SQL queries and outputs of *e-commerce data analysis project*

Basic Queries

1. Find the top 10 most frequently purchased products.

Code:

```
SELECT oi.product_id, p.product_category,
       COUNT(oi.product_id) AS freq_purchased
FROM order_items oi
JOIN products p ON p.product_id = oi.product_id
GROUP BY oi.product_id, p.product_category
ORDER BY freq_purchased DESC
LIMIT 10;
```

Output:

product_id	product_category	freq_purchased
aca2eb7d00ea1a7b8ebd4e68314663af	Furniture Decoration	527
99a4788cb24856965c36a24e339b6058	bed table bath	488
422879e10f46682990de24d770e7f83d	Garden tools	484
389d119b48cf3043d311335e499d9c6b	Garden tools	392
368c6c730842d78016ad823897a372db	Garden tools	388
53759a2ecddad2bb87a079a1f1519f73	Garden tools	373
d1c427060a0f73f6b889a5c7c61f2ac4	computer accessories	343
53b36df67ebb7c41585e8d54d6772e08	Watches present	323
154e7e31ebfa092203795c972e5804a6	HEALTH BEAUTY	281
3dd2a17168ec895c781a9191c1e95ad7	computer accessories	274

2. List the number of sellers operating in each state.

Code:

```
SELECT seller_state,
       COUNT(DISTINCT seller_id) AS num_sellers
FROM sellers
GROUP BY seller_state
ORDER BY num_sellers DESC;
```

Output:

seller_state	num_sellers
SP	1849
PR	349
MG	244
SC	190
RJ	171
RS	129
GO	40
DF	30
ES	23
BA	19
CE	13

3. Find the number of products listed in each category.

Code:

```
SELECT product_category,
       COUNT(*) AS total_products
FROM products
WHERE product_category IS NOT NULL
GROUP BY product_category
ORDER BY total_products DESC;
```

Output:

product_category	total_products
bed table bath	3029
sport leisure	2867
Furniture Decoration	2657
HEALTH BEAUTY	2444
housewares	2335
automotive	1900
computer accessories	1639
toys	1411
Watches present	1329
telephony	1134
babies	919

4. Find the total sales per category.

Code:

```
-- I have included shipping cost as well.
SELECT p.product_category AS product_category, ROUND(SUM(oi.price +
oi.freight_value),2) AS category_sales
FROM products p
JOIN order_items oi ON oi.product_id = p.product_id
GROUP BY p.product_category
ORDER BY category_sales DESC;
```

Output:

product_category	category_sales
HEALTH BEAUTY	1441248.07
Watches present	1305541.61
bed table bath	1241681.72
sport leisure	1156656.48
computer accessories	1059272.4
Furniture Decoration	902511.79
housewares	778397.77
Cool Stuff	719329.95
automotive	685384.32
Garden tools	584219.21
toys	561372.55

5. Calculate the percentage of orders that were paid in installments.

Code:

```
SELECT
COUNT((CASE WHEN payment_installments > 1 THEN order_id END))*100/COUNT(*) AS
perc_of_installment_orders
FROM payments;
```

Output:

	perc_of_installment_orders
▶	49.4176

Intermediate Queries

1. Calculate the average delivery time (difference between order purchase and delivery) by state.

Code:

```
SELECT c.customer_state,
       AVG(datediff(o.order_delivered_customer_date, o.order_purchase_timestamp))
AS avg_delivery_time_days
FROM orders o
JOIN customers c ON c.customer_id = o.customer_id
WHERE o.order_delivered_customer_date IS NOT NULL
GROUP BY c.customer_state
ORDER BY avg_delivery_time_days;
```

Output:

customer_state	avg_delivery_time_days
SP	8.7005
PR	11.9380
MG	11.9465
DF	12.8990
SC	14.9075
RJ	15.2377
RS	15.2485
GO	15.5360
MS	15.5449
ES	15.7238
TO	17.5985
MT	18.0034
PE	18.3955
RN	19.2236
BA	19.2786
RO	19.2840
PI	19.3950

2. Identify the top 5 product categories with the highest return rate.

Code:

```
-- To get different order status to decide which comes under returns.
SELECT DISTINCT order_status,
       COUNT(*) AS num_orders
FROM orders
GROUP BY order_status;
```

Output:

order_status	num_orders
delivered	96478
invoiced	314
shipped	1107
processing	301
unavailable	609
canceled	625
created	5
approved	2

Code cont.:

```
-- For this analysis, assuming all order status other than delivered as
returns/not delivered.
SELECT p.product_category AS top_return_product_category,
       CONCAT(ROUND(COUNT(CASE WHEN o.order_status <> 'delivered' THEN 1
END)*100/ COUNT(*),2),'%') AS return_rate
FROM products p
JOIN order_items oi ON oi.product_id = p.product_id
JOIN orders o ON o.order_id = oi.order_id
WHERE p.product_category IS NOT NULL
GROUP BY p.product_category
ORDER BY COUNT(CASE WHEN o.order_status <> 'delivered' THEN 1 END)/ COUNT(*) DESC
-- need for this instead return_rate in ORDER BY is because with concat of %
retun_rate becomes string and hence sorts incorrectly
LIMIT 5;
```

Output:

top_return_product_category	return_rate
Fashion Children's Clothing	12.50%
PC Gamer	11.11%
Kitchen portable and food coach	6.67%
Fashion Women's Clothing	6.25%
Art	5.74%

3. Find the relationship between installment payments and order value (average order value when paid in installments vs. full payment).

Code:

```
/*
In the payments table one order(i.e. one order_id) has multiple rows and they are
not for different installments,
instead different payment_type i.e. customer has choosen to pay using different
method
(credit card + multiple vouchers or UPI + multiple vouchers) for that order even
if done in one-installment(full_payment at once).
For better understanding run the below query,
and match the total_payment_for_order_id with order_items table for a specific
order_id

SELECT *,
       COUNT(*) OVER(PARTITION BY order_id) AS count_order_id,
       SUM(payment_value) OVER(PARTITION BY order_id) AS
total_payment_for_order_id
FROM payments
ORDER BY count_order_id DESC, order_id, payment_sequential;

-- matching in order_items
SELECT *
FROM order_items
WHERE order_id = 'fa65dad1b0e818e3ccc5cb0e39231352';
*/
WITH order_payments AS (
       SELECT order_id,
              -- Handling 2 outlier rows with '0' installments and treating them
as '1'
         CASE WHEN MAX(payment_installments) = 0 THEN 1
                     ELSE MAX(payment_installments)
              END AS installments,
         SUM(payment_value) AS total_payment
    FROM payments
    GROUP BY order_id
)
SELECT installments,
       ROUND(AVG(total_payment), 2) AS avg_order_value,
    COUNT(order_id) AS 'num_orders'
FROM order_payments
GROUP BY installments
ORDER BY installments;
```

Output:

installments	avg_order_value	num_orders
1	121.04	48270
2	129.12	12363
3	144.36	10429
4	165.06	7070
5	184.89	5227
6	211.59	3908
7	189.34	1622
8	309.77	4251
9	204.58	644
10	418.75	5315
11	124.93	23
12	323.05	133
13	150.46	16
14	176.31	15
15	447.3	74
16	292.69	5
17	174.6	8
18	400.10	27

4. Find the average number of products per order, grouped by customer state.

Code:

```
WITH state_order_items AS (
	SELECT c.customer_state, oi.order_id,
		COUNT(*) AS num_items
	FROM order_items oi
	JOIN orders o ON o.order_id = oi.order_id
	JOIN customers c ON c.customer_id = o.customer_id
	GROUP BY c.customer_state, oi.order_id
)
SELECT customer_state,
	AVG(num_items) AS avg_items_per_order
FROM state_order_items
GROUP BY customer_state
ORDER BY avg_items_per_order DESC;
```

Output:

customer_state	avg_items_per_order
AP	1.2059
MT	1.1683
GO	1.1624
SC	1.1561
MS	1.1551
PR	1.1485
RS	1.1478
SP	1.1468
RJ	1.1424
MG	1.1373
AC	1.1358
DF	1.1322
PB	1.1316
BA	1.1313
RR	1.1304

5. Calculate the percentage of total revenue contributed by each product category.

Code:

```
SELECT p.product_category,
    ROUND(SUM(oi.price)*100/(SELECT SUM(price) FROM order_items),2) AS
perc_contribution_in_sales
FROM order_items oi
JOIN products p ON p.product_id = oi.product_id
GROUP BY p.product_category
ORDER BY perc_contribution_in_sales DESC;
```

Output:

product_category	perc_contribution_in_sales
HEALTH BEAUTY	9.26
Watches present	8.87
bed table bath	7.63
sport leisure	7.27
computer accessories	6.71
Furniture Decoration	5.37
Cool Stuff	4.67
housewares	4.65
automotive	4.36
Garden tools	3.57
toys	3.56
babies	3.03
perfumery	2.94
telephony	2.38
Furniture office	2.02
stationary store	1.7
PCs	1.64

Advanced Queries

1. Distribute the orders across spend tiers (Very Low, Low, Medium, High, Very High).

Code:

```
-- Let's create a grouped frequency distribution table first to get appropriate
spending groups
WITH order_totals AS (
    SELECT o.order_id,
        SUM(p.payment_value) AS total_order_value
    FROM orders o
    JOIN payments p ON o.order_id = p.order_id
    WHERE o.order_status = 'delivered'
    GROUP BY o.order_id
)
SELECT
    CONCAT(FLOOR(total_order_value/100)*100, '-',
FLOOR(total_order_value/100)*100 + 99) AS value_range,
    COUNT(*) AS order_count
FROM order_totals
GROUP BY FLOOR(total_order_value/100)
ORDER BY FLOOR(total_order_value/100);
```

Output:

value_range	order_count
0-99	45917
100-199	31018
200-299	9637
300-399	4073
400-499	1756
500-599	996
600-699	825
700-799	490
800-899	409
900-999	249
1000-1099	216
1100-1199	131
1200-1299	137
1300-1399	126
1400-1499	85
1500-1599	45
1600-1699	60
1700-1799	36

Code cont.:

```
-- On basis of above table clustering orders into 5 tiers as:-
WITH order_totals AS (
    SELECT o.order_id,
             SUM(p.payment_value) AS total_order_value
    FROM orders o
    JOIN payments p ON o.order_id = p.order_id
    WHERE o.order_status = 'delivered'
    GROUP BY o.order_id
),
order_tiers AS (
    SELECT order_id,
        CASE
            WHEN total_order_value < 50 THEN 'Very Low'
            WHEN total_order_value BETWEEN 50 AND 100 THEN 'Low'
            WHEN total_order_value BETWEEN 100 AND 300 THEN 'Medium'
            WHEN total_order_value BETWEEN 300 AND 500 THEN 'High'
            ELSE 'Very High'
        END AS spend_tier
    FROM order_totals
)
SELECT spend_tier,
    COUNT(*) AS order_count,
    CONCAT(ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2),'%') AS
percentage_distribution
FROM order_tiers
GROUP BY spend_tier
ORDER BY -- custom sorting
    CASE spend_tier
        WHEN 'Very Low' THEN 1
        WHEN 'Low' THEN 2
        WHEN 'Medium' THEN 3
        WHEN 'High' THEN 4
        WHEN 'Very High' THEN 5
    END;
```

Output:

	spend_tier	order_count	percentage_distribution
▶	Very Low	16451	17.05%
	Low	29469	30.55%
	Medium	40652	42.14%
	High	5829	6.04%
	Very High	4076	4.22%

2. Find top 5 sellers' market share in each category.

Code:

```
WITH seller_category_sales AS(
      SELECT p.product_category, oi.seller_id,
            ROUND(SUM(oi.price + oi.freight_value),2) AS seller_category_sale,
       DENSE_RANK() OVER(PARTITION BY p.product_category ORDER BY SUM(oi.price +
oi.freight_value) DESC) AS seller_rank
    FROM order_items oi
      JOIN products p ON p.product_id = oi.product_id
      WHERE p.product_category IS NOT NULL
      GROUP BY p.product_category, oi.seller_id
)
SELECT product_category, seller_id, seller_category_sale,
    seller_rank,
      ROUND(seller_category_sale*100/SUM(seller_category_sale) OVER(PARTITION BY
product_category),2) AS market_share_perc
FROM seller_category_sales
WHERE seller_rank <= 5
ORDER BY product_category, seller_category_sale DESC;

-- CTE could be avoided here by calculating market_share directly in select
without using CTE, but for better readability I used CTE
```

Output:

product_category	seller_id	seller_category_sale	seller_rank	market_share_perc
Agro Industria e Comercio	e59aa562b9f8076dd550fcddf0e73491	33803.97	1	54.38
Agro Industria e Comercio	6bd69102ab48df500790a8cecfc285c2	8359.02	2	13.45
Agro Industria e Comercio	f08a5b9dd6767129688d001acafc21e5	7796.21	3	12.54
Agro Industria e Comercio	2528744c5ef5d955adc318720a94d2e7	6508.7	4	10.47
Agro Industria e Comercio	31ae0774c17fabd06ff707cc5bde005f	5690.23	5	9.15
Art	c31eff8334d6b3047ed34bebd4d62c36	12852.22	1	58.1
Art	ee27a8f15b1dded4d213a468ba4eb391	6726.66	2	30.41
Art	955fee9216a65b617aa5c0531780ce60	1067.7	3	4.83
Art	b561927807645834b59ef0d16ba55a24	836.9	4	3.78
Art	0d85bbda9889ce1f7e63778d24f346eb	638.77	5	2.89
Arts and Crafts	55f7a3319d80f7fdf078b8f03e6725fe	745.7	1	40.89
Arts and Crafts	04ee0ec01589969663ba5967c0e0bdc0	314.32	2	17.23
Arts and Crafts	05730013efda596306417c3b09302475	306.72	3	16.82
Arts and Crafts	0ddefe3c7a032b91f4e25b9c3a08fca1	270.25	4	14.82
Arts and Crafts	ef728fa1f17436c91ed1ccd03dcf9631	186.82	5	10.24
audio	7d13fca15225358621be4086e1eb0964	19487.87	1	45.64
audio	4869f7a5dfa277a7dca6462dcf3b52b2	10005.43	2	23.43
audio	b33e7c55446eabf8fe1a42d037ac7d6d	5804.2	3	13.8

3. Calculate the month_on_month growth rate of total sales.

Code:

```
WITH monthly_sales AS (
SELECT
      YEAR(o.order_purchase_timestamp) AS year,
    MONTH(o.order_purchase_timestamp) AS month_num,
    MONTHNAME(o.order_purchase_timestamp) AS month,
    ROUND(SUM(p.payment_value),2) AS monthly_sales
FROM orders o
JOIN payments p ON p.order_id = o.order_id
GROUP BY year, month_num, month
ORDER BY year, month_num
)
SELECT year, month, monthly_sales,
    LAG(monthly_sales,1) OVER(ORDER BY year, month_num) AS prev_month_sale,
    ROUND((monthly_sales - LAG(monthly_sales,1) OVER(ORDER BY year,
month_num))*100/LAG(monthly_sales,1) OVER(ORDER BY year, month_num),2) AS
    Mom_growth
    FROM monthly_sales;
```

Output:

year	month	monthly_sales	prev_month_sale	Mom_growth
2016	September	252.24	NULL	NULL
2016	October	59090.48	252.24	23326.29
2016	December	19.62	59090.48	-99.97
2017	January	138488.04	19.62	705751.38
2017	February	291908.01	138488.04	110.78
2017	March	449863.6	291908.01	54.11
2017	April	417788.03	449863.6	-7.13
2017	May	592918.82	417788.03	41.92
2017	June	511276.38	592918.82	-13.77
2017	July	592382.92	511276.38	15.86
2017	August	674396.32	592382.92	13.84
2017	September	727762.45	674396.32	7.91
2017	October	779677.88	727762.45	7.13
2017	November	1194882.8	779677.88	53.25
2017	December	878401.48	1194882.8	-26.49
2018	January	1115004.18	878401.48	26.94
2018	February	992463.34	1115004.18	-10.99
2018	March	1159652.12	992463.34	16.85

4. Calculate the cumulative sales per month for each year.

Code:

```
WITH monthly_sales AS (
SELECT
      YEAR(o.order_purchase_timestamp) AS year,
    MONTH(o.order_purchase_timestamp) AS month_num,
    MONTHNAME(o.order_purchase_timestamp) AS month,
    ROUND(SUM(p.payment_value),2) AS monthly_sales
FROM orders o
JOIN payments p ON p.order_id = o.order_id
GROUP BY year, month_num, month
ORDER BY year, month_num
)
SELECT
      CONCAT(month,' ', year) AS Month,
    monthly_sales,
    ROUND(SUM(monthly_sales) OVER(PARTITION BY year ORDER BY month_num),2) AS
monthly_cumulative_sales
FROM monthly_sales
ORDER BY year, month_num;
```

Output:

Month	monthly_sales	monthly_cumulative_sales
September 2016	252.24	252.24
October 2016	59090.48	59342.72
December 2016	19.62	59362.34
January 2017	138488.04	138488.04
February 2017	291908.01	430396.05
March 2017	449863.6	880259.65
April 2017	417788.03	1298047.68
May 2017	592918.82	1890966.5
June 2017	511276.38	2402242.88
July 2017	592382.92	2994625.8
August 2017	674396.32	3669022.12
September 2017	727762.45	4396784.57
October 2017	779677.88	5176462.45
November 2017	1194882.8	6371345.25
December 2017	878401.48	7249746.73
January 2018	1115004.18	1115004.18
February 2018	992463.34	2107467.52

5. Statewise Delivery performance (Eary, On Time, Late).

Code:

```
WITH delivery_performance AS (
    SELECT c.customer_state,
        CASE
            WHEN DATEDIFF(o.order_delivered_customer_date,
o.order_estimated_delivery_date) < 0 THEN 'Early'
            WHEN DATEDIFF(o.order_delivered_customer_date,
o.order_estimated_delivery_date) = 0 THEN 'On Time'
            ELSE 'Late'
        END AS delivery_status
    FROM orders o
    JOIN customers c ON c.customer_id = o.customer_id
    WHERE o.order_status = 'delivered'
      AND o.order_delivered_customer_date IS NOT NULL
      AND o.order_estimated_delivery_date IS NOT NULL
)
SELECT customer_state, delivery_status,
    COUNT(*) AS orders_count,
    ROUND(COUNT(*)*100.0 / SUM(COUNT(*)) OVER(PARTITION BY customer_state), 2) AS
percentage
FROM delivery_performance
GROUP BY customer_state, delivery_status
ORDER BY customer_state, delivery_status;
```

Output:

customer_state	delivery_status	orders_count	percentage
AC	Early	77	96.25
AC	Late	3	3.75
AL	Early	302	76.07
AL	Late	85	21.41
AL	On Time	10	2.52
AM	Early	139	95.86
AM	Late	4	2.76
AM	On Time	2	1.38
AP	Early	64	95.52
AP	Late	2	2.99
AP	On Time	1	1.49
BA	Early	2799	85.96
BA	Late	396	12.16
BA	On Time	61	1.87
CE	Early	1083	84.68
CE	Late	176	13.76
CE	On Time	20	1.56