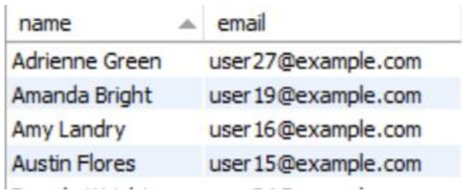
SQL Queries with Outputs:

Level 1: Basics

-- Query: 1. Retrieve customer names and emails for email marketing

```
SELECT name , email FROM customers;
```

Output:

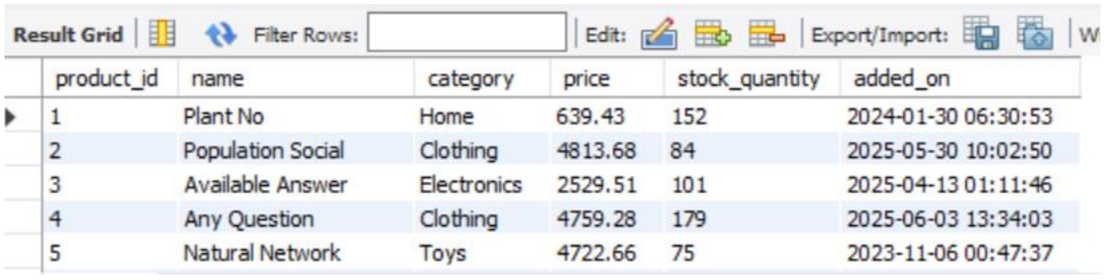


name	email
Adrienne Green	user27@example.com
Amanda Bright	user19@example.com
Amy Landry	user16@example.com
Austin Flores	user15@example.com

-- Query: 2. View complete product catalog with all available details

```
SELECT * FROM products;
```

Output:

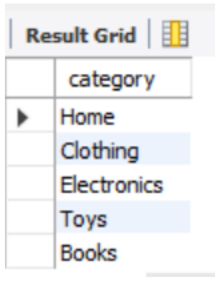


product_id	name	category	price	stock_quantity	added_on
1	Plant No	Home	639.43	152	2024-01-30 06:30:53
2	Population Social	Clothing	4813.68	84	2025-05-30 10:02:50
3	Available Answer	Electronics	2529.51	101	2025-04-13 01:11:46
4	Any Question	Clothing	4759.28	179	2025-06-03 13:34:03
5	Natural Network	Toys	4722.66	75	2023-11-06 00:47:37

-- Query: 3. List all unique product categories

```
SELECT DISTINCT category FROM products;
```

Output:

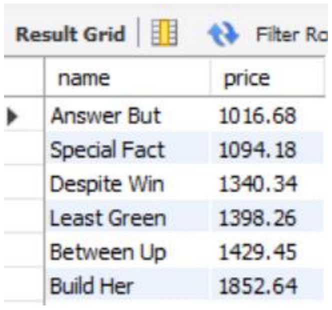


category
Home
Clothing
Electronics
Toys
Books

-- Query: 4. Show all products priced above ₹1,000

```
SELECT name , price FROM products WHERE price > 1000 ORDER BY
price;
```

Output:



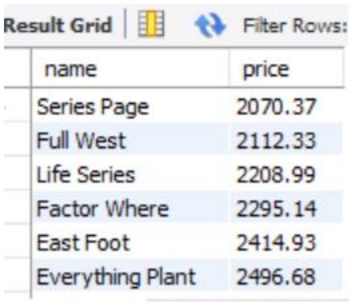
Result Grid | Filter Ro

name	price
Answer But	1016.68
Special Fact	1094.18
Despite Win	1340.34
Least Green	1398.26
Between Up	1429.45
Build Her	1852.64

-- Query: 5. Display products within a mid-range price bracket (₹2,000 to ₹5,000)

```
SELECT name , price FROM products WHERE price BETWEEN 2000 AND
5000 ORDER BY price;
```

Output:



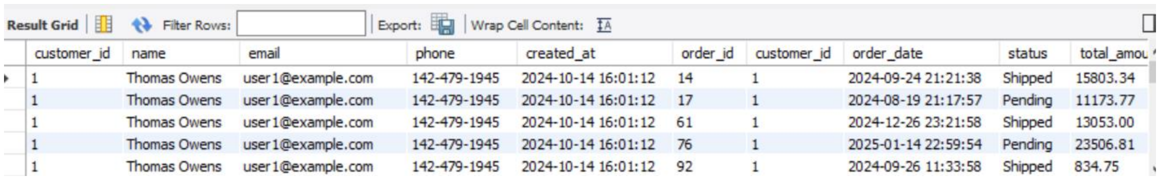
Result Grid | Filter Rows:

name	price
Series Page	2070.37
Full West	2112.33
Life Series	2208.99
Factor Where	2295.14
East Foot	2414.93
Everything Plant	2496.68

-- Query: 6. Fetch data for specific customer IDs (e.g., from loyalty program list)

```
SELECT * FROM customers WHERE customer_id IN(1,2,3);
```

Output:



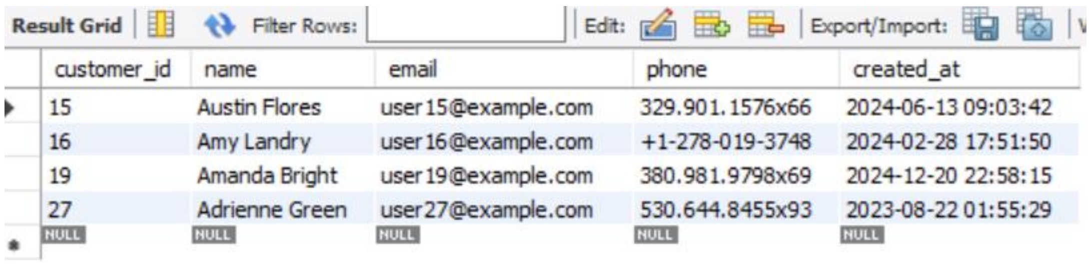
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	name	email	phone	created_at	order_id	customer_id	order_date	status	total_amou
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	14	1	2024-09-24 21:21:38	Shipped	15803.34
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	17	1	2024-08-19 21:17:57	Pending	11173.77
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	61	1	2024-12-26 23:21:58	Shipped	13053.00
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	76	1	2025-01-14 22:59:54	Pending	23506.81
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	92	1	2024-09-26 11:33:58	Shipped	834.75

-- Query: 7. Identify customers whose names start with the letter 'A'

```
SELECT * FROM customers WHERE name LIKE 'A%';
```

Output:

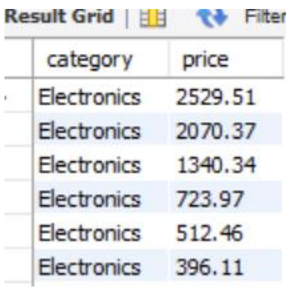


customer_id	name	email	phone	created_at
15	Austin Flores	user15@example.com	329.901.1576x66	2024-06-13 09:03:42
16	Amy Landry	user16@example.com	+1-278-019-3748	2024-02-28 17:51:50
19	Amanda Bright	user19@example.com	380.981.9798x69	2024-12-20 22:58:15
27	Adrienne Green	user27@example.com	530.644.8455x93	2023-08-22 01:55:29
NULL	NULL	NULL	NULL	NULL

-- Query: 8. List electronics products priced under ₹3,000

```
SELECT category, price FROM products WHERE category='Electronics'
AND price < 3000 ORDER BY price DESC;
```

Output:

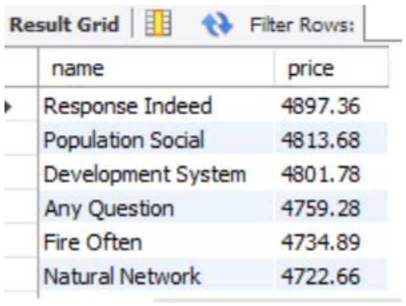


category	price
Electronics	2529.51
Electronics	2070.37
Electronics	1340.34
Electronics	723.97
Electronics	512.46
Electronics	396.11

-- Query: 9. Display product names and prices in descending order of price

```
SELECT name , price FROM products ORDER BY price DESC;
```

Output:

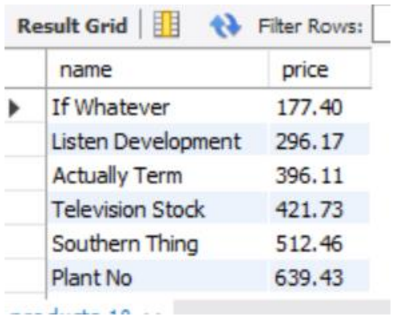


name	price
Response Indeed	4897.36
Population Social	4813.68
Development System	4801.78
Any Question	4759.28
Fire Often	4734.89
Natural Network	4722.66

-- Query: 10. Display product names and prices, sorted by price and then by name

```
SELECT name , price FROM products ORDER BY price , name;
```

Output:



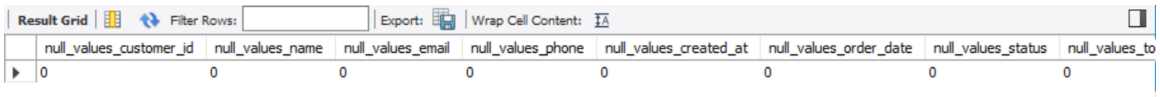
name	price
If Whatever	177.40
Listen Development	296.17
Actually Term	396.11
Television Stock	421.73
Southern Thing	512.46
Plant No	639.43

Level 2: Filtering and Formatting

-- Query: 1. Retrieve orders where customer information is missing (possibly due to data migration or deletion)

```
SELECT

SUM(CASE WHEN customer_id IS NULL THEN 1 ELSE 0 END) AS
null_values_customer_id,

SUM(CASE WHEN name IS NULL THEN 1 ELSE 0 END) AS
null_values_name,

SUM(CASE WHEN email IS NULL THEN 1 ELSE 0 END) AS
null_values_email,

SUM(CASE WHEN phone IS NULL THEN 1 ELSE 0 END) AS
null_values_phone,

SUM(CASE WHEN created_at IS NULL THEN 1 ELSE 0 END) AS
null_values_created_at,

SUM(CASE WHEN order_date IS NULL THEN 1 ELSE 0 END) AS
null_values_order_date,

SUM(CASE WHEN status IS NULL THEN 1 ELSE 0 END) AS
null_values_status,

SUM(CASE WHEN total_amount IS NULL THEN 1 ELSE 0 END) AS
null_values_total_amount

FROM

(SELECT c.customer_id, c.name , c.email, c.phone, c.created_at,
od.order_id, od.order_date, od.status, od.total_amount

FROM orders AS od

LEFT JOIN customers AS c ON od.customer_id = c.customer_id) AS X;
```

Output:

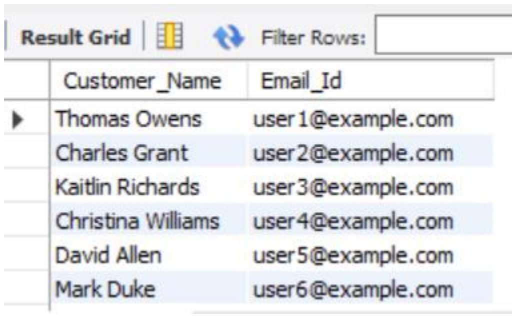


null_values_customer_id	null_values_name	null_values_email	null_values_phone	null_values_created_at	null_values_order_date	null_values_status	null_values_to
0	0	0	0	0	0	0	0

-- Query: 2. Display customer names and emails using column aliases for frontend readability

```
SELECT name AS customer_name , email AS customer_email FROM
customers;
```

Output:

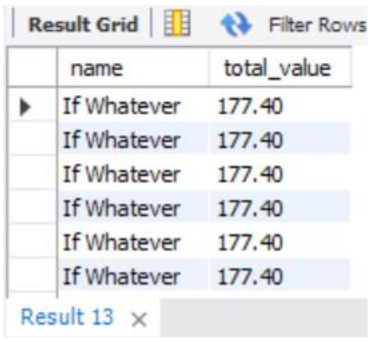


Customer_Name	Email_Id
Thomas Owens	user1@example.com
Charles Grant	user2@example.com
Kaitlin Richards	user3@example.com
Christina Williams	user4@example.com
David Allen	user5@example.com
Mark Duke	user6@example.com

-- Query: 3. Calculate total value per item ordered by multiplying quantity and item price

```
SELECT p.name , (o.quantity * o.item_price) AS total_value

FROM order_items AS o

JOIN products AS p ON p.product_id = o.product_id

ORDER BY total_value;
```

Output:

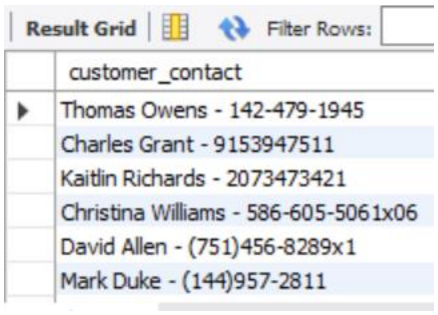


name	total_value
If Whatever	177.40
If Whatever	177.40
If Whatever	177.40
If Whatever	177.40
If Whatever	177.40
If Whatever	177.40

-- Query: 4. Combine customer name and phone number in a single column

```
SELECT CONCAT(name,' - ',phone) AS customer_contact FROM
customers;
```

Output:

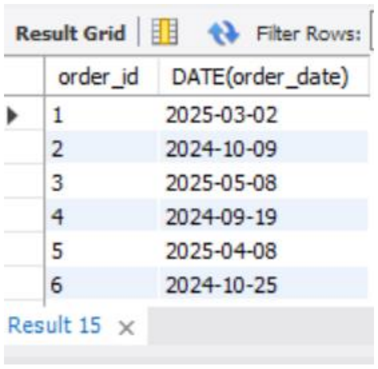


```
customer_contact
Thomas Owens - 142-479-1945
Charles Grant - 9153947511
Kaitlin Richards - 2073473421
Christina Williams - 586-605-5061x06
David Allen - (751)456-8289x1
Mark Duke - (144)957-2811
```

-- Query: 5. Extract only the date part from order timestamps for date-wise reporting

```
SELECT order_id , DATE(order_date) AS order_date_only FROM
orders;
```

Output:

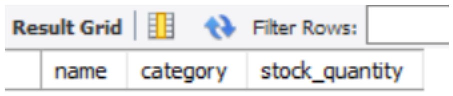


```
order_id  DATE(order_date)
1         2025-03-02
2         2024-10-09
3         2025-05-08
4         2024-09-19
5         2025-04-08
6         2024-10-25
```

-- Query: 6. List products that do not have any stock left

```
SELECT name , category , stock_quantity

FROM products

WHERE stock_quantity IS NULL;
```

Output:



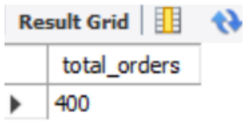
```
name   category   stock_quantity
```

Level 3: Aggregations

-- Query: 1. Count the total number of orders placed

```
SELECT COUNT(*) AS total_orders

FROM orders;
```

Output:

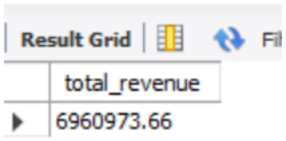


```
total_orders
400
```

-- Query: 2. Calculate the total revenue collected from all orders

```
SELECT SUM(quantity * item_price) AS total_revenue

FROM order_items;


--          OR          --


SELECT SUM(total_amount) AS total_revenue

FROM orders;
```

Output:

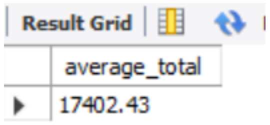


```
total_revenue
6960973.66
```

-- Query: 3. Calculate the average order value

```
SELECT ROUND(AVG(total_amount),2) AS average_total

FROM orders;
```

Output:

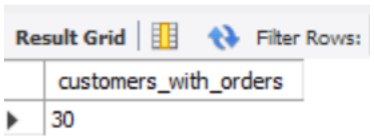


```
average_total
17402.43
```

-- Query: 4. Count the number of customers who have placed at least one order

```
SELECT COUNT(DISTINCT(customer_id)) AS customers_with_orders

FROM customers;
```

Output:

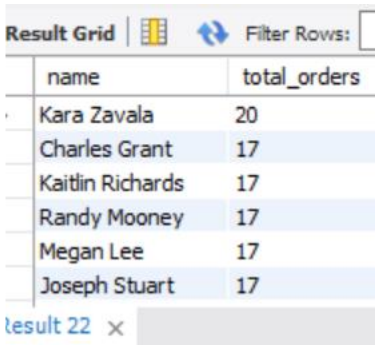


```
customers_with_orders
30
```

-- Query: 5. Find the number of orders placed by each customer

```
SELECT c.name , COUNT(od.order_id) AS total_orders FROM customers
AS c JOIN orders AS od ON od.customer_id = c.customer_id GROUP BY
c.name ORDER BY total_orders DESC;
```

Output:

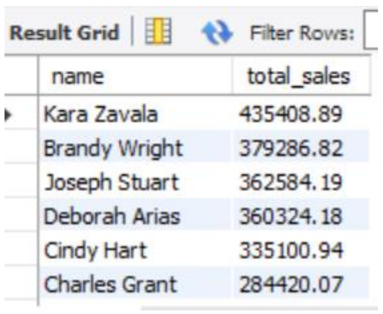


name	total_orders
Kara Zavala	20
Charles Grant	17
Kaitlin Richards	17
Randy Mooney	17
Megan Lee	17
Joseph Stuart	17

-- Query: 6. Find total sales amount made by each customer

```
SELECT c.name , SUM(od.total_amount) AS total_sales FROM
customers AS c JOIN orders AS od ON od.customer_id =
c.customer_id GROUP BY c.name ORDER BY total_sales DESC;
```

Output:

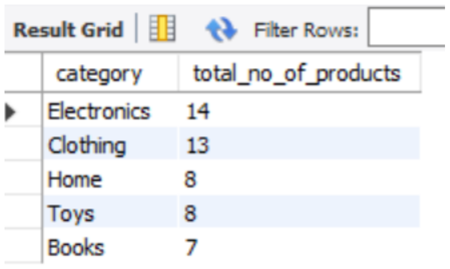


name	total_sales
Kara Zavala	435408.89
Brandy Wright	379286.82
Joseph Stuart	362584.19
Deborah Arias	360324.18
Cindy Hart	335100.94
Charles Grant	284420.07

-- Query: 7. List the number of products sold per category

```
SELECT category , COUNT(product_id) AS total_no_of_products FROM
products GROUP BY category ORDER BY total_no_of_products DESC;
```

Output:

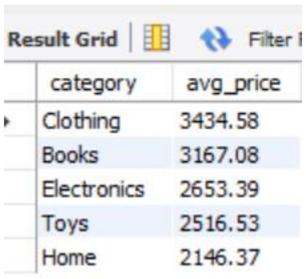


category	total_no_of_products
Electronics	14
Clothing	13
Home	8
Toys	8
Books	7

-- Query: 8. Find the average item price per category

```
SELECT category, ROUND(AVG(price),2) AS avg_price FROM products
GROUP BY category ORDER BY avg_price DESC;
```

Output:

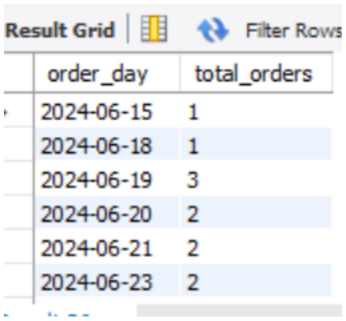


category	avg_price
Clothing	3434.58
Books	3167.08
Electronics	2653.39
Toys	2516.53
Home	2146.37

-- Query: 9. Show number of orders placed per day

```
SELECT DATE(order_date) AS order_day , COUNT(order_id) AS
total_orders FROM orders GROUP BY DATE(order_date) ORDER BY
order_day;
```

Output:

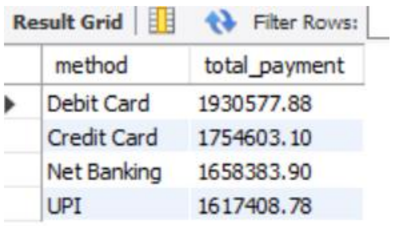


order_day	total_orders
2024-06-15	1
2024-06-18	1
2024-06-19	3
2024-06-20	2
2024-06-21	2
2024-06-23	2

-- Query: 10. List total payments received per payment method

```
SELECT method , SUM(amount_paid) AS total_payment FROM payments
GROUP BY method ORDER BY total_payment DESC;
```

Output:



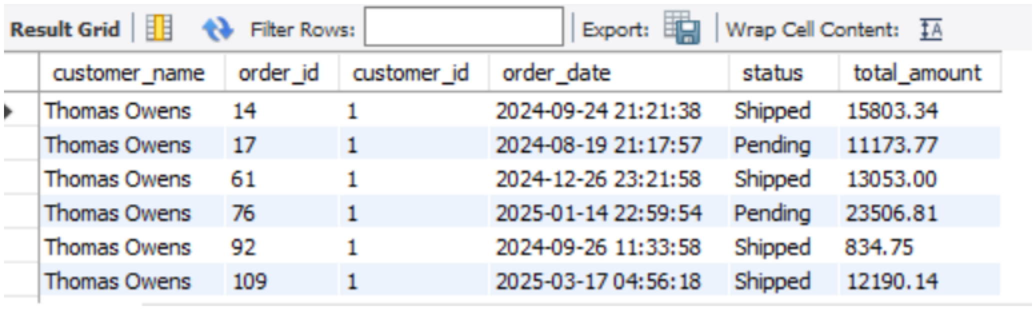
method	total_payment
Debit Card	1930577.88
Credit Card	1754603.10
Net Banking	1658383.90
UPI	1617408.78

Level 4: Multi-Table Queries

-- Query: 1. Retrieve order details along with the customer name (INNER JOIN)

```
SELECT c.name AS customer_name , od.* FROM customers AS c INNER
JOIN orders AS od ON od.customer_id = c.customer_id;
```

Output:

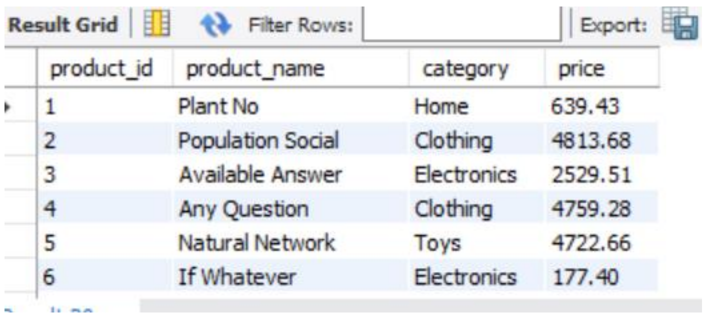


customer_name	order_id	customer_id	order_date	status	total_amount
Thomas Owens	14	1	2024-09-24 21:21:38	Shipped	15803.34
Thomas Owens	17	1	2024-08-19 21:17:57	Pending	11173.77
Thomas Owens	61	1	2024-12-26 23:21:58	Shipped	13053.00
Thomas Owens	76	1	2025-01-14 22:59:54	Pending	23506.81
Thomas Owens	92	1	2024-09-26 11:33:58	Shipped	834.75
Thomas Owens	109	1	2025-03-17 04:56:18	Shipped	12190.14

-- Query: 2. Get list of products that have been sold (INNER JOIN with order_items)

```
SELECT DISTINCT(p.product_id),p.name AS product_name, p.category
, p.price FROM products AS p JOIN order_items AS o ON
o.product_id = p.product_id;
```

Output:

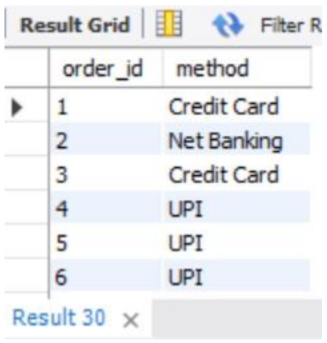


product_id	product_name	category	price
1	Plant No	Home	639.43
2	Population Social	Clothing	4813.68
3	Available Answer	Electronics	2529.51
4	Any Question	Clothing	4759.28
5	Natural Network	Toys	4722.66
6	If Whatever	Electronics	177.40

-- Query: 3. List all orders with their payment method (INNER JOIN)

```
SELECT od.order_id , p.method FROM orders AS od JOIN payments AS
p ON p.order_id = od.order_id ORDER BY od.order_id ;
```

Output:



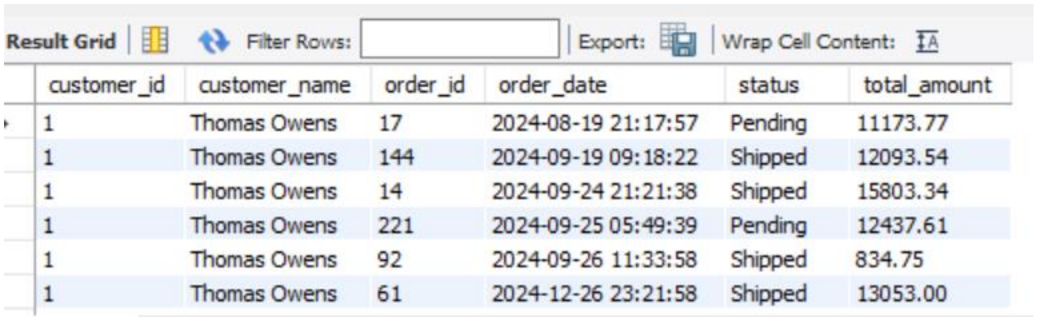
order_id	method
1	Credit Card
2	Net Banking
3	Credit Card
4	UPI
5	UPI
6	UPI

Result 30

-- Query: 4. Get list of customers and their orders (LEFT JOIN)

```
SELECT c.customer_id , c.name AS customer_name , od.order_id ,
od.order_date , od.status , od.total_amount FROM customers as c
LEFT JOIN orders AS od ON od.customer_id= c.customer_id ORDER BY
c.customer_id , od.order_date;
```

Output:



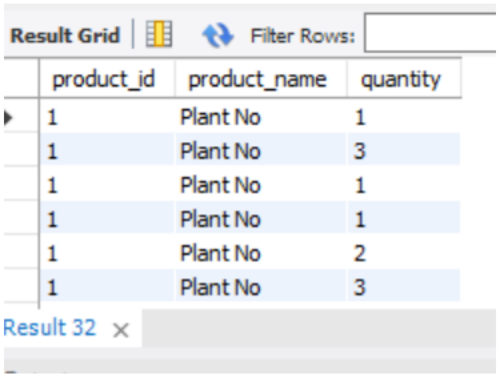
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	customer_name	order_id	order_date	status	total_amount
1	Thomas Owens	17	2024-08-19 21:17:57	Pending	11173.77
1	Thomas Owens	144	2024-09-19 09:18:22	Shipped	12093.54
1	Thomas Owens	14	2024-09-24 21:21:38	Shipped	15803.34
1	Thomas Owens	221	2024-09-25 05:49:39	Pending	12437.61
1	Thomas Owens	92	2024-09-26 11:33:58	Shipped	834.75
1	Thomas Owens	61	2024-12-26 23:21:58	Shipped	13053.00

-- Query: 5. List all products along with order item quantity (LEFT JOIN)

```
SELECT p.product_id , p.name AS product_name , o.quantity FROM
products AS p LEFT JOIN order_items AS o ON o.product_id =
p.product_id;
```

Output:



Result Grid | Filter Rows:

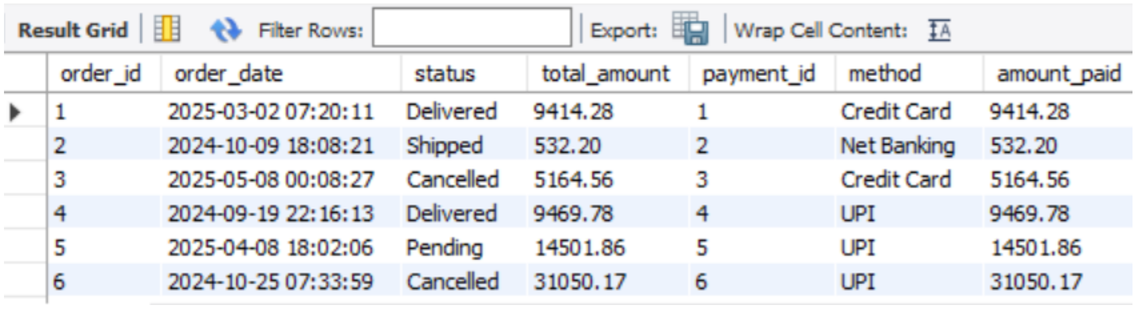
product_id	product_name	quantity
1	Plant No	1
1	Plant No	3
1	Plant No	1
1	Plant No	1
1	Plant No	2
1	Plant No	3

Result 32 ×

-- Query: 6. List all payments including those with no matching orders (RIGHT JOIN)

```
SELECT od.order_id , od.order_date , od.status , od.total_amount
, p.payment_id , p.method , p.amount_paid

FROM orders AS od

RIGHT JOIN payments AS p  ON od.order_id = p.order_id

ORDER BY p.payment_id;
```

Output:



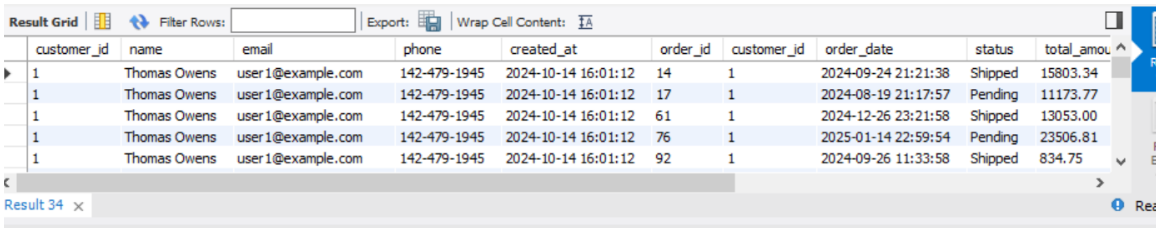
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	order_date	status	total_amount	payment_id	method	amount_paid
1	2025-03-02 07:20:11	Delivered	9414.28	1	Credit Card	9414.28
2	2024-10-09 18:08:21	Shipped	532.20	2	Net Banking	532.20
3	2025-05-08 00:08:27	Cancelled	5164.56	3	Credit Card	5164.56
4	2024-09-19 22:16:13	Delivered	9469.78	4	UPI	9469.78
5	2025-04-08 18:02:06	Pending	14501.86	5	UPI	14501.86
6	2024-10-25 07:33:59	Cancelled	31050.17	6	UPI	31050.17

-- Query: 7. Combine data from three tables: customer, order, and payment

```
SELECT * FROM customers AS c JOIN orders AS od ON od.customer_id
= c.customer_id JOIN payments AS p ON p.order_id = od.order_id;
```

Output:



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

customer_id	name	email	phone	created_at	order_id	customer_id	order_date	status	total_amou
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	14	1	2024-09-24 21:21:38	Shipped	15803.34
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	17	1	2024-08-19 21:17:57	Pending	11173.77
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	61	1	2024-12-26 23:21:58	Shipped	13053.00
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	76	1	2025-01-14 22:59:54	Pending	23506.81
1	Thomas Owens	user1@example.com	142-479-1945	2024-10-14 16:01:12	92	1	2024-09-26 11:33:58	Shipped	834.75

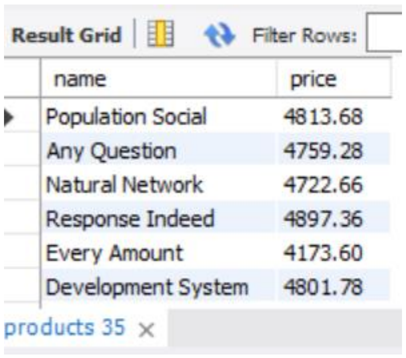
Result 34

Level 5: Subqueries

-- Query: 1. List all products priced above the average product price

```
SELECT name , price FROM products WHERE price > (SELECT
AVG(price) FROM products);
```

Output:



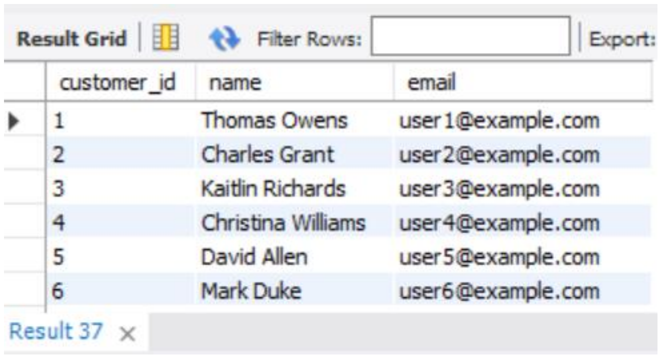
name	price
Population Social	4813.68
Any Question	4759.28
Natural Network	4722.66
Response Indeed	4897.36
Every Amount	4173.60
Development System	4801.78

-- Query: 2. Find customers who have placed at least one order

```
-- USING WINDOW FUNCTION

SELECT  DISTINCT customer_id , name , email FROM (SELECT
c.customer_id , c.name , c.email , od.order_id , ROW_NUMBER()
OVER(PARTITION BY c.customer_id ORDER BY od.order_id) AS rk FROM
customers AS c LEFT JOIN orders AS od ON od.customer_id =
c.customer_id) AS x WHERE rk = 1;
```

```
-- USING SUBQUERY
SELECT customer_id , name , email FROM customers WHERE
customer_id IN(SELECT DISTINCT customer_id FROM orders);
```

Output:

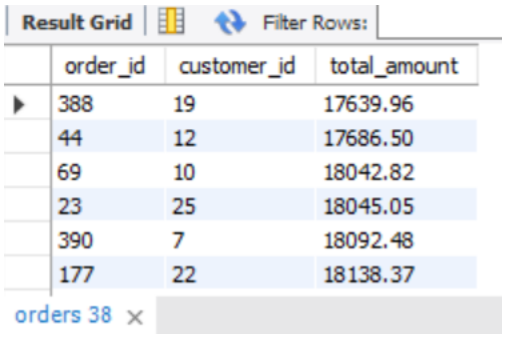


	customer_id	name	email
▶	1	Thomas Owens	user1@example.com
	2	Charles Grant	user2@example.com
	3	Kaitlin Richards	user3@example.com
	4	Christina Williams	user4@example.com
	5	David Allen	user5@example.com
	6	Mark Duke	user6@example.com

-- Query: 3. Show orders whose total amount is above the average for that customer

```
SELECT order_id , customer_id , total_amount FROM orders WHERE
total_amount > (SELECT AVG(total_amount) FROM orders) ORDER BY
total_amount;
```

Output:

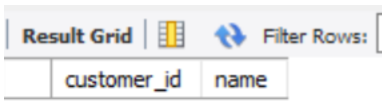


	order_id	customer_id	total_amount
▶	388	19	17639.96
	44	12	17686.50
	69	10	18042.82
	23	25	18045.05
	390	7	18092.48
	177	22	18138.37

-- Query: 4. Display customers who haven't placed any orders

```
SELECT c.customer_id , c.name

FROM customers AS c

JOIN orders AS od ON od.customer_id = c.customer_id

WHERE c.customer_id NOT IN (SELECT DISTINCT customer_id FROM
orders);
```

Output:

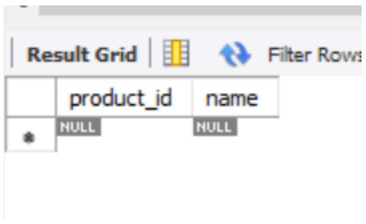


customer_id	name

-- Query: 5. Show products that were never ordered

```
SELECT product_id , name , category FROM products WHERE
product_id NOT IN (SELECT DISTINCT product_id FROM order_items);
```

Output:



	product_id	name
*	NULL	NULL

-- Query: 6. Show highest value order per customer

SELECT customer_id , order_id , name , total_amount , rk

FROM

(SELECT c.customer_id , c.name , od.order_id , od.total_amount , ROW_NUMBER() OVER(PARTITION BY c.customer_id ORDER BY od.total_amount DESC) AS rk

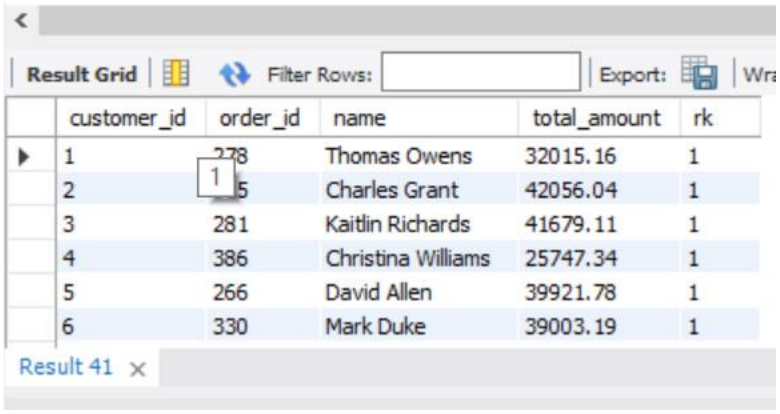
FROM customers AS c

JOIN orders AS od ON od.customer_id = c.customer_id) AS x

WHERE rk = 1

ORDER BY customer_id;

Output:

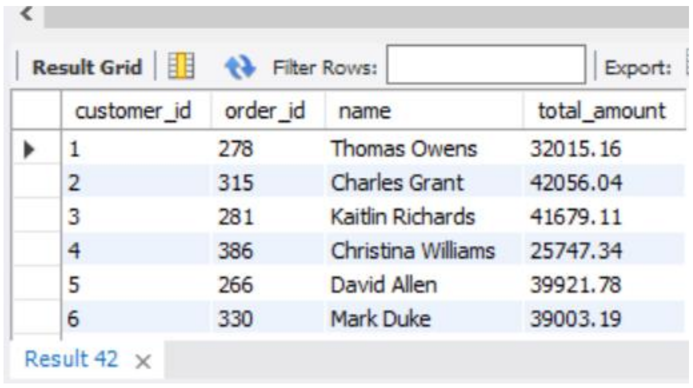


customer_id	order_id	name	total_amount	rk
1	278	Thomas Owens	32015.16	1
2	315	Charles Grant	42056.04	1
3	281	Kaitlin Richards	41679.11	1
4	386	Christina Williams	25747.34	1
5	266	David Allen	39921.78	1
6	330	Mark Duke	39003.19	1

-- Query: 7. Highest Order Per Customer (Including Names)

```
SELECT customer_id , order_id , name , total_amount

FROM

(SELECT c.customer_id , c.name , od.order_id , od.total_amount , ROW_NUMBER()
OVER(PARTITION BY c.customer_id ORDER BY od.total_amount DESC) AS rk

FROM customers AS c

JOIN orders AS od ON od.customer_id = c.customer_id) AS x

WHERE rk = 1

ORDER BY customer_id;
```

Output:



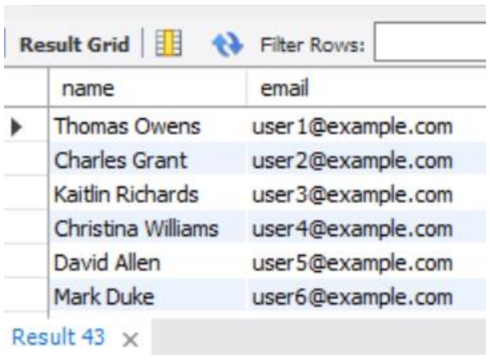
customer_id	order_id	name	total_amount
1	278	Thomas Owens	32015.16
2	315	Charles Grant	42056.04
3	281	Kaitlin Richards	41679.11
4	386	Christina Williams	25747.34
5	266	David Allen	39921.78
6	330	Mark Duke	39003.19

Level 6: Set Operations

-- Query: 1. List all customers who have either placed an order or written a product review

```
SELECT name, email

FROM customers

WHERE customer_id IN (SELECT customer_id FROM orders)

UNION

SELECT name, email

FROM customers

WHERE customer_id IN (SELECT customer_id FROM product_reviews);
```

Output:

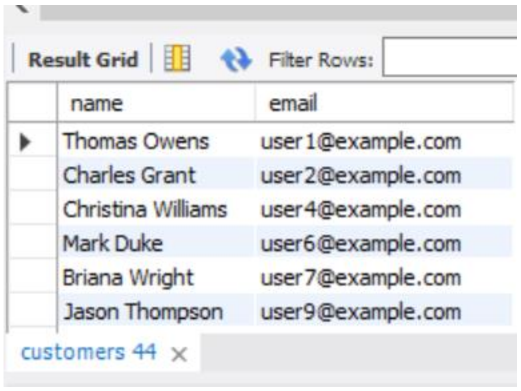


name	email
Thomas Owens	user1@example.com
Charles Grant	user2@example.com
Kaitlin Richards	user3@example.com
Christina Williams	user4@example.com
David Allen	user5@example.com
Mark Duke	user6@example.com

-- Query: 2. List all customers who have placed an order as well as reviewed a product [intersect not supported]

```
SELECT name, email

FROM customers

WHERE customer_id IN (SELECT customer_id FROM orders)

AND customer_id IN (SELECT customer_id FROM product_reviews);
```

Output:



name	email
Thomas Owens	user1@example.com
Charles Grant	user2@example.com
Christina Williams	user4@example.com
Mark Duke	user6@example.com
Briana Wright	user7@example.com
Jason Thompson	user9@example.com