# ANY and ALL Operators

*by Sophia*

WHAT'S COVERED

This tutorial explores using ANY and ALL operators to compare values with a list of values returned by a subquery.

1. Introduction
2. Subquery Example
3. Potential Error
4. Operators for Comparisons

## 1. Introduction

The ALL and ANY operators allow us to query data by comparing a value with a list of values that are returned by a subquery. This is an important distinction for the ANY and ALL operators, as they are focused on the lists from a subquery.

The syntax of the operator looks like:

<columnname> <operator> [ANY/ALL] (subquery);

The ANY operator is less restrictive than the ALL when it comes to comparisons. The ANY operator returns true if any of the values in the subquery meets the condition, otherwise it returns false. The ALL operator returns true if ALL of the values in the subquery meets the condition, otherwise it returns false.

## 2. Subquery Example

Let's take a look at an example where we're needing to compare the average invoice totals per country:

```
SELECT AVG(total)
FROM invoice
GROUP BY billing_country;
```

We will use that average by country as our subquery. If we wanted to find invoices that have a value higher than the average of any of the totals from the country, we would use the following:

```
SELECT *
FROM invoice
WHERE total > ANY(
SELECT AVG(total)
FROM invoice
GROUP BY billing_country);
```

Query Results
Row count: 179

| invoice_id | customer_id | invoice_date | billing_address | billing_city | billing_state | billing_country | billing_postal_code | total |
|---|---|---|---|---|---|---|---|---|
| 3 | 8 | 2009-01-03T00:00:00.000Z | Grétrystraat 63 | Brussels | | Belgium | 1000 | 6 |
| 4 | 14 | 2009-01-06T00:00:00.000Z | 8210 111 ST NW | Edmonton | AB | Canada | T6G 2C7 | 9 |
| 5 | 23 | 2009-01-11T00:00:00.000Z | 69 Salem Street | Boston | MA | USA | 2113 | 14 |
| 10 | 46 | 2009-02-03T00:00:00.000Z | 3 Chatham Street | Dublin | Dublin | Ireland | | 6 |

To find invoices that have a value higher than all of the averages from all countries:

```
SELECT *
FROM invoice
WHERE total > ALL(
SELECT AVG(total)
FROM invoice
GROUP BY billing_country);
```

Query Results
Row count: 123

| invoice_id | customer_id | invoice_date | billing_address | billing_city | billing_state | billing_country | billing_postal_code | total |
|---|---|---|---|---|---|---|---|---|
| 4 | 14 | 2009-01-06T00:00:00.000Z | 8210 111 ST NW | Edmonton | AB | Canada | T6G 2C7 | 9 |
| 5 | 23 | 2009-01-11T00:00:00.000Z | 69 Salem Street | Boston | MA | USA | 2113 | 14 |
| 11 | 52 | 2009-02-06T00:00:00.000Z | 202 Hoxton Street | London | | United Kingdom | N1 5LH | 9 |
| 12 | 2 | 2009-02-11T00:00:00.000Z | Theodor-Heuss-Straße 34 | Stuttgart | | Germany | 70174 | 14 |

Notice the count difference between the two queries. The first query (with the ANY operator) is less restrictive.

# 3. Potential Error

With the ANY and ALL operator, the subquery must return a single column to compare. If we return more than one column in the subquery, only the first result set is returned:

```
SELECT *
FROM invoice
WHERE (total,total) >= ALL(
SELECT AVG(total),max(total)
FROM invoice
GROUP BY billing_country);
```

Query Results
Row count: 123

| invoice_id | customer_id | invoice_date | billing_address | billing_city | billing_state | billing_country | billing_postal_code | total |
|---|---|---|---|---|---|---|---|---|
| 4 | 14 | 2009-01-06T00:00:00.000Z | 8210 111 ST NW | Edmonton | AB | Canada | T6G 2C7 | 9 |
| 5 | 23 | 2009-01-11T00:00:00.000Z | 69 Salem Street | Boston | MA | USA | 2113 | 14 |
| 11 | 52 | 2009-02-06T00:00:00.000Z | 202 Hoxton Street | London | | United Kingdom | N1 5LH | 9 |

If we just compared the total to the max of the totals:

```
SELECT *
FROM invoice
WHERE total >= ALL(
```

SELECT max(total)
FROM invoice
GROUP BY billing_country);
One record would be returned:


# 4. Operators for Comparisons

You may notice that the query above uses >= vs a >. We can compare the ANY and ALL using a variety of operators.

- If we use > ALL, the expression evaluates to true if a value in the subquery is greater than the largest value returned by the subquery.
- If we use >= ALL, the expression evaluates to true if a value in the subquery is greater than or equal to the largest value returned by the subquery.
- If we use < ALL, the expression evaluates to true if a value in the subquery is less than the smallest value returned by the subquery.
- If we use <= ALL, the expression evaluates to true if a value in the subquery is less than or equal to the smallest value returned by the subquery.
- If we use = ALL, the expression evaluates to true if a value in the subquery is equal to every value returned by the subquery.
- If we use != ALL, the expression evaluates to true if a value in the subquery is not equal to any value returned by the subquery.
- If we use > ANY, the expression evaluates to true if a value in the subquery is greater than the smallest value returned by the subquery.
- If we use >= ANY, the expression evaluates to true if a value in the subquery is greater than or equal to the smallest value returned by the subquery.
- If we use < ANY, the expression evaluates to true if a value in the subquery is less than the largest value returned by the subquery.
- If we use <= ANY, the expression evaluates to true if a value in the subquery is less than or equal to the largest value returned by the subquery.
- If we use = ANY, the expression evaluates to true if a value in the subquery is equal to any value returned by the subquery. It works like the IN operator.
- If we use <> ANY, this expression is not the same as the NOT IN. Rather, if we had a scenario where we had column <> ANY (a, b, c), it would look like x <> a OR x <>b OR x<>c.

---

## Video Transcription

[MUSIC PLAYING] Sub-queries aren't an issue if it returns a single grow as part of the result set. However, if we have an instance where we want to try to group some information based off of certain criteria.

For example, we might want to group it based on the building country within this piece here. If we run this part individually, we'll be able to see that it returns multiple different rows and result sets. However, if we try to run this query now, you'll see that it comes up with an error. The reason for that is because the sub-query itself is returning multiple different items. However, with our comparisons can only compare to a single option. We can use any and all as options becomes as part of the sub-query component, so that if we use any and we do a value that's greater than any, it'll check any value that's going to be part of the result set.

So basically with a greater than check, it's a look for the lowest value associated with it, and then do a comparison if that total is greater than the lowest value being returned, it's going to allow that to be displayed.

If we made a modification to it using all, it's going to ensure that whatever toll this is greater than all the values that are being returned. So basically in this case here, it's going to be larger than the largest value.

[MUSIC PLAYING]


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.


In this tutorial, we learned to use the ALL or ANY operator to compare values to a list of values returned by a subquery.

Source: Authored by Vincent Tran