Der SQL-Röntgenblick

DataBlick-Dokumentation · Stand: 29.08.2026

---

1. Was ist der Röntgenblick?

Eine der größten Hürden beim SQL-Lernen ist unsichtbar: SQLite wertet eine SELECT-Abfrage **nicht in der Reihenfolge aus, in der man sie schreibt**. Geschrieben wird `SELECT … FROM … WHERE … GROUP BY … HAVING … ORDER BY … LIMIT` – ausgewertet wird aber logisch in dieser Reihenfolge:

`FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY / LIMIT`

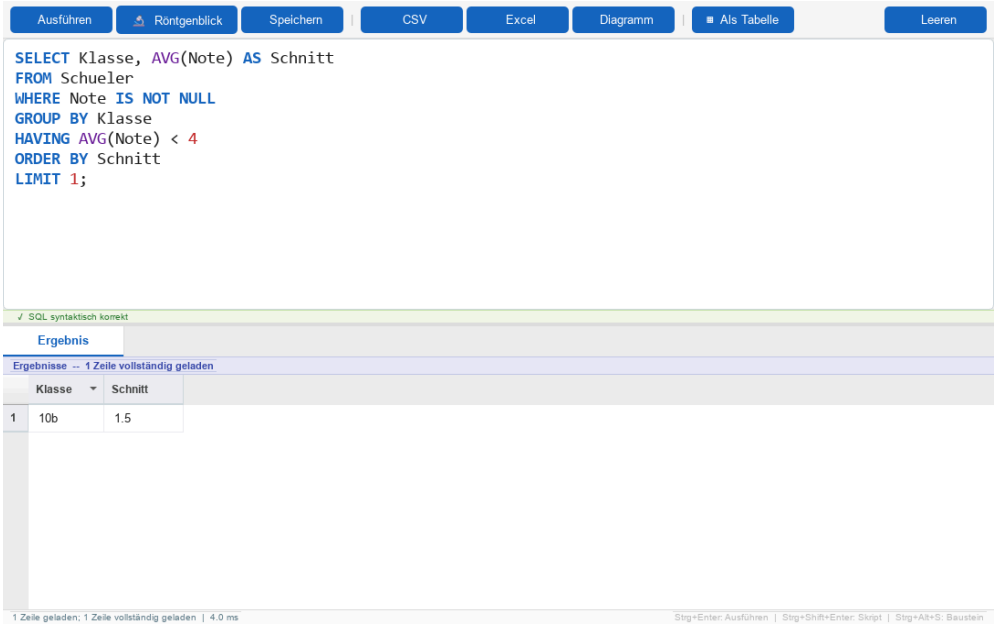
Der **Röntgenblick** macht diese verborgene Reihenfolge sichtbar. Er führt die Abfrage Klausel für Klausel aus und zeigt für jeden Schritt:

- die **Zeilen- bzw. Gruppenzahl** (mit Vergleich zum vorherigen Schritt),
- eine **Mini-Vorschau** der ersten fünf Zeilen des Zwischenergebnisses,
- das **Teil-SQL**, das für den Schritt ausgeführt wurde,
- eine kurze **Erklärung**, was in diesem Schritt passiert.

So wird aus der abstrakten Regel „HAVING filtert Gruppen, WHERE filtert Zeilen" eine beobachtbare Tatsache: Man sieht die Zahlen schrumpfen.

2. Wo finde ich ihn?

Der Röntgenblick sitzt als Button 🔬 **Röntgenblick** in der Toolbar des SQL-Editors, direkt neben „Ausführen" und „Skript ausführen":



SQL-Editor mit Röntgenblick-Button

Voraussetzungen:

1. Eine Datenbank ist geöffnet.
2. Im Editor steht **genau eine SELECT-Abfrage** (markierter Text zählt, sonst der gesamte Editorinhalt).
3. Ein Klick auf 🔬 startet die Analyse. Enthält die Abfrage benannte Parameter wie `:Name`, fragt DataBlick die Werte vorher ab – wie beim normalen Ausführen.

3. Ein kleines Beispiel

Wir verwenden eine Mini-Tabelle `Schueler` mit acht Zeilen. Sie lässt sich mit folgendem Skript anlegen (SQL-Editor → „Skript ausführen"):

```
CREATE TABLE Schueler (
    id     INTEGER PRIMARY KEY,
    Name   TEXT,
    Klasse TEXT,
    Note   REAL
);
INSERT INTO Schueler (Name, Klasse, Note) VALUES
    ('Anna',  '10a', 2.0),
    ('Ben',   '10a', 4.0),
    ('Carla', '10a', 3.0),
    ('David', '10b', 1.0),
    ('Emil',  '10b', 2.0),
    ('Feli',  '10c', NULL),
    ('Gero',  '10c', 5.0),
    ('Hana',  '10c', 5.0);
```

Die Frage: Welche Klasse hat den besten Notenschnitt – aber nur unter den Klassen, deren Schnitt besser als 4 ist, und ohne fehlende Noten?

Die Abfrage:

```
SELECT Klasse, AVG(Note) AS Schnitt
FROM Schueler
WHERE Note IS NOT NULL
GROUP BY Klasse
HAVING AVG(Note) < 4
ORDER BY Schnitt
LIMIT 1;
```

Ein Klick auf 🔬 **Röntgenblick** öffnet die Schrittfolge:

SQLite wertet eine SELECT-Abfrage logisch in dieser Reihenfolge aus – nicht in der Schreibreihenfolge. Jeder Schritt zeigt, wie sich die Zeilenmenge verändert.

Schritt 1 · FROM

8 Zeile(n)

SQLite bildet zuerst die Ausgangsmenge: alle Zeilen der beteiligten Tabellen.

```
SELECT * FROM Schueler
```

id	Name	Klasse	Note
1	Anna	10a	2.0
2	Ben	10a	4.0
3	Carla	10a	3.0
4	David	10b	1.0
5	Emil	10b	2.0

Schritt 2 · WHERE

7 Zeile(n) (vorher: 8)

Der Filter prüft jede Zeile der Ausgangsmenge und behält nur die Zeilen, für die die Bedingung wahr ist.

```
SELECT * FROM Schueler WHERE Note IS NOT NULL
```

id	Name	Klasse	Note
1	Anna	10a	2.0
2	Ben	10a	4.0
3	Carla	10a	3.0
4	David	10b	1.0
5	Emil	10b	2.0

Schritt 3 · GROUP BY

3 Gruppe(n) (vorher: 7)

Zeilen mit gleichem Wert werden zu Gruppen zusammengefasst. Aggregatfunktionen wie COUNT, SUM oder AVG rechnen ab jetzt pro Gruppe, nicht mehr pro Zeile.

```
SELECT Klasse, COUNT(*) FROM Schueler WHERE Note IS NOT NULL GROUP BY Klasse
```

Klasse	COUNT(*)
10a	3
10b	2
10c	2

Schritt 4 · HAVING

2 Gruppe(n) (vorher: 3)

HAVING filtert die fertigen Gruppen – anders als WHERE, das einzelne Zeilen vor der Gruppierung filtert.

```
SELECT Klasse, AVG(Note) AS Schnitt FROM Schueler WHERE Note IS NOT NULL GROUP BY Klasse HAVING AVG(Note) < 4
```

Klasse	Schnitt
10a	3.0
10b	1.5

Schritt 5 · SELECT

2 Zeile(n)

Erst jetzt werden die Ausgabespalten bestimmt – Spaltenwahl, Berechnungen und Aliasnamen.

```
SELECT Klasse, AVG(Note) AS Schnitt FROM Schueler WHERE Note IS NOT NULL GROUP BY Klasse HAVING AVG(Note) < 4
```

Klasse	Schnitt
10a	3.0
10b	1.5

Schritt 6 · ORDER BY / LIMIT

1 Zeile(n) (vorher: 2)

ORDER BY sortiert nur die Reihenfolge – die Zeilenanzahl ändert sich dadurch nicht. LIMIT schneidet das sortierte Ergebnis auf die angegebene Zeilenanzahl zu.

```
SELECT Klasse, AVG(Note) AS Schnitt FROM Schueler WHERE Note IS NOT NULL GROUP BY Klasse HAVING AVG(Note) < 4 ORDER BY Schnitt LIMIT 1
```

Klasse	Schnitt
10b	1.5

Der komplette Röntgenblick-Dialog

Der Durchlauf in Zahlen

Schritt	Klausel	Ergebnis	Was passiert
1	`FROM Schueler`	**8 Zeilen**	Ausgangsmenge: alle Datensätze der Tabelle
2	`WHERE Note IS NOT NULL`	**7 Zeilen**	Feli (Note = NULL) fliegt heraus
3	`GROUP BY Klasse`	**3 Gruppen**	10a, 10b, 10c – aus 7 Zeilen werden 3 Gruppen
4	`HAVING AVG(Note) < 4`	**2 Gruppen**	10c (Schnitt 5,0) fliegt heraus
5	`SELECT Klasse, AVG(Note)`	**2 Zeilen**	Erst jetzt entstehen die Ausgabespalten
6	`ORDER BY Schnitt LIMIT 1`	**1 Zeile**	Sortieren ändert nichts an der Anzahl – LIMIT schneidet zu

Das Endergebnis: **10b mit Schnitt 1,5**.

GROUP BY und HAVING im Detail

Die Schritte 3 und 4 sind der Kern des Beispiels – hier passiert der Übergang von *Zeilen* zu *Gruppen*:

Schritte 3 und 4: GROUP BY und HAVING

- In **Schritt 3** zeigt die Vorschau pro Klasse die Zeilenzahl (`COUNT(*)`): 10a hat 3 Zeilen, 10b und 10c je 2. Die Zählerzeile „3 Gruppe(n) (vorher: 7)" macht den Übergang sichtbar: Sieben Zeilen wurden zu drei Gruppen zusammengefasst.
- In **Schritt 4** filtert `HAVING AVG(Note) < 4` die *fertigen Gruppen*. 10c hat den Schnitt 5,0 und fällt heraus – übrig bleiben 10a (3,0) und 10b (1,5).

Der didaktische Kernsatz: `WHERE` hat in Schritt 2 einzelne *Zeilen* gefiltert (Feli, Note = NULL), *bevor* gruppiert wurde. `HAVING` filtert in Schritt 4 ganze *Gruppen*, *nachdem* gruppiert wurde. Deshalb darf in `HAVING` eine Aggregatfunktion stehen, in `WHERE` nicht.

4. Was man am Röntgenblick lernen kann

Warum funktioniert `WHERE Schnitt < 4` nicht? Der Alias `Schnitt` entsteht erst in Schritt 5 (SELECT). In Schritt 2 (WHERE) existiert er noch gar nicht – SQLite meldet „no such column". `ORDER BY Schnitt` funktioniert dagegen, weil ORDER BY (Schritt 6) *nach* SELECT ausgewertet wird. Mit dem Röntgenblick ist das keine Merkregel mehr, sondern direkt ablesbar.

Warum ändert ORDER BY nichts an der Zeilenzahl? Schritt 6 zeigt: Sortieren ordnet nur um. Erst LIMIT reduziert die Menge – und zwar *nach* dem Sortieren. Genau deshalb liefert `ORDER BY Schnitt LIMIT 1` die beste Klasse und nicht irgendeine.

Wo verschwinden NULL-Werte? Schritt 2 zeigt es konkret: `Note IS NOT NULL` entfernt Feli. Ohne diese Bedingung würde `AVG(Note)` NULL-Werte zwar ignorieren, aber `COUNT(*)` würde sie mitzählen – ein beliebter Prüfungsfehler, der sich hier direkt vorführen lässt.

5. Grenzen und Hinweise

Situation	Verhalten
Mehrere Statements im Editor	Hinweis: Der Röntgenblick untersucht genau eine Abfrage
`WITH …` (CTE)	Höflicher Hinweis; Tipp: die innere SELECT-Abfrage einzeln durchleuchten
`UNION` / `EXCEPT` / `INTERSECT`	Hinweis: Verbundabfragen werden nicht zerlegt
Abfrage ohne FROM (z. B. `SELECT 1`)	Hinweis: nichts zu zerlegen
Alias im `GROUP BY` (z. B. `GROUP BY Schnitt`)	Der betroffene Zwischenschritt zeigt einen Hinweis statt einer Vorschau; die übrigen Schritte laufen normal
Sehr große Tabellen	Jeder Schritt läuft im Hintergrund und zeigt bei Bedarf einen Abbrechen-Dialog – die Oberfläche friert nicht ein

Weitere Hinweise:

- Die Vorschau zeigt maximal **5 Zeilen** pro Schritt; die Zeilen-/ Gruppenzahl bezieht sich immer auf das vollständige Zwischenergebnis.
- Der Röntgenblick verändert **keine Daten** – alle Teilabfragen sind reine SELECTs.
- Benannte Parameter (`:Name`, `@Name`, `$Name`) werden einmal abgefragt und für alle Schritte verwendet. Positionsparameter (`?`) werden wie beim normalen Ausführen nicht unterstützt.

6. Tipps für den Unterricht

1. **Vorhersagen lassen:** Vor dem Klick auf 🔬 schätzen die Lernenden die Zeilenzahl jedes Schritts – der Dialog ist die Selbstkontrolle.

2. **Fehler provozieren:** `WHERE Schnitt < 4` schreiben lassen, Fehler lesen, dann mit dem Röntgenblick begründen, warum ORDER BY den Alias kennt und WHERE nicht.
3. **WHERE gegen HAVING tauschen:** Was passiert mit `HAVING Note IS NOT NULL`? Der Röntgenblick zeigt, warum die Bedingung an dieser Stelle nicht (oder anders) wirkt.
4. **LIMIT ohne ORDER BY:** `LIMIT 1` ohne Sortierung liefert eine *beliebige* Zeile – im Schritt 6 sichtbar, weil die Erklärung nur noch vom Zuschneiden spricht.

Siehe auch: Hilfe → Tool-Bedienung → „Bedien-Helfer" (F1) für die Kurzfassung direkt in DataBlick.