# Sales Trend Analysis Report

Source: TASK 6 DA - Sales Trend Analysis using aggregations. (SQL script provided for PostgreSQL)

This document contains a recommended SQL script and a sample result table (simulated data).

## 1. PostgreSQL SQL Script

-- Sales Trend Analysis (PostgreSQL)  
-- Dataset: online\_sales.orders(order\_id, order\_date, amount, product\_id)  
-- Goal: Monthly revenue and order volume aggregations (year + month)  
-- Uses EXTRACT, GROUP BY year/month, SUM, COUNT(DISTINCT order\_id)  
  
/\* 1) Monthly revenue & distinct order volume (year-month) \*/  
SELECT  
 EXTRACT(YEAR FROM order\_date)::INT AS year,  
 EXTRACT(MONTH FROM order\_date)::INT AS month,  
 TO\_CHAR(order\_date, 'YYYY-MM') AS year\_month,  
 SUM(amount)::NUMERIC(12,2) AS monthly\_revenue,  
 COUNT(DISTINCT order\_id) AS unique\_order\_count,  
 COUNT(\*) AS row\_count,  
 (SUM(amount) / NULLIF(COUNT(DISTINCT order\_id),0))::NUMERIC(12,2) AS avg\_order\_value  
FROM online\_sales.orders  
WHERE order\_date BETWEEN '2023-01-01' AND '2024-12-31' -- adjust range as needed  
GROUP BY year, month, year\_month  
ORDER BY year, month;  
  
-- 2) Top 3 months by revenue (over the same period)  
SELECT year, month, year\_month, monthly\_revenue, unique\_order\_count  
FROM (  
 SELECT  
 EXTRACT(YEAR FROM order\_date)::INT AS year,  
 EXTRACT(MONTH FROM order\_date)::INT AS month,  
 TO\_CHAR(order\_date, 'YYYY-MM') AS year\_month,  
 SUM(amount)::NUMERIC(12,2) AS monthly\_revenue,  
 COUNT(DISTINCT order\_id) AS unique\_order\_count,  
 ROW\_NUMBER() OVER (ORDER BY SUM(amount) DESC) AS rn  
 FROM online\_sales.orders  
 WHERE order\_date BETWEEN '2023-01-01' AND '2024-12-31'  
 GROUP BY year, month, year\_month  
) t  
WHERE rn <= 3  
ORDER BY monthly\_revenue DESC;  
  
-- 3) Monthly growth vs previous month (% change)  
WITH monthly AS (  
 SELECT  
 DATE\_TRUNC('month', order\_date)::DATE AS month\_start,  
 SUM(amount) AS monthly\_revenue  
 FROM online\_sales.orders  
 GROUP BY month\_start  
)  
SELECT  
 m.month\_start,  
 m.monthly\_revenue,  
 LAG(m.monthly\_revenue) OVER (ORDER BY m.month\_start) AS prev\_revenue,  
 CASE WHEN LAG(m.monthly\_revenue) OVER (ORDER BY m.month\_start) IS NULL THEN NULL  
 WHEN LAG(m.monthly\_revenue) OVER (ORDER BY m.month\_start) = 0 THEN NULL  
 ELSE ROUND((m.monthly\_revenue - LAG(m.monthly\_revenue) OVER (ORDER BY m.month\_start)) /  
 LAG(m.monthly\_revenue) OVER (ORDER BY m.month\_start) \* 100, 2)  
 END AS pct\_change\_vs\_prev\_month  
FROM monthly m  
ORDER BY m.month\_start;  
  
-- Notes:  
-- 1) Use COALESCE()/NULLIF to handle NULLs and divide-by-zero safely.  
-- 2) Replace date filters with the time window you need (hint: use LIMIT or WHERE).  
-- 3) For big tables, ensure you have an index on order\_date for faster grouping/filters.

## 2. Sample Result Table (Monthly Aggregation)

|  |  |  |  |
| --- | --- | --- | --- |
| Year-Month | Monthly Revenue | Unique Orders | Avg Order Value |
| 2023-01 | 29830.59 | 121 | 246.53 |
| 2023-02 | 30475.56 | 118 | 258.27 |
| 2023-03 | 25767.75 | 97 | 265.65 |
| 2023-04 | 26857.71 | 111 | 241.96 |
| 2023-05 | 31462.28 | 121 | 260.02 |
| 2023-06 | 22894.85 | 90 | 254.39 |
| 2023-07 | 32437.02 | 136 | 238.51 |
| 2023-08 | 32974.77 | 126 | 261.70 |
| 2023-09 | 34026.83 | 131 | 259.75 |
| 2023-10 | 31160.64 | 128 | 243.44 |
| 2023-11 | 30193.36 | 120 | 251.61 |
| 2023-12 | 30598.72 | 134 | 228.35 |

## 3. Top 3 Months by Revenue (Sample)

|  |  |  |
| --- | --- | --- |
| Year-Month | Monthly Revenue | Unique Orders |
| 2023-09 | 34026.83 | 131 |
| 2024-11 | 33710.95 | 126 |
| 2023-08 | 32974.77 | 126 |

## 4. Notes & How to run the SQL

1) Run the SQL script in a PostgreSQL client (psql, pgAdmin, DBeaver, etc.) connected to your database.  
2) Update the date range in the WHERE clause to match the period you want to analyze.  
3) For large datasets, add an index on order\_date (e.g., CREATE INDEX idx\_orders\_date ON online\_sales.orders(order\_date));  
4) Use COALESCE or NULLIF where necessary to avoid division-by-zero.  
5) If your order\_id is unique per row, COUNT(DISTINCT order\_id) equals COUNT(\*); otherwise use DISTINCT to count unique orders.