wird in den Spalten durch die Attributwerte. Jede Tabelle hat einen **Primärschlüssel** (oft auch „PK“), der Datensätze eindeutig identifiziert. Wenn die Datensatzmerkmale einfach durchnummierbar sind, kann man diese Merkmale als Primärschlüssel verwenden. Der Aufbau einer Tabelle kann mit **Klassenkarte** oder **Tabellenkarte** dargestellt werden. Durch das Zeichen „TABELLENKARTE“ ist der Primärschlüssel (Datensatz Spalte1, Datensatz Spalte2, ...). Zum Beispiel:  
**Person**(pk, String name, int alter, ...)

## SQL Spickzettel

Folgender SQL-Spickzettel enthält alle SQL-Grunderlagen der 9. Klasse. Bei click (qualif.) ihm bei allen SQL-Aufgaben benutzen. Über das Vorlesegesymbol können ihr dem Spickzettel als eigenes PDF öffnen.

## Übung: SQL Island

sql-island.informatik.uni-kl.de/
1. Was sind die Primärschlüsse der Tabellen, die die einzelnen Objekte eindeutig identifizieren? – Normalisierung der Datenbanken
2. Schafft die Tabellen der Datenbank mit Klassenkarten das:

PERSON	ARTIST	ALBUM
int bestandnr String name int alter boolean hatHaustier	String gegebenName String geburtsdatum String name int bestandnr	int alter String name int bestandnr

SONG
String titel String artist String genre

## Wdh: Von der Klasse zur Tabelle



- Zeichnet zu zweit eine Tabelle, in der man alle Objekte der Klasse Person sammeln kann.
- Tragt eure beiden Objekte (vom Objektkarten-Memory) in die Tabelle ein.
- Ordnet die folgenden Begriffe den Teilen der Tabelle zu.

Achtung: Nicht alle Begriffe passen und manches hat mehrere Begriffe!

Datensatz Tabelle Zelle Klasse Objekt Parameter Attribut Spalte Feld Methode Board Zeile Datentyp Attributwert

# Wdh: Von der Klasse zur Tabelle



- Zeichnet zu zweit eine Tabelle, in der man alle Objekte der Klasse Person sammeln kann.
- Tragt eure beiden Objekte (vom Objektkarten-Memory) in die Tabelle ein.
- Ordnet die folgenden Begriffe den Teilen der Tabelle zu.  
Achtung: Nicht alle Begriffe passen und manches hat mehrere Begriffe!

Datensatz Tabelle Zelle Klasse Objekt Parameter Attribut Spalte Feld Methode Board Zeile Datentyp Attributwert

Lösung:

Attribut/ Feld/  
Spaltenname

Tabelle					
name	alter	groesse	geschlecht	brille	...
Herrmann	24	1.62	m	false	...
Zelle/ Attributwert		...	...	...	...

Klasse/ Spaltennamen  
Datensatz/Zeile/  
Objekt

Nicht verwendete Begriffe: Parameter, Methode, Board, Datentyp

Feld: Wird oft synonym zu Attribut verwendet, v.a. in Programmen wie LibreOffice Base oder MS Access.

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in . Die repräsentieren die  
**Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die (=Zeilen) entsprechen  
**Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen (oft  
**auch „ID“**), der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummeriert.  
Im Tabellenschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet.  
Der Aufbau einer Tabelle kann mit oder dargestellt werden.

Dessen Aufbau ist:

TABELLENNAME(Datentyp Primärschlüssel , Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Objekte** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **auch „ID“**, der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummeriert. Im Tabellenschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet. Der Aufbau einer Tabelle kann mit oder dargestellt werden.

Dessen Aufbau ist:

TABELLENNAME(Datentyp Primärschlüssel , Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Spaltenüberschriften** repräsentieren die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Zeilen** (=Zeilen) entsprechen **Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **ID**, der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummeriert. Im Tabellenschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet. Der Aufbau einer Tabelle kann mit oder dargestellt werden.

Dessen Aufbau ist:

TABELLENNAME(Datentyp Primärschlüssel , Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Spaltenüberschriften** repräsentieren die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Datensätze** (=Zeilen) entsprechen **Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **ID**, der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummieriert. Im Tabellschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet. Der Aufbau einer Tabelle kann mit **SQL** oder **ER-Diagramm** dargestellt werden.

Dessen Aufbau ist:

TABELLENNNAME(Datentyp Primärschlüssel , Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Spaltenüberschriften** repräsentieren die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Datensätze** (=Zeilen) entsprechen **Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **Primärschlüssel** (oft auch „**ID**“), der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummiert. Im Tabellschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet.

Der Aufbau einer Tabelle kann mit oder dargestellt werden.

Dessen Aufbau ist:

TABELLENNNAME(Datentyp Primärschlüssel , Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Spaltenüberschriften** repräsentieren die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Datensätze** (=Zeilen) entsprechen **Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **Primärschlüssel** (oft auch „ID“), der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummeriert. Im Tabellenschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet.

Der Aufbau einer Tabelle kann mit **Klassenkarte** oder dargestellt werden.

Dessen Aufbau ist:

TABELLENNAME(Datentyp Primärschlüssel, Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Spaltenüberschriften** repräsentieren die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Datensätze** (=Zeilen) entsprechen **Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **Primärschlüssel** (oft auch „ID“), der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummeriert. Im Tabellenschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet.

Der Aufbau einer Tabelle kann mit **Klassenkarte** oder **Tabellenschema** dargestellt werden.

Dessen Aufbau ist:

TABELLENNAME(Datentyp Primärschlüssel, Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

## Wdh: Aufbau von (relationalen) Datenbanken



Datenbanken speichern Datensätze in **Tabellen**. Die **Spaltenüberschriften** repräsentieren die **Attribute** (Synonym: Feld) und bilden zusammen eine **Klasse**. Die **Datensätze** (=Zeilen) entsprechen **Objekten** und in den Spalten stehen die Attributwerte. Jede Tabelle hat einen **Primärschlüssel** (oft auch „ID“), der Datensätze eindeutig identifiziert. Oft werden die Datensätze hiermit einfach durchnummeriert. Im Tabellenschema wird er unterstrichen und im Klassendiagramm immer als erstes Attribut aufgelistet.

Der Aufbau einer Tabelle kann mit **Klassenkarte** oder **Tabellenschema** dargestellt werden.

Dessen Aufbau ist:

TABELLENNAME(Datentyp Primärschlüssel , Datentyp Spalte1, Datentyp Spalte2, ...)

Zum Beispiel:

`Person(int id, String name, int alter, ...)`



Folgender SQL-Spickzettel enthält alle SQL-Grundlagen der 9. Klasse. Ihr dürft (sollt!) ihn bei allen SQL-Aufgaben benutzen. Über das Vorlagensymbol  oben könnt ihr den Spickzettel als eigenes PDF öffnen.

Inf 9 Grundlagen	<b>Spickzettel SQL</b>	
---------------------	------------------------	---

<b>SELECT</b>	Spaltenliste	<ul style="list-style-type: none"> <li>- DISTINCT vermeidet Duplikate.</li> <li>- Aggregatfunktionen (COUNT, SUM, MAX, MIN, AVG) für Berechnungen</li> <li>- SELECT * für "alle Spalten"</li> <li>- AS Aliasname</li> </ul>
<b>FROM</b>	Tabelle	
<b>WHERE</b>	Bedingung	<ul style="list-style-type: none"> <li>- Wird meist mit Vergleichen (<code>&lt;</code>, <code>&lt;=</code>, <code>=</code>, <code>&gt;</code>, <code>&gt;=</code>, ... ) formuliert.</li> <li>- Verknüpfung von mehreren Vergleichen mit logischen Funktionen (AND, OR, NOT)</li> </ul>
<b>GROUP BY</b>	Spaltenliste	
<b>HAVING</b>	Bedingung	
<b>ORDER BY</b>	Spaltenliste	<ul style="list-style-type: none"> <li>- ASC für aufsteigend (Standard)</li> <li>- DESC für absteigend</li> </ul>

Im Detail gilt:

#### Grundlegende SQL-Abfrage

SELECT	Es muss mindestens ein Spaltenname angegeben werden. Die entsprechende(n) Spalte(n) sind dann Teil der Ergebnistabelle. SELECT * bewirkt, dass alle Spalten angezeigt werden.
DISTINCT	Duplikate von Datensätzen werden nicht angezeigt.
AS	Eine Spalte in der Ergebnistabelle kann anders benannt werden als in der Ausgangstabelle. Dies ist vor allem bei der Verwendung von Aggregatfunktionen hilfreich.
FROM	Hier muss angegeben werden, aus welcher Tabelle die Informationen für die Abfrage genommen werden sollen.
ORDER BY	Die Ergebnistabelle wird nach der oder den angegebenen Spalten sortiert. Standardmäßig wird aufsteigend sortiert. Mit dem Zusatz DESC bzw. ASC wird absteigend bzw. aufsteigend sortiert.
Beispiele	<pre>SELECT DISTINCT kontinent AS "enthaltene Kontinente" FROM Land  SELECT name, flaeche, hauptstadt FROM LAND ORDER BY flaeche DESC</pre>

#### Auswahl von Datensätzen über Bedingungen

WHERE	In der Ergebnistabelle werden nur die Datensätze (Zeilen) angezeigt, welche die angegebene Bedingung erfüllen. Eine Bedingung wird mit einem Vergleich formuliert. Neben den typischen Vergleichsoperatoren wie <code>&lt;</code> , <code>&lt;=</code> , <code>=</code> , <code>&gt;</code> , <code>&gt;=</code> , usw. sind insbesondere auch IS NULL und LIKE wichtig. Mehrere Vergleiche können durch die logischen
-------	--

	Funktionen AND, OR und NOT verknüpft werden. Ggf. müssen die einzelnen Ausdrücke dabei sinnvoll geklammert werden
Beispiel	
WHERE	jahr > 2015
AND	laufzeit > 90
AND	NOT fsk = 18
LIKE	Kann in einer Bedingung zur Mustererkennung von Einträgen verwendet werden. Einzelne Platzhalter (wildcards) werden häufig eingesetzt:
	<ul style="list-style-type: none"> <li>• % steht für beliebig viele Zeichen, auch keines (* bei MS Access)</li> <li>• _ für genau ein beliebiges Zeichen (? bei MS Access)</li> </ul>
Beispiel	
WHERE	titel LIKE "You%" – findet alle Titel die mit "You" beginnen
	Groß-/Kleinunterschreitung wird nicht berücksichtigt
WHERE	titel LIKE "%love%" – findet alle Titel die "love" enthalten
WHERE	titel LIKE "l___" – findet alle Titel die mit l beginnen und genau 4 Zeichen lang sind
NULL	Bedeutet, dass kein Wert in einer Zelle eingetragen ist.
IS NULL	Überprüft (in einer Bedingung), ob kein Wert in einer Zelle eingetragen ist.

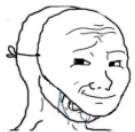
#### Aggregatfunktionen

AVG	Berechnet den Durchschnitt aller Werte einer Spalte.
COUNT	Gibt die Anzahl der Einträge einer Spalte aus.
MAX bzw. MIN	Gibt das Maximum bzw. Minimum aller Werte einer Spalte aus.
SUM	Berechnet die Summe aller Werte einer Spalte.
Beispiel	<pre>SELECT COUNT(*) AS "Anzahl afrikanischer Länder" FROM Land WHERE kontinent = "Afrika"</pre>

#### Gruppierung

GROUP BY	Datensätze mit demselben Wert in der angegeben Spalte werden gruppiert. Gruppierungen sind nur in Kombination mit Aggregatfunktionen sinnvoll.
HAVING	An gruppierte Datensätze werden Bedingungen mit HAVING formuliert.
Beispiel	<pre>SELECT fsk, MIN(laufzeit) FROM Film WHERE genre1="Filmkomödie" OR genre2="Filmkomödie" GROUP BY fsk HAVING fsk &lt; 6</pre>

SQL keywords should be  
in **lower case!**



```
select name, id  
from products  
where discount = 0  
order by price asc;
```

Noooo, they must be  
in **upper case!**

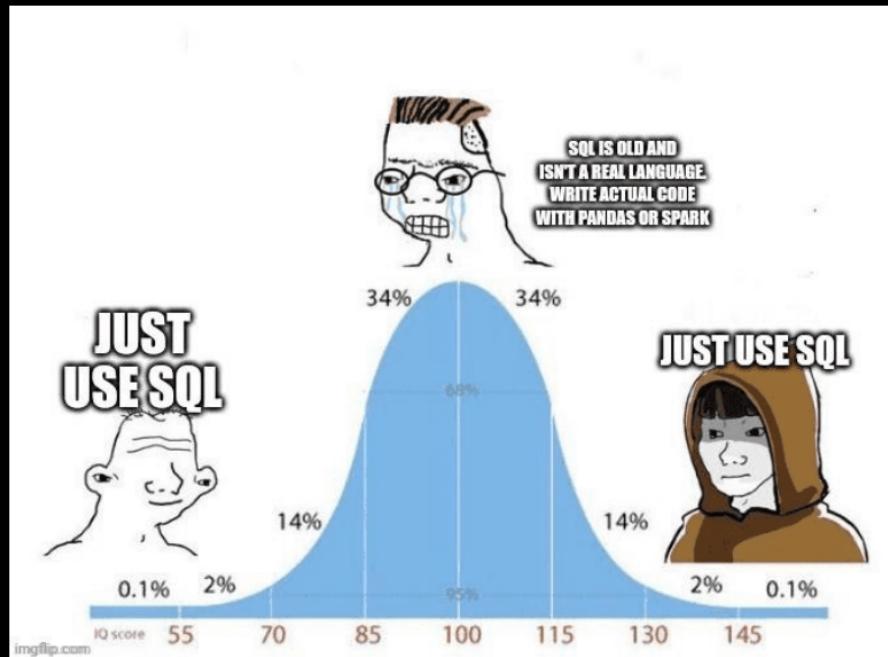


```
SELECT name, id  
FROM products  
WHERE discount = 0  
ORDER BY price ASC;
```



```
sElEcT nAmE, iD  
fRoM PrOdUcTs  
WhErE dIsCoUnT = 0  
OrDeR bY pRiCe Asc;
```

'Sarcastic Query Language' \* by u/casperdewith



# Übung: SQL Island



`sql-island.informatik.uni-k1.de/`

1. Was sind die Primärschlüssel der Tabellen, die die einzelnen Objekte eindeutig identifizieren?  
→ Notiert das vollständige Tabellenschema der Datenbank von SQL Island (mit Datentypen und Markierung der Primärschlüssel)
  
2. Stellt die Tabellen der Datenbank mit Klassenkarten dar.

# Übung: SQL Island



sql-island.informatik.uni-k1.de/

1. Was sind die Primärschlüssel der Tabellen, die die einzelnen Objekte eindeutig identifizieren?  
→ Notiert das vollständige Tabellenschema der Datenbank von SQL Island (mit Datentypen und Markierung der Primärschlüssel)  
`BEWOHNER(int bewohnernr , String name, int dorfnr, String geschlecht, String beruf, int gold, String status)`  
`GEGENSTAND(String gegenstand, int besitzer)`  
`DORF(int dorfnr, String name, int haeuptling)`
2. Stellt die Tabellen der Datenbank mit Klassenkarten dar.

# Übung: SQL Island



sql-island.informatik.uni-k1.de/

- Was sind die Primärschlüssel der Tabellen, die die einzelnen Objekte eindeutig identifizieren?  
→ Notiert das vollständige Tabellenschema der Datenbank von SQL Island (mit Datentypen und Markierung der Primärschlüssel)

BEWOHNER(int bewohnernr , String name, int dorfnr, String geschlecht, String beruf, int gold, String status)

GEGENSTAND(String gegenstand, int besitzer)

DORF(int dorfnr, String name, int haeuptling)

- Stellt die Tabellen der Datenbank mit Klassenkarten dar.

BEWOHNER
int bewohnernr
String name
int dorfnr
String geschlecht
String beruf
int gold
String status

GEGENSTAND
String gegenstand
int besitzer

DORF
int dorfnr
String name
int haeuptling

## Für Schnelle



**Für Schnelle:** Spielt SQL Island, der SQL-Spickzettel hilft euch dabei.

# Outline

Stunde 1+2

Stunde 3+4

Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12

Wdh: Von der Klasse zur Tabelle	
Zeichnet zu zweit eine Tabelle, in der man die Objekte der Klasse Person zusammen kann.	
Tragt neue bekannte Objekte (vom Objektorientierten Memory) ins Tabellenschema ein.	
Ordnet die folgenden Begriffe den Teilen der Tabelle zu: Achtung: Nicht alle Begriffe passen und manches hat mehrere Begriff! Datenwerte Tabella Zelle Klasse Objekt Parameter Attribut Spalt Name Metzger Board Seite Dienstweg Ambulanz	
Lösung:	
Nicht verwendbare Begriffe: Parameter, Methode, Board, Datentyp	
Passend sind als sprunge als Attribut verwendbar, v. u. in Programmen wie Microsoft Excel oder BI Anwend.	

Wdh: Aufbau von relationalen Datenbanken	
Datenbanken speichern Daten in	<b>TABellen</b> . Die Spaltenüberschriften representieren dabei <b>Attributnamen</b> . Feldinhalte sind eine <b>Classe</b> der <b>Datenwerte</b> ( <b>Attributwerte</b> ) und den Tabellen stehen die Attributwerte. Jede Tabelle hat einen <b>Primärschlüssel</b> (ohne „PK“), der Datensätze eindeutig identifiziert. Ohne werden die Datensätze hiermit einfach durchnummieriert.
Der Aufbau einer Tabelle kann mit	<b>Klassenskript</b> oder <b>Tabellenschem</b> dargestellt werden.
Zum Beispiel:	<b>PERSONEN(id, String name, int alter, ..)</b>

**SQL Spickzettel**

Folgender SQL-Spickzettel enthält alle SQL-Grundlagen der 9. Klasse. Ihr könnt (sollt) ihn bei allen SQL-Aufgaben benutzen. Über das Vorlagensymbol  oben könnet ihr den Spickzettel als eigenes PDF öffnen.

SQL Puzzle				
In dieser Aufgabe geht es immer um die Tabelle land, deren erste Datensätze du hier siehst:				
id	name	einwohner	flaeche	hauptstadt
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

**Lösung:**

1) iv)	4) i)	7) v)
2) viii)	5) ix)	8) ii)
3) vii)	6) iii)	9) vi)

**Wdh: SQL Basics**

Bearbeite die Aufgabe **Wdh - SQL Basics** auf [artemis.tum.de](http://artemis.tum.de). Artemis gibt dir immer, wenn du auf Submit drückst, die ersten Zeilen der Ergebnistabelle und ob deine SQL-Abfrage (bzw. welche Teile von ihr) richtig sind, aus.

Wenn du eine Abfrage richtig hast, notiere sie unten im Skript.

Falls du bei Gruppierung und Aggregatfunktionen Schwierigkeiten hast, hilft dir dieses [Video](#) (bitte Kopfhörer verwenden!): [bycs.link/simpleclub-group-sort-aggregat](https://bycs.link/simpleclub-group-sort-aggregat)

1) Vervollständige die SQL-Abfrage so, dass sie ID, Name, Art und URL aller Freibäder ausgibt.    `SELECT id, name, art, url FROM Schwimmbad WHERE art='Freibad'`

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- 1) iv)
- 2) viii)

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- 1) iv)
- 2) viii)
- 3) vii)

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- 1) iv)
- 2) viii)
- 3) vii)

- 4) i)

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- |          |        |
|----------|--------|
| 1) iv)   | 4) i)  |
| 2) viii) | 5) ix) |
| 3) vii)  |        |

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- |          |         |
|----------|---------|
| 1) iv)   | 4) i)   |
| 2) viii) | 5) ix)  |
| 3) vii)  | 6) iii) |

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- 1) iv)
- 2) viii)
- 3) vii)

- 4) i)
- 5) ix)
- 6) iii)

- 7) v)

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- 1) iv)
- 2) viii)
- 3) vii)

- 4) i)
- 5) ix)
- 6) iii)

- 7) v)
- 8) ii)

## SQL Puzzle



In dieser Aufgabe geht es immer um die Tabelle **land**, deren erste Datensätze du hier siehst:

<b>id</b>	<b>name</b>	<b>einwohner</b>	<b>flaeche</b>	<b>hauptstadt</b>
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

- 1) iv)
- 2) viii)
- 3) vii)

- 4) i)
- 5) ix)
- 6) iii)

- 7) v)
- 8) ii)
- 9) vi)



## Wdh: SQL Basics



Bearbeite die Aufgabe [Wdh - SQL Basics](#) auf [artemis.tum.de](http://artemis.tum.de). Artemis gibt dir immer, wenn du auf Submit drückst, die ersten Zeilen der Ergebnistabelle und ob deine SQL-Abfrage (bzw. welche Teile von ihr) richtig sind, aus.

Wenn du eine Abfrage richtig hast, notiere sie unten im Skript.

Falls du bei Gruppierung und Aggregatfunktionen Schwierigkeiten hast, hilft dir dieses [Video \(bitte Kopfhörer verwenden!\)](#): [bycs.link/simpleclub-group-sort-aggregat](http://bycs.link/simpleclub-group-sort-aggregat)

1) Vervollständige die SQL-Abfrage so, dass sie ID, Name, Art und URL aller Freibäder ausgibt.



## Wdh: SQL Basics



Bearbeite die Aufgabe [Wdh - SQL Basics](#) auf [artemis.tum.de](http://artemis.tum.de). Artemis gibt dir immer, wenn du auf Submit drückst, die ersten Zeilen der Ergebnistabelle und ob deine SQL-Abfrage (bzw. welche Teile von ihr) richtig sind, aus.

Wenn du eine Abfrage richtig hast, notiere sie unten im Skript.

Falls du bei Gruppierung und Aggregatfunktionen Schwierigkeiten hast, hilft dir dieses [Video \(bitte Kopfhörer verwenden!\)](#): [bycs.link/simpleclub-group-sort-aggregat](http://bycs.link/simpleclub-group-sort-aggregat)

1) Vervollständige die SQL-Abfrage so, dass sie ID, Name, Art und URL aller Freibäder ausgibt.

```
SELECT id, name, art, url  
FROM Schwimmbad  
WHERE art=Freibad"
```

## Wdh: SQL Basics



- 2) Schreibe eine SQL-Abfrage, die ausgibt, wie viele Gemeinden es im Regierungsbezirk "Oberbayern" gibt.
  
- 3) Schreibe eine SQL-Abfrage, die Name, Straße und URL (also die Internetadresse) aller Zoos in der Gemeinde mit Schlüssel "09162000" ausgibt.

## Wdh: SQL Basics



2) Schreibe eine SQL-Abfrage, die ausgibt, wie viele Gemeinden es im Regierungsbezirk "Oberbayern" gibt.

```
SELECT COUNT(*)  
FROM Gemeinde  
WHERE regierungsbezirk="Oberbayern"
```

3) Schreibe eine SQL-Abfrage, die Name, Straße und URL (also die Internetadresse) alle Zoos in der Gemeinde mit Schlüssel "09162000" ausgibt.

## Wdh: SQL Basics



2) Schreibe eine SQL-Abfrage, die ausgibt, wie viele Gemeinden es im Regierungsbezirk "Oberbayern" gibt.

```
SELECT COUNT(*)  
FROM Gemeinde  
WHERE regierungsbezirk="Oberbayern"
```

3) Schreibe eine SQL-Abfrage, die Name, Straße und URL (also die Internetadresse) aller Zoos in der Gemeinde mit Schlüssel "09162000" ausgibt.

```
SELECT name, strasse, url  
FROM Zoo  
WHERE gemeindeschlüssel = "09162000"
```



- 4) Schreibe eine SQL-Abfrage, die die Summe aller weiblichen Einwohnerinnen und die Summe aller männlichen Einwohner gruppiert nach Regierungsbezirk und den Namen des jeweiligen Regierungsbezirks ausgibt.
  
- 5) Schreibe eine SQL-Abfrage, die die durchschnittliche Fläche der Gemeinde eines Kreises (=Landkreis) und den Namen und Regierungsbezirk des jeweiligen Landkreises anzeigt. Sortiere die Ausgabe nach Name des Landkreises.



- 4) Schreibe eine SQL-Abfrage, die die Summe aller weiblichen Einwohnerinnen und die Summe aller männlichen Einwohner gruppiert nach Regierungsbezirk und den Namen des jeweiligen Regierungsbezirks ausgibt.

```
SELECT regierungsbezirk, SUM(einwohner_w), SUM(einwohner_m)
FROM gemeinde
GROUP BY regierungsbezirk
```

- 5) Schreibe eine SQL-Abfrage, die die durchschnittliche Fläche der Gemeinde eines Kreises (=Landkreis) und den Namen und Regierungsbezirk des jeweiligen Landkreises anzeigt. Sortiere die Ausgabe nach Name des Landkreises.

## Wdh: SQL Basics



- 4) Schreibe eine SQL-Abfrage, die die Summe aller weiblichen Einwohnerinnen und die Summe aller männlichen Einwohner gruppiert nach Regierungsbezirk und den Namen des jeweiligen Regierungsbezirks ausgibt.

```
SELECT regierungsbezirk, SUM(einwohner_w), SUM(einwohner_m)
FROM gemeinde
GROUP BY regierungsbezirk
```

- 5) Schreibe eine SQL-Abfrage, die die durchschnittliche Fläche der Gemeinde eines Kreises (=Landkreis) und den Namen und Regierungsbezirk des jeweiligen Landkreises anzeigt. Sortiere die Ausgabe nach Name des Landkreises.

```
SELECT regierungsbezirk, kreis, avg(flaeche)
FROM Gemeinde
GROUP BY regierungsbezirk,kreis
ORDER BY kreis
```



- 6) Schreibe eine SQL-Abfrage, die die Namen und Einwohnerzahlen aller Gemeinde, die mehr als 100.000 männliche und mehr als 100.000 weibliche Einwohner:innen haben, ausgibt.
- 7) Schreibe eine SQL-Abfrage, die die Namen und Einwohnerzahlen aller Gemeinde, die mehr als 75.000 männliche oder mehr als 75.000 weibliche Einwohner:innen haben, ausgibt.

## Wdh: SQL Basics



- 6) Schreibe eine SQL-Abfrage, die die Namen und Einwohnerzahlen aller Gemeinde, die mehr als 100.000 männliche und mehr als 100.000 weibliche Einwohner:innen haben, ausgibt.

```
SELECT name, einwohner_m, einwohner_w  
FROM Gemeinde  
WHERE einwohner_m > 100000  
AND einwohner_w > 100000
```

- 7) Schreibe eine SQL-Abfrage, die die Namen und Einwohnerzahlen aller Gemeinde, die mehr als 75.000 männliche oder mehr als 75.000 weibliche Einwohner:innen haben, ausgibt.

## Wdh: SQL Basics



- 6) Schreibe eine SQL-Abfrage, die die Namen und Einwohnerzahlen aller Gemeinde, die mehr als 100.000 männliche und mehr als 100.000 weibliche Einwohner:innen haben, ausgibt.

```
SELECT name, einwohner_m, einwohner_w  
FROM Gemeinde  
WHERE einwohner_m > 100000  
AND einwohner_w > 100000
```

- 7) Schreibe eine SQL-Abfrage, die die Namen und Einwohnerzahlen aller Gemeinde, die mehr als 75.000 männliche oder mehr als 75.000 weibliche Einwohner:innen haben, ausgibt.

```
SELECT name, einwohner_m, einwohner_w  
FROM Gemeinde  
WHERE einwohner_m > 75000  
OR einwohner_w > 75000
```



- 8) Schreibe eine SQL-Abfrage, die Name, Landkreis, Fläche und die Einwohnerzahlen aller Gemeinden ausgibt, die jeweils mehr als 50.000 männliche und weibliche Einwohner:innen oder eine Fläche größer als 100 km<sup>2</sup> hat.
- 9) Schreibe eine SQL-Abfrage, die die durchschnittlichen männlichen und weiblichen Einwohnerzahlen aller Gemeinde mit mehr als 100 km<sup>2</sup> Fläche pro Landkreis und den Namen des jeweiligen Landkreises ausgibt.

## Wdh: SQL Basics



8) Schreibe eine SQL-Abfrage, die Name, Landkreis, Fläche und die Einwohnerzahlen aller Gemeinden ausgibt, die jeweils mehr als 50.000 männliche und weibliche Einwohner:innen oder eine Fläche größer als 100 km<sup>2</sup> hat.

```
SELECT name, kreis, flaeche, einwohner_m, einwohner_w  
FROM Gemeinde  
WHERE (einwohner_m > 50000 AND einwohner_w > 50000)  
OR flaeche > 100
```

9) Schreibe eine SQL-Abfrage, die die durchschnittlichen männlichen und weiblichen Einwohnerzahlen aller Gemeinde mit mehr als 100 km<sup>2</sup> Fläche pro Landkreis und den Namen des jeweiligen Landkreises ausgibt.

## Wdh: SQL Basics



8) Schreibe eine SQL-Abfrage, die Name, Landkreis, Fläche und die Einwohnerzahlen aller Gemeinden ausgibt, die jeweils mehr als 50.000 männliche und weibliche Einwohner:innen oder eine Fläche größer als 100 km<sup>2</sup> hat.

```
SELECT name, kreis, flaeche, einwohner_m, einwohner_w  
FROM Gemeinde  
WHERE (einwohner_m > 50000 AND einwohner_w > 50000)  
OR flaeche > 100
```

9) Schreibe eine SQL-Abfrage, die die durchschnittlichen männlichen und weiblichen Einwohnerzahlen aller Gemeinde mit mehr als 100 km<sup>2</sup> Fläche pro Landkreis und den Namen des jeweiligen Landkreises ausgibt.

```
SELECT kreis, AVG(einwohner_m), AVG(einwohner_w)  
FROM Gemeinde  
WHERE flaeche > 100  
GROUP BY kreis
```

## Wdh: SQL Basics



10) Schreibe eine SQL-Abfrage, die die Anzahl von Wanderwegen, die zu einer Gemeinde führen in einer Spalte Anzahl und den jeweiligen Gemeindeschlüssel absteigend nach Anzahl sortiert, ausgibt.

## Wdh: SQL Basics



10) Schreibe eine SQL-Abfrage, die die Anzahl von Wanderwegen, die zu einer Gemeinde führen in einer Spalte Anzahl und den jeweiligen Gemeindeschlüssel absteigend nach Anzahl sortiert, ausgibt.

```
SELECT gemeindeschluessel,COUNT(*) as Anzahl  
FROM Wanderweg_zu_Gemeinde  
GROUP BY gemeindeschluessel  
ORDER BY Anzahl DESC
```

# Outline

Stunde 1+2

Stunde 3+4

Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12

## SQL Puzzle

In dieser Aufgabe geht es immer um die Tabelle land, deren erste Datensätze du hier siehst:

id	name	einwohner	flaeche	hauptstadt
1	Deutschland	83.24	358	Berlin
2	Frankreich	67.39	544	Paris
3	Brasilien	212.60	8516	Rio de Janeiro
...	...	...	...	...

Welche SQL-Abfrage (rechte Seite) führt zu welcher Ergebnistabelle (linke Seite)? Ordne richtig zu!

Lösung:

1) iv)  
2) viii)  
3) viii)

4) i)  
5) ix)  
6) iii)

7) v)  
8) ii)  
9) vi)

## Wdh: SQL Basics

Bearbeite die Aufgabe [Wdh - SQL Basics](#) auf [artemis.tum.de](#). Artemis gibt dir immer, wenn du auf Submit drückst, die ersten Zeilen der Ergebnistabelle und ob deine SQL-Abfrage (bzw. welche Teile von ihr) richtig sind, aus.

Wenn du eine Abfrage richtig hast, notiere sie unten im Skript.

Falls du bei Gruppierung und Aggregatfunktionen Schwierigkeiten hast, hilft dir dieses [Video](#) (bitte Kopfhörer verwenden!): [bycs.link/simpleclub-group-sort-aggregat](https://bycs.link/simpleclub-group-sort-aggregat)

1) Vervollständige die SQL-Abfrage so, dass sie ID, Name, Art und URL aller Freibäder ausgibt. `SELECT id, name, art, url FROM Schwimmbad WHERE art='Freibad'`

## Tabellenbeziehungen

1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.



## Tabellenbeziehungen: Fremdschlüssel

Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im Tabellschema werden die **Fremdschlüssel** durch überstrichen (manchmal auch unterpunkten) markiert. Ein Beispiel in SQL-Island ist der Häuptling eines Dorfes, der in der Tabelle Dorf mittels bewohnernr eingetragen wird. Die **bewohnernr** ist hierbei **Primärschlüssel** in der Tabelle Bewohner und **Fremdschlüssel** in der Tabelle Dorf (heißt hier aber **haeuptling**).

# Tabellenbeziehungen



1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.

SELECT * FROM dorf		
dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7

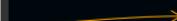
SELECT * FROM Bewohner						
bewohnernr	name	dorfnr	geschlecht	beruf	gold	status
1	Paul Backmann	1	m	Baecker	850	friedlich
2	Ernst Peng	3	m	Waffenschmied	280	friedlich
3	Rita Ochse	1	w	Baecker	350	friedlich
4	Carl Ochse	1	m	Kaufmann	250	friedlich
5	Dirty Dieter	3	m	Schmied	650	boese
6	Gerd Schlachter	2	m	Metzger	4850	boese
7	Peter Schlachter	3	m	Metzger	3250	boese
8	Arthur Schneiderpaule	2	m	Pilot	490	gefangen

# Tabellenbeziehungen



1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.

SELECT * FROM dorf		
dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7



SELECT * FROM Bewohner							
bewohnernr	name	dorfnr	geschlecht	beruf	gold	status	
1	Paul Backmann	1	m	Baecker	850	friedlich	
2	Ernst Peng	3	m	Waffenschmied	280	friedlich	
3	Rita Ochse	1	w	Baecker	350	friedlich	
4	Carl Ochse	1	m	Kaufmann	250	friedlich	
5	Dirty Dieter	3	m	Schmied	650	boese	
6	Gerd Schlachter	2	m	Metzger	4850	boese	
7	Peter Schlachter	3	m	Metzger	3250	boese	
8	Arthur Schneiderpaule	2	m	Pilot	490	gefangen	

# Tabellenbeziehungen



1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.

SELECT * FROM dorf		
dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7

SELECT * FROM Bewohner						
bewohnernr	name	dorfnr	geschlecht	beruf	gold	status
1	Paul Backmann	1	m	Baecker	850	friedlich
2	Ernst Peng	3	m	Waffenschmied	280	friedlich
3	Rita Ochse	1	w	Baecker	350	friedlich
4	Carl Ochse	1	m	Kaufmann	250	friedlich
5	Dirty Dieter	3	m	Schmied	650	boese
6	Gerd Schlachter	2	m	Metzger	4850	boese
7	Peter Schlachter	3	m	Metzger	3250	boese
8	Arthur Schneiderpaule	2	m	Pilot	490	gefangen

# Tabellenbeziehungen



1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.

SELECT * FROM dorf		
dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7

SELECT * FROM Bewohner						
bewohnernr	name	dorfnr	geschlecht	beruf	gold	status
1	Paul Backmann	1	m	Baecker	850	friedlich
2	Ernst Peng	3	m	Waffenschmied	280	friedlich
3	Rita Ochse	1	w	Baecker	350	friedlich
4	Carl Ochse	1	m	Kaufmann	250	friedlich
5	Dirty Dieter	3	m	Schmied	650	boese
6	Gerd Schlachter	2	m	Metzger	4850	boese
7	Peter Schlachter	3	m	Metzger	3250	boese
8	Arthur Schneiderpaule	2	m	Pilot	490	gefangen

# Tabellenbeziehungen



1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.

SELECT * FROM dorf			SELECT * FROM Bewohner						
dorfnr	name	haeuptling	bewohnernr	name	dorfnr	geschlecht	beruf	gold	status
1	Affenstadt	1	1	Paul Backmann	1	m	Baecker	850	friedlich
2	Gurkendorf	6	2	Ernst Peng	3	m	Waffenschmied	280	friedlich
3	Zwiebelhausen	7	3	Rita Ochse	1	w	Baecker	350	friedlich
			4	Carl Ochse	1	m	Kaufmann	250	friedlich
			5	Dirty Dieter	3	m	Schmied	650	boese
			6	Gerd Schlachter	2	m	Metzger	4850	boese
			7	Peter Schlachter	3	m	Metzger	3250	boese
			8	Arthur Schneiderpaule	2	m	Pilot	490	gefangen

# Tabellenbeziehungen



1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.

The diagram illustrates the relationship between two tables: `dorf` and `Bewohner`.

**Table dorf:**

```
SELECT * FROM dorf
```

dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7

**Table Bewohner:**

```
SELECT * FROM Bewohner
```

bewohnernr	name	dorfnr	geschlecht	beruf	gold	status
1	Paul Backmann	1	m	Baecker	850	friedlich
2	Ernst Peng	3	m	Waffenschmied	280	friedlich
3	Rita Ochse	1	w	Baecker	350	friedlich
4	Carl Ochse	1	m	Kaufmann	250	friedlich
5	Dirty Dieter	3	m	Schmied	650	boese
6	Gerd Schlachter	2	m	Metzger	4850	boese
7	Peter Schlachter	3	m	Metzger	3250	boese
8	Arthur Schneiderpaule	2	m	Pilot	490	gefangen

Relationships shown by arrows:

- An arrow points from the `haeuptling` column in the `dorf` table to the `bewohnernr` column in the `Bewohner` table, indicating that the value in the `haeuptling` column of the `dorf` table corresponds to the `bewohnernr` of the `Bewohner` table.
- Two curved arrows point from the `dorfnr` column in the `dorf` table to the `dorfnr` column in the `Bewohner` table, one for row 1 (Affenstadt) and one for row 3 (Zwiebelhausen).
- A blue curved arrow points from the `haeuptling` column in the `dorf` table to the `bewohnernr` column in the `Bewohner` table, indicating that the `haeuptling` value (1 or 6 or 7) corresponds to the `bewohnernr` value in the `Bewohner` table.

# Tabellenbeziehung im Klassendiagramm



1. Ergänze das Klassendiagramm entsprechend der beiden Tabellen oben.
2. Wie kann man die Beziehungen zwischen den beiden Tabellen im Klassendiagramm darstellen?

Tipp: Unsere Überlegungen von oben, helfen dabei.

Dorf
int dorfnr
String name

Bewohner
int bewohnernr
String name
String geschlecht
String beruf
int gold
String status

# Tabellenbeziehung im Klassendiagramm



1. Ergänze das Klassendiagramm entsprechend der beiden Tabellen oben.
2. Wie kann man die Beziehungen zwischen den beiden Tabellen im Klassendiagramm darstellen?

Tipp: Unsere Überlegungen von oben, helfen dabei.

Dorf
int dorfnr
String name

1  
haeuptling

Bewohner
int bewohnernr
String name
String geschlecht
String beruf
int gold
String status

## Tabellenbeziehungen: Fremdschlüssel



Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im **Tabellenschema** werden die durch (manchmal auch .

## Tabellenbeziehungen: Fremdschlüssel



Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im **Tabellenschema** werden die **Fremdschlüssel** durch \_\_\_\_\_ durch (manchmal auch \_\_\_\_\_).

## Tabellenbeziehungen: Fremdschlüssel



Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im **Tabellenschema** werden die **Fremdschlüssel** durch **überstreichen** (manchmal auch **markieren**) .

## Tabellenbeziehungen: Fremdschlüssel



Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im **Tabellenschema** werden die **Fremdschlüssel** durch **überstreichen** (manchmal auch **..... unterpunkten .....**) markiert. Ein Beispiel in SQL-Island ist der Häuptling eines Dorfes, der in der Tabelle Dorf mittels bewohnernr eingetragen wird. Die **bewohnernr** ist hierbei in der **Tabelle Bewohner** und in der **Tabelle Dorf** (heißt hier aber **haeuptling**).

## Tabellenbeziehungen: Fremdschlüssel



Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im **Tabellenschema** werden die **Fremdschlüssel** durch **überstreichen** (manchmal auch **..... unterpunkten .....**) markiert. Ein Beispiel in SQL-Island ist der Häuptling eines Dorfes, der in der Tabelle Dorf mittels bewohnernr eingetragen wird. Die **bewohnernr** ist hierbei **Primärschlüssel** in der **Tabelle Bewohner** und **in der Tabelle Dorf** (heißt hier aber **haeuptling**).

## Tabellenbeziehungen: Fremdschlüssel



Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem Fremdschlüssel. Im **Tabellenschema** werden die **Fremdschlüssel** durch **überstreichen** (manchmal auch **..... unterpunkten .....**) markiert. Ein Beispiel in SQL-Island ist der Häuptling eines Dorfes, der in der Tabelle Dorf mittels bewohnernr eingetragen wird. Die **bewohnernr** ist hierbei **Primärschlüssel** in der **Tabelle Bewohner** und **Fremdschlüssel** in der **Tabelle Dorf** (heißt hier aber **haeuptling**).

# Outline

Stunde 1+2

Stunde 3+4

Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12

## Tabellenbeziehungen

1. Visualisiere (mit Bleistift), wer Häuptling in welchem Dorf ist.
2. Überlege, wie du allgemein für diese zwei Tabellen darstellen kannst, wie sie (und ihre Spalten) miteinander in Beziehung stehen.



## Tabellenbeziehungen: Fremdschlüssel

Wenn Datensätze mittels Primärschlüssel in einer anderen Tabelle verwendet werden, spricht man dort von einem **Fremdschlüssel**. Im **Tabellschema** werden die **Fremdschlüssel** durch **überstreichen** (manchmal auch ... **unterpunkten**...) markiert. Ein Beispiel in SQL-Island ist der Häuptling eines Dorfes, der in der Tabelle **Dorf** mittels bewohnernr eingetragen wird. Die **bewohnernr** ist hierbei **Primärschlüssel** in der Tabelle **Bewohner** und **Fremdschlüssel** in der Tabelle **Dorf** (heißt hier aber **häuptling**).

### Tabellenbeziehungen im Klassendiagramm

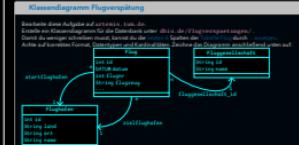
```
classDiagram
    class Dorf {
        int id
        String name
    }
    class Bewohner {
        int id
        String name
    }
    Dorf "1..1" --> "1..1" Bewohner : Häuptling
    Bewohner "*" --> "1..1" Dorf : Fremdschlüssel
```

• **Häuptling**: immer pro Dorf aus Fremdschlüssel.  
• **Fremdschlüssel**: mit einer Spalte der Tabellent, wird dort aber nicht eingeprägt.  
• **Primärschlüssel**: wichtig für den Index und muss genau so sein, da andere Systeme andere Bedürfnisse haben!  
• **Referenzierung**: an der Präsidentin gibt es immer 1 Dorf betreffend, da in einer Spalte immer unterschieden werden darf wenn es mehrere sind.

### Kardinalitäten

Die Kardinalität beschreibt, wie viele Objekte auf jeder Seite einer Beziehung stehen können. Es gibt folgende Arten:

- 1:1, z.B. **Häuptling** pro Dorf, der auch nur in einem Dorf Häuptling ist.
- 1:N, d.h. jeder Bewohner wohnt in einem Dorf, das aber **meheren** Bewohnern hat.
- m:n, z.B. **belegung** viele Lehrer pro Schulfachern + **belegung** viele Schulfachern pro Lehrer (in Datenbanken nicht direkt umsetzbar, sondern später mehr)



### SQL: Tabellen verbinden

Wir können jetzt Tabellen, die miteinander über Fremd- und Primärschlüssel in Beziehung stehen, nun miteinander verbinden, das bedeutet, dass wir die Daten aus zwei oder mehreren Tabellen zusammenführen. Das kann man mit einer **SELECT**-Anfrage tun.

```
SELECT * FROM Dorf WHERE id = 1000;
```

Zum Beispiel kann man in SQL-Mittel die Daten aller Dorf und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT * FROM Dorf INNER JOIN Bewohner ON Dorf.bewohnernr = Bewohner.id;
```

### Kreuzprodukt / Join

Möchte man Daten aus zwei Tabellen mit Beziehung zusammenbringen, gibt man beide Tabellen mit einem **JOIN** an. Die SQL-Anfrage sieht dann das **Kreuzprodukt** der Tabellen. Die Ergebnistabelle erhält alle Kombinationen von Datensätzen beider Tabellen. **INNER JOIN** und **LEFT JOIN** unterscheiden sich darin, dass ein **LEFT JOIN** auch alle Datensätze aus der linken Tabelle ausgibt, die nicht mit einer Zeile aus der rechten Tabelle verbunden sind. Zum Beispiel kann man in SQL-Mittel die Daten aller Dorf und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT * FROM Dorf INNER JOIN Bewohner ON Dorf.bewohnernr = Bewohner.id;
```

## Tabellenbeziehungen im Klassendiagramm



TabelleA
int id
String spalte1
...

TabelleB
int id
String spalte1
...

## Tabellenbeziehungen im Klassendiagramm



TabelleA
int id
String spalte1
...

TabelleB
int id
String spalte1
...

## Tabellenbeziehungen im Klassendiagramm



## Tabellenbeziehungen im Klassendiagramm



- Beziehungspfeil immer vom Fremd- zum Primärschlüssel.
- 'fremdschlüssel' ist eine Spalte der TabelleA, wird dort aber nicht eingetragen.
- Die Form der Pfeilspitze ist wichtig und muss genau so sein, da andere Spitzen andere Bedeutungen haben!
- Kardinalität an der Pfeilspitze ist immer 1 (bei Datenbanken), da in einer Spalte (eines Datensatzes) immer nur ein Wert stehen kann.

## Kardinalitäten



Die Kardinalität beschreibt, wie viele Objekte auf jeder Seite einer Beziehung stehen können. Es gibt folgende Arten:

- **1:1**, z.B. Häuptling pro Dorf, der auch nur in einem Dorf Häuptling ist.
- **1:n**, z.B. jeder Bewohner wohnt in einem Dorf, das aber Bewohner hat.
- **m:n**, z.B. Lehrer pro Schulkasse + Schulklassen pro Lehrer (in Datenbanken nicht direkt umsetzbar, dazu später mehr).

## Kardinalitäten



Die Kardinalität beschreibt, wie viele Objekte auf jeder Seite einer Beziehung stehen können. Es gibt folgende Arten:

- **1:1**, z.B. **ein** Häuptling pro Dorf, der auch nur in einem Dorf Häuptling ist.
- **1:n**, z.B. jeder Bewohner wohnt in einem Dorf, das aber Bewohner hat.
- **m:n**, z.B. Lehrer pro Schulkasse + Schulklassen pro Lehrer (in Datenbanken nicht direkt umsetzbar, dazu später mehr).

## Kardinalitäten



Die Kardinalität beschreibt, wie viele Objekte auf jeder Seite einer Beziehung stehen können. Es gibt folgende Arten:

- **1:1**, z.B. **ein** Häuptling pro Dorf, der auch nur in einem Dorf Häuptling ist.
- **1:n**, z.B. jeder Bewohner wohnt in einem Dorf, das aber **mehrere** Bewohner hat.
- **m:n**, z.B. Lehrer pro Schulkasse + Schulklassen pro Lehrer (in Datenbanken nicht direkt umsetzbar, dazu später mehr).

## Kardinalitäten



Die Kardinalität beschreibt, wie viele Objekte auf jeder Seite einer Beziehung stehen können. Es gibt folgende Arten:

- **1:1**, z.B. **ein** Häuptling pro Dorf, der auch nur in einem Dorf Häuptling ist.
- **1:n**, z.B. jeder Bewohner wohnt in einem Dorf, das aber **mehrere** Bewohner hat.
- **m:n**, z.B. **beliebig viele** Lehrer pro Schulkasse + Schulklassen pro Lehrer (in Datenbanken nicht direkt umsetzbar, dazu später mehr).

## Kardinalitäten



Die Kardinalität beschreibt, wie viele Objekte auf jeder Seite einer Beziehung stehen können. Es gibt folgende Arten:

- **1:1**, z.B. **ein** Häuptling pro Dorf, der auch nur in einem Dorf Häuptling ist.
- **1:n**, z.B. jeder Bewohner wohnt in einem Dorf, das aber **mehrere** Bewohner hat.
- **m:n**, z.B. **beliebig viele** Lehrer pro Schulkasse + **beliebig viele** Schulklassen pro Lehrer (in Datenbanken nicht direkt umsetzbar, dazu später mehr).



## Klassendiagramm Flugverspätung



Bearbeite diese Aufgabe auf [artemis.tum.de](#).

Erstelle ein Klassendiagramm für die Datenbank unter [dbiu.de/flugverspaetungen/](#).

Damit du weniger schreiben musst, kannst du die **letzten 6** Spalten der **Tabelle Flug** durch ... ersetzen.

Achte auf korrektes Format, Datentypen und Kardinalitäten. Zeichne das Diagramm anschließend unten auf:



# Klassendiagramm Flugverspätung

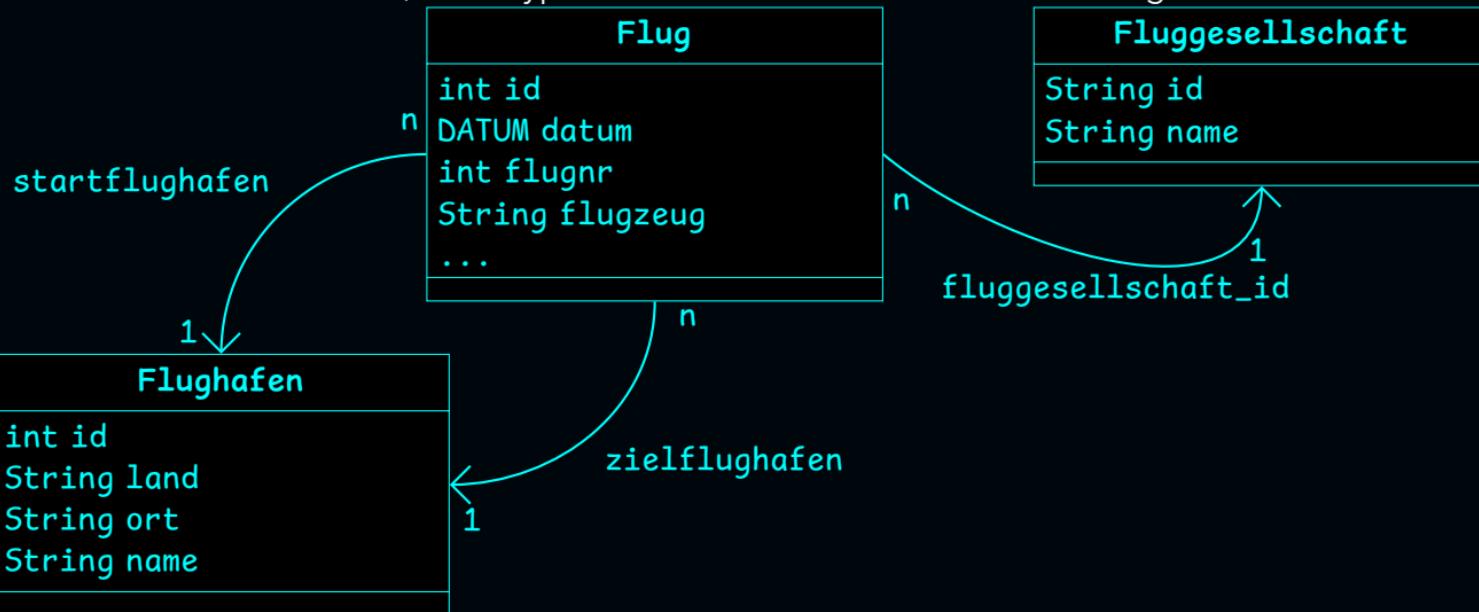


Bearbeite diese Aufgabe auf [artemis.tum.de](http://artemis.tum.de).

Erstelle ein Klassendiagramm für die Datenbank unter [dbiu.de/flugverspaetungen/](http://dbiu.de/flugverspaetungen/).

Damit du weniger schreiben musst, kannst du die **letzten 6** Spalten der Tabelle Flug durch ... ersetzen.

Achte auf korrektes Format, Datentypen und Kardinalitäten. Zeichne das Diagramm anschließend unten auf:



## SQL: Tabellen verbinden



Wir kennen jetzt Tabellen, die miteinander über Fremd- und Primärschlüssel in Beziehung stehen. Nun möchten wir aus diesen Tabellen auch zusammengehörende Datensätze abfragen.  
Öffne dafür [www.dbiu.de/flugverspaetungen](http://www.dbiu.de/flugverspaetungen) und führe folgende SQL-Abfrage aus:

```
SELECT *  
FROM Fluggesellschaft, Flug
```

<b>id</b>	<b>name</b>
2PQ	21 Air LLC
Q5	40-Mile Air
CIQ	A/S Conair
...	...

<b>id</b>	<b>datum</b>	<b>fluggesellschaft_id</b>	<b>flugnr</b>	...
1	1987-10-01	AA	516	...
2	1987-10-01	AA	516	...
3	1987-10-01	AA	516	...
...	...	...	...	...

**Fluggesellschaft**

**Flug**

**SELECT \***  
**FROM Fluggesellschaft, Flug**

**Ergebnistabelle**

<b>id</b>	<b>name</b>	<b>id</b>	<b>datum</b>	<b>fluggesellschaft_id</b>	<b>flugnr</b>	...
2PQ	21 Air LLC	1	1987-10-01	AA	516	...
Q5	40-Mile Air	1	1987-10-01	AA	516	...
CIQ	A/S Conair	1	1987-10-01	AA	516	...
AAE	AAA Airlines	1	1987-10-01	AA	516	...
ACI	AAA Action Air Carrier Inc.	1	1987-10-01	AA	516	...

## SQL: Tabellen verbinden



Was beobachtest du? Werden nur zusammengehörende Datensätze angezeigt? Falls nicht, nach welchem Muster werden die beiden Tabellen miteinander kombiniert?

## SQL: Tabellen verbinden



Was beobachtest du? Werden nur zusammengehörende Datensätze angezeigt? Falls nicht, nach welchem Muster werden die beiden Tabellen miteinander kombiniert?

**Nein, es werden alle Datensätze aus einer mit allen Datensätzen aus der anderen kombiniert und die Spalten einfach hintereinander aufgeregert.**

## Kreuzprodukt / Join



Möchte man Daten aus zwei Tabellen mit Beziehung zueinander abfragen, gibt man beide Tabellen **mit Komma getrennt nach FROM** an.

Die SQL-Abfrage bildet dann das der Tabellen. Die Ergebnistabelle enthält von Datensätzen beider Tabellen (**Merkregel:** ).

Um nur zusammengehörige Datensätze (also solche, die miteinander in Beziehung stehen, z.B. eine Bewohner mit seinem Dorf) auszuwählen, ergänzt man als **Selektion** eine **Gleichheitsbedingung** zwischen Fremd- und zugehörigem . Dann spricht man von einem .

Zum Beispiel kann man in SQL-Island die Daten aller Dörfer und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT *
FROM Dorf, Bewohner
WHERE Dorf.haeuptling = Bewohner.bewohnernr
```

## Kreuzprodukt / Join



Möchte man Daten aus zwei Tabellen mit Beziehung zueinander abfragen, gibt man beide Tabellen **mit Komma getrennt nach FROM** an.

Die SQL-Abfrage bildet dann das **Kreuzprodukt** der Tabellen. Die Ergebnistabelle enthält von Datensätzen beider Tabellen **(Merkregel:** ).

Um nur zusammengehörige Datensätze (also solche, die miteinander in Beziehung stehen, z.B. eine Bewohner mit seinem Dorf) auszuwählen, ergänzt man als **Selektion** eine **Gleichheitsbedingung** zwischen Fremd- und zugehörigem . Dann spricht man von einem .

Zum Beispiel kann man in SQL-Island die Daten aller Dörfer und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT *
FROM Dorf, Bewohner
WHERE Dorf.haeuptling = Bewohner.bewohnernr
```

## Kreuzprodukt / Join



Möchte man Daten aus zwei Tabellen mit Beziehung zueinander abfragen, gibt man beide Tabellen **mit Komma getrennt nach FROM** an.

Die SQL-Abfrage bildet dann das **Kreuzprodukt** der Tabellen. Die Ergebnistabelle enthält **alle Kombinationen** von Datensätzen beider Tabellen (**Merkregel**: ).

Um nur zusammengehörige Datensätze (also solche, die miteinander in Beziehung stehen, z.B. eine Bewohner mit seinem Dorf) auszuwählen, ergänzt man als **Selektion** eine **Gleichheitsbedingung** zwischen Fremd- und zugehörigem . Dann spricht man von einem .

Zum Beispiel kann man in SQL-Island die Daten aller Dörfer und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT *
  FROM Dorf, Bewohner
 WHERE Dorf.haeuptling = Bewohner.bewohnernr
```

## Kreuzprodukt / Join



Möchte man Daten aus zwei Tabellen mit Beziehung zueinander abfragen, gibt man beide Tabellen **mit Komma getrennt nach FROM** an.

Die SQL-Abfrage bildet dann das **Kreuzprodukt** der Tabellen. Die Ergebnistabelle enthält **alle Kombinationen** von Datensätzen beider Tabellen (**Merkregel: Jeder mit jedem**).

Um nur zusammengehörige Datensätze (also solche, die miteinander in Beziehung stehen, z.B. eine Bewohner mit seinem Dorf) auszuwählen, ergänzt man als **Selektion** eine **Gleichheitsbedingung** zwischen Fremd- und zugehörigem . Dann spricht man von einem .

Zum Beispiel kann man in SQL-Island die Daten aller Dörfer und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT *
FROM Dorf, Bewohner
WHERE Dorf.haeuptling = Bewohner.bewohnernr
```

## Kreuzprodukt / Join



Möchte man Daten aus zwei Tabellen mit Beziehung zueinander abfragen, gibt man beide Tabellen **mit Komma getrennt nach FROM** an.

Die SQL-Abfrage bildet dann das **Kreuzprodukt** der Tabellen. Die Ergebnistabelle enthält **alle Kombinationen** von Datensätzen beider Tabellen (**Merkregel: Jeder mit jedem**).

Um nur zusammengehörige Datensätze (also solche, die miteinander in Beziehung stehen, z.B. eine Bewohner mit seinem Dorf) auszuwählen, ergänzt man als **Selektion** eine **Gleichheitsbedingung** zwischen Fremd- und zugehörigem **Primärschlüssel**. Dann spricht man von einem .

Zum Beispiel kann man in SQL-Island die Daten aller Dörfer und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT *
  FROM Dorf, Bewohner
 WHERE Dorf.haeuptling = Bewohner.bewohnernr
```

## Kreuzprodukt / Join



Möchte man Daten aus zwei Tabellen mit Beziehung zueinander abfragen, gibt man beide Tabellen **mit Komma getrennt nach FROM** an.

Die SQL-Abfrage bildet dann das **Kreuzprodukt** der Tabellen. Die Ergebnistabelle enthält **alle Kombinationen** von Datensätzen beider Tabellen (**Merkregel: Jeder mit jedem**).

Um nur zusammengehörige Datensätze (also solche, die miteinander in Beziehung stehen, z.B. eine Bewohner mit seinem Dorf) auszuwählen, ergänzt man als **Selektion** eine **Gleichheitsbedingung** zwischen Fremd- und zugehörigem **Primärschlüssel**. Dann spricht man von einem **Join**.

Zum Beispiel kann man in SQL-Island die Daten aller Dörfer und ihrer zugehörigen Häuptlinge so ausgeben:

```
SELECT *
  FROM Dorf, Bewohner
 WHERE Dorf.haeuptling = Bewohner.bewohnernr
```

# Outline

Stunde 1+2

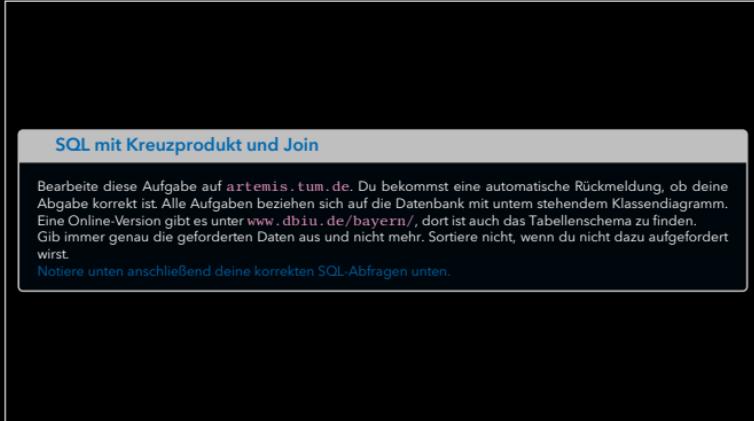
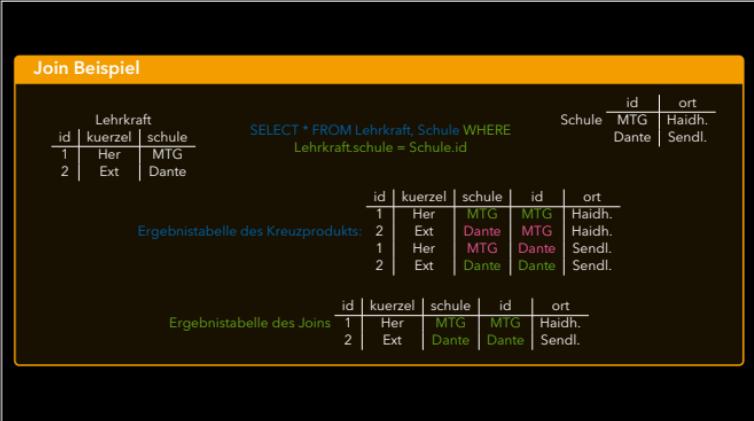
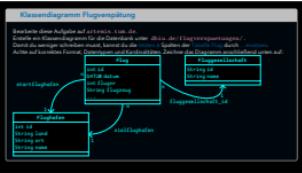
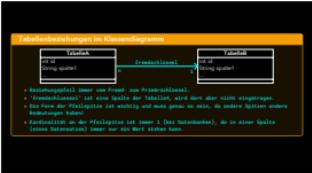
Stunde 3+4

Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12



# Join Beispiel



Lehrkraft		
id	kuerzel	schule
1	Her	MTG
2	Ext	Dante

```
SELECT *
FROM Lehrkraft, Schule
WHERE Lehrkraft.schule = Schule.id
```

Schule	
id	ort
MTG	Haidh.
Dante	Sendl.

Ergebnistabelle des Kreuzprodukts:

id	kuerzel	schule	id	ort
1	Her	MTG	MTG	Haidh.
2	Ext	Dante	MTG	Haidh.
1	Her	MTG	Dante	Sendl.
2	Ext	Dante	Dante	Sendl.

Ergebnistabelle des Joins

id	kuerzel	schule	id	ort
1	Her	MTG	MTG	Haidh.
2	Ext	Dante	Dante	Sendl.



## SQL mit Kreuzprodukt und Join

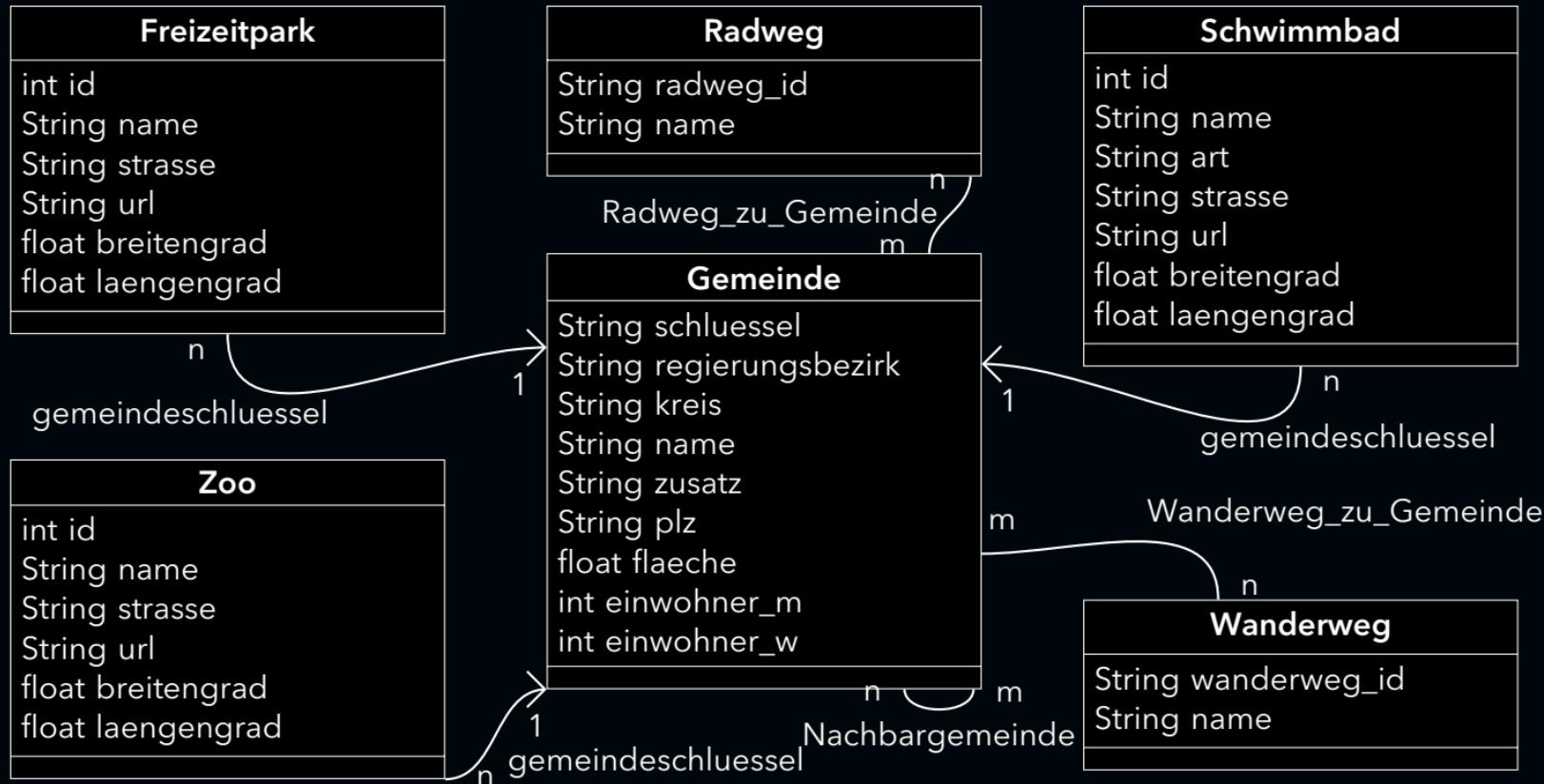


Bearbeite diese Aufgabe auf [artemis.tum.de](http://artemis.tum.de). Du bekommst eine automatische Rückmeldung, ob deine Abgabe korrekt ist. Alle Aufgaben beziehen sich auf die Datenbank mit unten stehendem Klassendiagramm. Eine Online-Version gibt es unter [www.dbiu.de/bayern/](http://www.dbiu.de/bayern/), dort ist auch das Tabellenschema zu finden. Gib immer genau die geforderten Daten aus und nicht mehr. Sortiere nicht, wenn du nicht dazu aufgefordert wirst.

**Notiere unten anschließend deine korrekten SQL-Abfragen unten.**



# SQL mit Kreuzprodukt und Join





## SQL mit Kreuzprodukt und Join



Verändere die SQL-Abfrage so, dass die Namen und Internetadressen (=url) aller Zoos und der Name und Regierungsbezirk der jeweiligen Gemeinde ausgegeben wird:

```
SELECT Zoo.name, Gemeinde.name
```

```
FROM Zoo, Gemeinde
```



## SQL mit Kreuzprodukt und Join



Verändere die SQL-Abfrage so, dass die Namen und Internetadressen (=url) aller Zoos und der Name und Regierungsbezirk der jeweiligen Gemeinde ausgegeben wird:

```
SELECT Zoo.name, Gemeinde.name ,Gemeinde.regierungsbezirk, Zoo.url  
FROM Zoo, Gemeinde
```



## SQL mit Kreuzprodukt und Join



Verändere die SQL-Abfrage so, dass die Namen und Internetadressen (=url) aller Zoos und der Name und Regierungsbezirk der jeweiligen Gemeinde ausgegeben wird:

```
SELECT Zoo.name, Gemeinde.name ,Gemeinde.regierungsbezirk, Zoo.url  
FROM Zoo, Gemeinde  
  
WHERE Zoo.gemeindeschluessel = Gemeinde.schluessel
```



## SQL mit Kreuzprodukt und Join



Verändere die SQL-Abfrage so, dass die Namen und Straßen aller Freizeitparks und die Namen der jeweils zugehörigen Gemeinde ausgegeben wird.

**SELECT Freizeitpark.name, Gemeinde.name**

**FROM Freizeitpark, Gemeinde**



## SQL mit Kreuzprodukt und Join



Verändere die SQL-Abfrage so, dass die Namen und Straßen aller Freizeitparks und die Namen der jeweils zugehörigen Gemeinde ausgegeben wird.

```
SELECT Freizeitpark.name, Gemeinde.name , Freizeitpark.strasse  
FROM Freizeitpark, Gemeinde
```



## SQL mit Kreuzprodukt und Join



Verändere die SQL-Abfrage so, dass die Namen und Straßen aller Freizeitparks und die Namen der jeweils zugehörigen Gemeinde ausgegeben wird.

```
SELECT Freizeitpark.name, Gemeinde.name , Freizeitpark.strasse  
FROM Freizeitpark, Gemeinde  
WHERE Gemeinde.schluessel = Freizeitpark.gemeindeschluessel
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die Namen und Art aller Schwimmbäder und den Namen und alle Einwohnerzahlen der zugehörigen Gemeinden ausgibt.



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die Namen und Art aller Schwimmbäder und den Namen und alle Einwohnerzahlen der zugehörigen Gemeinden ausgibt.

```
SELECT Schwimmbad.name, Schwimmbad.art,  
Gemeinde.name, Gemeinde.einwohner_m, Gemeinde.einwohner_w  
FROM Schwimmbad, Gemeinde  
WHERE Gemeinde.schlüssel = Schwimmbad.gemeindeschlüssel
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Anzahl an Schwimmbädern in Gemeinden mit **mehr** als 1000 weiblichen Einwohnerinnen ausgibt.

**Tipp:** Hier brauchst du mehrere verknüpfte Bedingungen



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Anzahl an Schwimmbädern in Gemeinden mit **mehr** als 1000 weiblichen Einwohnerinnen ausgibt.

**Tipp:** Hier brauchst du mehrere verknüpfte Bedingungen

```
SELECT COUNT(*)
FROM Schwimmbad, Gemeinde
WHERE Gemeinde.schlüssel = Schwimmbad.gemeindeschlüssel
AND Gemeinde.einwohner_w > 1000
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Namen aller Gemeinde in Oberbayern oder Niederbayern, zu denen ein Wanderweg führt, ausgibt. Dopplungen dürfen auftreten und sollte nicht entfernt werden!

**Tipp:** Hier brauchst du wieder mehrere verknüpfte Bedingungen. Überlege bei der Verknüpfung von Bedingungen, ob du Klammern setzen musst!



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Namen aller Gemeinde in Oberbayern oder Niederbayern, zu denen ein Wanderweg führt, ausgibt. Dopplungen dürfen auftreten und sollte nicht entfernt werden!

**Tipp:** Hier brauchst du wieder mehrere verknüpfte Bedingungen. Überlege bei der Verknüpfung von Bedingungen, ob du Klammern setzen musst!

```
SELECT Gemeinde.name  
FROM Gemeinde,Wanderweg_zu_Gemeinde  
WHERE Gemeinde.schlüssel = Wanderweg_zu_Gemeinde.gemeindeschlüssel  
AND (Gemeinde.regierungsbezirk='Oberbayern'  
OR Gemeinde.regierungsbezirk='Niederbayern')
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die aus den Tabellen Gemeinde und Wanderweg\_zu\_Gemeinde die Anzahl der Wanderwege, die zu Gemeinden mit mehr als 500 000 männlichen Einwohnern führen, ausgibt.



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die aus den Tabellen Gemeinde und Wanderweg\_zu\_Gemeinde die Anzahl der Wanderwege, die zu Gemeinden mit mehr als 500 000 männlichen Einwohnern führen, ausgibt.

```
SELECT COUNT(*)
FROM Gemeinde, Wanderweg_zu_Gemeinde
WHERE Gemeinde.schlüssel = Wanderweg_zu_Gemeinde.gemeindeschlüssel
AND einwohner_m > 500000
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die eine Liste mit den Namen aller Gemeinden, die ein 'Freibad' haben, und die Namen der jeweiligen Freibäder ausgibt.



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die eine Liste mit den Namen aller Gemeinden, die ein 'Freibad' haben, und die Namen der jeweiligen Freibäder ausgibt.

```
SELECT Gemeinde.name, Schwimmbad.name  
FROM Gemeinde, Schwimmbad  
WHERE Gemeinde.schlüssel=Schwimmbad.gemeindeschlüssel  
AND Schwimmbad.art='Freibad'
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Anzahl an Radwegen, die an Gemeinden im PLZ-Bereich **größer** als 96400 angrenzen, ausgibt.



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Anzahl an Radwegen, die an Gemeinden im PLZ-Bereich **größer** als 96400 angrenzen, ausgibt.

```
SELECT COUNT(*)
FROM Gemeinde, Radweg_zu_Gemeinde
WHERE Gemeinde.schlüssel=Radweg_zu_Gemeinde.gemeindeschlüssel
AND Gemeinde.plz > 96400
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Namen aller Zoos in einer Gemeinde namens 'Erlangen' ausgibt.



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die Namen aller Zoos in einer Gemeinde namens 'Erlangen' ausgibt.

```
SELECT Zoo.name  
FROM Zoo,Gemeinde  
WHERE Zoo.gemeindeschluessel = Gemeinde.schluessel  
AND Gemeinde.name='Erlangen'
```



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die IDs aller Radwege, die zu Gemeinden in Oberfranken oder Unterfranken führen, ausgibt. Dopplungen sollen nicht entfernt werden.



## SQL mit Kreuzprodukt und Join



Schreibe eine SQL-Abfrage, die die IDs aller Radwege, die zu Gemeinden in Oberfranken oder Unterfranken führen, ausgibt. Dopplungen sollen nicht entfernt werden.

```
SELECT Radweg_zu_Gemeinde.radweg_id  
FROM Radweg_zu_Gemeinde, Gemeinde  
WHERE Gemeinde.schlüssel = Radweg_zu_Gemeinde.gemeindeschlüssel  
AND (Gemeinde.regierungsbezirk = 'Oberfranken'  
OR Gemeinde.regierungsbezirk='Unterfranken')
```

