Google Cloud

Types of derived tables

Previously, you learned that a derived table uses results from a query as if it were an actual table in the database. These queries are nested within an outer query, which uses the results of the derived table to create its own results. Sometimes, you might want to join or generate data in a way that isn't efficient or even possible when creating Views or Explores. Derived tables are useful when your existing tables don't have the fields, aggregations, or formats you need for your dashboard reports. Using a derived table can help you join data, create custom calculations, or generate necessary columns for your reporting needs. In this reading, you'll explore the two types of derived tables: Native derived and SQL-based. You'll also learn how derived tables persist.

Native derived tables

Native derived tables use existing LookML dimensions and measures. This helps create cleaner, easier to read code for data models. For example, you can create a native derived table to calculate sales data like Total Order Amount by Customer, or determine whether an order was a customer's first purchase. These calculations don't exist in the native table, but by creating a derived table query, you can generate them.

This is the query you would write for this scenario:

Unset

```
include: "/models/**/thelook.model.lkml"
view: order_facts {
      view_label: "Order Facts"
      derived_table: {
            explore_source: order_items {
                  column: order_id {field:
order_items.order_id_no_actions }
                  column: items_in_order { field: order_items.count
}
                  column: order_amount { field:
order_items.total_sale_price }
                  column: order_cost { field:
inventory_items.total_cost }
                  column: user_id {field: order_items.user_id }
                  column: created_at {field:
```

```
order_items.created_raw}
                column: order_gross_margin {field:
order_items.total_gross_margin}
                derived_column: order_sequence_number {
                     sql: RANK() OVER (PARTITION BY user_id
ORDER BY created_at) ;;
                }
           }
           datagroup_trigger: ecommerce_etl
     }
```

SQL-based derived tables

SQL-based derived tables use SQL to define the derived table instead of existing native tables. This is useful when you need to apply specific functions or data structures that aren't available in LookML. For example, this Repeat Purchase Facts table is generated completely using SQL code:

Unset

```
view: repeat_purchase_facts {
     view_label: "Repeat Purchase Facts"
     derived_table: {
          datagroup_trigger: ecommerce_etl
          sql: SELECT
                order_items.order_id as order_id
                , order_items.created_at
                , COUNT(DISTINCT repeat_order_items.id) AS
number_subsequent_orders
                , MIN(repeat_order_items.created_at) AS
next_order_date
                , MIN(repeat_order_items.order_id) AS
next_order_id
           FROM looker-private-demo.ecomm.order_items as
order_items
           LEFT JOIN looker-private-demo.ecomm.order_items
repeat_order_items
                ON order_items.user_id =
repeat_order_items.user_id
```

```
                AND order_items.created_at <
repeat_order_items.created_at
          GROUP BY 1, 2
                ;;
          }
```

How derived tables persist

Because derived tables are queries based on other tables, they don't necessarily persist.To ensure they remain present and useful, you have to define a persistence strategy in the parameters of your code. There are two types of persistence strategy parameters: Looker-managed and database-managed.

The easiest persistence parameters to use are the Looker-managed parameters. This means Looker executes whatever persistence strategy you define automatically. There are four Looker-managed persistence parameters that can be added to a derived table's query:

1. sql_trigger_value: This parameter triggers the regeneration of a persistent derived table (PDT) based on a SQL statement that you provide.
2. interval_trigger: This parameter triggers the regeneration of a PDT based on a time interval that you provide.
3. datagroup_trigger: Datagroups are the most flexible method for initiating persistence measures for derived tables. If you define the datagroup using sql_trigger or interval_trigger, you can use datagroup_group trigger to instruct Looker to build a PDT.
4. persist_for: This parameter sets a time limit for how long the derived table should persist before it expires.

Database-managed persistence parameters are more advanced. The primary database-managed persistence parameter is materialized_view: yes:.

- materialized_view: yes: A materialized view is a query result that's stored as a table in the scratch schema of your database. It's similar to PDTs, except that the database maintains and refreshes the data instead of Looker.

Key takeaways

Derived tables are a great way to generate useful tables to draw insights from your reports if the data you need isn't already available in your View or Explores. Whether you're creating a derived table based on existing elements, or using raw SQL, using a derived table can help you join data, create custom calculations, or generate necessary columns to meet your reporting requirements. As a cloud data professional working with tools like LookML, recognizing how

you can apply sub-queries like derived tables will make your work developing dashboards as code much easier.