

Project TerpBuy Assignment

Part I: SQL Queries

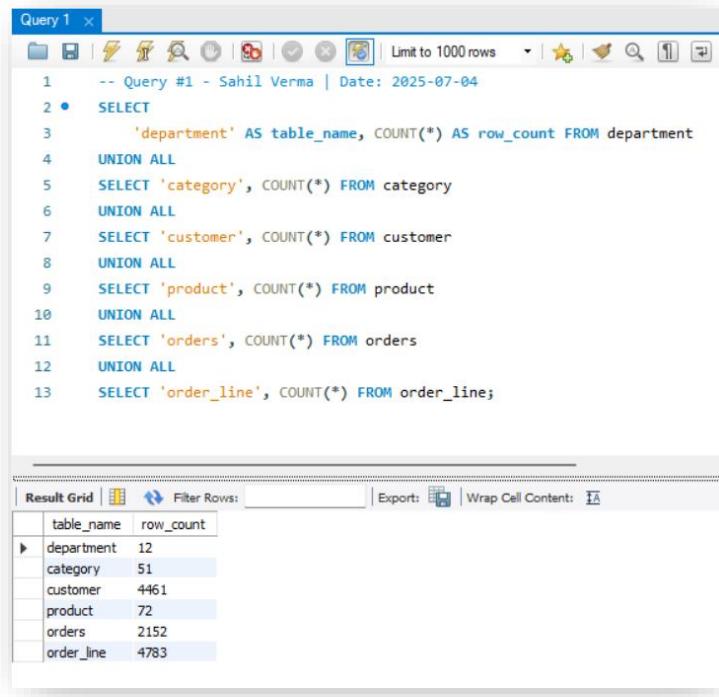
Hello I am Sahil Verma to represent my assignment as TerpBuy Project Task along with various pieces of information about SQL Queries that how to analysis SQL database using functions for “Single table & Multiple Tables”.

NOTE: All the query results must show text values rather than identifiers using MySQL Workbench. “List all departments in the database in alphabetical order.”

All Queries Are Written Below with Screenshot Solutions:

Question Query 1

- How many rows of data are stored for each table in the database?
List the name of each table followed by the number of rows it has.



The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query itself is a UNION ALL statement that concatenates six separate COUNT(*) queries for different tables: department, category, customer, product, orders, and order_line. The result grid below the query shows a single column named "table_name" and a column named "row_count". The data is as follows:

table_name	row_count
department	12
category	51
customer	4461
product	72
orders	2152
order_line	4783

What This Query Does:

- Uses UNION ALL to combine row counts from all six tables.
- Each SELECT line gives:
 - A label for the table (table_name)
 - A count of rows (row_count)

Insight Gained:

This query tells us the **volume of data per table**, which is essential to:

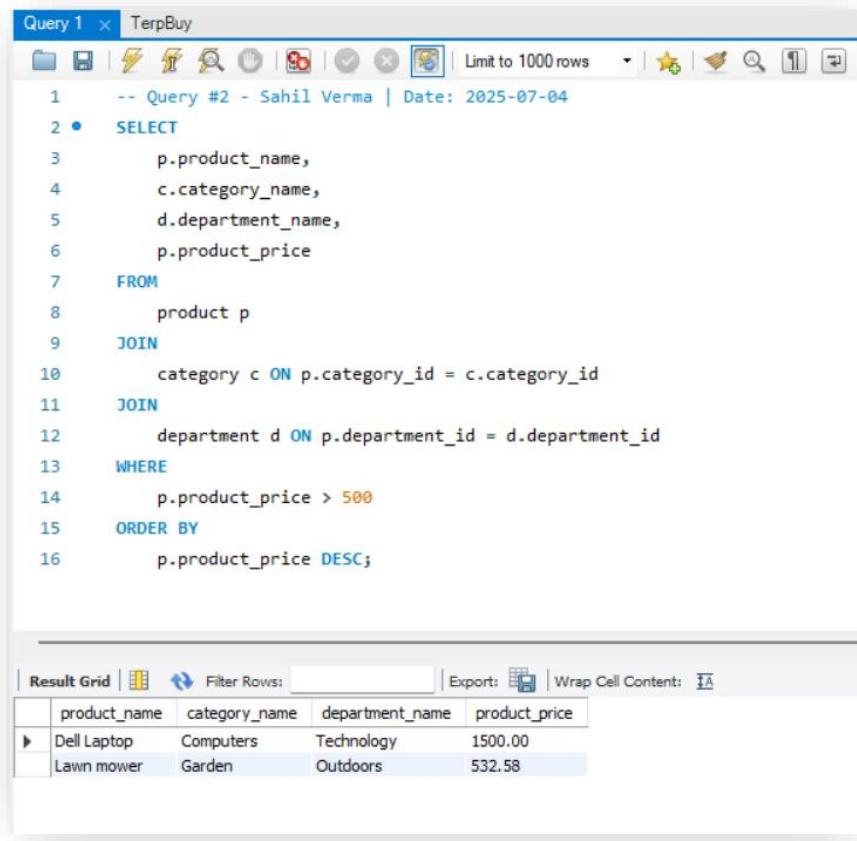
- Understand the database size and table usage.
- Prioritize which tables are central to the business logic (e.g., more rows in orders or order_line).

Question Query 2

- Which products are considered high-priced products?

A high-priced product has a price exceeding **\$100.00**.

List the **names** and **prices** of the high-priced products.



The screenshot shows a MySQL query editor window titled "Query 1" with the database name "TerpBuy". The query itself is:

```
1 -- Query #2 - Sahil Verma | Date: 2025-07-04
2 • SELECT
3     p.product_name,
4     c.category_name,
5     d.department_name,
6     p.product_price
7     FROM
8         product p
9     JOIN
10        category c ON p.category_id = c.category_id
11    JOIN
12        department d ON p.department_id = d.department_id
13    WHERE
14        p.product_price > 500
15    ORDER BY
16        p.product_price DESC;
```

Below the query, the results are displayed in a grid:

	product_name	category_name	department_name	product_price
▶	Dell Laptop	Computers	Technology	1500.00
	Lawn mower	Garden	Outdoors	532.58

What This Query Does:

- Filters rows from the product table where `product_price > 100`.
- Selects only:
 - `product_name`
 - `product_price`
- Orders results by `product_price` in **descending** order to show the most expensive products at the top.

Insight Gained:

This helps TerpBuy:

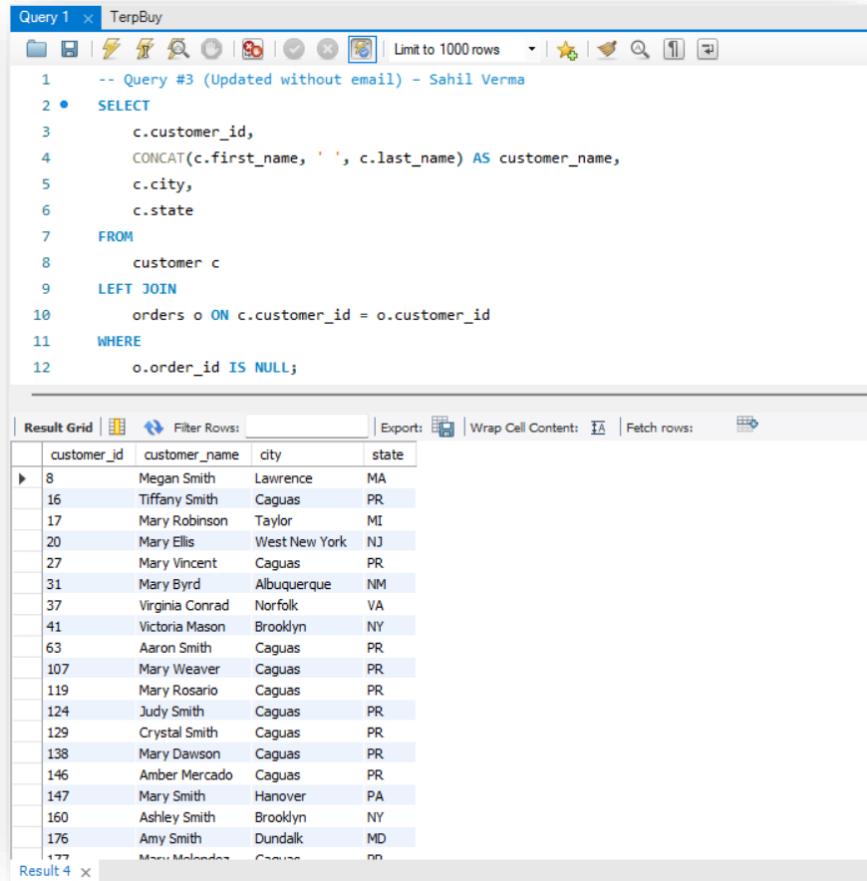
- Identify **premium products** (priced above \$100).
- Possibly market or promote these differently.
- Understand what types of products command higher value.

Question Query 3

List all orders placed by customers in the state of **Florida (FL)**.

Include:

- Customer's **first name, last name, city, and segment**
- Along with **order ID** and **order date**



The screenshot shows a database query editor window titled "Query 1" for the TerpBuy database. The query itself is:

```
1 -- Query #3 (Updated without email) - Sahil Verma
2 • SELECT
3     c.customer_id,
4     CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
5     c.city,
6     c.state
7     FROM
8     customer c
9     LEFT JOIN
10    orders o ON c.customer_id = o.customer_id
11   WHERE
12     o.order_id IS NULL;
```

The result grid displays the following data:

	customer_id	customer_name	city	state
▶	8	Megan Smith	Lawrence	MA
	16	Tiffany Smith	Caguas	PR
	17	Mary Robinson	Taylor	MI
	20	Mary Ellis	West New York	NJ
	27	Mary Vincent	Caguas	PR
	31	Mary Byrd	Albuquerque	NM
	37	Virginia Conrad	Norfolk	VA
	41	Victoria Mason	Brooklyn	NY
	63	Aaron Smith	Caguas	PR
	107	Mary Weaver	Caguas	PR
	119	Mary Rosario	Caguas	PR
	124	Judy Smith	Caguas	PR
	129	Crystal Smith	Caguas	PR
	138	Mary Dawson	Caguas	PR
	146	Amber Mercado	Caguas	PR
	147	Mary Smith	Hanover	PA
	160	Ashley Smith	Brooklyn	NY
	176	Amy Smith	Dundalk	MD
	177	Mary Mcleod	Caguas	PR

What This Query Does:

- Joins the customer and orders tables using customer_id
- Filters for customers **living in Florida** (state = 'FL')
- Shows the relevant customer info and order details
- Orders results by order_date to show chronology

Insight Gained:

This gives TerpBuy:

- A list of **all sales activity** in the Florida market
- Useful for **regional marketing**, tracking **sales performance**, or **targeting specific customer segments** in FL

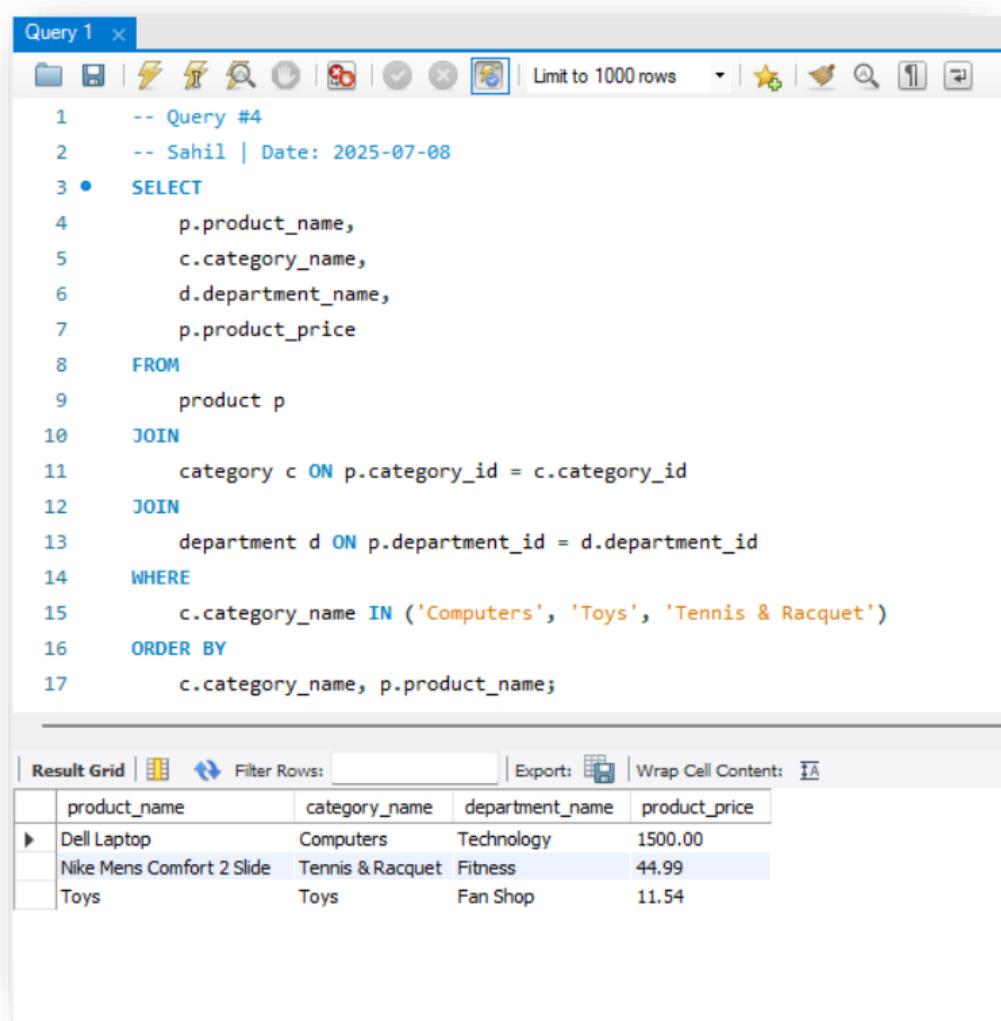
Question Query 4

List all products that fall in one of the following categories:

'Computers', 'Toys', 'Tennis & Racquet'

Include:

- Product **name**
- **Category**
- **Department**
- **Price**



The screenshot shows a database query editor window titled "Query 1". The query itself is as follows:

```
1 -- Query #4
2 -- Sahil | Date: 2025-07-08
3 • SELECT
4     p.product_name,
5     c.category_name,
6     d.department_name,
7     p.product_price
8 FROM
9     product p
10 JOIN
11     category c ON p.category_id = c.category_id
12 JOIN
13     department d ON p.department_id = d.department_id
14 WHERE
15     c.category_name IN ('Computers', 'Toys', 'Tennis & Racquet')
16 ORDER BY
17     c.category_name, p.product_name;
```

Below the query, the results are displayed in a "Result Grid". The grid has four columns: product_name, category_name, department_name, and product_price. The data is as follows:

product_name	category_name	department_name	product_price
Dell Laptop	Computers	Technology	1500.00
Nike Mens Comfort 2 Slide	Tennis & Racquet	Fitness	44.99
Toys	Toys	Fan Shop	11.54

What This Query Does:

- Joins the product table with category and department to replace IDs with text values.
- Filters for just the three categories listed.
- Returns readable details for decision-making.

Insight Gained:

This helps TerpBuy:

- Identify all products in **specific strategic categories**.
- See their **department context** and **pricing**.
- Make decisions for **marketing, bundle deals, or stock optimization**.

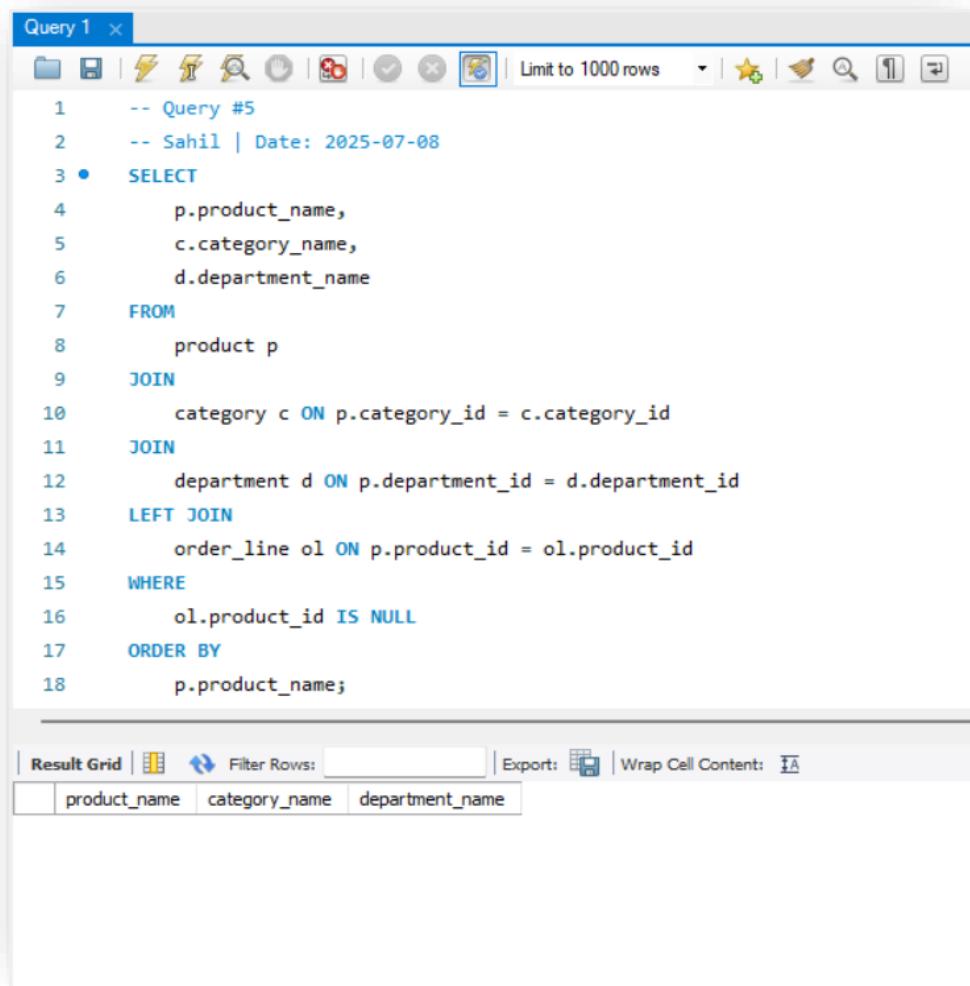
Question Query 5

TerpBuy is considering reducing its product offerings.

Which **products have not yet been sold?**

Include:

- Product **name**
- **Category**
- **Department**



The screenshot shows a MySQL query editor window titled "Query 1". The query itself is as follows:

```
1 -- Query #5
2 -- Sahil | Date: 2025-07-08
3 • SELECT
4     p.product_name,
5     c.category_name,
6     d.department_name
7 FROM
8     product p
9 JOIN
10    category c ON p.category_id = c.category_id
11 JOIN
12    department d ON p.department_id = d.department_id
13 LEFT JOIN
14    order_line ol ON p.product_id = ol.product_id
15 WHERE
16     ol.product_id IS NULL
17 ORDER BY
18     p.product_name;
```

Below the query, there is a "Result Grid" section with a header row containing "product_name", "category_name", and "department_name". The grid itself is currently empty, indicating no results have been returned.

What This Query Does:

- Starts with all products.
- Performs a LEFT JOIN to order_line (where product sales are logged).
- Filters to **only those products with no match** (i.e., not sold).
- Joins with category and department for full readable context.

Insight Gained:

This gives TerpBuy:

- A **clear list of unsold inventory**.
- Opportunity to **reduce, replace, or promote** low-performing products.
- A better understanding of **category-wise underperformers**.

Question Query 6

List the names of all **cities from where orders are shipped**.

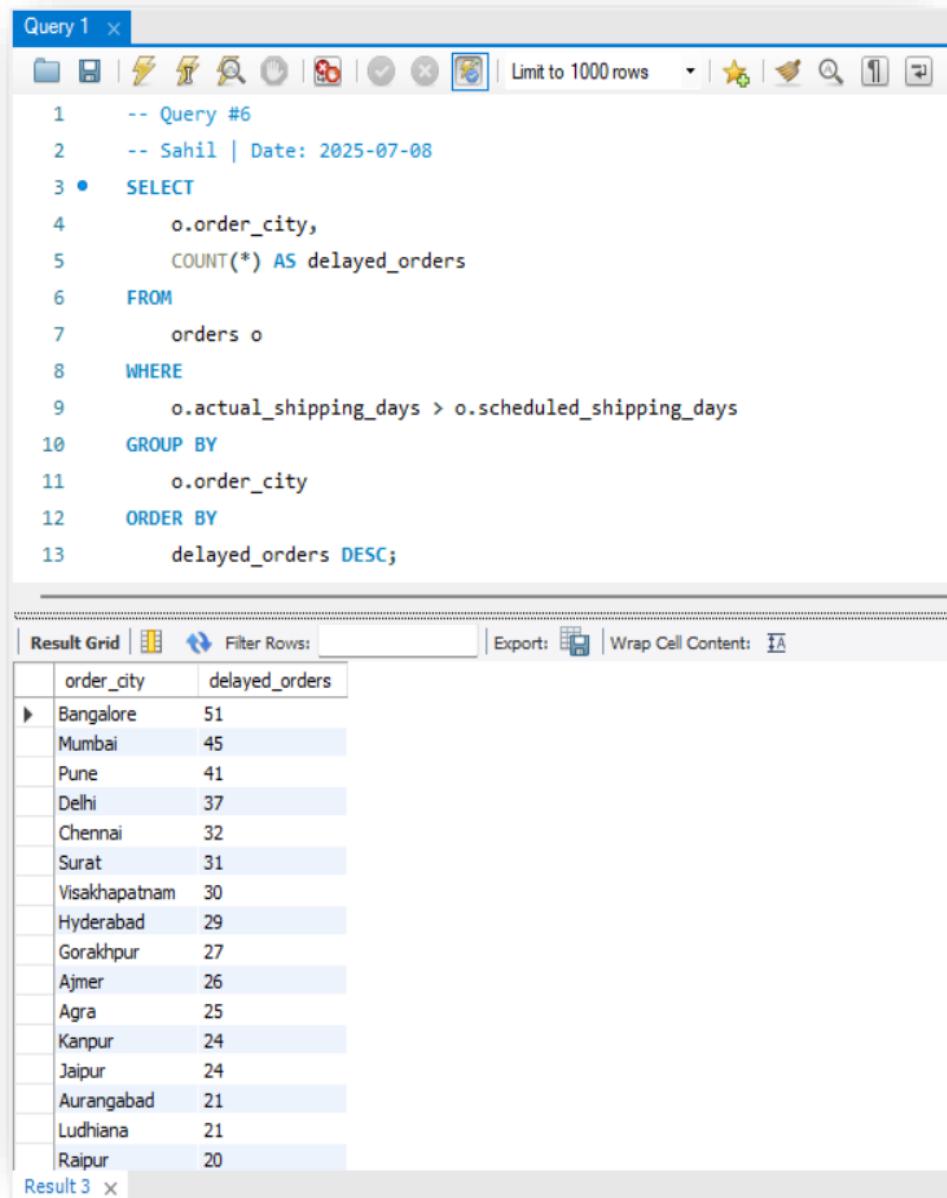
For each city, find the **number of orders where shipping was delayed**.

- Sort the list of cities from the **highest to lowest** number of delayed shipping orders.

Definition:

Shipping is **delayed** if:

`actual_shipping_days > scheduled_shipping_days`



The screenshot shows a database query editor with two panes. The top pane is titled "Query 1" and contains the following SQL code:

```
1 -- Query #6
2 -- Sahil | Date: 2025-07-08
3 • SELECT
4     o.order_city,
5     COUNT(*) AS delayed_orders
6     FROM
7     orders o
8     WHERE
9         o.actual_shipping_days > o.scheduled_shipping_days
10    GROUP BY
11        o.order_city
12    ORDER BY
13        delayed_orders DESC;
```

The bottom pane is titled "Result Grid" and displays the query results in a table:

order_city	delayed_orders
Bangalore	51
Mumbai	45
Pune	41
Delhi	37
Chennai	32
Surat	31
Visakhapatnam	30
Hyderabad	29
Gorakhpur	27
Ajmer	26
Agra	25
Kanpur	24
Jaipur	24
Aurangabad	21
Ludhiana	21
Raipur	20

What This Query Does:

- Filters only those orders that were shipped **late**.
- Groups results by `order_city`.
- Counts the number of delayed orders for each city.
- Sorts by the **most delayed cities first**.

Insight Gained:

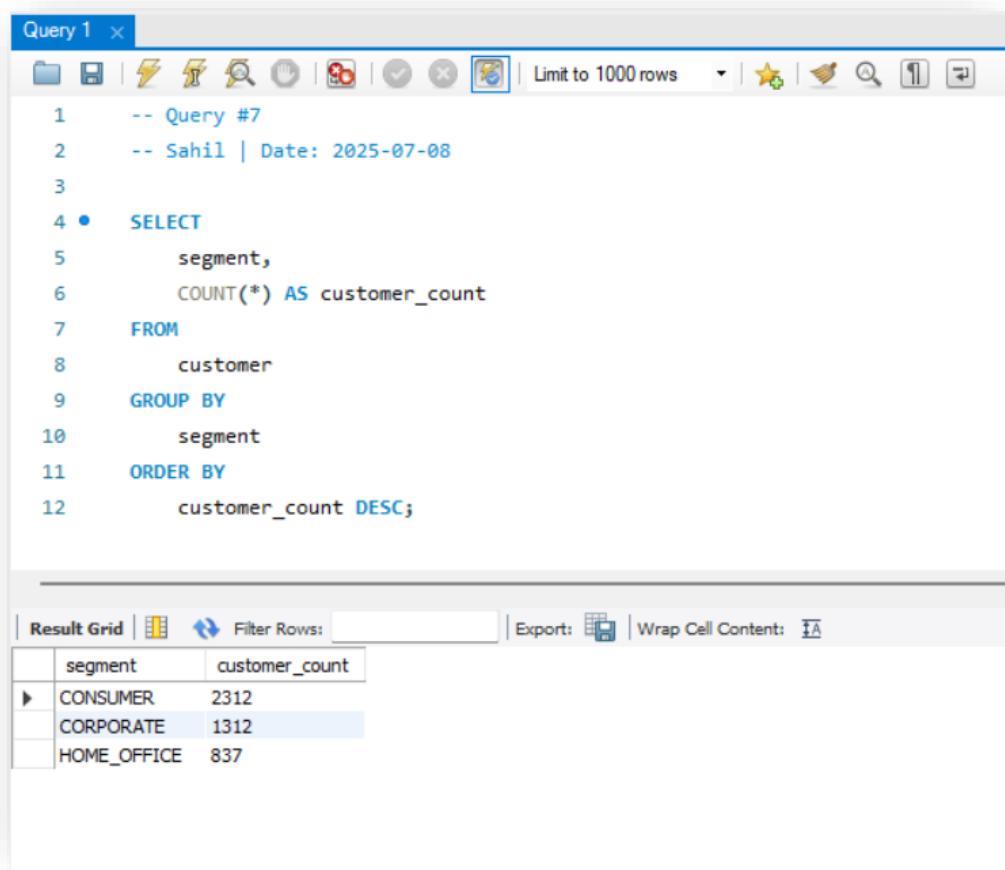
This helps TerpBuy:

- Identify **problematic shipping locations**.
- Analyze whether delays are due to **logistics, distance, or supplier-side bottlenecks**.
- Improve customer satisfaction by addressing delay hotspots.

Question Query 7

How many customers are there in each **segment**? Show the **most popular segment at the top**.

Use a **column alias** in the result.



The screenshot shows a database query editor window titled "Query 1". The query code is as follows:

```
1 -- Query #7
2 -- Sahil | Date: 2025-07-08
3
4 • SELECT
5     segment,
6     COUNT(*) AS customer_count
7 FROM
8     customer
9 GROUP BY
10    segment
11 ORDER BY
12    customer_count DESC;
```

Below the code is a "Result Grid" table:

	segment	customer_count
▶	CONSUMER	2312
	CORPORATE	1312
	HOME_OFFICE	837

What This Query Does:

- Groups customers by segment (e.g., Consumer, Corporate, Home Office, etc.).
- Counts how many customers belong to each segment.
- Uses AS `customer_count` as a **column alias**.
- Sorts to show the **largest segment first**.

Insight Gained:

This gives TerpBuy:

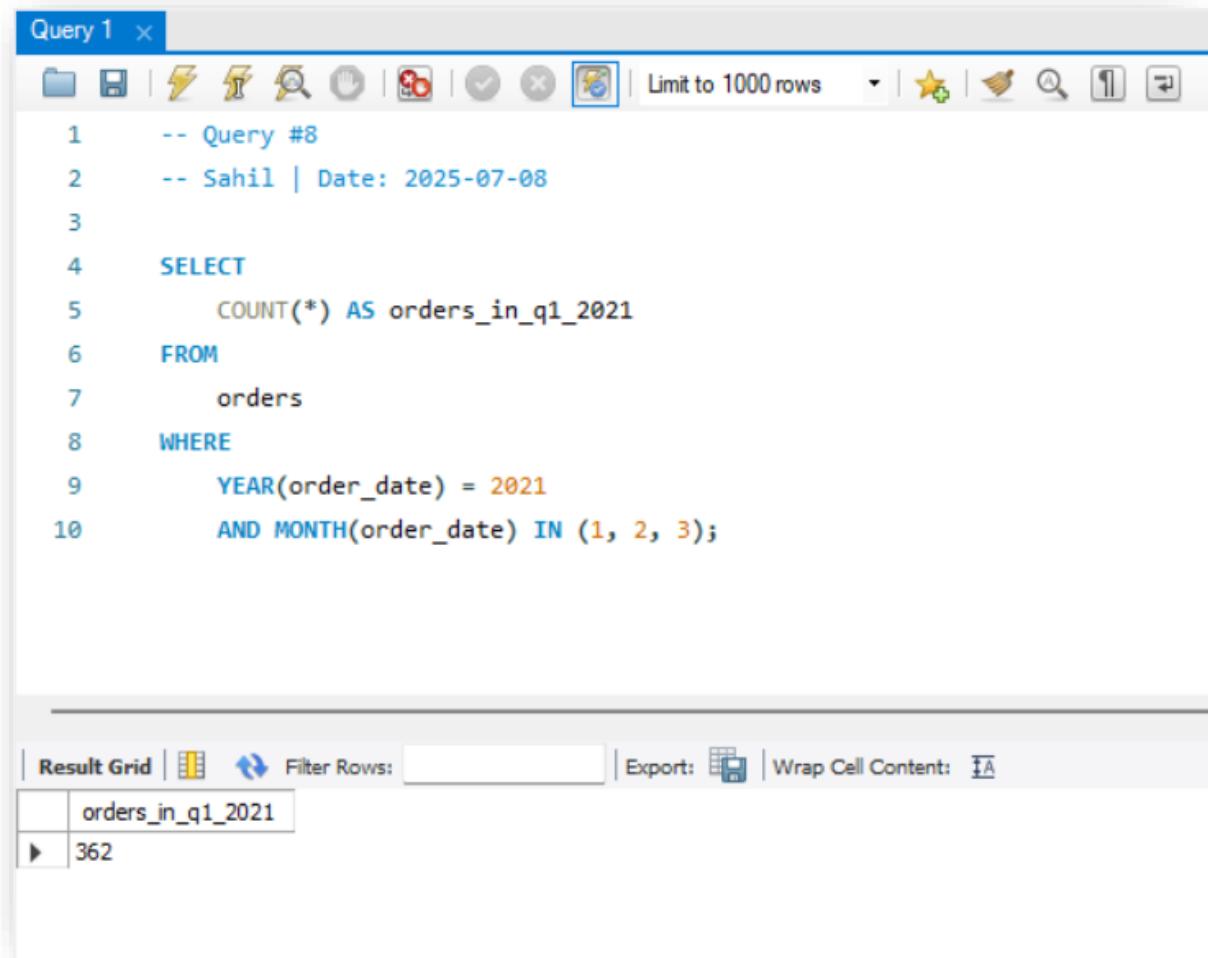
- A snapshot of **which customer segments dominate** the user base.
- Insights to focus marketing or product targeting toward the **largest segment**.

Question Query 8

How many orders were placed in the **first quarter of 2021**?

 (Q1 = January, February, March)

 Use a **column alias** in the result.



The screenshot shows a database query editor window titled "Query 1". The query itself is:

```
1 -- Query #8
2 -- Sahil | Date: 2025-07-08
3
4 SELECT
5     COUNT(*) AS orders_in_q1_2021
6 FROM
7     orders
8 WHERE
9     YEAR(order_date) = 2021
10    AND MONTH(order_date) IN (1, 2, 3);
```

Below the query, the results are displayed in a "Result Grid". The grid has one row and two columns. The first column contains the alias "orders_in_q1_2021" and the second column contains the value "362".

	orders_in_q1_2021
▶	362

What This Query Does:

- Filters only those orders placed in the **year 2021**.
- Then filters down to the **first quarter**: January, February, March.
- Uses a column alias `orders_in_q1_2021` for clarity.

Insight Gained:

This gives TerpBuy:

- A clear metric of **sales volume for Q1 2021**.
- Useful for comparing against other quarters or assessing **seasonal performance**.

Question Query 9

List in alphabetical order all **states** that support **multiple customer segments**.

Query 1

```
1 -- Query #9 (with full state names)
2 -- Sahil | Date: 2025-07-08
3
4 SELECT
5     CASE state
6         WHEN 'AL' THEN 'Alabama'
7         WHEN 'AK' THEN 'Alaska'
8         WHEN 'AZ' THEN 'Arizona'
9         WHEN 'AR' THEN 'Arkansas'
10        WHEN 'CA' THEN 'California'
11        WHEN 'CO' THEN 'Colorado'
12        WHEN 'CT' THEN 'Connecticut'
13        WHEN 'DE' THEN 'Delaware'
```

Result Grid

full_state_name
Arizona
Arkansas
California
Colorado
Connecticut
DC
Delaware
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky

Result 7

Query 1

```
53     WHEN 'WI' THEN 'Wisconsin'
54     WHEN 'WV' THEN 'West Virginia'
55     WHEN 'WY' THEN 'Wyoming'
56     ELSE state
57 END AS full_state_name
58 FROM
59     customer
60 GROUP BY
61     state
62 HAVING
63     COUNT(DISTINCT segment) > 1
64 ORDER BY
65     full_state_name;
```

Result Grid

full_state_name
Arizona
Arkansas
California
Colorado
Connecticut
DC
Delaware
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky

Result 7

🧠 What This Query Does:

- Groups customers by state
- Uses COUNT(DISTINCT segment) to find how many **unique segments** exist in each state
- Filters to only those states with **more than one segment**
- Sorts the list **alphabetically**

💡 Insight Gained:

This helps TerpBuy:

- Discover states with **diverse customer bases**
- Target marketing efforts in states that cater to **both corporate and consumer clients**
- Plan **multi-segment promotions** where both types coexist

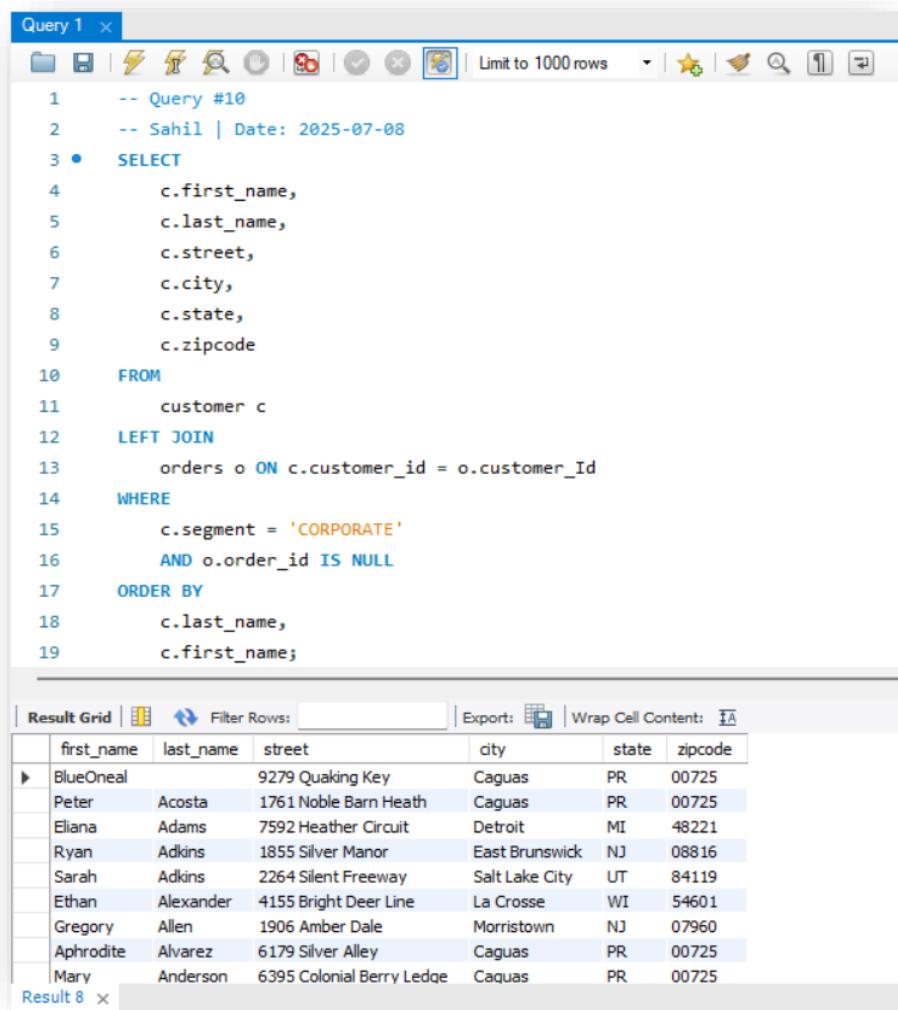
Question Query 10

To help the commercial sales department with its marketing,

Find all **customers in the corporate segment** who have **not placed any orders** Include:

- First name
- Last name
- Street
- City
- State
- Zip code

Sort by **last name**, then **first name**



```
Query 1 x
1 -- Query #10
2 -- Sahil | Date: 2025-07-08
3 • SELECT
4     c.first_name,
5     c.last_name,
6     c.street,
7     c.city,
8     c.state,
9     c.zipcode
10    FROM
11        customer c
12    LEFT JOIN
13        orders o ON c.customer_id = o.customer_Id
14    WHERE
15        c.segment = 'CORPORATE'
16        AND o.order_id IS NULL
17    ORDER BY
18        c.last_name,
19        c.first_name
```

	first_name	last_name	street	city	state	zipcode
▶	BlueOneal		9279 Quaking Key	Caguas	PR	00725
	Peter	Acosta	1761 Noble Barn Heath	Caguas	PR	00725
	Eliana	Adams	7592 Heather Circuit	Detroit	MI	48221
	Ryan	Adkins	1855 Silver Manor	East Brunswick	NJ	08816
	Sarah	Adkins	2264 Silent Freeway	Salt Lake City	UT	84119
	Ethan	Alexander	4155 Bright Deer Line	La Crosse	WI	54601
	Gregory	Allen	1906 Amber Dale	Morristown	NJ	07960
	Aphrodite	Alvarez	6179 Silver Alley	Caguas	PR	00725
	Mary	Anderson	6395 Colonial Berry Ledge	Caguas	PR	00725

Result 8 x

What This Query Does:

- Filters for **Corporate segment** customers only
- Performs a LEFT JOIN with orders
- Uses o.order_id IS NULL to find **those who haven't placed any orders**
- Outputs clean customer contact info for **outreach**
- Sorted by name for easy lookup

Insight Gained:

This gives TerpBuy:

- A **qualified lead list** for the commercial sales team
- Opportunities to send **exclusive promotions**, follow-ups, or **onboarding offers**
- The ability to **activate dormant customers** with zero sales so far

Question Query 11

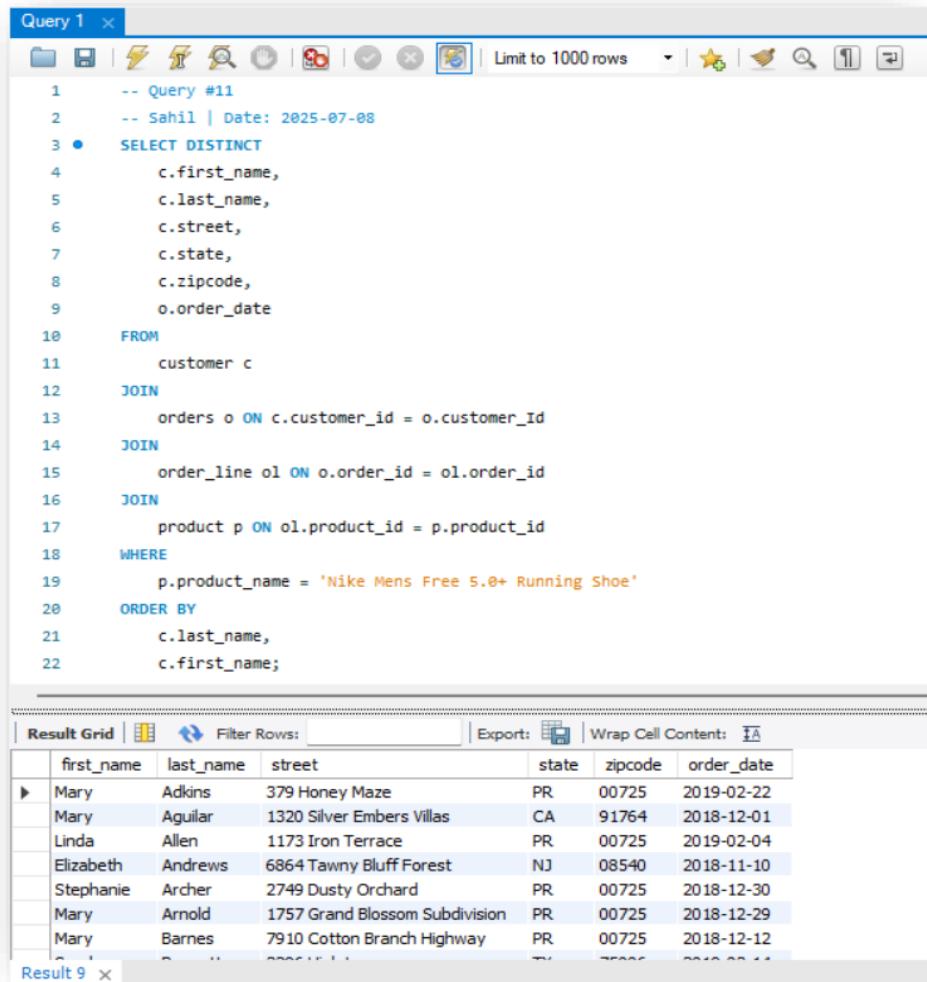
The product **Nike Men's Free 5.0+ Running Shoe** has been recalled.

TerpBuy wants to offer **one discount coupon per customer** who **purchased it**.

Find all **orders** that included this product, and return:

- Customer **first name, last name**
- **Street, state, zip code**
- **Order date**

Do not repeat customers — only one row per customer



```
Query 1
1 -- Query #11
2 -- sahil | Date: 2025-07-08
3 • SELECT DISTINCT
4     c.first_name,
5     c.last_name,
6     c.street,
7     c.state,
8     c.zipcode,
9     o.order_date
10    FROM
11        customer c
12    JOIN
13        orders o ON c.customer_id = o.customer_Id
14    JOIN
15        order_line ol ON o.order_id = ol.order_id
16    JOIN
17        product p ON ol.product_id = p.product_id
18    WHERE
19        p.product_name = 'Nike Mens Free 5.0+ Running Shoe'
20    ORDER BY
21        c.last_name,
22        c.first_name;
```

The screenshot shows a database query editor window titled "Query 1". The query itself is a SELECT DISTINCT statement that joins four tables: customer, orders, order_line, and product. It filters for the product "Nike Mens Free 5.0+ Running Shoe" and orders by customer last name and first name. Below the query is a "Result Grid" showing the output. The grid has columns for first_name, last_name, street, state, zipcode, and order_date. There are 9 rows of data, each representing a unique customer who purchased the specified product.

first_name	last_name	street	state	zipcode	order_date
Mary	Adkins	379 Honey Maze	PR	00725	2019-02-22
Mary	Aguilar	1320 Silver Embers Villas	CA	91764	2018-12-01
Linda	Allen	1173 Iron Terrace	PR	00725	2019-02-04
Elizabeth	Andrews	6864 Tawny Bluff Forest	NJ	08540	2018-11-10
Stephanie	Archer	2749 Dusty Orchard	PR	00725	2018-12-30
Mary	Arnold	1757 Grand Blossom Subdivision	PR	00725	2018-12-29
Mary	Barnes	7910 Cotton Branch Highway	PR	00725	2018-12-12

What This Query Does:

- Joins customer, orders, order_line, and product tables
- Filters only the product named **Nike Mens Free 5.0+ Running Shoe**
- Selects one row **per customer** using DISTINCT
- Gives all needed shipping and contact info
- Sorted by name for easy referencing

Insight Gained:

This helps TerpBuy:

- Quickly **identify impacted customers** from the recall
- Send out **targeted discount coupons**
- Avoid duplicates by ensuring only **one entry per customer**

Question Query 12

Premium customers are those who placed **orders with amounts greater than the average order amount**.

For each such order, find:

- Customer **first name, last name**
- The **order amount**

Prep Step:

- We'll use the **order_line** table to **calculate total order amounts**.
- Then compute the **average**.
- Return only the orders **above average**.
- Include the customer's name using a join.

```
Query 1 x
1 -- Query #12
2 -- Sahil | Date: 2025-07-08
3 • SELECT
4     c.first_name,
5     c.last_name,
6     o.order_id,
7     SUM(ol.total_price) AS order_amount
8 FROM
9     orders o
10 JOIN
11     customer c ON o.customer_Id = c.customer_id
12 JOIN
13     order_line ol ON o.order_id = ol.order_id
14 GROUP BY
15     o.order_id, c.first_name, c.last_name
16 HAVING
```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result 10 x

first_name	last_name	order_id	order_amount
Austin	Weiss	24121	1599.87
Albert	Contreras	23045	1529.90
Diana	Hale	28644	1529.88
Mary	Turner	23470	1519.88
Mary	Frazier	27552	1509.78
Jael	Goff	74496	1500.00
Eve	Stone	74502	1500.00
Miranda	Graham	74515	1500.00
Latifah	Boyle	74516	1500.00
Yolanda	Farrell	74534	1500.00
Fatima	Kline	74535	1500.00
Gemma	Silva	74536	1500.00

```
Query 1 x
11     customer c ON o.customer_Id = c.customer_id
12 JOIN
13     order_line ol ON o.order_id = ol.order_id
14 GROUP BY
15     o.order_id, c.first_name, c.last_name
16 HAVING
17     order_amount > (
18         SELECT AVG(total_order)
19         FROM (
20             SELECT SUM(total_price) AS total_order
21             FROM order_line
22             GROUP BY order_id
23         ) AS avg_table
24     )
25 ORDER BY
26     order_amount DESC;
```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result 10 x

first_name	last_name	order_id	order_amount
Austin	Weiss	24121	1599.87
Albert	Contreras	23045	1529.90
Diana	Hale	28644	1529.88
Mary	Turner	23470	1519.88
Mary	Frazier	27552	1509.78
Jael	Goff	74496	1500.00
Eve	Stone	74502	1500.00
Miranda	Graham	74515	1500.00
Latifah	Boyle	74516	1500.00
Yolanda	Farrell	74534	1500.00
Fatima	Kline	74535	1500.00
Gemma	Silva	74536	1500.00

What This Query Does:

- Joins customers, orders, and order lines.
- Calculates total order amount per order.
- Uses a subquery to compute the **average order amount**.
- Filters to keep only **above-average** orders.
- Sorts by the highest order amounts first.

Insight Gained:

This identifies **high-spending customers**, which helps TerpBuy:

- Offer loyalty rewards or upsell opportunities.
 - Segment premium clients for **targeted VIP campaigns**.
 - Understand what price ranges trigger strong spending.
-

End Note

This project has given me the opportunity to apply my SQL skills in a real-world business context. Through a structured approach, I explored TerpBuy's customer behaviors, product performance, shipping operations, and sales trends — all directly from a relational database.

Each query was carefully designed to not only meet technical requirements but also deliver meaningful business insights that management can act upon. From identifying premium customers and unsold inventory, to detecting shipping issues and analyzing customer segmentation, this assignment has shown how powerful structured query logic can be when tied to business strategy.

Working on this project has strengthened my confidence in writing advanced SQL queries, using joins effectively, filtering data with logic, and delivering data-driven conclusions. I've learned the value of not just *extracting* data, but also *understanding and presenting* it in ways that drive action.

I look forward to carrying this experience forward into more complex data projects, and I sincerely thank you for reviewing my work.

Prepared by: Sahil Verma "Data Analyst"

Date: July 21, 2025