Fictional Sales Data: SQL EDA (EN)

Introduction

In this project we'll perform an exploratory data analysis of a fictional sales dataset.

The dataset contains 3 tables encompassing 2 years (Jan 2023 - May 2024):

1. **Customers** containing customer info (ID, age, address, validity)
2. **Products** (ID, name, price, price validity period)
3. **Orders** (ID, product, customer, quantity, date).

Data Source

The original dataset is available on Kaggle at this link.

NB: the dates in the original dataset have been modified/updated using Excel randomization formulae.

Data Description

customers

customer_id - customer ID

customer_address - customer address

customer_age - customer age

effective_start_date - date when the customer enrolled with the program

effective_end_date - date when the customer profile should be deactivated if the customer is inactive

products

product_id - product ID

product_name - product name

product_price - product price

effective_start_date - price validity start date

effective_end_date - price validity end date

orders

order_id - order ID

product_id - product ID

customer_id - customer ID

product_quantity - product quantity

order_date - order date

Working with the Data

1. Creating a temporary table based off a join between *orders* and *products* that will store the total sum of purchase (*purchase_sum*). Note that it is important to get the valid price right because the products table stores all the prices along with their validity period.

```
CREATE TEMP TABLE orders_temp AS
SELECT o.*,
    p.product_price,
    o.product_quantity * p.product_price AS purchase_sum
FROM orders AS o
LEFT JOIN products AS p
ON o.product_id = p.product_id
    AND o.order_date BETWEEN p.effective_start_date AND p.effective_end_date;
```

	order_id integer	product_id integer	customer_id integer	product_quantity integer	order_date date	product_price double precision	purchase_sum double precision
1	141234	582	174831	1	2023-07-18	700	700
2	141235	981	105312	1	2023-08-22	12.99	12.99
3	141236	277	209194	2	2023-02-25	15	30
4	141237	443	143165	1	2023-05-27	139	139
5	141238	277	120517	1	2023-03-24	12.99	12.99

2. Let's see when we hit 100,000 in sales

```
WITH t1 AS
(SELECT order_id,
    order_date,
    ROUND(SUM(purchase_sum) OVER (ORDER BY order_date)::NUMERIC, 2) AS total_sal
es
FROM orders_temp
ORDER BY order_date)
SELECT order_id,
    order_date,
    total_sales
FROM t1
WHERE total_sales >= 100000
ORDER BY total_sales
LIMIT 1;
```

	order_id integer	order_date date	total_sales numeric
1	238456	2023-01-17	103423.21

3. Calculating the moving average total of sales for the last 2 available months of our dataset

```
SELECT *
FROM
```

```
(SELECT order_date,
    ROUND(SUM(SUM(purchase_sum)) OVER (ORDER BY order_date ROWS BETWEEN 6 PRECED
ING AND CURRENT ROW)::NUMERIC, 2) AS total_sales,
    ROUND(AVG(SUM(purchase_sum)) OVER (ORDER BY order_date ROWS BETWEEN 6 PRECED
ING AND CURRENT ROW)::NUMERIC, 2) AS avg_sales
FROM orders_temp
GROUP BY order_date)
WHERE order_date >= ('2024-04-01')::DATE+INTERVAL '6 DAYS';
```

	order_date date	total_sales numeric	avg_sales numeric
1	2024-04-07	263201.03	37600.15
2	2024-04-08	273104.54	39014.93
3	2024-04-09	271918.00	38845.43
4	2024-04-10	264966.81	37852.40
5	2024-04-11	275977.28	39425.33
6	2024-04-12	275709.20	39387.03
7	2024-04-13	280975.77	40139.40
8	2024-04-14	274368.05	39195.44

4. Creating a function to identify the Nth most active customer (by number of orders)

```
CREATE OR REPLACE FUNCTION NthMostActiveCustomercnt (N INT)
    RETURNS TABLE (customer_id INT, order_cnt INT) AS $$
BEGIN
    RETURN QUERY(
        WITH number_of_orders AS(
        SELECT orders_temp.customer_id,
            COUNT(orders_temp.order_id) AS order_cnt
        FROM orders_temp
        GROUP BY orders_temp.customer_id
        ORDER BY order_cnt),
        ranked AS (
        SELECT number_of_orders.customer_id,
            number_of_orders.order_cnt,
            DENSE_RANK() OVER (ORDER BY number_of_orders.order_cnt DESC) AS rank
ing
        FROM number_of_orders)
        SELECT ranked.customer_id,
            ranked.order_cnt::INTEGER
        FROM ranked
        WHERE ranked.ranking = N);

    END;
    $$ LANGUAGE plpgsql
```

Let's see which customer made the most purchases:

```
SELECT * FROM NthMostActiveCustomercnt(1);
```

	customer_id integer	order_cnt integer
1	136485	9

5. Creating a function to identify the Nth most active customer (by the total purchase sum)

```
CREATE OR REPLACE FUNCTION NthMostActiveCustomerSum (N INT)
    RETURNS TABLE (customer_id INT, order_sum INT) AS $$
BEGIN
    RETURN QUERY(
        WITH number_of_orders AS(
        SELECT orders_temp.customer_id,
            SUM(orders_temp.purchase_sum) AS order_sum
        FROM orders_temp
        GROUP BY orders_temp.customer_id
        ORDER BY order_sum),
        ranked AS (
        SELECT number_of_orders.customer_id,
            number_of_orders.order_sum,
            DENSE_RANK() OVER (ORDER BY number_of_orders.order_sum DESC) AS rank
ing
        FROM number_of_orders)
        SELECT ranked.customer_id,
            ranked.order_sum::INTEGER
        FROM ranked
        WHERE ranked.ranking = N);

    END;
    $$ LANGUAGE plpgsql
```

Let's see which customer made the most purchases by total purchase sum:

	customer_id integer	order_sum integer
1	225849	4650

6. Creating a function to identify the Nth most purchased product

```
CREATE OR REPLACE FUNCTION NthMostPurchasedProduct (N INT)
    RETURNS TABLE (product_id INT, product_name VARCHAR(512)) AS $$

BEGIN
    RETURN QUERY(
        WITH total_quantity AS(
        SELECT o.product_id,
            p.product_name,
            SUM(o.product_quantity) AS prd_qty
        FROM orders_temp AS o
        LEFT JOIN (SELECT DISTINCT(products.product_name), products.product_id F
ROM products) AS p
```

```
            USING (product_id)
        GROUP BY o.product_id, p.product_name
        ORDER BY prd_qty DESC),
        ranked AS (
        SELECT total_quantity.product_id,
            total_quantity.product_name,
            DENSE_RANK() OVER (ORDER BY total_quantity.prd_qty DESC) AS ranking
        FROM total_quantity)
        SELECT ranked.product_id,
            ranked.product_name
        FROM ranked
        WHERE ranked.ranking = N);
    END;
    $$ LANGUAGE plpgsql
```

The most purchased product is a battery pack:

	product_id integer	product_name character varying	prd_qty integer
1	692	AAA Batteries (4-pack)	29665

7. Creating a function that displays product sales at a certain month & year:

```
CREATE OR REPLACE FUNCTION ProductSales (product VARCHAR(30),
mnth INT,
    yr INT)
    RETURNS TABLE (product_name VARCHAR(512),
    product_id INT,
    sales FLOAT)
LANGUAGE plpgsql
AS $$

DECLARE
    rslt RECORD;

BEGIN
    FOR rslt IN(SELECT o.product_id,
        p.product_name,
        SUM(o.purchase_sum) AS prd_sum
        FROM orders_temp AS o
        LEFT JOIN (SELECT DISTINCT(products.product_name), products.product_id F
ROM products) AS p
            USING (product_id)
        WHERE LOWER(p.product_name) LIKE LOWER(product)
            AND DATE_PART('MONTH', o.order_date) = mnth AND DATE_PART('YEAR', o.
order_date)= yr
        GROUP BY o.product_id, p.product_name
        ORDER BY prd_sum DESC)
        LOOP product_name := rslt.product_name;
            product_id := rslt.product_id;
            sales := ROUND(rslt.prd_sum::NUMERIC, 2);
```

```
        RETURN NEXT;
        END LOOP;
END;
$$
```

For example, here are the sales of monitors in January 2024:

```
SELECT * FROM ProductSales('%onitor', 01, 2024);
```

	product_name character varying	product_id integer	sales double precision
1	34in Ultrawide Monitor	493	179080
2	20in Monitor	168	34426.87
3	27in FHD Monitor	443	21000

Conclusions

Within this mini-project we took a look at some fictional sales data. Using temporary table functionality we got access to the total purchase sum of each order based on the respective price validity period. We have also calculated the moving average of sales for the last 2 available months of data.

Table-valued functions allowed us to create some functions to automate such calculations as the most active customer, the most popular product and product sales in certain month.

Going forward, some other possible perspectives we could enrich this analysis with may enclude a look at sales tendencies by customer age.

Links

Link to the schema codes of all 3 tables

Contacts

lianazaripovar@gmail.com

tg: @zaripova_liana