SQL Highlights — Customer Behaviour Analysis

PostgreSQL (pgAdmin) · 3,900 retail transactions

> *A curated excerpt of the analysis SQL — CTEs, window functions, conditional aggregation, subqueries and type casting. Public retail dataset; no client data.*

```sql
-- ============================================================================
--  SQL HIGHLIGHTS — Customer Shopping Behaviour Analysis (PostgreSQL / pgAdmin)
--  A curated excerpt of the analysis. Public retail dataset, 3,900 transactions.
--  Question numbers match the full analysis (Q1–Q10).
-- ============================================================================


-- ----------------------------------------------------------------------------
-- Q2. Which customers used a discount but STILL spent more than average?
--     A subquery supplies the benchmark, so the threshold is data-driven
--     rather than hard-coded.
-- ----------------------------------------------------------------------------
SELECT customer_id, purchase_amount
FROM public.customer
WHERE discount_applied = 'Yes'
  AND purchase_amount >= (SELECT AVG(purchase_amount) FROM public.customer);


-- ----------------------------------------------------------------------------
-- Q3. What are the top 5 products by average review rating?
--     Note the cast: Postgres stored review_rating as DOUBLE PRECISION, and
--     ROUND() does not behave predictably on floating point — so the column is
--     cast to NUMERIC first to get clean, readable 2-dp output.
-- ----------------------------------------------------------------------------
SELECT
    item_purchased,
    ROUND(AVG(review_rating::numeric), 2) AS "Average Product Review Rating"
FROM public.customer
GROUP BY item_purchased
ORDER BY AVG(review_rating) DESC
LIMIT 5;


-- ----------------------------------------------------------------------------
-- Q5. Do subscribed customers spend more?
--     Average spend and total revenue compared across both groups.
-- ----------------------------------------------------------------------------
SELECT
    subscription_status,
    COUNT(customer_id)              AS total_customers,
```

```sql
    ROUND(AVG(purchase_amount), 2)  AS avg_spend,
    ROUND(SUM(purchase_amount), 2)  AS total_revenue
FROM public.customer
GROUP BY subscription_status
ORDER BY total_revenue DESC;


-- ------------------------------------------------------------------------
-- Q6. Which 5 products have the highest percentage of discounted purchases?
--     Conditional aggregation computes the percentage in a single pass —
--     no self-join, no second scan of the table.
-- ------------------------------------------------------------------------
SELECT
    item_purchased,
    ROUND(100 * SUM(CASE WHEN discount_applied = 'Yes' THEN 1 ELSE 0 END)
              / COUNT(*), 2) AS discount_rate
FROM public.customer
GROUP BY item_purchased
ORDER BY discount_rate DESC
LIMIT 5;


-- ------------------------------------------------------------------------
-- Q7. Segment customers into New / Returning / Loyal, and count each segment.
--     A CTE stages the classification so the final aggregation stays readable.
--     Rules:  1 purchase = New · 2–10 = Returning · >10 = Loyal
-- ------------------------------------------------------------------------
WITH customer_type AS (
    SELECT
        customer_id,
        previous_purchases,
        CASE
            WHEN previous_purchases = 1                 THEN 'New'
            WHEN previous_purchases BETWEEN 2 AND 10    THEN 'Returning'
            ELSE                                             'Loyal'
        END AS customer_segment
    FROM public.customer
)
SELECT
    customer_segment,
    COUNT(*) AS "Number of Customers per segment"
FROM customer_type
GROUP BY customer_segment
ORDER BY "Number of Customers per segment" DESC;


-- ------------------------------------------------------------------------
-- Q8. What are the top 3 most-purchased products WITHIN each category?
--     A window function ranks items inside each category partition.
--
--     Why ROW_NUMBER() and not RANK() / DENSE_RANK()?
```

```sql
--     Several items share the same order count. RANK() and DENSE_RANK() would
--     award tied items the same rank, which can return FEWER than three
--     distinct products per category. ROW_NUMBER() always assigns a unique
--     position, guaranteeing exactly a top 3.
-- ------------------------------------------------------------------------
WITH item_counts AS (
    SELECT
        category,
        item_purchased,
        COUNT(customer_id) AS total_orders,
        ROW_NUMBER() OVER (
            PARTITION BY category                -- restart ranking per category
            ORDER BY COUNT(customer_id) DESC     -- most-ordered item ranks first
        ) AS item_rank
    FROM public.customer
    GROUP BY category, item_purchased
)
SELECT
    item_rank,
    category,
    item_purchased,
    total_orders
FROM item_counts
WHERE item_rank <= 3;



-- ------------------------------------------------------------------------
-- Q10. What is the revenue contribution of each age group?
-- ------------------------------------------------------------------------
SELECT
    age_group,
    SUM(purchase_amount) AS total_revenue
FROM public.customer
GROUP BY age_group
ORDER BY total_revenue DESC;
```