ALIASING

Aliasing in SQL: Datenbankabfragen mit Aliasnamen übersichtlicher gestalten

Aliasing ist eine Technik in SQL, mit der Spalten- oder Tabellennamen temporär umbenannt werden. Dies macht Abfragen übersichtlicher und erleichtert die Lesbarkeit, insbesondere bei komplexen Berechnungen oder wenn Daten aus mehreren Tabellen verknüpft werden.

Hier das von Masterschool verlinkte Youtube Intermediate SQL Tutorial | Aliasing

Was ist ein Alias?

Ein **Alias** ist ein temporärer Name, der einem Spalten- oder Tabellennamen für die Dauer einer Abfrage zugewiesen wird. Mithilfe des Schlüsselworts `AS` lassen sich kürzere oder aussagekräftigere Bezeichnungen definieren, ohne die zugrunde liegende Datenstruktur zu verändern.

Syntax für Aliasing:

SELECT spalte AS alias_name

FROM tabelle AS alias_name;Anwendungsfälle und Beispiele für Aliasing

1. **Alias für Spaltennamen**

Aliase für Spalten sind hilfreich, wenn:

- Der Original-Spaltenname unübersichtlich ist.
- Ein berechneter Wert unter einem spezifischen Namen angezeigt werden soll.

Beispiel: Angenommen, die Tabelle **EmployeeDemographics** enthält die Spalten `FirstName` und `LastName`. Wir können einen Alias verwenden, um eine neue Spalte `FullName` zu erstellen, die den vollständigen Namen enthält:

SELECT FirstName || ' ' || LastName AS FullName, Age

FROM EmployeeDemographics;**Erklärung:** Hier werden `FirstName` und `LastName` kombiniert und als `FullName` angezeigt.

2. **Alias für Tabellen**

Tabellen-Aliase sind besonders nützlich in komplexen Abfragen mit mehreren Tabellen oder JOINs, da sie die Abfrage lesbarer und kürzer machen.

Beispiel: Stellen wir uns eine Abfrage vor, die die Tabelle **EmployeeDemographics** und eine zweite Tabelle **EmployeeSalaries** verbindet, um das Gehalt für jeden Mitarbeiter anzuzeigen:

SELECT ed.FirstName, ed.LastName, es.Salary

FROM EmployeeDemographics AS edJOIN EmployeeSalaries AS es ON ed.EmployeeID = es.EmployeeID;**Erklärung:** In diesem Beispiel verwenden wir `ed` und `es` als Aliase für die Tabellen. So wird die Abfrage kürzer und übersichtlicher, insbesondere bei komplexeren JOINs.

3. **Alias für berechnete Felder**

Aliase eignen sich hervorragend für berechnete Felder, die direkt in einer Abfrage erstellt werden. Dies ist nützlich, um aggregierte Werte oder bestimmte Berechnungen mit aussagekräftigen Namen zu versehen.

Beispiel: Berechnen wir das Alter in Tagen, basierend auf dem Alter in Jahren:

SELECT FirstName, LastName, Age * 365 AS Age

InDaysFROM EmployeeDemographics;**Erklärung:** Hier zeigt `AgeInDays` das Alter der Mitarbeiter in Tagen an, basierend auf dem Wert in der Spalte `Age`.

4. **Alias in Subqueries (Unterabfragen)**

Aliasing ist auch in Unterabfragen hilfreich, wenn die Abfrage selbst komplex wird. Es erleichtert das Verständnis der verschachtelten Struktur und verbessert die Lesbarkeit.

Beispiel: Angenommen, wir möchten den Durchschnitt der Gehälter pro Abteilung anzeigen. Eine Unterabfrage könnte verwendet werden, um die Abteilungen und Gehälter zu gruppieren:

SELECT Department, AVG(Salary) AS AvgSalaryFROM (

SELECT Department, Salary

FROM EmployeeSalaries

) AS DeptSalariesGROUP BY Department;Zusammenfassung

- **Spalten-Aliase** geben berechneten oder zusammengeführten Feldern klare Namen.
- **Tabellen-Aliase** vereinfachen lange und verschachtelte Abfragen, besonders bei JOINs.
- **Aliase in Unterabfragen** helfen, komplexe Abfragen verständlicher und wartbarer zu gestalten.
-

Aliasing macht SQL-Abfragen lesbarer und strukturiert, was die Zusammenarbeit und das Verständnis von Abfragen erheblich erleichtert.

Hier nochmal eine detaillierte Erklärung

Verstehe! Ich erkläre es dir ganz von Grund auf, Schritt für Schritt, sodass es für einen Anfänger klar wird.

Hier ist die Abfrage, die wir uns anschauen:

SELECT Demo.employee_id, Sal.Salary

FROM employee_demographics AS Demo

JOIN employee_salary AS Sal

ON Demo.employee_id = Sal.employee_id

ORDER BY Sal.Salary;Lass uns das Wort für Wort durchgehen.

Grundkonzept: Was macht diese Abfrage?

Diese Abfrage zieht Informationen aus zwei Tabellen:

1. `employee_demographics` : Enthält demografische Informationen über Mitarbeiter, z. B. Namen und IDs.
2. `employee_salary` : Enthält Informationen über die Gehälter der Mitarbeiter.

Die Abfrage:

- Zeigt die **Mitarbeiter-ID** und das **Gehalt** jedes Mitarbeiters an, indem sie Daten aus beiden Tabellen verknüpft.
- **Sortiert** das Ergebnis nach dem Gehalt (aufsteigend, vom niedrigsten zum höchsten Gehalt).

Schritt-für-Schritt-Erklärung

1. SELECT Demo.employee_id, Sal.Salary

`SELECT` sagt SQL, dass wir Daten auswählen möchten, und zwar:

- `Demo.employee_id` : Dies zeigt die `employee_id` aus der ersten Tabelle (`employee_demographics`). **Demo** ist hier nur ein Kurzname, den wir später definieren.
- `Sal.Salary` : Dies zeigt das `Salary` -Feld aus der zweiten Tabelle (`employee_salary`). Auch **Sal** ist ein Kurzname.

Warum Kurzname? Anstatt immer die vollen Tabellennamen zu schreiben, können wir diese mit „Spitznamen“ (Alias) kürzer schreiben, was es uns später leichter macht.

2. FROM employee_demographics AS Demo

`FROM` sagt SQL, welche Tabelle wir verwenden möchten. Hier:

- `FROM employee_demographics` : Wir sagen, dass wir mit der Tabelle

`employee_demographics` beginnen.

- `AS Demo` : Wir geben dieser Tabelle den Spitznamen **Demo**. Ab jetzt können wir `Demo` statt `employee_demographics` schreiben.

3. JOIN employee_salary AS Sal

`JOIN` bedeutet „verbinde die beiden Tabellen“. Die zweite Tabelle, die wir verknüpfen möchten, ist `employee_salary` .

- `AS Sal` : Auch dieser Tabelle geben wir einen Spitznamen: **Sal**. Ab jetzt können wir `Sal` schreiben, anstatt `employee_salary` .

Zusammengefasst: Jetzt haben wir zwei Tabellen:

- **Demo** (für `employee_demographics`)
- **Sal** (für `employee_salary`)

4. ON Demo.employee_id = Sal.employee_id

`ON` beschreibt die Bedingung, unter der die Tabellen verknüpft werden sollen.

- `Demo.employee_id = Sal.employee_id` : Das bedeutet: Verbinde die beiden Tabellen, aber nur bei Zeilen, wo die `employee_id` in beiden Tabellen gleich ist.

Beispiel: Wenn die `employee_id` in `Demo` und `Sal` den Wert `1001` hat, dann wird diese Zeile in die Ergebnisliste aufgenommen. Wenn keine Übereinstimmung gefunden wird, bleibt die Zeile außen vor.

5. ORDER BY Sal.Salary

`ORDER BY` sagt SQL, dass die Ergebnisse sortiert werden sollen.

- `ORDER BY Sal.Salary` : Wir sortieren das Ergebnis nach `Salary` . Standardmäßig wird hier aufsteigend (vom niedrigsten zum höchsten Gehalt) sortiert.