Informatik 10 - Datenbanken (Teil 1)

Informatik 10 - Datenbanken (Teil 1)

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 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: Klassen und Objekte

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

Klassenkarte

Objektkarte

Wdh: Klassen und Objekte

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

Klassenkarte

Objektkarte

Wdh: Klassen und Objekte

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

Wdh: Klassen und Objekte

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

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

Objektkarte

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()

Objektkarte

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()

Objektkarte

spitze Ecken

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()

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

Objektkarte

spitze Ecken

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()

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

Objektkarte

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

spitze Ecken

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

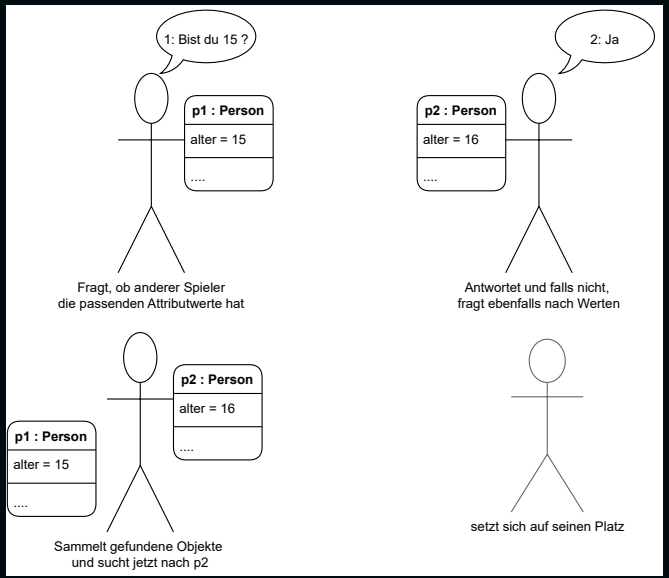
Objektkarte

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 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:

Klassenkarte

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

spitze Ecken

← Klassenname

Objektname : Klassenname →

Attribute

← Methoden

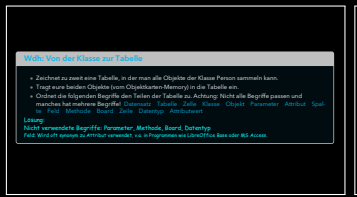
Objektkarte

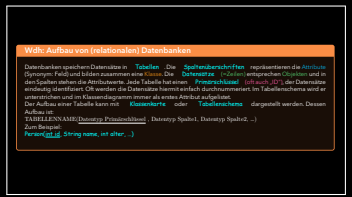
p1 : Person
hobby = "Klettern"
alter = 23
hatHaustier = false
peinlichesErlebnis = "..."

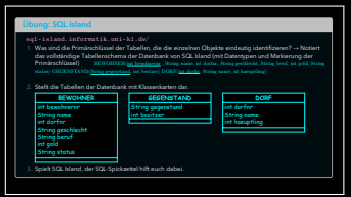
runde Ecken

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 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, 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 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 **Spaltenüberschriften** 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 **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 **(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 **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 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 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, …)

SQL Spickzettel

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

Spickzettel SQL

SELECT *Spaltenliste*
- **DISTINCT** vermeidet Duplikate.
- Aggregatfunktionen (**COUNT**, **SUM**, **MAX**, **MIN**, **AVG**) für Berechnungen
- SELECT * für "alle Spalten"
- **AS** Aliasname

FROM *Tabelle*

WHERE *Bedingung*
- Wird meist mit **Vergleichen** (<, <=, =, <>, ...) formuliert.
- Verknüpfung von mehreren Vergleichen mit **logischen Funktionen** (**AND**, **OR**, **NOT**)

GROUP BY *Spaltenliste*

HAVING *Bedingung*

ORDER BY *Spaltenliste*
- **ASC** für aufsteigend (Standard)
- **DESC** für absteigend

Im Detail gilt:

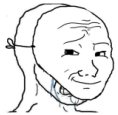
Grundlegende SQL-Abfrage	
SELECT	Es muss mindestens ein Spaltenname angegeben werden. Die entsprechende(n) Spalte(n) sind dann Teil der Ergebnistabelle. SELECT * bewirkt, dass alle Spalten angezeigt werden.
DISTINCT	Duplikate von Datensätzen werden nicht angezeigt.
AS	Eine Spalte in der Ergebnistabelle 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	SELECT DISTINCT kontinent AS "enthaltene Kontinente" FROM Land SELECT name, flaeche, hauptstadt FROM LAND ORDER BY flaeche DESC

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 <, <=, =, <>, 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. Folgende zwei Platzhalter (wildcards) werden häufig eingesetzt: • % steht für beliebig viele Zeichen, auch keines (* bei MS Access) • _ für genau ein beliebiges Zeichen (? bei MS Access) Beispiele: • WHERE titel LIKE "You%" – findet alle Titel die mit "You" beginnen Groß-/Kleinschreibung 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	SELECT COUNT(*) AS "Anzahl afrikanischer Länder " FROM Land WHERE kontinent = "Afrika"

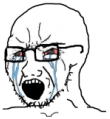
Gruppierung	
GROUP BY	Datensätze mit demselben Wert in der angegeben Spalte werden gruppiert. Gruppierungen sind nur in Kombination mit Aggregatfunktionen sinnvoll.
HAVING	An gruppierte Datensätze werden Bedingungen mit HAVING formuliert.
Beispiel	SELECT fsk, MIN(laufzeit) FROM Film WHERE genre1="Filmkomödie" OR genre2="Filmkomödie" GROUP BY fsk HAVING fsk <16

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



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

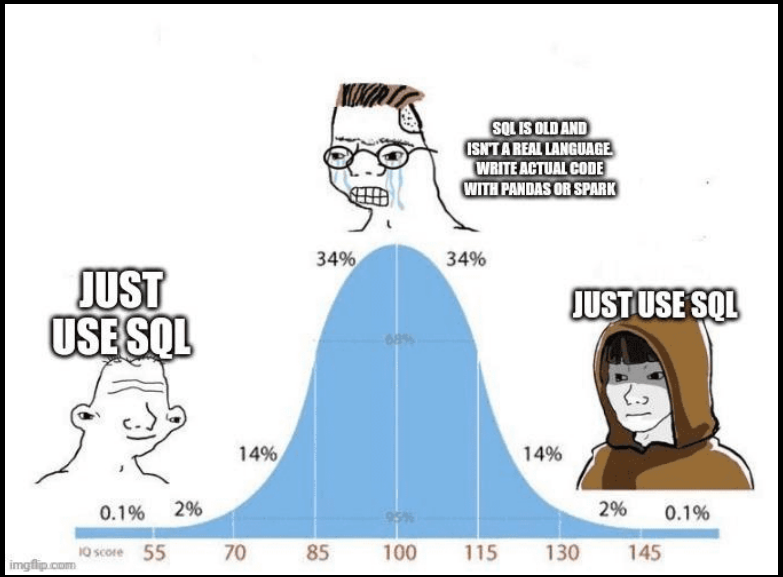
Noooo, they must be in **upper case**!



```
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-kl.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.

3. Spielt SQL Island, der SQL-Spickzettel hilft euch dabei.

Übung: SQL Island

`sql-island.informatik.uni-kl.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.

3. Spielt SQL Island, der SQL-Spickzettel hilft euch dabei.

Übung: SQL Island

`sql-island.informatik.uni-kl.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.

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

3. Spielt SQL Island, der SQL-Spickzettel hilft euch dabei.

Outline

Stunde 1+2

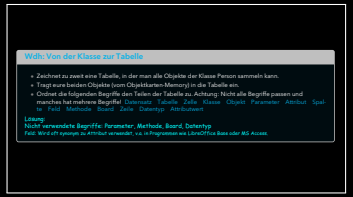
Stunde 3+4

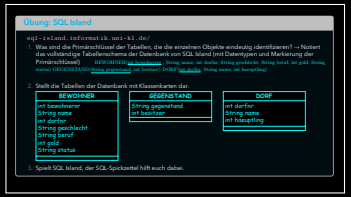
Stunde 5+6

Stunde 7+8

Stunde 9+10

Stunde 11+12





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) 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`. 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`

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

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!

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)

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) vii)

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) vii)

4) i)

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) vii)
4) i)
5) ix)

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

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) 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:

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) 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:

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) 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`. 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`

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`. 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`

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) alle Zoos in der Gemeinde mit Schluessel ‘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 Schluessel ‘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 Schluessel ‘09162000’ ausgibt.

```
SELECT name, strasse, url
FROM Zoo
WHERE gemeindeschluessel = '09162000'
```

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.

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.

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
```

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.

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
```

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 km² hat.

9) Schreibe eine SQL-Abfrage, die die durchschnittlichen männlichen und weiblichen Einwohnerzahlen aller Gemeinde mit mehr als 100 km² 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 km² 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 km² 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 km² 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 km² 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) 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`. 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`

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

```
SELECT id, name, art, urlFROM SchwimmbadWHERE 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 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).

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

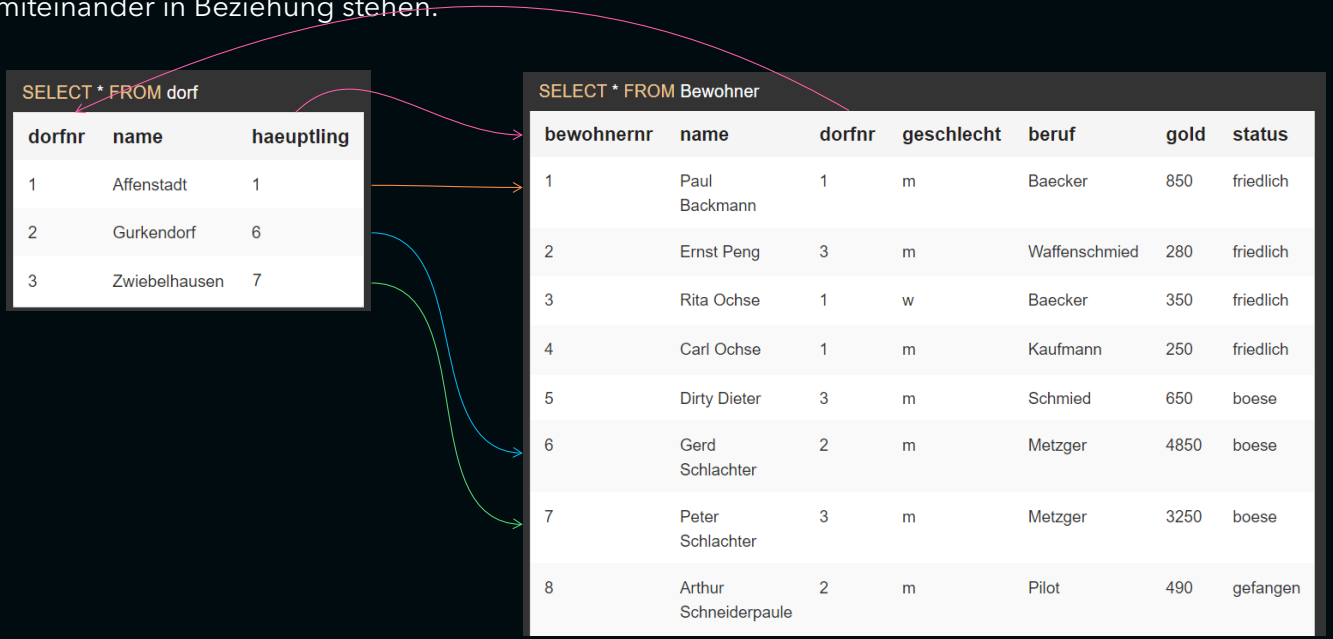
dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7

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

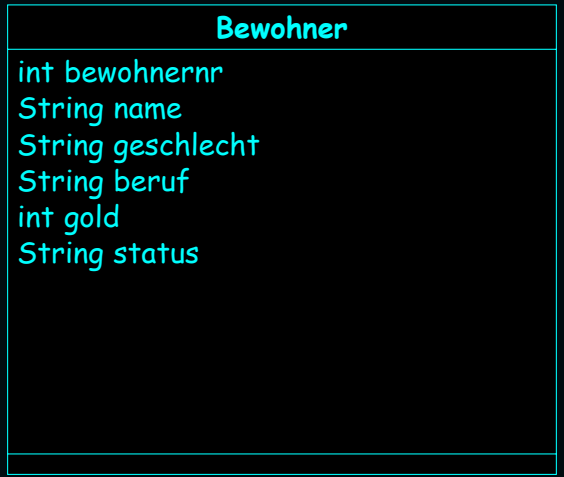
dorfnr	name	haeuptling
1	Affenstadt	1
2	Gurkendorf	6
3	Zwiebelhausen	7

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

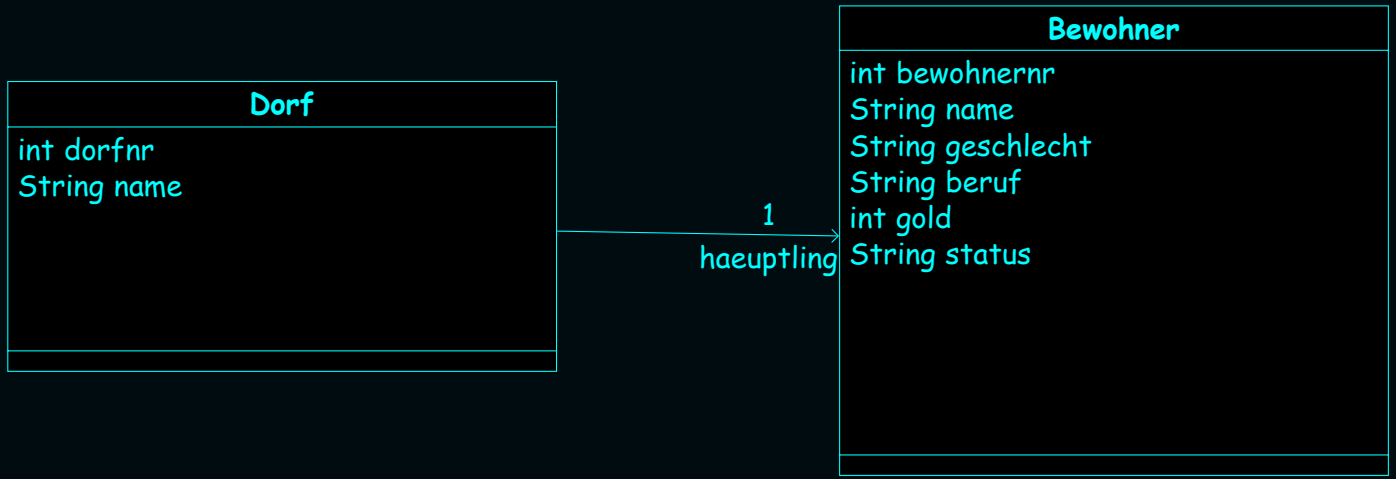
Tabellenbeziehung im Klassendiagramm

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.



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.



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
(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 .

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

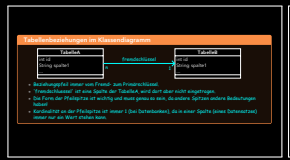
Stunde 11+12

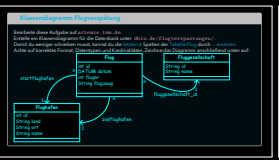
Tabellenbeziehungen

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 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).





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.
- 'fremdschluessel' 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 Schulklasse + 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 Schulklasse + 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 Schulklasse + 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 Schulklasse + 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:

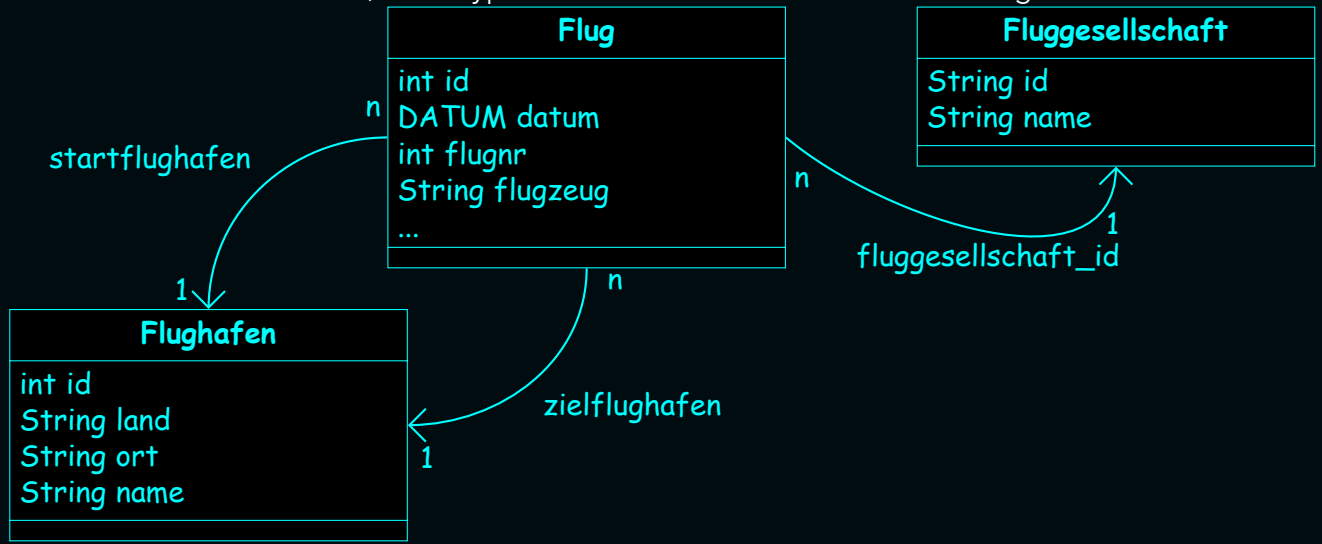
- **1:1**, z.B. ein Häuptling pro Dorf, der auch nur in einem Dorf Häuptling ist.
- **1:n**, z.B. jeder Bewohner wohnt in einem Dorf, das aber mehrere Bewohner hat.
- **m:n**, z.B. beliebig viele Lehrer pro Schulklasse + beliebig viele Schulklassen pro Lehrer (in Datenbanken nicht direkt umsetzbar, dazu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



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` und führe folgende SQL-Abfrage aus:

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

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

Fluggesellschaft

id	datum	fluggesellschaft_id	flugnr	...
1	1987-10-01	AA	516	...
2	1987-10-01	AA	516	...
3	1987-10-01	AA	516	...
...	...	...	...	...

Flug

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

Ergebnistabelle

id	name	id	datum	fluggesellschaft_id	flugnr	...
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 aufgereiht.

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 miteinenader 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 miteinenader 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 miteinenader 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 miteinenader 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 miteinenader 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 miteinenader 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

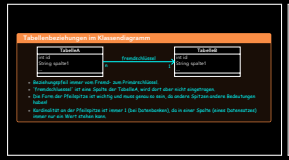
Stunde 3+4

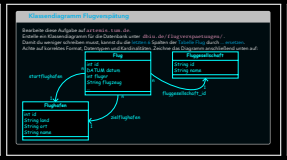
Stunde 5+6

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. Du bekommst eine automatische Rückmeldung, ob deine Abgabe korrekt ist. Alle Aufgaben beziehen sich auf die Datenbank mit untem stehendem Klassendiagramm. Eine Online-Version gibt es unter 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.

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.

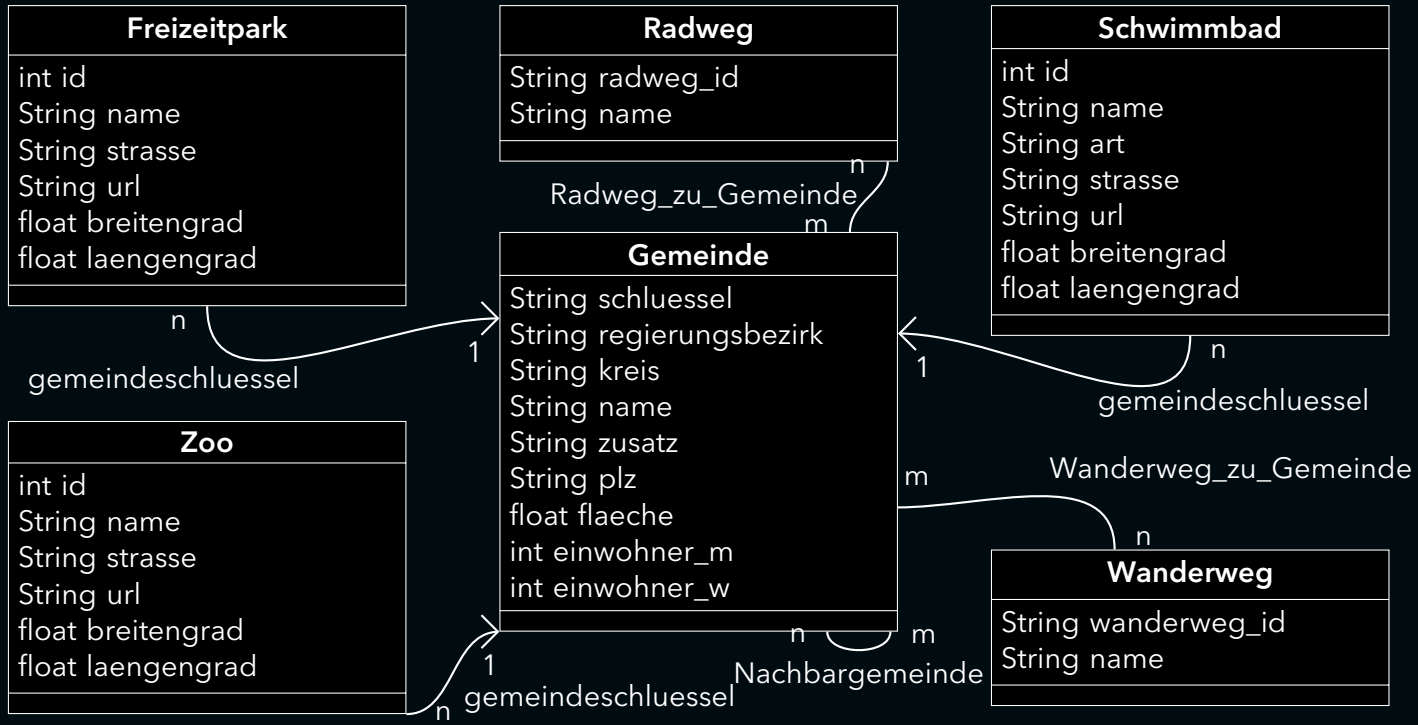
Ergebnistabelle des Joins

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`. Du bekommst eine automatische Rückmeldung, ob deine Abgabe korrekt ist. Alle Aufgaben beziehen sich auf die Datenbank mit untem stehendem Klassendiagramm. Eine Online-Version gibt es unter `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.schluessel = Schwimmbad.gemeindeschluessel
```

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.schluessel = Schwimmbad.gemeindeschluessel
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.schluessel = Wanderweg_zu_Gemeinde.gemeindeschluessel
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.schluessel = Wanderweg_zu_Gemeinde.gemeindeschluessel
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.schluessel=Schwimmbad.gemeindeschluessel
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.schluessel=Radweg_zu_Gemeinde.gemeindeschluessel
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.schluessel = Radweg_zu_Gemeinde.gemeindeschluessel
AND (Gemeinde.regierungsbezirk = 'Oberfranken'
OR Gemeinde.regierungsbezirk='Unterfranken')
```