DATA DRIVEN DECISIONS

Sommersemester 2026

Datenmanagement & -analyse

Prof. Dr. Christoph M. Flath

Data Driven Decisions Group, Universität Würzburg

Agenda

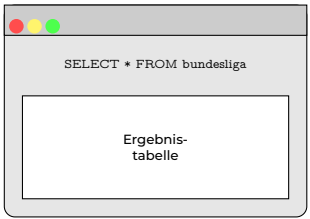
1 Kursüberblick & Motivation
2 Erste SELECT-Abfragen
3 Die WHERE-Klausel
4 Filtern mit Bedingungen
5 Logische Operatoren
6 Komplexe Abfragen
7 Erste Visualisierungen
8 Zusammenfassung

Theorie und Praxis wechseln sich ab.

Wie arbeiten wir?

In der Vorlesung

- Konzepte auf Folien
- Live-Demos im Notebook
- Sie coden mit!



Unsere Werkzeuge

marimo Interaktive Python-Notebooks im Browser

SQLite Datenbank ohne Server-Setup

Kein Setup nötig

marimo läuft direkt im Browser – auch auf dem Tablet!

Wer nutzt eigentlich Datenbanken?

DATA DRIVEN DECISIONS

Spoiler: Fast jeder – auch Sie, jeden Tag.

Ihr Smartphone Kontakte, Nachrichten, Fotos

Netflix/Spotify 200+ Mio. Nutzerprofile

Instagram 2+ Mrd. Bilder, Likes

Ihre Bank Konten, Überweisungen

Amazon 12+ Mio. Produkte

Die Uni Noten, Stundenpläne

Die Gemeinsamkeit

Überall stecken **relationale Datenbanken** – und **SQL** ist die Sprache, um mit ihnen zu sprechen.

Warum SQL lernen?

Arbeitsmarkt

- #1 nachgefragte Daten-Kompetenz
- Grundlage für Business Intelligence
- Jede Data-Science-Stelle setzt SQL voraus

Gehalt (Durchschnitt DE)

Data Analyst: 55.000/Jahr
Mit SQL-Expertise: +15%

Für Ihr Studium

- Abschlussarbeiten mit echten Daten
- Praktika in Controlling, Marketing
- Selbstständige Datenauswertung

Fun Fact

SQL wurde 1974 entwickelt – älter als die meisten Programmiersprachen, aber relevanter denn je!

Worum geht es in diesem Kurs?

DATA DRIVEN DECISIONS

Datenmanagement & -analyse verbindet zwei Welten:

Datenmanagement

- Wie speichern wir Daten effizient?
- Wie vermeiden wir Fehler und Redundanz?
- Wie greifen wir auf Daten zu?

Datenanalyse

- Wie explorieren wir Daten?
- Wie erkennen wir Muster?
- Wie ziehen wir Schlüsse?

Die Verbindung

SQL ist die gemeinsame Sprache – sowohl für Datenbankabfragen als auch für analytische Auswertungen.

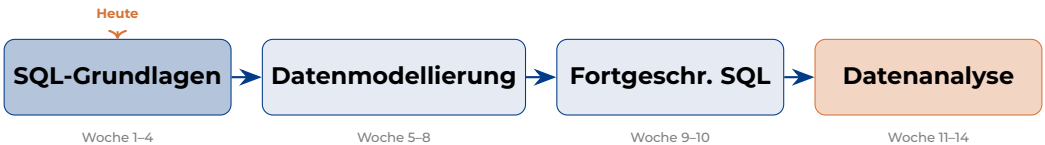
Was Sie am Ende können werden

1. Konzeptionelle und relationale Datenmodelle **erstellen und bewerten**
2. Komplexe SQL-Abfragen zur Datenextraktion **formulieren**
3. Strukturierte Datenanalyseprozesse **anwenden**
4. Explorative Datenanalysen **durchführen und visualisieren**
5. Zeitreihen- und Textdaten **analysieren**
6. Statistische Hypothesentests und A/B-Tests **interpretieren**

Heute

Wir starten mit dem Fundament: **SQL-Grundlagen** – SELECT, FROM, WHERE

Der Kurs im Überblick



Roter Faden

SQL begleitet uns durch den gesamten Kurs – von einfachen Abfragen bis zu komplexen Analysen.

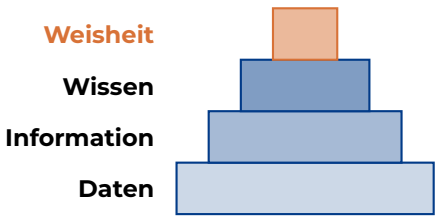
Von Daten zu Wissen

Daten Rohe Fakten ohne Kontext
„42, 38, 35, 33, 31, …"

Information Daten mit Bedeutung
„Bayern hat 42 Punkte, Dortmund 38"

Wissen Information + Interpretation
„Bayern führt mit 4 Punkten Vorsprung"

SQL hilft uns, aus Daten Information zu extrahieren.



Was ist eine Datenbank?

Eine **Datenbank** ist eine organisierte Sammlung von Daten.

Ohne Datenbank: ✗

- Excel-Dateien auf Laufwerken
- Wer hat die aktuelle Version?
- Wie verknüpfe ich Daten?
- Was passiert bei Fehlern?

Mit Datenbank: ✓

- Zentrale Datenhaltung
- Konsistenz durch Regeln
- Mächtige Abfragesprache
- Mehrbenutzerzugriff

Relationale Datenbank

Speichert Daten in **Tabellen** mit Zeilen und Spalten. Beziehungen zwischen Tabellen werden durch **Schlüssel** hergestellt.

Tabellen: Die Grundbausteine

Spalte

Mannschaft	Spiele	Siege	Punkte	Tore
Bayern	17	13	41	53:18
Leverkusen	17	12	39	41:22
Stuttgart	17	10	34	40:25

Zeile

- Jede **Zeile** ist ein Datensatz (z.B. eine Mannschaft)
- Jede **Spalte** ist ein Attribut (z.B. Punkte, Tore)
- Jede **Zelle** enthält einen Wert

Was ist SQL?

DATA DRIVEN DECISIONS

SQL = Structured Query Language („Sequel”)

- Standardsprache für relationale Datenbanken
- Entwickelt in den 1970ern bei IBM
- **Deklarativ**: Wir sagen *was* wir wollen, nicht *wie*
- Läuft auf allen gängigen Systemen

Beispiel

```
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE Punkte > 30;
```

„Zeige mir Mannschaft und Punkte aller Teams mit mehr als 30 Punkten.”

Unser Beispiel: Die Bundesliga-Tabelle

Warum Bundesliga?

- Bekannte Domäne – jeder kennt die Regeln
- Strukturierte Daten – perfekt für SQL
- Interessante Fragen:
  - Wer wird Meister?
  - Wer steigt ab?
  - Welches Team schiesst die meisten Tore?

Live-Daten

Unsere Notebooks laden die **aktuelle** Bundesliga-Tabelle aus dem Internet!

Die SELECT-Anweisung: Grundstruktur

DATA DRIVEN DECISIONS

Syntax

```
SELECT spalte1, spalte2, ...
FROM tabelle;
```

Alle Spalten:

```
SELECT *
FROM bundesliga;
```

→ Zeigt die gesamte Tabelle

Bestimmte Spalten:

```
SELECT Mannschaft, Punkte
FROM bundesliga;
```

→ Nur diese 2 Spalten

Merke

∗ bedeutet „alle Spalten" – praktisch zum Erkunden, aber in Produktionscode besser explizit sein.

Aufgabe: Zeige alle Mannschaften mit ihren Punkten.

1. **Was wollen wir sehen?** → Mannschaft und Punkte
2. **Woher kommen die Daten?** → Tabelle `bundesliga`

```
SELECT Mannschaft, Punkte
FROM bundesliga;
```

Mannschaft	Punkte
Bayern München	41
Bayer Leverkusen	39
VfB Stuttgart	34
...	...

Häufige Anfängerfehler bei SELECT

DATA DRIVEN DECISIONS

✗ Falsch

Spaltenname falsch geschrieben:

```
SELECT Manschaft, Punkte
FROM bundesliga;
```

Error: no such column

Komma vergessen:

```
SELECT Mannschaft Punkte
FROM bundesliga;
```

Syntax error

✓ Richtig

```
SELECT Mannschaft, Punkte
FROM bundesliga;
```

Tipp

SQL ist case-insensitive – `select` = `SELECT`. Wir schreiben Schlüsselwörter trotzdem groß zur besseren Lesbarkeit.

Hands-on

Erste SELECT-Abfragen

marimo: 01-sql-grundlagen.py

Aufgaben 2.1 – 2.8

Scaffolded → Selbstständig → Debugging

Zeilen filtern mit WHERE

Problem: Wir wollen nicht *alle* Zeilen, sondern nur bestimmte.

Syntax

```
SELECT spalte1, spalte2, ...
FROM tabelle
WHERE bedingung;
```

Beispiel: Nur Top-Teams

```
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE Punkte > 30;
```

Ergebnis: Nur Zeilen, bei denen die Bedingung `TRUE` ist.

WHERE Schritt für Schritt

Aufgabe: Finde alle Teams mit negativer Tordifferenz.

1. **Was zeigen?** → Mannschaft, Tordifferenz
2. **Woher?** → bundesliga
3. **Welche Zeilen?** → Tordifferenz kleiner als 0

```
SELECT Mannschaft, Tordifferenz
FROM bundesliga
WHERE Tordifferenz < 0;
```

Mannschaft	Tordifferenz
Werder Bremen	-3
VfL Wolfsburg	-3
Holstein Kiel	-27
...	...

Vergleichsoperatoren

Operator	Bedeutung	Beispiel
=	gleich	Spiele = 17
<> oder !=	ungleich	Mannschaft <> 'Bayern'
<	kleiner als	Niederlagen < 3
>	größer als	Punkte > 30
<=	kleiner oder gleich	Niederlagen <= 5
>=	größer oder gleich	Siege >= 10

Achtung: Gleichheit

In SQL ist = der Gleichheitsoperator (nicht == wie in Python).

Beispiele: Vergleichsoperatoren

Numerische Vergleiche

```
-- Mindestens 10 Siege
SELECT Mannschaft, Siege
FROM bundesliga
WHERE Siege >= 10;
```

```
-- Weniger als 20 Punkte
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE Punkte < 20;
```

Exakte Werte

```
-- Genau 17 Spiele
SELECT Mannschaft
FROM bundesliga
WHERE Spiele = 17;
```

```
-- Alle ausser 0 Niederlagen
SELECT Mannschaft
FROM bundesliga
WHERE Niederlagen <> 0;
```

Bei Texten (Strings) werden Werte in **Anführungszeichen** gesetzt:

```
SELECT *
FROM bundesliga
WHERE Mannschaft = 'Bayern München';
```

Gross-/Kleinschreibung

Je nach Datenbank:
'Bayern' ≠ 'bayern'
SQLite: meist case-insensitive

Anführungszeichen

'einfach' für Textwerte
"doppelt" für Spaltennamen (optional)

Häufige WHERE-Fehler

✗ Falsch

Text ohne Anführungszeichen:

```
WHERE Mannschaft = Bayern
```

Error: no such column: Bayern

Falscher Operator für Text:

```
WHERE Mannschaft > 'B'
```

Funktioniert, aber alphabetisch!

✓ Richtig

```
WHERE Mannschaft = 'Bayern'
```

Merkregel

Zahlen: ohne Quotes
Text: mit 'einfachen' Quotes

Diskutieren Sie mit Ihrem Nachbarn (2 Minuten):

1. Welche Spalten hat unsere Bundesliga-Tabelle?
2. Welche Fragen könnten Sie mit WHERE beantworten?

Beispiel-Ideen:

- Welche Teams haben mehr als 40 Tore geschossen?
- Welche Teams haben weniger als 3 Unentschieden?
- Welches Team hat genau 30 Punkte?

Hands-on

Filtern mit WHERE

marimo: 01-sql-grundlagen.py

Aufgaben 4.1 – 4.8

Inkl. Vorhersage-Aufgaben und Debugging

Pause

15 Minuten

Kaffee holen, Fragen notieren, kurz bewegen!

Agenda

1 Kursüberblick & Motivation
2 Erste SELECT-Abfragen
3 Die WHERE-Klausel
4 Filtern mit Bedingungen
▸ **5 Logische Operatoren**
6 Komplexe Abfragen
7 Erste Visualisierungen
8 Zusammenfassung

Mehrere Bedingungen: AND, OR, NOT

Problem: Eine einzelne Bedingung reicht oft nicht aus.

Operator	Bedeutung
`AND`	**Beide** Bedingungen müssen wahr sein
`OR`	**Mindestens eine** Bedingung muss wahr sein
`NOT`	**Negiert** die Bedingung

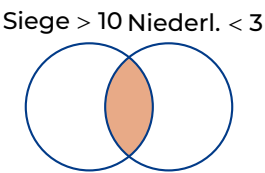
Alltagsbeispiel

„Ich möchte ein Restaurant, das **italienisch** ist **UND** weniger als 20 kostet."
→ Beide Kriterien müssen erfüllt sein (AND)

AND: Beide Bedingungen müssen gelten


```
SELECT Mannschaft, Siege, Niederlagen
FROM bundesliga
WHERE Siege > 10 AND Niederlagen < 3;
```

Bedeutung: Teams mit mehr als 10 Siegen **UND** weniger als 3 Niederlagen.

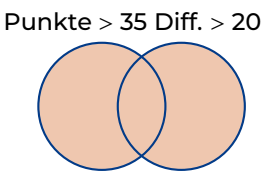


Nur die Schnittmenge

OR: Mindestens eine Bedingung muss gelten


```
SELECT Mannschaft, Punkte, Tordifferenz
FROM bundesliga
WHERE Punkte > 35 OR Tordifferenz > 20;
```

Bedeutung: Teams mit mehr als 35 Punkten **ODER** Tordifferenz > 20.



Alles was markiert ist

NOT: Bedingung negieren


```
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE NOT Mannschaft = 'Bayern München';
```

Bedeutung: Alle Teams **ausser** Bayern München.

Äquivalent:

```
WHERE Mannschaft <> 'Bayern'
```

NOT ist nützlich bei:

- NOT LIKE
- NOT IN
- NOT BETWEEN

Wahrheitstabellen

AND

A	B	A AND B
T	T	**T**
T	F	F
F	T	F
F	F	F

Nur wenn **beide** wahr

OR

A	B	A OR B
T	T	**T**
T	F	**T**
F	T	**T**
F	F	F

Wenn **mindestens eine** wahr

NOT

A	NOT A
T	F
F	**T**

Dreht um

Merkregel

AND ist restriktiv (weniger Ergebnisse)
OR ist permissiv (mehr Ergebnisse)

Klammern: Reihenfolge kontrollieren

DATA DRIVEN DECISIONS

Achtung: Priorität

AND bindet stärker als OR!

```
-- Was bedeutet das?
WHERE Punkte > 30 OR Punkte > 20 AND Siege > 10
```

Klammern: Reihenfolge kontrollieren

DATA DRIVEN DECISIONS

Achtung: Priorität

`AND` bindet stärker als `OR`!

```
-- Was bedeutet das?
WHERE Punkte > 30 OR Punkte > 20 AND Siege > 10
```

Wird interpretiert als:

```
WHERE Punkte > 30 OR (Punkte > 20 AND Siege > 10)
```

Besser: Explizite Klammern

```
WHERE (Punkte > 30 OR Punkte > 20) AND Siege > 10
```

Tipp: Im Zweifel **immer Klammern setzen**!

BETWEEN: Wertebereiche


```
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE Punkte BETWEEN 20 AND 30;
```

Bedeutung: Punkte zwischen 20 und 30 **(inklusiv!)**.

Äquivalent zu:

```
WHERE Punkte >= 20
  AND Punkte <= 30
```

Auch für Text:

```
WHERE Mannschaft
  BETWEEN 'A' AND 'M'
```

(alphabetisch A-M)

IN: Liste von Werten


```
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE Mannschaft IN ('Bayern München', 'Borussia Dortmund',
                     'RB Leipzig', 'Bayer Leverkusen');
```

Bedeutung: Mannschaft ist einer der aufgelisteten Werte.

Äquivalent zu:

```
WHERE Mannschaft = 'Bayern'
   OR Mannschaft = 'Dortmund'
   OR Mannschaft = 'Leipzig'
   OR ...
```

Vorteil:

- Kürzerer Code
- Leichter lesbar
- Weniger Fehleranfällig

LIKE: Mustersuche

Wildcard	Bedeutung
%	Beliebig viele Zeichen (auch 0)
_	Genau ein Zeichen

```
-- Beginnt mit 'B'
WHERE Mannschaft LIKE 'B%'
```

Bayern, Bochum, Bremen, ...

```
-- Endet mit 'en'
WHERE Mannschaft LIKE '%en'
```

Bremen, München, ...

```
-- Enthält 'burg'
WHERE Mannschaft LIKE '%burg%'
```

Augsburg, Freiburg, Hamburg

```
-- Zweiter Buchstabe ist 'a'
WHERE Mannschaft LIKE '_a%'
```

Bayern, ...

Welche Abfrage findet mehr Ergebnisse?

1. `WHERE Punkte > 30 AND Siege > 10`
2. `WHERE Punkte > 30 OR Siege > 10`

Welche Abfrage findet mehr Ergebnisse?

1. `WHERE Punkte > 30 AND Siege > 10`
2. `WHERE Punkte > 30 OR Siege > 10`

Antwort

2 (OR) findet mehr Ergebnisse!

- AND: Beide Bedingungen müssen erfüllt sein → restriktiv
- OR: Eine reicht → permissiv

Hands-on

Komplexe Abfragen

marimo: 01-sql-grundlagen.py

Aufgaben 6.1 – 6.12 + Freie Exploration

40 Minuten – nehmen Sie sich Zeit!

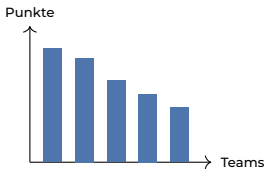
SQL liefert Daten – Visualisierung macht sie verständlich.

Kernidee: Werte auf x- und y-Achse abbilden

- `x="Mannschaft"` – Kategorien auf x-Achse
- `y="Punkte"` – Werte auf y-Achse

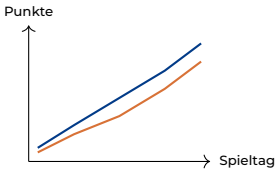
Zwei Datenstrukturen – Zwei Charttypen

Querschnitt (Tabelle)



- 18 Teams, 1 Zeitpunkt
- “Wer führt?”
- ⇒ **Balkendiagramm**

Zeitreihe (Verlauf)



- 1 Team, 34 Spieltage
- “Wie entwickelt es sich?”
- ⇒ **Liniendiagramm**

Beispiel: Balkendiagramm

SQL-Abfrage:

```sql
SELECT Mannschaft, Punkte
FROM bundesliga
WHERE Punkte > 50;
```

Visualisierung mit plotly:

```python
px.bar(ergebnis,
    x="Mannschaft",
    y="Punkte"
)
```

[Interaktiver Chart im Notebook]

Zeigt: Top-Teams nach Punkten

Beispiel: Liniendiagramm

SQL-Abfrage:

```sql
SELECT Spieltag, Punkte_Kumuliert
FROM bundesliga_spieltage
WHERE Mannschaft = 'Bayern';
```

Visualisierung mit plotly:

```python
px.line(ergebnis,
    x="Spieltag",
    y="Punkte_Kumuliert"
)
```

[Interaktiver Chart im Notebook]

Zeigt: Punkteverlauf über die Saison

Vorschau: Mehr Visualisierung

In den nächsten Vorlesungen:

- Mit `ORDER BY` Daten sortieren – dann Top 10 visualisieren
- Mehrere Teams vergleichen (verschiedene Farben)
- Aggregierte Daten (Durchschnitte, Summen) darstellen

Visualisierung begleitet uns durch den ganzen Kurs!

Agenda

Typische Anfängerfehler

Syntax-Fehler

- Semikolon vergessen
- Komma zwischen Spalten vergessen
- Falsche Anführungszeichen
- Spaltenname falsch geschrieben

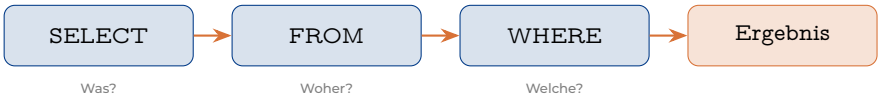
Logik-Fehler

- AND/OR verwechselt
- Klammern vergessen
- `= NULL` statt `IS NULL`
- `>` vs `>=` verwechselt

Keine Sorge!

Fehler gehören zum Lernen. Die Fehlermeldungen in SQL sind meist hilfreich – lesen Sie sie aufmerksam!

Was wir heute gelernt haben



SQL-Grundmuster

```
SELECT spalten FROM tabelle WHERE bedingung;
```

SELECT *	Alle Spalten
WHERE Punkte > 30	Numerischer Vergleich
WHERE Mannschaft = 'Bayern'	Text-Vergleich
AND, OR, NOT	Logische Verknüpfung
BETWEEN, IN, LIKE	Spezial-Operatoren

Was ist das Ergebnis dieser Abfrage?

```
SELECT Mannschaft FROM bundesliga;
```

1. Alle Spalten der Tabelle bundesliga
2. Nur die Spalte Mannschaft für alle Teams
3. Nur Teams, die „Mannschaft" heissen
4. Eine Fehlermeldung

Was ist das Ergebnis dieser Abfrage?

```
SELECT Mannschaft FROM bundesliga;
```

1. Alle Spalten der Tabelle bundesliga
2. Nur die Spalte Mannschaft für alle Teams
3. Nur Teams, die „Mannschaft" heissen
4. Eine Fehlermeldung

Antwort

2 – Die Abfrage zeigt nur die Spalte „Mannschaft" für alle 18 Teams.

Welche Abfrage findet Teams mit 10+ Siegen UND höchstens 5 Niederlagen?

1. `WHERE Siege > 10 OR Niederlagen < 5`
2. `WHERE Siege > 10 AND Niederlagen < 5`
3. `WHERE Siege >= 10 AND Niederlagen <= 5`
4. `WHERE Siege >= 10 OR Niederlagen <= 5`

Welche Abfrage findet Teams mit 10+ Siegen UND höchstens 5 Niederlagen?

1. `WHERE Siege > 10 OR Niederlagen < 5`
2. `WHERE Siege > 10 AND Niederlagen < 5`
3. `WHERE Siege >= 10 AND Niederlagen <= 5`
4. `WHERE Siege >= 10 OR Niederlagen <= 5`

Antwort

3 – „mindestens 10" = `>= 10`, „höchstens 5" = `<= 5`, beide müssen gelten = `AND`.

Welches Muster findet alle Teams mit „Borussia” im Namen?

1. LIKE ’Borussia’
2. LIKE ’Borussia%’
3. LIKE ’%Borussia%’
4. LIKE ’_Borussia_’

Welches Muster findet alle Teams mit „Borussia” im Namen?

1. `LIKE 'Borussia'`
2. `LIKE 'Borussia%'`
3. `LIKE '%Borussia%'`
4. `LIKE '_Borussia_'`

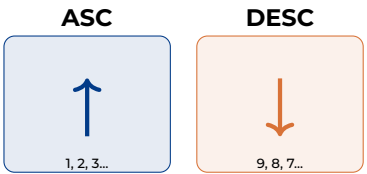
Antwort

3 – `%Borussia%` findet „Borussia” an beliebiger Stelle im Namen (Borussia Dortmund, Borussia M'gladbach).

Konzept	Syntax
Alle Spalten	`SELECT * FROM tabelle`
Bestimmte Spalten	`SELECT spalte1, spalte2 FROM tabelle`
Filtern	`WHERE bedingung`
Vergleiche	`=, <>, <, >, <=, >=`
Und-Verknüpfung	`WHERE a AND b`
Oder-Verknüpfung	`WHERE a OR b`
Negation	`WHERE NOT a`
Wertebereich	`WHERE x BETWEEN 10 AND 20`
Werteliste	`WHERE x IN ('a', 'b', 'c')`
Mustersuche	`WHERE x LIKE 'B%'`

Vorlesung 2: SQL für Datenexploration

- Ergebnisse **sortieren** mit `ORDER BY`
- **Eindeutige Werte** mit `DISTINCT`
- Ergebnisse **begrenzen** mit `LIMIT`
- Umgang mit **fehlenden Werten** (`NULL`)



Vorbereitung

Spielen Sie mit den Bundesliga-Daten!
Stellen Sie eigene Fragen.

Fragen?

marimo: 01-sql-grundlagen.py

Weiterarbeiten und experimentieren!