

Sommersemester 2026

Datenmanagement & -analyse

Prof. Dr. Christoph M. Flath

Lehrstuhl für Wirtschaftsinformatik und Business Analytics

Julius-Maximilians-Universität Würzburg

- 1 Rückblick & ORDER BY
- 2 DISTINCT & LIKE
- 3 NULL-Werte verstehen
- 4 Visualisierung: Streudiagramme
- 5 Zusammenfassung

Lernziele

Daten sortieren, Duplikate entfernen, Muster finden, mit fehlenden Werten umgehen.

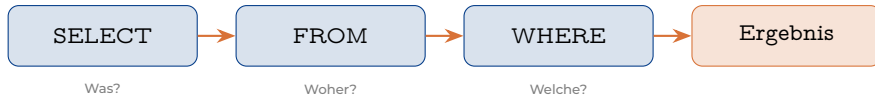
► **1 Rückblick & ORDER BY**

2 DISTINCT & LIKE

3 NULL-Werte verstehen

4 Visualisierung: Streudiagramme

5 Zusammenfassung



- SELECT – Spalten auswählen
- WHERE – Zeilen filtern
- AND, OR, NOT – Bedingungen kombinieren
- BETWEEN, IN, LIKE – Weitere Filter

Heute

Wie **ordnen** wir die Ergebnisse? Wie finden wir **eindeutige** Werte? Was passiert bei **fehlenden** Daten?

Letzte Woche: Finale Tabelle (1 Zeitpunkt, 18 Teams)

Diese Woche: Alle 34 Spieltage – der **Verlauf** der Saison!

```
SELECT * FROM bundesliga_spieltage LIMIT 5;
```

Spieltag	Mannschaft	Punkte_Spiel	Punkte_Kumuliert	...
1	Bayern München	3	3	...
2	Bayern München	3	6	...
3	Bayern München	1	7	...
...

Erkenntnis

Die finale Tabelle ist nur WHERE Spieltag = 34!

Finale Tabelle rekonstruieren:

```
SELECT Mannschaft, Punkte_Kumuliert AS Punkte  
FROM bundesliga_spieltage  
WHERE Spieltag = 34  
ORDER BY Punkte DESC;
```

Oder: Tabelle nach Spieltag 17 (Winterpause)?

```
SELECT Mannschaft, Punkte_Kumuliert AS Punkte  
FROM bundesliga_spieltage  
WHERE Spieltag = 17  
ORDER BY Punkte DESC;
```

⇒ Zeigt, wer zur Halbzeit vorne lag!

Stellen Sie sich vor: Sie haben 1000 Kunden und wollen die Top 10 nach Umsatz.

Ohne Sortierung:

- Daten in zufälliger Reihenfolge
- Manuelles Durchsuchen nötig
- Top-Werte nicht erkennbar

Mit Sortierung:

- Höchste Werte zuerst
- Sofortige Übersicht
- Muster erkennbar

Bundesliga-Beispiel

Wer ist Tabellenführer? → Nach Punkten sortieren!

Wer steigt ab? → Nach Punkten aufsteigend, letzte 3 ansehen.

Syntax

```
SELECT spalten  
FROM tabelle  
ORDER BY spalte [ASC|DESC];
```

Aufsteigend (Standard)

```
SELECT Mannschaft, Punkte  
FROM bundesliga  
ORDER BY Punkte ASC;
```

10, 15, 20, 25, ...

Absteigend

```
SELECT Mannschaft, Punkte  
FROM bundesliga  
ORDER BY Punkte DESC;
```

50, 45, 40, 35, ...

Merke

ASC = ascending (aufsteigend) – ist der **Default**
DESC = descending (absteigend)

Aufgabe: Zeige die Tabelle sortiert nach Punkten (beste zuerst).

- ① **Was zeigen?** → Mannschaft, Punkte
- ② **Woher?** → bundesliga
- ③ **Wie sortieren?** → Nach Punkte, absteigend

```
SELECT Mannschaft , Punkte  
FROM bundesliga  
ORDER BY Punkte DESC;
```

Mannschaft	Punkte
Bayern München	50
Borussia Dortmund	42
VfB Stuttgart	36
...	...

Was passiert bei Gleichstand?

```
SELECT Mannschaft, Punkte, Tordifferenz  
FROM bundesliga  
ORDER BY Punkte DESC, Tordifferenz DESC;
```

Mannschaft	Punkte	Tordifferenz
Bayern	50	+56
Dortmund	42	+21
Hoffenheim	36	+16
Stuttgart	36	+10

Erklärung

Erst nach Punkten sortiert, bei Gleichstand (36) entscheidet die Tordifferenz.
Hoffenheim vor Stuttgart wegen besserer Tordifferenz!

Zahlen sortieren:

ORDER BY Punkte DESC

→ 50, 42, 36, 35, ...

Numerisch korrekt!

Text sortieren:

ORDER BY Mannschaft ASC

→ Augsburg, Bayern, Bochum, ...

Alphabetisch!

Achtung bei Text

- Gross-/Kleinschreibung kann relevant sein
- Umlaute: ä, ö, ü werden je nach Datenbank unterschiedlich sortiert
- Zahlen als Text: '10' kommt vor '2' (weil '1' < '2')

```
SELECT Mannschaft, Punkte  
FROM bundesliga  
ORDER BY Punkte DESC  
LIMIT 5;
```

→ Die **Top 5** Teams nach Punkten

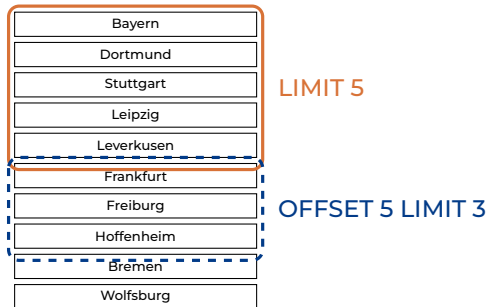
Typische Anwendungen:

- Top 10 Listen
- Stichproben ziehen
- Performance (grosse Tabellen)
- Debugging (schneller Überblick)

Mit OFFSET:

```
SELECT Mannschaft  
FROM bundesliga  
ORDER BY Punkte DESC  
LIMIT 5 OFFSET 5;
```

→ Plätze 6-10



```
-- Top 5
SELECT * FROM bundesliga ORDER BY Punkte DESC LIMIT 5;

-- Plätze 6-8 (überspringe 5, nimm 3)
SELECT * FROM bundesliga ORDER BY Punkte DESC LIMIT 3 OFFSET 5;
```

X Falsch:

Sortierung vor WHERE:

```
SELECT Mannschaft  
ORDER BY Punkte  
FROM bundesliga;
```

Syntax Error!

Spalte nicht im SELECT:

```
SELECT Mannschaft  
FROM bundesliga  
ORDER BY Punkte;
```

Funktioniert meist, aber unübersichtlich

✓ Richtig:

```
SELECT Mannschaft, Punkte  
FROM bundesliga  
ORDER BY Punkte DESC;
```

Reihenfolge

SELECT → FROM → WHERE →
ORDER BY → LIMIT

Diskutieren Sie mit Ihrem Nachbarn (2 Minuten):

- 1 Wann würden Sie ASC verwenden, wann DESC?
- 2 Was passiert, wenn Sie nach einer Spalte sortieren, die nicht im SELECT steht?
- 3 Wie würden Sie die “schlechtesten 3 Teams” finden?

Lösungsideen

- ASC: Alphabetisch, kleinste Werte zuerst, älteste Daten
- DESC: Ranglisten, neueste Daten, höchste Werte
- Schlechteste 3: ORDER BY Punkte ASC LIMIT 3

Hands-on

Daten sortieren

marimo: 02-sql-exploration.py

Aufgaben 2.1 – 2.8

Geführt → Scaffolded → Selbstständig → Debugging

- 1 Rückblick & ORDER BY
- ▶ **2 DISTINCT & LIKE**
- 3 NULL-Werte verstehen
- 4 Visualisierung: Streudiagramme
- 5 Zusammenfassung

Problem

Welche verschiedenen Werte gibt es in einer Spalte?

Ohne DISTINCT:

```
SELECT Spiele  
FROM bundesliga;
```

19, 19, 18, 19, 18, 19, ...
(18 Zeilen, viele Duplikate)

Mit DISTINCT:

```
SELECT DISTINCT Spiele  
FROM bundesliga;
```

18, 19
(nur 2 verschiedene Werte)

Typische Anwendungen

- Welche Kategorien gibt es?
- Welche Länder sind vertreten?
- Welche Positionen existieren bei den Spielern?

Aufgabe: Welche verschiedenen Spielstände (Anzahl Spiele) gibt es?

- 1 **Was suchen wir?** → Verschiedene Werte in "Spiele"
- 2 **Duplikate entfernen?** → Ja, mit DISTINCT
- 3 **Sortieren?** → Optional, aber hilfreich

```
SELECT DISTINCT Spiele  
FROM bundesliga  
ORDER BY Spiele;
```

Spiele
18
19

Erkenntnis: Nicht alle Teams haben gleich viele Spiele absolviert!

```
SELECT DISTINCT Siege , Niederlagen  
FROM bundesliga ;
```

→ Alle **Kombinationen** von Siege und Niederlagen

Siege	Niederlagen
16	1
12	1
11	4
11	5
...	...

Wichtig

Jede *Kombination* ist eindeutig, nicht jede Spalte einzeln!
(11, 4) und (11, 5) sind verschiedene Kombinationen.

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

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

Bayern, Bochum, Bremen, Borussia...

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

Augsburg, Freiburg, Wolfsburg

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

Bremen, München, Leverkusen...

```
-- Genau 6 Zeichen  
WHERE Mannschaft LIKE '_____'
```

Bayern (6 Buchstaben)

Zweiter Buchstabe ist 'a':

```
WHERE Mannschaft LIKE '_a%'
```

Bayern, Hamburg (falls vorhanden)

Enthält Zahl:

```
WHERE Mannschaft LIKE '%1%'
```

1. FC Union Berlin, 1. FSV Mainz 05, 1. FC Heidenheim

Beginnt mit 'B' oder 'F':

```
WHERE Mannschaft LIKE 'B%'
OR Mannschaft LIKE 'F%'
```

Bayern, Bremen, Bochum, Frankfurt, Freiburg...

NOT LIKE:

```
WHERE Mannschaft
NOT LIKE '%FC%'
```

Alle ohne "FC" im Namen

Achtung

Je nach Datenbank ist LIKE case-sensitive oder nicht!

DuckDB / PostgreSQL:

- LIKE ist case-**sensitive**
- ILIKE ist case-**insensitive**

SQLite:

- LIKE ist case-**insensitive**
- GLOB ist case-**sensitive**

Sichere Lösung (funktioniert überall)

```
WHERE LOWER(Mannschaft) LIKE LOWER( '%bayern%' )
```

Beide Seiten in Kleinbuchstaben umwandeln!

Schnelles Quiz:

- ① Was ist der Unterschied zwischen `SELECT Spiele` und `SELECT DISTINCT Spiele`?
- ② Welches LIKE-Muster findet alle Teams mit "Borussia" im Namen?
 - A) `LIKE 'Borussia'`
 - B) `LIKE 'Borussia%'`
 - C) `LIKE '%Borussia%'`

Antworten

1. Ohne `DISTINCT`: alle Zeilen (mit Duplikaten). Mit `DISTINCT`: nur eindeutige Werte.
2. C – `%Borussia%` findet "Borussia" an beliebiger Stelle.

Hands-on

Eindeutige Werte & Mustersuche

marimo: 02-sql-exploration.py

Aufgaben 4.1 – 4.8

Inkl. Vorhersage-Aufgaben: Was wird das Ergebnis sein?

Pause

15 Minuten

Kaffee holen, Fragen notieren, kurz bewegen!

- 1 Rückblick & ORDER BY
- 2 DISTINCT & LIKE
- ▶ **3 NULL-Werte verstehen**
- 4 Visualisierung: Streudiagramme
- 5 Zusammenfassung

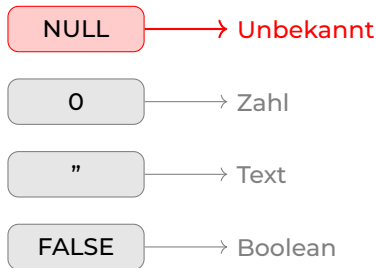
NULL bedeutet: Der Wert ist **unbekannt** oder **nicht vorhanden**.

NULL ist NICHT:

- Nicht die Zahl 0
- Nicht ein leerer String "
- Nicht das Wort 'NULL'
- Nicht FALSE

Beispiele für NULL:

- Geburtsdatum unbekannt
- Noch keine Tore geschossen (unbekannt, nicht 0!)
- Telefonnummer nicht angegeben
- Spieler hat keinen Spitznamen



Reales Beispiel: Spieler-Datenbank

Name	Verein	Tore	Vorlagen	Spitzname
Müller	Bayern	8	4	Mülli
Neuer	Bayern	0	NULL	NULL
Gündogan	NULL	NULL	2	Günni
Musiala	Bayern	12	7	NULL

- Neuer: 0 Tore (bekannt!), Vorlagen unbekannt
- Gündogan: Verein fehlt (vereinslos?), Tore nicht erfasst
- Musiala: Hat keinen bekannten Spitznamen

Problem

Wie rechnen wir mit NULL? Was ist $5 + \text{NULL}$? Was ist $\text{NULL} > 10$?

Wichtig

= NULL funktioniert **nicht!** Verwende IS NULL.

✗ Falsch:

```
SELECT *  
FROM spieler  
WHERE Tore = NULL;
```

→ Liefert **keine** Ergebnisse!

Warum?

NULL = NULL ist nicht TRUE, sondern UNKNOWN!

✓ Richtig:

```
SELECT *  
FROM spieler  
WHERE Tore IS NULL;
```

→ Alle Spieler ohne Tor-Angabe

```
SELECT *  
FROM spieler  
WHERE Tore IS NOT NULL;
```

→ Alle mit Tor-Angabe

Mit NULL gibt es **drei** mögliche Wahrheitswerte:

TRUE

FALSE

UNKNOWN

Beispiele

- $5 > 3 \rightarrow \text{TRUE}$
- $5 < 3 \rightarrow \text{FALSE}$
- $5 > \text{NULL} \rightarrow \text{UNKNOWN}$ (Wir wissen nicht, ob 5 größer ist!)
- $\text{NULL} = \text{NULL} \rightarrow \text{UNKNOWN}$ (Nicht TRUE!)

Konsequenz

WHERE gibt nur Zeilen zurück, bei denen die Bedingung **TRUE** ist – nicht

AND mit NULL

A	B	A AND B
T	?	?
F	?	F
?	?	?

FALSE "gewinnt" immer

OR mit NULL

A	B	A OR B
T	?	T
F	?	?
?	?	?

TRUE "gewinnt" immer

NOT mit NULL

A	NOT A
T	F
F	T
?	?

Bleibt UNKNOWN

Merkregel

? = UNKNOWN. Wenn das Ergebnis **nicht sicher** bestimmt werden kann, bleibt es UNKNOWN.

Ausnahme: FALSE AND anything = FALSE, TRUE OR anything = TRUE

Syntax

```
COALESCE(wert1, wert2, wert3, ...)
```

Gibt den **ersten Nicht-NULL-Wert** zurück.

```
SELECT
    Name,
    Tore AS Tore_Original,
    COALESCE(Tore, 0) AS Tore_Bereinigt
FROM spieler;
```

Name	Tore_Original	Tore_Bereinigt
Müller	8	8
Neuer	0	0
Gündogan	NULL	0

Aufgabe: Zeige alle Spieler mit Toren, ersetze NULL durch 0.

- 1 **Was zeigen?** → Name und Tore
- 2 **NULL ersetzen?** → Ja, mit COALESCE
- 3 **Ersatzwert?** → 0

```
SELECT  
    Name,  
    COALESCE(Tore, 0) AS Tore  
FROM spieler;
```

Wie COALESCE funktioniert

COALESCE(NULL, 0) → 0 (erster Wert ist NULL, also nimm zweiten)
COALESCE(8, 0) → 8 (erster Wert ist nicht NULL, nimm ihn)
COALESCE(NULL, NULL, 5) → 5 (erst der dritte ist nicht NULL)

```
SELECT  
    Name,  
    COALESCE(Spitzname, Vorname, Name) AS Anzeigename  
FROM spieler;
```

Logik:

- 1 Wenn Spitzname vorhanden → nimm Spitzname
- 2 Sonst wenn Vorname vorhanden → nimm Vorname
- 3 Sonst → nimm Name

Name	Spitzname	Vorname	Anzeigename
Müller	Müllli	Thomas	Müllli
Neuer	NULL	Manuel	Manuel
Musiala	NULL	Jamal	Jamal

Problem

Jede Rechnung mit NULL ergibt NULL!

```
-- Problem: Scorerpunkte berechnen
SELECT Name, Tore + Vorlagen AS Scorerpunkte
FROM spieler;
```

Name	Tore	Vorlagen	Scorerpunkte
Müller	8	4	12
Neuer	0	NULL	NULL
Gündogan	NULL	2	NULL

```
-- Lösung: COALESCE verwenden
SELECT Name,
       COALESCE(Tore, 0) + COALESCE(Vorlagen, 0) AS Scorerpunkte
FROM spieler;
```

Jetzt: Neuer = 0, Gündogan = 2

Situation	Lösung
NULL prüfen	IS NULL / IS NOT NULL
NULL ersetzen	COALESCE(wert, ersatz)
Rechnen mit NULL	Vorher mit COALESCE ersetzen
Vergleich mit NULL	IS NULL (nicht = NULL!)

Goldene Regeln

- 1 NULL ist nicht 0, nicht "", nicht FALSE – es ist **unbekannt**
- 2 Verwende IS NULL, niemals = NULL
- 3 Bei Berechnungen: erst NULL behandeln mit COALESCE
- 4 WHERE filtert UNKNOWN als FALSE

Hands-on

Umgang mit NULL

marimo: 02-sql-exploration.py

Aufgaben 6.1 – 6.12

40 Minuten – Debugging & Freie Exploration

- 1 Rückblick & ORDER BY
- 2 DISTINCT & LIKE
- 3 NULL-Werte verstehen
- ▶ **4 Visualisierung: Streudiagramme**
- 5 Zusammenfassung

Neu: `px.scatter()` – zeigt Beziehung zwischen zwei Variablen

Frage: Wer schießt viele Tore, kassiert aber auch viele?

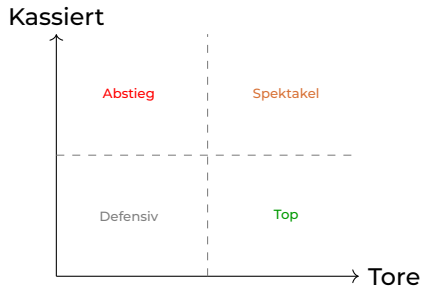
```
SELECT Mannschaft, ToreGeschossen, ToreKassiert  
FROM bundesliga;
```

```
px.scatter(ergebnis ,  
            x="ToreGeschossen",  
            y="ToreKassiert",  
            hover_name="Mannschaft",  
            title="Offensive vs. Defensive")
```

⇒ Jeder Punkt = ein Team. Hover zeigt den Namen.

Quadranten:

- **Oben rechts:** Viele Tore, viele kassiert
(offensiv, aber anfällig)
- **Unten rechts:** Viele Tore, wenig kassiert
(Spitzenteams)
- **Unten links:** Wenig Tore, wenig kassiert
(defensiv, langweilig)
- **Oben links:** Wenig Tore, viele kassiert
(Abstiegskandidaten)



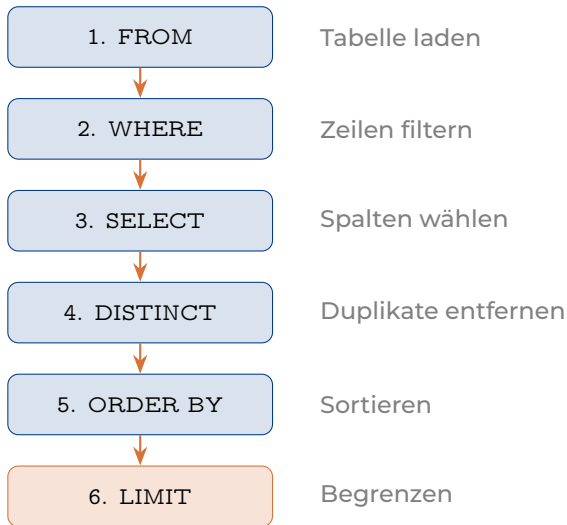
Neu: color= – färbt Punkte nach einer Kategorie

```
px.scatter(ergebnis ,  
            x="ToreGeschossen" ,  
            y="ToreKassiert" ,  
            color="Liga" ,    -- kategoriale Variable!  
            hover_name="Mannschaft")
```

Wann color= verwenden?

- Gruppen vergleichen (Liga, Position, Kategorie)
- Muster zwischen Gruppen erkennen
- **Nicht:** für kontinuierliche Werte wie Punkte

- 1 Rückblick & ORDER BY
- 2 DISTINCT & LIKE
- 3 NULL-Werte verstehen
- 4 Visualisierung: Streudiagramme
- **5 Zusammenfassung**



SQL	Syntax
Aufsteigend	ORDER BY x ASC
Absteigend	ORDER BY x DESC
Begrenzen	LIMIT n
Überspringen	OFFSET m
Eindeutig	DISTINCT
NULL prüfen	IS NULL
NULL ersetzen	COALESCE()

Visualisierung	Funktion
Balkendiagramm	px.bar()
Streudiagramm	px.scatter()
Histogramm	px.histogram()
Liniendiagramm	px.line()
Kreisdiagramm	px.pie()

Workflow

SQL-Abfrage → DataFrame → Plotly Chart

Was zeigt diese Abfrage?

```
SELECT Mannschaft, Punkte FROM bundesliga ORDER BY Punkte LIMIT 3;
```

- ① Die 3 Teams mit den meisten Punkten
- ② Die 3 Teams mit den wenigsten Punkten
- ③ Die ersten 3 Teams alphabetisch
- ④ Einen Fehler

Was zeigt diese Abfrage?

```
SELECT Mannschaft, Punkte FROM bundesliga ORDER BY Punkte LIMIT 3;
```

- ① Die 3 Teams mit den meisten Punkten
- ② **Die 3 Teams mit den wenigsten Punkten**
- ③ Die ersten 3 Teams alphabetisch
- ④ Einen Fehler

Erklärung

ORDER BY Punkte ohne ASC/DESC sortiert **aufsteigend** (ASC ist Default).
Also: kleinste Werte zuerst → Abstiegskandidaten!

Was ist das Ergebnis von:

```
SELECT * FROM spieler WHERE Tore > 5 OR Tore <= 5;
```

- ① Alle Spieler
- ② Nur Spieler mit Tore-Wert (nicht NULL)
- ③ Nur Spieler mit mehr als 5 Toren
- ④ Leeres Ergebnis

Was ist das Ergebnis von:

```
SELECT * FROM spieler WHERE Tore > 5 OR Tore <= 5;
```

- ① Alle Spieler
- ② **Nur Spieler mit Tore-Wert (nicht NULL)**
- ③ Nur Spieler mit mehr als 5 Toren
- ④ Leeres Ergebnis

Erklärung

Wenn Tore NULL ist:

NULL > 5 → UNKNOWN

NULL <= 5 → UNKNOWN

UNKNOWN OR UNKNOWN → UNKNOWN

WHERE filtert UNKNOWN als FALSE → Zeile wird nicht angezeigt!

Was gibt COALESCE(NULL, NULL, 'Hallo', 'Welt') zurück?

- ① NULL
- ② 'Hallo'
- ③ 'Welt'
- ④ 'Hallo Welt'

Was gibt COALESCE(NULL, NULL, 'Hallo', 'Welt') zurück?

- ① NULL
- ② 'Hallo'
- ③ 'Welt'
- ④ 'Hallo Welt'

Erklärung

COALESCE gibt den **ersten Nicht-NULL-Wert** zurück:

- Argument 1: NULL → weiter
- Argument 2: NULL → weiter
- Argument 3: 'Hallo' → **gefunden!**

'Welt' wird gar nicht mehr betrachtet.

Konzept	Syntax
Aufsteigend sortieren	ORDER BY spalte ASC
Absteigend sortieren	ORDER BY spalte DESC
Top N	ORDER BY spalte DESC LIMIT N
Bottom N	ORDER BY spalte ASC LIMIT N
Plätze X bis Y	LIMIT (Y-X+1) OFFSET (X-1)
Eindeutige Werte	SELECT DISTINCT spalte
Muster (beginnt mit)	LIKE 'Text%'
Muster (enthält)	LIKE '%Text%'
Muster (endet mit)	LIKE '%Text'
NULL finden	WHERE spalte IS NULL
Nicht NULL	WHERE spalte IS NOT NULL
NULL ersetzen	COALESCE(spalte, ersatz)

Vorlesung 3: Aggregation & Gruppierung

- Aggregatfunktionen: COUNT, SUM, AVG, MIN, MAX
- Gruppierung mit GROUP BY
- Filter auf Gruppen mit HAVING
- Unterschied WHERE vs. HAVING

Vorgeschmack

- Wie viele Teams haben mehr als 10 Siege? → COUNT
- Was ist der Punktedurchschnitt? → AVG
- Wie viele Tore wurden insgesamt geschossen? → SUM
- Welches Team hat die meisten Tore? → MAX

Fragen?

marimo: 02-sql-exploration.py

Weiter experimentieren!