Filters to Specify Data

by Sophia

WHAT'S COVERED

This tutorial explores using various filters to return specific results using different options with the WHERE and HAVING clauses in two parts:

1. Understanding the Complexity
2. Stepping Through an Example

1. Understanding the Complexity

The complexity of the SELECT statements can increase based on the criteria being asked. In some cases, we may make use of both the WHERE and HAVING clauses. Recall that the WHERE clause is used as the filter for individual rows and the HAVING clause is for groups of rows.

Anything that we have in the aggregate function results are items that we can use in the HAVING clause. As an example, if we were asked to get all of the invoices and cost from the invoice_line table of those that had the cost as greater than 1, ordered by the invoice_id, the query would look like the following:

```
SELECT invoice_id, SUM(unit_price * quantity)
FROM invoice_line
GROUP BY invoice_id
HAVING SUM(unit_price * quantity) > 1
ORDER BY invoice_id;
```

Remember that the WHERE clause sees one row at a time, so we would not be able to evaluate the SUM across all of the invoice_id values. The HAVING clause is executed after the groups have been created.

Query Results

Row count: 357

invoice_id	sum
1	1.98
2	3.96
3	5.94
4	8.91
5	13.86

If we were asked to expand on this to find invoice_id values that were greater than 100, we could add this to the HAVING clause to act similar to the WHERE clause:

```
SELECT invoice_id, SUM(unit_price * quantity)
FROM invoice_line
GROUP BY invoice_id
HAVING SUM(unit_price * quantity) > 1
AND invoice_id > 100
ORDER BY invoice_id;
```

Query Results

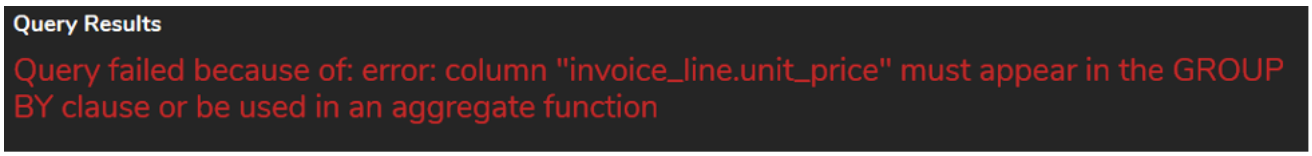
Row count: 270

invoice_id	sum
101	5.94
102	9.91
103	15.86
105	1.98
106	1.98

The reason we could do this is due to the fact that the invoice_id is part of the GROUP BY clause. However, if we needed to filter based on another column like the unit_price to check if it was more than 1 before we grouped them, and added it to the HAVING clause:

```
SELECT invoice_id, SUM(unit_price * quantity)
FROM invoice_line
GROUP BY invoice_id
HAVING SUM(unit_price * quantity) > 1
AND unit_price > 1
ORDER BY invoice_id;
```

We would have an error generated:



This is due to the fact that the unit_price column is not part of the GROUP BY field nor a result of an aggregate function. To be valid in the HAVING clause, we can only compare the aggregate functions or the column part of the GROUP BY. For it to be a valid query, the check on the unit_price needs to be moved to the WHERE clause:

```
SELECT invoice_id, SUM(unit_price * quantity)
FROM invoice_line
WHERE unit_price > 1
GROUP BY invoice_id
HAVING SUM(unit_price * quantity) > 1
ORDER BY invoice_id;
```

We should see now that the results look quite different:

Query Results

Row count: 30

invoice_id	sum
87	1.99
88	17.91
89	9.95
96	15.92
97	1.99

This is because we are filtering out the rows that have the unit_price being greater than 1 before we combine each into groups.

---

2. Stepping Through an Example

Let's look at another scenario where we are interested in invoices for a set of customers (customer_id between 20 and 30) that have their billing country in the USA. We want to find those that have had at least one invoice that has a total larger than 15. We want to also get the total amount they have ordered at all times. This may seem like a very complex query, but we will want to break the query down first.

First, we know that we are looking for data using the invoice table.

```
SELECT *
```

FROM invoice;

Exploring this data, we know that we want to look at a specific set of customers. We can identify two criteria without aggregate conditions that we can add to a WHERE clause:

```
SELECT *
FROM invoice
WHERE billing_country = 'USA'
AND customer_id BETWEEN 20 AND 30;
```

This gives us 63 records that fit these criteria:

Query Results

Row count: 63

invoice_id	customer_id	invoice_date	billing_address	billing_city	billing_state	billing_country	billing_postal_code	total
5	23	2009-01-11T00:00:00.000Z	69 Salem Street	Boston	MA	USA	2113	14
16	21	2009-03-05T00:00:00.000Z	801 W 4th Street	Reno	NV	USA	89503	4
17	25	2009-03-06T00:00:00.000Z	319 N. Frances Street	Madison	WI	USA	53703	6
38	21	2009-06-07T00:00:00.000Z	801 W 4th Street	Reno	NV	USA	89503	6
39	27	2009-06-10T00:00:00.000Z	1033 N Park Ave	Tucson	AZ	USA	85719	9
60	23	2009-09-11T00:00:00.000Z	69 Salem Street	Boston	MA	USA	2113	9
69	25	2009-10-25T00:00:00.000Z	319 N. Frances Street	Madison	WI	USA	53703	1

However, we need only the customer_id to be returned along with the SUM of the total and the MAX of the total. The customer_id should be what we are grouping by as well. This would change our SELECT statement to look like:

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

Query Results

Row count: 9

customer_id	sum	max
25	43	19
21	38	14
26	48	24
27	38	14
23	38	14
20	40	14
22	40	14
28	44	14
24	44	16

The next step is to find the groups that have the maximum of the total being greater than 15. As this is looking at an aggregate function, it has to go into the HAVING clause.

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

Query Results

Row count: 3

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

In looking at the query, one thing to note is that the customer_id is in the GROUP BY clause, so the comparison for the customer_id could have also appeared in the HAVING clause as well, like this:

```
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;
```

This would deliver the same result set:

Query Results

Row count: 3

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

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.

SUMMARY

Building a complex statement with various filters for rows and groups should be built one step at a time.

Source: Authored by Vincent Tran