

Beispieltabelle

In diesem Lehrgang werden wir die Tabellen **films** und **Soundtracks** verwenden.

Filme					
movie_name	director	Umsatz	Datum	Genre	Sprache
Avengers: Zeitalter des Ultron	J. Whedon	1400000000	2015	Aktion	Englisch
Amelie	Jean-Pierre Jeunet	1740000000	2002	Drama	Französisch

Soundtracks		
Soundtrack	Komponist	Datum
Der Anfang: Musik aus dem Kinofilm	Hans Zimmer	2010
Le Fabuleux Destin D'Amelie Poulain	Yann Tiersen	2001

1. SELECT

Die Anweisung `SELECT` wird verwendet, um anzugeben, welche Spalten einer Datenbanktabelle in das Ergebnis aufgenommen werden sollen.

In diesem Beispiel werden nur die Spalten `movie_name` und `director` ausgewählt; andere Spalten werden nicht zurückgegeben.



```
SELECT movie_name, director  
FROM films;
```

[🔮 Code erklären](#)

POWERED BY datalab

Um alle Spalten einer Tabelle auszuwählen, kannst du `SELECT *` (sprich: "Stern") verwenden. Dieses Beispiel wählt alle Spalten aus der `films` Tabelle.

```
SELECT *  
FROM films;
```

[🔮 Code erklären](#)

POWERED BY datalab

2. LIMIT

Die Begrenzung der Anzahl von Zeilen, die von einer Tabelle zurückgegeben werden, ist ein nützlicher Trick, um die Abfragezeit zu verkürzen. Um die Anzahl der Zeilen zu begrenzen, kannst du den Befehl `LIMIT` verwenden. Dieses Beispiel wählt alle Spalten der `films` Tabelle aus und beschränkt die Ergebnisse dann auf die ersten zehn Zeilen.

```
SELECT *  
FROM films  
LIMIT 10;
```

[🔮 Code erklären](#)

POWERED BY datalab

3. AS

Um eine Spalte oder eine Tabelle bei der Rückgabe von Ergebnissen umzubenennen, kannst du mit dem Befehl `AS` einen Alias für deine Ausgaben festlegen. In diesem Beispiel wird die Spalte `movie_name` ausgewählt und ihr der Alias `movie_title` gegeben.

```
SELECT movie_name AS movie_title  
FROM films;
```

[🔮 Code erklären](#)

POWERED BY datalab

 **SELECT DISTINCT**

Datensätze enthalten oft doppelte Zeilen oder Werte in einer Spalte. Wenn du `SELECT` mit `DISTINCT` kombinierst, fallen Duplikate weg. Dieses Beispiel gibt die eindeutigen Werte in der Spalte `director` zurück.

```
SELECT DISTINCT director
FROM films;
```

[🔗 Code erklären](#)POWERED BY  datalab

5. COUNT

`COUNT()` gibt die Anzahl der Zeilen in der Tabelle oder Gruppe zurück. Dieses Beispiel gibt die Anzahl der Zeilen in der Tabelle `films` zurück und benennt das Ergebnis dann in `number_of_films` um.

```
SELECT COUNT(*) AS number_of_films
FROM films;
```

[🔗 Code erklären](#)POWERED BY  datalab

SQL kostenlos lernen

Einführung in SQL

Beginner ⌚ 2 Std. 👤 975.7K learners

Lerne in nur zwei Stunden, wie man relationale Datenbanken mit SQL erstellt und abfragt.

See Details →

Datenbearbeitung in SQL

Beginner ⌚ 4 Std. 👤 250.8K learners

Beherrsche die komplexen SQL-Abfragen, die notwendig sind, um eine Vielzahl von datenwissenschaftlichen Fragen zu beantworten und robuste Datensätze für die Analyse in PostareSQL vorzubereiten.

See Details →

See More →

6. MIN

`MIN()` gibt den Mindestwert in einer numerischen Spalte zurück. Bei Textspalten gibt `MIN()` den ersten Wert in alphabetischer Reihenfolge zurück. In diesem Beispiel wird der Film mit den geringsten Einnahmen zurückgegeben.

```
SELECT MIN(revenue) AS minimum_revenue,  
FROM films;
```



Code erklären

POWERED BY datalab

7. MAX

`MAX()` gibt den Maximalwert in einer numerischen Spalte zurück. Bei Textspalten gibt `MAX()` den letzten Wert in alphabetischer Reihenfolge zurück. Dieses Beispiel liefert den

Film mit den meisten Einnahmen.

```
SELECT MAX(revenue) AS maximum_revenue,  
FROM films;
```



Code erklären

POWERED BY datalab

8. SUM

SUM() gibt die Summe der numerischen Werte zurück. Dieses Beispiel liefert die Gesamteinnahmen aller Filme, die in der Tabelle **films** aufgeführt sind.

```
SELECT SUM(revenue) AS total_revenue  
FROM films;
```



Code erklären

POWERED BY datalab

9. AVERAGE

AVERAGE berechnet das arithmetische Mittel einer Spalte. In diesem Beispiel werden die durchschnittlichen Einnahmen aller in der Tabelle **films** aufgeführten Filme ermittelt.

```
SELECT AVERAGE(revenue) AS average_earned  
FROM films;
```



Code erklären

POWERED BY datalab

10. WHERE

Die WHERE Klausel filtert Zeilen, die einer bestimmten Bedingung entsprechen. Im Folgenden filtern wir zum Beispiel Filme, die mehr als 500 Millionen Dollar eingespielt haben

```
SELECT revenue  
FROM films  
WHERE revenue > 500000000;
```



Code erklären

POWERED BY datalab

Andere bedingte Operatoren wie <, >, >=, <=, == (equals), != (not equals) können
 filtern verwendet werden.

11. HAVING

Die HAVING-Klausel ähnelt der WHERE-Klausel, aber sie kann nur mit Aggregatfunktionen verwendet werden, während WHERE das nicht kann. In der folgenden Abfrage wählen wir zum Beispiel alle Filmgenres aus, die mindestens 50 Filme in ihrer Kategorie haben:

```
SELECT movie_name, director, date, COUNT(genre)
FROM films
GROUP BY genre
HAVING COUNT(genre) >= 50;
```

[Code erklären](#)

POWERED BY datalab

Hier ist ein weiteres Beispiel, das Filme nach Altersfreigabe gruppiert und nur die Freigaben mit einem durchschnittlichen Umsatz von über 100 Millionen auswählt:

```
SELECT movie_name, director, date, revenue
FROM films
GROUP BY age_rating
HAVING AVERAGE(genre) >= 100;
```

[Code erklären](#)

POWERED BY datalab

12. AND

AND Operator wird verwendet, um Zeilen zu filtern, die mehr als eine Bedingung erfüllen. Im folgenden Beispiel werden wir nach englischen Filmen filtern, die mehr als 500 Millionen Dollar eingespielt haben.

```
SELECT *
FROM films
WHERE revenue > 500000000 AND
      language == "English";
```

[Code erklären](#)

POWERED BY datalab

13. OR

OR ist ein weiterer bedingter Operator, mit dem du Zeilen unterteilen kannst, wenn eine der durch OR getrennten Bedingungen wahr ist. Dieses Beispiel liefert englische Filme, die weniger als 100 Millionen Dollar einnahmen, oder französische Filme, die mehr als



500 Millionen Dollar einnehmen.

```
SELECT *  
FROM films  
WHERE (revenue < 1000000000 AND language == "English") OR  
      (revenue > 5000000000 AND language == "French");
```



POWERED BY  datalab

14. BETWEEN

BETWEEN ermöglicht es dir, Zeilen innerhalb eines bestimmten Bereichs zu unterteilen, was die WHERE Klauseln einfacher und übersichtlicher macht. Im obigen Beispiel wählen wir alle Filme aus, die zwischen 2020 und 2022 veröffentlicht werden.

```
SELECT *  
FROM films  
WHERE date BETWEEN 2020 AND 2022;
```



 Code erklären


POWERED BY  datalab

15. IN

Der IN-Operator ist eine Abkürzung für mehrere OR Anweisungen. Dieses Beispiel liefert alle Filme, die in einem der folgenden Jahre veröffentlicht wurden: 1998, 1966, 2001 und 2012.

```
SELECT movie_name, date, revenue  
FROM films  
WHERE date IN (1998, 1966, 2001, 2012);
```



 Code erklären

POWERED BY  datalab

16. LIKE

Mit dem Operator LIKE kannst du nach Mustern in einer Textspalte suchen, indem du spezielle Zeichenfolgen, sogenannte Platzhalter, verwendest. Mit Platzhaltern kannst du Textzeichenfolgen finden, die einem bestimmten Muster entsprechen. Wenn du z.B. "A% W%" als Platzhalter für die Direktoren-Spalte verwendest, findest du alle Direktoren, deren Vorname mit A und deren Nachname mit W beginnt.

```
SELECT *  
FROM films  
WHERE director LIKE "A% W%";
```

[🔗 Code erklären](#)

POWERED BY datalab

17. GROUP BY

GROUP BY ermöglicht es dir, Zeilen auf der Grundlage von Spaltenwerten zu gruppieren. GROUP BY wird normalerweise mit Aggregatfunktionen wie COUNT, MIN, MAX, AVERAGE, und anderen verwendet. Im folgenden Beispiel finden wir die durchschnittlichen Einnahmen für jedes Filmgenre.

```
SELECT AVERAGE(revenue), movie_name, director, date  
FROM films  
GROUP BY genre;
```

[🔗 Code erklären](#)

POWERED BY datalab

Wir hätten auch MIN, MAX oder COUNT(revenue) verwenden können, um die niedrigsten und höchsten Einnahmen oder die Anzahl der Filme in jedem Genre zu ermitteln. Beachte, dass die Anweisung GROUP BY praktisch keine Auswirkungen auf die Abfrage hat, wenn sie nicht mit einer Aggregatfunktion verwendet wird.

18. ORDER BY

ORDER BY kannst du Zeilen anhand eines Spaltenwerts ordnen. Du kannst nach aufsteigender (Standard) oder absteigender Reihenfolge sortieren, indem du die ASC oder DESC hinzufügst. In diesem Beispiel werden die Einnahmen in aufsteigender Reihenfolge geordnet.

```
SELECT *  
FROM films  
ORDER BY revenue ASC;
```

[🔗 Code erklären](#)

POWERED BY datalab

19. UPDATE

UPDATE wird verwendet, um die Werte einzelner Zellen in einer bestehenden Tabelle zu ändern. Es wird zusammen mit dem Schlüsselwort SET verwendet. In diesem Beispiel



wird der Direktor "J. Whedon" auf "Joss Whedon" aktualisiert.

```
UPDATE films
SET director = "Joss Whedon"
WHERE director = "J. Whedon"
```



Code erklären

POWERED BY datalab

Die WHERE Klausel ist beim Schreiben von UPDATE Anweisungen entscheidend. Ohne sie hätte die obige Abfrage alle Filme von Joss Whedon ergeben.

20. ALTER TABLE

Mit der Anweisung ALTER TABLE kannst du die Eigenschaften der Tabelle und ihrer Spalten ändern (nicht die tatsächlichen Zellwerte). Zum Beispiel das Ändern von Spaltennamen, das Hinzufügen neuer Spalten, das Löschen von Spalten oder das Ändern ihres Datentyps. Die folgenden Beispiele zeigen, wie die Spalte date weggelassen und die Spalte age_rating hinzugefügt wird.

```
ALTER TABLE films
DROP COLUMN date;
```



Code erklären

POWERED BY datalab

```
ALTER TABLE films
ADD COLUMN age_rating;
```



Code erklären

POWERED BY datalab

21. CREATE TABLE

CREATE TABLE erstellt eine neue Tabelle in einer Datenbank. Im Folgenden erstellen wir eine Tabelle "Bibliotheken" mit vier Spalten: eine ganzzahlige Spalte namens id , eine Zeichenspalte namens name , eine Zeichenspalte namens version und eine ganzzahlige Spalte namens num_downloads .

```
CREATE TABLE libraries (  
  lib_id int,  
  name varchar(100),  
  version varchar(100),  
  num_downloads int  
)
```

[Code erklären](#)

POWERED BY datalab

22. INSERT INTO

INSERT INTO Anweisung kann verwendet werden, um neue Zeilen zu einer Tabelle hinzuzufügen. In diesem Beispiel fügen wir den Film “Doctor Strange” zur **films** Tabelle hinzu.

```
INSERT INTO films (movie_name, director, revenue, date, genre, language)  
VALUES ("Doctor Strange 2", "Sam Raimi", 409000000, 2022, "Action", "English")
```



POWERED BY datalab

23. INNER JOIN

Der Befehl INNER JOIN wählt die Zeilen aus, deren Werte in beiden Tabellen übereinstimmen. In der folgenden Abfrage sind wir zum Beispiel die Tabellen Filme und Soundtracks auf einer gemeinsamen date Spalte, die Filme und Soundtracks zurückgibt, die in denselben Jahren veröffentlicht wurden.

```
SELECT *  
FROM films  
INNER JOIN soundtracks  
ON films.date = soundtracks.date  
LIMIT 10;
```

[Code erklären](#)

POWERED BY datalab

24. LEFT JOIN

Eine LEFT JOIN behält alle ursprünglichen Datensätze in der linken Tabelle und gibt fehlende Werte für alle Spalten der rechten Tabelle zurück, in denen das Verknüpfungsfeld keine Übereinstimmung gefunden hat.



```
SELECT *  
FROM films  
LEFT JOIN soundtracks  
ON films.date = soundtracks.date  
LIMIT 10;
```

[🔗 Code erklären](#)

POWERED BY datalab

25. RIGHT JOIN

Eine RIGHT JOIN behält alle ursprünglichen Datensätze in der rechten Tabelle und gibt fehlende Werte für alle Spalten der linken Tabelle zurück, in denen das Verknüpfungsfeld keine Übereinstimmung gefunden hat.

```
SELECT *  
FROM films  
LEFT JOIN soundtracks  
ON films.date = soundtracks.date  
LIMIT 10;
```

[🔗 Code erklären](#)

POWERED BY datalab

SQL-Befehle Spickzettel

Um mehr über SQL zu erfahren, solltest du dir die folgenden Ressourcen ansehen:

- [SQL-Grundlagen Spickzettel](#)
- [SQL Joins Spickzettel](#)
- [Wie man SQL lernt](#)

Werde SQL-zertifiziert

Beweise mit einer Zertifizierung, dass deine SQL-Kenntnisse für den Job geeignet sind.

Meine Karriere ankurbeln

