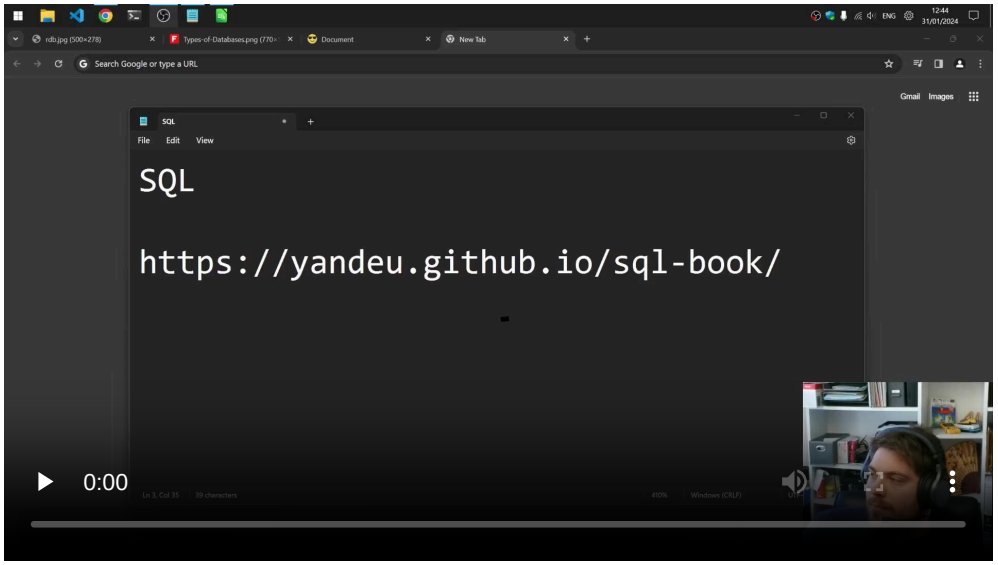
SQL Intro


Search Google or type a URL
Gmail
Images
SQL
File
Edit
View
SQL
https://yandeu.github.io/sql-book/
0:00

1 - CRUD Operationen

CRUD steht für Create, Read, Update, Delete. Diese sind basic Operatoren für die Interaktion einer SQL-Datenbank.
Die CRUD Operationen werden wie folgt übersetzt:

CRUD	SQL	HTTP
Create	INSERT INTO	POST
Read	SELECT	GET
Update	UPDATE	PATCH
Delete	DELETE FROM	DELETE

SELECT

`SELECT` wird verwendet, um Daten aus einer Datenbank auszuwählen.

```
SELECT column1, column2, ...
FROM table_name;
```

Hier sind Spalte1, Spalte2, ... die Felder der Tabelle, aus der Sie Daten auswählen möchten. Wenn Sie alle in der Tabelle verfügbaren Felder auswählen möchten, verwenden Sie die folgende Syntax:

```
SELECT * FROM table_name;
```

Beispiel:

```
SELECT name, city FROM users;
```

INSERT INTO

Die Anweisung `INSERT INTO` wird verwendet, um neue Datensätze in eine Tabelle einzufügen.

```
INSERT INTO table_name
  (column1, column2, column3, ...)
VALUES
  (value1, value2, value3, ...);
```

Es ist auch möglich mehrere Datensätze mit nur einem `INSERT INTO` einzufügen.

```
INSERT INTO table_name
  (column1, column2, column3, ...)
VALUES
  (value1, value2, value3, ...),
  (value1, value2, value3, ...),
  (value1, value2, value3, ...);
```

Beispiel:

```
INSERT INTO users
  (name, age, country)
VALUES
  ('Joe', 26, 'UK');
```

```
INSERT INTO users
  (name, age, country)
VALUES
  ('Joe', 26, 'UK'),
  ('Melanie', 28, 'CH'),
  ('Oceane', 25, 'FR');
```

UPDATE

`UPDATE` wird verwendet, um die vorhandenen Datensätze in einer Tabelle zu ändern.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Beispiel:

```
UPDATE users
SET age = 26, city = 'Bern'
WHERE id = 7;
```

DELETE

`DELETE` wird verwendet, um vorhandene Datensätze in einer Tabelle zu löschen.

```
DELETE FROM table_name WHERE condition;
```

Beispiel:

```
/* löschen den user mit der id 12 */
DELETE FROM users WHERE id = 12;
```

```
/* lösche alle user die in Interlaken leben */
DELETE FROM users WHERE city = 'Interlaken';
```

2 - WHERE Clause / SFW

SFW steht für **S**ELECT **F**ROM **W**HERE

Wie du womöglich bereits im letzten Kapitel gemerkt hat, wird `WHERE` zum Filtern von Datensätzen verwendet.
Es wird verwendet, um nur die Datensätze zu extrahieren, die eine bestimmte Bedingung erfüllen.

`WHERE` wird nicht nur mit `SELECT` verwenden, sonder auch `UPDATE`, `DELETE`, etc.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Beispiel:

```
SELECT name, age
FROM users
WHERE country = 'CH';
```

Operatoren in der WHERE Clause

Die folgenden Operatoren können in der WHERE-Klausel verwendet werden:

Operator	Beschreibung
=	Gleich
>	Grösser als
<	Kleiner als
>=	Grösser als oder gleich
<=	Kleiner als oder gleich
!=	Nicht gleich

Beispiel:

```sql
/* Erhalte alle user die älter als 25 sind. */
SELECT *
FROM users
WHERE age > 25;
```

AND und OR Operatoren

Zusätzlich die den Operatoren im letzten Abschnitt, kann WHERE clause auch mit **AND** und **OR** genutzt werden.

Beispiel:

```sql
/* Erhalte alle name der Users aus der Schweiz und Deutschland. */
SELECT name
FROM users
WHERE country = 'CH' OR country = 'DE';
```

Kombination vom AND und OR

Klammern müssen verwenden werden bei komplexeren Operationen.

Beispiel:

```sql
/* Erhalte alle Users aus der Schweiz und Deutschland,
welche über 25 Jahre alt sind. */
SELECT name
FROM users
WHERE age > 25 AND (country = 'CH' OR country = 'DE');
```

3 - Aggregate Funktionen

Eine Aggregat Funktion in SQL gibt nach der Berechnung mehrerer Werte einer Spalte einen Wert zurück.

Verschiedene Arten von SQL-Aggregatfunktionen sind:

Funktion	Beschreibung
COUNT()	Anzahl der Zeilen
SUM()	Summe
AVG()	Durchschnitt
MIN()	Minimum
MAX()	Maximum

```
/* Count gibt die Anzahl der Zeilen zurück */
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

Beispiele:

```
/* Total Anzahl Users */
SELECT COUNT(id) FROM users;
```

```
/* Anzahl Users aus der Schweiz */
SELECT COUNT(id)
FROM users
WHERE country = 'CH';
```

```
/* Durchschnitts Alter aller User aus Italien */
SELECT AVG(age)
FROM users
WHERE country = 'IT';
```

4 - GROUP BY

Die `GROUP BY`-Anweisung gruppiert Zeilen mit denselben Werten in Zusammenfassungszeilen, wie z.B. "Finde die Anzahl der Users pro Land“.

Die `GROUP BY`-Anweisung wird häufig mit Aggregatfunktionen (letztes Kapitel) verwendet, um die Ergebnismenge nach einer oder mehreren Spalten zu gruppieren.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition /* optional */
GROUP BY column_name(s);
```

Beispiel:

```
SELECT COUNT(id), country
FROM users
GROUP BY country;

/*
 count | country
-------+---------
     1 | US
     1 | NL
    10 | CH
     1 | DE
     2 | FR
     1 | UK
     2 | IT
*/
```

5 - ORDER BY

Das Schlüsselwort `ORDER BY` wird verwendet, um die Ergebnismenge in aufsteigender oder absteigender Reihenfolge zu sortieren.

Das Schlüsselwort `ORDER BY` sortiert die Datensätze standardmäßig in **aufsteigender** Reihenfolge (`ASC`). Um die Datensätze in **absteigender** Reihenfolge zu sortieren, verwenden Sie das Schlüsselwort `DESC`.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Beispiel:

```
SELECT * FROM users
ORDER BY country;
```

```
SELECT * FROM users
ORDER BY country DESC;
```

ORDER BY - Mehrere Spalten

Die folgende SQL-Anweisung erhält alle Users zurück, sortiert nach der Spalte "country" und "city". Das bedeutet, dass nach Land sortiert wird. Wenn jedoch einige Zeilen dasselbe Land enthalten, werden sie nach Stadt sortiert:

```
SELECT * from users
ORDER BY country, city;
```

Die folgende SQL-Anweisung erhält alle Users zurück, aufsteigend nach Land und absteigend nach der Spalte Stadt sortiert:

```
SELECT * FROM users
ORDER BY country ASC, city DESC;
```

6 - Pagination

In digitalen Medien umfasst die Paginierung die Aufteilung von Inhalten in einzelne Seiten, normalerweise zur Navigation. Beispielsweise könnte ein Blog die Paginierung verwenden, um seine Beiträge in separate Seiten mit jeweils einer begrenzten Anzahl von Einträgen aufzuteilen.

Die SQL Befehle sind `LIMIT` und `OFFSET`.

Syntax:

```
SELECT column_name(s)
FROM table_name
LIMIT number
OFFSET number;
```

Beispiel:

Im ersten Query erhältst du die ersten 5 Users, im 2. Query die Users 6-10 und im 3. Query die Users 11-15

```
SELECT id, name
FROM users
LIMIT 5
OFFSET 0;
```

```
SELECT id, name
FROM users
LIMIT 5
OFFSET 5;
```

```
SELECT id, name
FROM users
LIMIT 5
OFFSET 10;
```

7 - LIKE Operator

Der LIKE-Operator wird in einer WHERE-Klausel verwendet, um nach einem angegebenen Muster in einer Spalte zu suchen. Das Prozentzeichen (%) steht für null, ein oder mehrere Zeichen.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE-Operator	**Beschreibung**
WHERE name LIKE '**a%**'	Findet alle Werte, die mit **a beginnen**
WHERE name LIKE '**%a**'	Findet alle Werte, die mit **a enden**
WHERE name LIKE '**%a%**'	Findet alle Werte, die an beliebiger Position ein **a enthalten**

8 - CREATE TABLE

Bis jetzt haben wir mit einer vorgegebenen Tabelle gearbeitet. Jetzt lernen wir wie wir eine neue Tabelle erstellen können.

Tabelle Erstellen

Datentyp	Beschreibung
SERIAL PRIMARY KEY	Automatisch hochzählende des Primärschlüssel (id)
INT	Ganzzahl
FLOAT	Dezimalzahl mit Nachkommastelle
TEXT	Sehr langer Text
VARCHAR(255)	Kleiner Text (max 255 Zeichen; kann auch kleiner sein)
TIMESTAMP	Zeitstempel

Hier ein Beispiel wie die Tabelle `users` definiert wird.

```
CREATE TABLE users(
  id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  age INT,
  city VARCHAR(255),
  country VARCHAR(2)
);
```

Einschränkungen

Einschränkungen	Beschreibung
NOT NULL	Nicht leer
UNIQUE	Einzigartig
REFERENCES	Fremdschlüssel auf andere Tabelle
DEFAULT	Ein standart Wert wenn nichts angeben

Beispiel:

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

Zu beachten: TIMESTAMP verwenden hier den Zusatz `WITH TME ZONE`, was auch die Zeitzone im Timestamp festhält, und als Standard Wert wir `now()` verwenden, was uns die aktuelle Zeit zurück gibt.

Aufgabe

Löst jetzt Aufgabe 1-3.

Fremdschlüssel

Wenn wir wie im Intro Video eine Relation (also Verbindung) zwischen 2 Tabellen herstellen wollen, müssen wir einen Fremdschlüssel verwenden.

Der Primärschlüssel einer Tabelle ist ja die `id`, wir müssen also irgendwie diese `id` in einer anderen Tabelle speichern können.
Dies schreiben wir wie folgt:
`user_id INT NOT NULL REFERENCES users(id)`.

- `user_id`: Der Name dieser Spalte (users table in singular + `_id`)
- `INT`: Damit ist die `id` gemeint, welche ja eigentlich ein Int (integer) ist
- `NOT NULL`: Ist optional, je nach dem der Fremdschlüssel hier zwingend ist oder nicht
- `REFERENCES users(id)`: Erstellt einen Verbindung zur `id` im users table

Hier ein Beispiel einer Posts Tabelle:

```
CREATE TABLE posts (
  id SERIAL PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  text TEXT DEFAULT 'TODO',
  user_id INT NOT NULL REFERENCES users(id),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

Aufgabe

Löst jetzt Aufgabe 4.

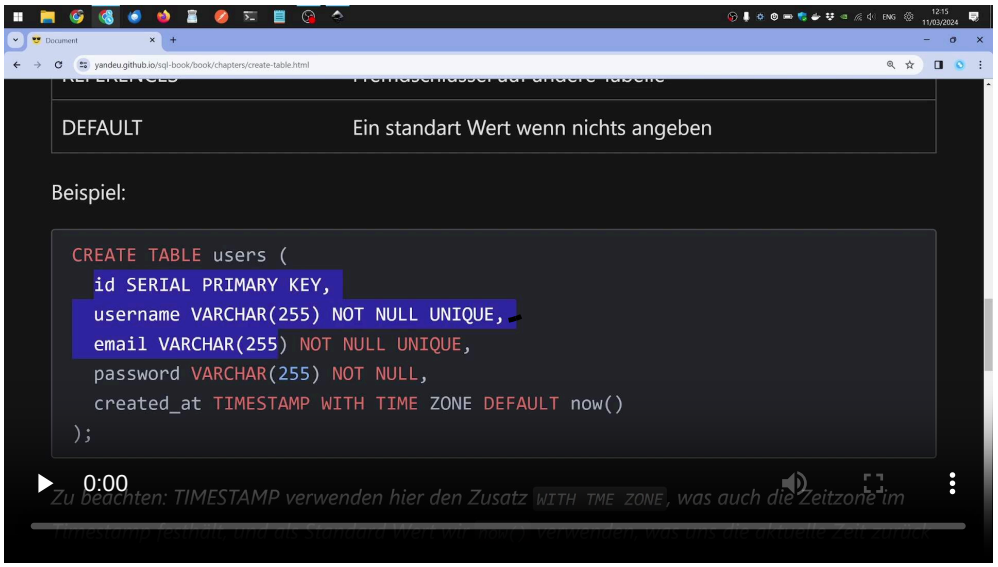
SQL Editor (Zusatz)

Mit `DROP TABLE table_name;` kannst du eine bestehende Tabelle löschen.

Teste deine Queries im Editor:
hier öffnen

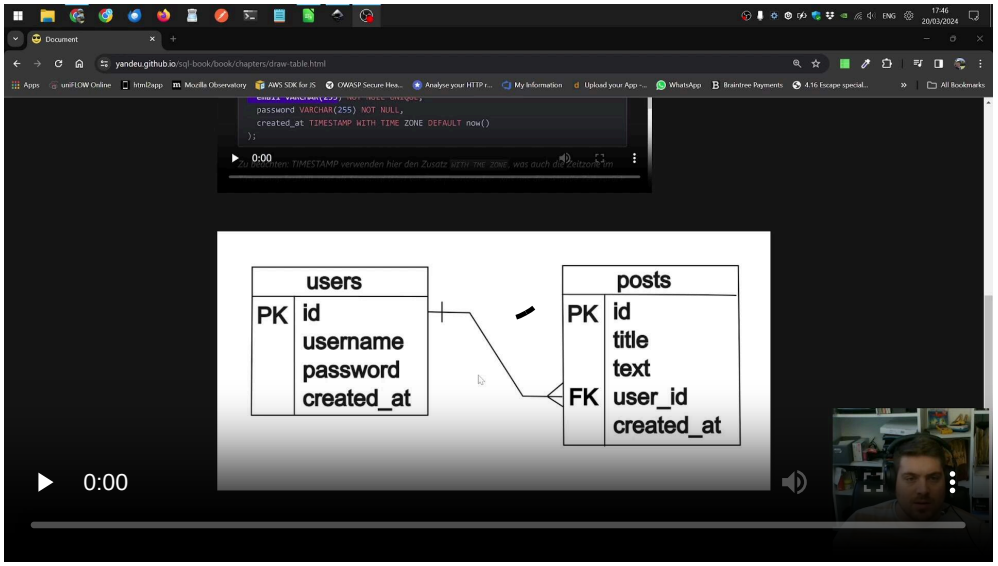
9 - Draw Table

Im letzten Kapitel lernten wir Tabellen mit SQL code zu erstellen. In diesem Kapitel Zeichnen wir SQL Tabellen.



10 - Many to Many

Video



Aufgaben - CRUD Operationen

1. Erhalte das Alter und die Stadt aller Users.

```
CODE
```

2. Füge den neuen User Elena, 22 aus Bern hinzu.

```
CODE
```

3. Ändere das Alter von User 11 zu 18.

```
CODE
```

4. Löschen den User mit der ID 8.

```
CODE
```

5. Wie heisst der User den du mit folgendem Query erhältst?

```sql
SELECT name FROM users WHERE id = 5;
```

```
CODE
```

Aufgaben - WHERE Clause / SFW

1. Erhalte das Alter und die Stadt aller Users.

```
CODE
```

2. Lösche alle Users mit dem Alter von 20 bis 25.

```
CODE
```

3. Erhalte das alter aller User, die nicht in der Schweiz wohnen.

```
CODE
```

4. Ersetzen bei allen Usern mit Wohnort Biel; Den Wohnort zu Biel/Bienne.

```
CODE
```

5. Welche Namen erhältst du bei Folgendem Query? Liste alle Namen auf.

```
SELECT name
FROM users
WHERE id = '10' OR city = 'Thun';
```

```
CODE
```

6. Welche Users werden bei folgendem Query gelöscht? Liste die Namen der gelöschen Users auf.
   Ergänze auch die fehlenden Wörter im Query

```
___ ___ users
WHERE age >= 18 AND (city = 'Biel' OR city = 'Bern');
```

```
CODE
```

7. Beschreibe in Worten was das folgende Query macht.

```
SELECT *
FROM users
WHERE city = 'Pescara' OR country = 'DE';
```

```
CODE
```

Aufgaben - Aggregate Funktionen

1. Erhalte das kleinste Alter aller Users.

```
CODE
```

2. Welche Zahl erhalte ich von folgendem Query?

```sql
SELECT SUM(age)
FROM users
WHERE name = 'Sabrina';
```

```
CODE
```

3. Erhalte das Durchschnittsalter aller Users aus Italien.

```
CODE
```

4. Welche Zahl erhalte ich bei folgendem Query?

```sql
SELECT MIN(id)
FROM users
WHERE age >= 25 AND age <= 30;
```

```
CODE
```

5. Welchen Namen erhalte ich bei folgendem Query?

```sql
SELECT MAX(name)
FROM users
WHERE country != 'CH';
```

```
CODE
```

Aufgaben - GROUP BY

1. Erhalte das kleinste Alter aller Users, gruppiert nach Land.

```
CODE
```

2. Was erhalte ich bei folgendem Query?

```
SELECT MIN(id), country
FROM users
WHERE age >= 20
GROUP BY country;
```

```
CODE
```

3. Erhalte die Anzahl Users aus der Schweiz, gruppiert nach Stadt.

```
CODE
```

Aufgaben - ORDER BY

1. Sortiere alle Users nach alter.

```
CODE
```

2. Welcher Name ist bei folgendem Query zuoberst in der Ausgabe?

```
SELECT name FROM users
WHERE id > 11
ORDER BY name DESC;
```

```
CODE
```

3. Erhalte das Durchschnittsalter aller volljährigen Users, gruppiert und sortiert nach Land.

```
CODE
```

Aufgaben - Pagination

1. Welche ids erhältst du bei folgendem Query?

```
SELECT id
FROM users
LIMIT 2
OFFSET 8;
```

```
CODE
```

2. Schreibe ein Query bei dem du nur die ersten 5 Users erhältst; sortiert (absteigend) nach Alter.

```
CODE
```

3. Pro Seite zeigt deine App 4 Users an. Ein Website Besucher greift auf Seite 3 zu. Welches query schreibst du?
   Ergänze die fehlenden Wörter im Query

```
SELECT * FROM users
LIMIT _
OFFSET _;
```

```
CODE
```

Aufgaben - LIKE

1. Was erhältst du bei folgendem Query?

```
SELECT city FROM users
WHERE name LIKE 'Sa%';
```

```
CODE
```

2. Welche Zahl erhältst du bei folgendem Query?

```
SELECT SUM(age) FROM users
WHERE city LIKE '%el%';
```

```
CODE
```

3. Schreibe ein Query bei dem du alle ids der Users, deren Name mit "ne" endet, erhältst?

```
CODE
```

Aufgaben - CREATE TABLE

1. Was ist der Unterschied zwischen Int und Float?

```
CODE
```

2. Erstelle eine Tabelle names "cars". Jedes Auto hat einen Namen, einen Hersteller, die Anzahl an Türen und optional auch die Farbe.

```
CODE
```

3. Erstelle eine Tabelle "manufacturers" (Auto Hersteller). Jeder Hersteller hat einen Namen und ein Gründungsjahr.

```
CODE
```

4. Jetzt wo die oberen Tabellen erstellt worden sind. Welche Tabelle muss angepasst werden damit eine Verbindung zwischen beiden Tabellen entsteht? Schreibe die Tabelle, welche angepasst werden muss, nochmals neu hier unten dran:
   Nimm dir Zeit diese Aufgabe zu lösen.

```
CODE
```

Aufgaben - Draw Table (One to Many)

Aufgabe: Zu jeder Aufgabe ein Schema zeichnen + SQL Code schreiben:

1. Eine Datenbank in der Schule.

- Jede **Schule** hat 1 oder mehrere **Gebäude**.
- Jedes **Gebäude** hat 1 oder mehrere **Zimmer**.
- Erfinde pro Tabelle mindestens 4 zusätzliche Spalten.

2. Bei einer Bank:

- Jeder **Kunden** hat ein oder mehrere **Konti**
- Jedes **Konto** hat mehrere **Bewegungen/Transaktionen**
- Alle **Konti** haben das Attribut iban.
- Jede **Bewegung/Transaction** hat einen Zeitstempel, einen Betrag, eine Währung.

Aufgaben - Many to Many

Aufgabe: Zu jeder Aufgabe ein Schema zeichnen + SQL Code schreiben:

1. In der Filmindustrie

- Jeder **Schauspieler** spielt in mehreren **Filmen**
- In jedem **Film** spielen mehrere **Schauspieler**
- Jeder **Film** hat einen **Regisseur**
- Jeder **Film** hat das Attribute «length» welches die Länge des Films in Minuten darstellt

2. In der Schule

- Jeder **key** kann bei mehreren **doors** benutzt werden
- Zu jeder **door** passen mehrere **keys**
- Jeder **teacher** hat einen oder mehrere **keys**
- Jeder **teacher** hat einen name
- Jede **door** hat eine description

Lösungen

CRUD

CRUD-1

```
SELECT age, city FROM users;
```

CRUD-2

```
INSERT INTO users
  (name, age, city)
VALUES
  ('Elena', 22, 'Bern');
```

CRUD-3

```
UPDATE users
SET age = 18
WHERE id = 584;
```

CRUD-4

```
DELETE FROM users WHERE id = 84;
```

CRUD-5

Sara

SFW

SFW-1

```
SELECT age, city FROM users;
```

SFW-2

```
DELETE FROM users
WHERE age >= 20 AND age <= 25;
```

SFW-3

```
SELECT age FROM users
WHERE country != 'CH';
```

SFW-4

```
UPDATE users
SET city = 'Biel/Bienne'
WHERE city = 'Biel';
```

SFW-5

Melanie, Oceane, Manuela, Mauro

SFW-6

DELETE FROM
Agate, Sara, Viktor

SFW-7

Gibt alle Users die in Deutschland wohnen oder in der Stadt Pescara zurück.

Aggregate

Aggregate-1

```
SELECT MIN(age) FROM users;
```

Aggregate-2

49

Aggregate-3

```
SELECT AVG(age) FROM users
WHERE country = 'IT';
```

Aggregate-4

5

Aggregate-5

Thomas

GroupBy

GroupBy-1

```
SELECT MIN(age), country
FROM users
GROUP BY country;
```

GroupBy-2

10 FR | 2 IT | 14 NL | 8 UK | 5 CH | 20 DE

GroupBy-3

```
SELECT COUNT(id), city
FROM users
WHERE country = 'CH'
GROUP BY city;
```

OrderBy

OrderBy-1

```
SELECT * FROM users
ORDER BY age;
```

OrderBy-2

Thomas

OrderBy-3

```
SELECT AVG(age), country
FROM users
WHERE age >= 18
GROUP BY country
ORDER BY country;
```

Pagination

Pagination-1

10, 11

Pagination-2

```
SELECT * FROM users
ORDER BY age DESC
LIMIT 5;
```

Pagination-3

4, 8

Like

Like-1

Bern, Pescara, Zuzwil

Like-2

65

Like-3

```
SELECT id FROM users
WHERE name LIKE '%ne';
```

Lösungen

CREATE TABLE

1. Was ist der Unterschied zwischen Int und Float?

Int = ganze Zahlen; Float = mit Kommastellen

2. Erstelle eine Tabelle names "cars". Jedes Auto hat einen Namen, einen Hersteller, die Anzahl an Türen und optional auch die Farbe.

```
CREATE TABLE cars (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  manufacturer VARCHAR(255) NOT NULL,
  doors_count INT NOT NULL,
  color VARCHAR(255)
);
```

3. Erstelle eine Tabelle "manufacturers" (Auto Hersteller). Jeder Hersteller hat einen Namen und ein Gründungsjahr.

```
CREATE TABLE manufacturers (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  year INT NOT NULL
);
```

4. Jetzt wo die oberen Tabellen erstellt worden sind. Welche Tabelle muss angepasst werden damit eine Verbindung zwischen beiden Tabellen entsteht? Schreibe die Tabelle, welche angepasst werden muss, nochmals neu hier unten dran:
   Nimm dir Zeit diese Aufgabe zu lösen.

```
CREATE TABLE cars (
  ...,
  manufacturer_id INT NOT NULL REFERENCES manufacturers(id),
  ...
);
```

Draw Table (One to Many)

1.

```
CREATE TABLE schulen (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  ort VARCHAR(255)
);

CREATE TABLE gebäude (
  id SERIAL PRIMARY KEY,
  schule_id INT NOT NULL REFERENCES schulen(id),
  nummer INT,
  farbe VARCHAR(255)
);

CREATE TABLE zimmer (
  id SERIAL PRIMARY KEY,
  gebäude_id INT NOT NULL REFERENCES gebäude(id),
  m2 INT,
  ausrichtung VARCHAR(255)
);
```

2.

```
CREATE TABLE kunden (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  ort VARCHAR(255)
);

CREATE TABLE konti (
  id SERIAL PRIMARY KEY,
  kunde_id INT NOT NULL REFERENCES kunden(id),
  nummer INT,
  währung VARCHAR(255)
);

CREATE TABLE transaktionen (
  id SERIAL PRIMARY KEY,
  konto_id INT NOT NULL REFERENCES konti(id),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

Many to Many

1.

```
CREATE TABLE schauspieler (
  id SERIAL PRIMARY KEY,
);

CREATE TABLE schau_filme (
  id SERIAL PRIMARY KEY,
  schau_id INT NOT NULL REFERENCES schauspieler(id),
  film_id INT NOT NULL REFERENCES filme(id),
);

CREATE TABLE filme (
  id SERIAL PRIMARY KEY,
  regisseur_id INT NOT NULL REFERENCES regisseur(id),
  length INT,
);

CREATE TABLE regisseur (
  id SERIAL PRIMARY KEY,
);
```

2.

```
CREATE TABLE keys (
  id SERIAL PRIMARY KEY,
  teacher_id INT NOT NULL REFERENCES teachers(id),
);

CREATE TABLE keys_doors (
  id SERIAL PRIMARY KEY,
  key_id INT NOT NULL REFERENCES keys(id),
  door_id INT NOT NULL REFERENCES doors(id),
);

CREATE TABLE doors (
  id SERIAL PRIMARY KEY,
);

CREATE TABLE teachers (
  id SERIAL PRIMARY KEY,
);
```

Test 1

Wann

06./07. März 2025

Was

Kapitel 1 - 7

Infos

Ihr benötigt:

- Einseitiges A5 CheatSheet (von Hand geschrieben im Original)

Vorbereitungsaufgaben

- Übungstest cars
- Test Vorbereitung users

cars

*Die **cars** Tabelle findest du hier*

Erhalte die Anzahl Autos, welche in der Tabelle, sind zurück.

```
CODE
```

Welche **Zahl** erhalte ich bei folgendem Query:

```
SELECT AVG(hp) FROM cars
WHERE brand = 'Kia';
```

```
CODE
```

Füge einen neues Auto zur Tabelle hinzu: Ein blauer VW aus dem Jahr 2009.

```
CODE
```

Erhalte die Anzahl der Autos gruppiert nach Farbe. Aber exclusive deren mit Jahrgang 2020 oder älter.

```
CODE
```

Welche **Zahl** erhalte ich bei folgendem Query:

```
SELECT year FROM cars
WHERE brand = 'VW' OR hp >= 95
ORDER BY id
LIMIT 1;
```

```
CODE
```

Lösche alle Autos deren Name zwei kleine D's ('dd') enthalten.

```
CODE
```

Sortiere alle Autos nach "brand" und innerhalb des brands nach "hp".

```
CODE
```

Ändere beim Auto "Fanionter" die Farbe zu yellow.

```
CODE
```

Welche **4 Zahlen** erhalte ich bei folgendem Query:

```
SELECT hp, year FROM cars
ORDER BY brand ASC, color DESC
LIMIT 2;
```

```
CODE
```

Erhalte alle Autos mit dem Jahrgang 2001.

```
CODE
```

cars Lösungen

1.

```
SELECT COUNT(id) FROM cars;
```

2.

100

3.

```
INSERT INTO cars
  (brand, color, year)
VALUES
  ('VW', 'blue', 2009);
```

4.

```
SELECT COUNT(id), color
FROM cars
WHERE year > 2020
GROUP BY color;
```

5.

2010

6.

```
DELETE FROM cars
WHERE name = '%dd%';
```

7.

```
SELECT * FROM cars
ORDER BY brand, hp;
```

8.

```
UPDATE cars
SET color = 'yellow'
WHERE code_name = 'Fanionter';
```

9.

80, 2005
255, 2010

10.

```
SELECT * FROM cars
WHERE year = 2001;
```

Vorbereitungstest mit users table

*Die **users** Tabelle findest du hier*

1. Erhalte den Namen und Id aller Users.

```
CODE
```

2. Was erhalte ich bei folgendem Query?

```
SELECT country
FROM users
WHERE name = 'Sabrina';
```

```
CODE
```

3. Ändere das Alter von User 2 zu 26.

```
CODE
```

4. Welche Zahl erhalte ich bei folgendem Query?

```
SELECT MIN(age)
FROM users
WHERE country = 'UK' OR country = 'IT';
```

```
CODE
```

5. Löschen alle User deren Stadt mit "d" oder "n" enden.

```
CODE
```

6. Füge den neuen User Pumba, 28 aus UK hinzu.

```
CODE
```

7. Welche Zahlen erhalte ich bei folgendem Query?

```
SELECT age
FROM users
ORDER BY age DESC
LIMIT 3;
```

```
CODE
```

8. Schreibe ein Query bei dem ich die Summe aller ids pro Land erhalte.

```
CODE
```

9. Welche Zahl erhalte ich bei folgendem Query?

```sql
SELECT AVG(age)
FROM users
WHERE country LIKE 'U%';
```

```
CODE
```

10. Sortiere alle User nach Alter.

```
CODE
```

11. Welcher 2 name erhalte ich bei folgendem Query?

```sql
SELECT name
FROM users
WHERE age >= 25
ORDER BY id DESC
LIMIT 2
OFFSET 1;
```

```
CODE
```

1. Erhalte den Namen und Id aller Users.

```sql
SELECT name, id FROM users;
```

2. Was erhalte ich bei folgendem Query?

```sql
SELECT country
FROM users
WHERE name = 'Sabrina';
```

```
IT, CH
```

3. Ändere das Alter von User 2 zu 26.

```sql
UPDATE users
SET age = 26
WHERE id = 2;
```

4. Welche Zahl erhalte ich bei folgendem Query?

```sql
SELECT MIN(age)
FROM users
WHERE country = 'UK' OR country = 'IT';
```

```
26
```

5. Löschen alle User deren Stadt mit "d" oder "n" enden.

```sql
DELETE FROM users
WHERE city LIKE '%d'
   OR city LIKE '%n';
```

6. Füge den neuen User Pumba, 28 aus UK hinzu.

```sql
INSERT INTO users
  (name, age, country)
VALUES
  ('Pumba', 28, 'UK');
```

7. Welche Zahlen erhalte ich bei folgendem Query?

```sql
SELECT age
FROM users
ORDER BY age DESC
LIMIT 3;
```

```
32, 30, 28
```

8. Schreibe ein Query bei dem ich die Summe aller ids pro Land erhalte.

```sql
SELECT SUM(id), country
FROM users
GROUP BY country;
```

9. Welche Zahl erhalte ich bei folgendem Query?

```sql
SELECT AVG(age)
FROM users
WHERE country LIKE 'U%';
```

```
21.5
```

10. Sortiere alle User nach Alter.

```sql
SELECT * FROM users
ORDER BY age;
```

11. Welcher 2 name erhalte ich bei folgendem Query?

```sql
SELECT name
FROM users
WHERE age >= 25
ORDER BY id DESC
LIMIT 2
OFFSET 1;
```

```
Thomas, Tessa
```

Test 2

Wann

24./25. April 2025

Was

- Kapitel 8-10
  - create-table
  - draw-table
  - many-to-many

Infos

Ihr benötigt:

- Stifte

Theorie

Als Hilfe nutzt folgende Theorieteile:

- sql-intro
- Kapitel 8-10
  - create-table
  - draw-table
  - many-to-many

Tables - users

id	name	age	city	country
1	Agate	18	Biel	CH
2	Ruben	32	[NULL]	IT
3	Leopold	17	Biel	CH
5	Sara	28	Bern	CH
6	Sabrina	28	Pescara	IT
7	Viktor	26	Bern	CH
8	Joe	26	[NULL]	UK
9	Melanie	28	Thun	CH
10	Oceane	25	[NULL]	FR
11	Kelly	17	[NULL]	US
13	Sabrina	21	Zuzwil	CH
14	Tessa	30	Apeldoorn	NL
15	Manuela	22	Thun	CH
16	Pavel	22	Orpund	CH
17	Mauro	16	Thun	CH
19	Louise	17	Paris	FR
20	Thomas	28	[NULL]	DE
21	Matteo	26	Orpund	CH

cars

id	code_name	brand	color	hp	year
3	Cyanretch	Ford	silver	80	2005
4	Fuddyduddy	Jaguar	silver	90	2011
6	Chartreuse	VW	black	120	2010
7	Wigout	Ford	gray	65	2015
8	Hedonism	VW	black	220	2017
10	Cacophony	Mazda	blue	75	2022
12	Iconomachy	Kia	blue	120	2001
13	Jetsam	Mazda	[NULL]	80	1998
15	Curieman	Ford	silver	255	2010
16	Widdershins	Mazda	black	180	2005
17	Luciphyllous	Ford	black	95	2004
18	Jiffy	Kia	gray	80	2017
20	Delative	Ford	gray	120	2017
21	Snuffle	Mazda	violet	180	2019
25	Fanionter	Mazda	[NULL]	70	2006
26	Madcap	VW	red	100	2012
28	Hexeris	VW	silver	105	2009

Tables - posts

todo