

# **Target SQL Business Case Study**

**Presented By:**

Saisree Ekkaldevi

# Introduction

## About Target

Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.

This particular business case focuses on the operations of Target in Brazil and provides insightful information about 100,000 orders placed between 2016 and 2018. The dataset offers a comprehensive view of various dimensions including the order status, price, payment and freight performance, customer location, product attributes, and customer reviews.

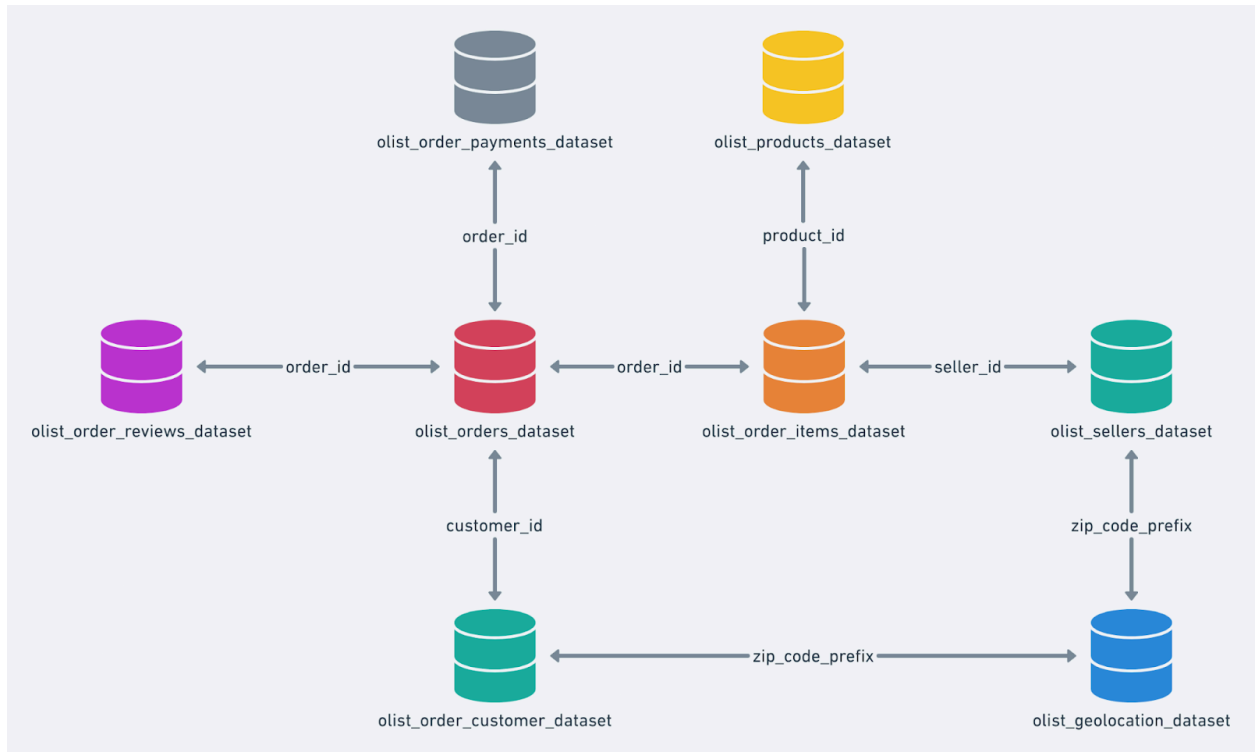
By analyzing this extensive dataset, it becomes possible to gain valuable insights into Target's operations in Brazil. The information can shed light on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency, customer demographics, product characteristics, and customer satisfaction levels.

## About the Dataset

The dataset is structured into multiple related tables, capturing customers, orders, payments, products, sellers, reviews, and geolocation data. Each table provides unique insights into a different dimension of the business.

- **Customers Table**
  - customer\_id: ID of the consumer
  - customer\_unique\_id: Unique customer identifier
  - customer\_zip\_code\_prefix: Location zip code
  - customer\_city: City name
  - customer\_state: State code (e.g., São Paulo – SP)
- **Sellers Table**
  - seller\_id: Unique seller ID
  - seller\_zip\_code\_prefix: Seller location
  - seller\_city: City name
  - seller\_state: State code
- **Orders Table**
  - order\_id: Unique order identifier
  - customer\_id: Customer reference
  - order\_status: Delivered, shipped, canceled, etc.
  - order\_purchase\_timestamp: Time of purchase
  - order\_delivered\_customer\_date: Actual delivery date

- order\_estimated\_delivery\_date: Estimated delivery date
- **Order Items Table**
  - order\_id
  - order\_item\_id
  - product\_id
  - seller\_id
  - shipping\_limit\_date
  - price
  - freight\_value
- **Geolocation Table**
  - geolocation\_zip\_code\_prefix
  - geolocation\_lat
  - geolocation\_lng
  - geolocation\_city
  - Geolocation\_state
- **Payments Table**
  - order\_id: Order reference
  - payment\_type: Mode of payment (credit card, boleto, etc.)
  - payment\_installments: Number of installments (for EMI)
  - payment\_value: Total amount paid
- **Reviews Table**
  - review\_id
  - order\_id
  - review\_score: Rating (1–5)
  - review\_comment\_title & review\_comment\_message
  - review\_creation\_date
  - review\_answer\_timestamp
- **Products Table**
  - product\_id
  - product\_category\_name
  - product\_weight\_g
  - product\_length\_cm
  - product\_height\_cm
  - product\_width\_cm
  - product\_photos\_qty
  - product\_description\_length
- **ER Diagram**



*Figure 1: Entity Relationship (ER) Diagram of Target Brazil Dataset.*

# Initial Exploration

## 1. Data type of all columns in the “customers” table.

Screenshot:

customers

Query

Open in ▾

Share ▾

Copy

Snapshot

Delete

Export

Refresh

Schema

Details

Preview

Table Explorer

Preview

Insights

Lineage

Data Profile

Data Quality

Filter

Enter property name or value

?

<input type="checkbox"/>	Field name	Type	Mode	Key	Collation	Default Value	Policy Tags ?	Data Policies	Des
<input type="checkbox"/>	customer_id	STRING	NULLABLE	-	-	-	-	-	-
<input type="checkbox"/>	customer_unique_id	STRING	NULLABLE	-	-	-	-	-	-
<input type="checkbox"/>	customer_zip_code_prefix	INTEGER	NULLABLE	-	-	-	-	-	-
<input type="checkbox"/>	customer_city	STRING	NULLABLE	-	-	-	-	-	-
<input type="checkbox"/>	customer_state	STRING	NULLABLE	-	-	-	-	-	-

Edit schema

View row access policies

### Insight:

The “customers” table contains five columns. Most attributes are STRING type, except customer\_zip\_code\_prefix which is INTEGER. All columns are NULLABLE, suggesting that missing values are possible.

- customer\_id - STRING
- customer\_unique\_id - STRING
- customer\_zip\_code\_prefix - INTEGER
- customer\_city - STRING
- customer\_state - STRING

## 2. Get the time range between which the orders were placed.

Query:

```
SELECT MIN(order_purchase_timestamp) AS start_date, MAX(order_purchase_timestamp) AS end_date
FROM `Target.orders`
```

Screenshot:

The screenshot shows a web-based SQL interface. At the top, there are tabs for 'geolocation', 'customers', 'orders', and an active tab '\*Untitled...ery'. Below the tabs is a search bar and a toolbar with buttons: 'Run', 'Save', 'Download', 'Share', 'Schedule', 'Open in', and 'More'. The query editor contains the following SQL code:

```
1 SELECT MIN(order_purchase_timestamp) AS start_date, MAX(order_purchase_timestamp) AS end_date
2 FROM `Target.orders`
```

Below the query editor, a green checkmark indicates 'Query completed'. The 'Query results' section shows a table with the following data:

Job information	Results	Visualization	JSON	Execution details	Execution graph
Row	start_date	end_date			
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC			

At the bottom right, there is a pagination control showing 'Results per page: 50' and '1 - 1 of 1'.

Insight:

The dataset covers orders from September 2016 to August 2018.

### 3.Count the Cities & States of customers who ordered during the given period.

Query:

```
SELECT  
COUNT(DISTINCT(customer_city)) AS city_count,  
COUNT(DISTINCT(customer_state)) AS state_count  
FROM `Target.customers`;
```

Screenshot:

The screenshot shows a SQL query interface with a toolbar at the top containing buttons for Run, Save, Download, Share, Schedule, Open in, and More. The query editor contains the following SQL code:

```
1 SELECT COUNT(DISTINCT(customer_city)) AS city_count,COUNT(DISTINCT(customer_state)) AS state_count  
2 FROM `Target.customers`
```

Below the query editor, a status bar indicates: "This query will process 1.55 MB when run." The results section is titled "Query results" and includes a "Save results" button. The results are displayed in a table with the following structure:

Row	city_count	state_count
1	4119	27

At the bottom of the results section, there is a pagination control showing "Results per page: 50" and "1 - 1 of 1".

Insight:

4119 cities, 27 states

# In-depth Exploration

## 1. Is there a growing trend in the no. of orders places over the past years?

Query:

```
SELECT COUNT(order_id) AS Total_orders, EXTRACT(year FROM order_purchase_timestamp) AS Year  
  
FROM `Target.orders`  
  
GROUP BY Year  
  
ORDER BY Year
```

Screenshot:

The screenshot shows a SQL query editor interface. At the top, there are tabs for 'customers', 'orders', and an active tab for an 'Untitled query'. The query text is:   
1 SELECT COUNT(order\_id) AS Total\_orders, EXTRACT(year FROM order\_purchase\_timestamp) AS Year  
2 FROM `Target.orders`  
3 GROUP BY Year  
4 ORDER BY Year  
Below the query editor, a status bar indicates 'This query will process 3.98 MB when run.' Below that, the 'Query results' section is visible, showing a table with 3 rows of data. The table has columns 'Total\_orders' and 'Year'. The data shows a clear upward trend: 329 orders in 2016, 45,101 in 2017, and 54,011 in 2018. The interface also includes buttons for 'Run', 'Save', 'Download', 'Share', 'Schedule', 'Open in', and 'More'.

Row	Total_orders	Year
1	329	2016
2	45101	2017
3	54011	2018

Insight:

There is a clear growing trend in the number of orders. Orders increased sharply from 329 in 2016 to 45,101 in 2017, and continued to rise to 54,011 in 2018 (till August). This indicates strong year-over-year growth in Target's Brazilian e-commerce operations.



## 2. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

Query:

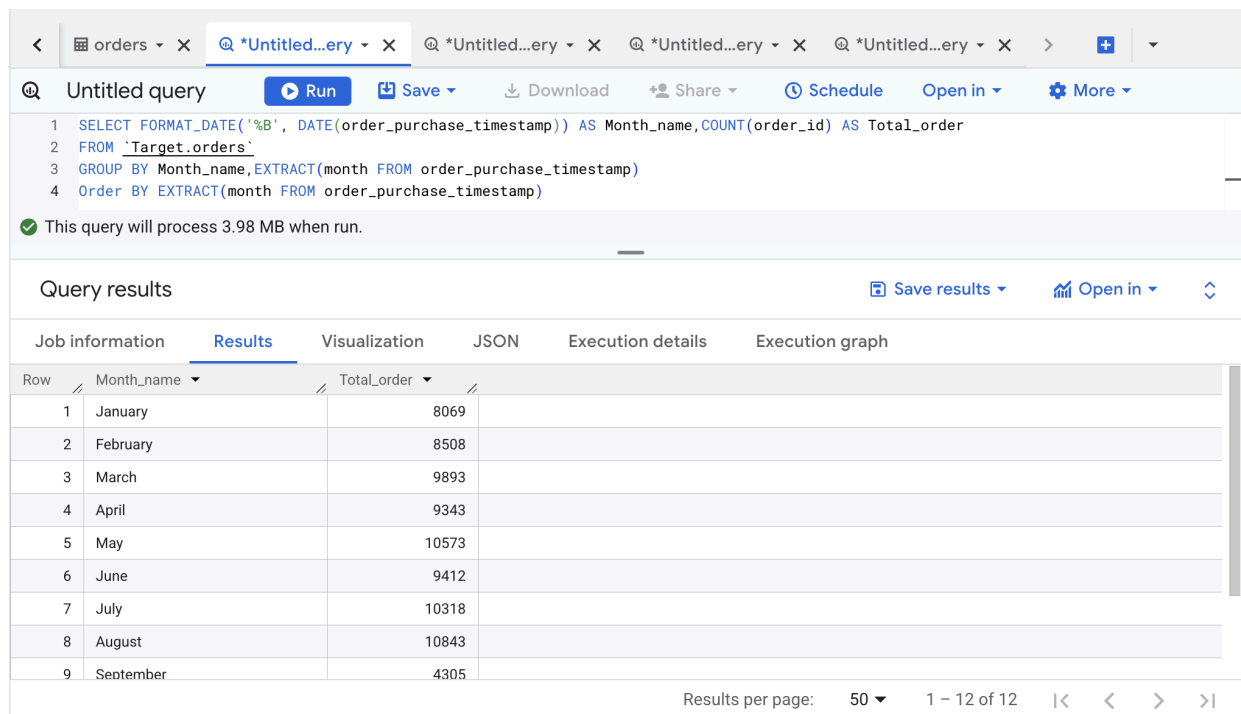
```
SELECT FORMAT_DATE('%B', DATE(order_purchase_timestamp)) AS Month_name, COUNT(order_id) AS
Total_order

FROM `Target.orders`

GROUP BY Month_name, EXTRACT(month FROM order_purchase_timestamp)

Order BY EXTRACT(month FROM order_purchase_timestamp)
```

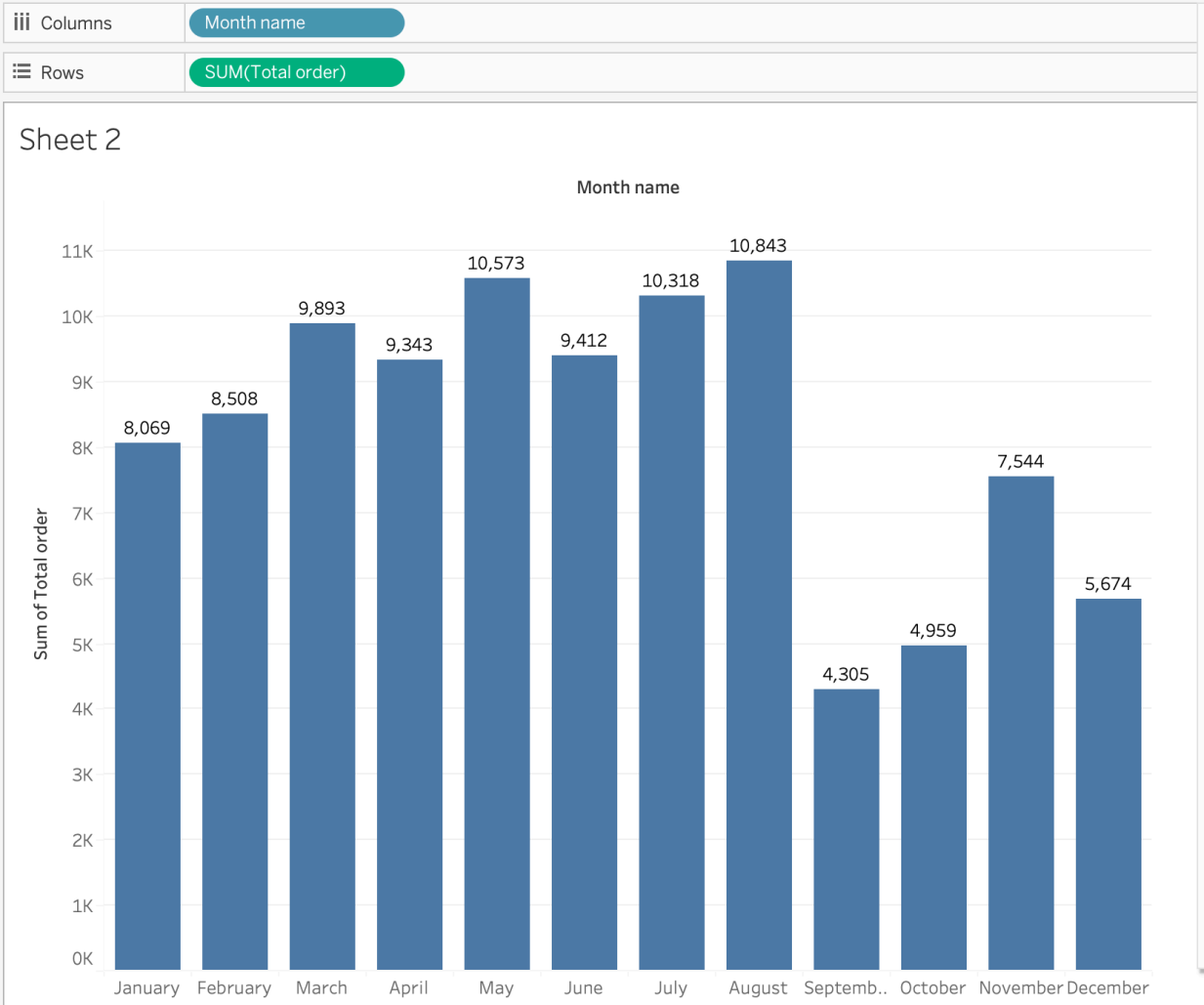
Screenshot:



The screenshot shows a SQL query editor interface. At the top, there's a tab labeled "orders" and a search bar. Below the search bar, the query is displayed in a monospace font. The query is a SQL statement that selects the month name and the count of orders, grouped by month name and ordered by month. The query is executed, and the results are shown in a table below. The table has two columns: "Month\_name" and "Total\_order". The results show the number of orders for each month from January to September. The number of orders increases from January to August and then decreases in September.

Row	Month_name	Total_order
1	January	8069
2	February	8508
3	March	9893
4	April	9343
5	May	10573
6	June	9412
7	July	10318
8	August	10843
9	September	4305

Insight:



The analysis reveals clear monthly seasonality in customer orders. Orders peak in mid-year (May–August), dip significantly during September and October, and recover during November (Black Friday) and December (holiday shopping). This seasonal trend indicates that Target should plan inventory, logistics, and marketing campaigns around mid-year sales and end-of-year festivals while optimizing resources for the September–October slowdown.

### 3. During what time of the day, do the Brazilian customers mostly place their orders?

(Dawn, Morning, Afternoon or Night)

- a. 0-6 hrs : Dawn
- b. 7-12 hrs : Mornings
- c. 13-18 hrs : Afternoon
- d. 19 - 23 hrs : Night

Query:

```
SELECT
CASE
WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN "Dawn"
WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN "Mornings"
WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN "Afternoon"
ELSE "Night"
END AS time_of_day,
COUNT(order_id) AS total_orders
FROM `Target.orders`
GROUP BY time_of_day
ORDER BY total_orders DESC
```

Screenshot:

The screenshot shows a SQL query editor interface with a toolbar at the top containing icons for Run, Save, Download, Share, Schedule, Open in, and More. The query is as follows:

```
1 SELECT
2 CASE
3 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN "Dawn"
4 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN "Mornings"
5 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN "Afternoon"
6 ELSE "Night"
7 END AS time_of_day,
8 COUNT(order_id) AS total_orders
9 FROM `Target.orders`
10 GROUP BY time_of_day
11 ORDER BY total_orders DESC
```

Below the query, a status message indicates: "This query will process 3.98 MB when run." The "Query results" section shows a table with the following data:

Row	time_of_day	total_orders
1	Afternoon	38135
2	Night	28331
3	Mornings	27733
4	Dawn	5242

At the bottom right, the pagination information shows "Results per page: 50" and "1 - 4 of 4".

## Insight:



The analysis shows that Brazilian customers mostly place their orders during the Afternoon (13–18 hrs), followed by the Night (19–23 hrs). Mornings also contribute significantly, whereas very few orders are placed during Dawn (0–6 hrs). This highlights that Target should schedule promotional campaigns, website push notifications, and advertisements during Afternoon and Night hours to maximize engagement and conversion.

# Evolution of E-commerce Orders in Brazil

## 1. Get the month on month no. of orders placed in each state.

Query:

```
SELECT c.customer_state,FORMAT_DATE('%B %Y', DATE(order_purchase_timestamp)) AS
month_year,COUNT(DISTINCT(o.order_id)) AS total_orders

FROM `Target.orders` AS o

JOIN `Target.customers` AS c ON o.customer_id = c.customer_id

GROUP BY customer_state,month_year

ORDER BY month_year,customer_state
```

Screenshot:

The screenshot shows a SQL query editor with a query that calculates the number of orders by state and month. Below the query, a message indicates the query will process 10.81 MB. The results are displayed in a table with columns for customer\_state, month\_year, and total\_orders. The table shows data for states AC, AL, AM, BA, CE, DF, ES, and GO for April 2017. The total number of orders for each state is: AC (5), AL (23), AM (13), BA (93), CE (43), DF (35), ES (46), and GO (41). The interface includes a 'Job history' section at the bottom and a 'Refresh' button.

Query results

Job information	Results	Visualization	JSON	Execution details	Execution graph
Row	customer_state	month_year	total_orders		
1	AC	April 2017	5		
2	AL	April 2017	23		
3	AM	April 2017	13		
4	BA	April 2017	93		
5	CE	April 2017	43		
6	DF	April 2017	35		
7	ES	April 2017	46		
8	GO	April 2017	41		

Results per page: 50 1 - 50 of 565 |< < > >|

Job history Refresh

Insight:

The analysis shows that customer orders are heavily concentrated in major states such as SP, RJ, MG, and BA, while smaller states contribute only a small share. This suggests the need for state-wise differentiated strategies: aggressive growth strategies in core states and awareness/penetration campaigns in smaller ones.

## 2.How are the customers distributed across all the states?

**Query:**

`SELECT c.customer_state, COUNT(DISTINCT c.customer_id) AS customers`

`FROM `Target.customers` AS c`

`JOIN `Target.orders` AS o ON c.customer_id = o.customer_id`

`GROUP BY c.customer_state`

`ORDER BY customers`

**Screenshot:**

The screenshot shows a SQL query editor with the following query:

```
1 SELECT c.customer_state, COUNT(DISTINCT c.customer_id) AS customers
2 FROM `Target.customers` AS c
3 JOIN `Target.orders` AS o ON c.customer_id = o.customer_id
4 GROUP BY c.customer_state
5 ORDER BY customers
```

Below the query, a status bar indicates: "This query will process 6.83 MB when run."

The "Query results" section shows a table with the following data:

Row	customer_state	customers
1	RR	46
2	AP	68
3	AC	81
4	AM	148
5	RO	253
6	TO	280
7	SE	350

At the bottom right, the pagination shows "Results per page: 50" and "1 - 27 of 27".

**Insight:**

Customer distribution is highly uneven across Brazilian states. While SP, RJ, and MG dominate in terms of customer base, northern states like RR, AP, and AC contribute minimally, indicating a need for expansion strategies in underserved regions.

## **Impact on Economy**

**1. Get the % increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only).**

**You can use the “payment\_value” column in the payments table to get the cost of orders.**

**Query:**

**Screenshot:**

**Insight:**

## 2. Calculate the Total & Average value of order price for each state.

Query:

```
SELECT
c.customer_state AS state,
SUM(oi.price) AS total_order_price,
AVG(oi.price) AS avg_order_price
FROM `Target.order_items` AS oi
JOIN `Target.orders` AS o ON oi.order_id = o.order_id
JOIN `Target.customers` AS c ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY total_order_price DESC;
```

Screenshot:

The screenshot shows a SQL query editor interface. At the top, there's a toolbar with buttons for 'Run', 'Save', 'Download', 'Share', and 'Schedule'. Below the toolbar, the SQL query is entered in a text area. A status message indicates the query will process 14.56 MB. Below the query editor, the 'Query results' section is visible, showing a table with 7 rows of data. The table has columns for 'state', 'total\_order\_price', and 'avg\_order\_price'. The results are sorted by 'total\_order\_price' in descending order. At the bottom, there's a pagination bar showing 'Results per page: 50' and '1 - 27 of 27'.

Job information	Results	Visualization	JSON	Execution details	Execution graph
Row	state	total_order_price	avg_order_price		
1	SP	5202955.049999...	109.6536291597...		
2	RJ	1824092.669999...	125.1178180945...		
3	MG	1585308.030000...	120.7485741488...		
4	RS	750304.0200000...	120.3374530874...		
5	PR	683083.760000001	119.0041393728...		
6	SC	520553.3399999...	124.6535775862...		
7	BA	511349.9899999...	134.6012082126...		

Insight:



### 3. Calculate the Total & Average value of order freight for each stage.

#### Query:

```
SELECT
c.customer_state AS state,
SUM(oi.freight_value) AS total_freight_value,
AVG(oi.freight_value) AS avg_freight_value
FROM `Target.order_items` AS oi
JOIN `Target.orders` AS o ON oi.order_id = o.order_id
JOIN `Target.customers` AS c ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY state;
```

#### Screenshot:

The screenshot shows a SQL query editor interface. At the top, there's a toolbar with buttons for 'Run', 'Save', 'Download', 'Share', and 'Schedule'. Below the toolbar, the query is entered in a text area. A status message indicates the query will process 14.56 MB. Below the query, there's a 'Query results' section with tabs for 'Job information', 'Results', 'Visualization', 'JSON', 'Execution details', and 'Execution graph'. The 'Results' tab is active, showing a table with 7 rows of data. The table has columns for 'Row', 'state', 'total\_freight\_value', and 'avg\_freight\_value'. The data is sorted by state (AC, AL, AM, AP, BA, CE, DE). At the bottom, there's a pagination bar showing 'Results per page: 50' and '1 - 27 of 27'.

```
1 SELECT
2   c.customer_state AS state,
3   SUM(oi.freight_value) AS total_freight_value,
4   AVG(oi.freight_value) AS avg_freight_value
5 FROM `Target.order_items` AS oi
6 JOIN `Target.orders` AS o ON oi.order_id = o.order_id
7 JOIN `Target.customers` AS c ON o.customer_id = c.customer_id
8 GROUP BY c.customer_state
9 ORDER BY state;
```

✓ This query will process 14.56 MB when run.

Query results [Save results](#) [Open in](#)

Job information	Results	Visualization	JSON	Execution details	Execution graph
Row	state	total_freight_value	avg_freight_value		
1	AC	3686.749999999...	40.07336956521...		
2	AL	15914.590000000...	35.84367117117...		
3	AM	5478.890000000...	33.20539393939...		
4	AP	2788.499999999...	34.00609756097...		
5	BA	100156.6800000...	26.36395893656...		
6	CE	48351.590000000...	32.71420162381...		
7	DE	50625.500000000...	21.04125404506...		

Results per page: 50 1 - 27 of 27

#### Insight:

# Analysis on Sales,Freight & Delivery Time

1.Find the no.of days taken to deliver each order from the order's purchase date as delivery time.Also,calculate the difference (in days) between the estimated & actual delivery date of an order.

## Query:

```
SELECT order_id,DATE(order_purchase_timestamp) AS purchase_date,DATE(order_delivered_customer_date) AS delivered_date,DATE(order_estimated_delivery_date) AS estimated_date,DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,DAY) AS time_to_deliver,DATE_DIFF(order_delivered_customer_date,order_estimated_delivery_date,DAY) AS diff_estimated_delivery FROM `Target.orders` WHERE order_status = "delivered" ORDER BY purchase_date
```

## Screenshot:

The screenshot shows a SQL query editor interface with a query titled 'Untitled query'. The query is as follows:

```
1 SELECT order_id,DATE(order_purchase_timestamp) AS purchase_date,DATE(order_delivered_customer_date) AS delivered_date,DATE(order_estimated_delivery_date) AS estimated_date,DATE_DIFF
2 (order_delivered_customer_date,order_purchase_timestamp,DAY) AS time_to_deliver,DATE_DIFF
3 (order_delivered_customer_date,order_estimated_delivery_date,DAY) AS diff_estimated_delivery
4 FROM `Target.orders`
5 WHERE order_status = "delivered"
6 ORDER BY purchase_date
```

Below the query, a status message indicates: "This query will process 6.52 MB when run."

The 'Query results' section shows a table with the following data:

Row	order_id	purchase_date	delivered_date	estimated_date	time_to_deliver	diff_es
1	bfb0f9bdef84302105ad712db...	2016-09-15	2016-11-09	2016-10-04	54	
2	3b697a20d9e427646d9256791...	2016-10-03	2016-10-26	2016-10-27	23	
3	be5bc2f0da14d8071e2d45451a...	2016-10-03	2016-10-27	2016-11-07	24	
4	cd3b8574c82b42fc8129f6d502...	2016-10-03	2016-10-14	2016-11-23	10	
5	d207cc272675637bfd0062edf...	2016-10-03	2016-10-31	2016-11-23	27	

At the bottom, the 'Job history' section is visible with a 'Refresh' button.

## Insight:

## 2. Find out the 5 states with the highest & lowest average freight value.

### Query:

```
WITH order_freight AS (  
  SELECT  
    o.order_id,  
    c.customer_state AS state,  
    SUM(oi.freight_value) AS order_freight  
  FROM `Target.order_items` AS oi  
  JOIN `Target.orders` AS o USING (order_id)  
  JOIN `Target.customers` AS c ON o.customer_id = c.customer_id  
  GROUP BY o.order_id, state  
,  
state_avg AS (  
  SELECT  
    ofr.state,  
    AVG(ofr.order_freight) AS avg_freight  
  FROM order_freight AS ofr  
  GROUP BY ofr.state  
,  
top5 AS (  
  SELECT state, avg_freight, 'Highest' AS category  
  FROM state_avg  
  ORDER BY avg_freight DESC  
  LIMIT 5  
,  
bottom5 AS (  
  SELECT state, avg_freight, 'Lowest' AS category  
  FROM state_avg  
  ORDER BY avg_freight ASC  
  LIMIT 5  
)  
SELECT * FROM top5  
UNION ALL  
SELECT * FROM bottom5  
ORDER BY category,  
  CASE WHEN category='Highest' THEN -avg_freight ELSE avg_freight END;
```

Screenshot:

Untitled query

Run

Save

Download

Share

Schedule

Open in

More

12SELECT

This query will process 14.56 MB when run.

Query results

Save results

Open in

Job information

Results

Visualization

JSON

Execution details

Execution graph

Row	state	avg_freight	category
1	RR	48.59108695652...	Highest
2	PB	48.34535714285...	Highest
3	RO	46.22421052631...	Highest
4	AC	45.51543209876...	Highest
5	PI	43.03894523326...	Highest
6	SP	17.37095033232...	Lowest
7	MG	23.46270443520...	Lowest
8	PR	23.57976790716...	Lowest
9	DF	23.82376470588...	Lowest
10	RJ	23.94525231154...	Lowest

Results per page: 501 – 10 of 10

Job history

Refresh

Insight:

- **Highest Freight States** → RR, PB, RO, AC, PI (avg freight > 43 BRL).
  - Remote & less connected, with weaker logistics networks, leading to higher costs.
- **Lowest Freight States** → SP, MG, PR, DF, RJ (avg freight between 17–24 BRL).
  - Major urban hubs with dense population & efficient logistics, lowering costs.

### 3. Find out the 5 states with the highest & lowest average delivery time.

#### Query:

```
WITH delivery_time AS (
  SELECT
    o.order_id,
    c.customer_state AS state,
    DATE_DIFF(DATE(o.order_delivered_customer_date), DATE(o.order_purchase_timestamp), DAY) AS delivery_days
  FROM `Target.orders` AS o
  JOIN `Target.customers` AS c
    ON o.customer_id = c.customer_id
  WHERE o.order_status = 'delivered'
  AND o.order_delivered_customer_date IS NOT NULL
),
state_avg AS (
  SELECT
    state,
    AVG(delivery_days) AS avg_delivery_days
  FROM delivery_time
  GROUP BY state
),
top5 AS (
  SELECT state, avg_delivery_days, 'Slowest' AS category
  FROM state_avg
  ORDER BY avg_delivery_days DESC
  LIMIT 5
),
bottom5 AS (
  SELECT state, avg_delivery_days, 'Fastest' AS category
  FROM state_avg
  ORDER BY avg_delivery_days ASC
  LIMIT 5
)
SELECT * FROM top5
UNION ALL
SELECT * FROM bottom5
ORDER BY category,
  CASE WHEN category='Slowest' THEN -avg_delivery_days ELSE avg_delivery_days END;
```

## Screenshot:

Untitled query	Run	Save	Download	Share	Schedule	Open in	More
9 WHERE o.order_status = 'delivered'							
This query will process 9.36 MB when run.							
Query results							
Save results Open in							
Job information Results Visualization JSON Execution details Execution graph							
Row	state	avg_delivery_days	category				
1	SP	8.70057292438388	Fastest				
2	PR	11.93804590696...	Fastest				
3	MG	11.94495332041...	Fastest				
4	DF	12.89903846153...	Fastest				
5	SC	14.90298928369...	Fastest				
6	RR	29.34146341463...	Slowest				
7	AP	27.17910447761...	Slowest				
8	AM	26.35862068965...	Slowest				
9	AL	24.50125944584...	Slowest				
10	PA	23.72515856236...	Slowest				
Results per page: 50 1 - 10 of 10							

## Insight:

### Fastest States:

- São Paulo (SP) has the quickest deliveries, averaging just 8.7 days, followed by Paraná (PR), Minas Gerais (MG), and Distrito Federal (DF) (11–13 days).
- These states are major urban and logistics hubs, which explains the efficiency in delivery speed.

### Slowest States:

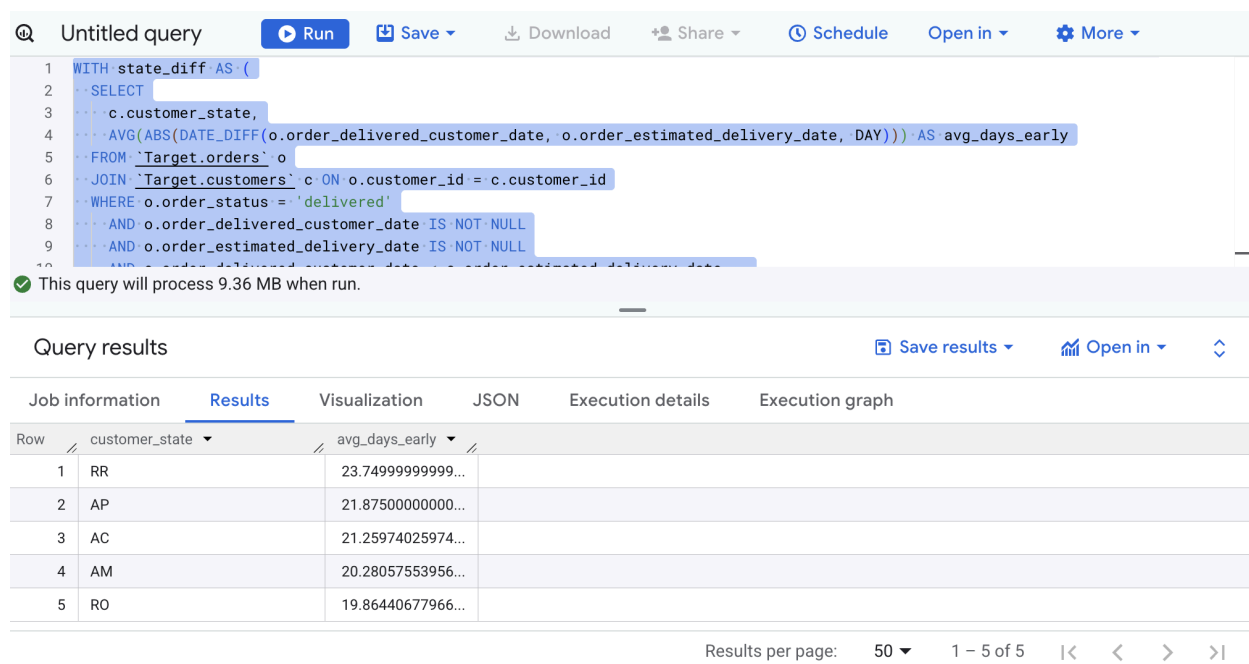
- States like Roraima (RR) (29.3 days), Amapá (AP) (27.2 days), and Amazonas (AM) (26.3 days) show the longest delivery times.
- These regions are geographically remote, with logistical challenges such as limited transport routes and greater reliance on air/river freight.

4. Find out the 5 states where the order delivery is really fast as compared to the estimated date of delivery. You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state

Query:

```
WITH state_diff AS (  
  SELECT  
    c.customer_state,  
    AVG(ABS(DATE_DIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date, DAY))) AS  
    avg_days_early  
  FROM `Target.orders` o  
  JOIN `Target.customers` c ON o.customer_id = c.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  
  AND o.order_delivered_customer_date < o.order_estimated_delivery_date  
  GROUP BY c.customer_state  
)  
SELECT customer_state, avg_days_early  
FROM state_diff  
ORDER BY avg_days_early DESC  
LIMIT 5;
```

Screenshot:



The screenshot shows a SQL query editor with the following query:

```
WITH state_diff AS (  
  SELECT  
    c.customer_state,  
    AVG(ABS(DATE_DIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date, DAY))) AS avg_days_early  
  FROM `Target.orders` o  
  JOIN `Target.customers` c ON o.customer_id = c.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  
  AND o.order_delivered_customer_date < o.order_estimated_delivery_date  
  GROUP BY c.customer_state  
)  
SELECT customer_state, avg_days_early  
FROM state_diff  
ORDER BY avg_days_early DESC  
LIMIT 5;
```

Below the query editor, a message states: "This query will process 9.36 MB when run."

The query results are displayed in a table with the following columns: Row, customer\_state, and avg\_days\_early. The results are ordered by avg\_days\_early in descending order.

Row	customer_state	avg_days_early
1	RR	23.749999999999...
2	AP	21.875000000000...
3	AC	21.25974025974...
4	AM	20.28057553956...
5	RO	19.86440677966...

At the bottom of the results table, there is a pagination bar showing "Results per page: 50" and "1 - 5 of 5".

**Insight:**

*“Customers in RR, AP, AC, AM, and RO received their orders on average 20+ days before the estimated delivery date, indicating that Target’s delivery promises are too conservative in these states. This creates an opportunity to optimize estimated delivery timelines for better customer communication while still delighting them with fast service.”*



# Analysis Based on Payments

1. Find the month on month no. of orders placed using different payment types.

Query:

```
WITH CTE AS(  
  SELECT FORMAT_DATE("%Y-%m",DATE(o.order_purchase_timestamp)) AS  
  month_year,p.payment_type,o.order_id  
  FROM `Target.orders` AS o  
  JOIN `Target.payments` AS p ON o.order_id = p.order_id)  
SELECT month_year,payment_type,COUNT(DISTINCT order_id) AS orders  
FROM CTE  
GROUP BY month_year,payment_type  
ORDER BY month_year,payment_type
```

Screenshot:

The screenshot shows a SQL query editor interface. At the top, there's a toolbar with buttons for 'Run', 'Share', 'Schedule', 'Open in', 'More', 'Save', and 'Download'. Below the toolbar, the SQL query is entered in a text area. A status message indicates the query will process 8.47 MB when run. Below the query editor, the 'Query results' section is displayed, showing a table with 5 rows of data. The table has columns for 'month\_year', 'payment\_type', and 'orders'. The results show data for September 2016 and October 2016, with payment types including credit\_card, UPI, debit\_card, and voucher. At the bottom, there's a 'Job history' section with a 'Refresh' button.

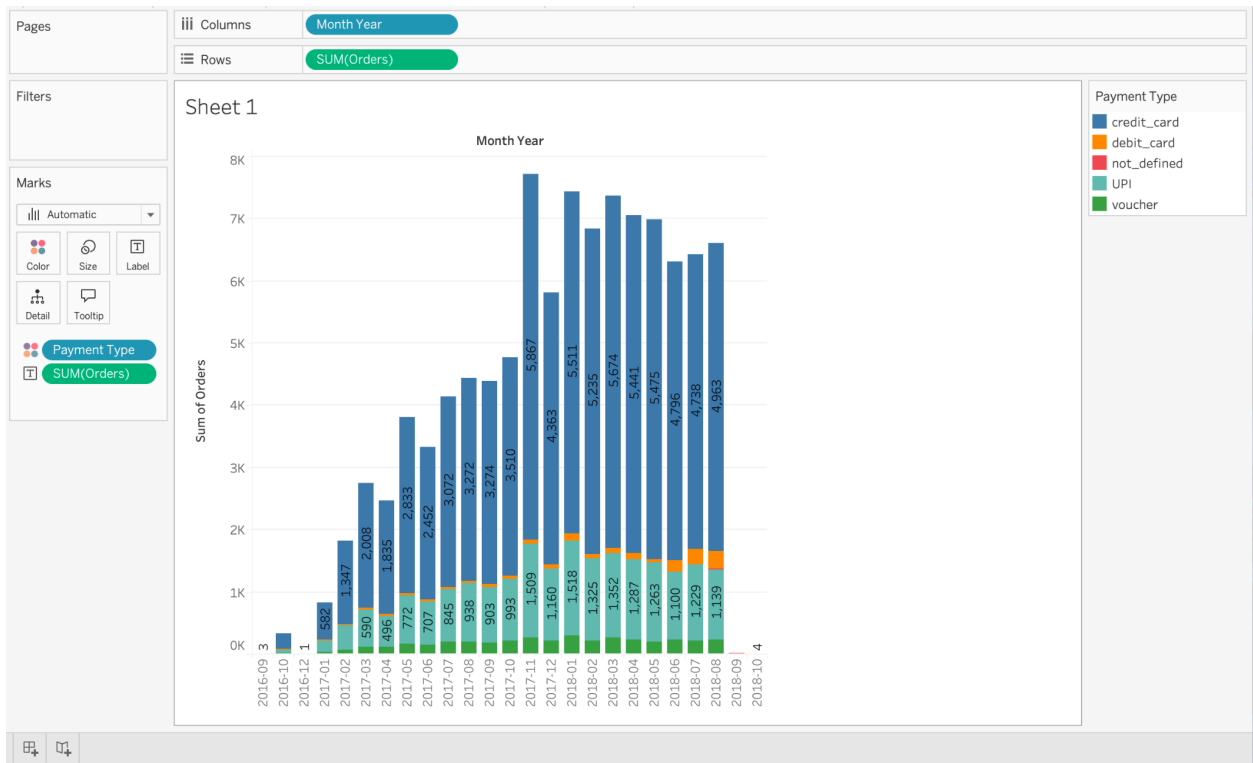
Query results

Row	month_year	payment_type	orders
1	2016-09	credit_card	3
2	2016-10	UPI	63
3	2016-10	credit_card	253
4	2016-10	debit_card	2
5	2016-10	voucher	11

Results per page: 50 1 - 50 of 90

Job history Refresh

Insight:



Analysis of month-on-month orders by payment type shows that credit cards are the overwhelmingly preferred method of payment, driving overall order growth. While other methods maintain steady volumes, they do not show the same growth trajectory. Seasonal spikes in December and January reflect holiday shopping, further emphasizing credit card dominance. Alternative methods (debit, vouchers, UPI) remain niche, suggesting limited customer adoption.

## 2.Find the no.of orders placed on the basis of the payment installments that have been paid.

### Query:

```
SELECT p.payment_installments,COUNT(DISTINCT p.order_id) AS orders
FROM `Target.payments` AS p
JOIN `Target.orders` AS o ON p.order_id = o.order_id
GROUP BY payment_installments
ORDER BY payment_installments
```

### Screenshot:

The screenshot shows a SQL query editor interface. The query is as follows:

```
1 SELECT p.payment_installments,COUNT(DISTINCT p.order_id) AS orders
2 FROM `Target.payments` AS p
3 JOIN `Target.orders` AS o ON p.order_id = o.order_id
4 GROUP BY payment_installments
5 ORDER BY payment_installments
```

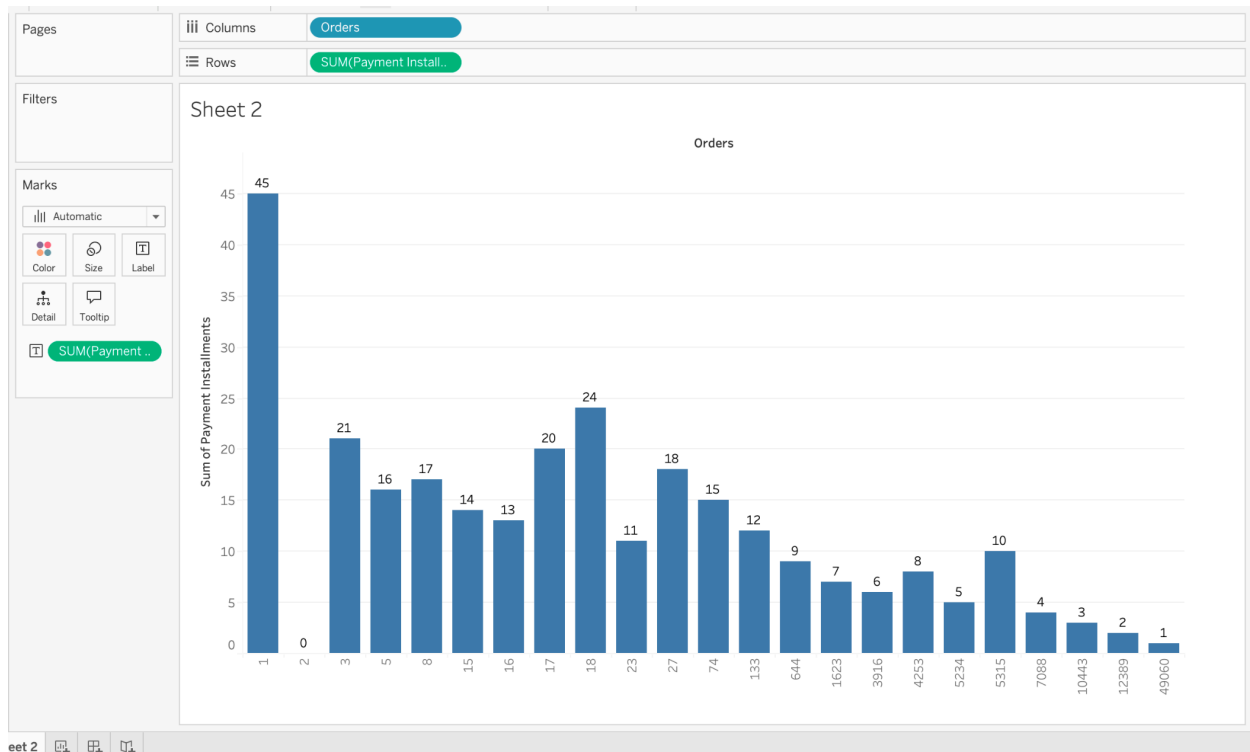
Below the query, a status message indicates: "This query will process 7.39 MB when run."

The "Query results" section is active, showing a table with the following data:

Row	payment_installments	orders
1	0	2
2	1	49060
3	2	12389
4	3	10443
5	4	7088
6	5	5234
7	6	3916
8	7	1623

At the bottom of the results section, it says "Results per page: 50" and "1 - 24 of 24".

### Insight:



The majority of orders are paid in a single installment, showing customer preference for upfront payments. Installment plans up to 6 months see moderate adoption, while longer installment options are rarely chosen. This suggests that offering short-term installment promotions (3–6 months) could align well with customer behavior, while extended EMI schemes may not add much value.