

# PROJECT 1

## ShopFast

### STEP 1: Creating a new Database,


create database ShopFast;

### STEP 2: using the new database,

use ShopFast;

### STEP 3: Inserting the values through given csv files,

Import Flat File 'ShopFast'

 **Modify Columns**

Introduction

Specify Input File

Preview Data

Modify Columns

Summary

Results

Help

**Modify Columns**

This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.

Column Name	Data Type	Primary Key	<input checked="" type="checkbox"/> Allow Nulls	
customer_id	tinyint	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
name	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
email	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
city	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
signup_date	date	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Row granularity of error reporting (performance impact with smaller ranges) 

No Range

< Previous

Next >

Close

Import Flat File 'ShopFast'

**Modify Columns**

Introduction  
Specify Input File  
Preview Data  
**Modify Columns**  
Summary  
Results

Help

**Modify Columns**  
This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.

Column Name	Data Type	Primary Key	<input checked="" type="checkbox"/> Allow Nulls
order_item_id	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>
order_id	tinyint	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
product_id	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>
quantity	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>
price_per_unit	float	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Row granularity of error reporting (performance impact with smaller ranges)
No Range

< Previous
Next >
Cancel

Import Flat File 'ShopFast'

**Modify Columns**

Introduction  
Specify Input File  
Preview Data  
Modify Columns  
Summary  
**Results**

Help


**Modify Columns**  
This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.

Column Name	Data Type	Primary Key	<input checked="" type="checkbox"/> Allow Nulls
order_id	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>
customer_id	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>
order_date	date	<input type="checkbox"/>	<input checked="" type="checkbox"/>
delivery_date	date	<input type="checkbox"/>	<input checked="" type="checkbox"/>
status	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
total_amount	float	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Row granularity of error reporting (performance impact with smaller ranges)
No Range

< Previous
Next >
Close

Import Flat File 'ShopFast'

**Modify Columns**

Introduction

Specify Input File

Preview Data

Modify Columns

Summary

Results

Help

**Modify Columns**  
This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.

Column Name	Data Type	Primary Key	<input checked="" type="checkbox"/> Allow Nulls	
product_id	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
product_name	nvarchar(50)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
category	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
launch_date	date	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
stock_quantity	tinyint	<input type="checkbox"/>	<input checked="" type="checkbox"/>	


Row granularity of error reporting (performance impact with smaller ranges) No Range

< Previous

Next >

Cancel

Import Flat File 'ShopFast'

**Results**

Introduction

Specify Input File


Preview Data

Modify Columns


Summary

Results

Help

 **Operation Complete**

Summary:

Name	Result
 Insert Data	<a href="#">Success</a>

< Previous

Next >

Close

#### STEP 4: Check whether all the data's are inserted properly,

select \* from CUSTOMERS;

select \* from ORDER\_ITEMS;

select \* from ORDERS;

select \* from PRODUCTS;

SQLQuery3.sql - HP...AVILION\sjlak (72))\* X 2\_6\_SQL\_Practice\_te...AVILION\sjlak (79)

```
create database ShopFast;

use ShopFast;

select * from CUSTOMERS;

select * from ORDER_ITEMS;

select * from ORDERS;

select * from PRODUCTS;
```

90 %

Results Messages

	customer_id	name	email	city	signup_date
1	1	Alice	alice@example.com	Delhi	2022-01-15
2	2	Bob	bob@example.com	Mumbai	2022-02-20
3	3	Charlie	charlie@example.com	Bangalore	2022-03-10
4	4	Diana	diana@example.com	Hyderabad	2022-04-01
5	5	Ethan	ethan@example.com	Chennai	2022-05-25

	order_item_id	order_id	product_id	quantity	price_per_unit
1	1	101	1	2	50.25
2	2	102	2	1	200
3	3	103	1	1	175
4	4	104	3	3	100
5	5	105	4	1	120

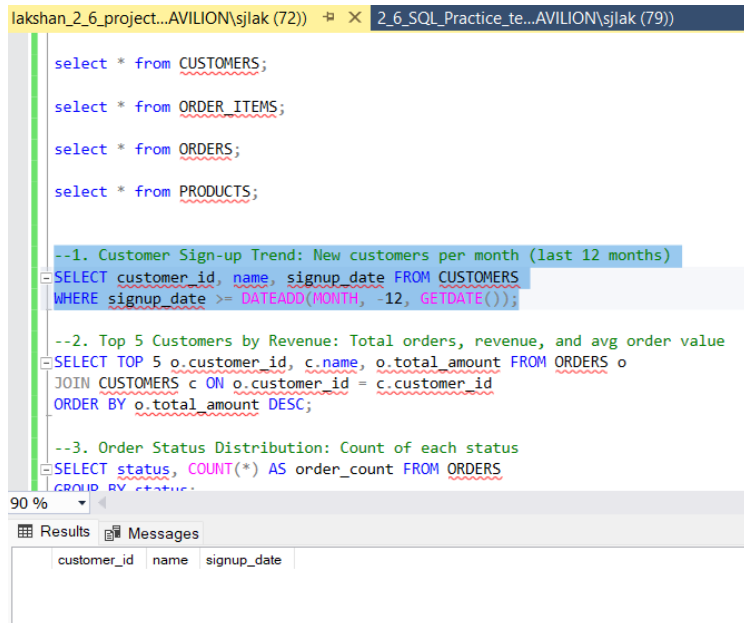
	order_id	customer_id	order_date	delivery_date	status	total_amount
1	101	1	2023-01-10	2023-01-15	Delivered	150.5
2	102	2	2023-02-12	2023-02-18	Delivered	200
3	103	1	2023-02-28	2023-03-05	Returned	175
4	104	3	2023-03-05	2023-03-08	Delivered	300
5	105	4	2023-03-15	NULL	Pending	120

	product_id	product_name	category	launch_date	stock_quantity
1	2	Bluetooth Speaker	Electronics	2022-01-03	50
2	4	Notebook	Stationery	2022-01-07	200
3	1	Pen Drive	Electronics	2022-01-01	100
4	3	Wireless Mouse	Electronics	2022-01-05	75

## STEP 5: Answers for all Problem Set

--1. Customer Sign-up Trend: New customers per month (last 12 months)

```
SELECT customer_id, name, signup_date FROM CUSTOMERS
WHERE signup_date >= DATEADD(MONTH, -12, GETDATE());
```



The screenshot shows a SQL Server Enterprise Manager interface. At the top, there are two tabs: 'lakshan\_2\_6\_project...AVILION\sjlak (72)' and '2\_6\_SQL\_Practice\_te...AVILION\sjlak (79)'. The active window contains the following SQL code:

```
select * from CUSTOMERS;
select * from ORDER_ITEMS;
select * from ORDERS;
select * from PRODUCTS;

--1. Customer Sign-up Trend: New customers per month (last 12 months)
SELECT customer_id, name, signup_date FROM CUSTOMERS
WHERE signup_date >= DATEADD(MONTH, -12, GETDATE());

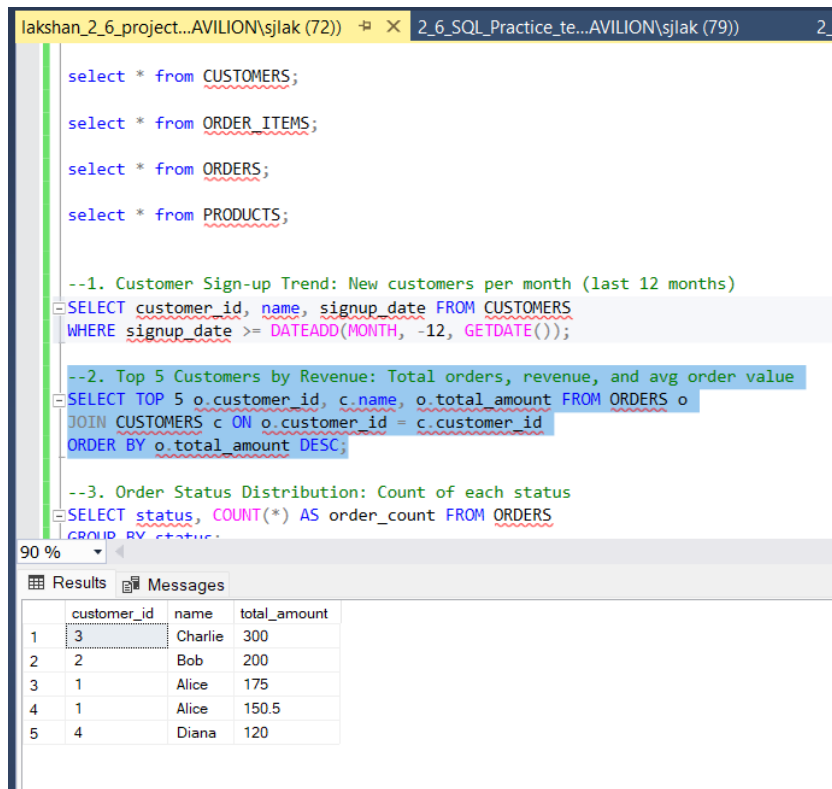
--2. Top 5 Customers by Revenue: Total orders, revenue, and avg order value
SELECT TOP 5 o.customer_id, c.name, o.total_amount FROM ORDERS o
JOIN CUSTOMERS c ON o.customer_id = c.customer_id
ORDER BY o.total_amount DESC;

--3. Order Status Distribution: Count of each status
SELECT status, COUNT(*) AS order_count FROM ORDERS
GROUP BY status;
```

Below the code, there is a 'Results' tab showing a grid with the following columns: 'customer\_id', 'name', and 'signup\_date'. The grid is currently empty.

--2. Top 5 Customers by Revenue: Total orders, revenue, and avg order value

```
SELECT TOP 5 o.customer_id, c.name, o.total_amount FROM ORDERS o
JOIN CUSTOMERS c ON o.customer_id = c.customer_id
ORDER BY o.total_amount DESC;
```

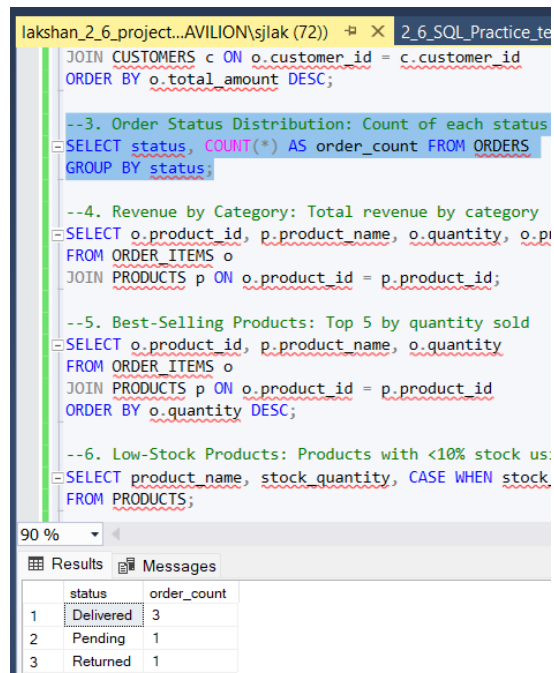


The screenshot shows the same SQL Server Enterprise Manager interface as before. The active window contains the same SQL code as in the previous screenshot. The 'Results' tab now shows a grid with the following columns: 'customer\_id', 'name', and 'total\_amount'. The grid contains 5 rows of data:

	customer_id	name	total_amount
1	3	Charlie	300
2	2	Bob	200
3	1	Alice	175
4	1	Alice	150.5
5	4	Diana	120

--3. Order Status Distribution: Count of each status

```
SELECT status, COUNT(*) AS order_count FROM ORDERS  
GROUP BY status;
```



The screenshot shows a SQL Server Enterprise Manager window with the following SQL code and results:

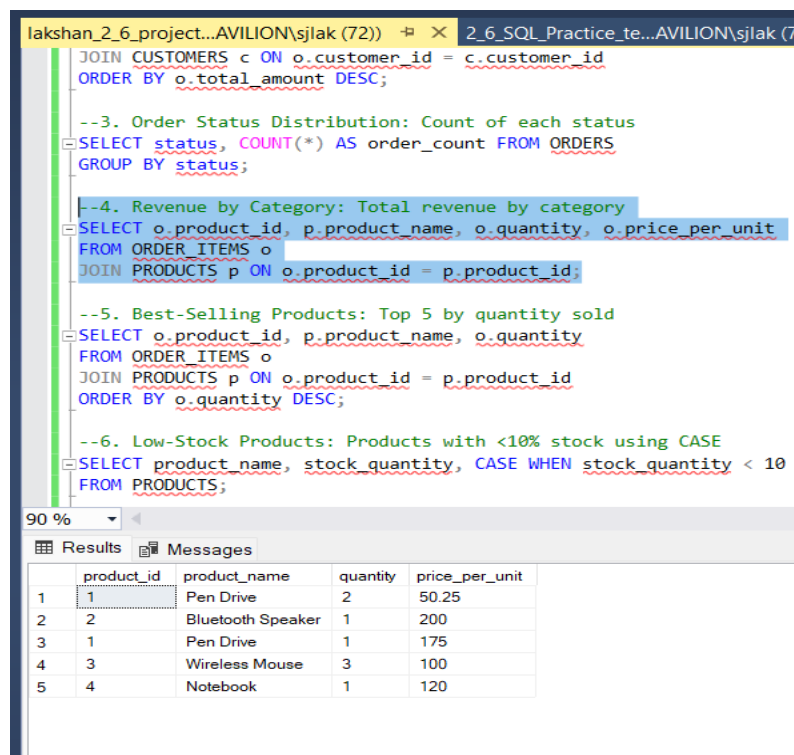
```
JOIN CUSTOMERS c ON o.customer_id = c.customer_id  
ORDER BY o.total amount DESC;  
  
--3. Order Status Distribution: Count of each status  
SELECT status, COUNT(*) AS order_count FROM ORDERS  
GROUP BY status;  
  
--4. Revenue by Category: Total revenue by category  
SELECT o.product_id, p.product_name, o.quantity, o.p  
FROM ORDER_ITEMS o  
JOIN PRODUCTS p ON o.product_id = p.product_id;  
  
--5. Best-Selling Products: Top 5 by quantity sold  
SELECT o.product_id, p.product_name, o.quantity  
FROM ORDER_ITEMS o  
JOIN PRODUCTS p ON o.product_id = p.product_id  
ORDER BY o.quantity DESC;  
  
--6. Low-Stock Products: Products with <10% stock us  
SELECT product_name, stock_quantity, CASE WHEN stock
```

The results pane shows the following data:

	status	order_count
1	Delivered	3
2	Pending	1
3	Returned	1

--4. Revenue by Category: Total revenue by category

```
SELECT o.product_id, p.product_name, o.quantity, o.price_per_unit  
FROM ORDER_ITEMS o  
JOIN PRODUCTS p ON o.product_id = p.product_id;
```



The screenshot shows a SQL Server Enterprise Manager window with the following SQL code and results:

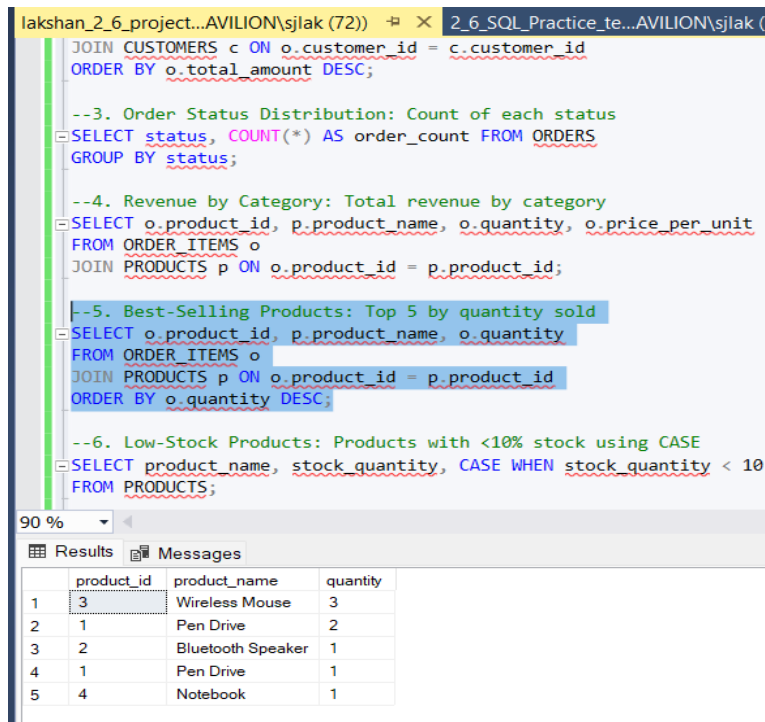
```
JOIN CUSTOMERS c ON o.customer_id = c.customer_id  
ORDER BY o.total amount DESC;  
  
--3. Order Status Distribution: Count of each status  
SELECT status, COUNT(*) AS order_count FROM ORDERS  
GROUP BY status;  
  
--4. Revenue by Category: Total revenue by category  
SELECT o.product_id, p.product_name, o.quantity, o.price_per_unit  
FROM ORDER_ITEMS o  
JOIN PRODUCTS p ON o.product_id = p.product_id;  
  
--5. Best-Selling Products: Top 5 by quantity sold  
SELECT o.product_id, p.product_name, o.quantity  
FROM ORDER_ITEMS o  
JOIN PRODUCTS p ON o.product_id = p.product_id  
ORDER BY o.quantity DESC;  
  
--6. Low-Stock Products: Products with <10% stock using CASE  
SELECT product_name, stock_quantity, CASE WHEN stock_quantity < 10  
FROM PRODUCTS;
```

The results pane shows the following data:

	product_id	product_name	quantity	price_per_unit
1	1	Pen Drive	2	50.25
2	2	Bluetooth Speaker	1	200
3	1	Pen Drive	1	175
4	3	Wireless Mouse	3	100
5	4	Notebook	1	120

--5. Best-Selling Products: Top 5 by quantity sold

```
SELECT o.product_id, p.product_name, o.quantity
FROM ORDER_ITEMS o
JOIN PRODUCTS p ON o.product_id = p.product_id
ORDER BY o.quantity DESC;
```



The screenshot shows a SQL Developer window with the following SQL code:

```
JOIN CUSTOMERS c ON o.customer_id = c.customer_id
ORDER BY o.total_amount DESC;

--3. Order Status Distribution: Count of each status
SELECT status, COUNT(*) AS order_count FROM ORDERS
GROUP BY status;

--4. Revenue by Category: Total revenue by category
SELECT o.product_id, p.product_name, o.quantity, o.price_per_unit
FROM ORDER_ITEMS o
JOIN PRODUCTS p ON o.product_id = p.product_id;

--5. Best-Selling Products: Top 5 by quantity sold
SELECT o.product_id, p.product_name, o.quantity
FROM ORDER_ITEMS o
JOIN PRODUCTS p ON o.product_id = p.product_id
ORDER BY o.quantity DESC;

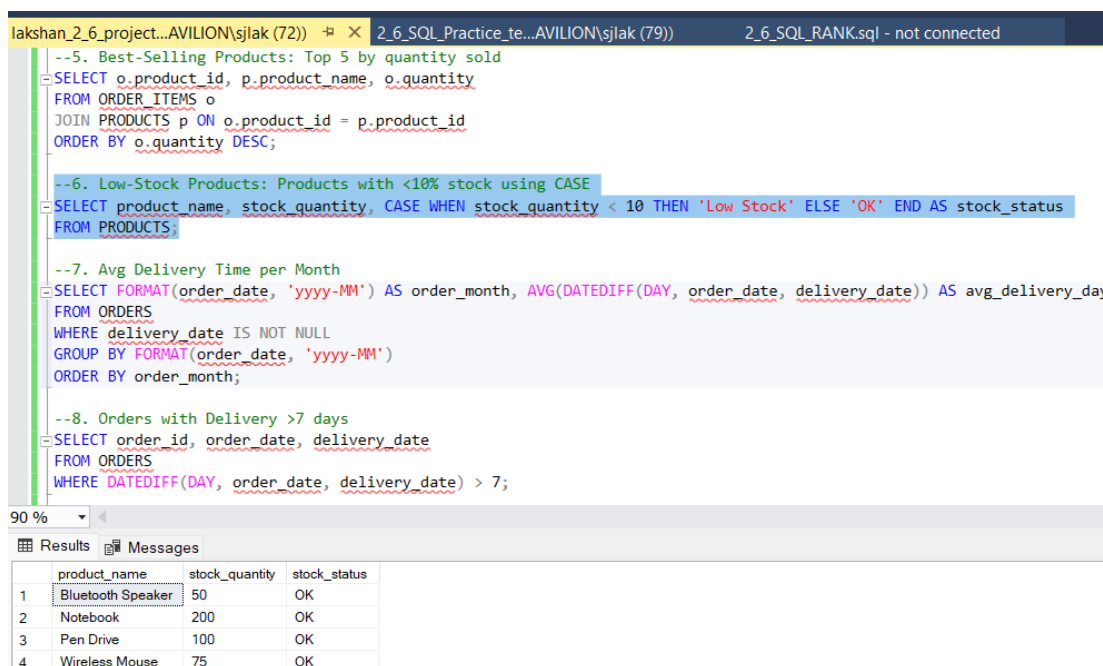
--6. Low-Stock Products: Products with <10% stock using CASE
SELECT product_name, stock_quantity, CASE WHEN stock_quantity < 10
FROM PRODUCTS;
```

The results grid shows the following data:

	product_id	product_name	quantity
1	3	Wireless Mouse	3
2	1	Pen Drive	2
3	2	Bluetooth Speaker	1
4	1	Pen Drive	1
5	4	Notebook	1

--6. Low-Stock Products: Products with <10% stock using CASE

```
SELECT product_name, stock_quantity, CASE WHEN stock_quantity < 10 THEN 'Low Stock' ELSE 'OK'
END AS stock_status
FROM PRODUCTS;
```



The screenshot shows a SQL Developer window with the following SQL code:

```
--5. Best-Selling Products: Top 5 by quantity sold
SELECT o.product_id, p.product_name, o.quantity
FROM ORDER_ITEMS o
JOIN PRODUCTS p ON o.product_id = p.product_id
ORDER BY o.quantity DESC;

--6. Low-Stock Products: Products with <10% stock using CASE
SELECT product_name, stock_quantity, CASE WHEN stock_quantity < 10 THEN 'Low Stock' ELSE 'OK' END AS stock_status
FROM PRODUCTS;

--7. Avg Delivery Time per Month
SELECT FORMAT(order_date, 'yyyy-MM') AS order_month, AVG(DATEDIFF(DAY, order_date, delivery_date)) AS avg_delivery_day
FROM ORDERS
WHERE delivery_date IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

--8. Orders with Delivery >7 days
SELECT order_id, order_date, delivery_date
FROM ORDERS
WHERE DATEDIFF(DAY, order_date, delivery_date) > 7;
```

The results grid shows the following data:

	product_name	stock_quantity	stock_status
1	Bluetooth Speaker	50	OK
2	Notebook	200	OK
3	Pen Drive	100	OK
4	Wireless Mouse	75	OK

### --7. Avg Delivery Time per Month

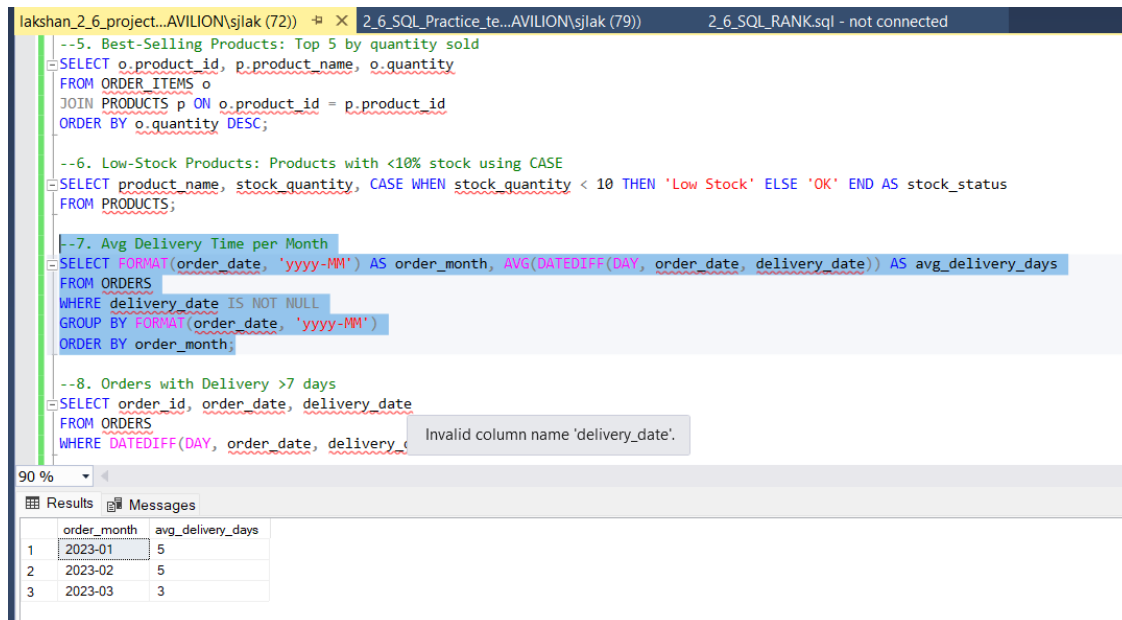
```
SELECT FORMAT(order_date, 'yyyy-MM') AS order_month, AVG(DATEDIFF(DAY, order_date,
delivery_date)) AS avg_delivery_days

FROM ORDERS

WHERE delivery_date IS NOT NULL

GROUP BY FORMAT(order_date, 'yyyy-MM')

ORDER BY order_month;
```



```
--5. Best-Selling Products: Top 5 by quantity sold
SELECT o.product_id, p.product_name, o.quantity
FROM ORDER_ITEMS o
JOIN PRODUCTS p ON o.product_id = p.product_id
ORDER BY o.quantity DESC;

--6. Low-Stock Products: Products with <10% stock using CASE
SELECT product_name, stock_quantity, CASE WHEN stock_quantity < 10 THEN 'Low Stock' ELSE 'OK' END AS stock_status
FROM PRODUCTS;

--7. Avg Delivery Time per Month
SELECT FORMAT(order_date, 'yyyy-MM') AS order_month, AVG(DATEDIFF(DAY, order_date, delivery_date)) AS avg_delivery_days
FROM ORDERS
WHERE delivery_date IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

--8. Orders with Delivery >7 days
SELECT order_id, order_date, delivery_date
FROM ORDERS
WHERE DATEDIFF(DAY, order_date, delivery_date) > 7;
```

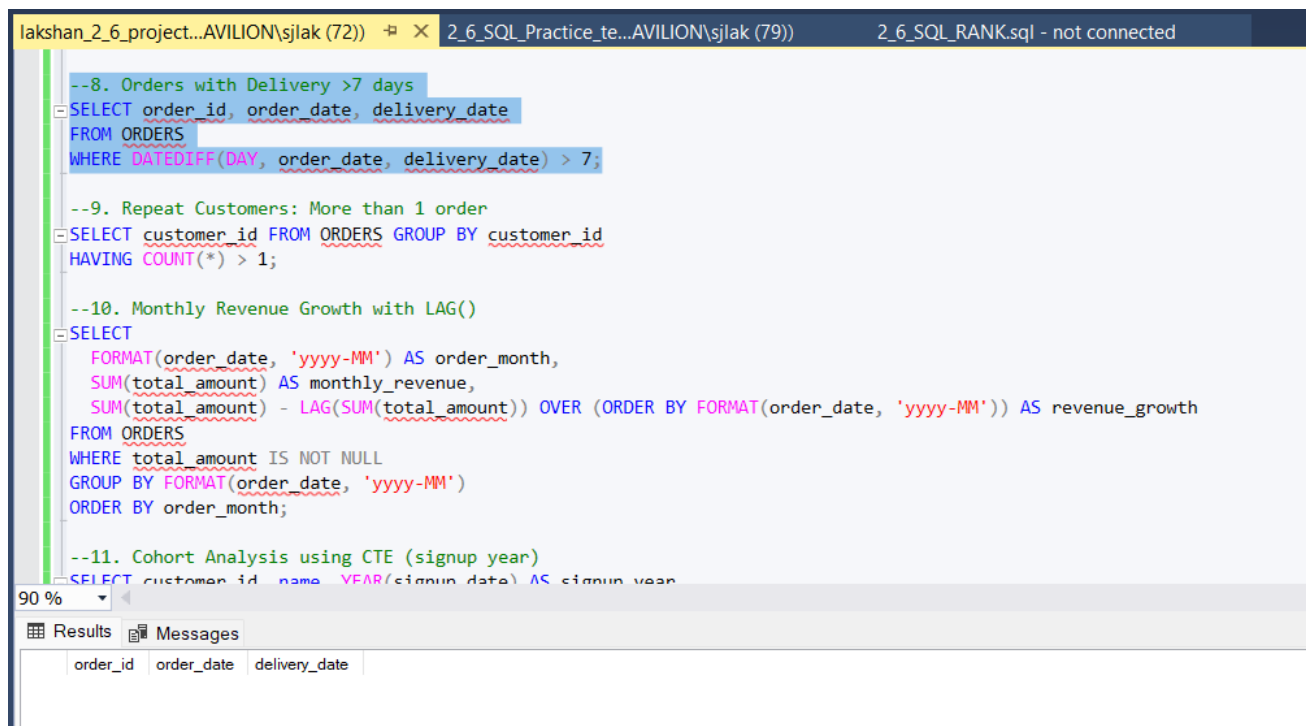
	order_month	avg_delivery_days
1	2023-01	5
2	2023-02	5
3	2023-03	3

### --8. Orders with Delivery >7 days

```
SELECT order_id, order_date, delivery_date

FROM ORDERS

WHERE DATEDIFF(DAY, order_date, delivery_date) > 7;
```



```
--8. Orders with Delivery >7 days
SELECT order_id, order_date, delivery_date
FROM ORDERS
WHERE DATEDIFF(DAY, order_date, delivery_date) > 7;

--9. Repeat Customers: More than 1 order
SELECT customer_id FROM ORDERS GROUP BY customer_id
HAVING COUNT(*) > 1;

--10. Monthly Revenue Growth with LAG()
SELECT
FORMAT(order_date, 'yyyy-MM') AS order_month,
SUM(total_amount) AS monthly_revenue,
SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY FORMAT(order_date, 'yyyy-MM')) AS revenue_growth
FROM ORDERS
WHERE total_amount IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

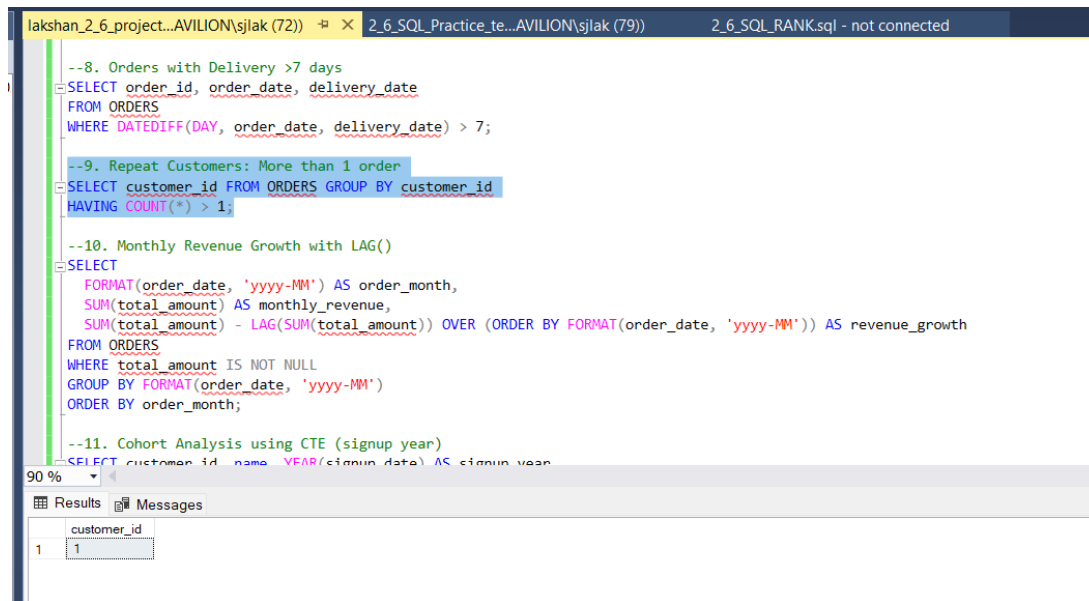
--11. Cohort Analysis using CTE (signup year)
SELECT customer_id, name, YEAR(signup_date) AS signup_year
```

order_id	order_date	delivery_date
1	2023-01-01	2023-01-06
2	2023-02-01	2023-02-06
3	2023-03-01	2023-03-04



--9. Repeat Customers: More than 1 order

```
SELECT customer_id FROM ORDERS GROUP BY customer_id
HAVING COUNT(*) > 1;
```



The screenshot shows a SQL Server Enterprise Manager interface. The top bar indicates the current database is 'lakshan\_2\_6\_project...AVILION\sjlak (72)' and the connection is '2\_6\_SQL\_Practice\_te...AVILION\sjlak (79)'. The query window contains the following SQL code:

```
--8. Orders with Delivery >7 days
SELECT order_id, order_date, delivery_date
FROM ORDERS
WHERE DATEDIFF(DAY, order_date, delivery_date) > 7;

--9. Repeat Customers: More than 1 order
SELECT customer_id FROM ORDERS GROUP BY customer_id
HAVING COUNT(*) > 1;

--10. Monthly Revenue Growth with LAG()
SELECT
    FORMAT(order_date, 'yyyy-MM') AS order_month,
    SUM(total_amount) AS monthly_revenue,
    SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY FORMAT(order_date, 'yyyy-MM')) AS revenue_growth
FROM ORDERS
WHERE total_amount IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

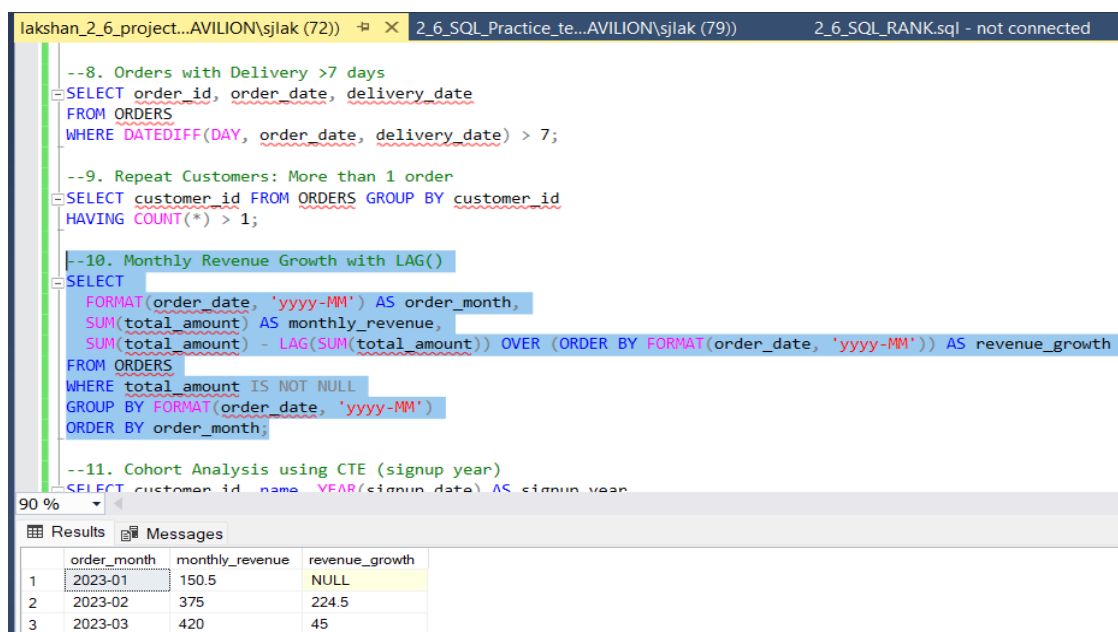
--11. Cohort Analysis using CTE (signup year)
SELECT customer_id, name, YEAR(signup_date) AS signup_year
```

The Results pane at the bottom shows the output of the third query (Repeat Customers):

customer_id
1

--10. Monthly Revenue Growth with LAG()

```
SELECT
    FORMAT(order_date, 'yyyy-MM') AS order_month,
    SUM(total_amount) AS monthly_revenue,
    SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY FORMAT(order_date, 'yyyy-MM')) AS
revenue_growth
FROM ORDERS
WHERE total_amount IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;
```

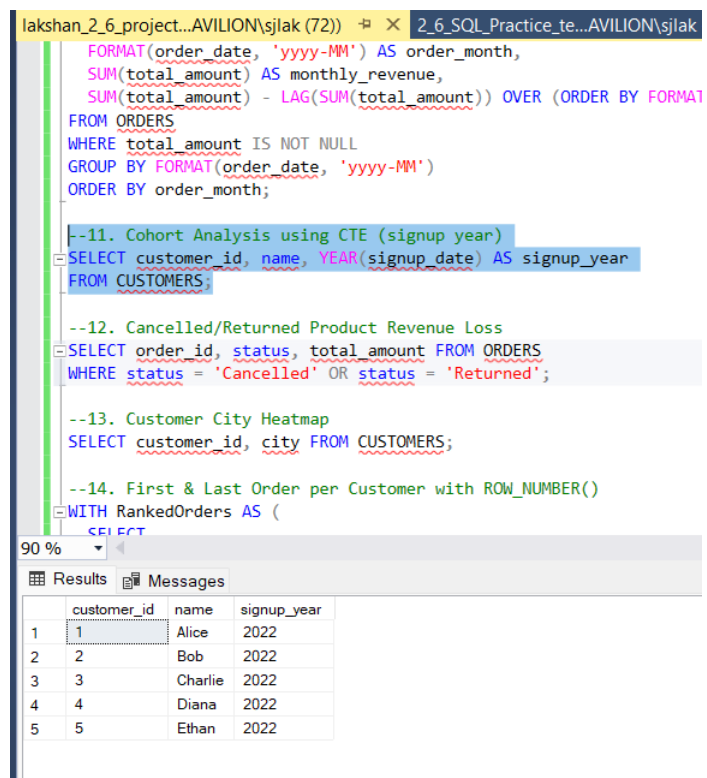


The screenshot shows the same SQL Server Enterprise Manager interface. The query window contains the same SQL code as the previous screenshot. The Results pane at the bottom shows the output of the fourth query (Monthly Revenue Growth with LAG()):

order_month	monthly_revenue	revenue_growth
2023-01	150.5	NULL
2023-02	375	224.5
2023-03	420	45

#### --11. Cohort Analysis using CTE (signup year)

```
SELECT customer_id, name, YEAR(signup_date) AS signup_year
FROM CUSTOMERS;
```



The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains several SQL queries, with the first one highlighted. The results pane shows the output of the first query, which is a table with 5 rows and 3 columns: customer\_id, name, and signup\_year.

```
lakshan_2_6_project...AVILION\sqlak (72)) 2_6_SQL_Practice_te...AVILION
FORMAT(order_date, 'yyyy-MM') AS order_month,
SUM(total_amount) AS monthly_revenue,
SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY FORMAT
FROM ORDERS
WHERE total_amount IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

--11. Cohort Analysis using CTE (signup year)
SELECT customer_id, name, YEAR(signup_date) AS signup_year
FROM CUSTOMERS;

--12. Cancelled/Returned Product Revenue Loss
SELECT order_id, status, total_amount FROM ORDERS
WHERE status = 'Cancelled' OR status = 'Returned';

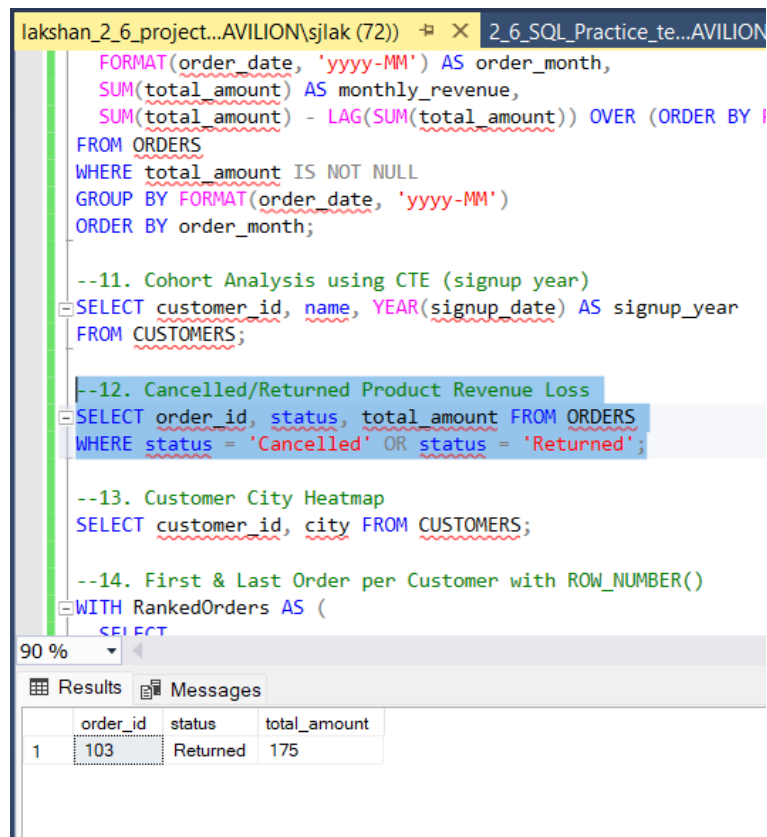
--13. Customer City Heatmap
SELECT customer_id, city FROM CUSTOMERS;

--14. First & Last Order per Customer with ROW_NUMBER()
WITH RankedOrders AS (
  SELECT
```

customer_id	name	signup_year
1	Alice	2022
2	Bob	2022
3	Charlie	2022
4	Diana	2022
5	Ethan	2022

#### --12. Cancelled/Returned Product Revenue Loss

```
SELECT order_id, status, total_amount FROM ORDERS
WHERE status = 'Cancelled' OR status = 'Returned';
```



The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains several SQL queries, with the second one highlighted. The results pane shows the output of the second query, which is a table with 1 row and 3 columns: order\_id, status, and total\_amount.

```
lakshan_2_6_project...AVILION\sqlak (72)) 2_6_SQL_Practice_te...AVILION
FORMAT(order_date, 'yyyy-MM') AS order_month,
SUM(total_amount) AS monthly_revenue,
SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY I
FROM ORDERS
WHERE total_amount IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

--11. Cohort Analysis using CTE (signup year)
SELECT customer_id, name, YEAR(signup_date) AS signup_year
FROM CUSTOMERS;

--12. Cancelled/Returned Product Revenue Loss
SELECT order_id, status, total_amount FROM ORDERS
WHERE status = 'Cancelled' OR status = 'Returned';

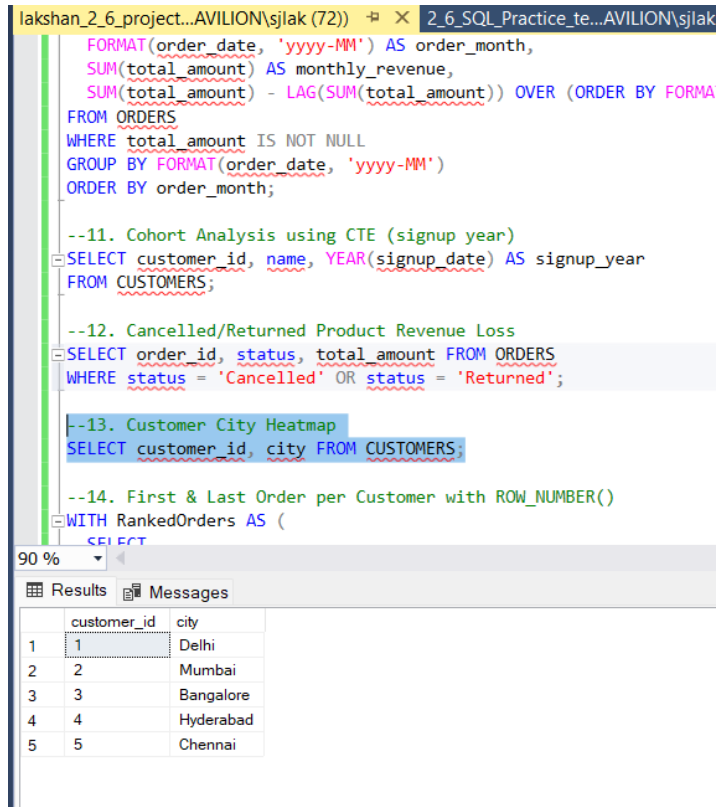
--13. Customer City Heatmap
SELECT customer_id, city FROM CUSTOMERS;

--14. First & Last Order per Customer with ROW_NUMBER()
WITH RankedOrders AS (
  SELECT
```

order_id	status	total_amount
103	Returned	175

### --13. Customer City Heatmap

```
SELECT customer_id, city FROM CUSTOMERS;
```



The screenshot shows a SQL Studio window with the following content:

```
lakshan_2_6_project...AVILION\sjlak (72) 2_6_SQL_Practice_te...AVILION\sjlak
FORMAT(order_date, 'yyyy-MM') AS order_month,
SUM(total_amount) AS monthly_revenue,
SUM(total_amount) - LAG(SUM(total_amount)) OVER (ORDER BY FORMA
FROM ORDERS
WHERE total_amount IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MM')
ORDER BY order_month;

--11. Cohort Analysis using CTE (signup year)
SELECT customer_id, name, YEAR(signup_date) AS signup_year
FROM CUSTOMERS;

--12. Cancelled/Returned Product Revenue Loss
SELECT order_id, status, total_amount FROM ORDERS
WHERE status = 'Cancelled' OR status = 'Returned';

--13. Customer City Heatmap
SELECT customer_id, city FROM CUSTOMERS;

--14. First & Last Order per Customer with ROW_NUMBER()
WITH RankedOrders AS (
  SELECT
```

Below the queries, there is a 'Results' tab showing a table with 5 rows and 2 columns:

	customer_id	city
1	1	Delhi
2	2	Mumbai
3	3	Bangalore
4	4	Hyderabad
5	5	Chennai

### --14. First & Last Order per Customer with ROW\_NUMBER()

```
WITH RankedOrders AS (
```

```
  SELECT
```

```
    customer_id,
```

```
    order_id,
```

```
    order_date,
```

```
    ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date ASC) AS rn_asc,
```

```
    ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date DESC) AS rn_desc
```

```
  FROM ORDERS
```

```
)
```

```
SELECT customer_id, order_id, order_date, 'First Order' AS order_type
```

```
FROM RankedOrders WHERE rn_asc = 1
```

```
UNION
```

```
SELECT customer_id, order_id, order_date, 'Last Order'
```

```
FROM RankedOrders WHERE rn_desc = 1;
```

```
--13. Customer City Heatmap
SELECT customer_id, city FROM CUSTOMERS;

--14. First & Last Order per Customer with ROW_NUMBER()
WITH RankedOrders AS (
    SELECT
        customer_id,
        order_id,
        order_date,
        ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date ASC) AS rn_asc,
        ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date DESC) AS rn_desc
    FROM ORDERS
)
SELECT customer_id, order_id, order_date, 'First Order' AS order_type
FROM RankedOrders WHERE rn_asc = 1
UNION
SELECT customer_id, order_id, order_date, 'Last Order'
FROM RankedOrders WHERE rn_desc = 1;

--15. NULL Handling: Orders with missing delivery/amount
SELECT order_id, delivery_date, total_amount FROM ORDERS
```

90 %

Results Messages

	customer_id	order_id	order_date	order_type
1	1	101	2023-01-10	First Order
2	1	103	2023-02-28	Last Order
3	2	102	2023-02-12	First Order
4	2	102	2023-02-12	Last Order
5	3	104	2023-03-05	First Order
6	3	104	2023-03-05	Last Order
7	4	105	2023-03-15	First Order
8	4	105	2023-03-15	Last Order

--15. NULL Handling: Orders with missing delivery/amount

SELECT order\_id, delivery\_date, total\_amount FROM ORDERS

WHERE delivery\_date IS NULL OR total\_amount IS NULL;

```
order_date,
ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date
ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date
FROM ORDERS
)
SELECT customer_id, order_id, order_date, 'First Order' AS order_type
FROM RankedOrders WHERE rn_asc = 1
UNION
SELECT customer_id, order_id, order_date, 'Last Order'
FROM RankedOrders WHERE rn_desc = 1;

--15. NULL Handling: Orders with missing delivery/amount
SELECT order_id, delivery_date, total_amount FROM ORDERS
WHERE delivery_date IS NULL OR total_amount IS NULL;
```

90 %

Results Messages

	order_id	delivery_date	total_amount
1	105	NULL	120