AS/ALIAS to Rename Tables and Columns

by Sophia

WHAT'S COVERED

In this lesson, you will learn how to use AS to create aliases for tables and columns. Specifically, this lesson will cover:

1. Table Aliases

A **table alias** enables us to assign tables or columns new names when a query is running. For a table in a SELECT statement, it could look like this:

```
SELECT *
FROM customer as c;
```

Essentially, we've renamed the table "c" instead of the customer. This can be very useful for making our queries easier to read when we start to join tables together, especially when we prefix our tables. For example:

```
SELECT album.album_id, artist.artist_id, track.track_id, album.title, artist.name, track.name
FROM track
INNER JOIN album USING (album_id)
INNER JOIN artist USING (artist_id)
ORDER BY album.album_id, artist.artist_id, track.track_id;
```

By using a shorter alias name for the tables, we can simplify the query, making it easier to read the column list:

```
SELECT al.album_id, ar.artist_id, t.track_id, al.title, ar.name, t.name
FROM track as t
INNER JOIN album as al USING (album_id)
INNER JOIN artist as ar USING (artist_id)
ORDER BY al.album_id, ar.artist_id, t.track_id;
```

KEY CONCEPT

When defining table aliases, you do not have to put the alias names in quotation marks unless they contain spaces. If you do use quotation marks, they must be double ones ("). Using single quotes will generate an error.

In our case, the table names are quite short. But imagine if much longer table names are used in the database. Alias names would be much easier to follow. Let's revisit the representative and department tables we created previously to see an example:

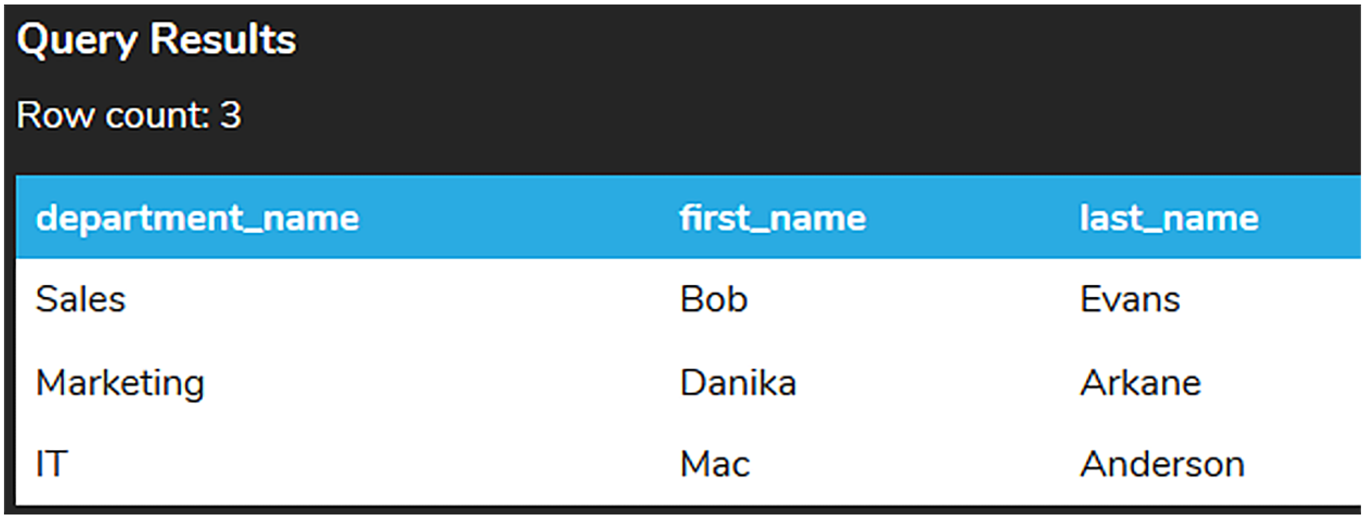
```
CREATE TABLE representative ( representative_id INT PRIMARY KEY, first_name VARCHAR (30) NOT NULL, last_name VARCHAR (30) NOT NULL );
CREATE TABLE department ( department_id INT PRIMARY KEY, department_name VARCHAR (100) NOT NULL, manager_id INT, constraint fk_manager FOR
INSERT INTO representative (representative_id, first_name, last_name)
VALUES (1, 'Bob','Evans'), (2, 'Tango','Rushmore'), (3, 'Danika','Arkane'), (4, 'Mac','Anderson');
INSERT INTO department (department_id, department_name,manager_id)
VALUES (1, 'Sales', 1), (2, 'Marketing', 3), (3, 'IT', 4), (4, 'Finance', null), (5, 'Support', null);
```

If we needed to use the prefixes for the column names, a query to list the departments and their managers would look like this:

```
SELECT department.department_name, representative.first_name, representative.last_name
FROM representative
JOIN department ON representative.representative_id = department.manager_id;
```

Through the use of table aliases, we can shorten all of the instances where we have the representative or department table names:

```
SELECT d.department_name, r.first_name, r.last_name
FROM representative as r
JOIN department as d ON r.representative_id = d.manager_id;
```



Query Results

Row count: 3

department_name	first_name	last_name
Sales	Bob	Evans
Marketing	Danika	Arkane
IT	Mac	Anderson

TERM TO KNOW

Table Alias
An alternative name assigned to a table for use in a query's result set.

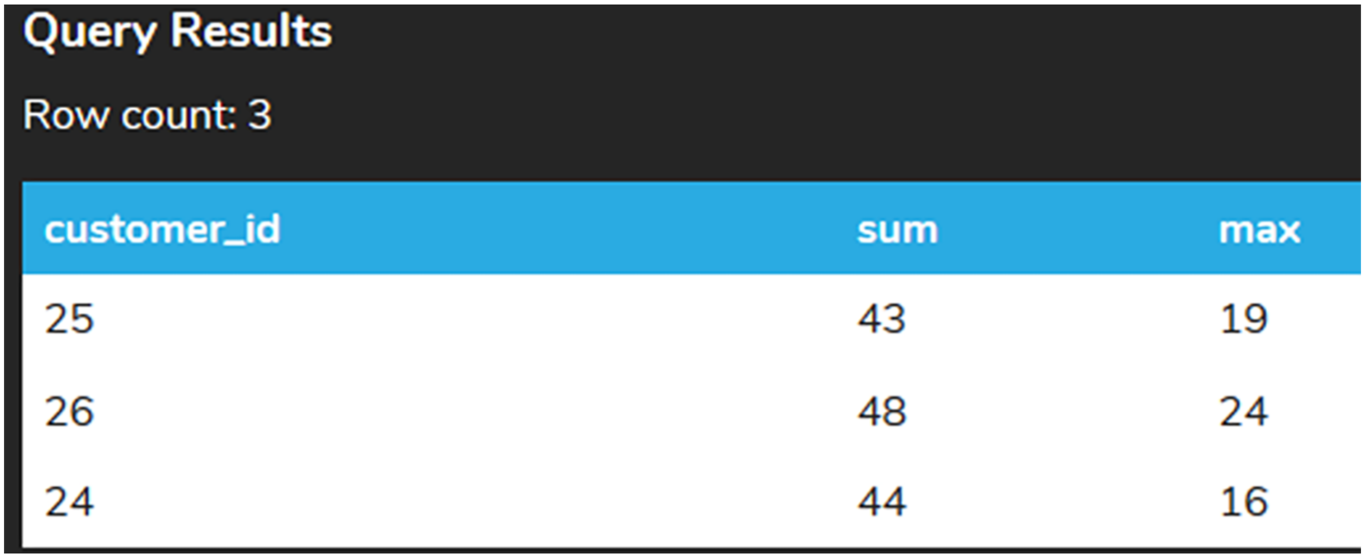
2. Column Aliases

We can also create aliases for column names within the SELECT clause. **Column aliases** can be especially useful for expressions. You may remember our lessons on aggregate functions, where the results would just show the function name. To a user reviewing the data, it may sometimes be confusing regarding what it represents.

For example, let's look back at one of the complex queries we created to find the max totals being greater than 15 and the customer_id being between 20 and 30:

```
SELECT customer_id, SUM(total), MAX(total)
FROM invoice
WHERE billing_country = 'USA'
GROUP BY customer_id
HAVING MAX(total) > 15
AND customer_id BETWEEN 20 AND 30;
```

The result set shows the following:



Query Results

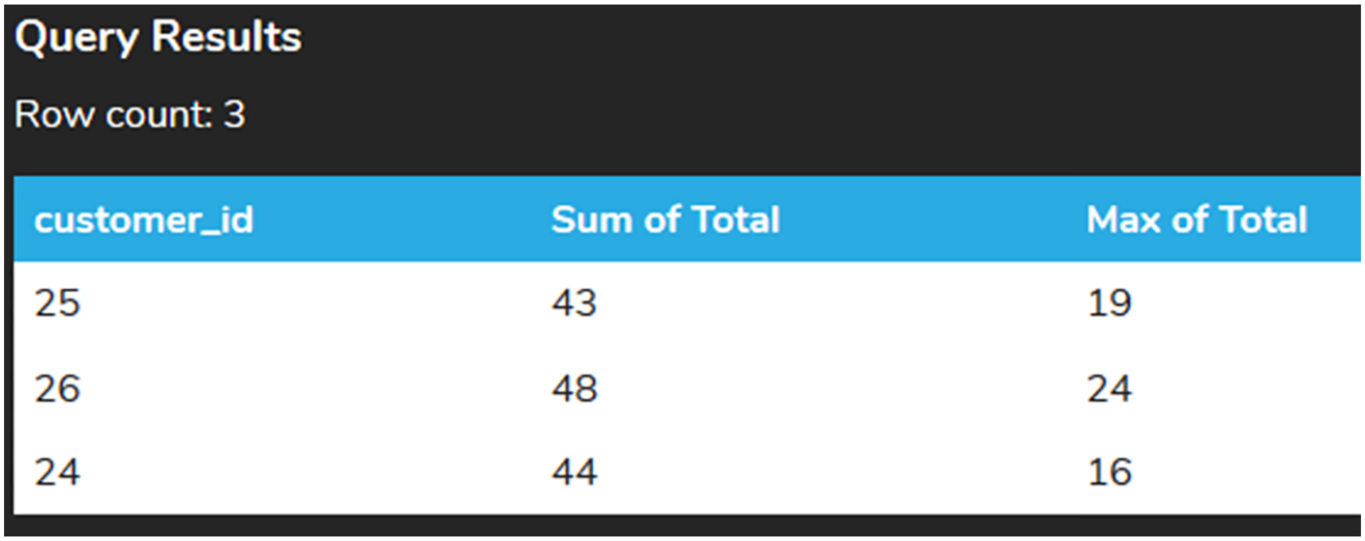
Row count: 3

customer_id	sum	max
25	43	19
26	48	24
24	44	16

We could change each of the column names so that it is clearer what each of the expressions represents:

```
SELECT customer_id, SUM(total) as "Sum of Total", MAX(total) as "Max of Total"
FROM invoice
WHERE billing_country = 'USA'
GROUP BY customer_id
HAVING MAX(total) > 15
AND customer_id BETWEEN 20 AND 30;
```

Notice that in this case, we used double quotes around the alias column names since there are spaces in them:



Query Results

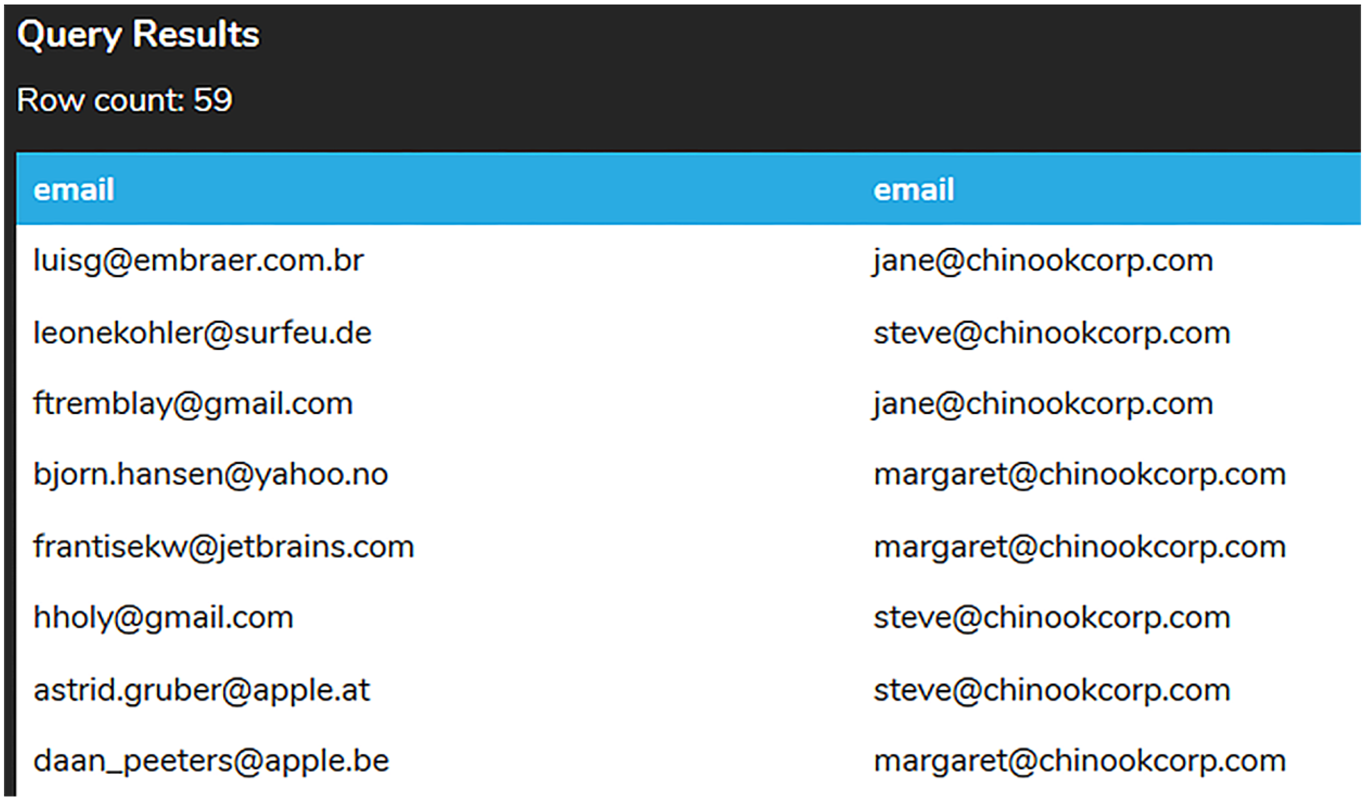
Row count: 3

customer_id	Sum of Total	Max of Total
25	43	19
26	48	24
24	44	16

This makes what the columns represent much clearer to the user.

Another example is the emails of the customers and the employees that support them:

```
SELECT customer.email, employee.email
FROM customer
INNER JOIN employee ON support_rep_id = employee_id;
```



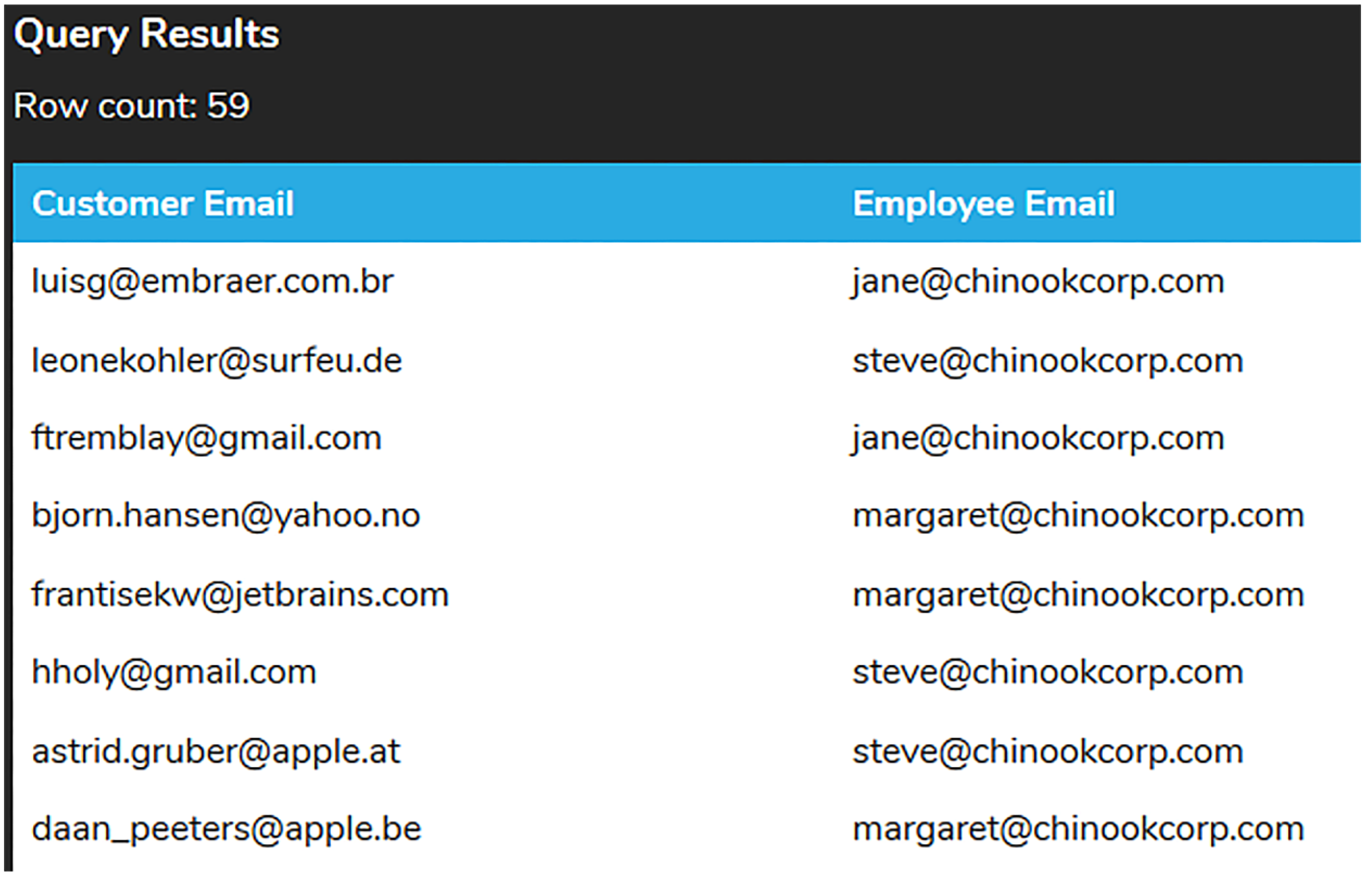
Query Results

Row count: 59

email	email
luisg@embraer.com.br	jane@chinookcorp.com
leonekohler@surfeu.de	steve@chinookcorp.com
ftremblay@gmail.com	jane@chinookcorp.com
bjorn.hansen@yahoo.no	margaret@chinookcorp.com
frantisekw@jetbrains.com	margaret@chinookcorp.com
hholy@gmail.com	steve@chinookcorp.com
astrid.gruber@apple.at	steve@chinookcorp.com
daan_peeters@apple.be	margaret@chinookcorp.com

Looking at the result set, it's unclear which email represents the customer's and the employee's emails, based on the column names. Here we can use both the table and alias to simplify the query and change the column names:

```
SELECT c.email as "Customer Email", e.email "Employee Email"
FROM customer AS c
INNER JOIN employee AS e ON support_rep_id = employee_id;
```



Query Results

Row count: 59

Customer Email	Employee Email
luisg@embraer.com.br	jane@chinookcorp.com
leonekohler@surfeu.de	steve@chinookcorp.com
ftremblay@gmail.com	jane@chinookcorp.com
bjorn.hansen@yahoo.no	margaret@chinookcorp.com
frantisekw@jetbrains.com	margaret@chinookcorp.com
hholy@gmail.com	steve@chinookcorp.com
astrid.gruber@apple.at	steve@chinookcorp.com
daan_peeters@apple.be	margaret@chinookcorp.com

WATCH

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

TERM TO KNOW

Column Alias

An alternative name assigned to a column for use in a query's result set.

SUMMARY

In this lesson, you learned that using column and **table aliases** in SQL queries can enhance their readability and conciseness, especially when dealing with long or complex table and column names. You can assign temporary names to the result set's columns using **column aliases**, resulting in a more intuitive and descriptive output. For example, you can rename calculated columns, aggregate functions, or columns with ambiguous names. It improves the clarity of query results and makes it easier for users to understand the data they are retrieving. In the same way, table aliases refer to renaming tables in a query with short and meaningful names. With aliases, long table names do not have to be repeatedly typed out, which is especially useful in queries involving multiple tables. SQL statements can be more succinct and easier to understand when using aliases.

TERMS TO KNOW

Column Alias

An alternative name assigned to a column for use in a query's result set.

Table Alias

An alternative name assigned to a table for use in a query's result set.