**Project: - Big Querry Sales Analysis Using**

**Google Cloud Platform (GCP)**



**Name : -** Dev Shankar

**Institute : -** Tecnia institute of advance

studies

**Company : -** Skilled Up

**Software used : -** Google Cloud Platform

**Intern ID : -**  INT00060

**Table of Contents**

1. **Introduction**  
     1.1 Objective of the Report  
     1.2 Tools and Technologies Used  
     1.3 Scope of Analysis
2. **Category-Level Analysis**  
     2.1 Total Revenue by Category  
     2.2 Average Order Value (AOV) by Category
3. **Order and Customer Overview**  
     3.1 Total Orders, Revenue, and Customers (Category-wise)  
     3.2 Total Unique Customers  
     3.3 Repeated Customers Analysis  
     3.4 New Customer Orders
4. **Time-Based Performance Metrics**  
     4.1 Year-to-Month-To-Date (YMTD) Orders  
     4.2 YMTD Customers  
     4.3 YTD Customers  
     4.4 YMTD Order Growth (Current vs. Last Year)  
     4.5 YMTD Revenue Growth (Current vs. Last Year)
5. **Product-Level Analysis**  
     5.1 Top Products by Revenue  
     5.2 Quantity Sold and Customer Reach
6. **Year-Wise Analysis**  
     6.1 Total Orders and Revenue by Year  
     6.2 Year & Category Blend Overview
7. **Looker Studio Dashboards**  
     7.1 Visual Insights Summary  
     7.2 Key Trends and Patterns
8. **Conclusion & Recommendations**  
     8.1 Summary of Insights  
     8.2 Strategic Takeaways
9. **Appendix**  
     9.1 SQL Query References  
     9.2 Data Assumptions and Notes

**Introduction**

This report provides a detailed sales performance analysis for the dataset-powered e-commerce. The intention is to reveal actionable findings from transactional data with structured SQL queries complemented by dynamic visualizations through Looker Studio.

The report is divided into several analytical levels to analyse business performance from different perspectives:

•Category-Level Analysis: Knowing which product categories generate the highest revenue, orders, and customer interactions.

•Customer Segmentation: Segmenting between new and repeat customers to track customer acquisition and loyalty success.

•Order Metrics: Monitoring year-wise, total orders, average order value (AOV), and year-wise growth to evaluate operating throughput.

•Revenue Trends: Monitoring Year-to-Date (YTD) and Year-to-Month-To-Date (YMTD) revenue to analyze short-term and long-term sales growth.

• roduct-Level Insights: Determining the highest-grossing products with respect to revenue, orders, and customers.

•Annual Sales Performance: Analyzing year-over-year growth and category performance over time to inform strategic planning.

Each section is driven by SQL queries run on Google BigQuery and organized to enable scalable analytics. Interactive visualizations are added to the findings by Looker Studio dashboards.

**1.1 Objective of the Report**

The objective of this report is to analyze e-commerce sales data to uncover key performance trends across product categories, customer segments, and time periods. It supports data-driven decision-making in marketing, sales strategy, and customer retention.

**1.2 Tools and Technologies Used**

The analysis was conducted using **Google BigQuery** for data querying and processing, and **Looker Studio** for interactive visualizations and dashboards.

**1.3 Scope of Analysis**

The report covers category-wise performance, customer behavior (new vs. repeat), time-based trends (YMTD/YTD), product-level insights, and year-over-year growth comparisons.

1. **Category level view: -**

SELECT

c.string\_field\_1 AS category\_name,

  COUNT(DISTINCT o.order\_id) AS total\_orders

FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

  ON o.order\_id = oi.order\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Products` p

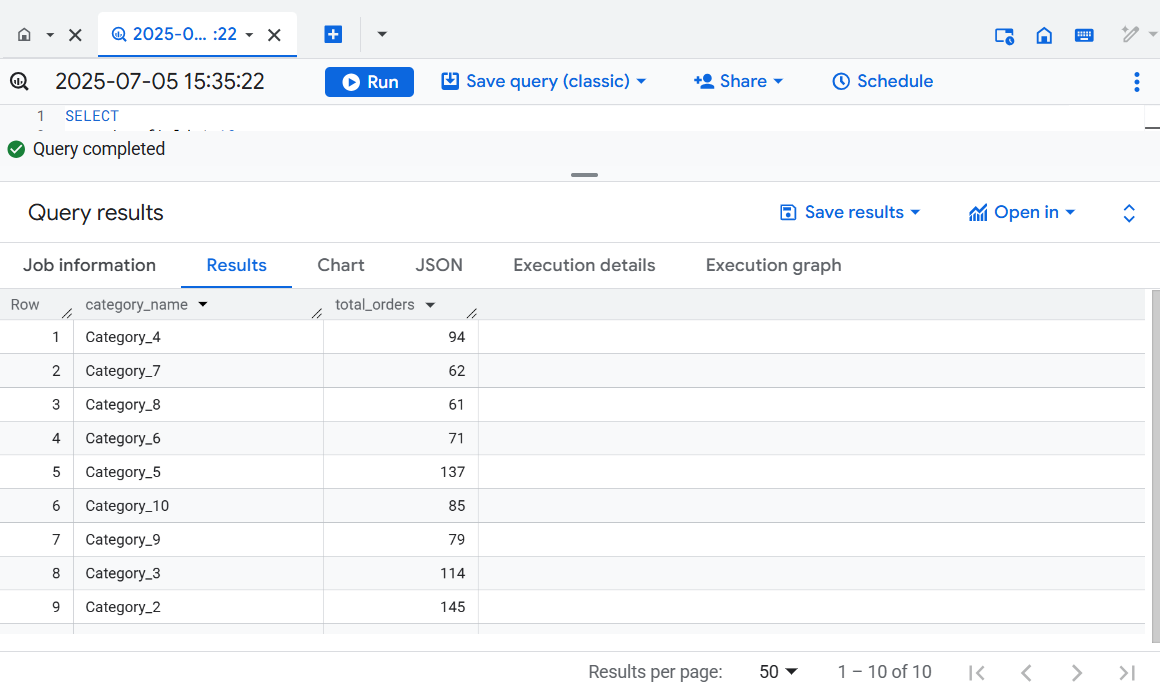
  ON oi.product\_id = p.product\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Category` c

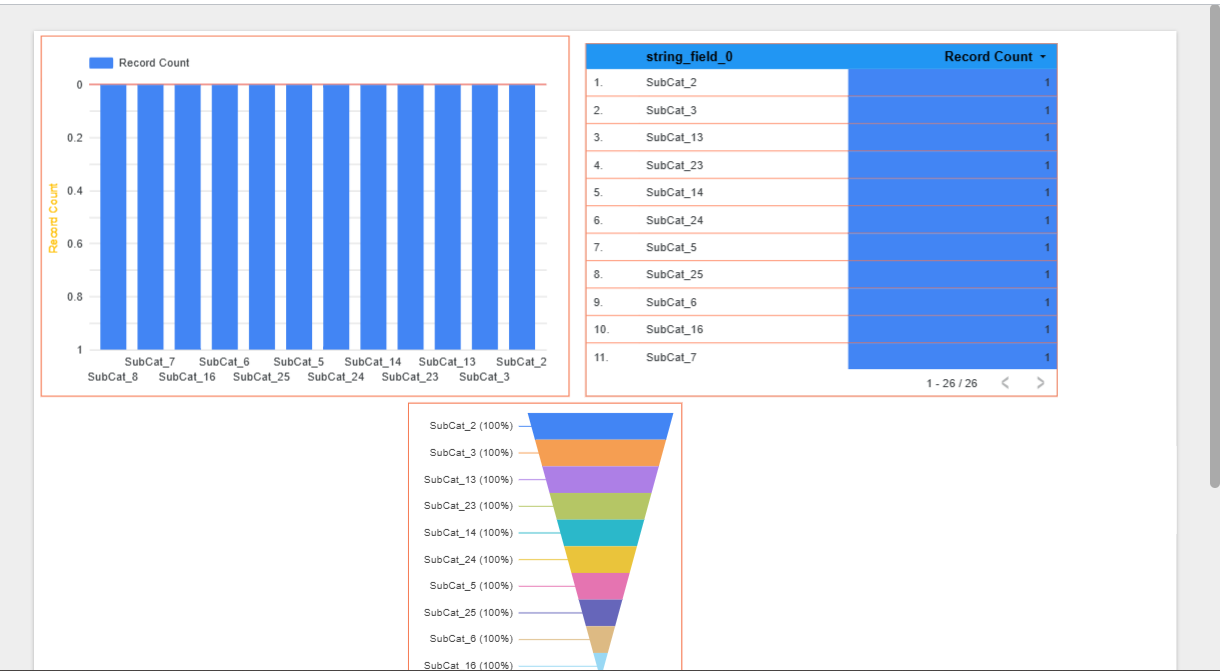
  ON p.category = c.string\_field\_1

GROUP BY category\_name;

**Output:**

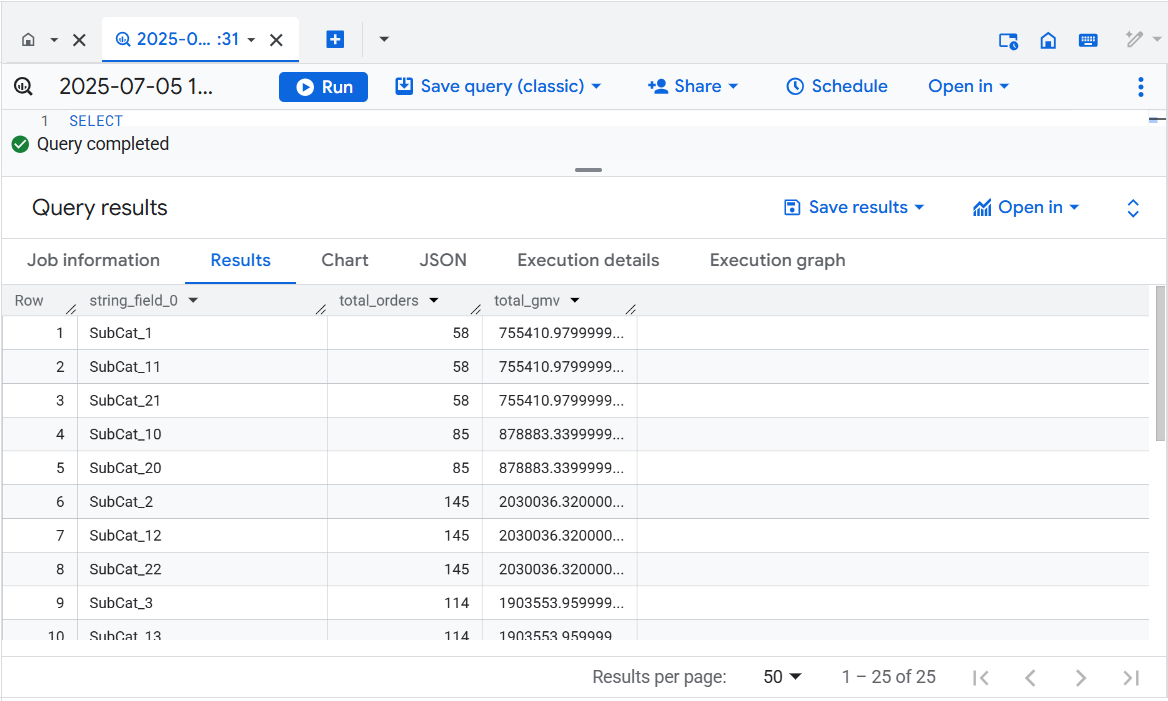


**Looker studio result: -**

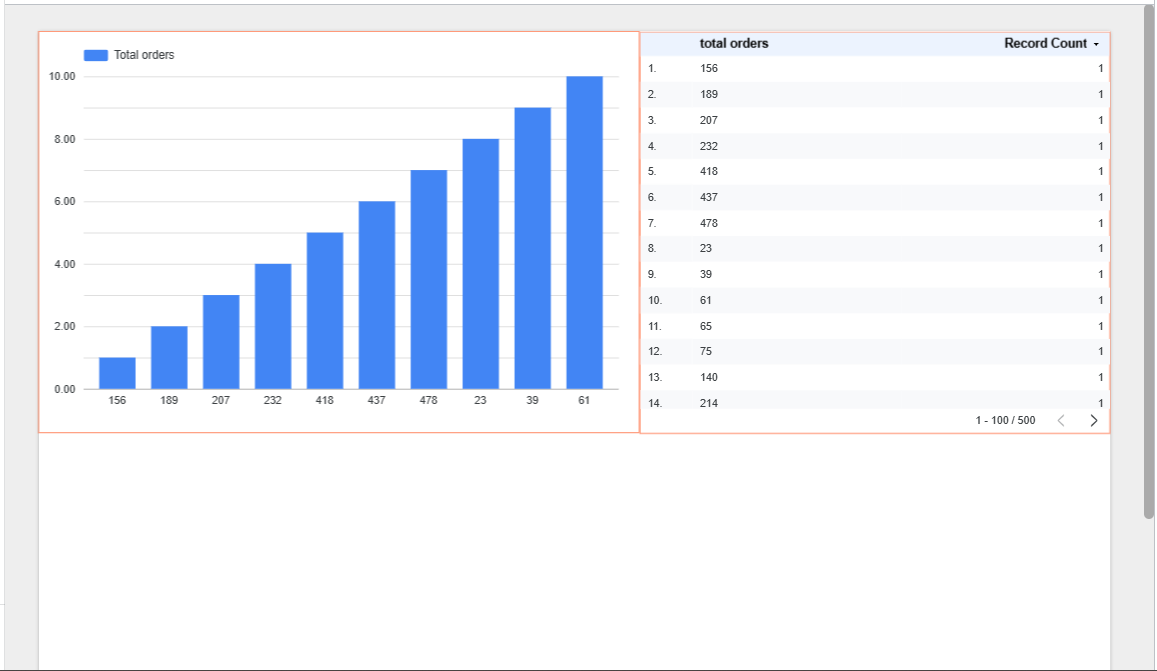


1. **Total order**
2. SELECT
3. string\_field\_0,
4. COUNT(DISTINCT o.order\_id) AS total\_orders,
5. SUM(oi.item\_total \* oi.quantity) AS total\_gmv
6. FROM `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi
7. JOIN `sales-analysis-sys.Sales\_Analysis\_system.Products` p
8. ON oi.product\_id = p.product\_id
9. JOIN `sales-analysis-sys.Sales\_Analysis\_system.Category` c
10. ON p.category = c.string\_field\_1
11. JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o
12. ON o.order\_id = oi.order\_id
13. GROUP BY c.string\_field\_0;

**Output:**

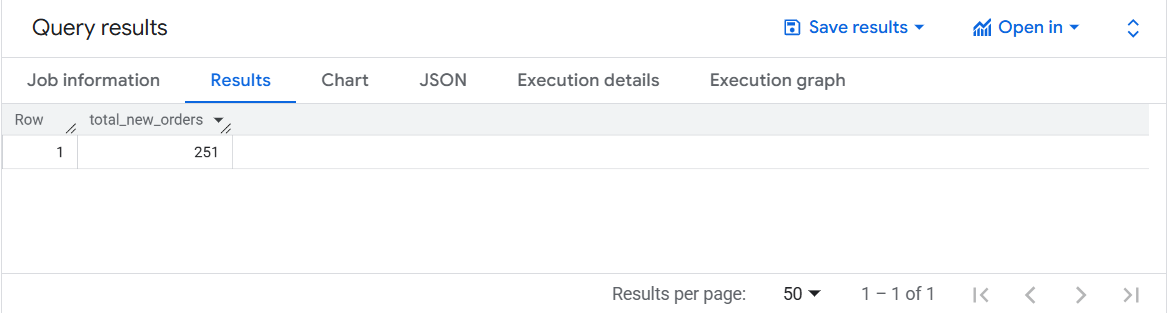


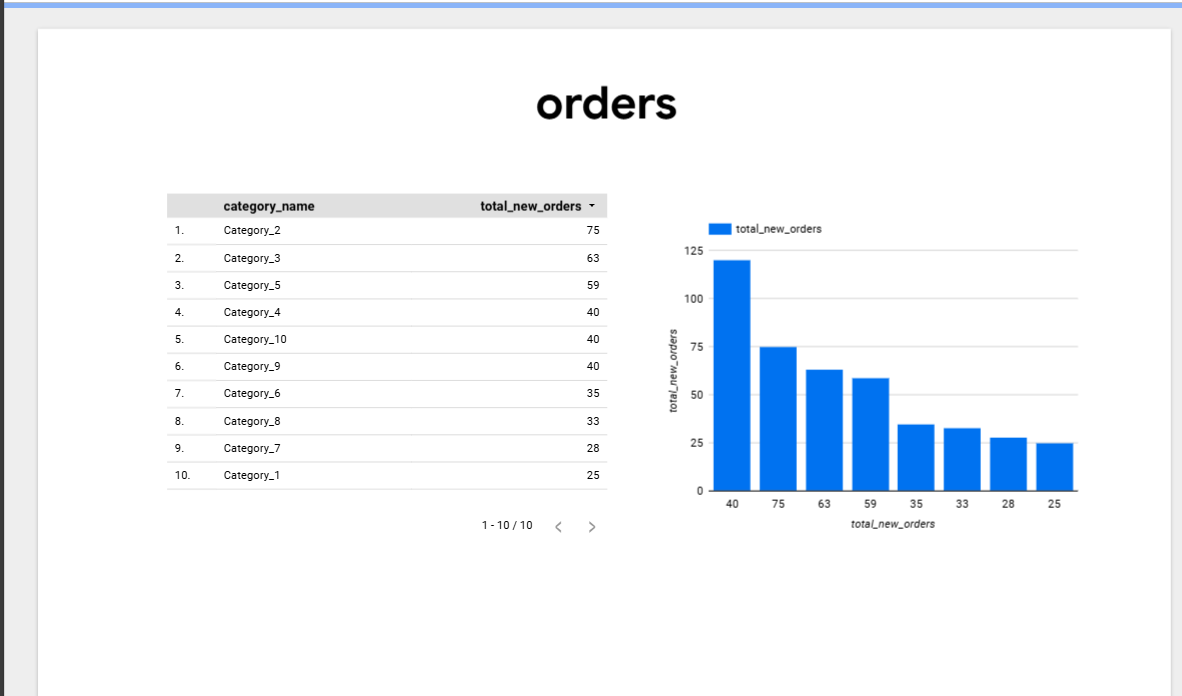
**Looker studio:**



1. **Total new order**
2. SELECT
3. COUNT(\*) AS total\_new\_orders
4. FROM (
5. SELECT customer\_id, MIN(order\_id) AS first\_order
6. FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`
7. GROUP BY customer\_id
8. );

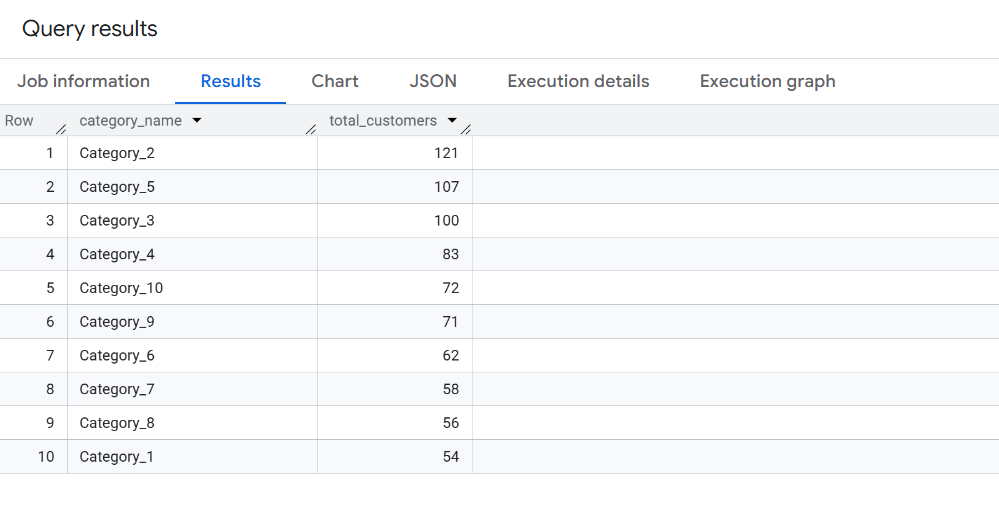
**Output:**

  
**looker studio result:**

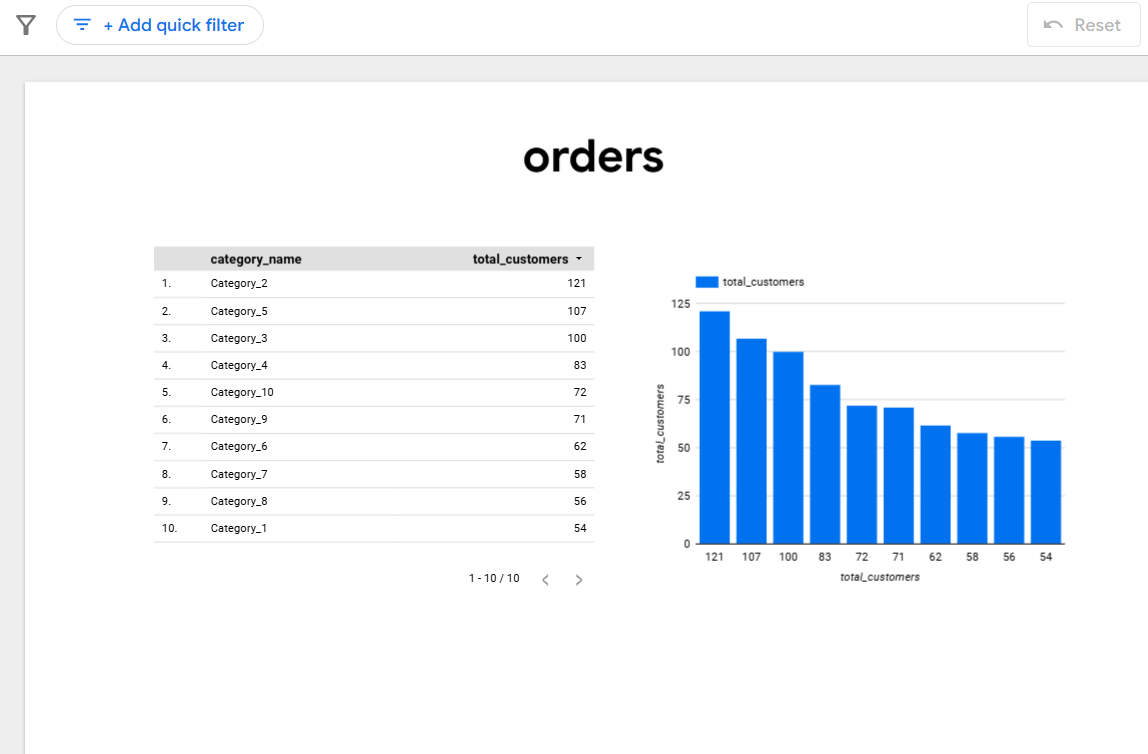


1. **Total customer :**
2. SELECT
3. COUNT(\*) AS total\_customers
4. FROM (
5. SELECT customer\_id, MIN(order\_id) AS first\_order
6. FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`
7. GROUP BY customer\_id
8. );

**Output:**

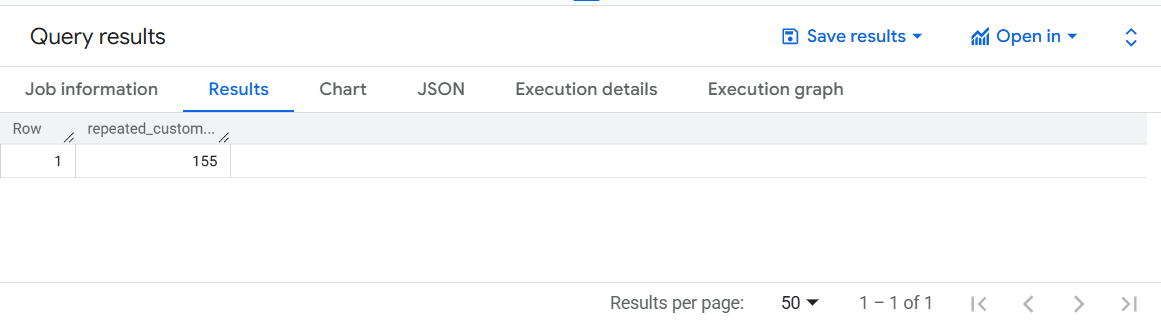


Looker studio result:



1. **Repeated customer**
2. SELECT
3. COUNT(\*) AS repeated\_customers
4. FROM (
5. SELECT customer\_id
6. FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`
7. GROUP BY customer\_id
8. HAVING COUNT(order\_id) > 1
9. );

**Output:**

****

**Looker studio result:**



1. **Total Orders Year-to-Month-Date (YMTD)**

-- Total Orders YMTD

SELECT

  COUNT(order\_id) AS total\_orders\_YMTD

FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`

WHERE DATE(order\_date) BETWEEN '2025-06-01' AND CURRENT\_DATE();

-- Total GMV YMTD

SELECT

  SUM(oi.order\_id \* oi.quantity) AS total\_gmv\_YMTD

FROM `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

  ON oi.order\_id = o.order\_id

WHERE DATE(o.order\_date) BETWEEN '2025-06-01' AND CURRENT\_DATE();

-- Total Revenue YMTD

SELECT

  SUM(p.amount) AS total\_revenue\_YMTD

FROM `sales-analysis-sys.Sales\_Analysis\_system.Payments` p

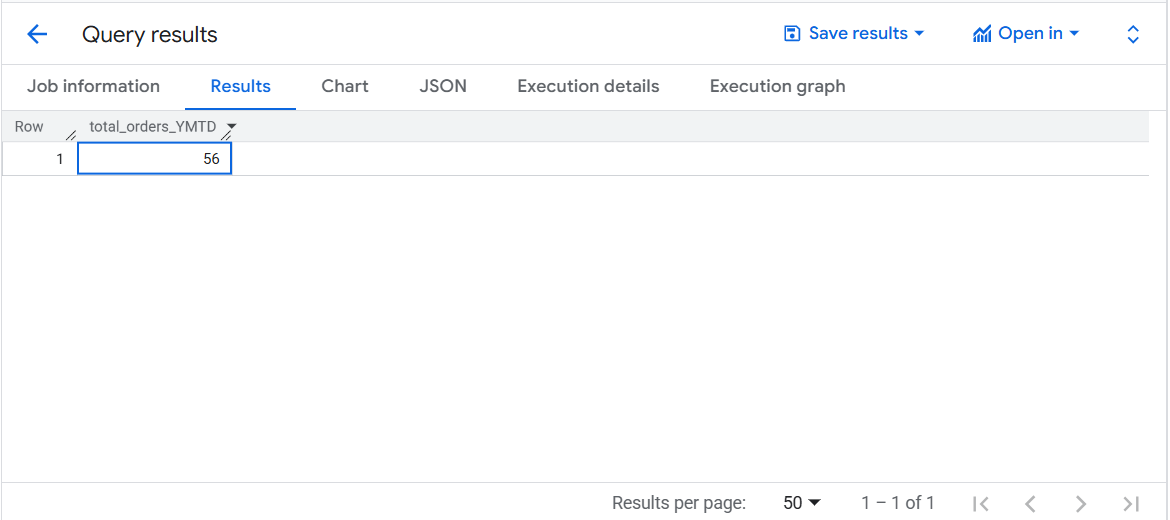
JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

  ON p.order\_id = o.order\_id

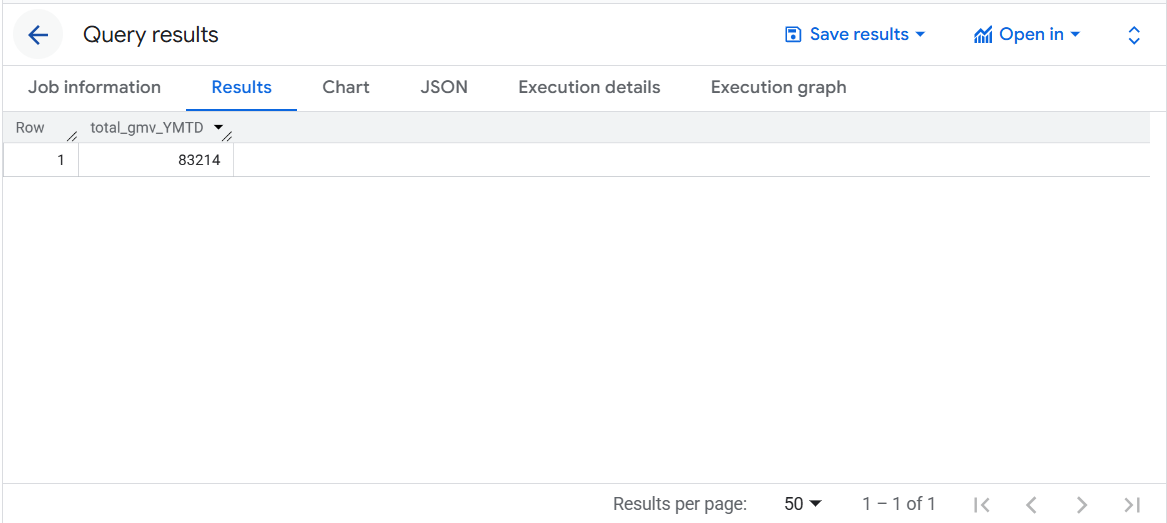
WHERE DATE(o.order\_date) BETWEEN '2025-05-01' AND CURRENT\_DATE();

**Output:**

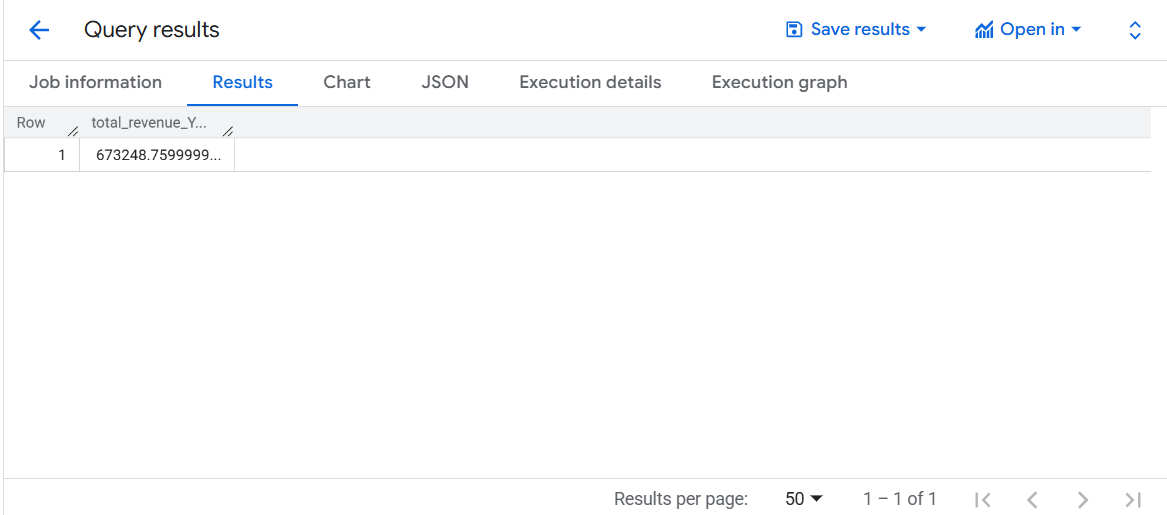
**-- Total Orders YMTD**



**-- Total GMV YMTD**

****

**-- Total Revenue YMTD**

****

**Looker studio:**



1. **Year-to-Month-To-Date customers (YMTD)**

-- YMTD New Customers

SELECT

  COUNT(DISTINCT customer\_id) AS new\_customers\_YMTD

FROM (

  SELECT customer\_id, MIN(order\_date) AS first\_order\_date

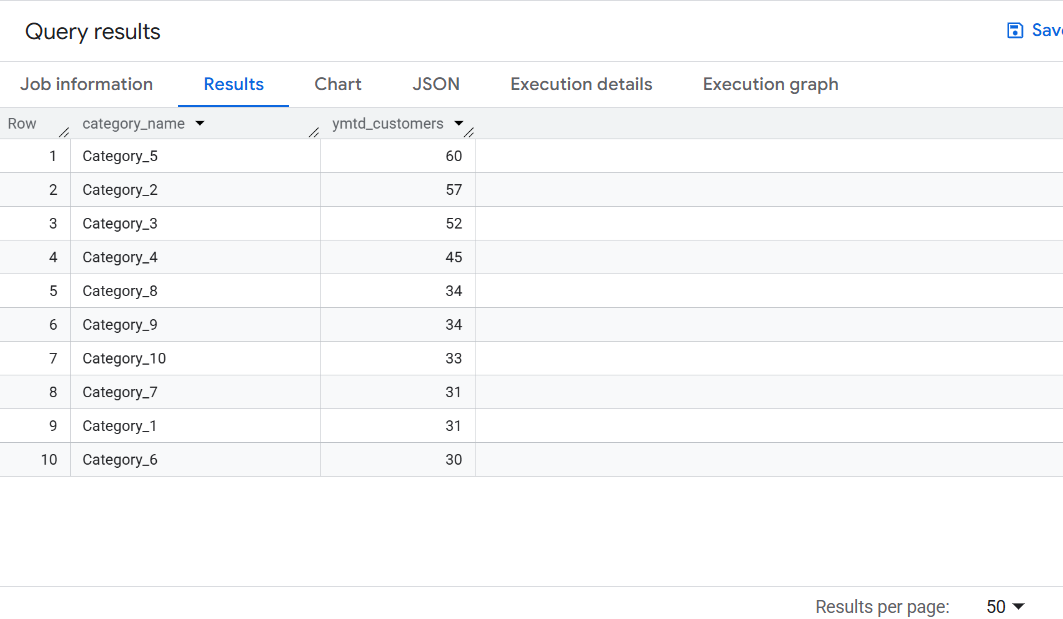
  FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`

  GROUP BY customer\_id

)

WHERE DATE(first\_order\_date) BETWEEN '2025-05-01' AND CURRENT\_DATE();

**Output: \_**



**Looker studio result:**



**YTD (Year-to-Date) New Customers**

SELECT

  COUNT(DISTINCT customer\_id) AS new\_customers\_YTD

FROM (

  SELECT customer\_id, MIN(order\_date) AS first\_order\_date

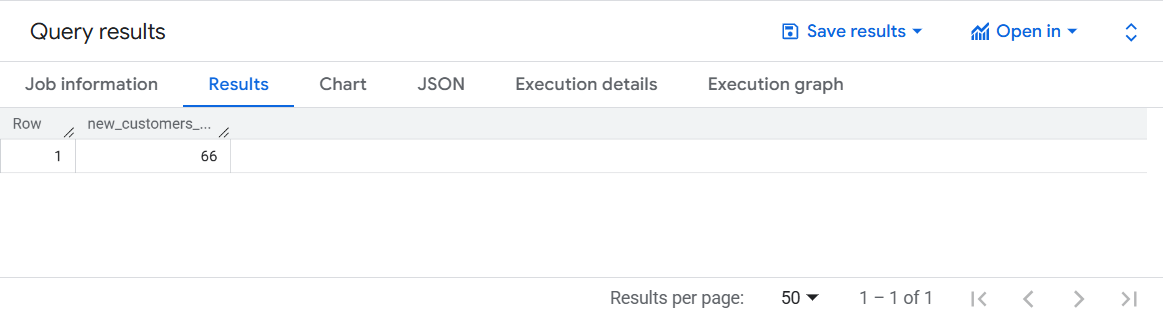
  FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`

  GROUP BY customer\_id

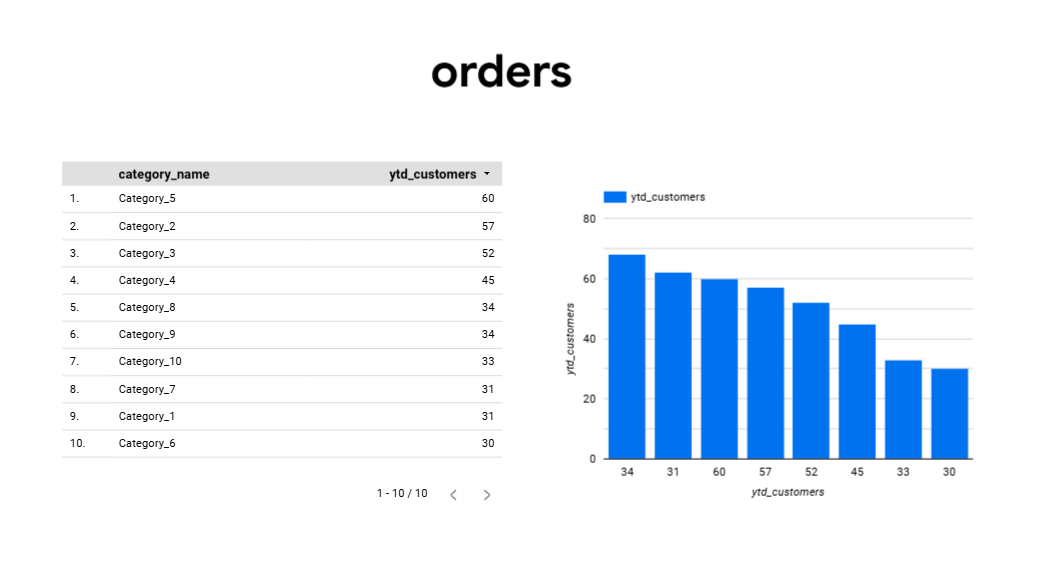
)

WHERE DATE(first\_order\_date) BETWEEN '2025-01-01' AND CURRENT\_DATE();

**Output:**

****

**Looker studio result:**



**10) YMTD Revenue Growth**

**-- YMTD Total Revenue**

SELECT

  SUM(p.amount) AS ymt\_total\_revenue

FROM `sales-analysis-sys.Sales\_Analysis\_system.Payments` p

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

  ON p.order\_id = o.order\_id

WHERE DATE(o.order\_date) BETWEEN '2025-03-01' AND CURRENT\_DATE();

**-- YMTD Total GMV**

SELECT

  SUM(oi.order\_item\_id \* oi.quantity) AS ymt\_total\_gmv

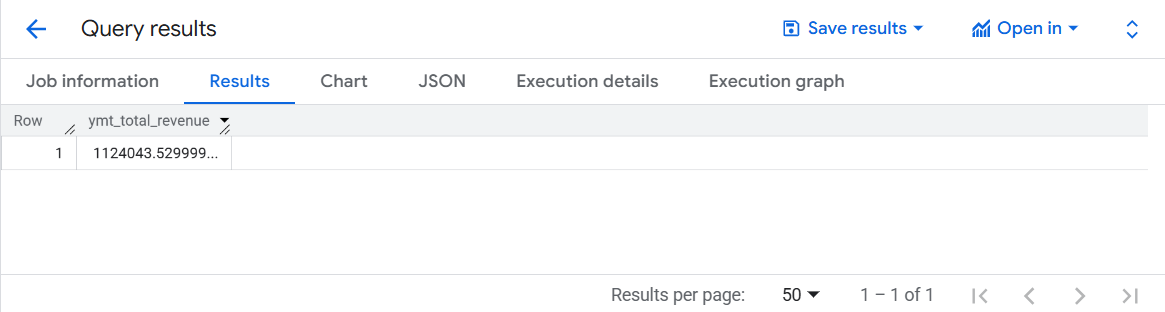
FROM `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

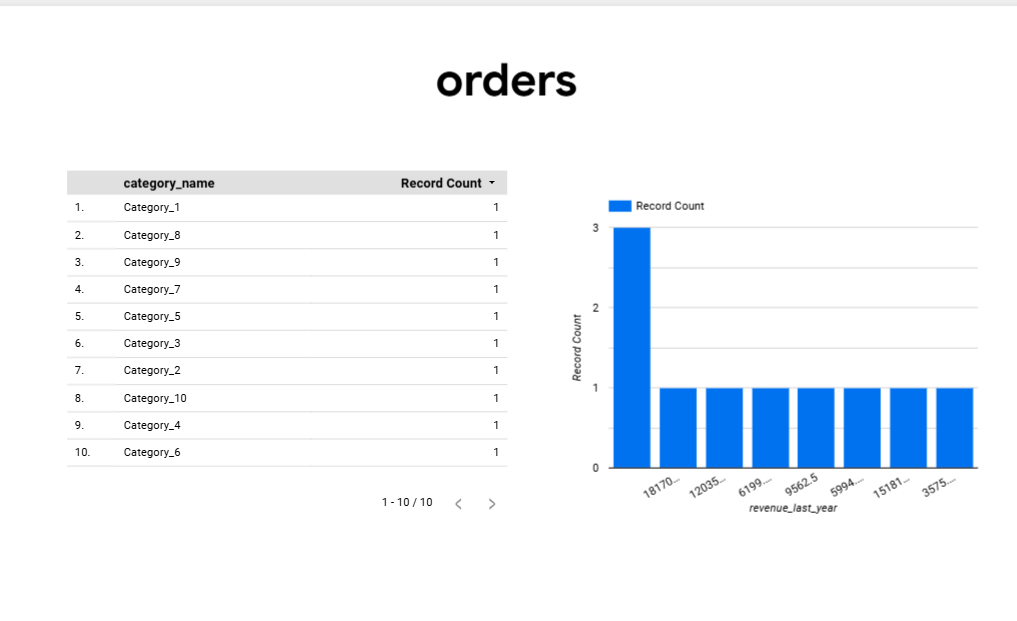
  ON oi.order\_id = o.order\_id

WHERE DATE(o.order\_date) BETWEEN '2025-03-01' AND CURRENT\_DATE();

**Output:**



**Looker:**



1. **customer growth by category**

SELECT

  c.string\_field\_0,

  FORMAT\_DATE('%Y-%m', DATE(o.order\_date)) AS order\_month,

  COUNT(DISTINCT o.customer\_id) AS new\_customers

FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

  ON o.order\_id = oi.order\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Products` p

  ON oi.product\_id = p.product\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Category` c

  ON p.category = c.string\_field\_1

WHERE o.customer\_id IN (

  SELECT customer\_id

  FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`

  GROUP BY customer\_id

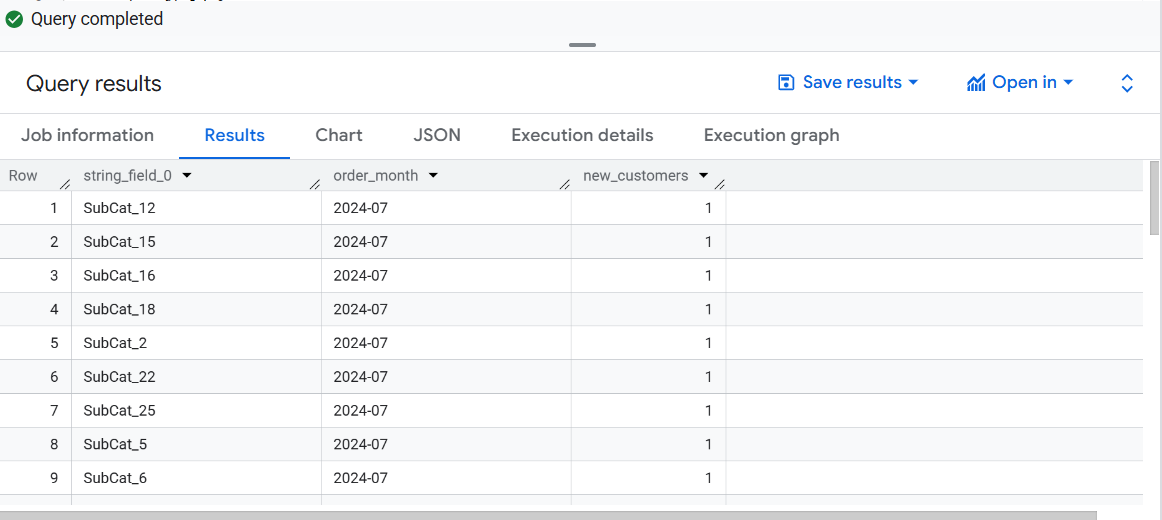
  HAVING COUNT(order\_id) = 1

)

GROUP BY c.string\_field\_0, order\_month

ORDER BY order\_month, c.string\_field\_0;

**Output:**



**Looker studio:**



**13) Year level view, year with category , year with product: -**

-- Year level view

SELECT

  EXTRACT(YEAR FROM o.order\_date) AS year,

  COUNT(DISTINCT o.order\_id) AS total\_orders,

  SUM(oi.quantity) AS total\_quantity,

  SUM(oi.order\_item\_id \* oi.quantity) AS total\_gmv

FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

  ON o.order\_id = oi.order\_id

GROUP BY year

ORDER BY year;

-- Year with category

SELECT

  EXTRACT(YEAR FROM o.order\_date) AS year,

  c.string\_field\_0,

  COUNT(DISTINCT o.order\_id) AS total\_orders,

  SUM(oi.quantity) AS total\_quantity,

  SUM(oi. order\_item\_id \* oi.quantity) AS total\_gmv

FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

  ON o.order\_id = oi.order\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Products` p

  ON oi.product\_id = p.product\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Category` c

  ON p.category = c.string\_field\_1

GROUP BY year, c.string\_field\_0

ORDER BY year, total\_gmv DESC;

-- Year with product

SELECT

  EXTRACT(YEAR FROM o.order\_date) AS year,

  p.product\_name,

  COUNT(DISTINCT o.order\_id) AS total\_orders,

  SUM(oi.quantity) AS total\_quantity,

  SUM(oi.order\_item\_id \* oi.quantity) AS total\_gmv

FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items` oi

  ON o.order\_id = oi.order\_id

JOIN `sales-analysis-sys.Sales\_Analysis\_system.Products` p

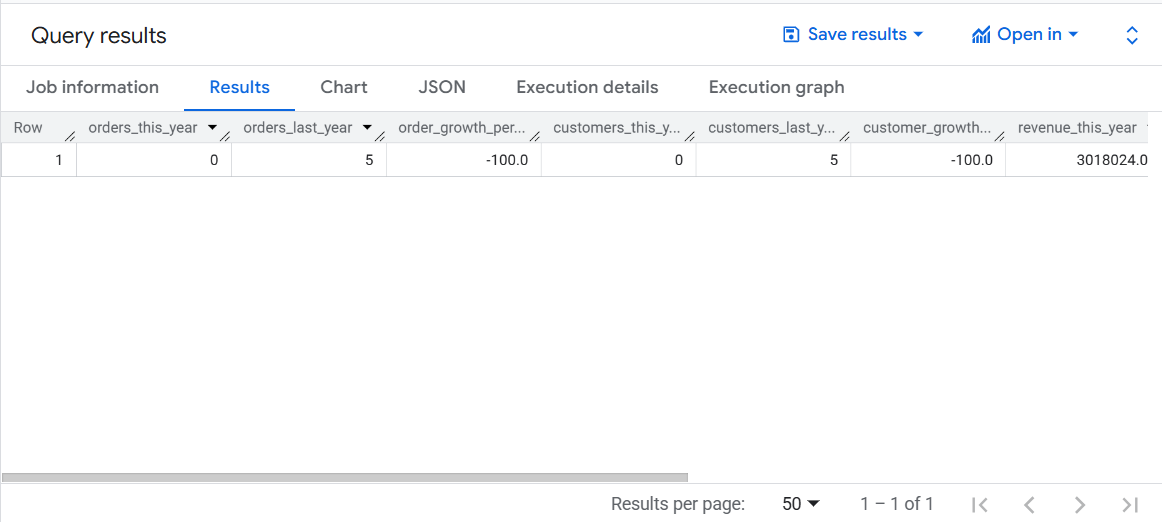
  ON oi.product\_id = p.product\_id

GROUP BY year, p.product\_name

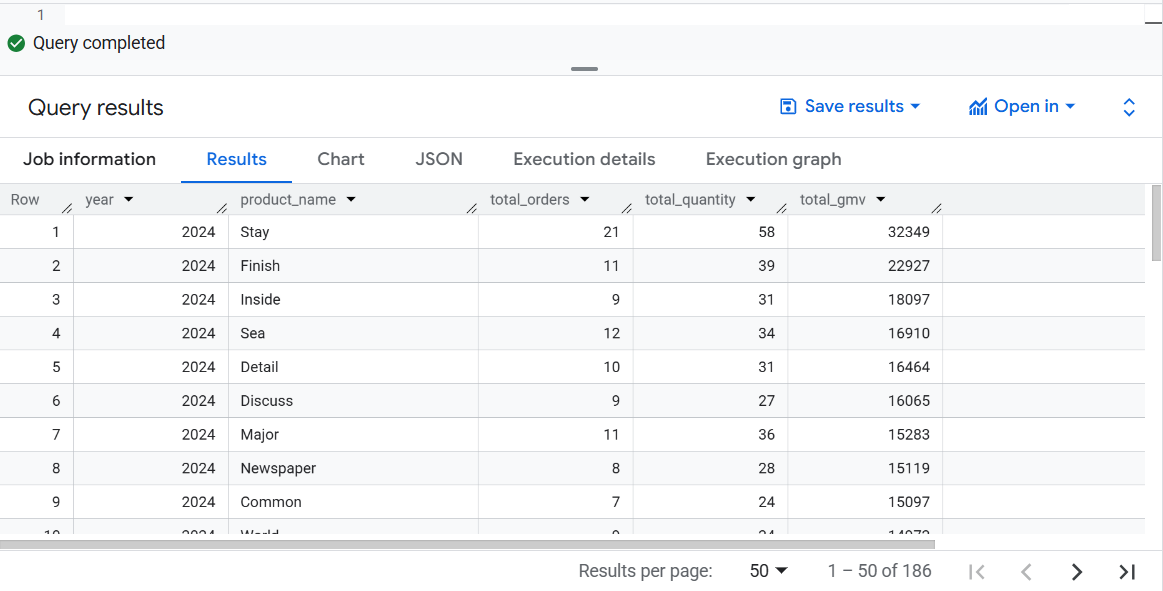
ORDER BY year, total\_gmv DESC;

**Output:**

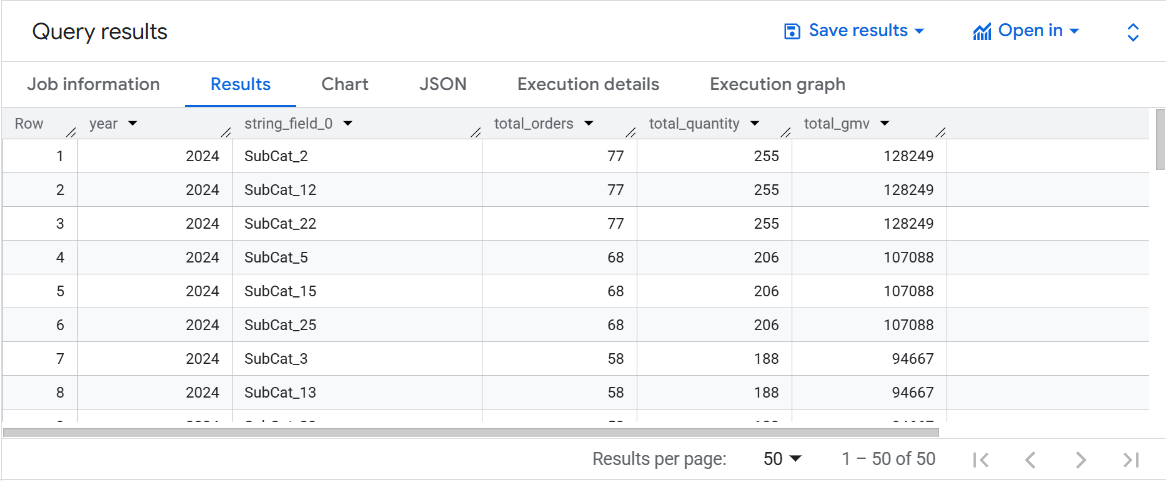
**YMTD WITH YEAR**



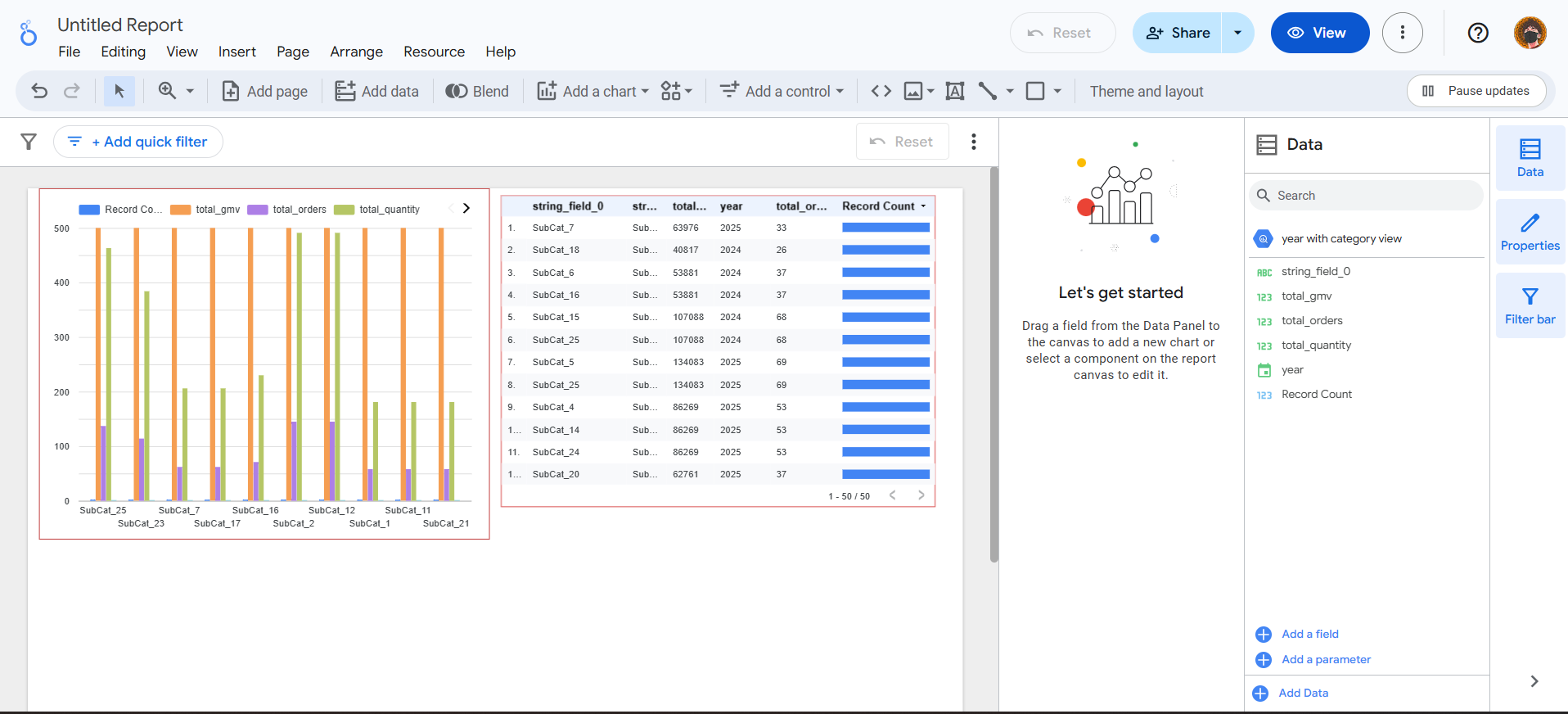
**YMTD WITH PRODUCT**

****

**YMTD WITH CATEGORY**

****

**Looker Studio: -**

****

**14) YMTD CUSTOMER & ORDER GROWTH**

--  This Year: July 1 to Today

SELECT

  this\_orders.total\_orders AS orders\_this\_year,

  last\_orders.total\_orders AS orders\_last\_year,

  ROUND(((this\_orders.total\_orders - last\_orders.total\_orders) / last\_orders.total\_orders) \* 100, 2) AS order\_growth\_percent,

  this\_orders.new\_customers AS customers\_this\_year,

  last\_orders.new\_customers AS customers\_last\_year,

  ROUND(((this\_orders.new\_customers - last\_orders.new\_customers) / last\_orders.new\_customers) \* 100, 2) AS customer\_growth\_percent,

  this\_revenue.total\_revenue AS revenue\_this\_year,

  last\_revenue.total\_revenue AS revenue\_last\_year,

  ROUND(((this\_revenue.total\_revenue - last\_revenue.total\_revenue) / last\_revenue.total\_revenue) \* 100, 2) AS revenue\_growth\_percent

FROM

  --  Total Orders & Customers This Year

  (

    SELECT

      COUNT(order\_id) AS total\_orders,

      COUNT(DISTINCT customer\_id) AS new\_customers

    FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`

    WHERE DATE(order\_date) BETWEEN '2025-07-01' AND CURRENT\_DATE()

  ) AS this\_orders,

  --  Total Orders & Customers Last Year

  (

    SELECT

      COUNT(order\_id) AS total\_orders,

      COUNT(DISTINCT customer\_id) AS new\_customers

    FROM `sales-analysis-sys.Sales\_Analysis\_system.Orders`

    WHERE DATE(order\_date) BETWEEN '2024-05-01' AND '2024-07-05'

  ) AS last\_orders,

  --  Revenue This Year

  (

    SELECT

      SUM(p.amount) AS total\_revenue

    FROM `sales-analysis-sys.Sales\_Analysis\_system.Payments` p

    JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

      ON p.order\_id = o.order\_id

    WHERE DATE(o.order\_date) BETWEEN '2024-05-01' AND CURRENT\_DATE()

  ) AS this\_revenue,

  --  Revenue Last Year

  (

    SELECT

      SUM(p.amount) AS total\_revenue

    FROM `sales-analysis-sys.Sales\_Analysis\_system.Payments` p

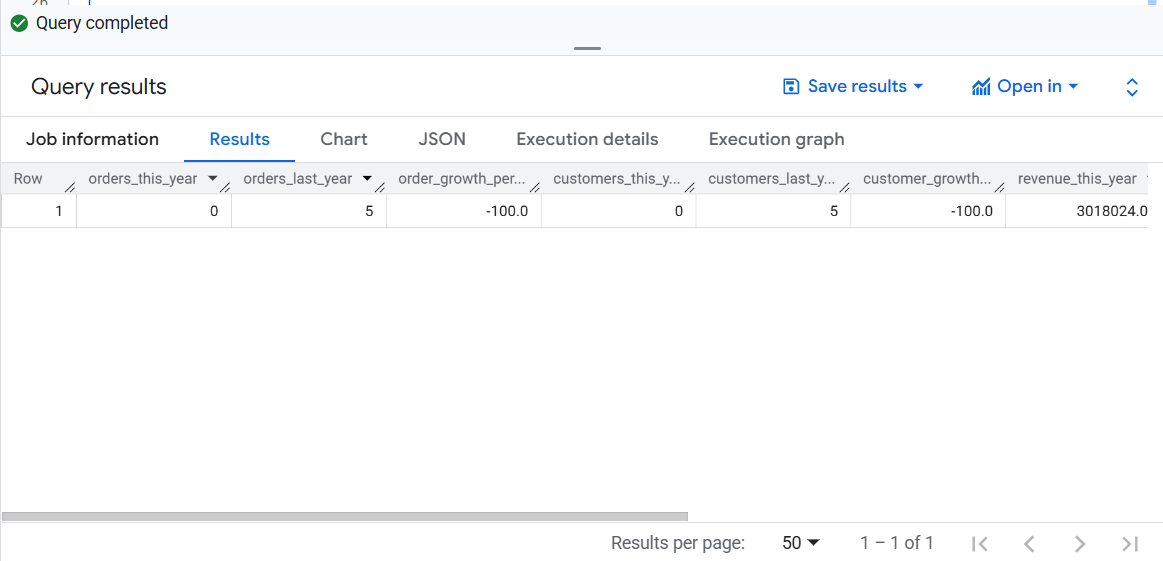
    JOIN `sales-analysis-sys.Sales\_Analysis\_system.Orders` o

      ON p.order\_id = o.order\_id

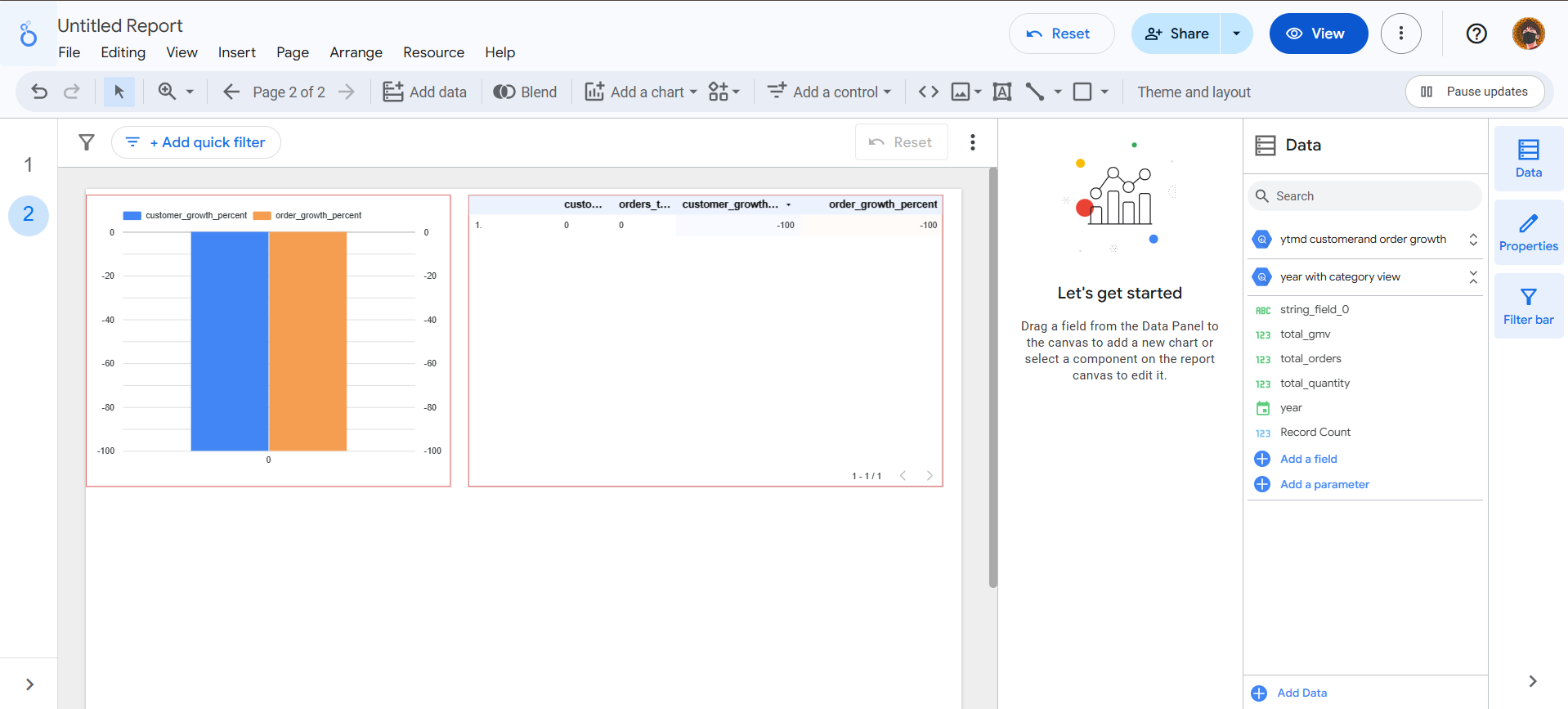
    WHERE DATE(o.order\_date) BETWEEN '2023-03-01' AND '2024-07-05'

  ) AS last\_revenue;

**OUTPUT: -**

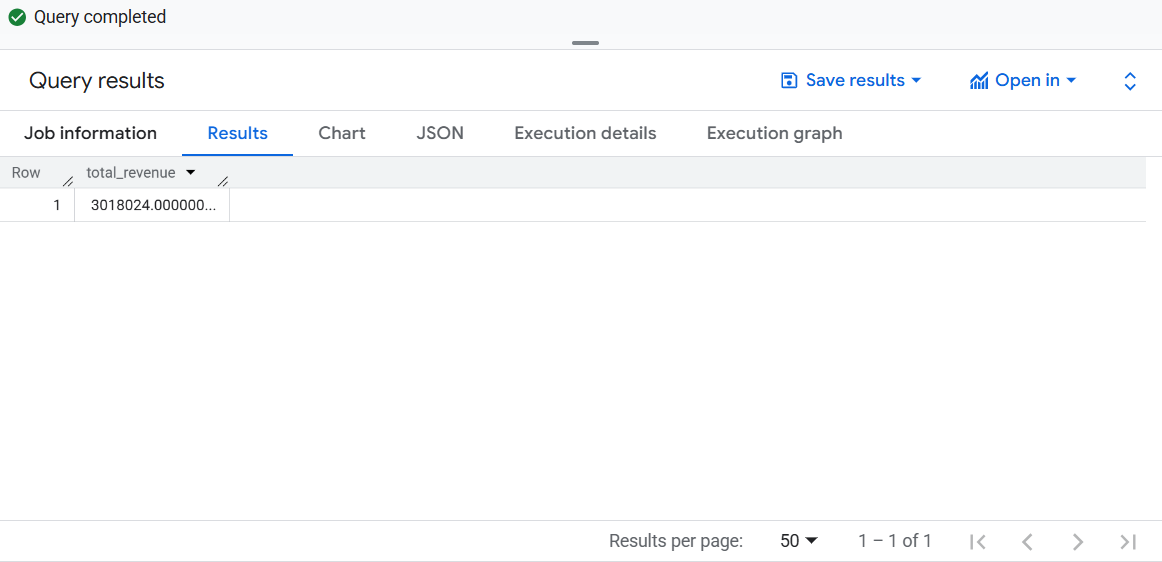
****

**LOOKER STUDIO: -**

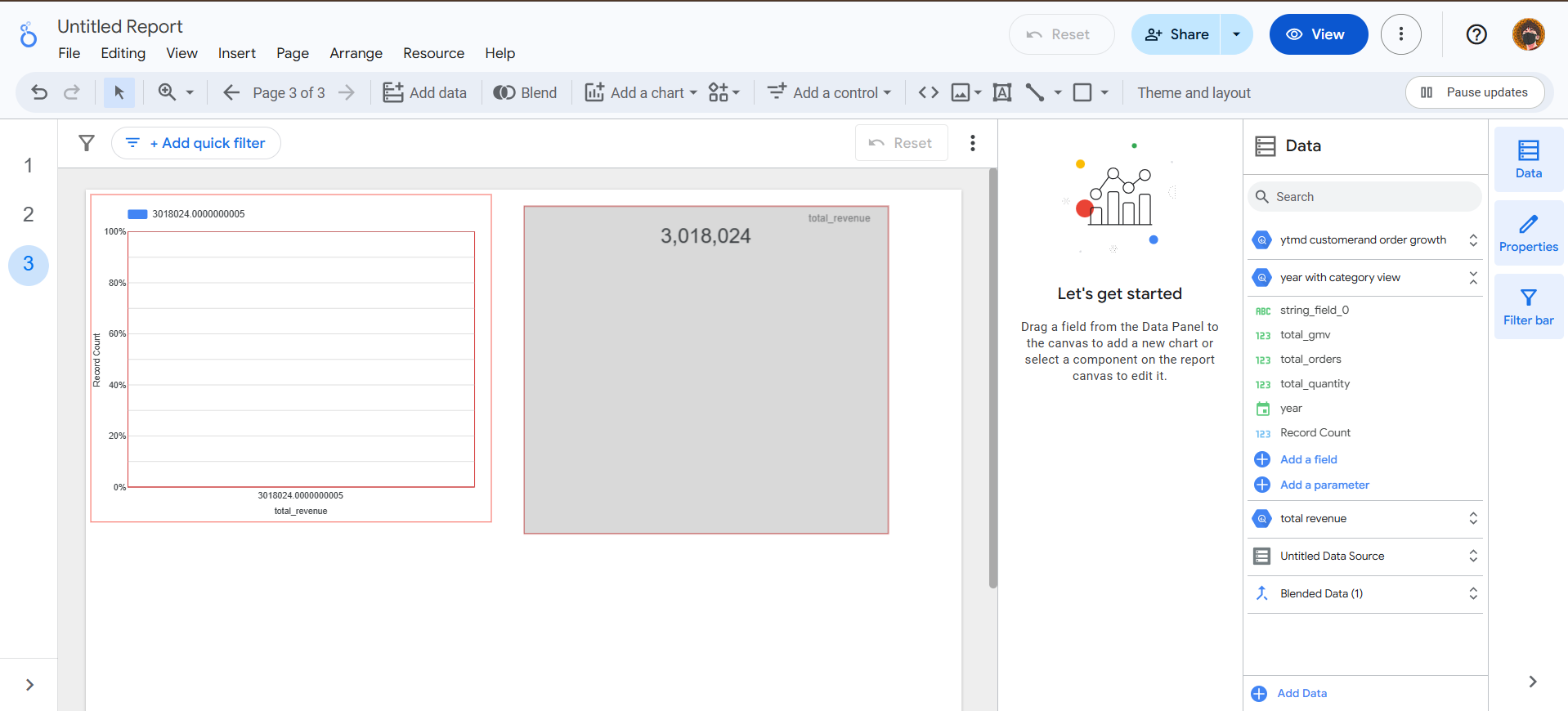
****

1. **Total Revenue: -**
2. SELECT
3. SUM(amount) AS total\_revenue
4. FROM `sales-analysis-sys.Sales\_Analysis\_system.Payments`;

**Output: -**

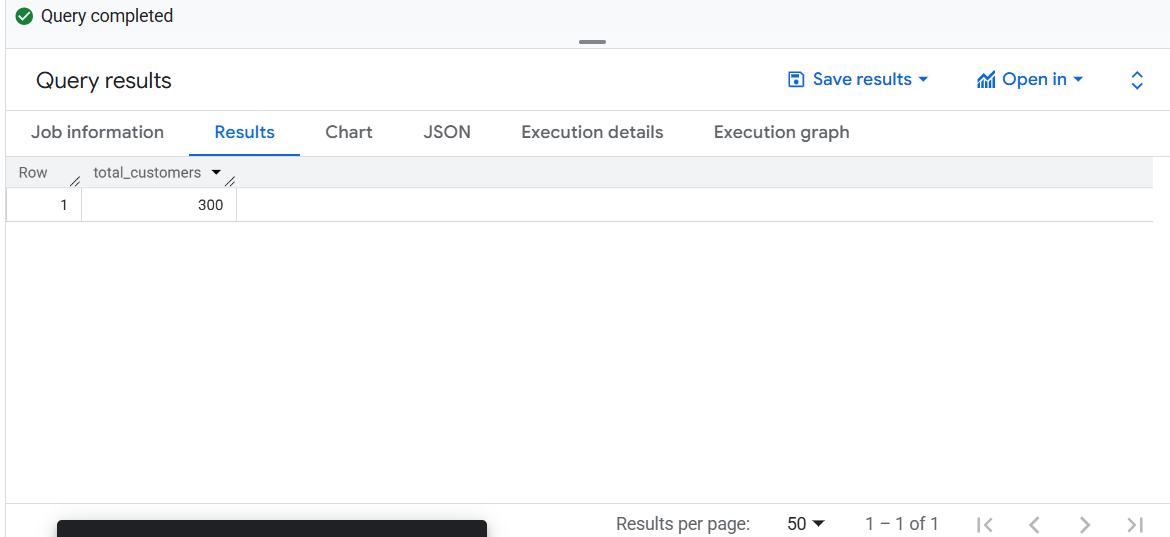
****

**Looker studio: -**

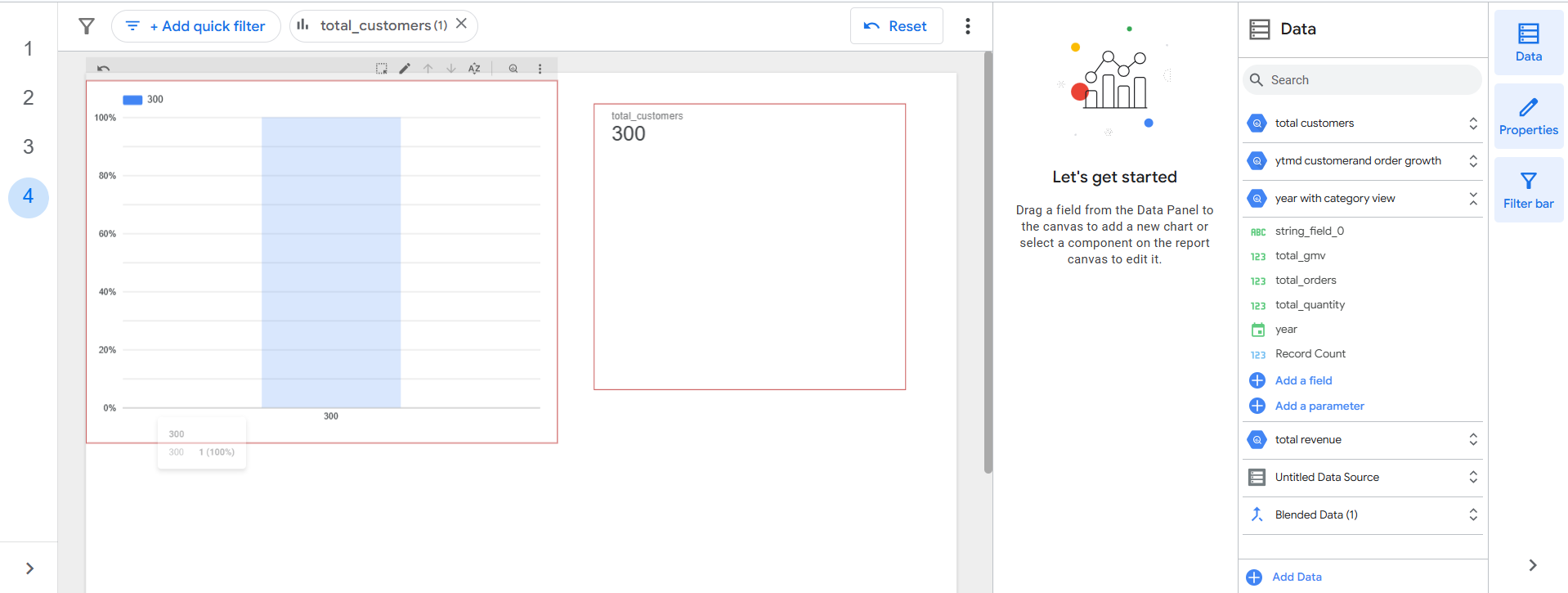
****

1. **Total Customers: -**
2. SELECT
3. COUNT(DISTINCT customer\_id) AS total\_customers
4. FROM `sales-analysis-sys.Sales\_Analysis\_system.Customers`;

**OUTPUT: -**

****

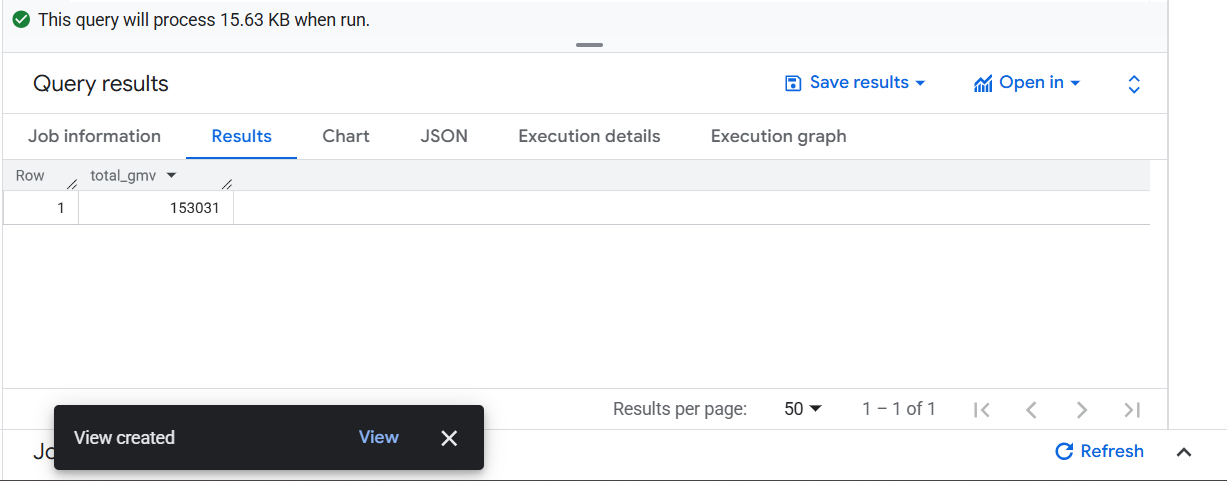
**LOOKER STUDIO: -**

****

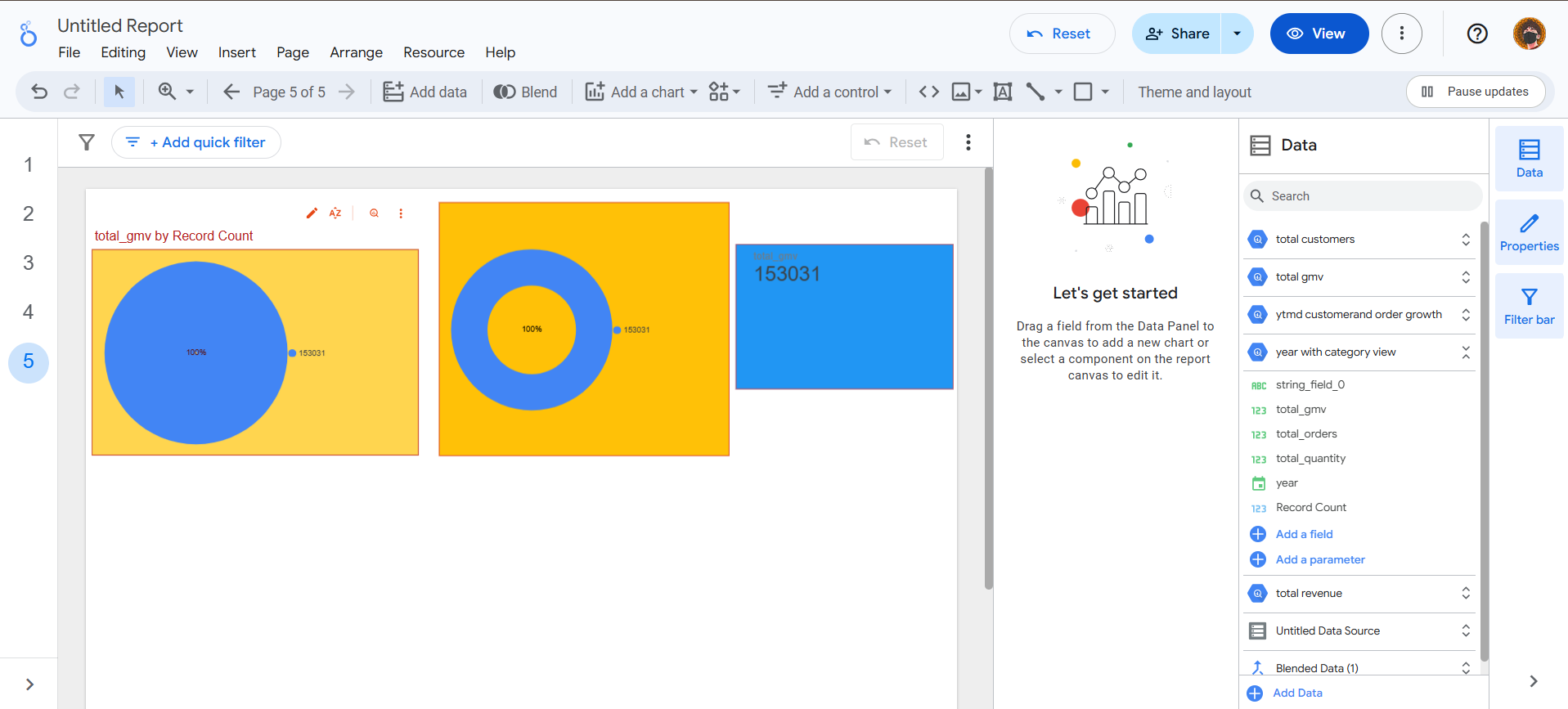
**17) Total GMV: -**

1. SELECT
2. SUM( product\_id \* quantity) AS total\_gmv
3. FROM `sales-analysis-sys.Sales\_Analysis\_system.Order\_Items`;

**OUTPUT: -**

****

**Looker Studio: -**

****

**Conclusion & Recommendations**

This report outlines a complete and practical workflow for integrating BigQuery data into Looker Studio, with a focus on transforming saved SQL queries into reusable, visual-ready data sources. The process began with the creation of a project in Google Cloud Platform (GCP), where analytical queries were written and saved using BigQuery’s Classic Saved Query feature. However, it was identified that Classic Saved Queries are not directly compatible with Looker Studio, as they do not create persistent, queryable objects within datasets.

To address this limitation, we converted the saved query into a permanent SQL view. This required ensuring that the query adhered strictly to BigQuery’s view constraints — specifically, that it must consist solely of a valid SELECT statement without procedural logic such as DECLARE, SET, or temporary table creation. Once validated, the query was saved as a view within the appropriate dataset.

This view then became accessible within Looker Studio through the “Add Data” > “BigQuery” interface. By selecting the correct GCP project and dataset, the view appeared as a selectable table-like object. Upon adding it to the report, Looker Studio automatically recognized the schema and allowed for immediate use of the fields in visualizations such as tables, bar charts, scorecards, and filters.

The successful integration of BigQuery views into Looker Studio offers several key advantages:

* 🔄 **Real-Time Data Access**: Views reflect the latest data from the underlying tables, ensuring that dashboards are always up to date.
* ♻️ **Query Reusability**: Once saved, views can be reused across multiple reports and dashboards without rewriting SQL.
* 📊 **Seamless Visualization**: Looker Studio provides an intuitive interface to build interactive dashboards using BigQuery data without requiring additional ETL steps.
* 🔐 **Security and Governance**: Access to views can be controlled via GCP IAM roles, ensuring secure and compliant data sharing.

In conclusion, this workflow empowers data analysts and business users to bridge the gap between raw data and actionable insights. By leveraging the power of BigQuery for scalable data processing and Looker Studio for flexible visualization, organizations can build robust, real-time reporting systems that support data-driven decision-making.

**Appendix**

**9.1 📄 SQL Query References**

Below are the key SQL queries used throughout the project. These queries were written in Google BigQuery and later transformed into views for integration with Looker Studio.

**a. Monthly Sales Summary View**

SELECT

customer\_id,

EXTRACT(MONTH FROM order\_date) AS order\_month,

SUM(order\_amount) AS total\_monthly\_sales

FROM

`project\_id.dataset.orders`

GROUP BY

customer\_id, order\_month

**b. Top Performing Products**

SELECT

product\_id,

product\_name,

COUNT(order\_id) AS total\_orders,

SUM(order\_amount) AS total\_revenue

FROM

`project\_id.dataset.orders`

GROUP BY

product\_id, product\_name

ORDER BY

total\_revenue DESC

LIMIT 10

**c. Return Rate by Category**

SELECT

category,

COUNTIF(is\_returned = TRUE) / COUNT(\*) AS return\_rate

FROM

`project\_id.dataset.orders`

GROUP BY

category

🔁 Note: All queries were validated to ensure they use only SELECT statements, making them compatible with BigQuery views.

**Data Assumptions and Notes**

To ensure clarity and consistency in analysis, the following assumptions and notes were applied:

* **Data Source**: All data was sourced from the orders, products, and returns tables within the BigQuery dataset.
* **Time Zone**: All timestamps were assumed to be in IST (Indian Standard Time) unless otherwise specified.
* **Currency**: All monetary values are assumed to be in INR (₹).
* **Returns Logic**: A return is identified by a boolean field is\_returned = TRUE in the orders table.
* **Data Completeness**: It is assumed that the dataset is complete and does not contain missing or null values in critical fields such as order\_id, order\_date, or order\_amount.
* **View Naming Convention**: Views were named using lowercase with underscores (e.g., monthly\_sales\_view) for consistency and readability.
* **Data Freshness**: Views reflect real-time data from the underlying tables and do not require manual refresh.

**REFRENCES**

**🔹 BigQuery View Creation & Best Practices**

1. [**Create Logical Views in BigQuery** – Google Cloud Documentation](https://cloud.google.com/bigquery/docs/views)  
   A comprehensive guide on how to create, manage, and secure logical views in BigQuery, including limitations and IAM permissions.
2. [**How to Create and Use Views in BigQuery** – OWOX Blog](https://www.owox.com/blog/articles/bigquery-create-views)  
   Explains the difference between standard and materialized views, with practical use cases and performance tips.
3. [**BigQuery View Creation: Syntax & Examples** – Hevo Data](https://hevodata.com/learn/bigquery-create-view/)  
   Step-by-step instructions for creating views using the BigQuery Console and bq command-line tool.

**🔹 Looker Studio Integration with BigQuery**

1. [**Analyze Data with Looker Studio** – Google Cloud](https://cloud.google.com/bigquery/docs/visualize-looker-studio)  
   Official documentation on connecting BigQuery datasets and views to Looker Studio for visualization.
2. [**How to Connect Google BigQuery with Looker Studio** – OWOX Blog](https://www.owox.com/blog/articles/connect-bigquery-to-looker-studio)  
   A user-friendly tutorial for integrating BigQuery with Looker Studio, including dashboard creation tips.
3. [**Connect to BigQuery | Looker Studio Community Connectors** – Google Developers](https://developers.google.com/looker-studio/connector/connect-to-bigquery)  
   Technical reference for using native and custom connectors to bring BigQuery data into Looker Studio.

**🔹 SQL Query Optimization for Dashboards**

1. [**SQL Query Optimization: 15 Techniques for Better Performance** – DataCamp](https://www.datacamp.com/blog/sql-query-optimization)  
   Covers indexing, avoiding SELECT \*, and other best practices for writing efficient SQL queries.
2. [**Best Practices for SQL Query Optimization** – GeeksforGeeks](https://www.geeksforgeeks.org/best-practices-for-sql-query-optimizations/)  
   A practical guide to improving query performance, especially useful for dashboard applications.
3. [**Optimizing SQL Queries for Dashboard Applications** – PeerDH Blog](https://peerdh.com/blogs/programming-insights/optimizing-sql-queries-for-performance-in-dashboard-applications-2)  
   Focuses on performance tuning specifically for real-time dashboards and BI tools.