SQL Project
Retail Store

AYUSH KUMAR SIDDHARTH

DDL SCRIPTS

```sql
CREATE DATABASE retail_store;
USE retail_store;
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phone VARCHAR(15),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    category VARCHAR(50) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INT NOT NULL DEFAULT 0,
    added_on DATETIME DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'Pending',
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```sql
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT NOT NULL CHECK (quantity > 0),
    item_price DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
CREATE TABLE payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    payment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    amount_paid DECIMAL(10,2) NOT NULL CHECK (amount_paid > 0),
    method VARCHAR(20) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
CREATE TABLE product_reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    customer_id INT,
    rating INT NOT NULL,
    review_text TEXT,
    review_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Questions

Level 1: Basics

1. Retrieve customer names and emails for email marketing
This helps the marketing team extract basic customer contact details for campaigns.

2. View complete product catalog with all available details
The product manager may want to review all product listings in one go.

3. List all unique product categories
Useful for analyzing the range of departments or for creating filters on the website.

4. Show all products priced above ₹1,000
This helps identify high-value items for premium promotions or pricing strategy reviews.

5. Display products within a mid-range price bracket (₹2,000 to ₹5,000)
A merchandising team might need this to create a mid-tier pricing campaign.

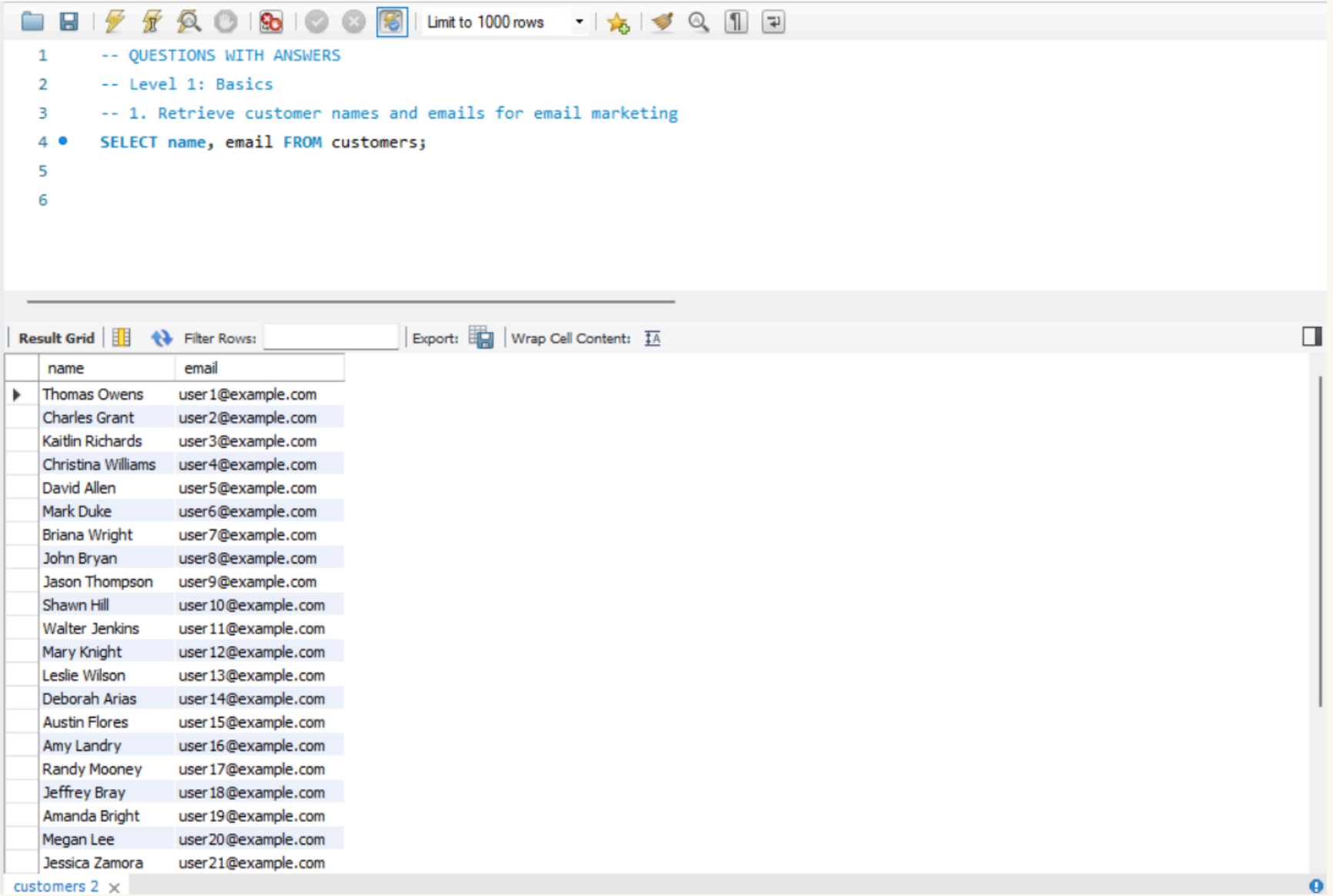
6. Fetch data for specific customer IDs (e.g., from loyalty program list)
This is used when customer IDs are pre-selected from another system.

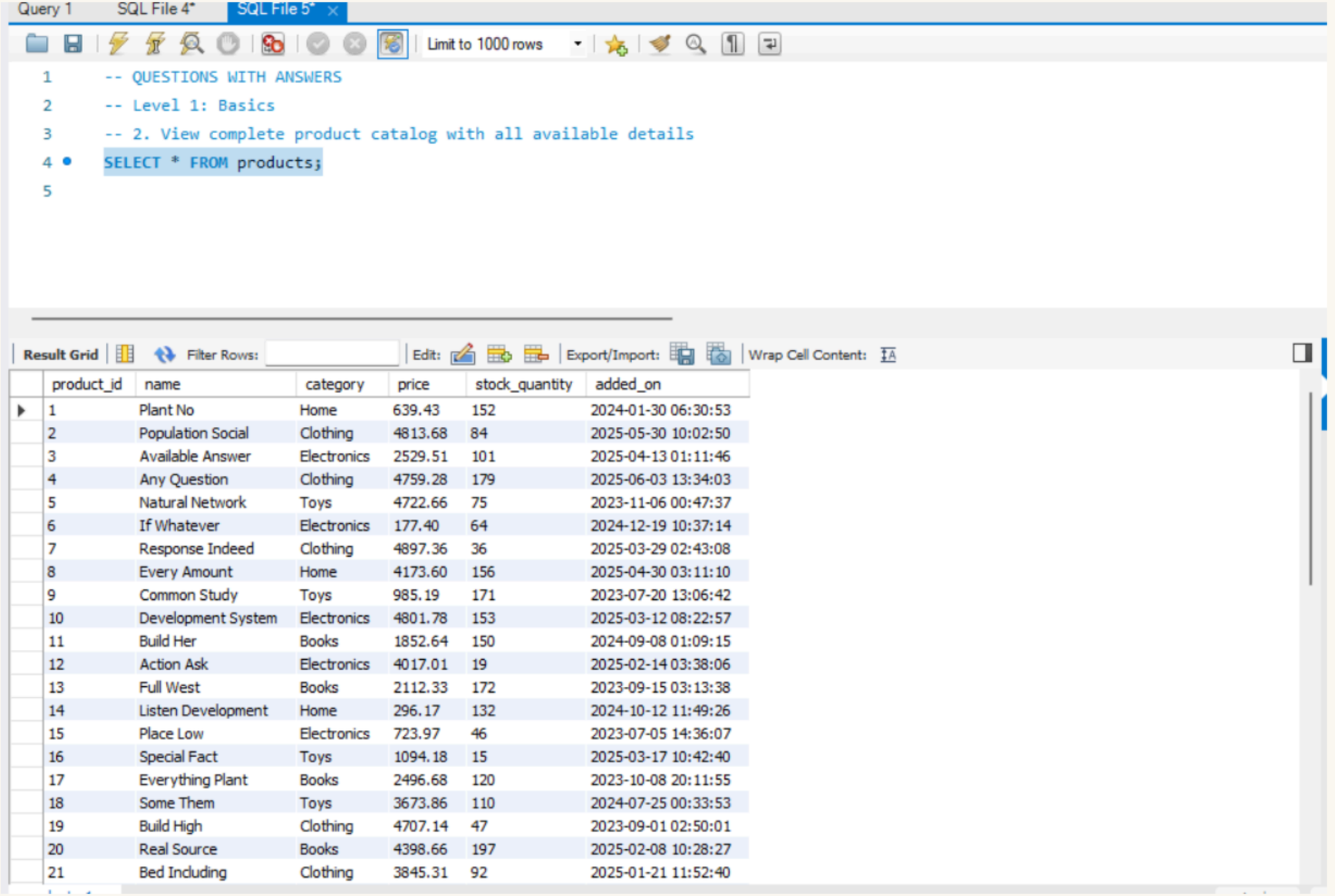
7. Identify customers whose names start with the letter 'A'
Used for alphabetical segmentation in outreach or app display.

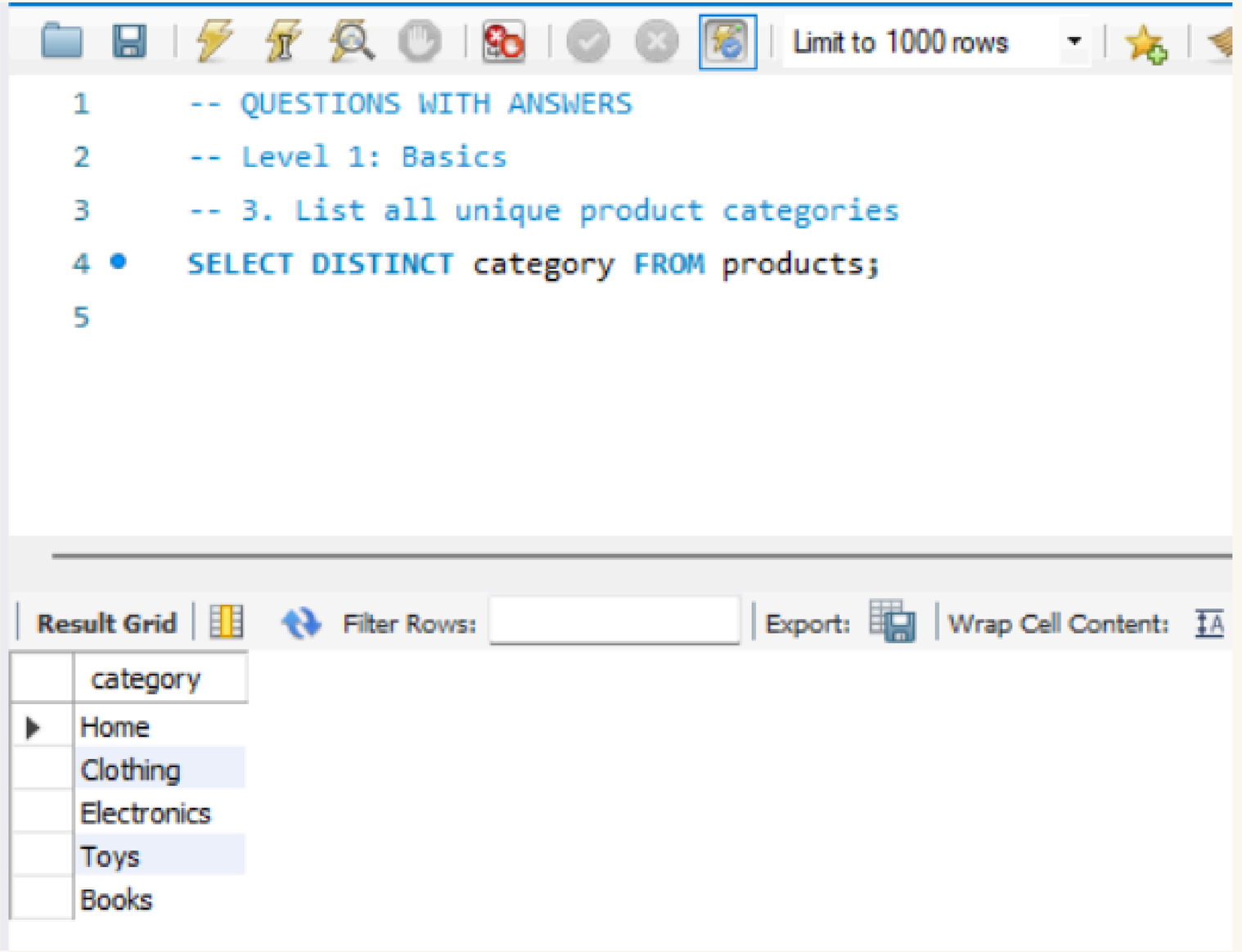
8. List electronics products priced under ₹3,000
Used by merchandising or frontend teams to showcase budget electronics.

9. Display product names and prices in descending order of price
This helps teams easily view and compare top-priced items.

10. Display product names and prices, sorted by price and then by name
The merchandising or catalog team may want to list products from most expensive to cheapest. If multiple products have the same price, they should be sorted alphabetically for clarity on storefronts or printed catalogs.


Limit to 1000 rows
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 1. Retrieve customer names and emails for email marketing
SELECT name, email FROM customers;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
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
Briana Wright user7@example.com
John Bryan user8@example.com
Jason Thompson user9@example.com
Shawn Hill user10@example.com
Walter Jenkins user11@example.com
Mary Knight user12@example.com
Leslie Wilson user13@example.com
Deborah Arias user14@example.com
Austin Flores user15@example.com
Amy Landry user16@example.com
Randy Mooney user17@example.com
Jeffrey Bray user18@example.com
Amanda Bright user19@example.com
Megan Lee user20@example.com
Jessica Zamora user21@example.com
customers 2


Query 1
SQL File 4*
SQL File 5*
Limit to 1000 rows
1 -- QUESTIONS WITH ANSWERS
2 -- Level 1: Basics
3 -- 2. View complete product catalog with all available details
4 SELECT * FROM products;
5
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
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
6 | If Whatever | Electronics | 177.40 | 64 | 2024-12-19 10:37:14
7 | Response Indeed | Clothing | 4897.36 | 36 | 2025-03-29 02:43:08
8 | Every Amount | Home | 4173.60 | 156 | 2025-04-30 03:11:10
9 | Common Study | Toys | 985.19 | 171 | 2023-07-20 13:06:42
10 | Development System | Electronics | 4801.78 | 153 | 2025-03-12 08:22:57
11 | Build Her | Books | 1852.64 | 150 | 2024-09-08 01:09:15
12 | Action Ask | Electronics | 4017.01 | 19 | 2025-02-14 03:38:06
13 | Full West | Books | 2112.33 | 172 | 2023-09-15 03:13:38
14 | Listen Development | Home | 296.17 | 132 | 2024-10-12 11:49:26
15 | Place Low | Electronics | 723.97 | 46 | 2023-07-05 14:36:07
16 | Special Fact | Toys | 1094.18 | 15 | 2025-03-17 10:42:40
17 | Everything Plant | Books | 2496.68 | 120 | 2023-10-08 20:11:55
18 | Some Them | Toys | 3673.86 | 110 | 2024-07-25 00:33:53
19 | Build High | Clothing | 4707.14 | 47 | 2023-09-01 02:50:01
20 | Real Source | Books | 4398.66 | 197 | 2025-02-08 10:28:27
21 | Bed Including | Clothing | 3845.31 | 92 | 2025-01-21 11:52:40


Limit to 1000 rows
1 -- QUESTIONS WITH ANSWERS
2 -- Level 1: Basics
3 -- 3. List all unique product categories
4 SELECT DISTINCT category FROM products;
5
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
category
Home
Clothing
Electronics
Toys
Books

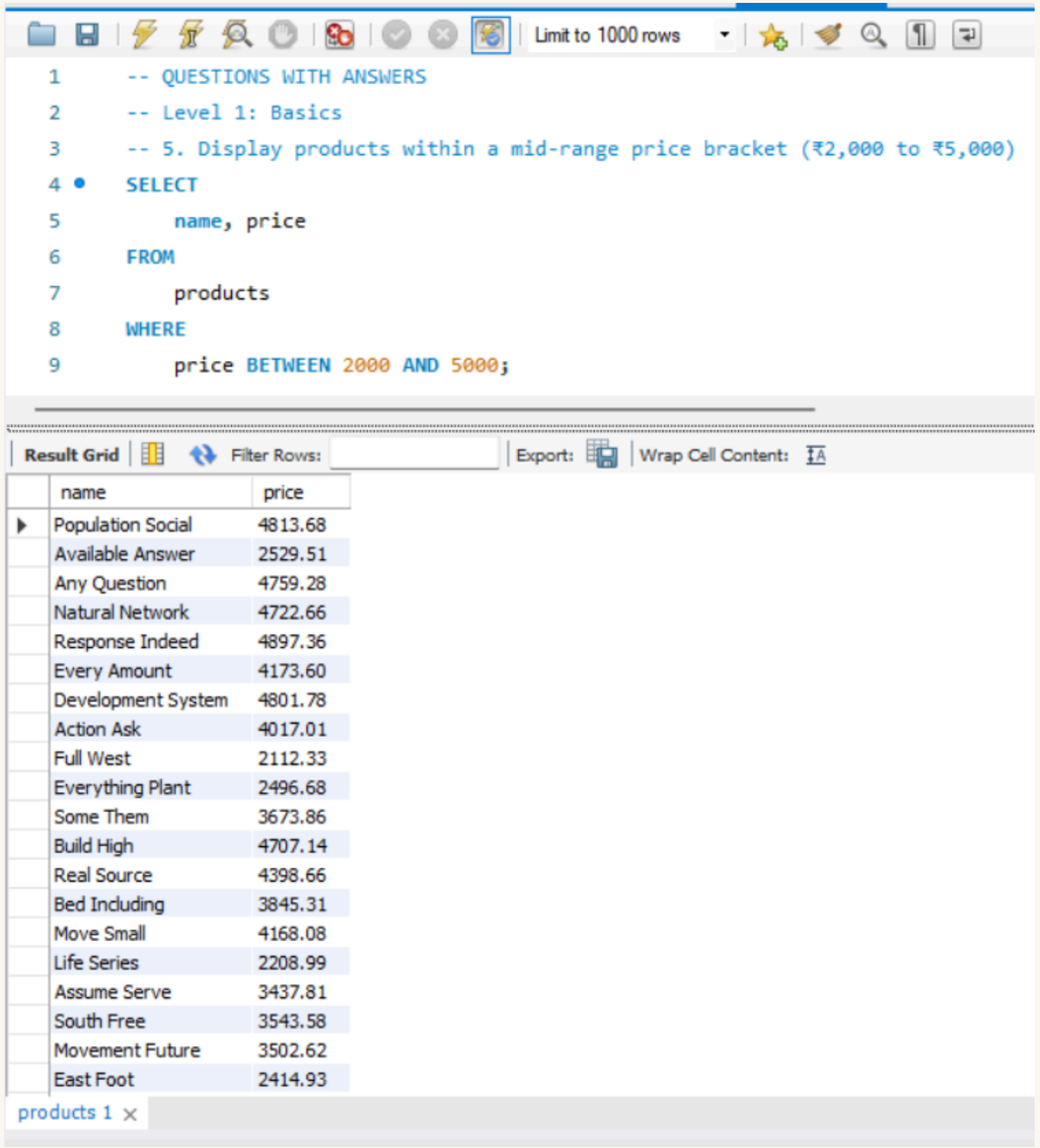
```sql
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 4. Show all products priced above ₹1,000
SELECT
    name, price
FROM
    products
WHERE
    price > 1000
ORDER BY price DESC;

```

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

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
Build High	4707.14
Serious Recognize	4523.10
Study Total	4413.68
Real Source	4398.66
Involve Officer	4244.09
Group But	4180.23
Every Amount	4173.60
Move Small	4168.08
Drop Remember	4160.45
Data Add	4063.38
Action Ask	4017.01
Weight Enter	3875.18

products 1 ×


Limit to 1000 rows
1 -- QUESTIONS WITH ANSWERS
2 -- Level 1: Basics
3 -- 5. Display products within a mid-range price bracket (₹2,000 to ₹5,000)
4 SELECT
5 name, price
6 FROM
7 products
8 WHERE
9 price BETWEEN 2000 AND 5000;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
name price
Population Social 4813.68
Available Answer 2529.51
Any Question 4759.28
Natural Network 4722.66
Response Indeed 4897.36
Every Amount 4173.60
Development System 4801.78
Action Ask 4017.01
Full West 2112.33
Everything Plant 2496.68
Some Them 3673.86
Build High 4707.14
Real Source 4398.66
Bed Including 3845.31
Move Small 4168.08
Life Series 2208.99
Assume Serve 3437.81
South Free 3543.58
Movement Future 3502.62
East Foot 2414.93
products 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 6. Fetch data for specific customer IDs (e.g., from loyalty program list)
SELECT name, category
FROM products
WHERE category <> 'Electronics';
-- Here product not in Electronic category
```

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

name	category
Move Small	Books
Life Series	Clothing
Assume Serve	Home
South Free	Clothing
Answer But	Home
East Foot	Toys
Future Sort	Home
Factor Where	Clothing
Involve Officer	Clothing
Television Stock	Clothing
Between Up	Toys
Study Total	Toys
Data Add	Books
Toward Minute	Clothing
Weight Enter	Clothing
Least Green	Toys
Age Treatment	Home
Group But	Clothing
Rest Something	Books
Stock Article	Home

products 2

```sql
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 7. Identify customers whose names start with the letter ‘A’
SELECT * FROM customers WHERE name LIKE 'A%';

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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

```sql
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 8. List electronics products priced under ₹3,000
SELECT category, product_id, name, price FROM products WHERE category = 'Electronics' AND price < 3000;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

category	product_id	name	price
Electronics	3	Available Answer	2529.51
Electronics	6	If Whatever	177.40
Electronics	15	Place Low	723.97
Electronics	31	Series Page	2070.37
Electronics	34	Despite Win	1340.34
Electronics	44	Actually Term	396.11
Electronics	47	Southern Thing	512.46
NULL	NULL	NULL	NULL

```sql
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 9. Display product names and prices in descending order of price
SELECT name, price FROM products ORDER BY price DESC;
```

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

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
Build High	4707.14
Serious Recognize	4523.10
Study Total	4413.68
Real Source	4398.66
Involve Officer	4244.09
Group But	4180.23
Every Amount	4173.60
Move Small	4168.08

products 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 1: Basics
-- 10. Display product names and prices, sorted by price and then by name
SELECT name, price FROM products ORDER BY price ASC, name ASC;

```

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

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
Place Low	723.97
Stock Article	834.75
Toward Minute	857.85
Common Study	985.19
Answer But	1016.68
Special Fact	1094.18
Despite Win	1340.34
Least Green	1398.26
Between Up	1429.45
Build Her	1852.64

products 1

Level 2: Filtering and Formatting

1. Retrieve orders where customer information is missing (possibly due to data migration or deletion)
Used to identify orphaned orders or test data where customer_id is not linked.

2. Display customer names and emails using column aliases for frontend readability
Useful for feeding into frontend displays or report headings that require user-friendly labels.

3. Calculate total value per item ordered by multiplying quantity and item price
This can help generate per-line item bill details or invoice breakdowns.

4. Combine customer name and phone number in a single column
Used to show brief customer summaries or contact lists.

5. Extract only the date part from order timestamps for date-wise reporting
Helps group or filter orders by date without considering time.

6. List products that do not have any stock left
This helps the inventory team identify out-of-stock items.

```sql
-- QUESTIONS WITH ANSWERS
-- Level 2: Filtering and Formatting
-- 1. Retrieve orders where customer information is missing (possibly due to data migration or deletion
SELECT * FROM orders WHERE customer_id IS NULL;
-- No Null values is find.
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

order_id	customer_id	order_date	status	total_amount
NULL	NULL	NULL	NULL	NULL

```sql
-- QUESTIONS WITH ANSWERS
-- Level 2: Filtering and Formatting
-- 2. Display customer names and emails using column aliases for frontend readability
SELECT name AS customer_name, email AS customer_email FROM customers;

```

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

customer_name	customer_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
Briana Wright	user7@example.com
John Bryan	user8@example.com
Jason Thompson	user9@example.com
Shawn Hill	user10@example.com
Walter Jenkins	user11@example.com
Mary Knight	user12@example.com
Leslie Wilson	user13@example.com
Deborah Arias	user14@example.com
Austin Flores	user15@example.com
Amy Landry	user16@example.com

customers 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 2: Filtering and Formatting
-- 3. Calculate total value per item ordered by multiplying quantity and item price
SELECT
    order_item_id,
    order_id,
    product_id,
    quantity,
    item_price,
    (quantity * item_price) AS total_value
FROM
```

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

order_item_id	order_id	product_id	quantity	item_price	total_value
1	1	19	2	4707.14	9414.28
2	2	6	3	177.40	532.20
3	3	9	3	985.19	2955.57
4	3	23	1	2208.99	2208.99
5	4	35	2	4734.89	9469.78
6	5	19	1	4707.14	4707.14
7	5	7	2	4897.36	9794.72
8	6	7	3	4897.36	14692.08
9	6	37	1	1429.45	1429.45
10	6	15	1	723.97	723.97

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 2: Filtering and Formatting
-- 4. Combine customer name and phone number in a single column
SELECT CONCAT(name, ' --- ', phone) AS contact_details
FROM customers;
```

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

contact_details
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
Briana Wright --- 223-833-9635
John Bryan --- 045.568.0798x27
Jason Thompson --- 1862659420
Shawn Hill --- (268)113-3152x7
Walter Jenkins --- 536-329-0817x71
Mary Knight --- 361-636-3802
Leslie Wilson --- +1-256-261-1984
Deborah Arias --- 811-821-2144x97
Austin Flores --- 329.901.1576x66
Amy Landry --- +1-278-019-3748
Randy Mooney --- (158)927-7313x3
Jeffrey Bray --- 164.962.0222x92

Result 2

```sql
-- QUESTIONS WITH ANSWERS
-- Level 2: Filtering and Formatting
-- 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;
```

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

order_id	order_date_only
1	2025-03-02
2	2024-10-09
3	2025-05-08
4	2024-09-19
5	2025-04-08
6	2024-10-25
7	2024-07-29
8	2024-07-30
9	2025-06-10
10	2025-02-16
11	2025-03-11
12	2025-01-15
13	2025-05-12
14	2024-09-24
15	2024-07-18
16	2024-09-09

Result 2

```sql
-- QUESTIONS WITH ANSWERS
-- Level 2: Filtering and Formatting
-- 6. List products that do not have any stock left
SELECT product_id, name, stock_quantity
FROM products
WHERE stock_quantity = 0;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	name	stock_quantity
NULL	NULL	NULL

Level 3: Aggregations

1. Count the total number of orders placed
Used by business managers to track order volume over time.

2. Calculate the total revenue collected from all orders
This gives the overall sales value.

3. Calculate the average order value
Used for understanding customer spending patterns.

4. Count the number of customers who have placed at least one order
This identifies active customers.

5. Find the number of orders placed by each customer
Helpful for identifying top or repeat customers.

6. Find total sales amount made by each customer

7. List the number of products sold per category
This helps category managers assess performance by department.

8. Find the average item price per category
Useful to compare pricing across departments.

9. Show number of orders placed per day
Used to track daily business activity and demand trends.

10. List total payments received per payment method
Helps the finance team understand preferred transaction modes.

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 1. Count the total number of orders placed

SELECT COUNT(*) AS total_orders
FROM orders;
```

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

total_orders
400

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 2. Calculate the total revenue collected from all orders
SELECT SUM(total_amount) AS total_revenue
FROM orders;
```

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

total_revenue
6960973.66

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 3. Calculate the average order value
SELECT AVG(total_amount) AS avg_order_value
FROM orders;
```

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

avg_order_value
17402.434150

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 4. Count the number of customers who have placed at least one order

SELECT customer_id, COUNT(order_id) AS orders_count
FROM orders
GROUP BY customer_id;

-- There are 30 customers who have placed atleast one order
```

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

customer_id	orders_count
10	13
11	9
12	11
13	12
14	15
15	15
16	16
17	17
18	9
19	15
20	17
21	13
22	15

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 5. Find the number of orders placed by each customer
SELECT customer_id, COUNT(order_id) AS orders_count
FROM orders
GROUP BY customer_id
ORDER BY orders_count DESC;
```

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

customer_id	orders_count
29	20
2	17
3	17
17	17
20	17
28	17
6	16
16	16
30	16
14	15
15	15
19	15
22	15

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 6. Find total sales amount made by each customer

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

```

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

customer_id	customer_name	total_sales
29	Kara Zavala	435408.89
24	Brandy Wright	379286.82
28	Joseph Stuart	362584.19
14	Deborah Arias	360324.18
25	Cindy Hart	335100.94
2	Charles Grant	284420.07
30	Victoria Acevedo	281841.38
16	Amy Landry	278469.73
5	David Allen	262504.19
17	Randy Mooney	262189.71
15	Austin Flores	260499.91
3	Kaitlin Richards	253783.31

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 7. List the number of products sold per category
SELECT p.category, SUM(oi.quantity) AS total_sold
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.category
ORDER BY total_sold DESC;
```

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

category	total_sold
Electronics	687
Clothing	559
Home	443
Toys	405
Books	350

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 8. Find the average item price per category
SELECT category,
       AVG(price) AS avg_item_price
FROM products
GROUP BY category
ORDER BY avg_item_price DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cel

category	avg_item_price
Clothing	3434.581538
Books	3167.084286
Electronics	2653.388571
Toys	2516.526250
Home	2146.367500

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 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;
```

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

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
2024-06-24	1
2024-06-25	1
2024-06-30	2
2024-07-03	1
2024-07-04	1
2024-07-05	3

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 3: Aggregations
-- 10. List total payments received per payment method
SELECT method AS payment_method,
       SUM(amount_paid) AS total_payments
FROM payments
GROUP BY method
ORDER BY total_payments DESC;
```

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

payment_method	total_payments
Debit Card	1930577.88
Credit Card	1754603.10
Net Banking	1658383.90
UPI	1617408.78

Level 4: Multi-Table Queries (JOINS)

1. Retrieve order details along with the customer name (INNER JOIN)
Used for displaying which customer placed each order.

2. Get list of products that have been sold (INNER JOIN with order_items)
Used to find which products were actually included in orders.

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

Used by finance or audit teams to see how each order was paid for.

4. Get list of customers and their orders (LEFT JOIN)
Used to find all customers and see who has or hasn't placed orders.

5. List all products along with order item quantity (LEFT JOIN)
Useful for inventory teams to track what sold and what hasn't.

6. List all payments including those with no matching orders (RIGHT JOIN)
Rare but used when ensuring all payments are mapped correctly.

7. Combine data from three tables: customer, order, and payment
Used for detailed transaction reports.

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 1. Retrieve order details along with the customer name (INNER JOIN)

SELECT o.order_id, o.order_date, o.total_amount,
       c.name AS customer_name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id;
```

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

order_id	order_date	total_amount	customer_name
14	2024-09-24 21:21:38	15803.34	Thomas Owens
17	2024-08-19 21:17:57	11173.77	Thomas Owens
61	2024-12-26 23:21:58	13053.00	Thomas Owens
76	2025-01-14 22:59:54	23506.81	Thomas Owens
92	2024-09-26 11:33:58	834.75	Thomas Owens
109	2025-03-17 04:56:18	12190.14	Thomas Owens
127	2025-06-10 18:01:32	20160.64	Thomas Owens
135	2025-03-11 22:58:23	8891.79	Thomas Owens
144	2024-09-19 09:18:22	12093.54	Thomas Owens
221	2024-09-25 05:49:39	12437.61	Thomas Owens
278	2025-02-01 20:05:38	32015.16	Thomas Owens
379	2025-03-22 15:57:58	21586.89	Thomas Owens
56	2024-06-23 15:59:57	6828.61	Charles Grant
169	2025-02-11 04:04:38	9426.30	Charles Grant
209	2024-10-16 10:07:48	7690.62	Charles Grant
223	2024-11-25 01:32:18	41020.75	Charles Grant

Result 4

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 2. Get list of products that have been sold (INNER JOIN with order_items)

SELECT DISTINCT p.product_id, p.name
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id;
```

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

product_id	name
1	Plant No
2	Population Social
3	Available Answer
4	Any Question
5	Natural Network
6	If Whatever
7	Response Indeed
8	Every Amount
9	Common Study
10	Development System
11	Build Her
12	Action Ask
13	Full West
14	Listen Development
15	Place Low
16	Special Fact
17	Everything Plant

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 3. List all orders with their payment method (INNER JOIN)

SELECT o.order_id, o.total_amount,
       p.method AS payment_method, p.payment_date
FROM orders o
LEFT JOIN payments p ON o.order_id = p.order_id;
```

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

order_id	total_amount	payment_method	payment_date
1	9414.28	Credit Card	2025-03-02 08:16:11
2	532.20	Net Banking	2024-10-09 19:00:21
3	5164.56	Credit Card	2025-05-08 00:55:27
4	9469.78	UPI	2024-09-19 22:28:13
5	14501.86	UPI	2025-04-08 18:26:06
6	31050.17	UPI	2024-10-25 08:16:59
7	3043.67	UPI	2024-07-29 12:45:47
8	32714.06	Net Banking	2024-07-30 23:42:49
9	24219.20	Net Banking	2025-06-10 17:36:25
10	24342.52	Debit Card	2025-02-16 13:43:59
11	16196.13	Credit Card	2025-03-11 15:05:46
12	9890.72	Credit Card	2025-01-15 14:00:53
13	9627.36	UPI	2025-05-12 14:01:29
14	15803.34	UPI	2024-09-24 22:11:38
15	12421.88	UPI	2024-07-18 03:45:24

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 4. Get list of customers and their orders (LEFT JOIN)

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

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

customer_id	customer_name	order_id	order_date	total_amount	status
1	Thomas Owens	17	2024-08-19 21:17:57	11173.77	Pending
1	Thomas Owens	144	2024-09-19 09:18:22	12093.54	Shipped
1	Thomas Owens	14	2024-09-24 21:21:38	15803.34	Shipped
1	Thomas Owens	221	2024-09-25 05:49:39	12437.61	Pending
1	Thomas Owens	92	2024-09-26 11:33:58	834.75	Shipped
1	Thomas Owens	61	2024-12-26 23:21:58	13053.00	Shipped
1	Thomas Owens	76	2025-01-14 22:59:54	23506.81	Pending
1	Thomas Owens	278	2025-02-01 20:05:38	32015.16	Delivered

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 5. List all products along with order item quantity (LEFT JOIN)

SELECT p.product_id,
       p.name AS product_name,
       p.category,
       oi.order_id,
       oi.quantity
FROM products p
LEFT JOIN order_items oi
       ON p.product_id = oi.product_id
ORDER BY p.product_id;
```

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

product_id	product_name	category	order_id	quantity
1	Plant No	Home	10	1
1	Plant No	Home	11	3
1	Plant No	Home	28	1
1	Plant No	Home	37	1
1	Plant No	Home	49	2
1	Plant No	Home	71	3
1	Plant No	Home	75	2
1	Plant No	Home	81	2
1	Plant No	Home	127	2
1	Plant No	Home	134	2
1	Plant No	Home	171	2
1	Plant No	Home	236	3

Result 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 6. List all payments including those with no matching orders (RIGHT JOIN)

SELECT p.payment_id,
       p.order_id,
       p.method AS payment_method,
       p.amount_paid,
       p.payment_date,
       o.customer_id,
       o.order_date,
       o.status
FROM orders o
RIGHT JOIN payments p
       ON o.order_id = p.order_id
ORDER BY p.payment_id;

```

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

payment_id	order_id	payment_method	amount_paid	payment_date	customer_id	order_date	status
1	1	Credit Card	9414.28	2025-03-02 08:16:11	20	2025-03-02 07:20:11	Delivered
2	2	Net Banking	532.20	2024-10-09 19:00:21	18	2024-10-09 18:08:21	Shipped
3	3	Credit Card	5164.56	2025-05-08 00:55:27	15	2025-05-08 00:08:27	Cancelled
4	4	UPI	9469.78	2024-09-19 22:28:13	11	2024-09-19 22:16:13	Delivered
5	5	UPI	14501.86	2025-04-08 18:26:06	12	2025-04-08 18:02:06	Pending
6	6	UPI	31050.17	2024-10-25 08:16:59	29	2024-10-25 07:33:59	Cancelled
7	7	UPI	3043.67	2024-07-29 12:45:47	22	2024-07-29 11:58:47	Shipped

Result 3

```sql
-- QUESTIONS WITH ANSWERS
-- Level 4: Multi-Table Queries (JOINS)
-- 7. Combine data from three tables: customer, order, and payment
SELECT c.customer_id,
       c.name AS customer_name,
       c.email,
       o.order_id,
       o.order_date,
       o.total_amount,
       o.status AS order_status,
       p.payment_id,
       p.method AS payment_method,
       p.amount_paid AS payment_amount,
       p.payment_date
FROM customers c
JOIN orders o
       ON c.customer_id = o.customer_id
LEFT JOIN payments p
       ON o.order_id = p.order_id
ORDER BY c.customer_id, o.order_date;

```

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

customer_id	customer_name	email	order_id	order_date	total_amount	order_status	payment_id	payment_method	payment_amount	payment_date
1	Thomas Owens	user1@example.com	17	2024-08-19 21:17:57	11173.77	Pending	17	UPI	11173.77	2024-08-19 21:49:57
1	Thomas Owens	user1@example.com	144	2024-09-19 09:18:22	12093.54	Shipped	144	UPI	12093.54	2024-09-19 10:10:22
1	Thomas Owens	user1@example.com	14	2024-09-24 21:21:38	15803.34	Shipped	14	UPI	15803.34	2024-09-24 22:11:38
1	Thomas Owens	user1@example.com	221	2024-09-25 05:49:39	12437.61	Pending	221	UPI	12437.61	2024-09-25 06:25:39
1	Thomas Owens	user1@example.com	92	2024-09-26 11:33:58	834.75	Shipped	92	Net Banking	834.75	2024-09-26 12:03:58
1	Thomas Owens	user1@example.com	61	2024-12-26 23:21:58	13053.00	Shipped	61	Net Banking	13053.00	2024-12-26 23:31:58

Level 5: Subqueries (Inner Queries)

1. List all products priced above the average product price
Used by pricing analysts to identify premium-priced products.

2. Find customers who have placed at least one order
Used to identify active customers for loyalty campaigns.

3. Show orders whose total amount is above the average for that customer
Used to detect unusually high purchases per customer.

4. Display customers who haven't placed any orders
Used for re-engagement campaigns targeting inactive users.

5. Show products that were never ordered
Helps with inventory clearance decisions or product deactivation.

6. Show highest value order per customer
Used to identify the largest transaction made by each customer.

7. Highest Order Per Customer (Including Names)
Used to identify the largest transaction made by each customer. Outputs name as well.

```sql
-- QUESTIONS WITH ANSWERS
-- Level 5: Subqueries (Inner Queries)
-- 1. List all products priced above the average product price

SELECT * FROM products
WHERE price > (SELECT AVG(price) FROM products);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	name	category	price	stock_quantity	added_on
2	Population Social	Clothing	4813.68	84	2025-05-30 10:02:50
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
7	Response Indeed	Clothing	4897.36	36	2025-03-29 02:43:08
8	Every Amount	Home	4173.60	156	2025-04-30 03:11:10
10	Development System	Electronics	4801.78	153	2025-03-12 08:22:57
12	Action Ask	Electronics	4017.01	19	2025-02-14 03:38:06
18	Some Them	Toys	3673.86	110	2024-07-25 00:33:53

products 3

```sql
-- QUESTIONS WITH ANSWERS
-- Level 5: Subqueries (Inner Queries)
-- 2. Find customers who have placed at least one order

SELECT c.customer_id, c.name, c.email
FROM customers c
WHERE EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.customer_id = c.customer_id
);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

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
7	Briana Wright	user7@example.com
8	John Bryan	user8@example.com

customers 5 ×

Output

```sql
-- QUESTIONS WITH ANSWERS
-- Level 5: Subqueries (Inner Queries)
-- 3. Show orders whose total amount is above the average for that customer

SELECT o.order_id,
       o.customer_id,
       o.total_amount,
       o.order_date
FROM orders o
WHERE o.total_amount > (
    SELECT AVG(o2.total_amount)
    FROM orders o2
    WHERE o2.customer_id = o.customer_id
);

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cont

order_id	customer_id	total_amount	order_date
6	29	31050.17	2024-10-25 07:33:59
8	19	32714.06	2024-07-30 22:49:49
9	6	24219.20	2025-06-10 17:00:25
10	28	24342.52	2025-02-16 12:45:59
14	1	15803.34	2024-09-24 21:21:38
16	5	22856.73	2024-09-09 03:00:40
18	13	32001.24	2024-06-15 12:57:04
21	22	25364.11	2025-03-12 18:53:46
22	20	21281.44	2025-02-21 11:21:06

orders 1

```sql
-- QUESTIONS WITH ANSWERS
-- Level 5: Subqueries (Inner Queries)
-- 4. Display customers who haven’t placed any orders

SELECT * FROM customers c
WHERE NOT EXISTS (
    SELECT 1 FROM orders o WHERE o.customer_id=c.customer_id
);

-- No Customer in it which haven't placed any order
```

Result Grid | Filter Rows: | Edit: | Export/Import:

customer_id	name	email	phone	created_at
NULL	NULL	NULL	NULL	NULL

```sql
-- Level 5: Subqueries (Inner Queries)
-- 5. Show products that were never ordered

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

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	name	category	price
NULL	NULL	NULL	NULL

```sql
-- QUESTIONS WITH ANSWERS
-- Level 5: Subqueries (Inner Queries)
-- 6. Show highest value order per customer

SELECT o.order_id, o.customer_id, o.total_amount
FROM orders o
WHERE o.total_amount = (
    SELECT MAX(total_amount)
    FROM orders
    WHERE customer_id=o.customer_id
);
```

Result Grid | Filter Rows: | Edit: | Export/

order_id	customer_id	total_amount
18	13	32001.24
33	8	36147.09
62	10	35723.17
66	27	26176.05
68	24	36159.92
80	12	38656.26
107	14	48042.06
121	16	37415.67
128	7	28589.04
133	11	37129.82
139	18	39857.19
174	19	35676.00
189	25	40872.29
196	21	24096.64

orders 1 ×

Level 6: Set Operations

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

Used to identify engaged customers from different activity areas.

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

Used to identify highly engaged customers for rewards.

```sql
-- QUESTIONS WITH ANSWERS
-- Level 6: Set Operations
-- 1. List all customers who have either placed an order or written a product review

SELECT customer_id FROM orders
UNION
SELECT customer_id FROM product_reviews;

-- All 30 Customers have written a product review
```

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

customer_id
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Result 5

```sql
-- QUESTIONS WITH ANSWERS
-- Level 6: Set Operations
-- 2. List all customers who have placed an order as well as reviewed a product

SELECT c.customer_id, c.name, c.email
FROM customers c
WHERE c.customer_id IN (SELECT customer_id FROM orders)
  AND c.customer_id IN (SELECT customer_id FROM product_reviews);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	name	email
1	Thomas Owens	user1@example.com
2	Charles Grant	user2@example.com
4	Christina Williams	user4@example.com
6	Mark Duke	user6@example.com
7	Briana Wright	user7@example.com
9	Jason Thompson	user9@example.com
10	Shawn Hill	user10@example.com
11	Walter Jenkins	user11@example.com

customers 4

The End

AYUSH KUMAR SIDDHARTH