📸 **SCREENSHOT # 1**

— Customers table (sample rows)

 # SOLUTION

```
SELECT * FROM customers;
```

customer_id	name	email	country
1	Asha Verma	asha@example.com	India
2	John Doe	john@example.com	USA
3	Sara Khan	sara@example.com	India
4	Liu Wei	liu@example.com	China
5	Carlos Mendez	carlos@example.com	Spain
NULL	NULL	NULL	NULL

📸 SCREENSHOT # 2

— Orders table (sample rows)

 # SOLUTION

```
SELECT * FROM orders;
```

	order_id	customer_id	order_date
▶	1	1	2025-11-20
	2	2	2025-11-21
	3	1	2025-11-22
	4	3	2025-11-23
	5	4	2025-11-24
*	NULL	NULL	NULL

📸 SCREENSHOT # 3

— Products table (sample rows)

 # SOLUTION

```
SELECT * FROM products;
```

	product_id	product_name	category
▶	1	Wireless Mouse	Accessories
	2	Gaming Keyboard	Accessories
	3	Office Chair	Furniture
	4	Bluetooth Speaker	Electronics
	5	Notebook	Stationery
*	NULL	NULL	NULL

📸 SCREENSHOT # 4

— Order Items table (sample rows)

 # SOLUTION

```
SELECT * FROM order_items;
```

item_id	order_id	product_id	quantity	price
1	1	1	2	12.50
2	1	5	5	2.00
3	2	2	1	45.00
4	3	3	1	120.00
5	3	1	1	12.50
6	4	4	2	30.00
7	5	2	3	45.00
8	5	5	10	2.00
NULL	NULL	NULL	NULL	NULL

Aggregations / Grouping

📸 **SCREENSHOT # 5 — Revenue by country**

— Explanation: join customers -> orders -> order_items, sum quantity*price per country.

 # SOLUTION

```
SELECT c.country, SUM(oi.quantity * oi.price) AS total_revenue
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY c.country
ORDER BY total_revenue DESC;
```

country	total_revenue
India	227.50
China	155.00
USA	45.00

Aggregations / Grouping

📸 **SCREENSHOT # 6 — Revenue by category**

— Explanation: group order items by product category to find category revenue.

 # SOLUTION

```
SELECT p.category, SUM(oi.quantity * oi.price) AS revenue
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.category
ORDER BY revenue DESC;
```

category	revenue
Accessories	217.50
Furniture	120.00
Electronics	60.00
Stationery	30.00

Joins examples

📸 **SCREENSHOT # 7 — INNER JOIN (customers who have orders)**

 # SOLUTION

```
SELECT DISTINCT c.customer_id, c.name, o.order_id,
o.order_date
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
ORDER BY c.customer_id;
```

customer_id	name	order_id	order_date
1	Asha Verma	1	2025-11-20
1	Asha Verma	3	2025-11-22
2	John Doe	2	2025-11-21
3	Sara Khan	4	2025-11-23
4	Liu Wei	5	2025-11-24

Joins example

📸 **SCREENSHOT # 8 — LEFT JOIN (all customers + orders if any)**

 # SOLUTION

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

customer_id	name	order_id	order_date
1	Asha Verma	1	2025-11-20
1	Asha Verma	3	2025-11-22
2	John Doe	2	2025-11-21
3	Sara Khan	4	2025-11-23
4	Liu Wei	5	2025-11-24
5	Carlos Mendez	NULL	NULL

Joins example

📸 **SCREENSHOT # 9 — RIGHT JOIN (all orders + customer info; MySQL supports RIGHT JOIN)**

 # SOLUTION

```
SELECT c.customer_id, c.name, o.order_id, o.order_date
FROM customers c
RIGHT JOIN orders o ON c.customer_id =
o.customer_id
ORDER BY o.order_id;
```

customer_id	name	order_id	order_date
1	Asha Verma	1	2025-11-20
2	John Doe	2	2025-11-21
1	Asha Verma	3	2025-11-22
3	Sara Khan	4	2025-11-23
4	Liu Wei	5	2025-11-24

Subquery example

📸 **SCREENSHOT # 10 — Top 5 spending customers (subquery approach)**

Explanation: inner query computes total_spent per customer; outer sorts & limits.

 # SOLUTION

```
SELECT name, total_spent
FROM (
   SELECT c.name, SUM(oi.quantity * oi.price) AS total_spent
   FROM customers c
   JOIN orders o ON c.customer_id = o.customer_id
   JOIN order_items oi ON o.order_id = oi.order_id
   GROUP BY c.customer_id
) t
ORDER BY total_spent DESC
LIMIT 5;
```

name	total_spent
Asha Verma	167.50
Liu Wei	155.00
Sara Khan	60.00
John Doe	45.00

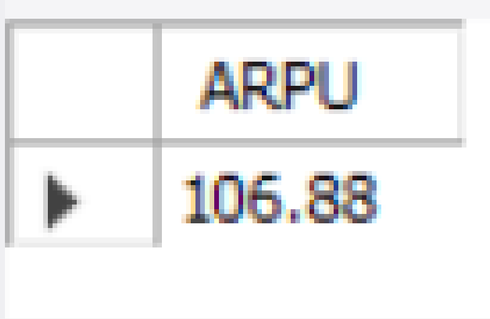
ARPU (Average Revenue Per User)

📸 **SCREENSHOT # 11 — ARPU (revenue / number of customers with orders)**

Explanation: divides total revenue by count of distinct customers who placed orders.

 # SOLUTION

```
SELECT
   ROUND(SUM(oi.quantity * oi.price) / NULLIF(COUNT(DISTINCT o.customer_id),0),2) AS ARPU
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id;
```



	ARPU
▶	106.88

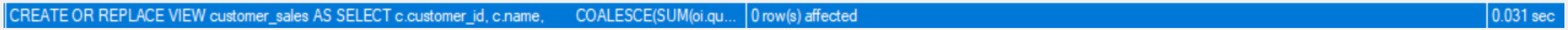
View creation

📸 **SCREENSHOT # 12 — CREATE VIEW success message**

Explanation: create a reusable view customer_sales with total_spent per customer.

 # SOLUTION

```
CREATE OR REPLACE VIEW customer_sales AS
SELECT c.customer_id, c.name,
      COALESCE(SUM(oi.quantity * oi.price),0) AS total_spent
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
LEFT JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY c.customer_id, c.name;
```



View creation

📸 **SCREENSHOT # 13 — View output (query the view)**

 # SOLUTION

```
SELECT * FROM customer_sales ORDER BY total_spent DESC;
```

customer_id	name	total_spent
1	Asha Verma	167.50
4	Liu Wei	155.00
3	Sara Khan	60.00
2	John Doe	45.00
5	Carlos Mendez	0.00

View creation

📸 **SCREENSHOT # 14 — CREATE INDEX success message**

Explanation: add index on orders.customer_id to speed joins.

 # SOLUTION

```
CREATE INDEX idx_orders_customer_id ON orders(customer_id);
```