SQL: ALIAS

We have seen this previously:

In SQL, an alias is a temporary name given to a table or column to make the query more readable, concise, or to rename a column or table for use within a query. Aliases are often used to simplify complex queries, provide more meaningful names to columns or tables, or to avoid name conflicts when joining tables.

```
SELECT column1 AS alias1, column2 AS alias2, ...
FROM table_name as alias3;
```

Example 1: Joining Multiple Tables

Suppose you have two tables: `Orders` and `Customers`. You want to retrieve the order ID, customer name, and total amount for each order. Using aliases can make this query clearer, especially since both tables might have columns with similar or identical names.

Tables:

`Orders` Table:

order_id	customer_id	order_date	total_amount
1	101	2024-01-05	250.00
2	102	2024-01-10	150.00
3	103	2024-01-15	400.00

`Customers` Table:

customer_id	customer_name
101	John Doe
102	Jane Smith
103	Alice Johnson

Query without aliases:

```
SELECT Orders.order_id, Customers.customer_name, Orders.tot
al_amount
```

```
FROM Orders
JOIN Customers ON Orders.customer_id = Customers.customer_i
d;
```

Query with aliases:

```
SELECT o.order_id, c.customer_name, o.total_amount
FROM Orders AS o
JOIN Customers AS c ON o.customer_id = c.customer_id;
```

Why the alias is helpful:

- **Simplifies references**: Instead of repeatedly typing out `Orders` and `Customers`, the aliases `o` and `c` make the query shorter and easier to read.
- **Clear context**: It quickly becomes clear which table each column is coming from, especially when both tables have a column named `customer_id`.

Example 2: Using Aliases for Calculated Columns

Imagine you want to calculate the total amount with tax included (assuming a tax rate of 10%). Aliases make the output more readable by providing meaningful names to calculated columns.

Query without alias:

```
SELECT order_id, total_amount, total_amount * 1.10
FROM Orders;
```

Query with alias:

```
SELECT order_id, total_amount, total_amount * 1.10 AS total
_with_tax
FROM Orders;
```

Why the alias is helpful:

- **Descriptive output**: The alias `total_with_tax` clearly indicates that this column represents the total amount including tax, making the results easier to interpret.

Example 3: Self-Join Using Table Aliases

A self-join is when a table is joined with itself. For instance, suppose you have an `Employees` table where each employee has a `manager_id` that refers to another employee's `employee_id`. You want to list employees along with their managers' names.

`Employees` Table:

employee_id	first_name	last_name	manager_id
1	John	Doe	NULL
2	Jane	Smith	1
3	Alice	Johnson	1
4	Bob	Brown	2

Query without aliases:

```
SELECT Employees.employee_id, Employees.first_name, Employe
es.manager_id, Managers.first_name
FROM Employees
JOIN Employees AS Managers ON Employees.manager_id = Manage
rs.employee_id;
```

Query with aliases:

```
SELECT e.employee_id, e.first_name, e.manager_id, m.first_n
ame AS manager_name
FROM Employees AS e
JOIN Employees AS m ON e.manager_id = m.employee_id;
```

Why the alias is helpful:

- **Disambiguation**: Using aliases `e` for the employee and `m` for the manager clarifies which role each instance of the `Employees` table is playing in the join.
- **Improves readability**: Shortens the table names, reducing visual clutter and making the query easier to follow.

Key Takeaways

- **Use aliases** when they make your queries easier to read and write, especially with multiple tables or self-joins.

- **Aliases improve clarity** when performing calculations or when column names are ambiguous.
- **Avoid unnecessary aliases** in simple queries where they do not add value or reduce clarity.

These examples demonstrate situations where using aliases can enhance the readability and functionality of SQL queries.