

Sommersemester 2026

# Datenmanagement & -analyse

Prof. Dr. Christoph M. Flath

*Data Driven Decisions Group, Universität Würzburg*

- 1 Rückblick & Aggregatfunktionen
- 2 GROUP BY: Daten gruppieren
- 3 HAVING & Vertiefung
- 4 Visualisierung: Aggregierte Daten
- 5 Zusammenfassung

## Lernziele

Daten aggregieren, gruppieren und zusammenfassende Statistiken berechnen.

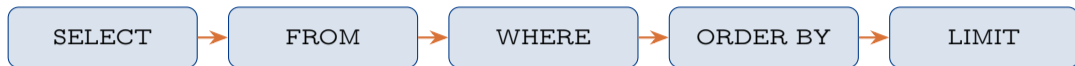
## ► 1 Rückblick & Aggregatfunktionen

2 GROUP BY: Daten gruppieren

3 HAVING & Vertiefung

4 Visualisierung: Aggregierte  
Daten

5 Zusammenfassung



- SELECT / DISTINCT – Spalten wählen
- WHERE – Zeilen filtern
- ORDER BY / LIMIT – Sortieren & Begrenzen
- IS NULL / COALESCE – Fehlende Werte

## Heute: Von Einzelwerten zu Zusammenfassungen

Wie viele Teams gibt es? Was ist die durchschnittliche Punktzahl? Welches Team hat die meisten Tore?

**COUNT**  
Anzahl

**SUM**  
Summe

**AVG**  
Durchschnitt

**MIN**  
Minimum

**MAX**  
Maximum

## Wichtig

Aggregatfunktionen fassen **viele Zeilen** zu **einem Wert** zusammen.

## Alle Zeilen zählen:

```
SELECT COUNT(*)  
FROM bundesliga;
```

→ 18 (alle Teams)

## Nicht-NULL Werte zählen:

```
SELECT COUNT(Spitzname)  
FROM spieler;
```

→ 6 (nur Spieler mit Spitzname)

## Eindeutige Werte zählen:

```
SELECT COUNT(DISTINCT Position)  
FROM spieler;
```

→ 4 (Tor, Abwehr, Mittelfeld, Sturm)

## Merke

COUNT(\*) zählt alle Zeilen  
COUNT(spalte) ignoriert NULL

## Summe:

```
SELECT SUM(Punkte)  
FROM bundesliga;
```

→ Gesamtpunkte aller Teams

```
SELECT SUM(ToreGeschossen)  
FROM bundesliga;
```

→ Alle Tore der Saison

## Durchschnitt:

```
SELECT AVG(Punkte)  
FROM bundesliga;
```

→ Durchschnittliche Punktzahl

```
SELECT AVG(Tordifferenz)  
FROM bundesliga;
```

→ ca. 0 (Nullsummenspiel!)

## Achtung bei NULL

AVG ignoriert NULL-Werte! Das kann den Durchschnitt verfälschen.

```
SELECT
  MIN(Punkte) AS Wenigste_Punkte,
  MAX(Punkte) AS Meiste_Punkte,
  MAX(Punkte) - MIN(Punkte) AS Spannweite
FROM bundesliga;
```

Wenigste_Punkte	Meiste_Punkte	Spannweite
12	50	38

## Auch für Text

MIN/MAX funktionieren auch alphabetisch:

MIN(Mannschaft) → '1. FC Heidenheim' (alphabetisch erstes)

```
SELECT
  COUNT(*) AS Anzahl_Teams,
  SUM(Siege) AS Gesamtsiege,
  AVG(Punkte) AS Schnitt_Punkte,
  MIN(Tordifferenz) AS Schlechteste_Diff,
  MAX(Tordifferenz) AS Beste_Diff
FROM bundesliga;
```

Teams	Siege	Schnitt	Min Diff	Max Diff
18	127	26.3	-27	+56

Eine Zeile, viele Informationen

Alle Aggregatfunktionen in einem SELECT liefern **eine Ergebniszeile**.

## COUNT und NULL:

```
SELECT
    COUNT(*) AS Alle ,
    COUNT(Tore) AS Mit_Toren
FROM spieler;
```

Alle	Mit_Toren
12	10

## AVG und NULL:

```
-- NULL wird ignoriert!
SELECT AVG(Tore)
FROM spieler;
```

Berechnet:  $(8 + 0 + 3 + 5 + \dots) / 10$

Nicht:  $(8 + 0 + 3 + 5 + \dots) / 12$

## Wichtig

AVG, SUM, MIN, MAX ignorieren NULL-Werte komplett!

Das kann den Durchschnitt verfälschen, wenn NULL eigentlich 0 bedeutet.

```
-- NULL als 0 behandeln
SELECT
    AVG(Tore) AS Schnitt_ohne_NULL,
    AVG(COALESCE(Tore, 0)) AS Schnitt_mit_NULL_als_0
FROM spieler;
```

Ohne NULL	Mit NULL als 0
5.7	4.75

## Wann COALESCE verwenden?

- Wenn NULL semantisch "0" bedeutet (z.B. keine Tore = 0 Tore)
- Wenn alle Zeilen in die Berechnung einfließen sollen

## Frage 1

```
SELECT COUNT(*) FROM bundesliga WHERE Punkte > 100
```

## Frage 1

```
SELECT COUNT(*) FROM bundesliga WHERE Punkte > 100
```

Antwort: 0 (kein Team hat mehr als 100 Punkte)

## Frage 2

```
SELECT AVG(Tordifferenz) FROM bundesliga
```

## Frage 1

```
SELECT COUNT(*) FROM bundesliga WHERE Punkte > 100
```

Antwort: 0 (kein Team hat mehr als 100 Punkte)

## Frage 2

```
SELECT AVG(Tordifferenz) FROM bundesliga
```

Antwort: ca. 0 (Nullsummenspiel – alle Tore sind auch Gegentore!)

## Frage 3

```
SELECT SUM(Siege) + SUM(Niederlagen) + SUM(Unentschieden) FROM bundesliga
```

## Frage 1

```
SELECT COUNT(*) FROM bundesliga WHERE Punkte > 100
```

Antwort: 0 (kein Team hat mehr als 100 Punkte)

## Frage 2

```
SELECT AVG(Tordifferenz) FROM bundesliga
```

Antwort: ca. 0 (Nullsummenspiel – alle Tore sind auch Gegentore!)

## Frage 3

```
SELECT SUM(Siege) + SUM(Niederlagen) + SUM(Unentschieden) FROM bundesliga
```

Antwort:  $18 \times 19 = 342$  (Gesamtzahl aller Spiele  $\times 2$ )

# Häufige Fehler bei Aggregationen

## Fehler 1: Mischen

```
-- FALSCH!  
SELECT Mannschaft , AVG(Punkte)  
FROM bundesliga ;
```

Mannschaft ist nicht aggregiert!

## Fehler 2: WHERE statt HAVING

```
-- FALSCH!  
SELECT Position , AVG(Tore)  
FROM spieler  
WHERE AVG(Tore) > 5  
GROUP BY Position ;
```

AVG existiert noch nicht bei WHERE!

## Richtig:

```
SELECT AVG(Punkte)  
FROM bundesliga ;
```

Nur Aggregat, keine Einzelspalten

## Richtig:

```
SELECT Position , AVG(Tore)  
FROM spieler  
GROUP BY Position  
HAVING AVG(Tore) > 5 ;
```

HAVING für Aggregat-Filter

```
-- Nur Teams mit positiver Tordifferenz  
SELECT  
    COUNT(*) AS Anzahl,  
    AVG(Punkte) AS Schnitt  
FROM bundesliga  
WHERE Tordifferenz > 0;
```

Anzahl	Schnitt
9	34.2

## Ablauf

- 1 WHERE filtert auf 9 Teams mit positiver Tordifferenz
- 2 COUNT und AVG werden nur für diese 9 berechnet

# Hands-on

## Erste Aggregationen

marimo: 03-sql-aggregation.py

Aufgaben 2.1 – 2.4

- 1 Rückblick & Aggregatfunktionen
- ▶ **2 GROUP BY: Daten gruppieren**
- 3 HAVING & Vertiefung
- 4 Visualisierung: Aggregierte Daten
- 5 Zusammenfassung

## Frage

Wie viele Tore hat jede **Position** im Durchschnitt?

**Bisher:** Eine Aggregation über **alle** Daten.

**Jetzt:** Eine Aggregation **pro Gruppe**.

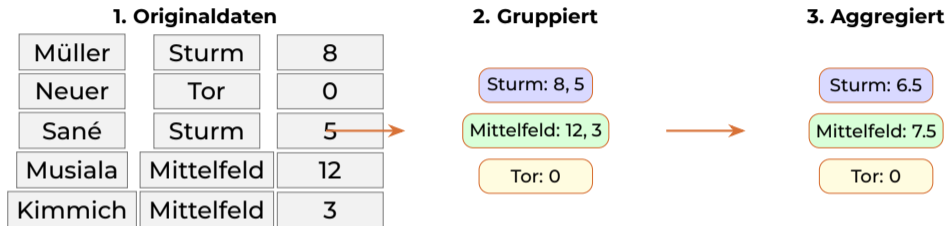


## Syntax

```
SELECT gruppe, AGGREGAT(spalte)
FROM tabelle
GROUP BY gruppe;
```

```
SELECT Position, AVG(Tore) AS Schnitt_Tore
FROM spieler
GROUP BY Position;
```

Position	Schnitt_Tore
Abwehr	1.0
Mittelfeld	8.3
Sturm	8.0
Tor	0.0



## Die goldene Regel

Im SELECT dürfen nur stehen:

1. Spalten aus GROUP BY
2. Aggregatfunktionen

## Falsch:

```
SELECT Position , Name, AVG(Tore)
FROM spieler
GROUP BY Position ;
```

Fehler! Welcher Name soll angezeigt werden?

## Richtig:

```
SELECT Position , AVG(Tore)
FROM spieler
GROUP BY Position ;
```

OK! Position ist in GROUP BY.

## Gedankenexperiment

Wenn wir nach Position gruppieren und es 3 Stürmer gibt – welcher Name sollte angezeigt werden? Müller? Sané? Füllkrug?

→ **Nicht eindeutig**, deshalb verboten!

```
SELECT Position , Verein , COUNT(*) AS Anzahl  
FROM spieler  
WHERE Verein IS NOT NULL  
GROUP BY Position , Verein;
```

Position	Verein	Anzahl
Abwehr	Bayer Leverkusen	1
Abwehr	Borussia Dortmund	1
Mittelfeld	Bayern München	2
Sturm	Bayern München	2
...	...	...

## Kombinations-Gruppen

Jede **Kombination** aus Position und Verein bildet eine eigene Gruppe.

## Frage 1: Wie viele Zeilen?

```
SELECT Position, COUNT(*) FROM spieler GROUP BY Position;
```

## Frage 1: Wie viele Zeilen?

```
SELECT Position, COUNT(*) FROM spieler GROUP BY Position;
```

Antwort: 4 Zeilen (Tor, Abwehr, Mittelfeld, Sturm)

## Frage 2: Wie viele Zeilen?

```
SELECT Spiele, COUNT(*) FROM bundesliga GROUP BY Spiele;
```

## Frage 1: Wie viele Zeilen?

```
SELECT Position, COUNT(*) FROM spieler GROUP BY Position;
```

Antwort: 4 Zeilen (Tor, Abwehr, Mittelfeld, Sturm)

## Frage 2: Wie viele Zeilen?

```
SELECT Spiele, COUNT(*) FROM bundesliga GROUP BY Spiele;
```

Antwort: 2 Zeilen (Teams mit 18 Spielen, Teams mit 19 Spielen)

## Merke

Anzahl der Ergebniszeilen = Anzahl der **unterschiedlichen Werte** in der GROUP BY Spalte

**Frage:** Wie unterscheidet sich die Effizienz der Teams nach Tabellenbereich?

```
SELECT
  CASE
    WHEN Punkte >= 45 THEN 'Spitze'
    WHEN Punkte >= 30 THEN 'Mittelfeld'
    WHEN Punkte >= 20 THEN 'Abstiegskampf'
    ELSE 'Abstiegszone'
  END AS Tabellenbereich,
  COUNT(*) AS Teams,
  ROUND(AVG(ToreGeschossen * 1.0 / Spiele), 2) AS Tore_pro_Spiel,
  ROUND(AVG(ToreKassiert * 1.0 / Spiele), 2) AS Gegentore_pro_Spiel
FROM bundesliga
GROUP BY Tabellenbereich
ORDER BY Tore_pro_Spiel DESC;
```

## Erkenntnis

Spitzenteams schießen mehr Tore UND kassieren weniger – doppelter Vorteil!

```
SELECT Position , AVG(Tore) AS Schnitt  
FROM spieler  
GROUP BY Position  
ORDER BY Schnitt DESC;
```

Position	Schnitt
Mittelfeld	8.3
Sturm	8.0
Abwehr	1.0
Tor	0.0

## Sortierung nach Aggregat

Mit ORDER BY können Sie die Gruppen nach dem berechneten Aggregat sortieren – z.B. um die "Top"-Gruppen zu finden.

**Frage:** Wie viele Teams haben eine positive/negative/neutrale Tordifferenz?

```
SELECT
  CASE
    WHEN Tordifferenz > 0 THEN 'Positiv'
    WHEN Tordifferenz < 0 THEN 'Negativ'
    ELSE 'Neutral'
  END AS Kategorie,
  COUNT(*) AS Anzahl,
  AVG(Punkte) AS Schnitt_Punkte
FROM bundesliga
GROUP BY Kategorie
ORDER BY Schnitt_Punkte DESC;
```

## Erkenntnisse

Teams mit positiver Tordifferenz haben im Schnitt deutlich mehr Punkte – Tore schießen korreliert mit Erfolg!

# Hands-on

## Gruppierung in der Praxis

marimo: 03-sql-aggregation.py

Aufgaben 4.1 – 4.4

# Pause

15 Minuten

- 1 Rückblick & Aggregatfunktionen
- 2 GROUP BY: Daten gruppieren
- ▶ **3 HAVING & Vertiefung**
- 4 Visualisierung: Aggregierte Daten
- 5 Zusammenfassung

## Frage

Welche Positionen haben **im Durchschnitt mehr als 5 Tore**?

## Erster Versuch (falsch):

WHERE AVG(Tore) > 5 ✗

## Problem

WHERE filtert **einzelne Zeilen** – aber AVG(Tore) existiert erst **nach** der Gruppierung!

→ Wir brauchen einen Filter, der **nach** GROUP BY wirkt.

## Syntax

```
SELECT gruppe, AGGREGAT(spalte)
FROM tabelle
GROUP BY gruppe
HAVING AGGREGAT(spalte) bedingung;
```

```
SELECT Position, AVG(Tore) AS Schnitt
FROM spieler
GROUP BY Position
HAVING AVG(Tore) > 5;
```

Position	Schnitt
Mittelfeld	8.3
Sturm	8.0

Abwehr (1.0) und Tor (0.0) wurden herausgefiltert.

## WHERE filtert **Zeilen**

(vor der Gruppierung)

```
SELECT Position , AVG(Tore)
FROM spieler
WHERE Tore > 0
GROUP BY Position ;
```

Nur Spieler mit Toren werden gruppiert.

## HAVING filtert **Gruppen**

(nach der Gruppierung)

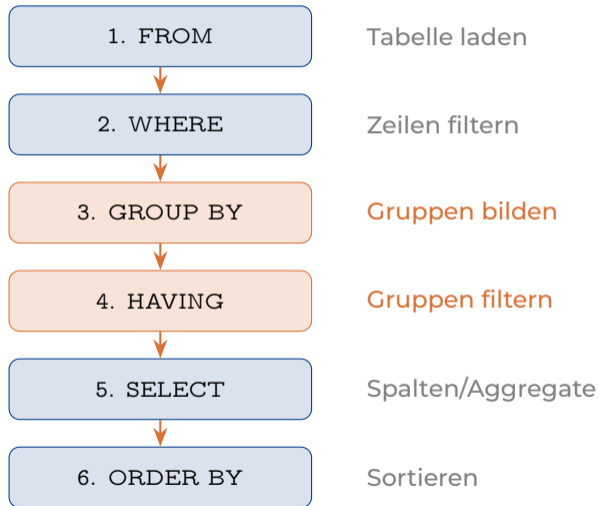
```
SELECT Position , AVG(Tore)
FROM spieler
GROUP BY Position
HAVING AVG(Tore) > 5;
```

Nur Gruppen mit hohem Schnitt werden angezeigt.

## Merkregel

**WHERE** = Filter auf **Rohdaten** (einzelne Zeilen)

**HAVING** = Filter auf **Aggregate** (Gruppen-Ergebnisse)



```
SELECT Position , COUNT(*) AS Anzahl , AVG(Tore) AS Schnitt
FROM spieler
WHERE Verein IS NOT NULL          -- Nur Spieler mit Verein
GROUP BY Position
HAVING COUNT(*) >= 2              -- Nur Positionen mit 2+ Spielern
ORDER BY Schnitt DESC;
```

## Reihenfolge der Ausführung:

- 1 FROM spieler – Alle 12 Spieler
- 2 WHERE Verein IS NOT NULL – 11 Spieler bleiben
- 3 GROUP BY Position – 4 Gruppen
- 4 HAVING COUNT(\*) >= 2 – 3 Gruppen bleiben
- 5 SELECT ... – Ergebnis berechnen
- 6 ORDER BY Schnitt DESC – Sortieren

## Top-N pro Gruppe:

```
-- Beste Tordifferenz  
-- pro Spielanzahl  
SELECT Spiele ,  
       MAX(Tordifferenz)  
FROM bundesliga  
GROUP BY Spiele;
```

## Verteilung analysieren:

```
-- Wie viele Teams  
-- pro Punktebereich?  
SELECT  
  CASE  
    WHEN Punkte >= 40 THEN 'Top'  
    WHEN Punkte >= 20 THEN 'Mitte'  
    ELSE 'Abstieg'  
  END AS Bereich ,  
  COUNT(*)  
FROM bundesliga  
GROUP BY Bereich;
```

```
SELECT Position ,  
       COUNT(*) AS Anzahl ,  
       AVG(Tore) AS Schnitt ,  
       SUM(Vorlagen) AS Gesamt_Vorlagen  
FROM spieler  
GROUP BY Position  
HAVING COUNT(*) >= 2  
       AND AVG(Tore) > 3;
```

Position	Anzahl	Schnitt	Vorlagen
Mittelfeld	4	8.3	26
Sturm	4	8.0	14

## Kombinierte Filter

HAVING kann mehrere Bedingungen mit AND/OR kombinieren.

## Welche Abfrage ist korrekt?

Aufgabe: Positionen mit mehr als 3 Spielern finden

A) ... WHERE COUNT(\*) > 3 GROUP BY Position

B) ... GROUP BY Position HAVING COUNT(\*) > 3

## Welche Abfrage ist korrekt?

Aufgabe: Positionen mit mehr als 3 Spielern finden

- A) ... WHERE COUNT(\*) > 3 GROUP BY Position
- B) ... GROUP BY Position HAVING COUNT(\*) > 3

Antwort: B – COUNT ist ein Aggregat, braucht HAVING

Aufgabe: Nur Spieler mit Verein gruppieren

- A) ... WHERE Verein IS NOT NULL GROUP BY Position
- B) ... GROUP BY Position HAVING Verein IS NOT NULL

## Welche Abfrage ist korrekt?

Aufgabe: Positionen mit mehr als 3 Spielern finden

- A) ... WHERE COUNT(\*) > 3 GROUP BY Position
- B) ... GROUP BY Position HAVING COUNT(\*) > 3

Antwort: B – COUNT ist ein Aggregat, braucht HAVING

Aufgabe: Nur Spieler mit Verein gruppieren

- A) ... WHERE Verein IS NOT NULL GROUP BY Position
- B) ... GROUP BY Position HAVING Verein IS NOT NULL

Antwort: A – Verein ist eine Zeilen-Eigenschaft, braucht WHERE

**Bestimme für jede Aufgabe: WHERE, HAVING oder beides?**

Aufgabe	Antwort
Nur Teams mit mehr als 40 Punkten	???

**Bestimme für jede Aufgabe: WHERE, HAVING oder beides?**

Aufgabe	Antwort
Nur Teams mit mehr als 40 Punkten	??? WHERE
Nur Positionen mit Torschnitt > 5	???

**Bestimme für jede Aufgabe: WHERE, HAVING oder beides?**

Aufgabe	Antwort
Nur Teams mit mehr als 40 Punkten	??? WHERE
Nur Positionen mit Torschnitt > 5	??? HAVING
Bayern-Spieler, gruppiert nach Position, nur Positionen mit mind. 2 Spielern	???

**Bestimme für jede Aufgabe: WHERE, HAVING oder beides?**

Aufgabe	Antwort
Nur Teams mit mehr als 40 Punkten	??? WHERE
Nur Positionen mit Torschnitt > 5	??? HAVING
Bayern-Spieler, gruppiert nach Position, nur Positionen mit mind. 2 Spielern	??? Beides!

## Merkregel

**WHERE** = Einzelne Zeilen ausschließen *bevor* gruppiert wird

**HAVING** = Ganze Gruppen ausschließen *nachdem* aggregiert wurde

## Abfrage 1:

```
SELECT Position , Name,  
        AVG(Tore)  
FROM spieler  
GROUP BY Position ;
```

## Abfrage 1:

```
SELECT Position , Name,  
        AVG(Tore)  
FROM spieler  
GROUP BY Position ;
```

Fehler: Name nicht in GROUP BY und  
nicht aggregiert

## Abfrage 2:

```
SELECT Position , AVG(Tore)  
FROM spieler  
WHERE AVG(Tore) > 5  
GROUP BY Position ;
```

## Abfrage 1:

```
SELECT Position , Name,  
        AVG(Tore)  
FROM spieler  
GROUP BY Position ;
```

Fehler: Name nicht in GROUP BY und  
nicht aggregiert

## Abfrage 2:

```
SELECT Position , AVG(Tore)  
FROM spieler  
WHERE AVG(Tore) > 5  
GROUP BY Position ;
```

Fehler: AVG in WHERE statt HAVING

## Abfrage 1:

```
SELECT Position , Name,  
       AVG(Tore)  
FROM spieler  
GROUP BY Position ;
```

Fehler: Name nicht in GROUP BY und nicht aggregiert

## Korrigiert:

```
SELECT Position , AVG(Tore)  
FROM spieler  
GROUP BY Position ;
```

## Abfrage 2:

```
SELECT Position , AVG(Tore)  
FROM spieler  
WHERE AVG(Tore) > 5  
GROUP BY Position ;
```

Fehler: AVG in WHERE statt HAVING

## Korrigiert:

```
SELECT Position , AVG(Tore)  
FROM spieler  
GROUP BY Position  
HAVING AVG(Tore) > 5;
```

```
SELECT Verein, COUNT(*) AS Spieleranzahl  
FROM spieler  
WHERE Verein IS NOT NULL  
GROUP BY Verein  
ORDER BY COUNT(*) DESC  
LIMIT 3;
```

Verein	Spieleranzahl
Bayern München	5
Bayer Leverkusen	2
...	...

## Top-N Analyse

Mit ORDER BY aggregat DESC LIMIT n finden Sie die "Top n" Gruppen.

```
SELECT
  'Bundesliga' AS Liga,
  COUNT(*) AS Teams,
  ROUND(AVG(Punkte), 1) AS Schnitt,
  MIN(Punkte) AS Min,
  MAX(Punkte) AS Max,
  MAX(Punkte) - MIN(Punkte) AS Spannweite
FROM bundesliga;
```

Liga	Teams	Schnitt	Min	Max	Spannweite
Bundesliga	18	26.3	12	50	38

## Deskriptive Statistik mit SQL

SQL eignet sich hervorragend für erste statistische Übersichten!

# Hands-on

## Komplexe Analysen

marimo: 03-sql-aggregation.py

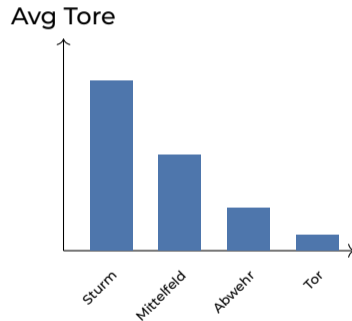
Aufgaben 6.1 – 6.5

- 1 Rückblick & Aggregatfunktionen
- 2 GROUP BY: Daten gruppieren
- 3 HAVING & Vertiefung
- **4 Visualisierung: Aggregierte Daten**
- 5 Zusammenfassung

## Aggregierte Daten → Balkendiagramm

```
SELECT  
    Position ,  
    AVG(Tore) AS Avg_Tore  
FROM spieler  
GROUP BY Position  
ORDER BY Avg_Tore DESC;
```

```
px.bar(ergebnis ,  
    x="Position" ,  
    y="Avg_Tore" ,  
    title="Tore nach Position")
```

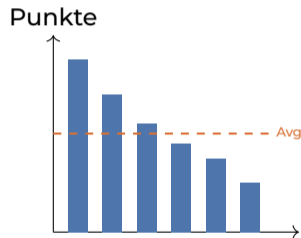


⇒ Jede Gruppe wird ein Balken

**Idee:** Aggregat berechnen, dann als Linie in den Plot einfügen

```
-- 1. Durchschnitt berechnen  
SELECT AVG(Punkte) FROM bundesliga;  
-- Ergebnis: 45.5
```

```
fig = px.bar(teams,  
             x="Mannschaft",  
             y="Punkte")  
  
# Durchschnittslinie  
fig.add_hline(y=45.5,  
              line_dash="dash",  
              annotation_text="Avg")
```



⇒ Wer liegt über/unter Durchschnitt?

- 1 Rückblick & Aggregatfunktionen
- 2 GROUP BY: Daten gruppieren
- 3 HAVING & Vertiefung
- 4 Visualisierung: Aggregierte Daten
- **5 Zusammenfassung**

Konzept	Syntax / Beispiel
Zählen	COUNT(*), COUNT(spalte), COUNT(DISTINCT spalte)
Summe	SUM(spalte)
Durchschnitt	AVG(spalte)
Extremwerte	MIN(spalte), MAX(spalte)
Gruppieren	GROUP BY spalte1, spalte2
Gruppen filtern	HAVING aggregat > wert

## Vollständige Abfrage

```
SELECT gruppe, COUNT(*), AVG(wert)
FROM tabelle
WHERE bedingung_zeilen
GROUP BY gruppe
HAVING COUNT(*) > 5
ORDER BY AVG(wert) DESC;
```

## Funktionen

COUNT(\*) – Zeilen  
COUNT(col) – Nicht-NULL  
SUM(col) – Summe  
AVG(col) – Mittelwert  
MIN/MAX(col) – Extrem

## Gruppierung

GROUP BY col  
Goldene Regel:  
SELECT darf nur:  
- GROUP BY Spalten  
- Aggregatfunktionen

## Filter

WHERE – Zeilen  
(vor Gruppierung)  
HAVING – Gruppen  
(nach Gruppierung)

Funktion	Syntax	Beispiel & Bedeutung
COUNT	COUNT(*) COUNT(spalte) COUNT(DISTINCT s)	Anzahl aller Zeilen Anzahl nicht-NULL Werte Anzahl eindeutiger Werte
SUM	SUM(spalte)	Summe aller Werte (ignoriert NULL)
AVG	AVG(spalte)	Arithmetisches Mittel (ignoriert NULL)
MIN	MIN(spalte)	Kleinster Wert (auch Text: alphabetisch)
MAX	MAX(spalte)	Größter Wert (auch Text: alphabetisch)

## Vollständiges Abfrage-Template

## Die goldene Regel vergessen

- Nur GROUP BY Spalten oder Aggregate im SELECT
- Sonst: Welcher Wert soll angezeigt werden?

## WHERE statt HAVING

- WHERE: vor Gruppierung
- HAVING: nach Gruppierung
- Aggregate nur in HAVING!

## NULL vergessen

- COUNT(\*) vs COUNT(spalte)
- AVG ignoriert NULL
- COALESCE wenn nötig

## Reihenfolge verwechseln

- SELECT – FROM – WHERE
- GROUP BY – HAVING
- ORDER BY – LIMIT

## 1 Spalte nicht in GROUP BY:

`SELECT Mannschaft, AVG(Punkte) FROM bundesliga;`

→ Entweder alle Spalten in GROUP BY oder aggregieren!

## 2 Aggregat in WHERE:

`WHERE AVG(Punkte) > 30`

→ HAVING verwenden!

## 3 COUNT(\*) vs COUNT(spalte) verwechseln:

Bei NULL-Werten verschiedene Ergebnisse!

## 4 NULL ignorieren bei AVG:

AVG(spalte) ignoriert NULL – ist das gewollt?

## 5 Alias in HAVING verwenden:

`HAVING Schnitt > 5` (in manchen DBs nicht erlaubt)

→ `HAVING AVG(Tore) > 5`

- 1 Was ist der Unterschied zwischen `COUNT(*)` und `COUNT(spalte)`?

- 1 Was ist der Unterschied zwischen COUNT(\*) und COUNT(spalte)?  
COUNT(\*) zählt alle Zeilen, COUNT(spalte) ignoriert NULL
- 2 Warum kann man AVG(Punkte) > 30 nicht in WHERE verwenden?

- 1 Was ist der Unterschied zwischen COUNT(\*) und COUNT(spalte)?  
COUNT(\*) zählt alle Zeilen, COUNT(spalte) ignoriert NULL
- 2 Warum kann man AVG(Punkte) > 30 nicht in WHERE verwenden?  
AVG existiert erst nach GROUP BY – WHERE filtert vorher
- 3 Was passiert, wenn Sie GROUP BY vergessen aber ein Aggregat verwenden?

- 1 Was ist der Unterschied zwischen COUNT(\*) und COUNT(spalte)?  
COUNT(\*) zählt alle Zeilen, COUNT(spalte) ignoriert NULL
- 2 Warum kann man AVG(Punkte) > 30 nicht in WHERE verwenden?  
AVG existiert erst nach GROUP BY – WHERE filtert vorher
- 3 Was passiert, wenn Sie GROUP BY vergessen aber ein Aggregat verwenden?  
Das Aggregat wird über ALLE Zeilen berechnet (eine Ergebniszeile)

## Business Intelligence:

- Umsatz pro Region, Monat, Produkt
- Durchschnittlicher Bestellwert pro Kunde
- Anzahl Bestellungen pro Tag

## Datenqualität:

- Wie viele NULL-Werte gibt es pro Spalte?
- Gibt es Duplikate? (COUNT vs COUNT DISTINCT)
- Wertebereich prüfen (MIN/MAX)

## Reporting:

- Zusammenfassungen für Dashboards
- KPIs berechnen
- Trends identifizieren (mit Zeitgruppen)

## Vorlesung 4: CRISP-DM & Fallstudien

- Der CRISP-DM-Prozess
- Fallstudie I: Dr. Harold Shipman – Anomalieerkennung
- Fallstudie II: Benford's Law – Betrugserkennung
- SQL für strukturierte Datenanalyse

### Vorgeschmack

Wie konnte Datenanalyse helfen, einen Serienmord aufzuklären?  
Warum beginnen 30% aller Zahlen mit der Ziffer 1?

# Fragen?

marimo: 03-sql-aggregation.py

Weiter experimentieren!