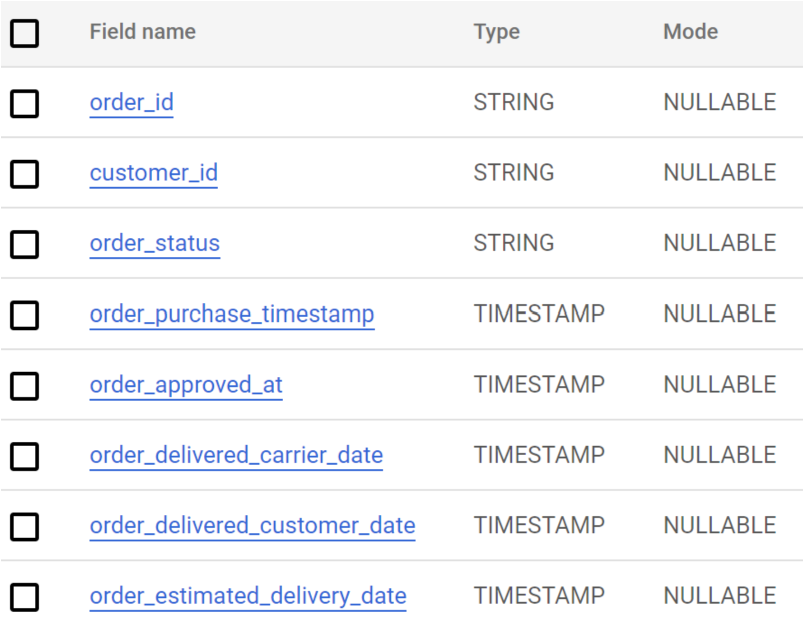
E-commerce Dataset Analysis

Using Google Bigquery SQL

Yasser A. Rahman
01/09/2023

Dataset Description

The dataset contains a real data of e-commerce retails transactions. The main fact table ‘orders’ contains a 99,441 transactions. The table has the following columns:

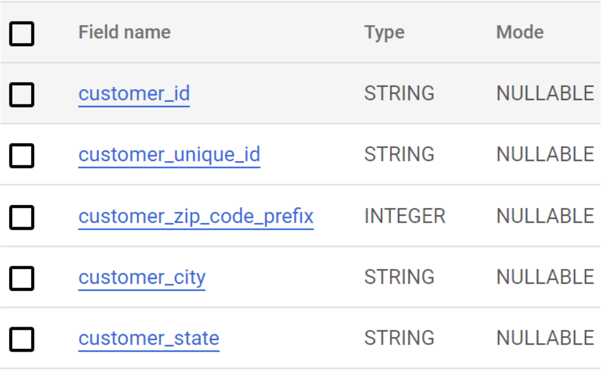


□	Field name	Type	Mode
□	order_id	STRING	NULLABLE
□	customer_id	STRING	NULLABLE
□	order_status	STRING	NULLABLE
□	order_purchase_timestamp	TIMESTAMP	NULLABLE
□	order_approved_at	TIMESTAMP	NULLABLE
□	order_delivered_carrier_date	TIMESTAMP	NULLABLE
□	order_delivered_customer_date	TIMESTAMP	NULLABLE
□	order_estimated_delivery_date	TIMESTAMP	NULLABLE

There are other associated tables as shown below:

Customers Table:

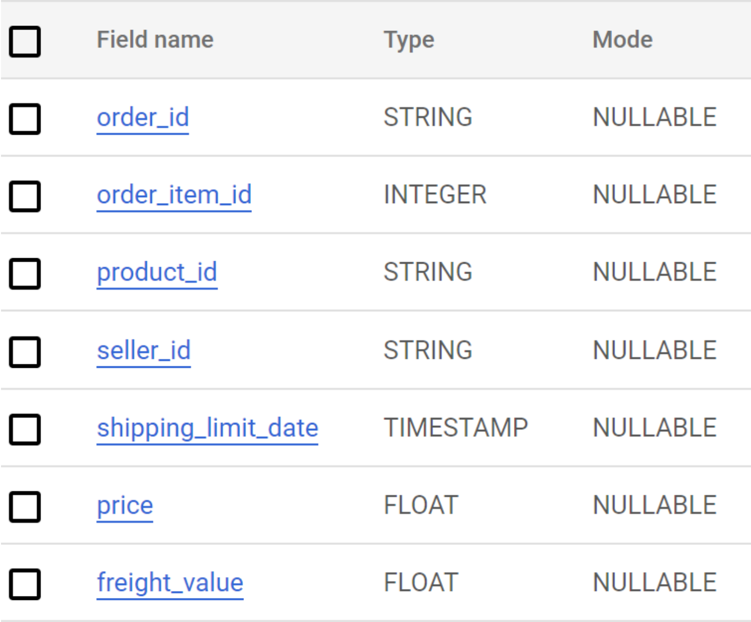
It has the same number of rows like orders table. But this is because each order will have a corresponding customer entry with a unique customer id.



□	Field name	Type	Mode
□	customer_id	STRING	NULLABLE
□	customer_unique_id	STRING	NULLABLE
□	customer_zip_code_prefix	INTEGER	NULLABLE
□	customer_city	STRING	NULLABLE
□	customer_state	STRING	NULLABLE

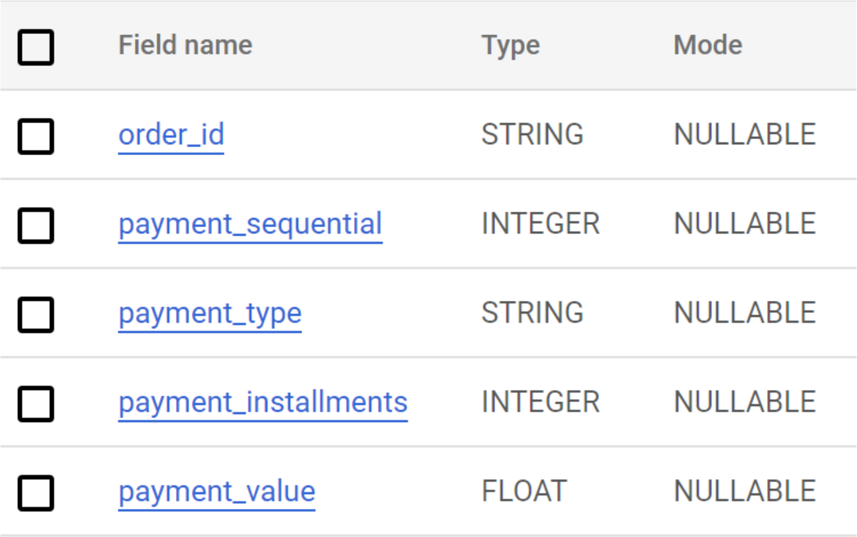
Order_items Table:

This table is to serve the relationship between orders and the items included in each order.



□	Field name	Type	Mode
□	order_id	STRING	NULLABLE
□	order_item_id	INTEGER	NULLABLE
□	product_id	STRING	NULLABLE
□	seller_id	STRING	NULLABLE
□	shipping_limit_date	TIMESTAMP	NULLABLE
□	price	FLOAT	NULLABLE
□	freight_value	FLOAT	NULLABLE

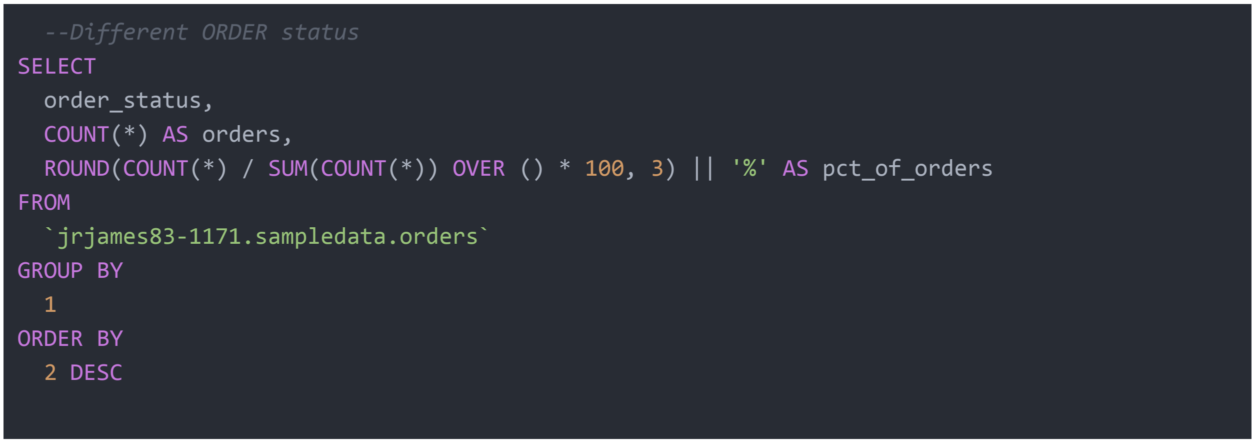
Order_payments Table:



□	Field name	Type	Mode
□	order_id	STRING	NULLABLE
□	payment_sequential	INTEGER	NULLABLE
□	payment_type	STRING	NULLABLE
□	payment_installments	INTEGER	NULLABLE
□	payment_value	FLOAT	NULLABLE

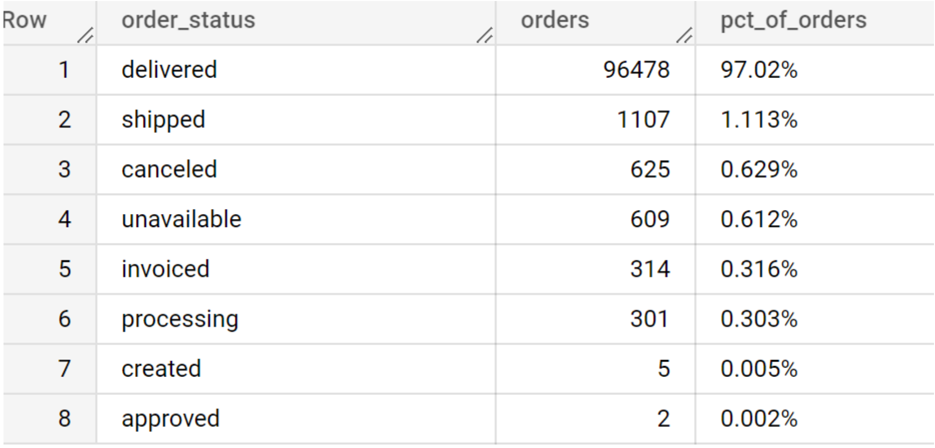
Q.1 Differe order status

```
--Different ORDER status
```

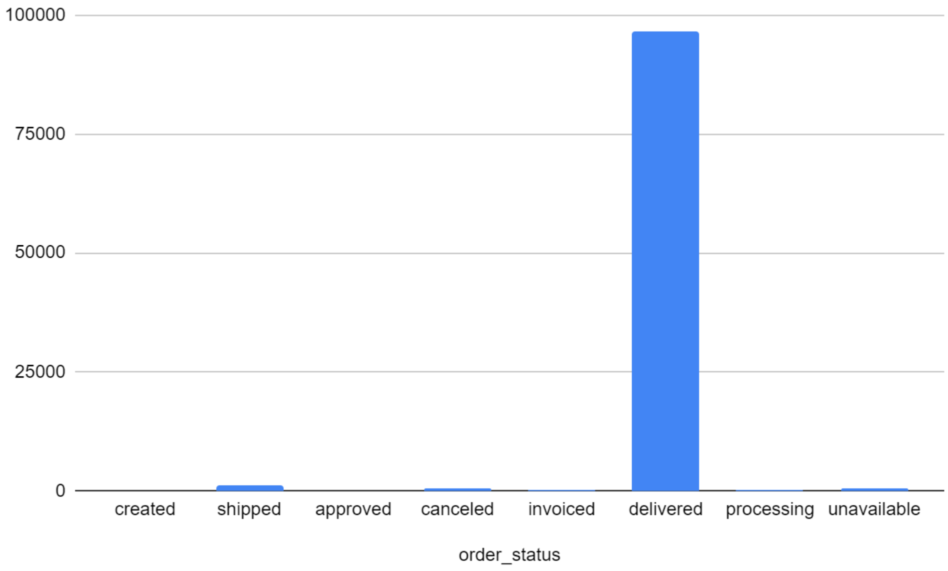


```
  --Different ORDER status
SELECT
  order_status,
  COUNT(*) AS orders,
  ROUND(COUNT(*) / SUM(COUNT(*)) OVER () * 100, 3) || '%' AS pct_of_orders
FROM
  `jrjames83-1171.sampledata.orders`
GROUP BY
  1
ORDER BY
  2 DESC
```

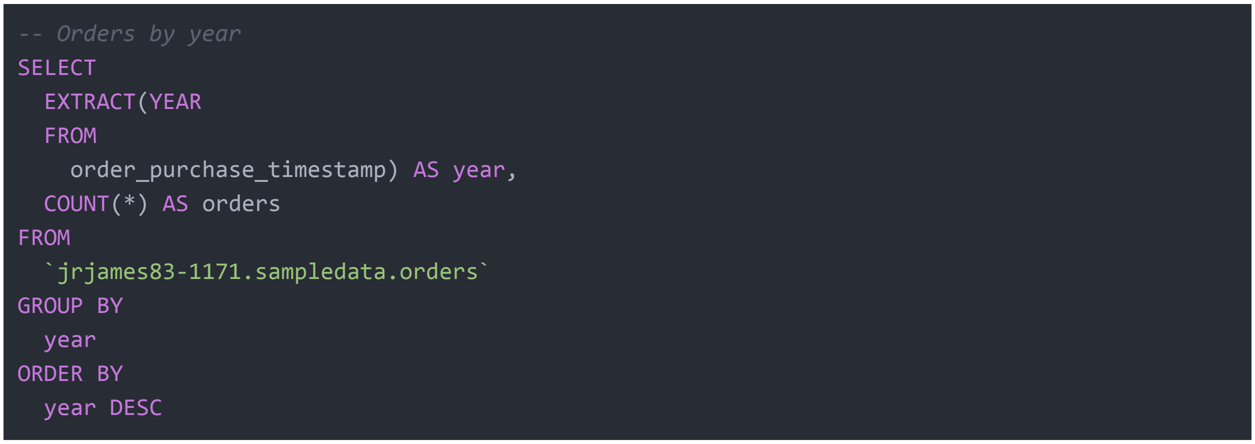
Sample output:



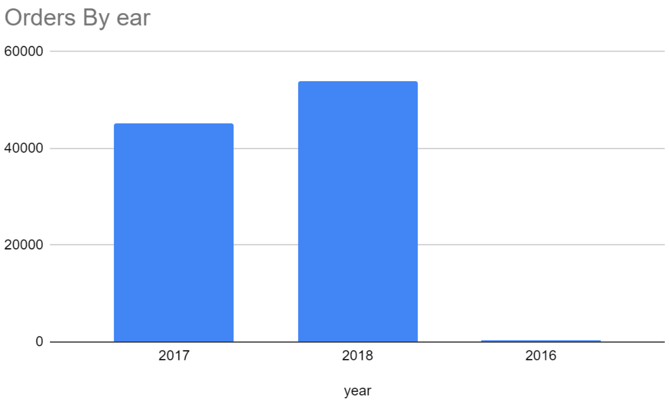
Row	order_status	orders	pct_of_orders
1	delivered	96478	97.02%
2	shipped	1107	1.113%
3	canceled	625	0.629%
4	unavailable	609	0.612%
5	invoiced	314	0.316%
6	processing	301	0.303%
7	created	5	0.005%
8	approved	2	0.002%



Q.2 How many orders were made each year?



```
-- Orders by year
SELECT
  EXTRACT(YEAR
  FROM
    order_purchase_timestamp) AS year,
  COUNT(*) AS orders
FROM
  `jrjames83-1171.sampledata.orders`
GROUP BY
  year
ORDER BY
  year DESC
```



Q.3 Orders Monthly Performance

```
-- Orders Monthly Performance
SELECT
  EXTRACT(YEAR FROM order_purchase_timestamp) AS Year,
  EXTRACT(MONTH FROM order_purchase_timestamp) AS Month,
  COUNT(*) AS orders
FROM
  `jrjames83-1171.sampledata.orders`
GROUP BY
  Year,Month
ORDER BY
  Year DESC, Month ASC
```

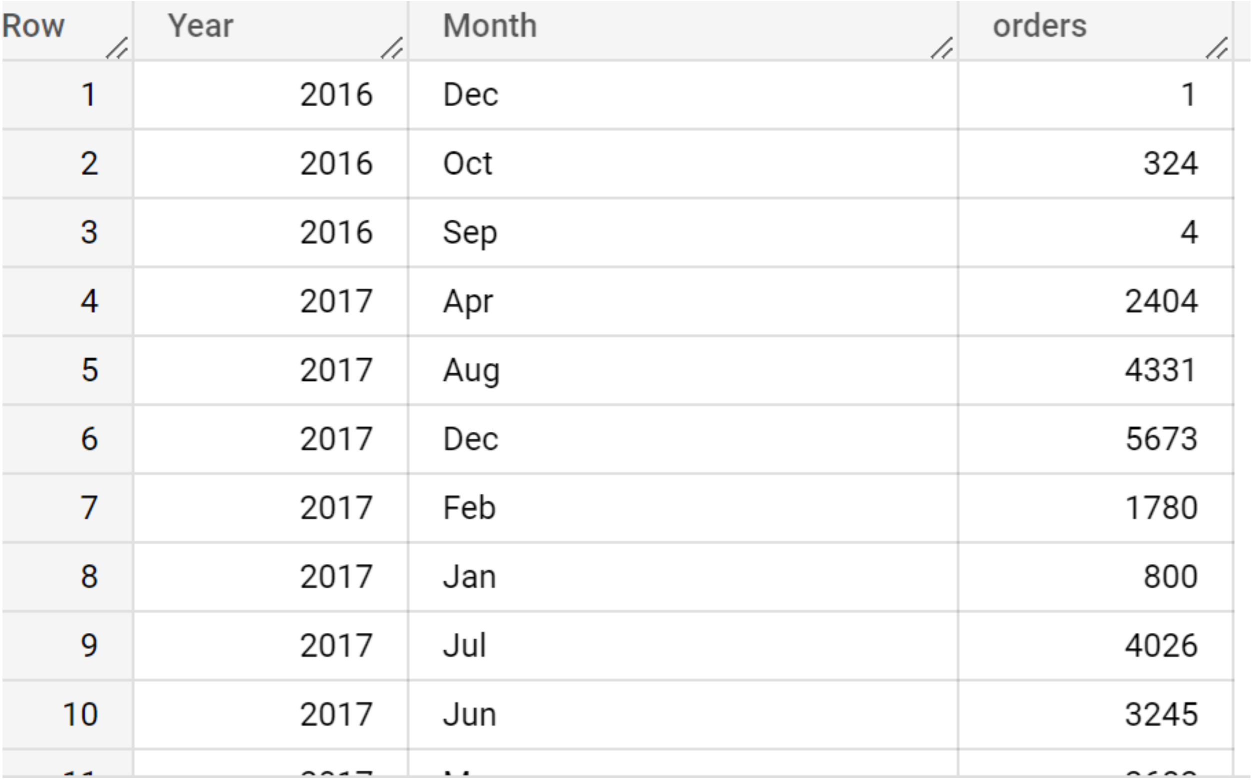
Sample Output:

Year	Month	orders
2016	9	4
2016	10	324
2016	12	1
2017	1	800
2017	2	1780
2017	3	2682
2017	4	2404
2017	5	3700
2017	6	3245
2017	7	4026
2017	8	4331
2017	9	4285
2017	10	4631
2017	11	7544
2017	12	5673
2018	1	7269
2018	2	6728
2018	3	7211
2018	4	6939
2018	5	6873
2018	6	6167

2018	7	6292
2018	8	6512
2018	9	16
2018	10	4

If we'd like to mention month name instead of month number, we shall use FORMAT() function.

```sql
-- Orders Monthly Performance with month name
SELECT
  EXTRACT(YEAR FROM order_purchase_timestamp) AS Year,
  FORMAT_DATE('%b',order_purchase_timestamp) AS Month,
  COUNT(*) AS orders
FROM
  `jrjames83-1171.sampledata.orders`
GROUP BY
  Year,Month
ORDER BY
  Year, Month
```



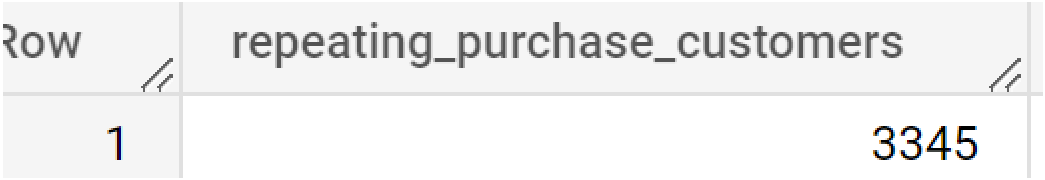
Row	Year	Month	orders
1	2016	Dec	1
2	2016	Oct	324
3	2016	Sep	4
4	2017	Apr	2404
5	2017	Aug	4331
6	2017	Dec	5673
7	2017	Feb	1780
8	2017	Jan	800
9	2017	Jul	4026
10	2017	Jun	3245

Q.4 How many customers have made more than one order?

```
  -- Get the number OF customers who made more than one ORDER
SELECT
  COUNT(c.customer_unique_id) - COUNT(DISTINCT c.customer_unique_id) AS
repeating_purchase_customers
FROM
  `jrjames83-1171.sampledata.orders` AS o
JOIN
  jrjames83-1171.sampledata.customers AS c
ON
  c.customer_id = o.customer_id;
```

Each order has a unique order_id and a unique customer_id associated with that order only. But to get any information related to customers, we need to use customer_unique_id instated.

Sample output

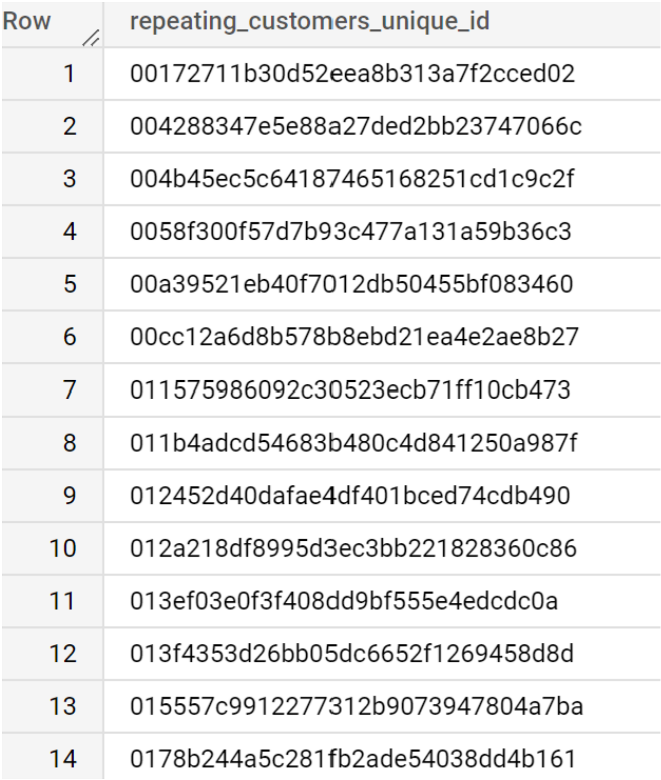


Row	repeating_purchase_customers
1	3345

Q5. Getting a list of the repeating-orders customers

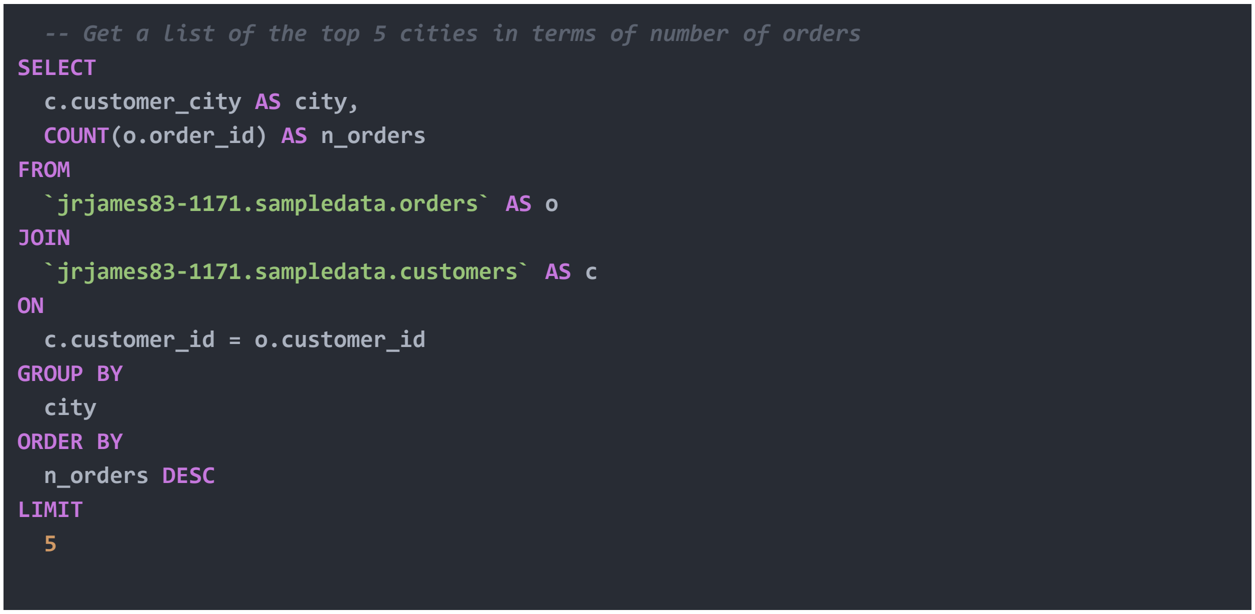
```
  -- Getting the list of repeating-orders customers
WITH
  t AS (
  SELECT
    c.customer_unique_id,
    ROW_NUMBER() OVER(PARTITION BY c.customer_unique_id ORDER BY order_purchase_timestamp)
AS order_number
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.customers` AS c
  ON
    c.customer_id = o.customer_id
  ORDER BY
    1,
    2 )
SELECT
  t.customer_unique_id
FROM
  t
WHERE
  t.order_number > 1
```

Sample Output:

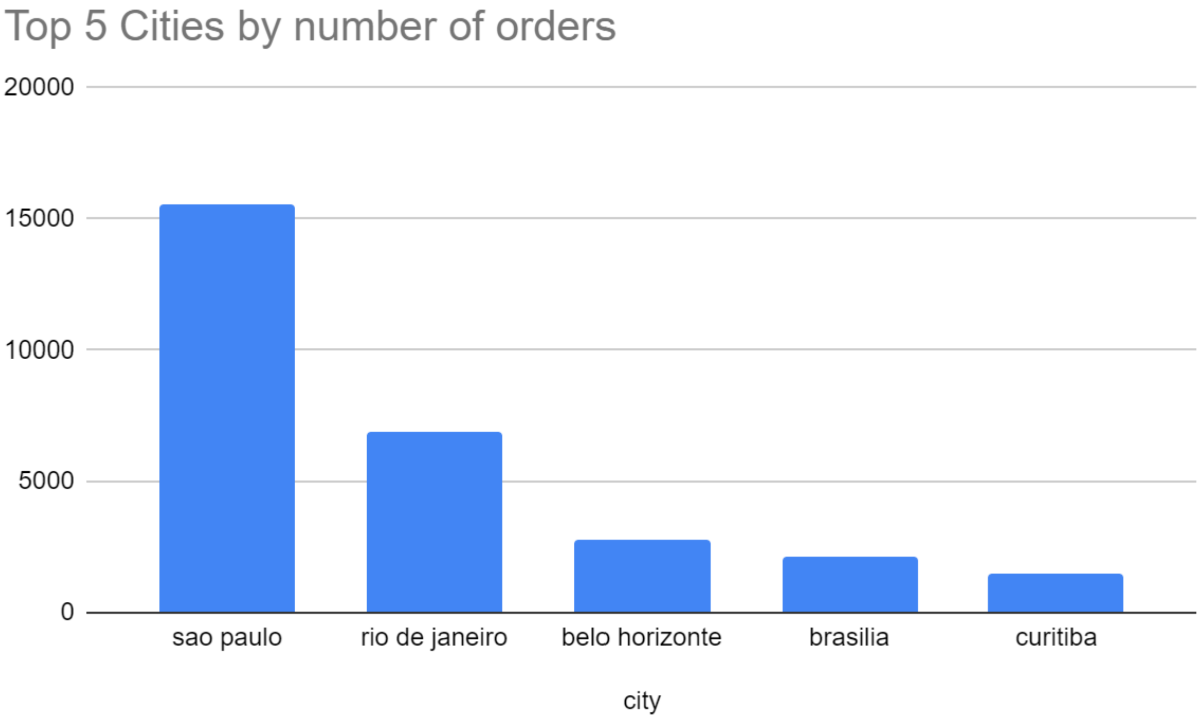


Row	repeating_customers_unique_id
1	00172711b30d52eea8b313a7f2cced02
2	004288347e5e88a27ded2bb23747066c
3	004b45ec5c64187465168251cd1c9c2f
4	0058f300f57d7b93c477a131a59b36c3
5	00a39521eb40f7012db50455bf083460
6	00cc12a6d8b578b8ebd21ea4e2ae8b27
7	011575986092c30523ecb71ff10cb473
8	011b4adcd54683b480c4d841250a987f
9	012452d40dafae4df401bced74cdb490
10	012a218df8995d3ec3bb221828360c86
11	013ef03e0f3f408dd9bf555e4edcdc0a
12	013f4353d26bb05dc6652f1269458d8d
13	015557c9912277312b9073947804a7ba
14	0178b244a5c281fb2ade54038dd4b161

Q.6 Top 5 cities in terms of the number of orders



```
  -- Get a list of the top 5 cities in terms of number of orders
SELECT
  c.customer_city AS city,
  COUNT(o.order_id) AS n_orders
FROM
  `jrjames83-1171.sampledata.orders` AS o
JOIN
  `jrjames83-1171.sampledata.customers` AS c
ON
  c.customer_id = o.customer_id
GROUP BY
  city
ORDER BY
  n_orders DESC
LIMIT
  5
```

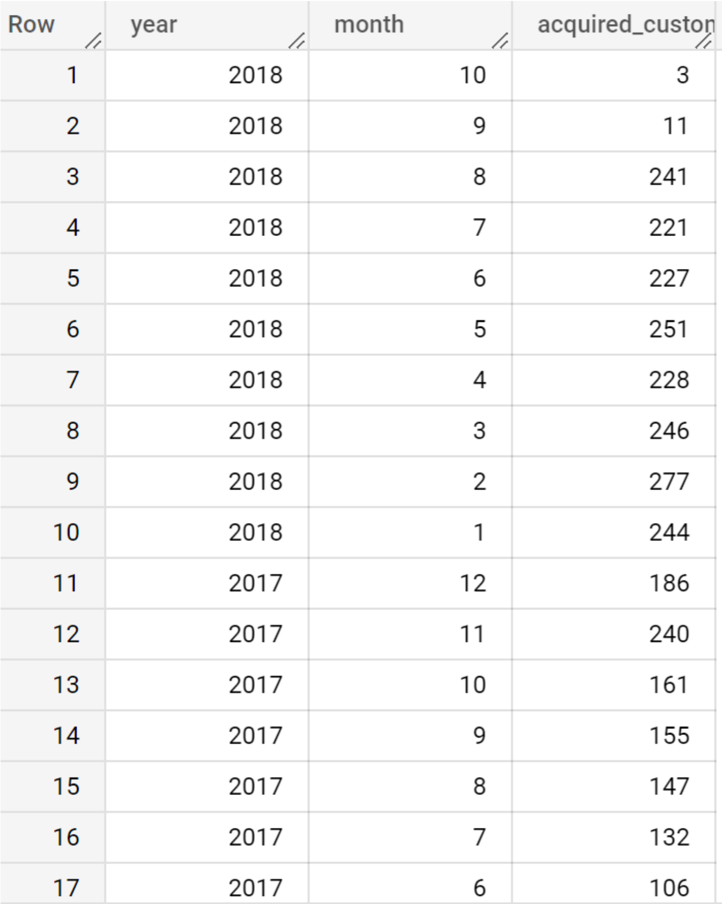


Q.7 How many customers were acquired each month and year?

```
-- Number of customers acqired by month and year
```

```
WITH
  t AS (
  SELECT
    c.customer_unique_id AS customer,
    o.order_purchase_timestamp AS purchase_date,
    ROW_NUMBER() OVER(PARTITION BY c.customer_unique_id ORDER BY order_purchase_timestamp)
AS order_number
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.customers` AS c
  ON
    c.customer_id = o.customer_id
  ORDER BY
    1,
    2,
    3)
SELECT
  EXTRACT( YEAR
  FROM
    t.purchase_date) AS year,
  EXTRACT( MONTH
  FROM
    t.purchase_date) AS month,
  COUNT(*) AS acquired_customers
FROM
  t
WHERE
  order_number > 1
GROUP BY
  year,
  month
ORDER BY
  year DESC,
  month DESC
```

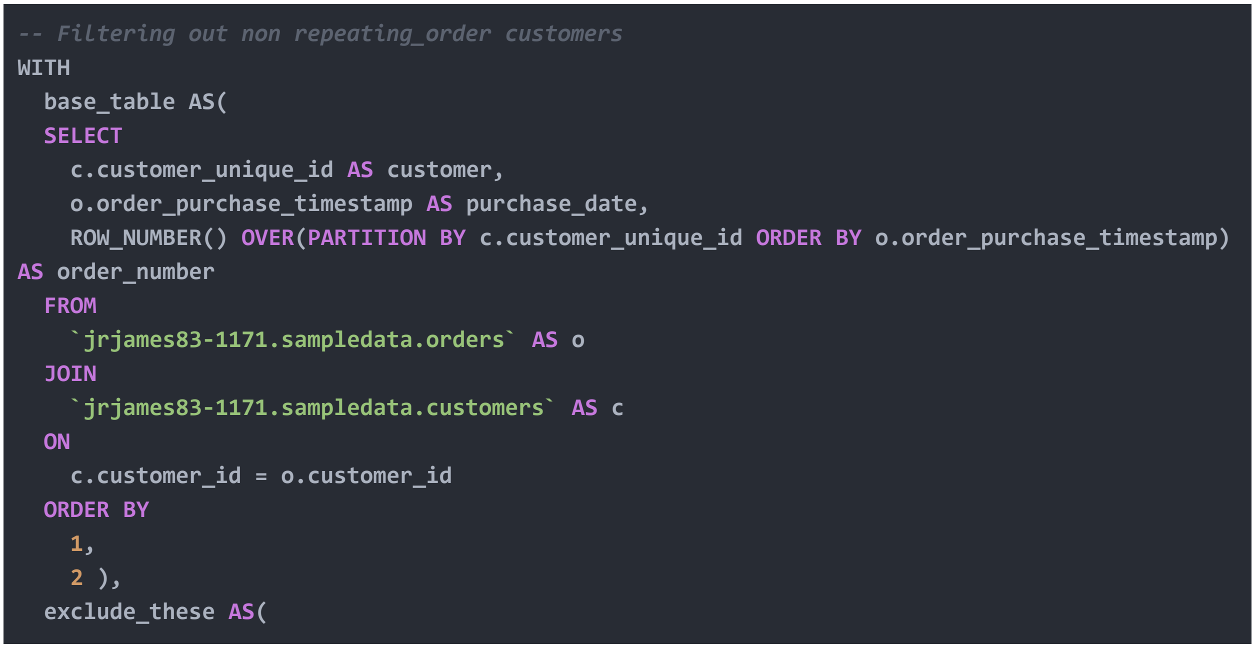
Sample Output:



Row	year	month	acquired_custon
1	2018	10	3
2	2018	9	11
3	2018	8	241
4	2018	7	221
5	2018	6	227
6	2018	5	251
7	2018	4	228
8	2018	3	246
9	2018	2	277
10	2018	1	244
11	2017	12	186
12	2017	11	240
13	2017	10	161
14	2017	9	155
15	2017	8	147
16	2017	7	132
17	2017	6	106

Q.8 How can we filter the table for the customer who made only one order?

The interpretation of these customers is MAX(order_number) = 1

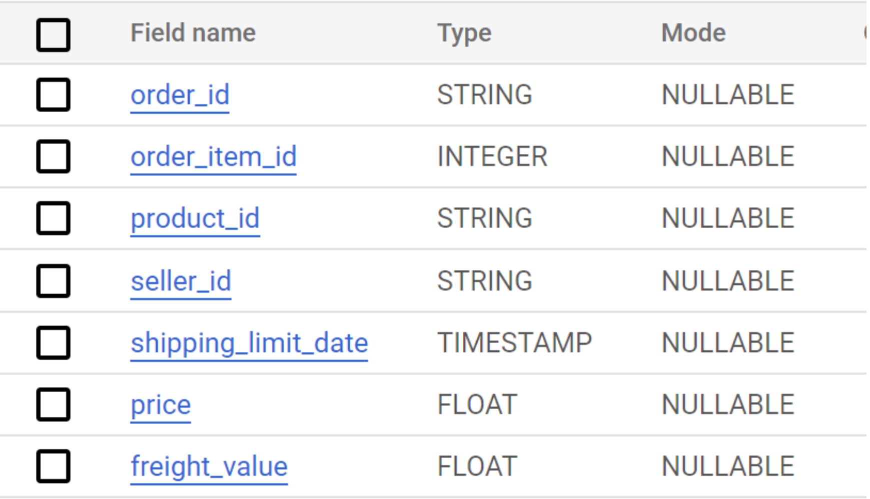


```
-- Filtering out non repeating_order customers
WITH
  base_table AS(
  SELECT
    c.customer_unique_id AS customer,
    o.order_purchase_timestamp AS purchase_date,
    ROW_NUMBER() OVER(PARTITION BY c.customer_unique_id ORDER BY o.order_purchase_timestamp)
AS order_number
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.customers` AS c
  ON
    c.customer_id = o.customer_id
  ORDER BY
    1,
    2 ),
  exclude_these AS(
  SELECT
    customer,
    MAX(order_number)
  FROM
    base_table
  GROUP BY
    1
  HAVING
    MAX(order_number) = 1 )
SELECT
  *
FROM
  base_table
WHERE
  base_table.customer NOT IN (
  SELECT
    customer
  FROM
    exclude_these)
ORDER BY
  1,
  3
```

Question: Can we do it using EXCEPT set operator?

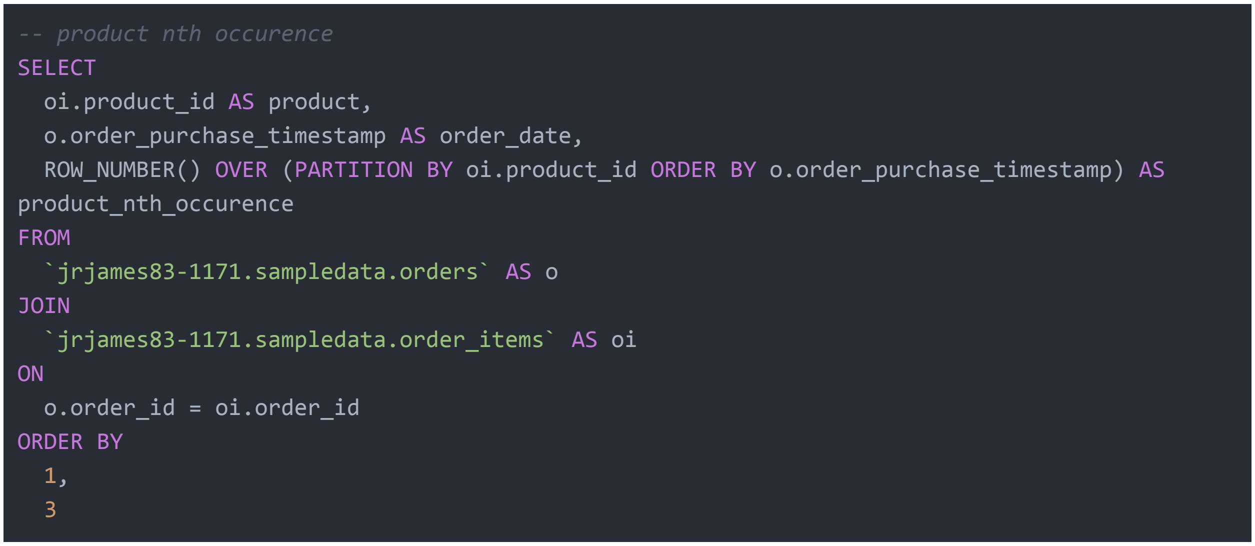
Q.9 When was the first time a product got ordered multiple times?

In order to know when it was the first time a specific product was ordered with an order quantity of more than 1, we need to use another table order_items.

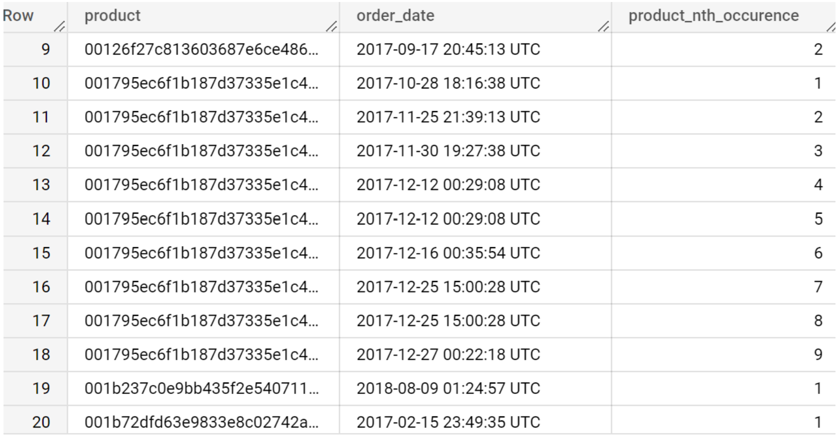


□	Field name	Type	Mode
□	order_id	STRING	NULLABLE
□	order_item_id	INTEGER	NULLABLE
□	product_id	STRING	NULLABLE
□	seller_id	STRING	NULLABLE
□	shipping_limit_date	TIMESTAMP	NULLABLE
□	price	FLOAT	NULLABLE
□	freight_value	FLOAT	NULLABLE

So, each order is associated with a record in that table. Order_item_id is the id for the items included in that order and it goes up sequentially from 1 to n items per order.



```
-- product nth occurence
SELECT
  oi.product_id AS product,
  o.order_purchase_timestamp AS order_date,
  ROW_NUMBER() OVER (PARTITION BY oi.product_id ORDER BY o.order_purchase_timestamp) AS
product_nth_occurence
FROM
  `jrjames83-1171.sampledata.orders` AS o
JOIN
  `jrjames83-1171.sampledata.order_items` AS oi
ON
  o.order_id = oi.order_id
ORDER BY
  1,
  3
```

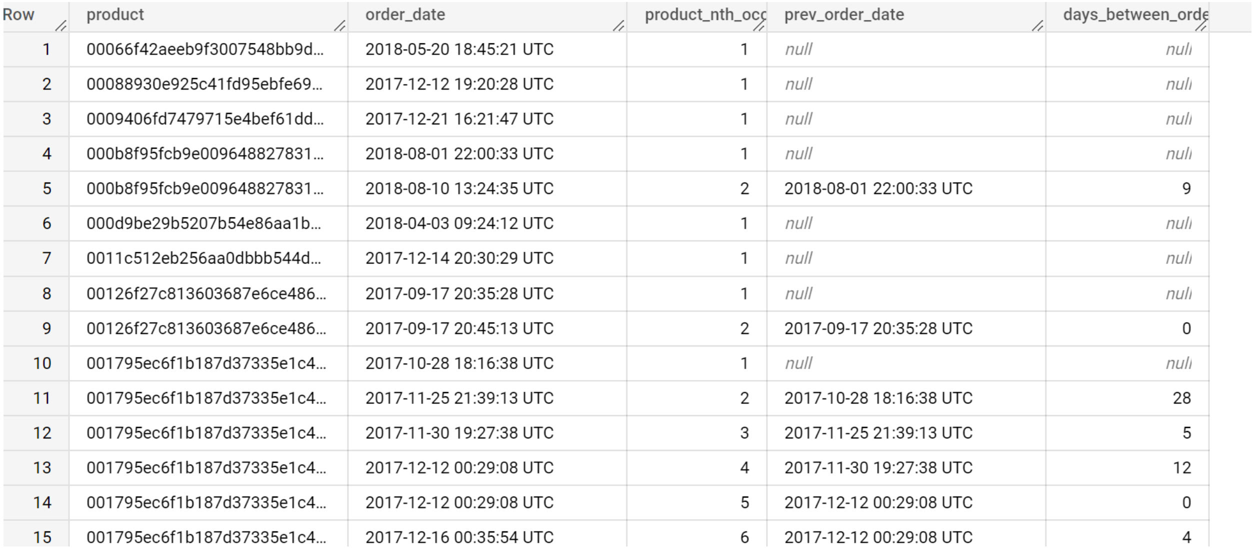


Row	product	order_date	product_nth_occurence
9	00126f27c813603687e6ce486...	2017-09-17 20:45:13 UTC	2
10	001795ec6f1b187d37335e1c4...	2017-10-28 18:16:38 UTC	1
11	001795ec6f1b187d37335e1c4...	2017-11-25 21:39:13 UTC	2
12	001795ec6f1b187d37335e1c4...	2017-11-30 19:27:38 UTC	3
13	001795ec6f1b187d37335e1c4...	2017-12-12 00:29:08 UTC	4
14	001795ec6f1b187d37335e1c4...	2017-12-12 00:29:08 UTC	5
15	001795ec6f1b187d37335e1c4...	2017-12-16 00:35:54 UTC	6
16	001795ec6f1b187d37335e1c4...	2017-12-25 15:00:28 UTC	7
17	001795ec6f1b187d37335e1c4...	2017-12-25 15:00:28 UTC	8
18	001795ec6f1b187d37335e1c4...	2017-12-27 00:22:18 UTC	9
19	001b237c0e9bb435f2e540711...	2018-08-09 01:24:57 UTC	1
20	001b72dfd63e9833e8c02742a...	2017-02-15 23:49:35 UTC	1

Q10. Days between orders for a specific product

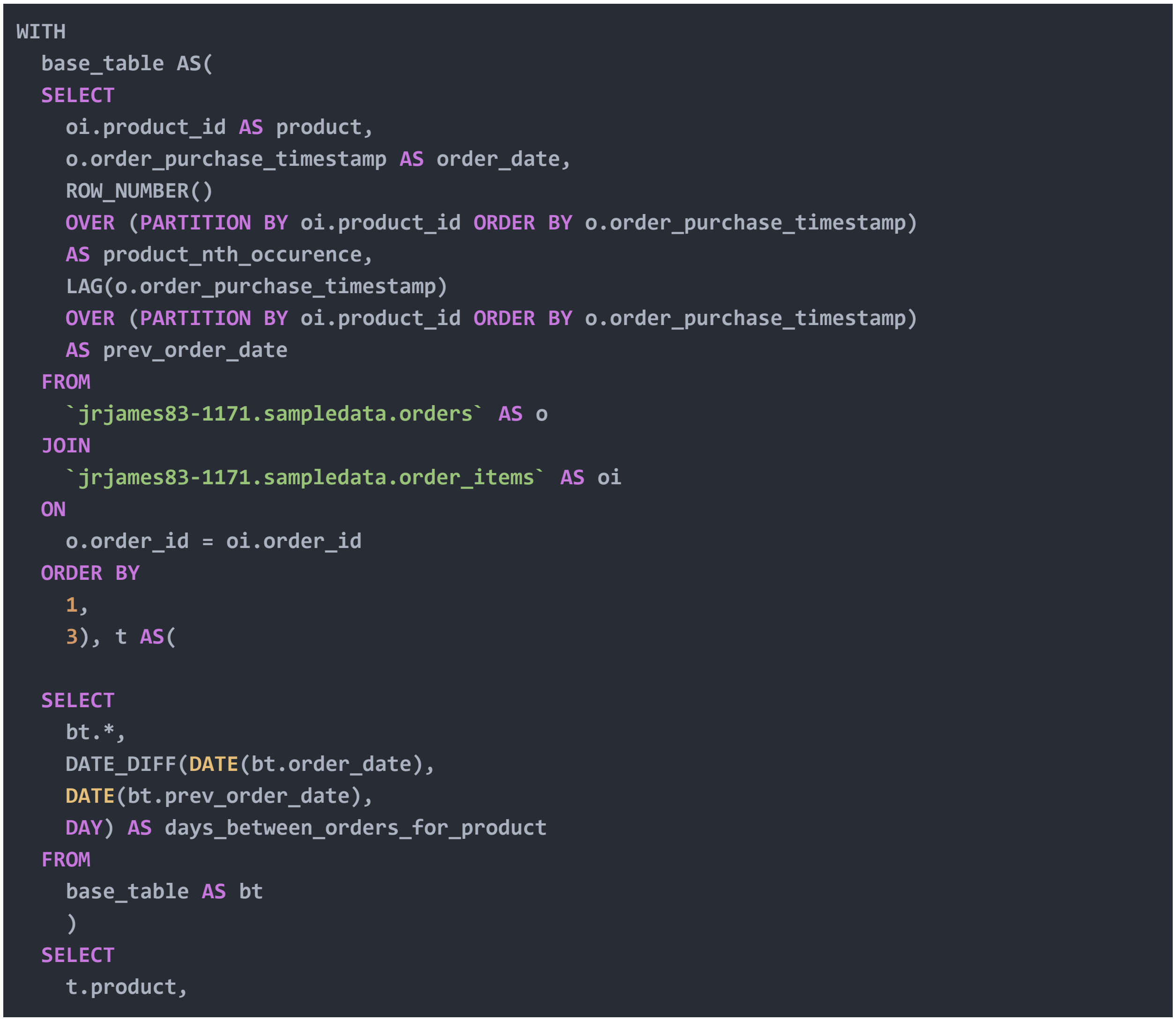
```
WITH
  base_table AS(
  SELECT
    oi.product_id AS product,
    o.order_purchase_timestamp AS order_date,
    ROW_NUMBER()
    OVER (PARTITION BY oi.product_id ORDER BY o.order_purchase_timestamp)
    AS product_nth_occurence,
    LAG(o.order_purchase_timestamp)
    OVER (PARTITION BY oi.product_id ORDER BY o.order_purchase_timestamp)
    AS prev_order_date
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.order_items` AS oi
  ON
    o.order_id = oi.order_id
  ORDER BY
    1,
    3)

SELECT
  bt.*,
  DATE_DIFF(DATE(bt.order_date),
  DATE(bt.prev_order_date),
  DAY) AS days_between_orders_for_product
FROM
  base_table AS bt
```

Sample Output:



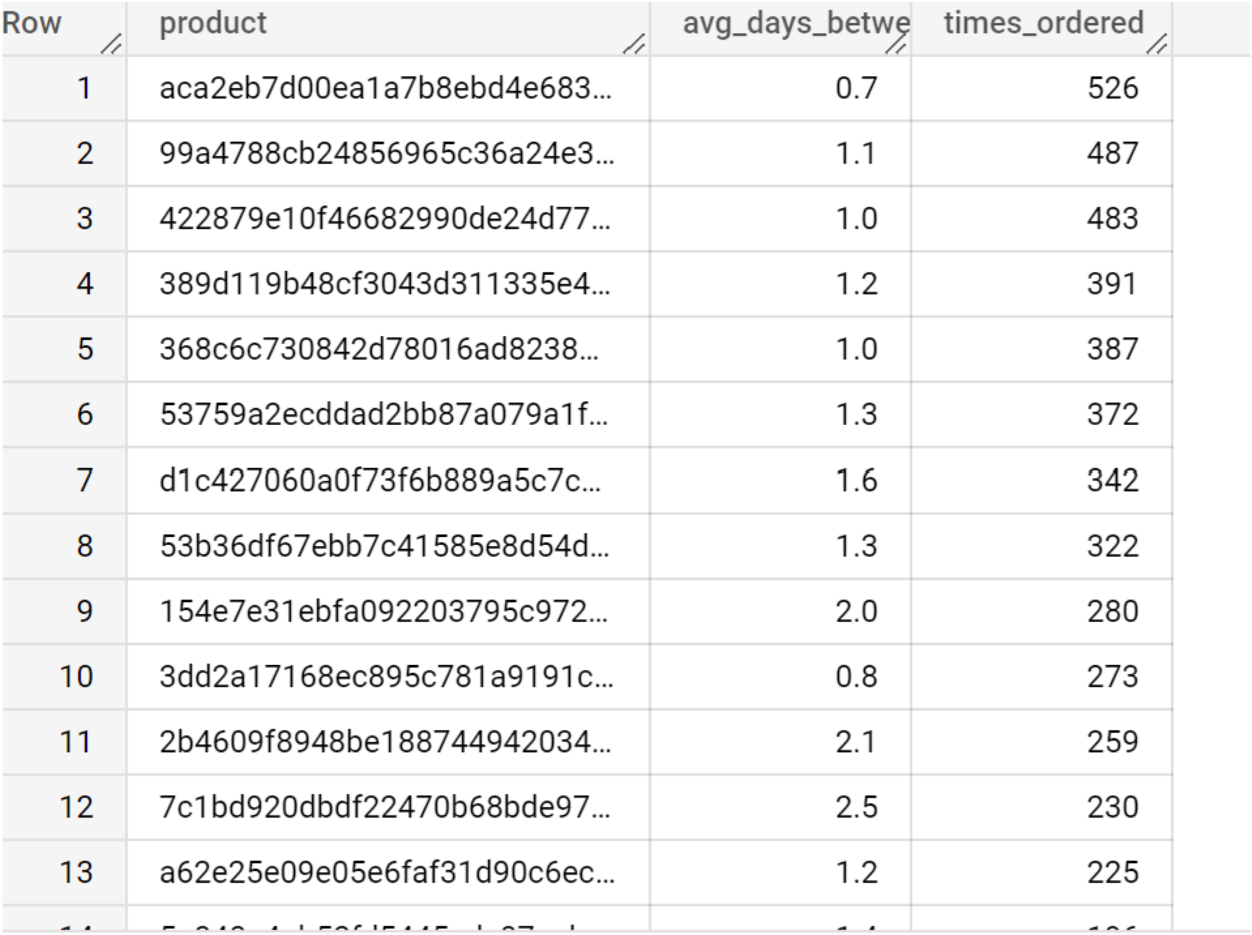
Row	product	order_date	product_nth_occ	prev_order_date	days_between_orde
1	00066f42aeeb9f3007548bb9d…	2018-05-20 18:45:21 UTC	1	null	null
2	00088930e925c41fd95ebfe69…	2017-12-12 19:20:28 UTC	1	null	null
3	0009406fd7479715e4bef61dd…	2017-12-21 16:21:47 UTC	1	null	null
4	000b8f95fcb9e009648827831…	2018-08-01 22:00:33 UTC	1	null	null
5	000b8f95fcb9e009648827831…	2018-08-10 13:24:35 UTC	2	2018-08-01 22:00:33 UTC	9
6	000d9be29b5207b54e86aa1b…	2018-04-03 09:24:12 UTC	1	null	null
7	0011c512eb256aa0dbbb544d…	2017-12-14 20:30:29 UTC	1	null	null
8	00126f27c813603687e6ce486…	2017-09-17 20:35:28 UTC	1	null	null
9	00126f27c813603687e6ce486…	2017-09-17 20:45:13 UTC	2	2017-09-17 20:35:28 UTC	0
10	001795ec6f1b187d37335e1c4…	2017-10-28 18:16:38 UTC	1	null	null
11	001795ec6f1b187d37335e1c4…	2017-11-25 21:39:13 UTC	2	2017-10-28 18:16:38 UTC	28
12	001795ec6f1b187d37335e1c4…	2017-11-30 19:27:38 UTC	3	2017-11-25 21:39:13 UTC	5
13	001795ec6f1b187d37335e1c4…	2017-12-12 00:29:08 UTC	4	2017-11-30 19:27:38 UTC	12
14	001795ec6f1b187d37335e1c4…	2017-12-12 00:29:08 UTC	5	2017-12-12 00:29:08 UTC	0
15	001795ec6f1b187d37335e1c4…	2017-12-16 00:35:54 UTC	6	2017-12-12 00:29:08 UTC	4

Q.11 Find the average days between orders for a specific product



```
WITH
  base_table AS(
  SELECT
    oi.product_id AS product,
    o.order_purchase_timestamp AS order_date,
    ROW_NUMBER()
    OVER (PARTITION BY oi.product_id ORDER BY o.order_purchase_timestamp)
    AS product_nth_occurence,
    LAG(o.order_purchase_timestamp)
    OVER (PARTITION BY oi.product_id ORDER BY o.order_purchase_timestamp)
    AS prev_order_date
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.order_items` AS oi
  ON
    o.order_id = oi.order_id
  ORDER BY
    1,
    3), t AS(

  SELECT
    bt.*,
    DATE_DIFF(DATE(bt.order_date),
    DATE(bt.prev_order_date),
    DAY) AS days_between_orders_for_product
  FROM
    base_table AS bt
    )
  SELECT
    t.product,
    ROUND(AVG(t.days_between_orders_for_product), 1)
     AS avg_days_between_orders,
    COUNT(*) AS times_ordered
  FROM t
  WHERE t.days_between_orders_for_product IS NOT NULL
  GROUP BY 1
  ORDER BY 3 DESC
```

Sample Output:

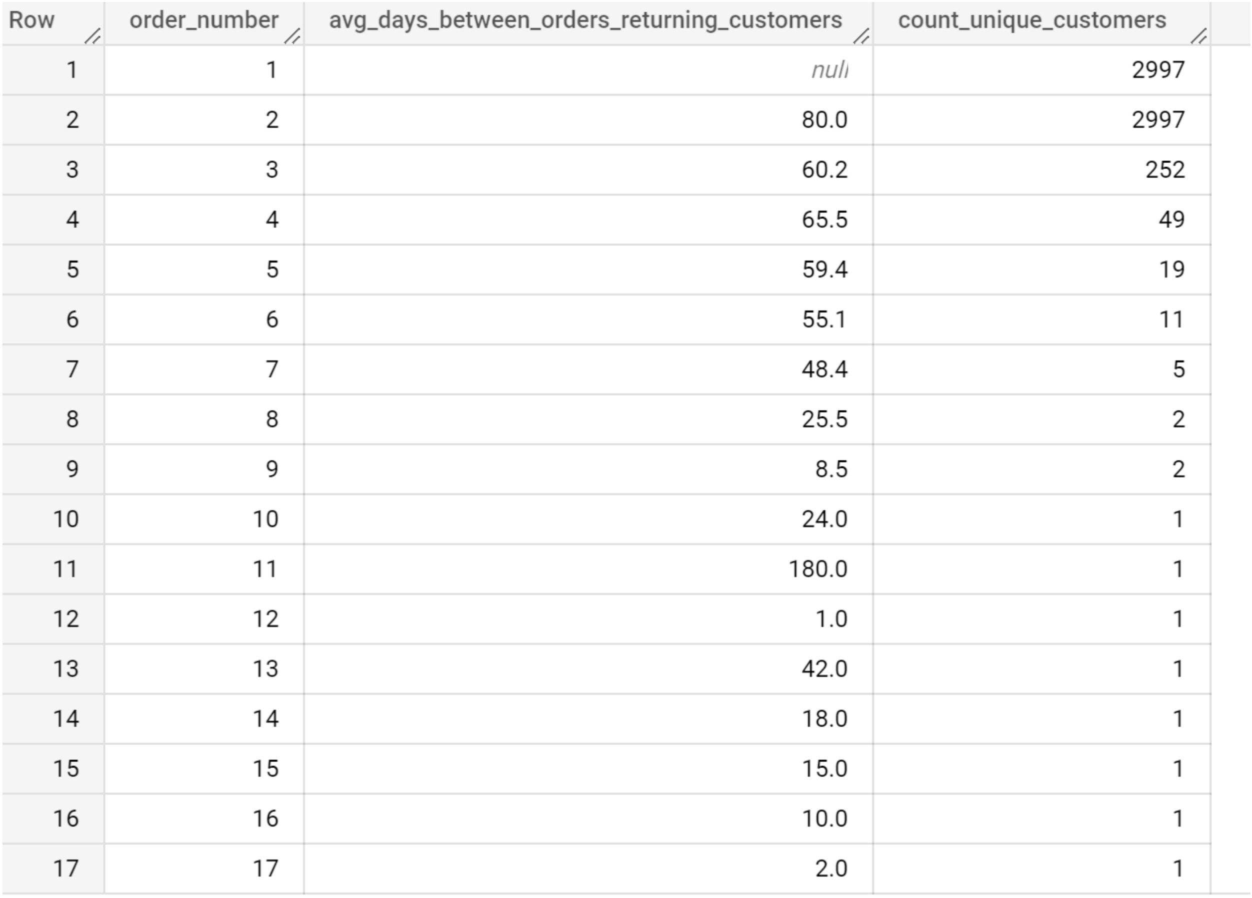


Row	product	avg_days_betwe	times_ordered
1	aca2eb7d00ea1a7b8ebd4e683...	0.7	526
2	99a4788cb24856965c36a24e3...	1.1	487
3	422879e10f46682990de24d77...	1.0	483
4	389d119b48cf3043d311335e4...	1.2	391
5	368c6c730842d78016ad8238...	1.0	387
6	53759a2ecddad2bb87a079a1f...	1.3	372
7	d1c427060a0f73f6b889a5c7c...	1.6	342
8	53b36df67ebb7c41585e8d54d...	1.3	322
9	154e7e31ebfa092203795c972...	2.0	280
10	3dd2a17168ec895c781a9191c...	0.8	273
11	2b4609f8948be188744942034...	2.1	259
12	7c1bd920dbdf22470b68bde97...	2.5	230
13	a62e25e09e05e6faf31d90c6ec...	1.2	225

Q.12 Average Days Between orders for returning customers:

```
WITH
  base_table AS
  (
  SELECT
    c.customer_unique_id AS customer,
    o.order_purchase_timestamp AS purchase_date,
    ROW_NUMBER() OVER(PARTITION BY c.customer_unique_id ORDER BY o.order_purchase_timestamp)
AS order_number,
    LAG(o.order_purchase_timestamp) OVER (PARTITION BY c.customer_unique_id
          ORDER BY o.order_purchase_timestamp) AS prev_customer_order_date
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.customers` AS c
  ON
    c.customer_id = o.customer_id
  ORDER BY
    1,
    2 )
    , exclude_these AS(
  SELECT
    customer,
    MAX(order_number)
  FROM
    base_table
  GROUP BY
    1
  HAVING
    MAX(order_number) = 1

    )
SELECT
  bt.order_number,
  ROUND(AVG(DATE_DIFF(bt.purchase_date, prev_customer_order_date, DAY)), 1) AS
avg_days_between_orders_returning_customers,
  COUNT(DISTINCT bt.customer) AS count_unique_customers
FROM
  base_table AS bt
WHERE
  bt.customer NOT IN (
  SELECT
    customer
  FROM
    exclude_these)
GROUP BY 1
ORDER BY 1
```

Sample Output:



Row	order_number	avg_days_between_orders_returning_customers	count_unique_customers
1	1	null	2997
2	2	80.0	2997
3	3	60.2	252
4	4	65.5	49
5	5	59.4	19
6	6	55.1	11
7	7	48.4	5
8	8	25.5	2
9	9	8.5	2
10	10	24.0	1
11	11	180.0	1
12	12	1.0	1
13	13	42.0	1
14	14	18.0	1
15	15	15.0	1
16	16	10.0	1
17	17	2.0	1

Q.13 Get the Hourly Revenue profile

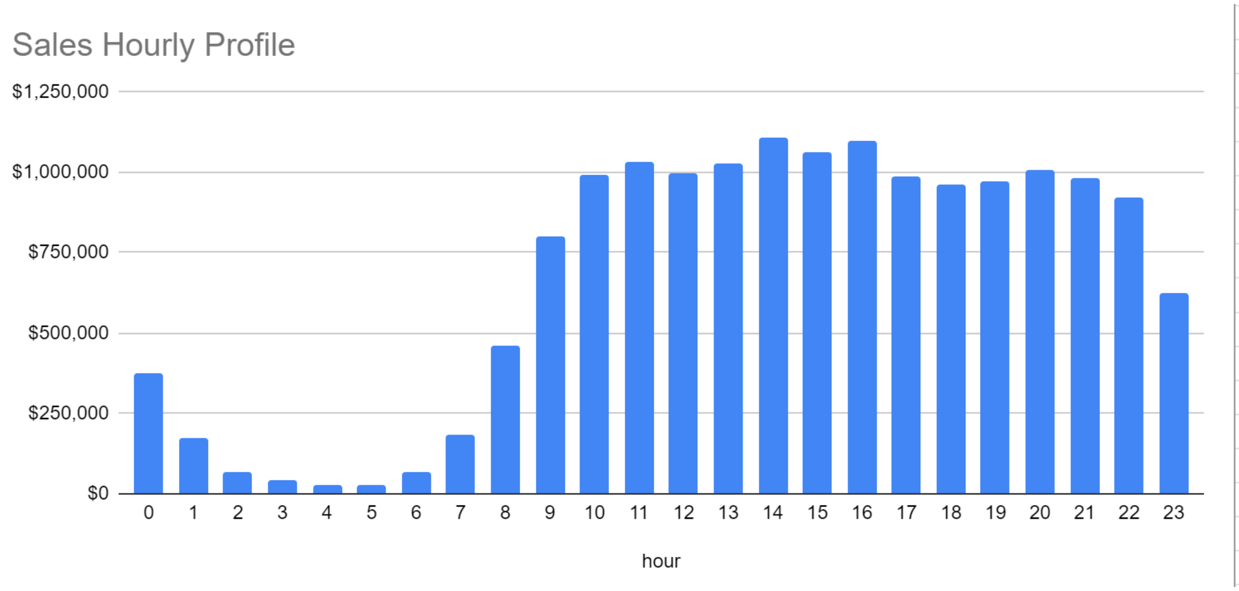
When we talk about hourly revenue profile, it means we mean the aggregated (summed up over all days for each hour) revenue → we shall use EXTRACT() instead of DATE_TRUN()

Aggregated Revenue Profile:

```
-- Hourly Aggregated Revenue Profile
SELECT
  EXTRACT( HOUR
  FROM
    o.order_purchase_timestamp) AS hour,
  ROUND(SUM(op.payment_value), 2) AS sales,
FROM
  `jrjames83-1171.sampledata.orders` AS o
JOIN
  `jrjames83-1171.sampledata.order_payments` AS op
ON
  op.order_id = o.order_id
```

```
GROUP BY
  1
ORDER BY
  1
```

Sample Output:

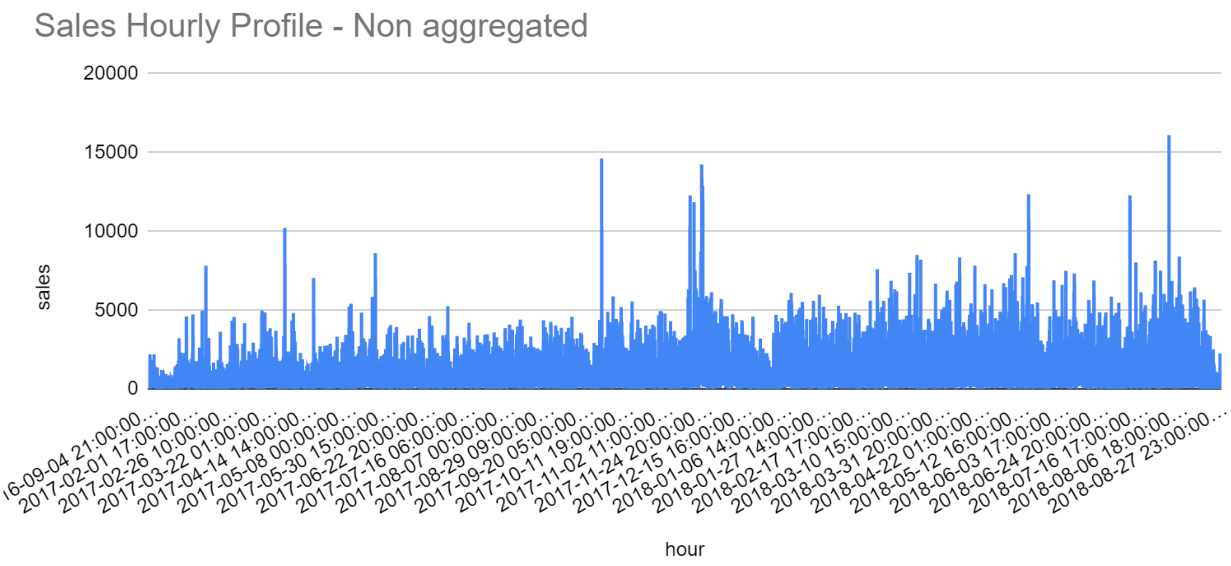


Non-aggregated Hourly series:

This one can be done using DATE_TRUNC() function.

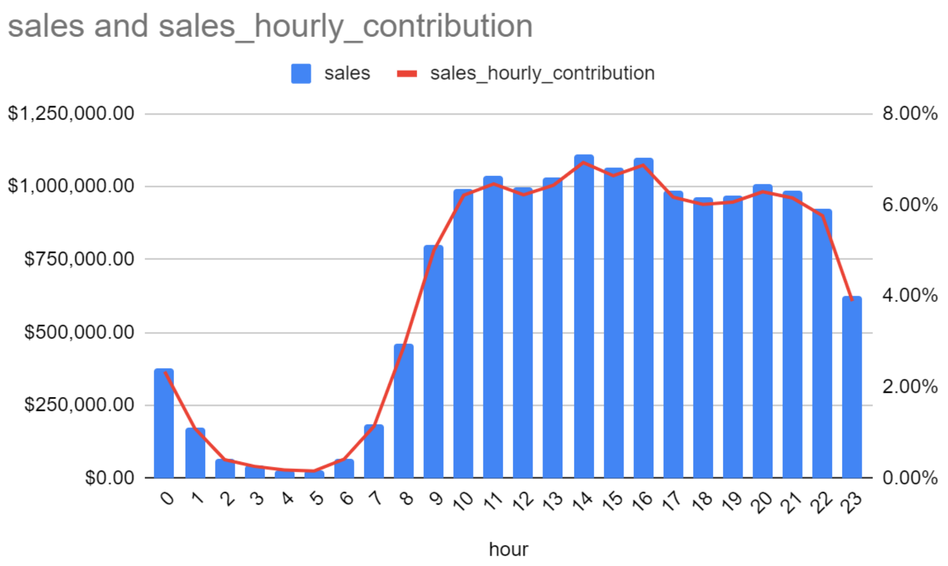
```
-- Hourly Non-Aggregated Revenue Profile
```

```
SELECT
  DATE_TRUNC(o.order_purchase_timestamp, HOUR) AS hour,
  ROUND(SUM(op.payment_value), 2) AS sales,
FROM
  `jrjames83-1171.sampledata.orders` AS o
JOIN
  `jrjames83-1171.sampledata.order_payments` AS op
ON
  op.order_id = o.order_id
GROUP BY
  1
ORDER BY
  1
```

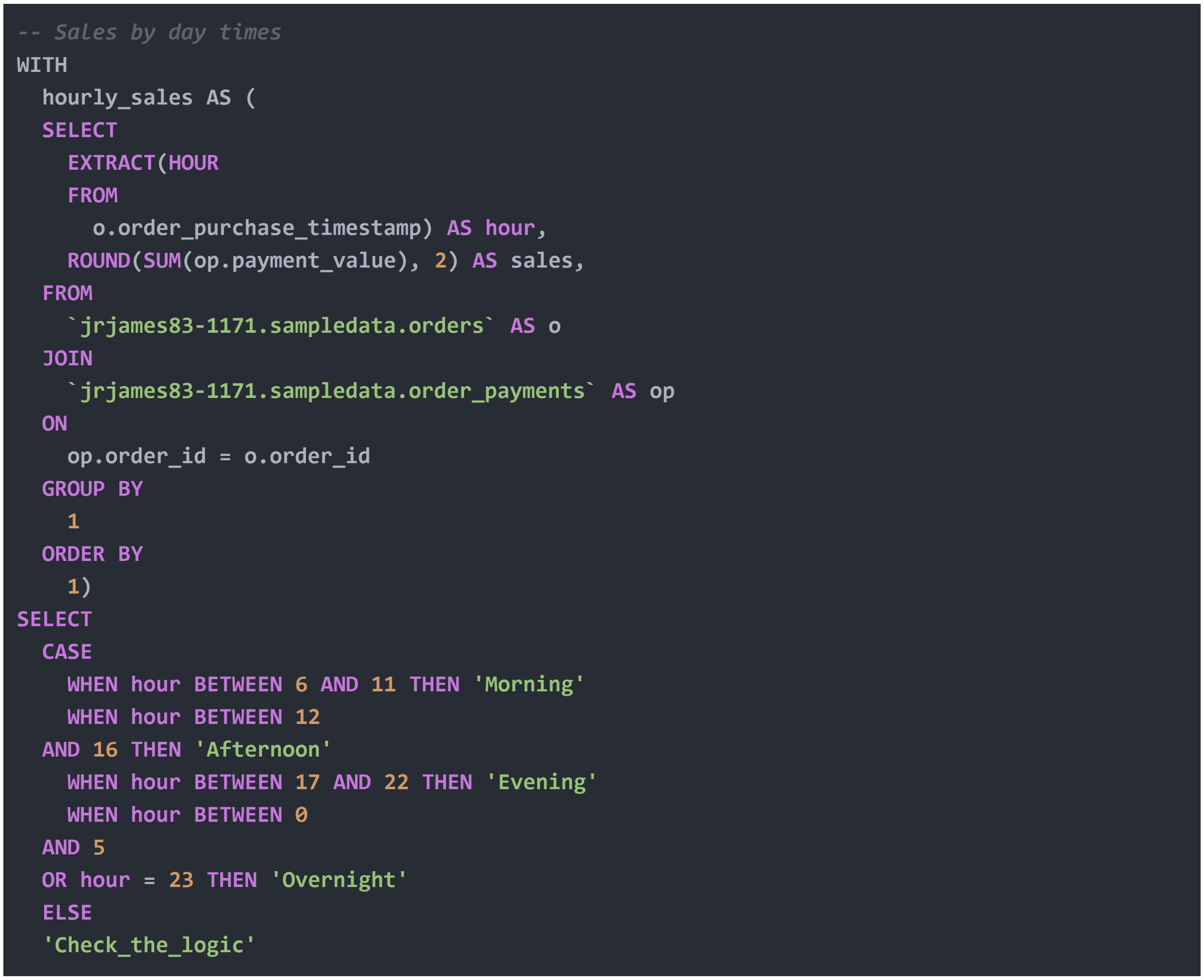


Now, let’s add a new variable to show the percentage of the hourly traffic from the total daily traffic.

```sql
-- Sales Hourly contribution of the daily sales.
SELECT
  EXTRACT(HOUR
  FROM
    o.order_purchase_timestamp) AS hour,
  ROUND(SUM(op.payment_value), 2) AS sales,
  ROUND(SUM(op.payment_value) / SUM(SUM(op.payment_value)) OVER () * 100, 2) || '%' AS
sales_hourly_contribution,
FROM
  `jrjames83-1171.sampledata.orders` AS o
JOIN
  `jrjames83-1171.sampledata.order_payments` AS op
ON
  op.order_id = o.order_id
GROUP BY
  1
ORDER BY
  1
```



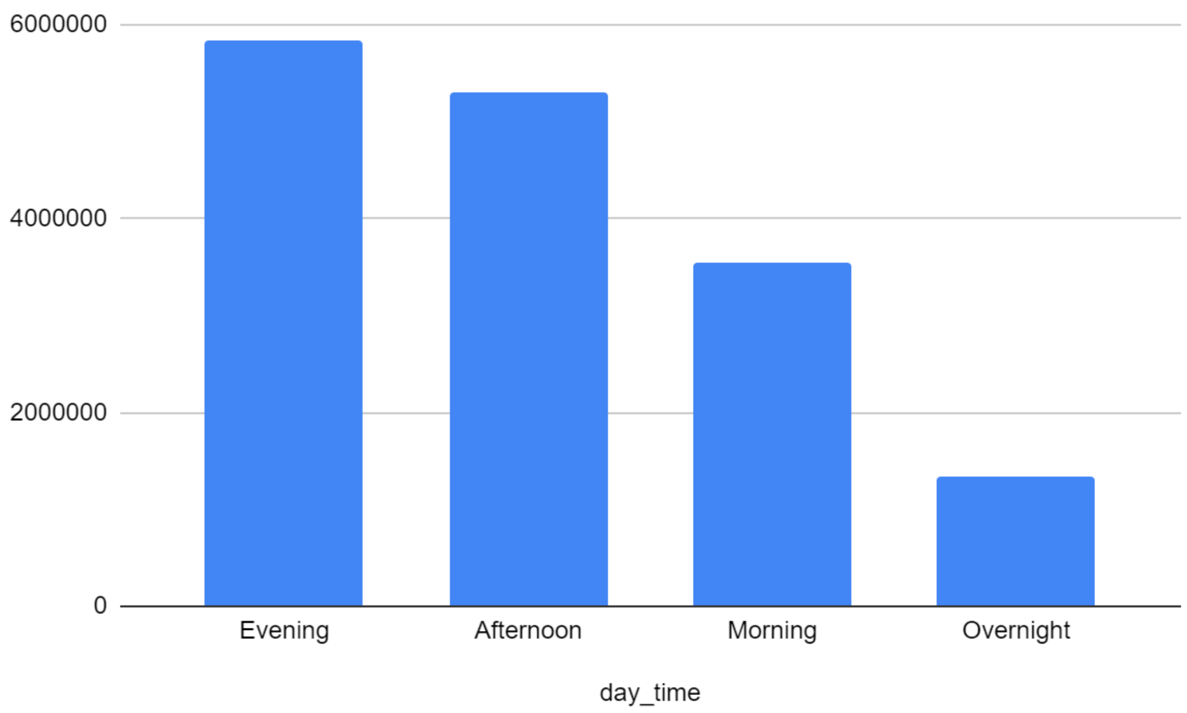
Q14. Sales by Day Times:



```
-- Sales by day times
WITH
  hourly_sales AS (
  SELECT
    EXTRACT(HOUR
    FROM
      o.order_purchase_timestamp) AS hour,
    ROUND(SUM(op.payment_value), 2) AS sales,
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.order_payments` AS op
  ON
    op.order_id = o.order_id
  GROUP BY
    1
  ORDER BY
    1)
SELECT
  CASE
    WHEN hour BETWEEN 6 AND 11 THEN 'Morning'
    WHEN hour BETWEEN 12
  AND 16 THEN 'Afternoon'
    WHEN hour BETWEEN 17 AND 22 THEN 'Evening'
    WHEN hour BETWEEN 0
  AND 5
  OR hour = 23 THEN 'Overnight'
  ELSE
  'Check_the_logic'
```

```
END
  AS day_time,
  SUM(sales) AS total_sales
FROM
  hourly_sales
GROUP BY
  1
```

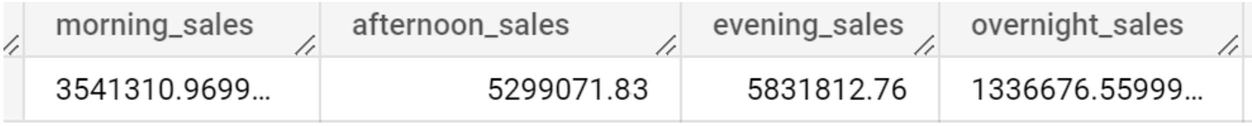
Sample Output:



Another way of using CASE to simulate SUMIF() function in Excel

```
-- Alternative CASE statement for simulating SUMIF()
WITH
  hourly_sales AS (
  SELECT
    EXTRACT(HOUR
    FROM
      o.order_purchase_timestamp) AS hour,
    ROUND(SUM(op.payment_value), 2) AS sales,
  FROM
    `jrjames83-1171.sampledata.orders` AS o
  JOIN
    `jrjames83-1171.sampledata.order_payments` AS op
  ON
    op.order_id = o.order_id
  GROUP BY
    1
  ORDER BY
    1)

SELECT
  SUM(CASE WHEN hour BETWEEN 6 AND 11 THEN sales ELSE 0 END) AS morning_sales,
  SUM(CASE WHEN hour BETWEEN 12 AND 16 THEN sales ELSE 0 END) AS afternoon_sales,
  SUM(CASE WHEN hour BETWEEN 17 AND 22 THEN sales ELSE 0 END) AS evening_sales,
  SUM(CASE WHEN hour BETWEEN 0 AND 5 OR hour = 23 THEN sales ELSE 0 END) AS overnight_sales

FROM
  hourly_sales
```

Sample Output:



morning_sales	afternoon_sales	evening_sales	overnight_sales
3541310.9699...	5299071.83	5831812.76	1336676.55999...