**Sales Trend Analysis Report**

**1. Objective**

The purpose of this task was to analyze monthly revenue and order volume using SQL aggregations on an online sales dataset. The goal was to practice grouping data by time periods and calculating key metrics like total sales and order counts.

**2. Data Preparation and Generation**

To simulate a realistic sales dataset, I created a MySQL table named online\_sales with the following columns:

* order\_id (INT, primary key, auto-increment)
* order\_date (DATE)
* amount (DECIMAL(10,2))
* product\_id (INT)

Since real data was not available, I wrote a stored procedure populate\_online\_sales() that automatically inserts 50 rows of random sales data into the table. The procedure works as follows:

* For each row, generates a random order\_date between January 1, 2023, and December 31, 2023.
* Generates a random sales amount between 10.00 and 510.00.
* Assigns a random product\_id between 1 and 10.

This procedure ensures a sufficiently varied dataset for aggregation analysis.

**3. SQL Query for Monthly Sales Trend**

To analyze sales trends, I wrote the following SQL query:

SELECT

YEAR(order\_date) AS order\_year,

MONTH(order\_date) AS order\_month,

SUM(amount) AS total\_revenue,

COUNT(DISTINCT order\_id) AS total\_orders

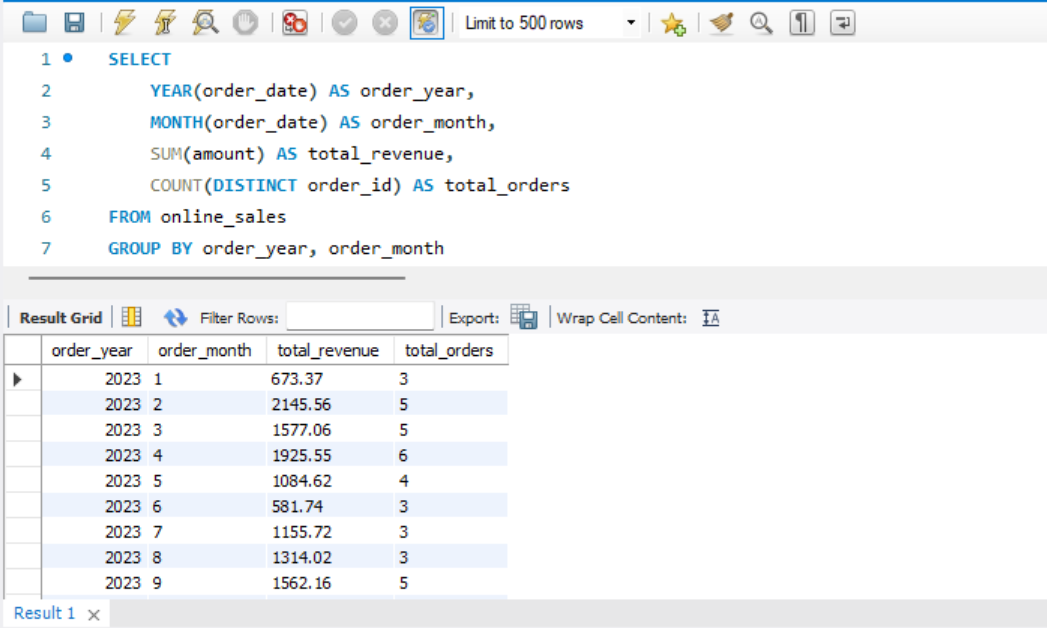
FROM online\_sales

GROUP BY order\_year, order\_month

ORDER BY order\_year, order\_month;

**Explanation:**

* YEAR(order\_date) and MONTH(order\_date) extract year and month from order dates.
* SUM(amount) calculates the total sales revenue for each month.
* COUNT(DISTINCT order\_id) counts unique orders per month.
* GROUP BY aggregates data by each year-month group.
* ORDER BY sorts the results chronologically.



**Query 2 Explanation:**

1. **SELECT clause**
   * YEAR(order\_date) AS order\_year: Extracts the year part from the order\_date field and labels it as order\_year. This allows grouping sales by year.
   * MONTH(order\_date) AS order\_month: Extracts the month part from the order\_date field and labels it as order\_month. This groups sales by month within each year.
   * SUM(amount) AS total\_revenue: Calculates the total revenue generated in each month by summing up the amount values for all orders in that month.
   * COUNT(DISTINCT order\_id) AS total\_orders: Counts the number of unique orders placed in each month, ensuring duplicate order IDs are counted only once.
2. **FROM clause**
   * Specifies the source table online\_sales from which the data is retrieved.
3. **GROUP BY clause**
   * Groups the results based on order\_year and order\_month. This means the aggregation functions (SUM and COUNT) are calculated for each unique combination of year and month.
4. **ORDER BY clause**
   * Sorts the final results in ascending chronological order by year and then by month, making it easier to observe sales trends over time.

**Why this query works well for trend analysis:**

* By extracting year and month, it allows grouping data in meaningful time intervals.
* Aggregation functions summarize large datasets into concise metrics like total revenue and order volume per month.
* Ordering ensures that the results are presented in a time-sequential way, which is essential for spotting trends and patterns.

A screenshot of a computer

AI-generated content may be incorrect.