VIEW & Complex Queries

by Sophia

WHAT'S COVERED

This tutorial explores using views to provide a useful report on a complex set of data in two parts:

1. Introduction
2. VIEW In Practice

1. Introduction

Views can also be used to help simplify very complex queries such as queries that require subqueries or ones that use aggregate data. This can get very complex without the use of views, especially if you have to write out the statements each time. The use of the views helps simplify a lot of that underlying content.

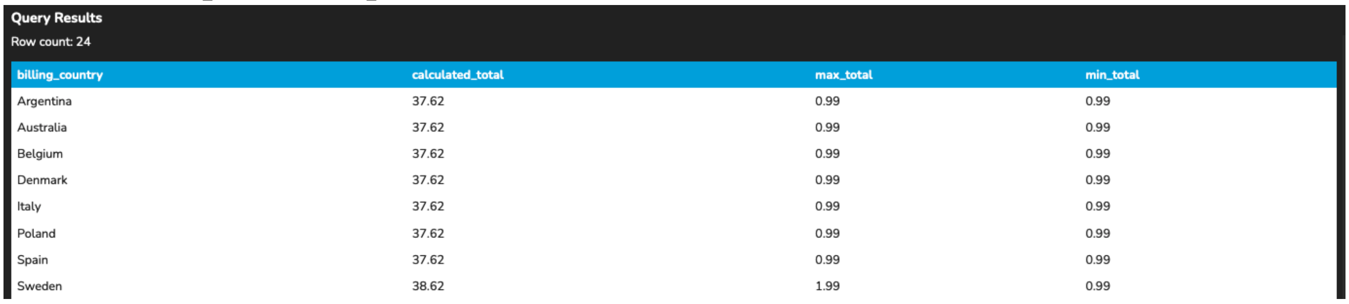
2. VIEW In Practice

For example, we may have a view created to list the calculated total of the amount per country:

```
CREATE VIEW invoice_country
AS
SELECT billing_country, SUM(quantity*unit_price) AS calculated_total, MAX(quantity*unit_price) AS max_total, MIN(quantity*unit_price) AS min_total
FROM invoice
INNER JOIN invoice_line ON invoice.invoice_id = invoice_line.invoice_id
GROUP BY invoice.billing_country;
```

This can get complex to look at specific criteria on the aggregate calculations but querying the view would result in the following results:

```
SELECT *
FROM invoice_country
ORDER BY calculated_total ASC, billing_country;
```



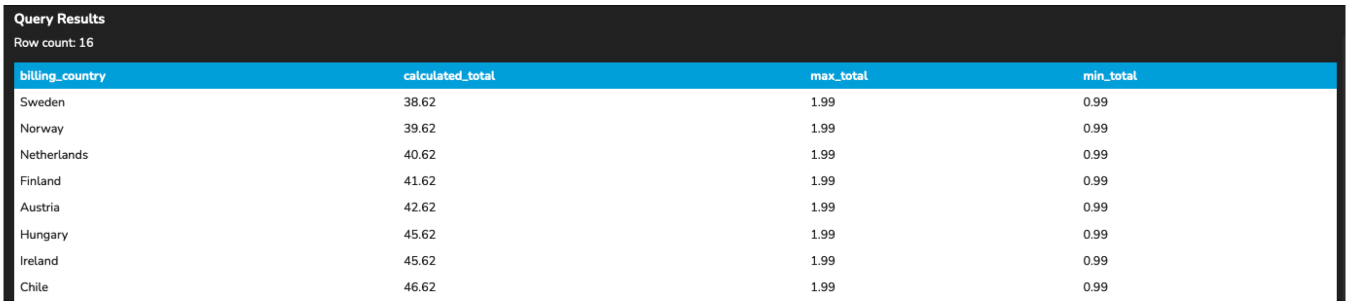
Query Results

Row count: 24

billing_country	calculated_total	max_total	min_total
Argentina	37.62	0.99	0.99
Australia	37.62	0.99	0.99
Belgium	37.62	0.99	0.99
Denmark	37.62	0.99	0.99
Italy	37.62	0.99	0.99
Poland	37.62	0.99	0.99
Spain	37.62	0.99	0.99
Sweden	38.62	1.99	0.99

From there, we may want to filter that data out further by doing:

```
SELECT *
FROM invoice_country
WHERE max_total = 1.99
ORDER BY calculated_total ASC, billing_country;
```

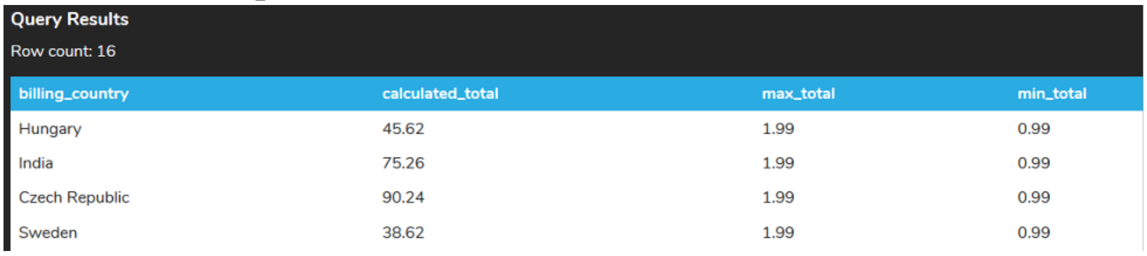

Query Results

Row count: 16

billing_country	calculated_total	max_total	min_total
Sweden	38.62	1.99	0.99
Norway	39.62	1.99	0.99
Netherlands	40.62	1.99	0.99
Finland	41.62	1.99	0.99
Austria	42.62	1.99	0.99
Hungary	45.62	1.99	0.99
Ireland	45.62	1.99	0.99
Chile	46.62	1.99	0.99

Compare this to the full statement on the base tables to get that same information:

```
SELECT billing_country, SUM(quantity*unit_price) AS calculated_total, MAX(quantity*unit_price) AS max_total, MIN(quantity*unit_price) AS min_total
FROM invoice
INNER JOIN invoice_line
ON invoice.invoice_id = invoice_line.invoice_id
GROUP BY invoice.billing_country
HAVING MAX(quantity*unit_price)  = 1.99;
```

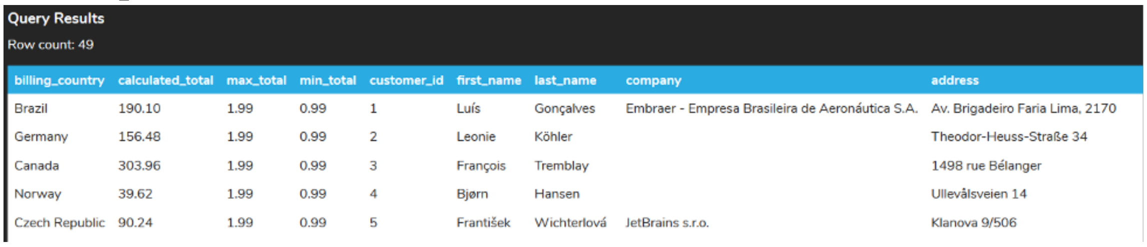

Query Results

Row count: 16

billing_country	calculated_total	max_total	min_total
Hungary	45.62	1.99	0.99
India	75.26	1.99	0.99
Czech Republic	90.24	1.99	0.99
Sweden	38.62	1.99	0.99

We can take the data from the view and use the information to query other tables. Using the same example above, if we find that the max_total = 1.99, we may want to list all of the customers in those countries while listing all of the countries' calculated criteria:

```
SELECT *
FROM invoice_country
INNER JOIN customer ON invoice_country.billing_country = customer.country
WHERE max_total = 1.99;
```


Query Results

Row count: 49

billing_country	calculated_total	max_total	min_total	customer_id	first_name	last_name	company	address
Brazil	190.10	1.99	0.99	1	Luís	Gonçalves	Embraer - Empresa Brasileira de Aeronáutica S.A.	Av. Brigadeiro Faria Lima, 2170
Germany	156.48	1.99	0.99	2	Leonie	Köhler		Theodor-Heuss-Straße 34
Canada	303.96	1.99	0.99	3	François	Tremblay		1498 rue Bélanger
Norway	39.62	1.99	0.99	4	Bjørn	Hansen		Ullevålsveien 14
Czech Republic	90.24	1.99	0.99	5	František	Wichterlová	JetBrains s.r.o.	Klanova 9/506

Think about what that the query would need to look like if you queried this using just the base tables. Consider the complexity of that type of statement. Aggregate data scenarios like this would be very difficult to create without the use of views.

Imagine, too, if you wanted to link all of the tables in our database together through a consistent data set to get a list of the track names that a customer has purchased. Rather than writing that query each time, having a view that joins all of the tables together may make it much simpler to query.

Video Transcription

[MUSIC PLAYING] Aggregate data, such as this for views, can be one of the easiest ways to be able to simplify a lot of the underlying select statements. For example in our case here, we're going to create a view that queries the invoice and invoice_line table together to identify the information associated with the totals in terms of the sum, the max, and the minimum for each country.

So in this case here, the query itself can be a little bit complex to be able to remember to write each time. In order to query from this particular view directly, we can go ahead and enter in something very simple once our view is created. We'll just a select star from the invoice country. If we select that, we'll get all the underlying information directly there.

Utilizing this you can actually join this with other tables as well. For example, if you want to join this with the customer table to be able to identify, say all the information within this table combined with the customer table data based off of a country, we can also do that as well, and then identify certain criteria that we're limit and filter from.

For example with the max total, we'll check on that to ensure that the value is equal to $1.99. This would be the simple approach in this case here. I can do a little query from the two way tables or the table and the view together and then having the resulting information being presented in this case. This would not be something that would be simple to be able to query utilizing just a basic select statement to be able to get that information.

[MUSIC PLAYING]

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

Complex queries can be created with views to help simplify a lot of the processing, especially when using aggregate functions.

Source: Authored by Vincent Tran