VIEW & Complex Queries

by Sophia

WHAT'S COVERED

This lesson explores using views to provide a useful report on a complex set of data. Specifically, this lesson will cover:

1. Basing a Complex Query on an Existing View

Views can simplify processing aggregated data by precomputing and storing aggregated data, making it readily accessible to users. This saves time for both the humans typing the commands and the database software processing them. Calculations for complex aggregations can be performed ahead of time and stored as a virtual table. This eliminates the need to recompute aggregations with each query.

Querying views using simple SELECT statements makes querying more intuitive and reduces the chance of errors. Precomputing aggregate values in a view ensures consistent aggregation logic across multiple queries, which helps maintain data accuracy and ensures that all users retrieve the same aggregate values.

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 m
FROM invoice
INNER JOIN invoice_line ON invoice.invoice_id = invoice_line.invoice_id
GROUP BY invoice.billing_country;
```

It 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 the following:

```
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 m
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 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 using views.

Imagine, too, if you wanted to link all of the tables in our database together through a consistent dataset 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.

WATCH

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter 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

In this lesson, you learned the benefits of creating views to perform complex aggregation operations, and then use the views as a basis for future queries you might create that require those aggregate values. Some of the benefits of this approach include consistency, reduced processing time, and reduced labor. Then you looked at some examples of **basing complex queries on existing views**, making queries simple that would otherwise be complex and difficult to create.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND FAITHE WEMPEN (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR **TERMS OF USE**.