SQL queries and outputs of *e_commerce data analysis project*

Basic Queries

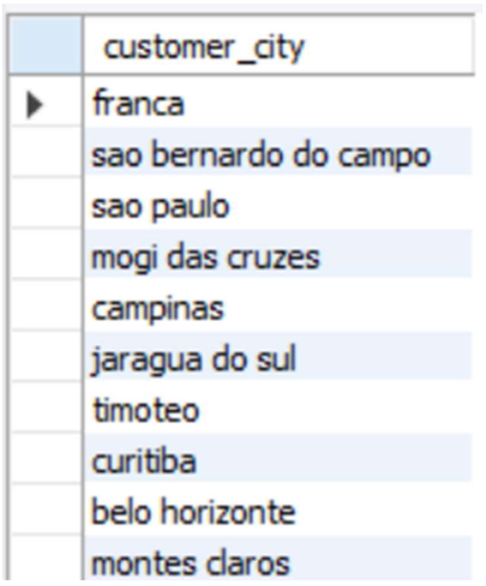
1. List all unique cities where customers are located.

Code:

SELECT DISTINCT customer_city

FROM customers;

Output:



customer_city
franca
sao bernardo do campo
sao paulo
mogi das cruzes
campinas
jaragua do sul
timoteo
curitiba
belo horizonte
montes claros

2. Count the number of orders placed in 2017.

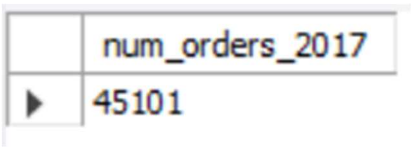
Code:

SELECT COUNT(order_id) AS num_orders_2017

FROM orders

WHERE YEAR(order_purchase_timestamp) = 2017;

Output:



num_orders_2017
45101

3. Find the total sales per category.

Code:

SELECT p.product_category AS product_category, ROUND(SUM(oi.price + oi.freight_value),2) AS category_sales -- if we want to include shipping cost as well

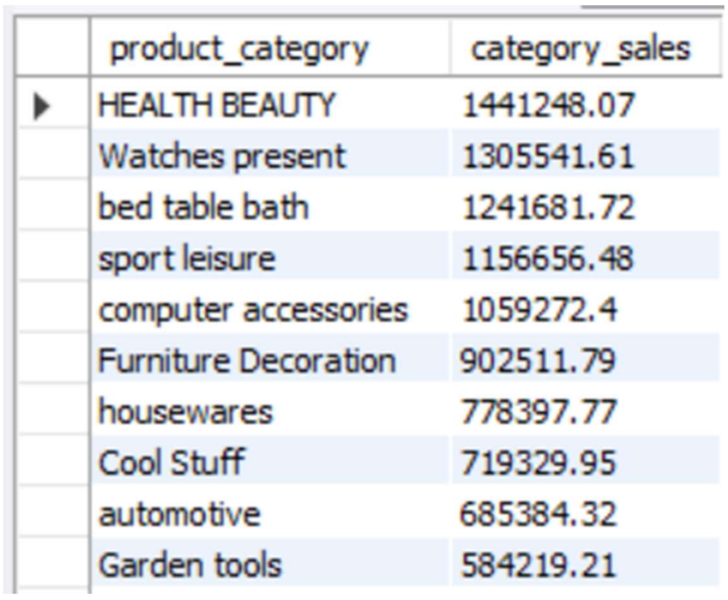
FROM products p

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

Code:

SELECT p.product_category AS product_category, ROUND(SUM(oi.price),2) AS category_sales -- if we don't want to include shipping cost as well

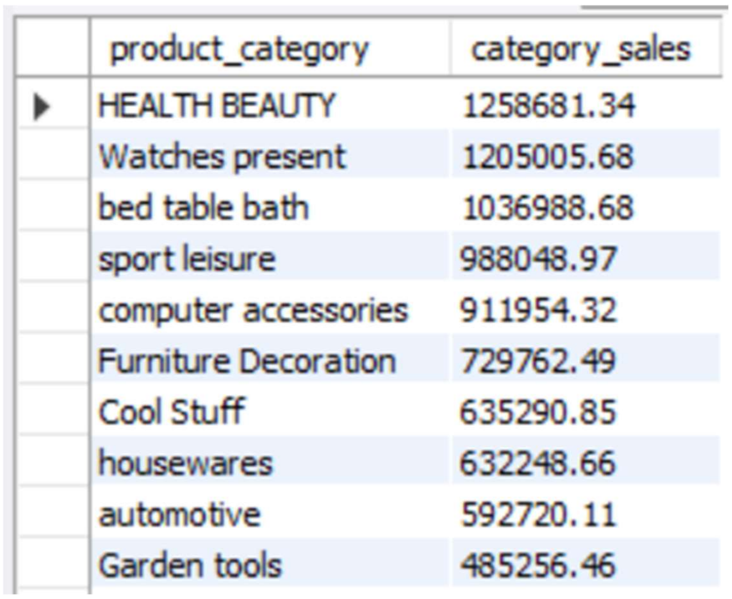
FROM products p

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	1258681.34
	Watches present	1205005.68
	bed table bath	1036988.68
	sport leisure	988048.97
	computer accessories	911954.32
	Furniture Decoration	729762.49
	Cool Stuff	635290.85
	housewares	632248.66
	automotive	592720.11
	Garden tools	485256.46

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

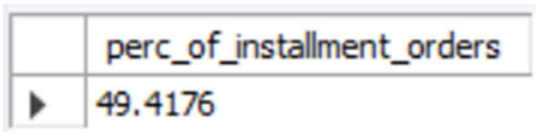
Code:

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

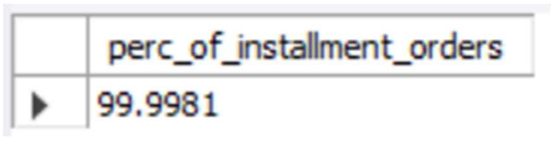
Code:

SELECT -- Considering payment_installments = 1 as installment as there are few rows with payment_installments = 0

*COUNT((CASE WHEN payment_installments >= 1 THEN order_id END))*100/COUNT(*) AS perc_of_installment_orders*

FROM payments;

Output:



	perc_of_installment_orders
▶	99.9981

5. Count the number of customers from each state.

Code:

SELECT

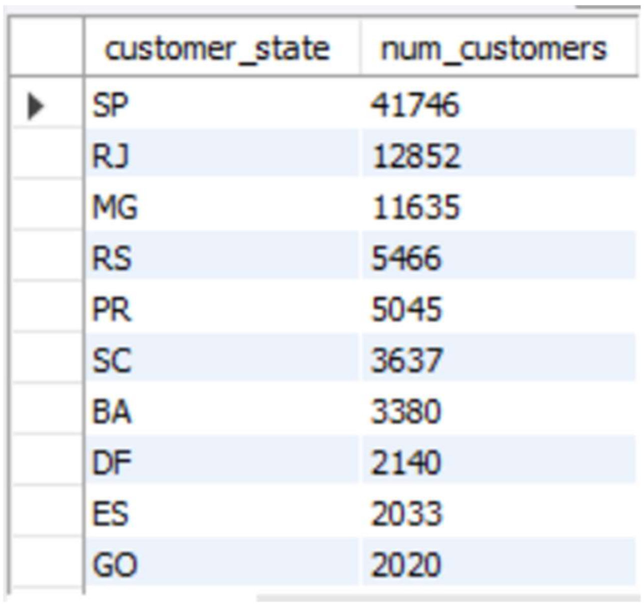
customer_state, COUNT(customer_id) AS num_customers

FROM customers

GROUP BY customer_state

ORDER BY num_customers DESC;

Output:



	customer_state	num_customers
▶	SP	41746
	RJ	12852
	MG	11635
	RS	5466
	PR	5045
	SC	3637
	BA	3380
	DF	2140
	ES	2033
	GO	2020

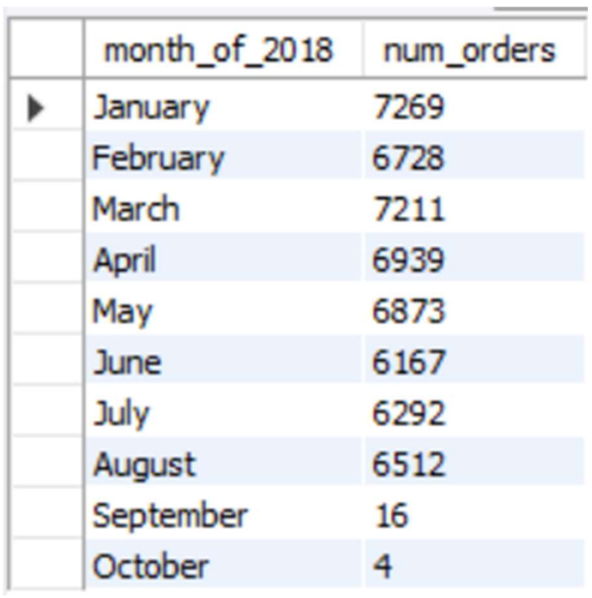
Intermediate Queries

1. Calculate the number of orders per month in 2018.

Code:

```
WITH num_orders_2018 AS(
        SELECT
        MONTHNAME(order_purchase_timestamp) AS month_of_2018,
        MONTH(order_purchase_timestamp) AS month_num,
        COUNT(order_id) AS num_orders
        FROM orders
        WHERE YEAR(order_purchase_timestamp) = 2018
        GROUP BY month_of_2018, month_num
        ORDER BY month_num
)
SELECT month_of_2018, num_orders
FROM num_orders_2018;
```

Output:



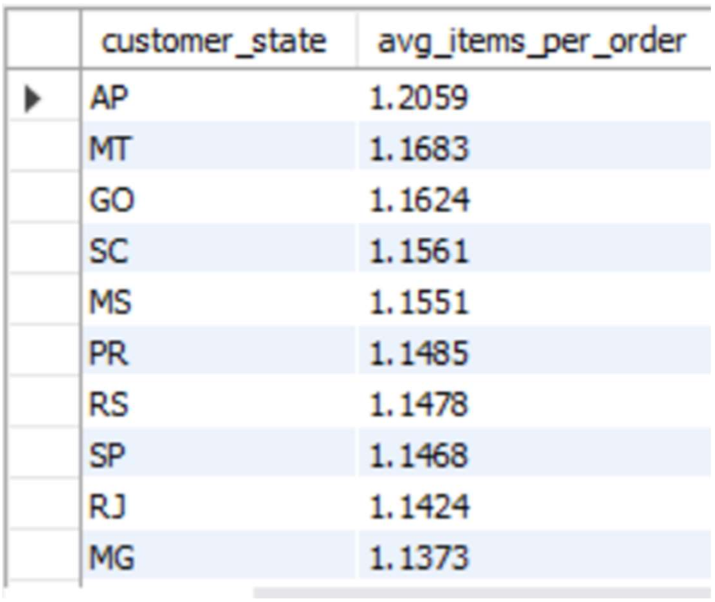
month_of_2018	num_orders
January	7269
February	6728
March	7211
April	6939
May	6873
June	6167
July	6292
August	6512
September	16
October	4

2. 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:



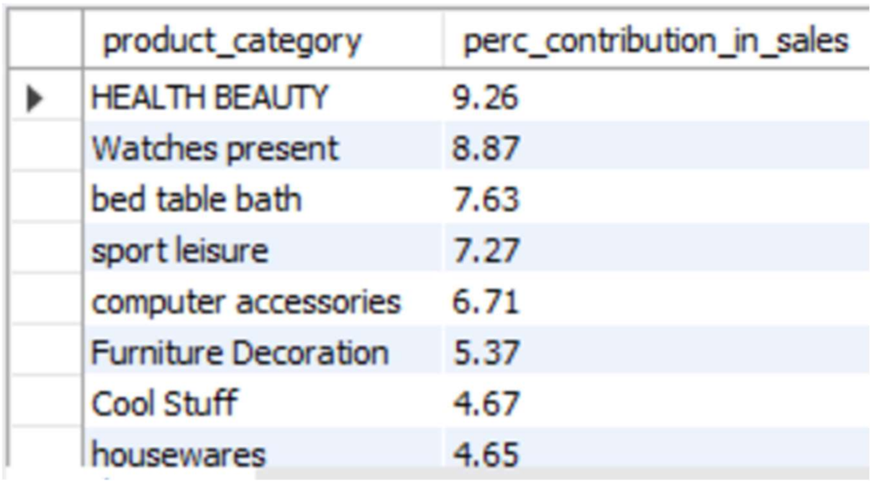
customer_state	avg_items_per_order
AP	1.2059
MT	1.1683
GO	1.1624
SC	1.1561
MS	1.1551
PR	1.1485
RS	1.1478
SP	1.1468
RJ	1.1424
MG	1.1373

3. 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

4. Identify the correlation between product price and the number of times a product has been purchased(total no. of products here will be from product_category)

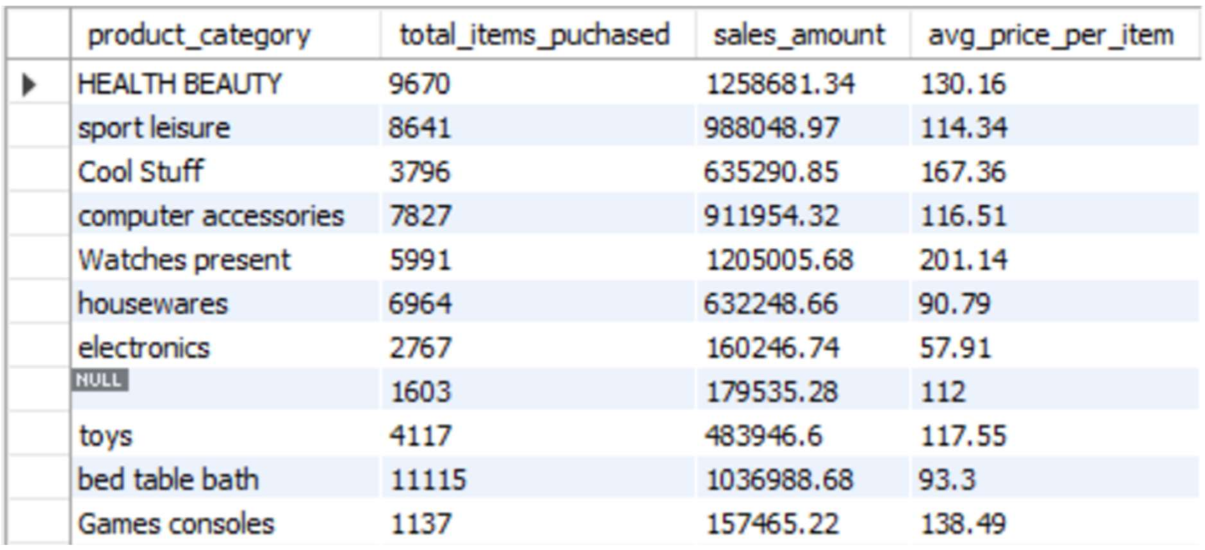
Code:

```
SELECT
        p.product_category,
  COUNT(oi.product_id) AS total_items_puchased,
  ROUND(SUM(oi.price),2) AS sales_amount,
  ROUND(AVG(price),2) AS avg_price_per_item
FROM order_items oi
JOIN products p ON p.product_id = oi.product_id
```

```
GROUP BY p.product_category;

/*

Correlation in python - code


query = """SELECT

            p.product_category,

    COUNT(oi.product_id) AS total_items_puchased,

    ROUND(SUM(oi.price),2) AS sales_amount,

    ROUND(AVG(price),2) AS avg_price_per_item

FROM order_items oi

JOIN products p ON p.product_id = oi.product_id

GROUP BY p.product_category;"""


cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data,columns = ["product_category", "total_items_puchased",
"sales_amount", "avg_price_per_item"])

df.head()

import numpy as np

arr1 = df["total_items_puchased"]

arr2 = df["avg_price_per_item"]


a = np.corrcoef([arr1,arr2])

print(f"The correlation coefficient is \033[1m{round(a[0][-1],3)}\033[0m." )

*/
```

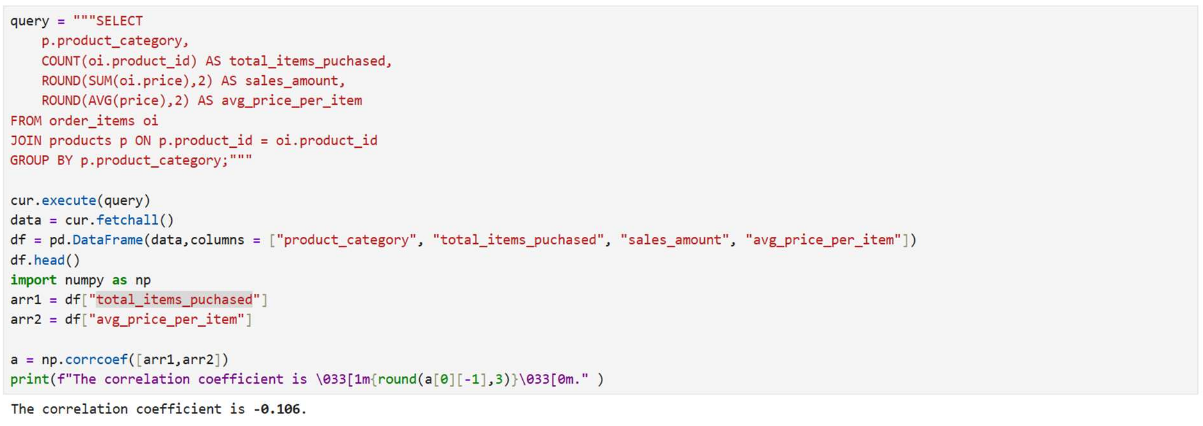
Output:

SQL Table



product_category	total_items_puchased	sales_amount	avg_price_per_item
HEALTH BEAUTY	9670	1258681.34	130.16
sport leisure	8641	988048.97	114.34
Cool Stuff	3796	635290.85	167.36
computer accessories	7827	911954.32	116.51
Watches present	5991	1205005.68	201.14
housewares	6964	632248.66	90.79
electronics	2767	160246.74	57.91
NULL	1603	179535.28	112
toys	4117	483946.6	117.55
bed table bath	11115	1036988.68	93.3
Games consoles	1137	157465.22	138.49

Python



```
query = """SELECT
    p.product_category,
    COUNT(oi.product_id) AS total_items_puchased,
    ROUND(SUM(oi.price),2) AS sales_amount,
    ROUND(AVG(price),2) AS avg_price_per_item
FROM order_items oi
JOIN products p ON p.product_id = oi.product_id
GROUP BY p.product_category;"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns = ["product_category", "total_items_puchased", "sales_amount", "avg_price_per_item"])
df.head()
import numpy as np
arr1 = df["total_items_puchased"]
arr2 = df["avg_price_per_item"]

a = np.corrcoef([arr1,arr2])
print(f"The correlation coefficient is \033[1m{round(a[0][-1],3)}\033[0m." )
```

```
The correlation coefficient is -0.106.
```

5. Calculate the total revenue generated by each seller, and rank them by revenue.

Code:

SELECT

oi.seller_id, ROUND(SUM(payment_value),2) AS seller_revenue,

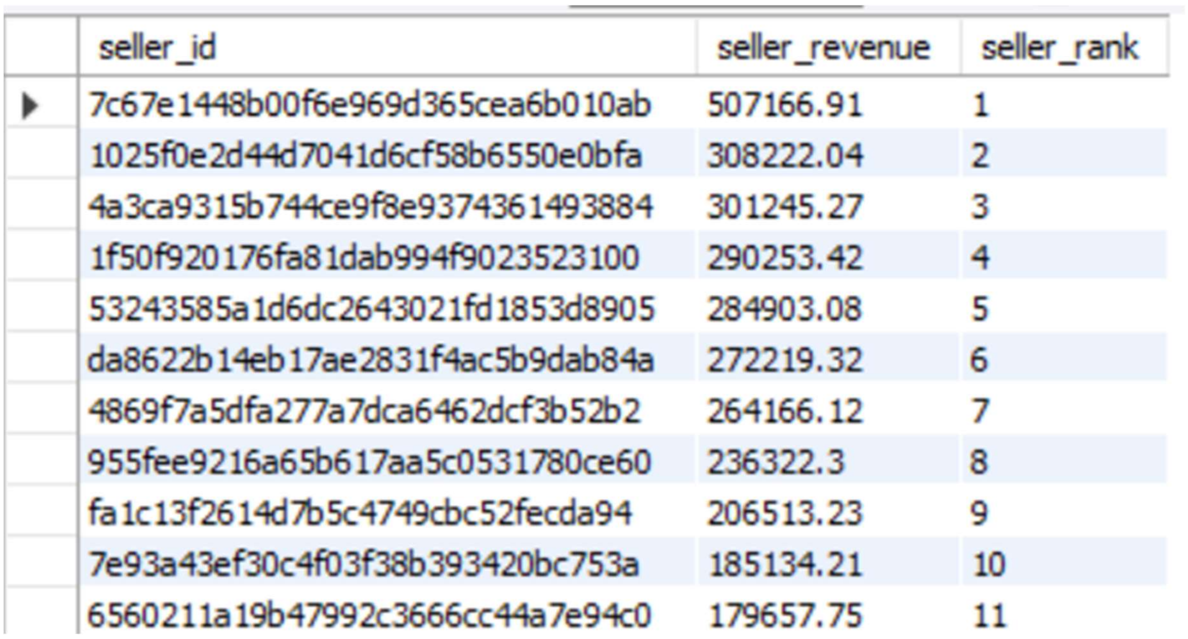
DENSE_RANK() OVER(ORDER BY SUM(payment_value) DESC) AS seller_rank

FROM order_items oi

JOIN payments p ON p.order_id = oi.order_id

GROUP BY oi.seller_id;

Output:



seller_id	seller_revenue	seller_rank
7c67e1448b00f6e969d365cea6b010ab	507166.91	1
1025f0e2d44d7041d6cf58b6550e0bfa	308222.04	2
4a3ca9315b744ce9f8e9374361493884	301245.27	3
1f50f920176fa81dab994f9023523100	290253.42	4
53243585a1d6dc2643021fd1853d8905	284903.08	5
da8622b14eb17ae2831f4ac5b9dab84a	272219.32	6
4869f7a5dfa277a7dca6462dcf3b52b2	264166.12	7
955fee9216a65b617aa5c0531780ce60	236322.3	8
fa1c13f2614d7b5c4749cbc52fecda94	206513.23	9
7e93a43ef30c4f03f38b393420bc753a	185134.21	10
6560211a19b47992c3666cc44a7e94c0	179657.75	11

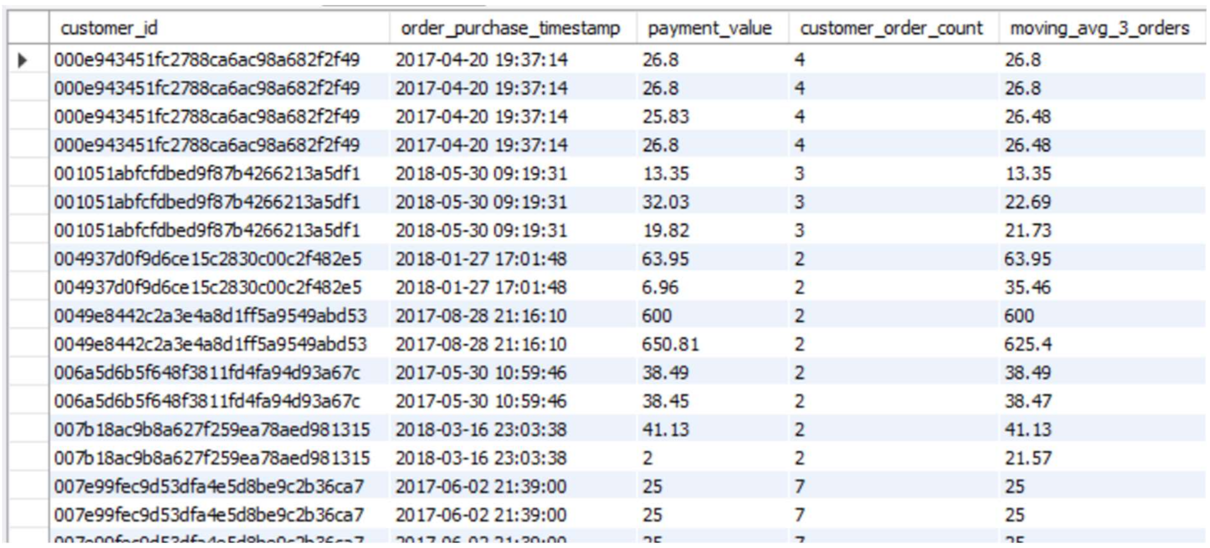
Advanced Queries

1. Calculate the moving average of order values for each customer over their order history.(say moving average of last 3 orders)

Code:

```
SELECT

t.customer_id, t.order_purchase_timestamp, t.payment_value, customer_order_count,
moving_avg_3_orders

FROM(SELECT

        o.customer_id, o.order_purchase_timestamp, p.payment_value,

  COUNT(*) OVER (PARTITION BY o.customer_id) AS customer_order_count,

  ROUND(AVG(p.payment_value)OVER(PARTITION BY o.customer_id ORDER BY
o.order_purchase_timestamp ROWS

  BETWEEN 2 PRECEDING AND CURRENT ROW),2)  AS moving_avg_3_orders

        FROM orders o

  JOIN payments p ON p.order_id = o.order_id) AS t

WHERE customer_order_count > 1

-- customers with More than 1 order will be filtered in this data and

-- hence moving average will be calculated as per 2, 3, 4 or more orders

;
```

Output:



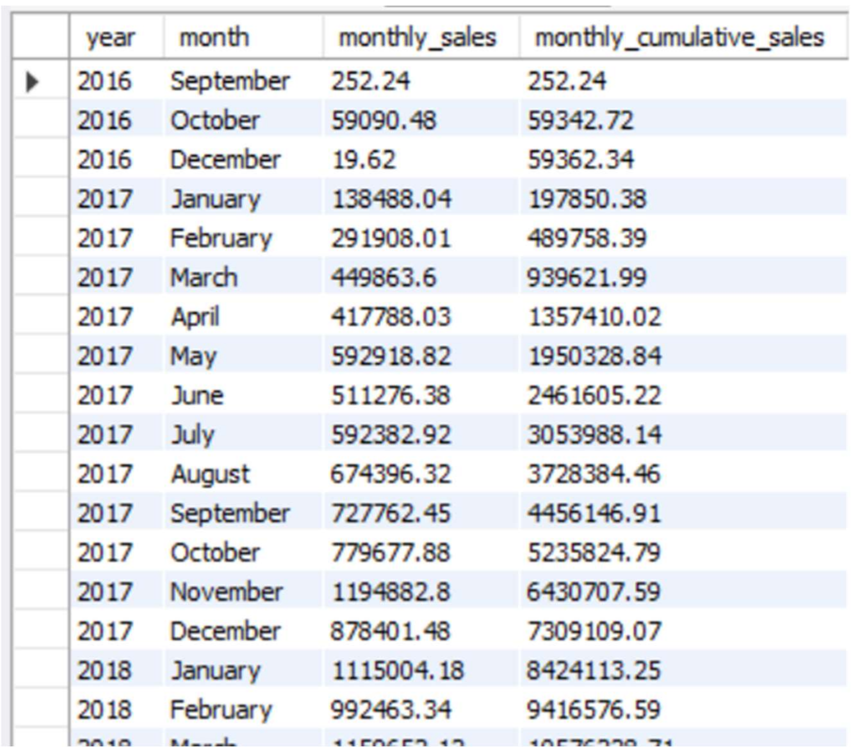
customer_id	order_purchase_timestamp	payment_value	customer_order_count	moving_avg_3_orders
000e943451fc2788ca6ac98a682f2f49	2017-04-20 19:37:14	26.8	4	26.8
000e943451fc2788ca6ac98a682f2f49	2017-04-20 19:37:14	26.8	4	26.8
000e943451fc2788ca6ac98a682f2f49	2017-04-20 19:37:14	25.83	4	26.48
000e943451fc2788ca6ac98a682f2f49	2017-04-20 19:37:14	26.8	4	26.48
001051abfcfdbed9f87b4266213a5df1	2018-05-30 09:19:31	13.35	3	13.35
001051abfcfdbed9f87b4266213a5df1	2018-05-30 09:19:31	32.03	3	22.69
001051abfcfdbed9f87b4266213a5df1	2018-05-30 09:19:31	19.82	3	21.73
004937d0f9d6ce15c2830c00c2f482e5	2018-01-27 17:01:48	63.95	2	63.95
004937d0f9d6ce15c2830c00c2f482e5	2018-01-27 17:01:48	6.96	2	35.46
0049e8442c2a3e4a8d1ff5a9549abd53	2017-08-28 21:16:10	600	2	600
0049e8442c2a3e4a8d1ff5a9549abd53	2017-08-28 21:16:10	650.81	2	625.4
006a5d6b5f648f3811fd4fa94d93a67c	2017-05-30 10:59:46	38.49	2	38.49
006a5d6b5f648f3811fd4fa94d93a67c	2017-05-30 10:59:46	38.45	2	38.47
007b18ac9b8a627f259ea78aed981315	2018-03-16 23:03:38	41.13	2	41.13
007b18ac9b8a627f259ea78aed981315	2018-03-16 23:03:38	2	2	21.57
007e99fec9d53dfa4e5d8be9c2b36ca7	2017-06-02 21:39:00	25	7	25
007e99fec9d53dfa4e5d8be9c2b36ca7	2017-06-02 21:39:00	25	7	25

2. 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
	year, month, monthly_sales,
   ROUND(SUM(monthly_sales) OVER(ORDER BY year, month_num),2) AS
monthly_cumulative_sales
   FROM monthly_sales;
```

Output:



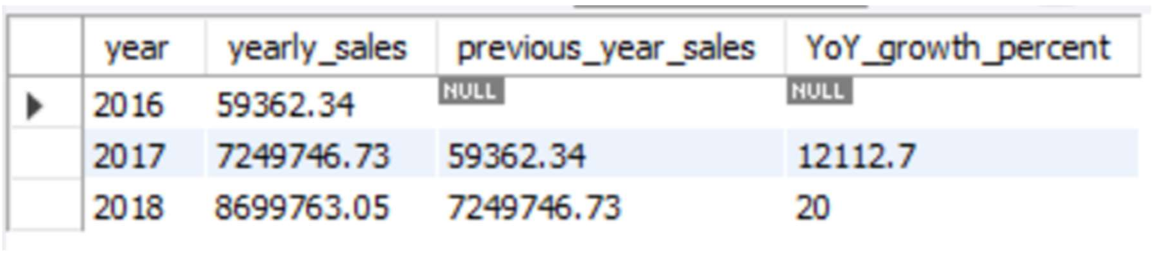
year	month	monthly_sales	monthly_cumulative_sales
2016	September	252.24	252.24
2016	October	59090.48	59342.72
2016	December	19.62	59362.34
2017	January	138488.04	197850.38
2017	February	291908.01	489758.39
2017	March	449863.6	939621.99
2017	April	417788.03	1357410.02
2017	May	592918.82	1950328.84
2017	June	511276.38	2461605.22
2017	July	592382.92	3053988.14
2017	August	674396.32	3728384.46
2017	September	727762.45	4456146.91
2017	October	779677.88	5235824.79
2017	November	1194882.8	6430707.59
2017	December	878401.48	7309109.07
2018	January	1115004.18	8424113.25
2018	February	992463.34	9416576.59
2018	March	1159652.12	10576228.71

3. Calculate the year-over-year growth rate of total sales.

Code:

```
SELECT
        year, yearly_sales, previous_year_sales,
  ROUND((yearly_sales - previous_year_sales)*100/previous_year_sales,2) AS
YoY_growth_percent
FROM (SELECT
                YEAR(o.order_purchase_timestamp) AS year,
ROUND(SUM(p.payment_value),2) AS yearly_sales,
                LAG(ROUND(SUM(p.payment_value),2),1) OVER(ORDER BY
YEAR(o.order_purchase_timestamp)) AS previous_year_sales
                FROM orders o
                JOIN payments p ON p.order_id = o.order_id
                GROUP BY year) AS year_sale;
```

Output:



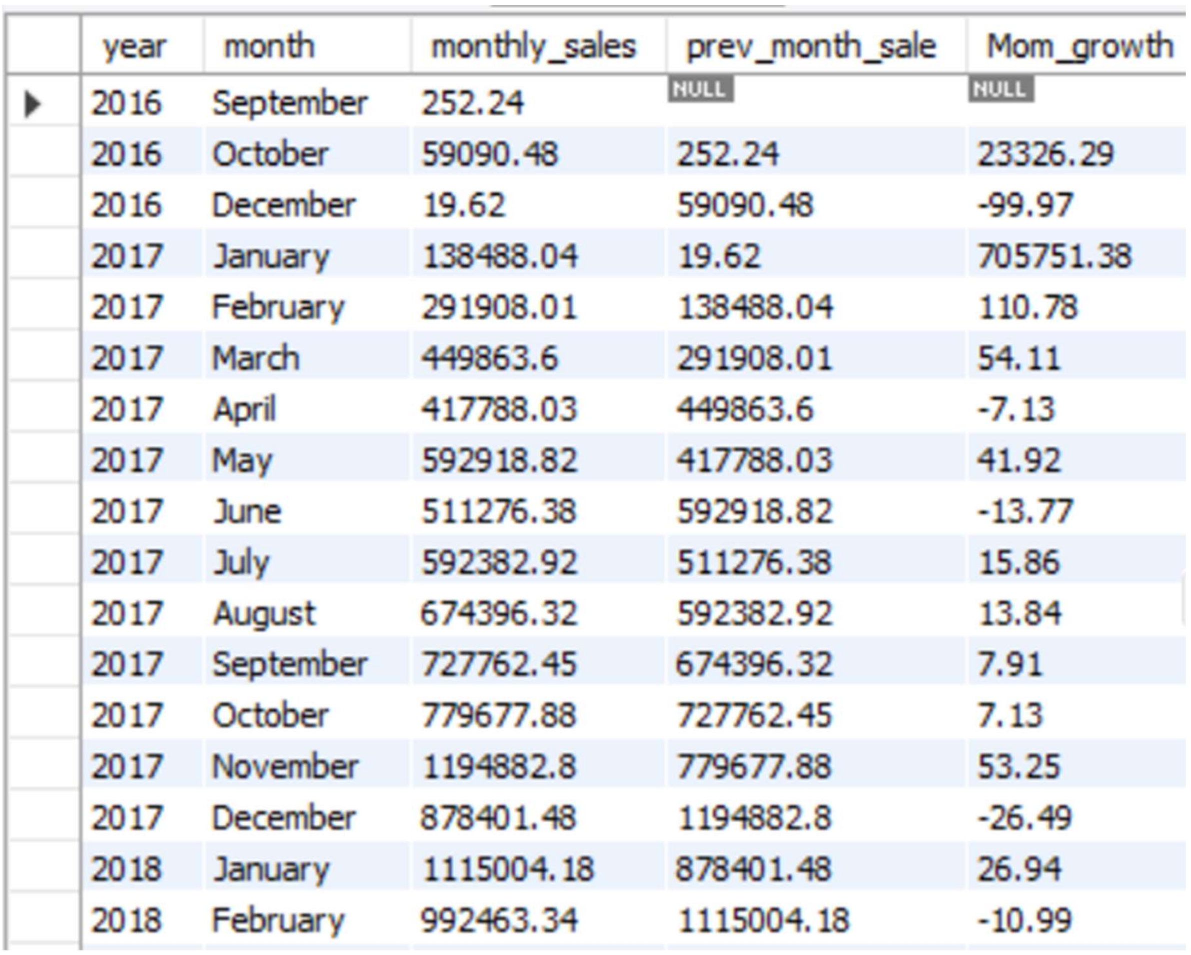
year	yearly_sales	previous_year_sales	YoY_growth_percent
2016	59362.34	NULL	NULL
2017	7249746.73	59362.34	12112.7
2018	8699763.05	7249746.73	20

4. 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:



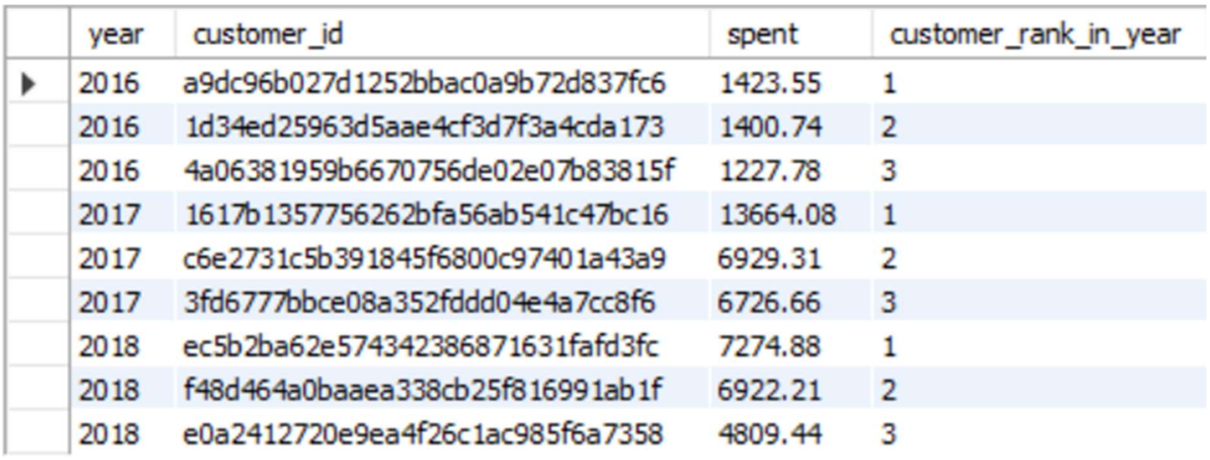
year	month	monthly_sales	prev_month_sale	Mom_growth
2016	September	252.24	NULL	NULL
2016	October	59090.48	252.24	23326.29
2016	December	19.62	59090.48	-99.97
2017	January	138488.04	19.62	705751.38
2017	February	291908.01	138488.04	110.78
2017	March	449863.6	291908.01	54.11
2017	April	417788.03	449863.6	-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

5. Identify the top 3 customers who spent the most money in each year.

Code:

```
WITH customer_spending AS (SELECT
        YEAR(o.order_purchase_timestamp) AS year,
  o.customer_id, ROUND(SUM(p.payment_value),2) AS spent,
  -- Don't partition by customer_id because then every customer will be assigned a new
group and respective rank i.e. 1
  DENSE_RANK() OVER(PARTITION BY YEAR(o.order_purchase_timestamp) ORDER BY
ROUND(SUM(p.payment_value),2) DESC) AS customer_rank_in_year
FROM orders o
JOIN payments p ON p.order_id = o.order_id
GROUP BY year, o.customer_id
ORDER BY year)
SELECT
        year, customer_id, spent, customer_rank_in_year
FROM customer_spending
WHERE customer_rank_in_year <= 3;
```

Output:



year	customer_id	spent	customer_rank_in_year
2016	a9dc96b027d1252bbac0a9b72d837fc6	1423.55	1
2016	1d34ed25963d5aae4cf3d7f3a4cda173	1400.74	2
2016	4a06381959b6670756de02e07b83815f	1227.78	3
2017	1617b1357756262bfa56ab541c47bc16	13664.08	1
2017	c6e2731c5b391845f6800c97401a43a9	6929.31	2
2017	3fd6777bbce08a352fddd04e4a7cc8f6	6726.66	3
2018	ec5b2ba62e574342386871631fafd3fc	7274.88	1
2018	f48d464a0baaea338cb25f816991ab1f	6922.21	2
2018	e0a2412720e9ea4f26c1ac985f6a7358	4809.44	3