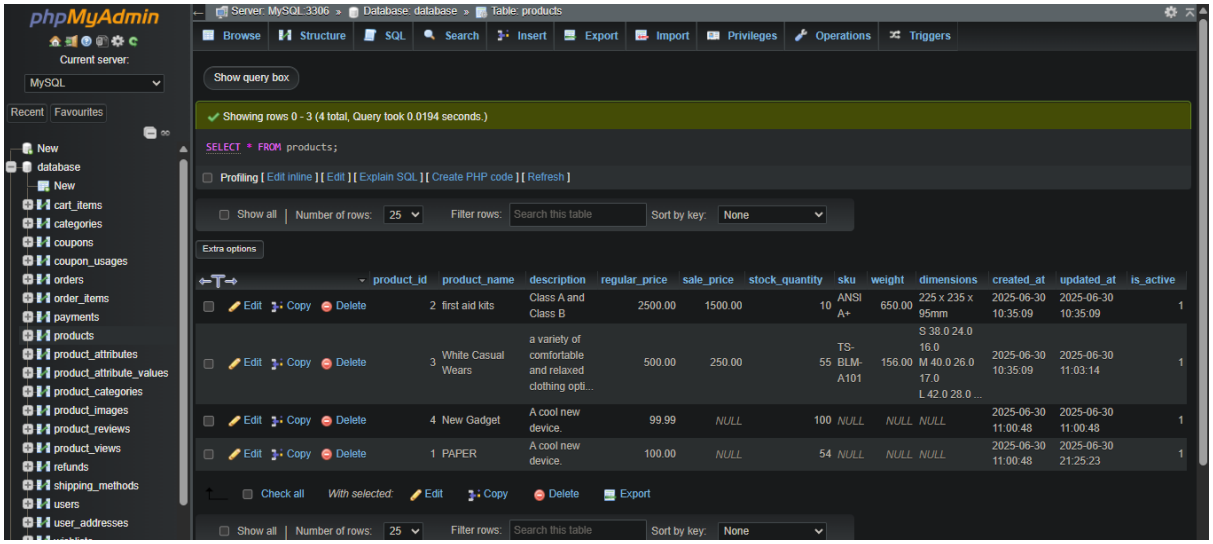
Product Management

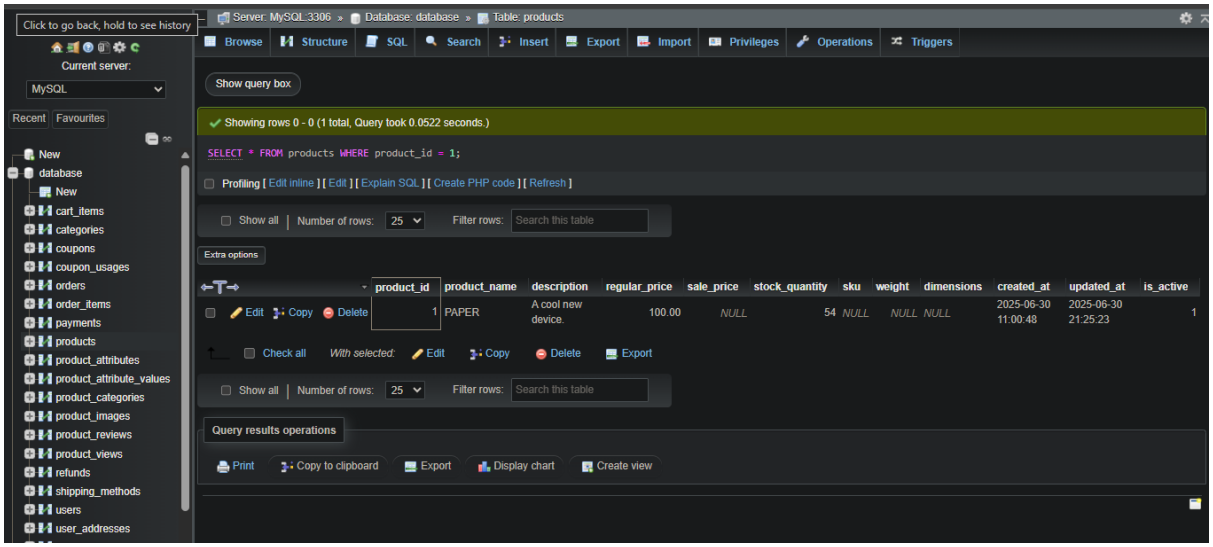
1. Get all products:

SELECT * FROM products;



2.Get product by ID:

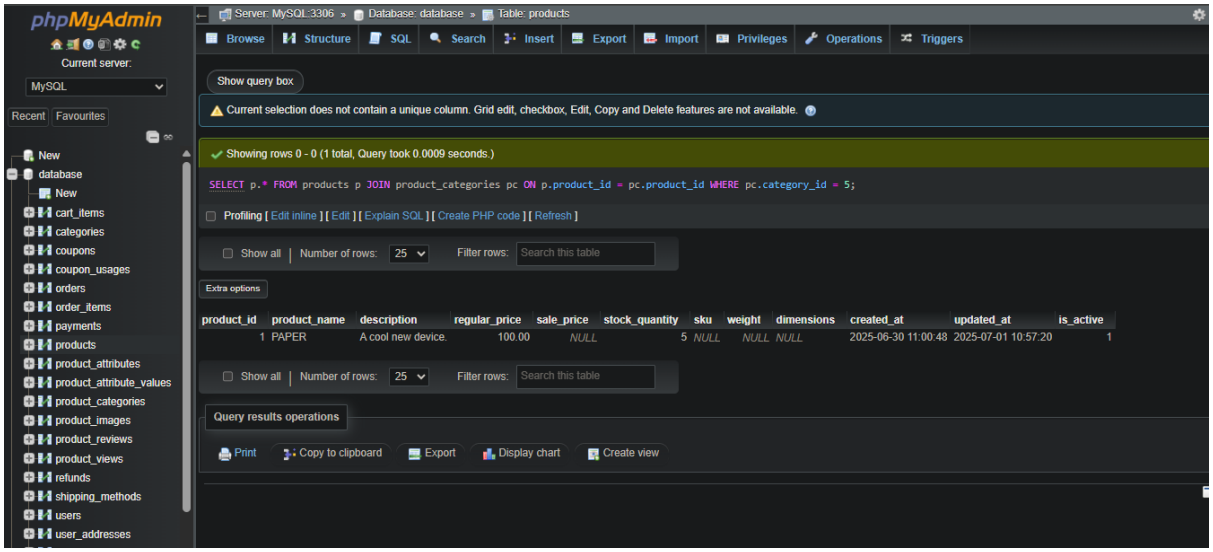
SELECT * FROM products WHERE product_id = 1



3.Get products by category:

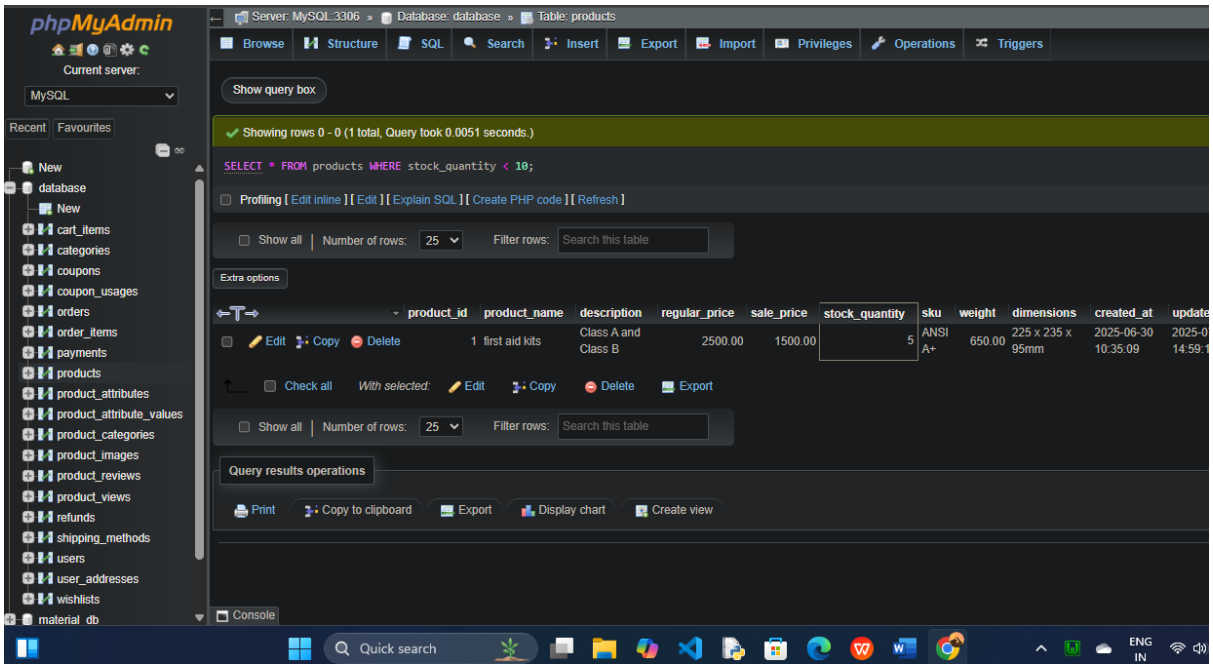
SELECT p.* FROM products p JOIN product_categories pc ON

p.product_id = pc.product_id WHERE pc.category_id = 5;



4.Get products with low stock (e.g., less than 10):

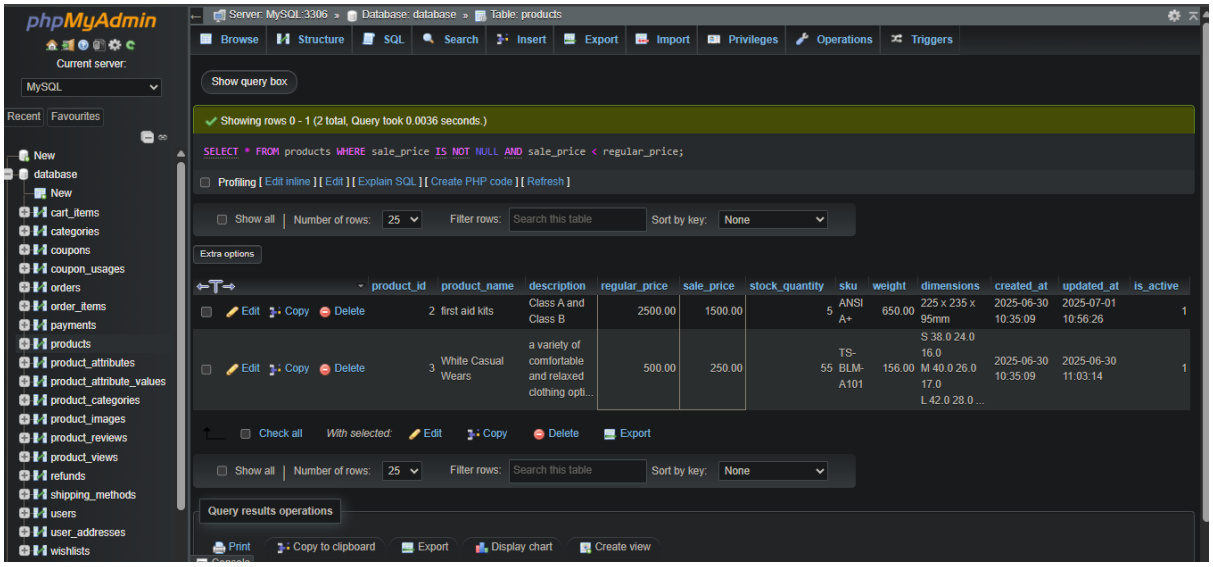
SELECT * FROM products WHERE stock_quantity < 10



5.Get products on sale:

SELECT * FROM products WHERE sale_price IS NOT NULL AND sale_price

< regular_price

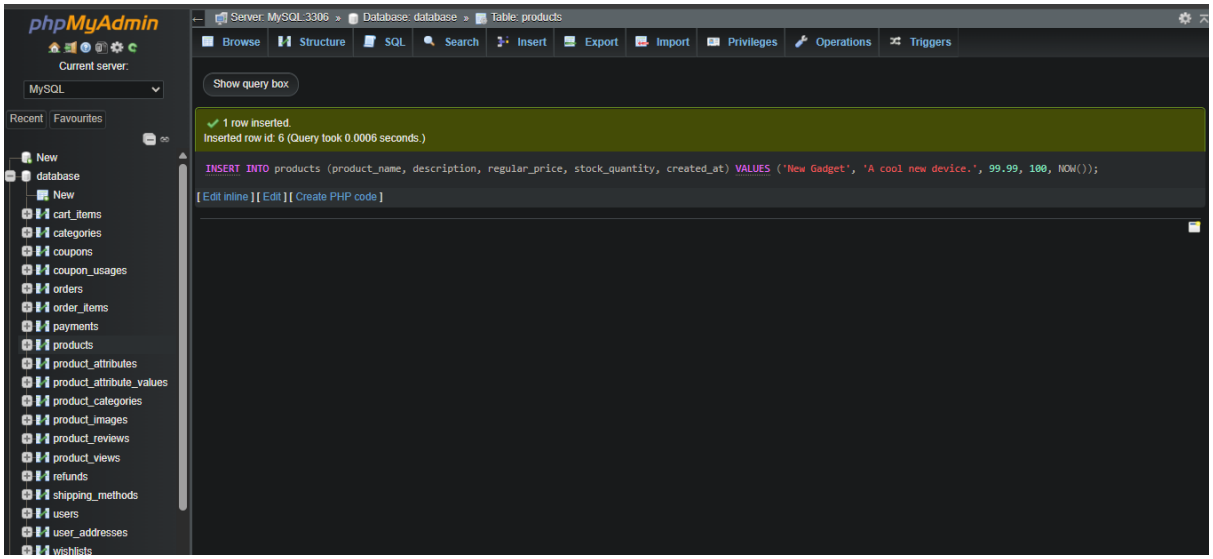


6.Add a new product:

INSERT INTO products (product_name, description, regular_price,

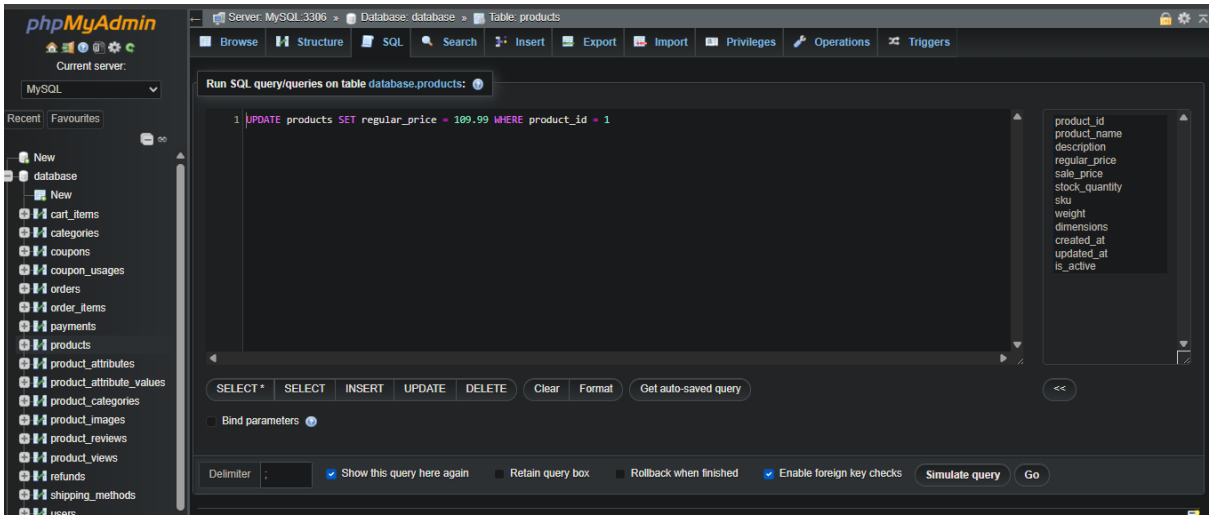
stock_quantity, created_at)

VALUES ('New Gadget', 'A cool new device.', 99.99, 100, NOW());



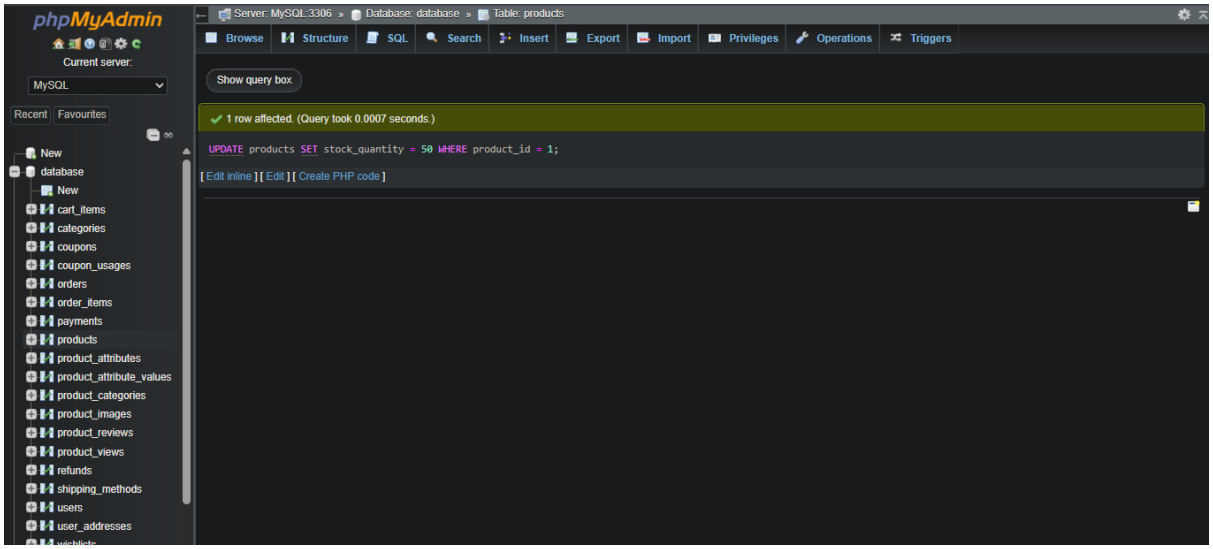
7.Update product price:

UPDATE products SET regular_price = 109.99 WHERE product_id = 1



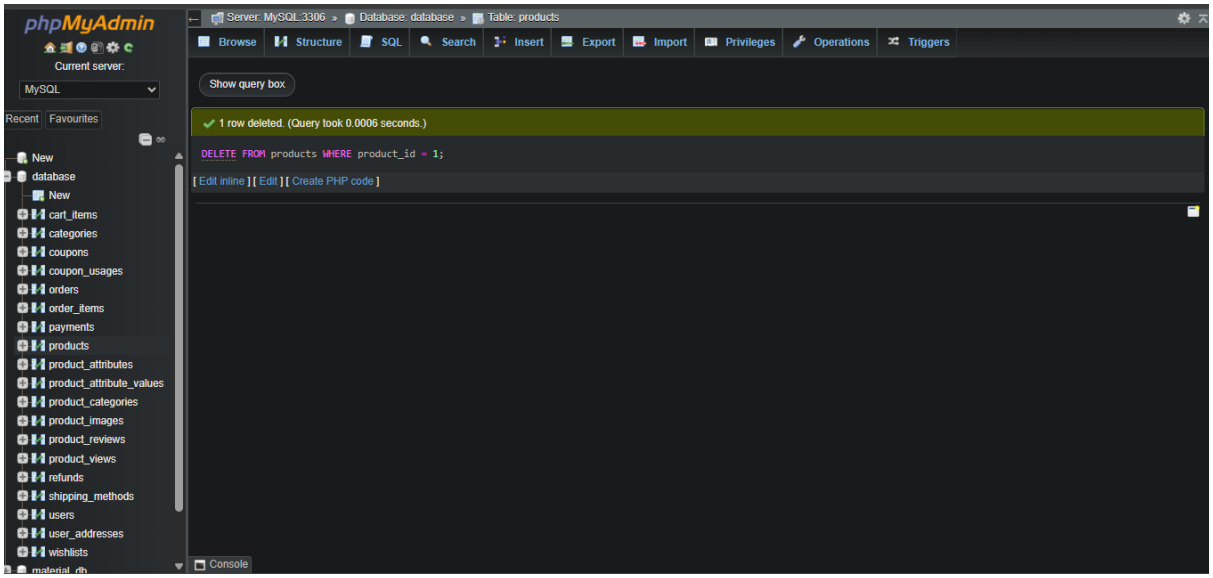
8.Update product stock:

UPDATE products SET stock_quantity = 50 WHERE product_id = 1;



9.Delete a product:

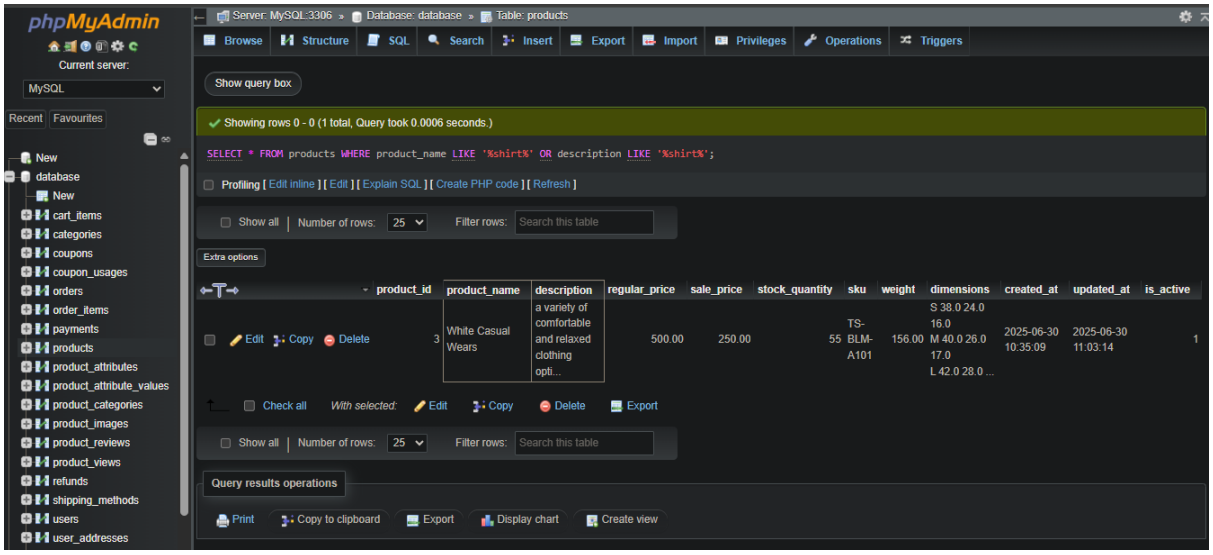
DELETE FROM products WHERE product_id = 1



10.Search products by keyword (e.g., 'shirt'):

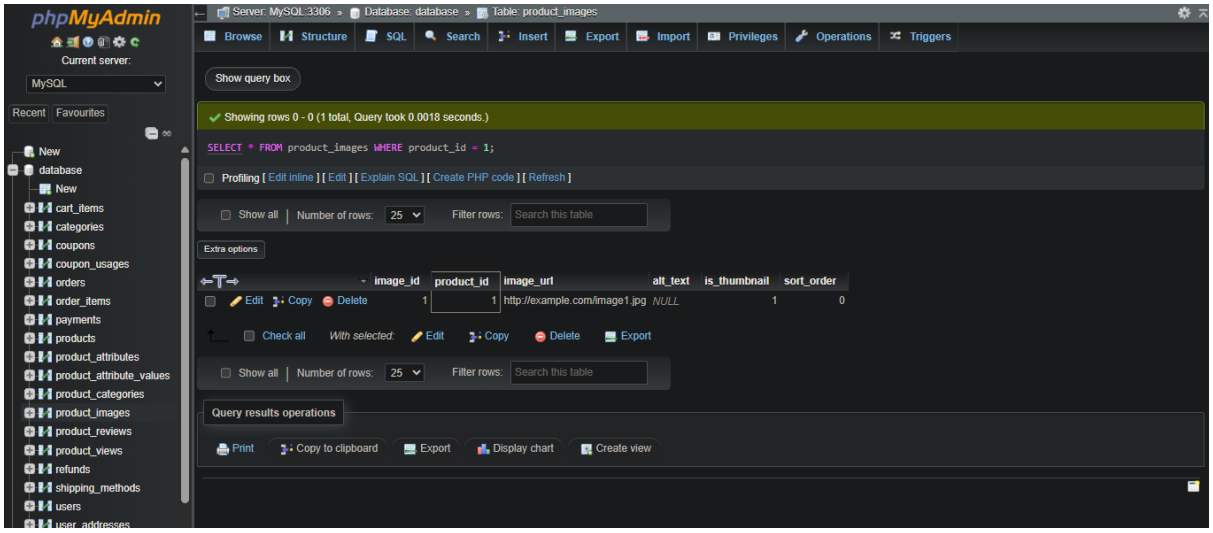
SELECT * FROM products WHERE product_name LIKE '%shirt%' OR

description LIKE '%shirt%'



11.Get product images for a product:

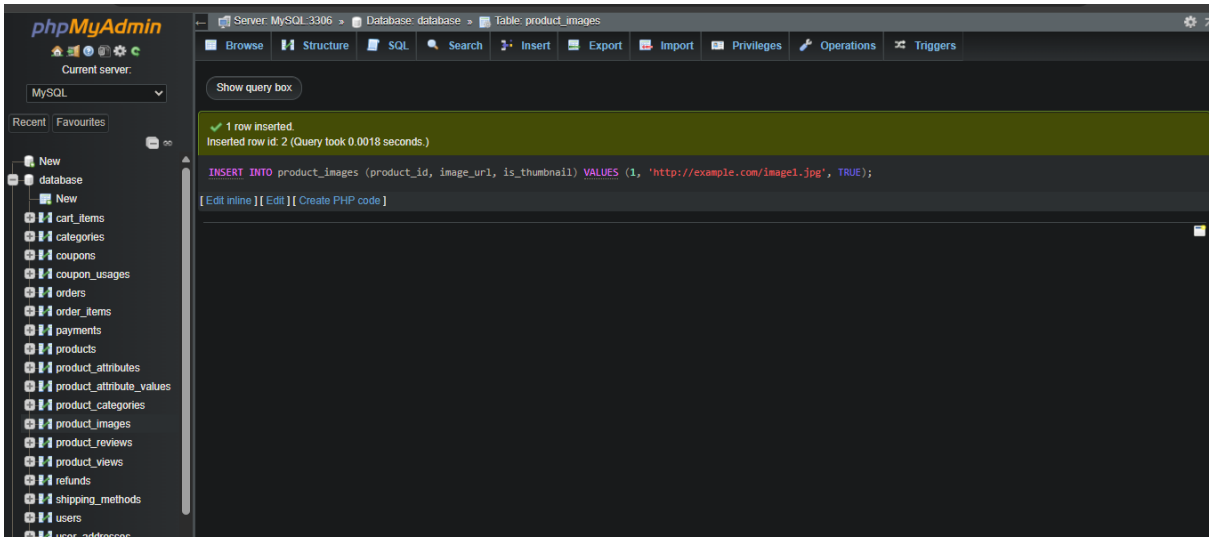
SELECT * FROM product_images WHERE product_id = 1



12.Add a new product image:

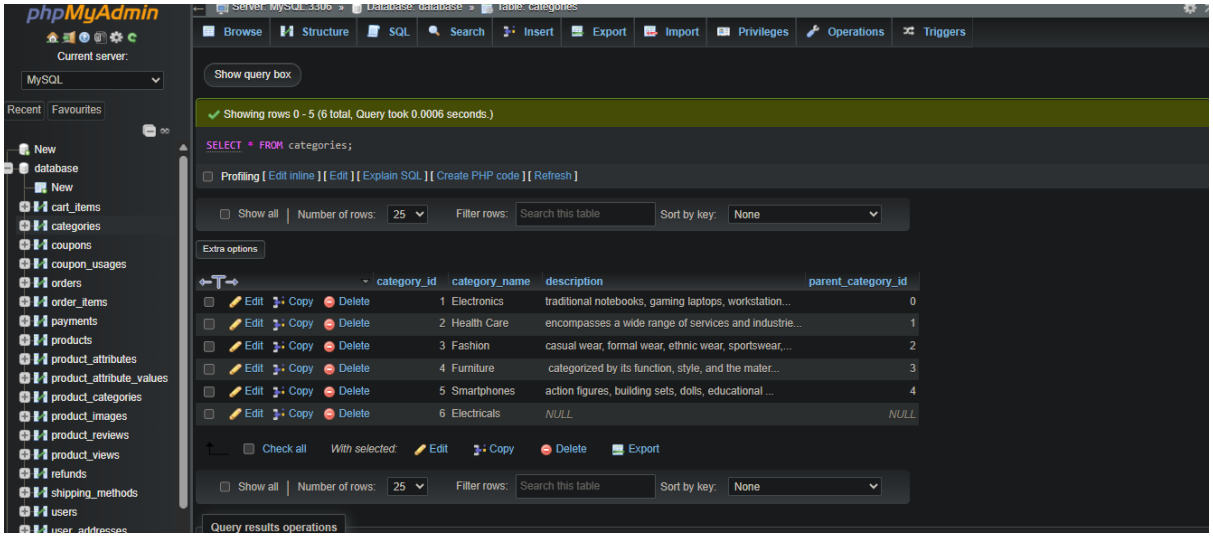
INSERT INTO product_images (product_id, image_url, is_thumbnail)

VALUES (1, 'http://example.com/image1.jpg', TRUE)



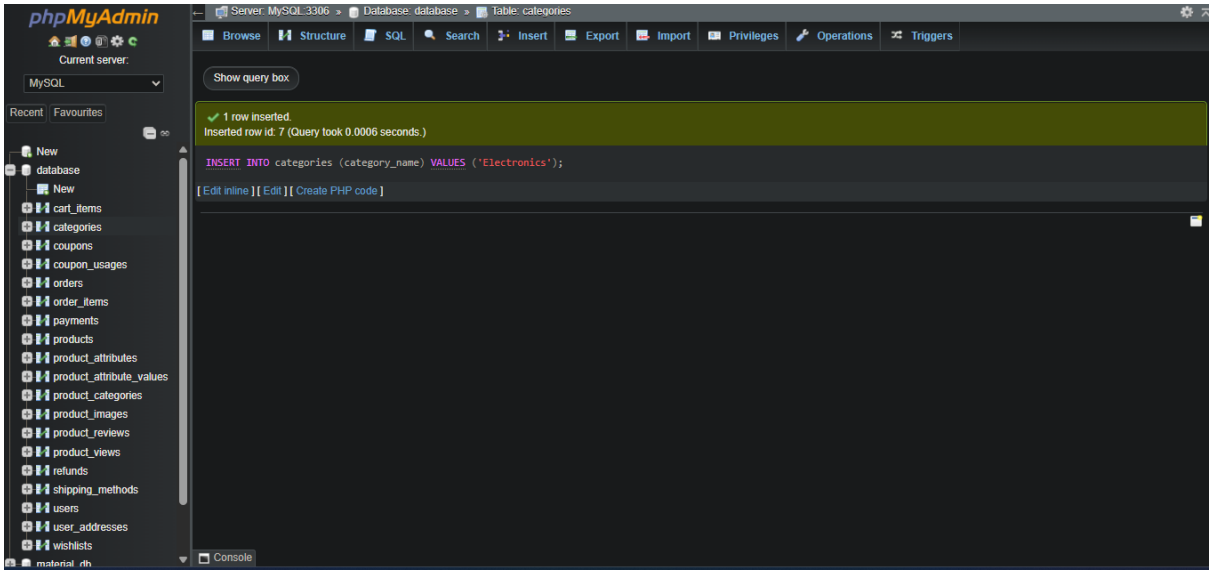
13.Get all categories:

SELECT * FROM categories



14.Add a new category:

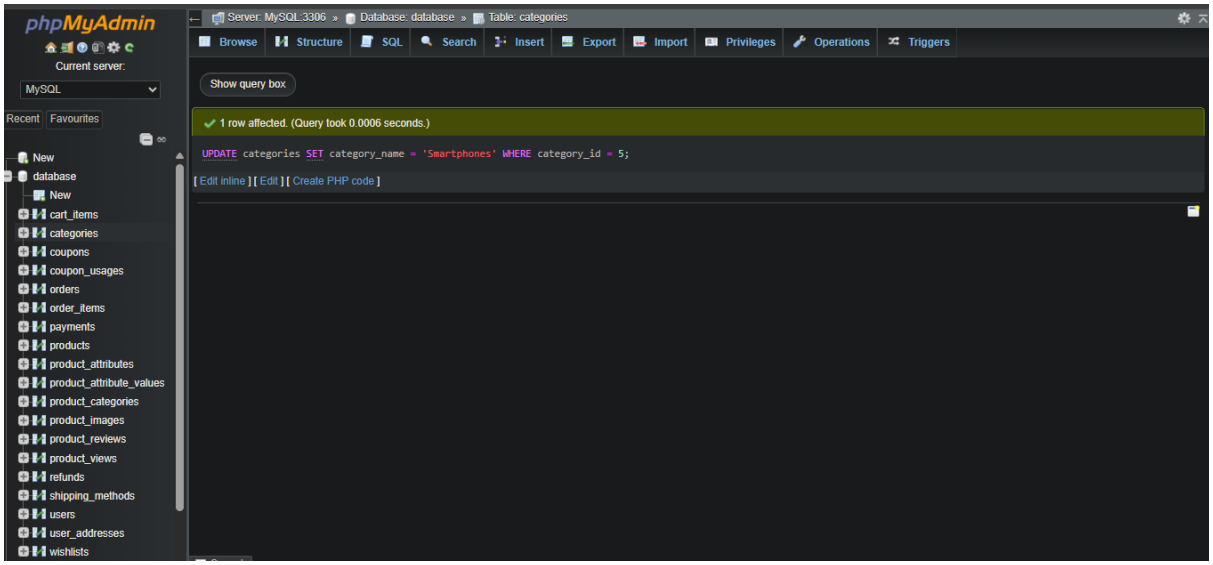
INSERT INTO categories (category_name) VALUES ('Electronics')



15.Update category name:

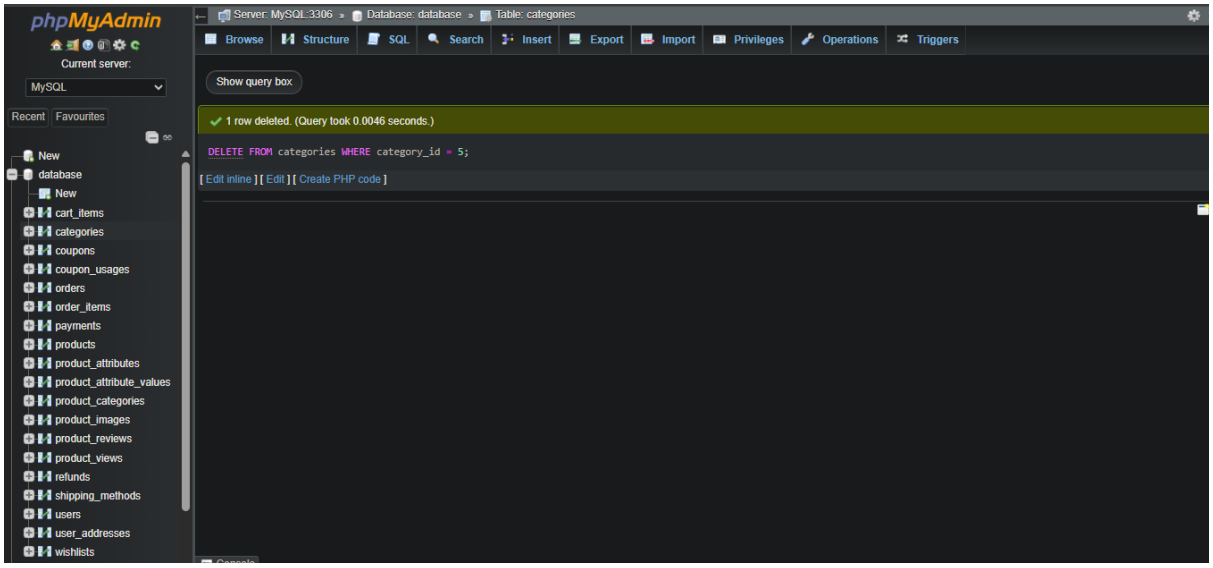
UPDATE categories SET category_name = 'Smartphones' WHERE

category_id = 5



16.Delete a category:

DELETE FROM categories WHERE category_id = 5



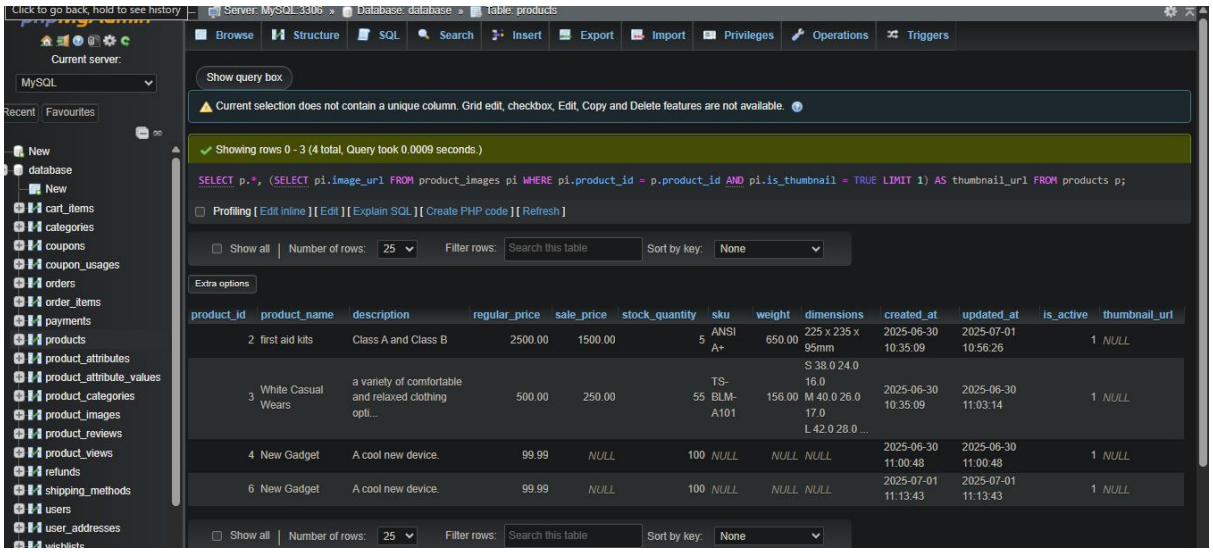
17.Get products with their primary image:

SELECT p.*, (SELECT pi.image_url FROM product_images pi WHERE

pi.product_id = p.product_id AND pi.is_thumbnail = TRUE LIMIT 1)

AS thumbnail_url

FROM products p



18.Get product attributes (e.g., size, color):

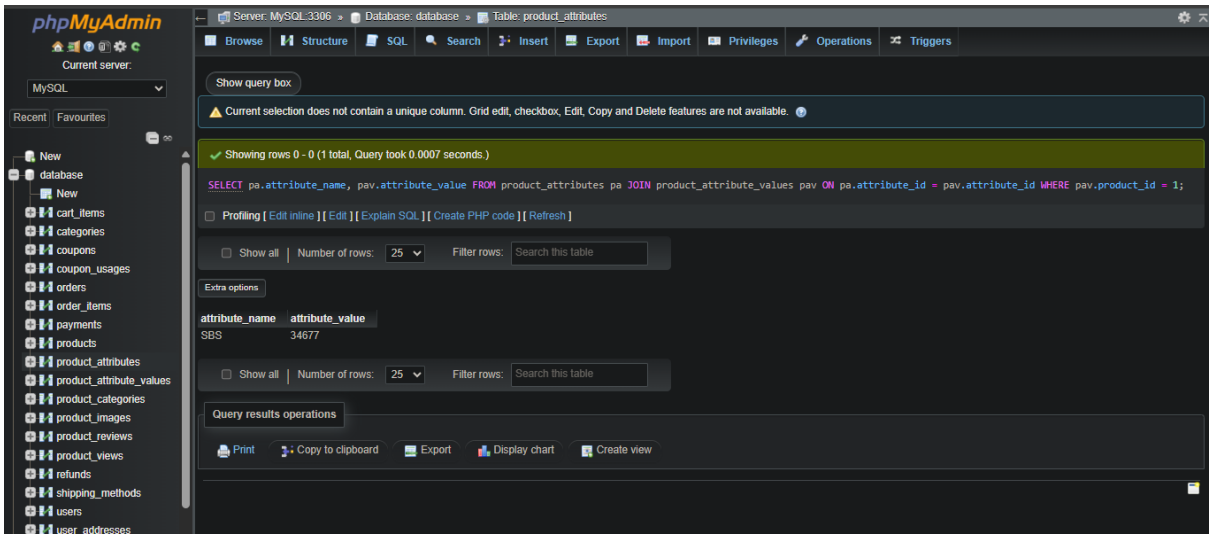
SELECT pa.attribute_name, pav.attribute_value

FROM product_attributes pa

JOIN product_attribute_values pav ON pa.attribute_id =

pav.attribute_id

WHERE pav.product_id = 1;

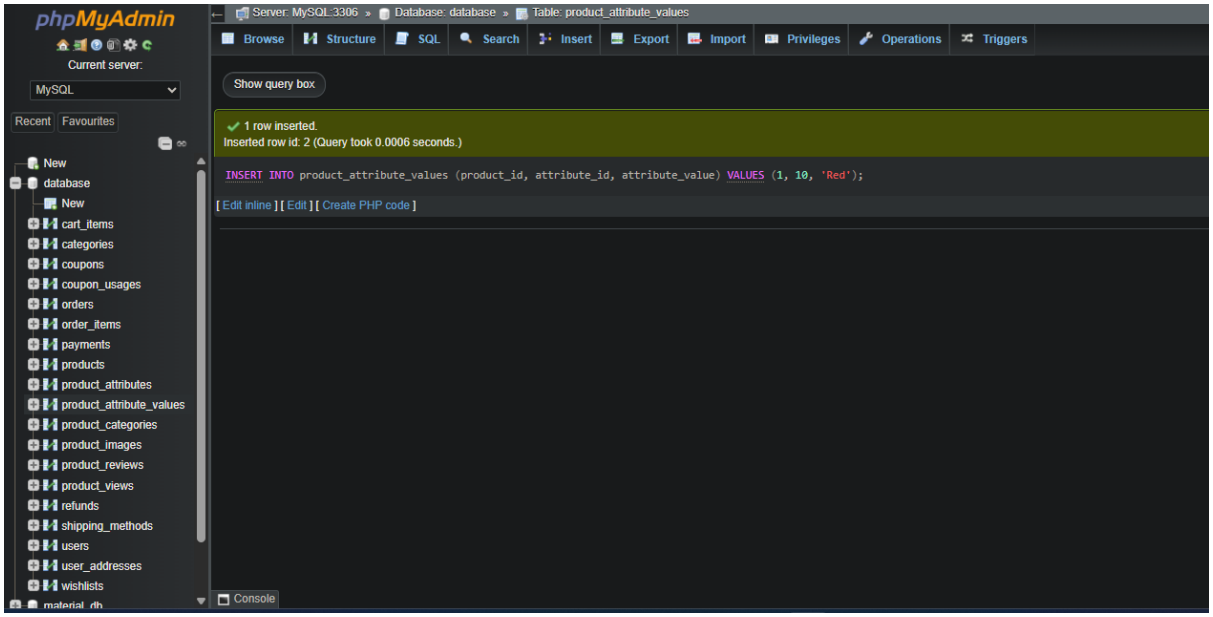


19.Add a new product attribute value:

INSERT INTO product_attribute_values (product_id, attribute_id,

attribute_value)

VALUES (1, 10, 'Red');



20.Get top 10 most viewed products:

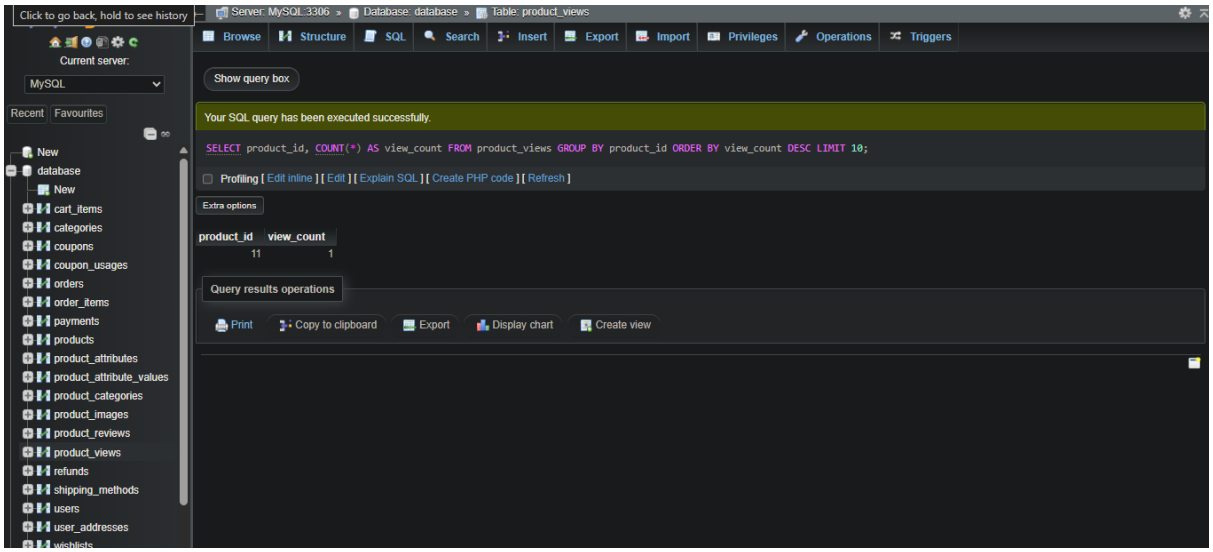
SELECT product_id, COUNT(*) AS view_count

FROM product_views

GROUP BY product_id

ORDER BY view_count DESC

LIMIT 10;



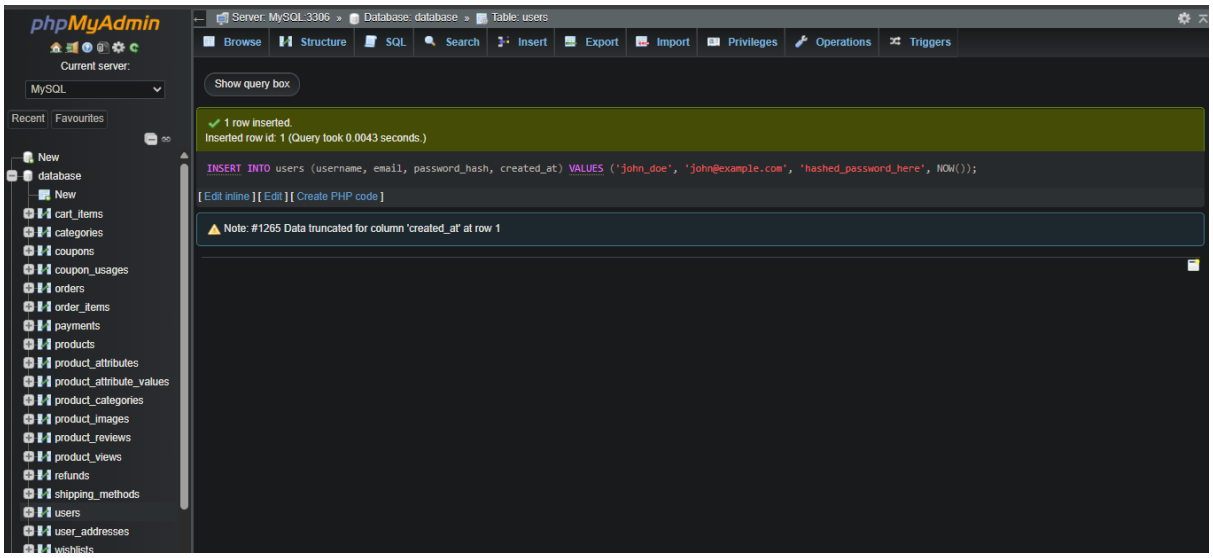
21.User & Authentication

1. Register a new user:

INSERT INTO users (username, email, password_hash, created_at)

VALUES ('john_doe', 'john@example.com', 'hashed_password_here',

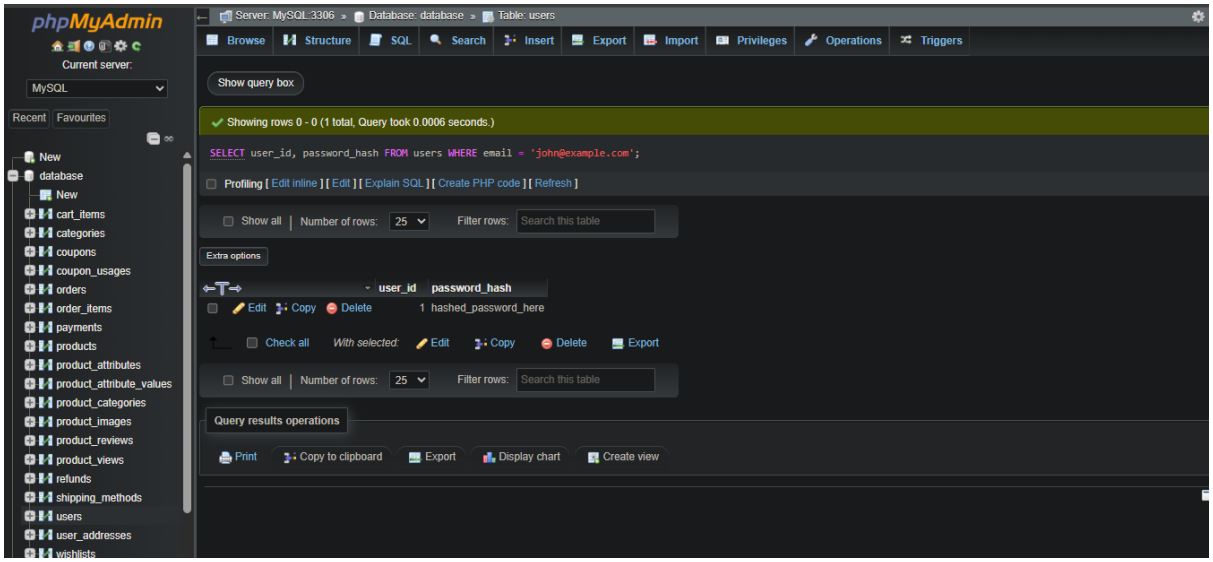
NOW())



22.login a user (retrieve password hash for verification):

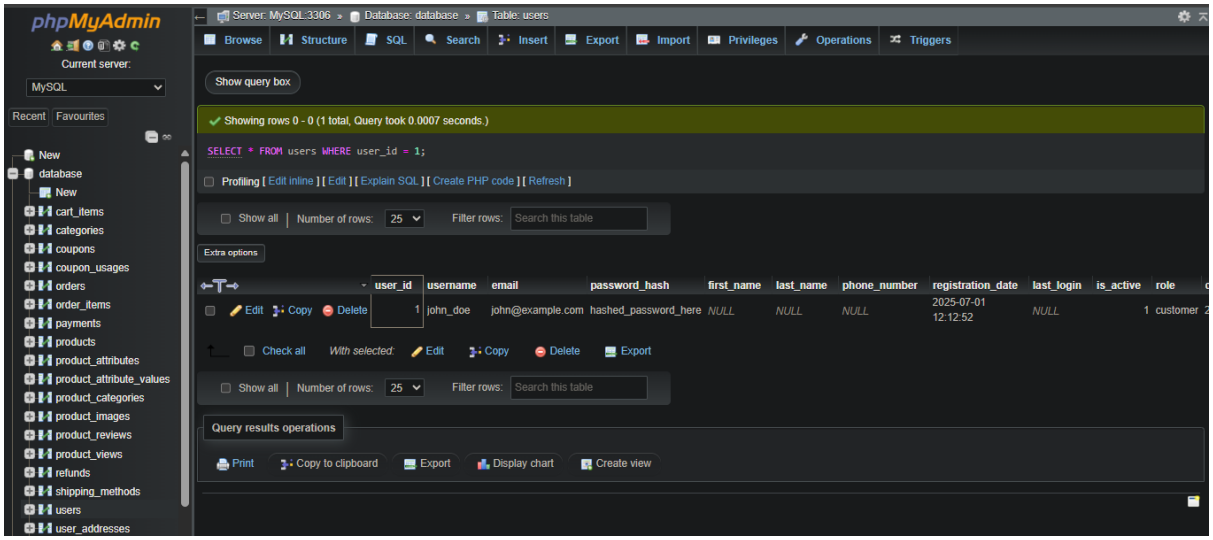
SELECT user_id, password_hash FROM users WHERE email =

'john@example.com'



23.Get user profile by ID:

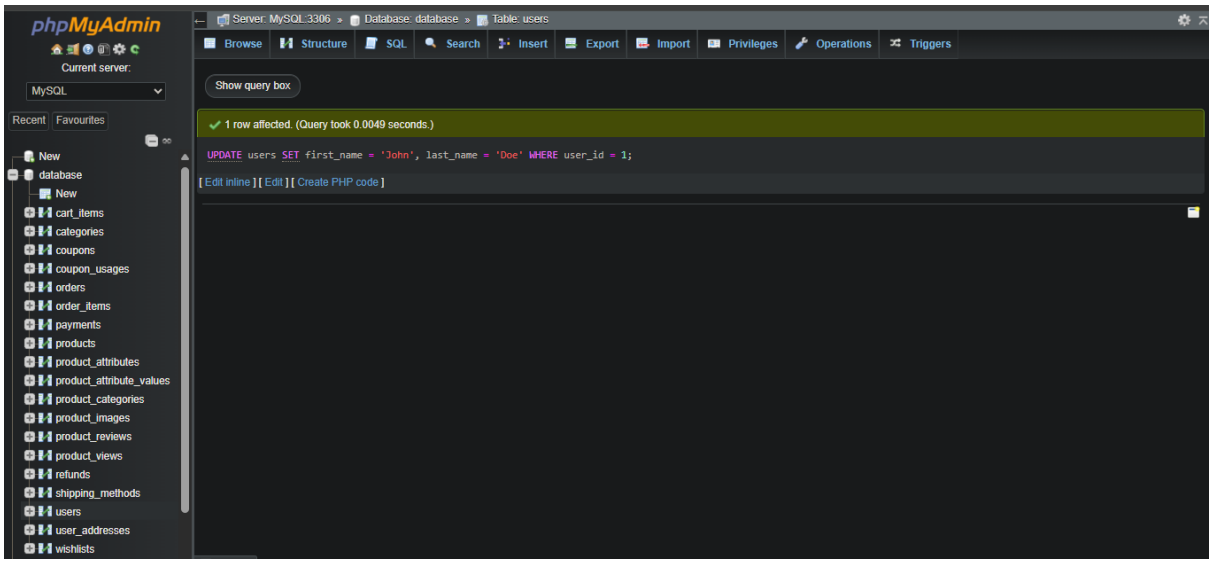
SELECT * FROM users WHERE user_id = 1;



24. Update user profile information:

UPDATE users SET first_name = 'John', last_name = 'Doe' WHERE

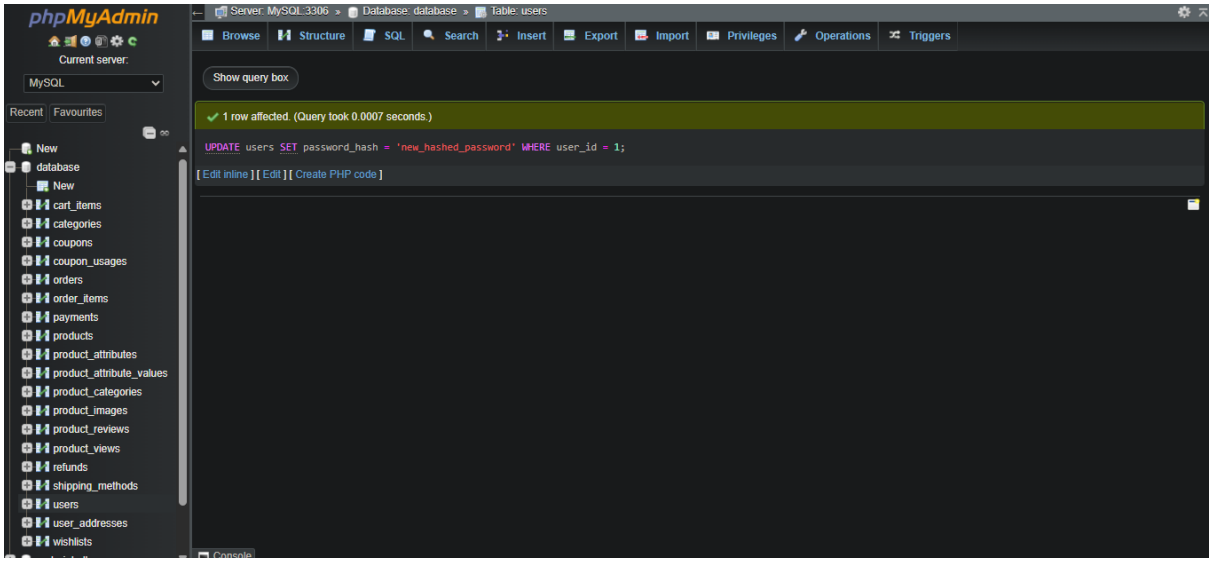
user_id = 1



25.Change user password:

UPDATE users SET password_hash = 'new_hashed_password' WHERE

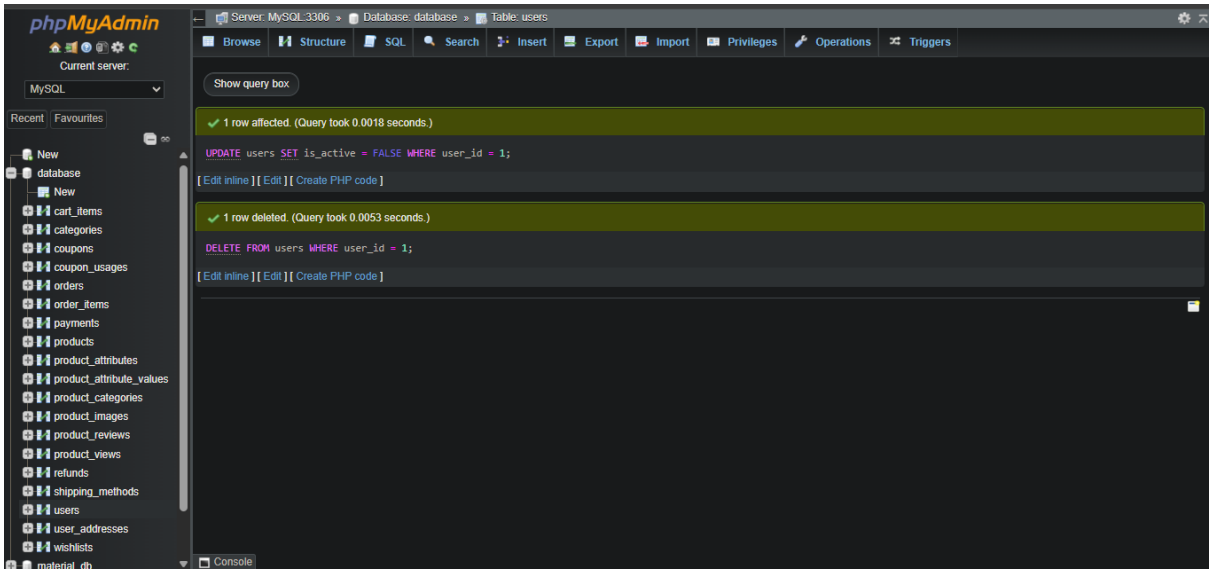
user_id = 1



26.Delete a user (soft delete recommended):

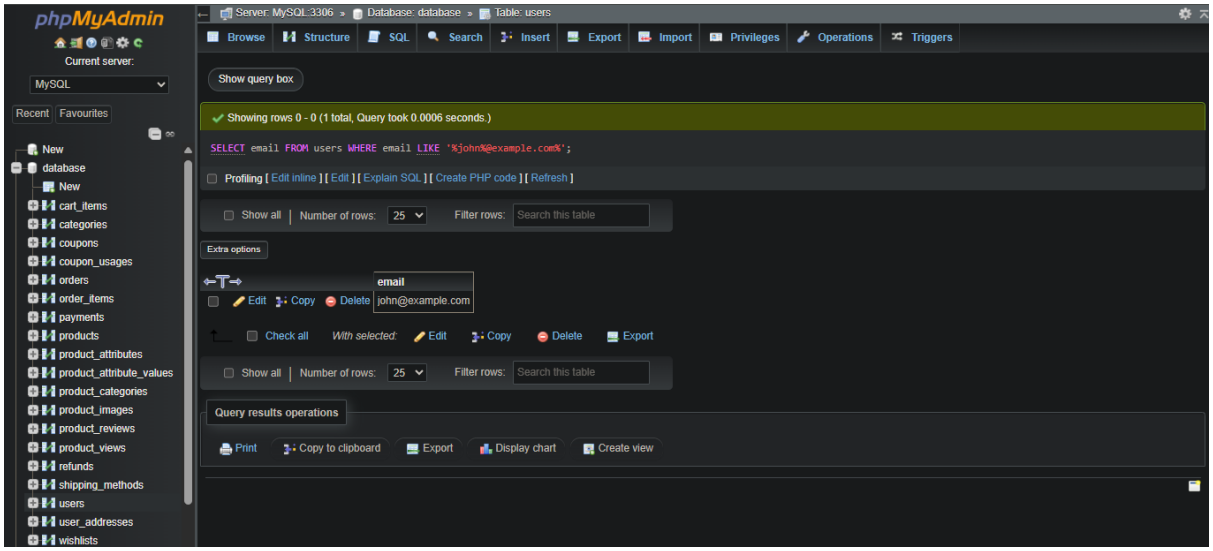
UPDATE users SET is_active = FALSE WHERE user_id = 1;

-- Or hard delete: DELETE FROM users WHERE user_id = 1



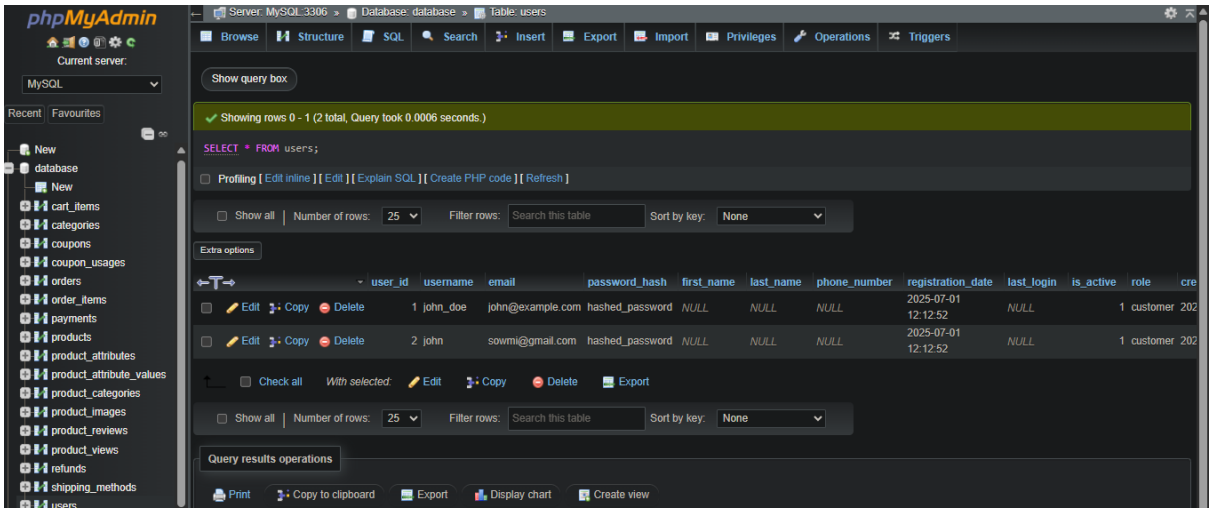
27.Check if email exists:

SELECT COUNT(*) FROM users WHERE email = '%john@example.com%'



28.Get all users (for admin panel):

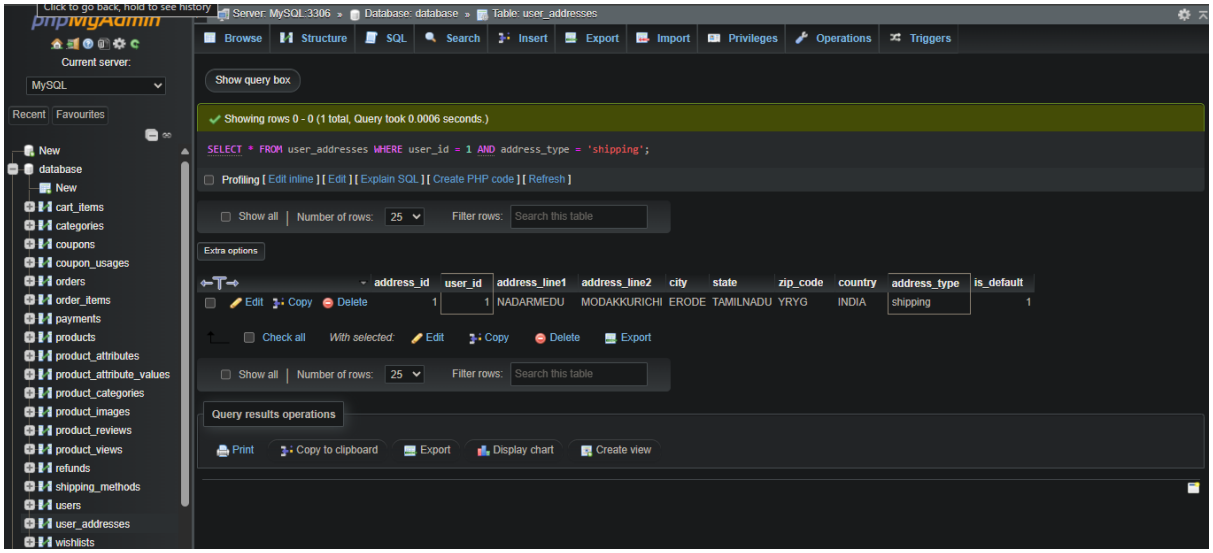
SELECT * FROM users



29.Get user's shipping addresses:

SELECT * FROM user_addresses WHERE user_id = 1 AND address_type =

'shipping';



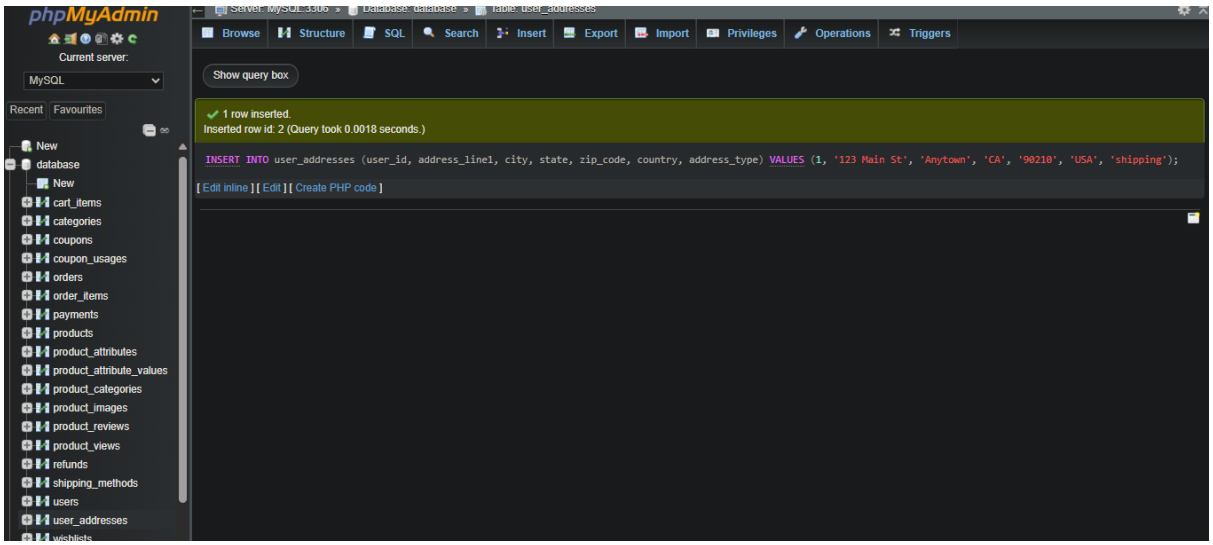
30.Add a new user address:

INSERT INTO user_addresses (user_id, address_line1, city, state,

zip_code, country, address_type)

VALUES (1, '123 Main St', 'Anytown', 'CA', '90210', 'USA',

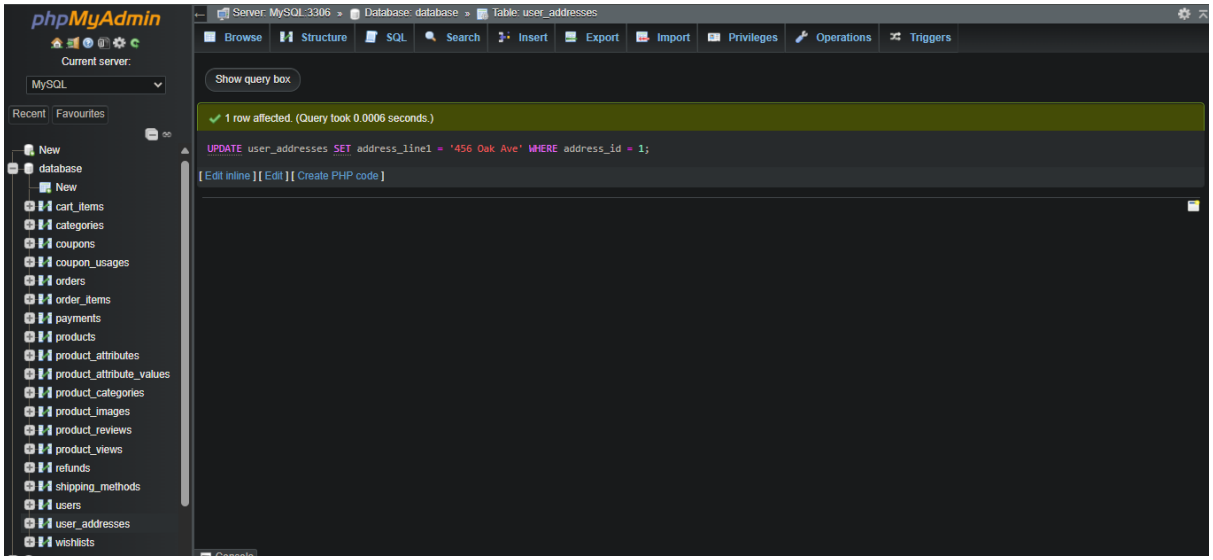
'shipping')



31.Update a user address:

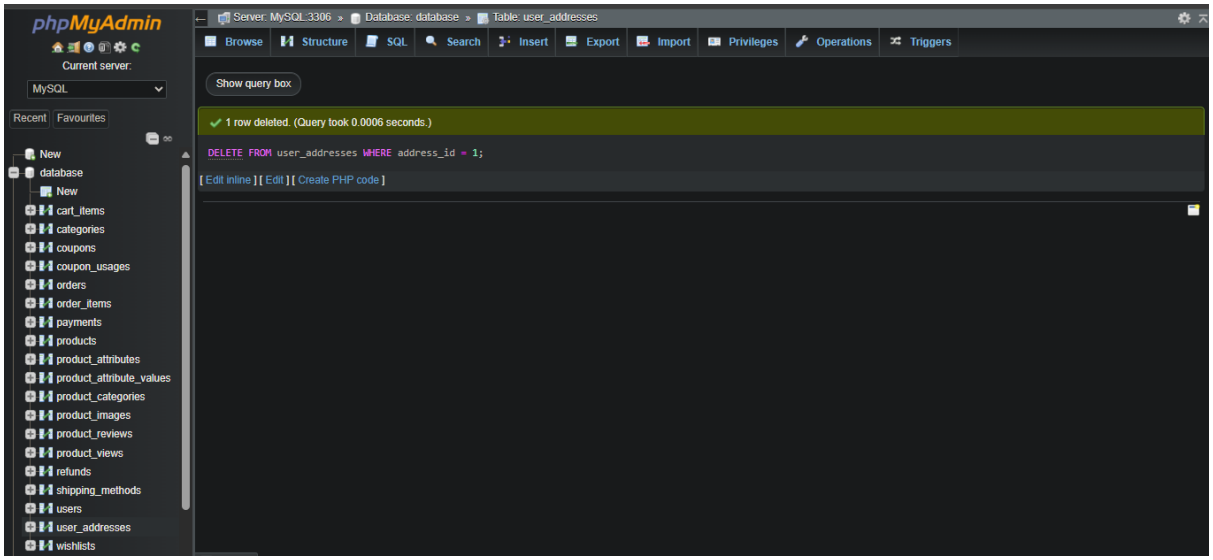
UPDATE user_addresses SET address_line1 = '456 Oak Ave' WHERE

address_id = 1;



32.Delete a user address:

DELETE FROM user_addresses WHERE address_id = 1

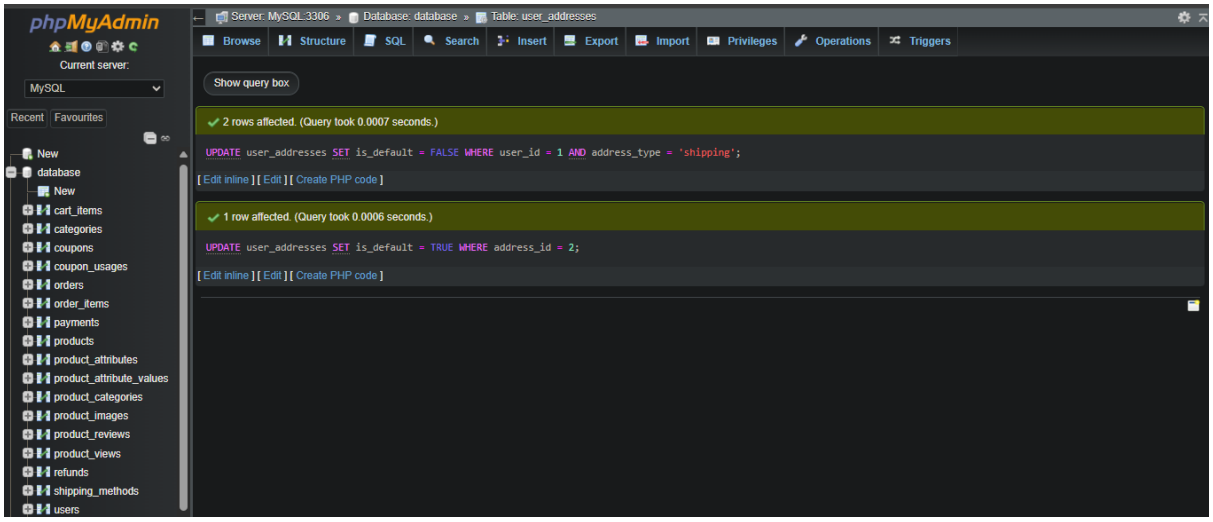


33.Set a default shipping address for a user:

UPDATE user_addresses SET is_default = FALSE WHERE user_id = 1 AND

address_type = 'shipping';

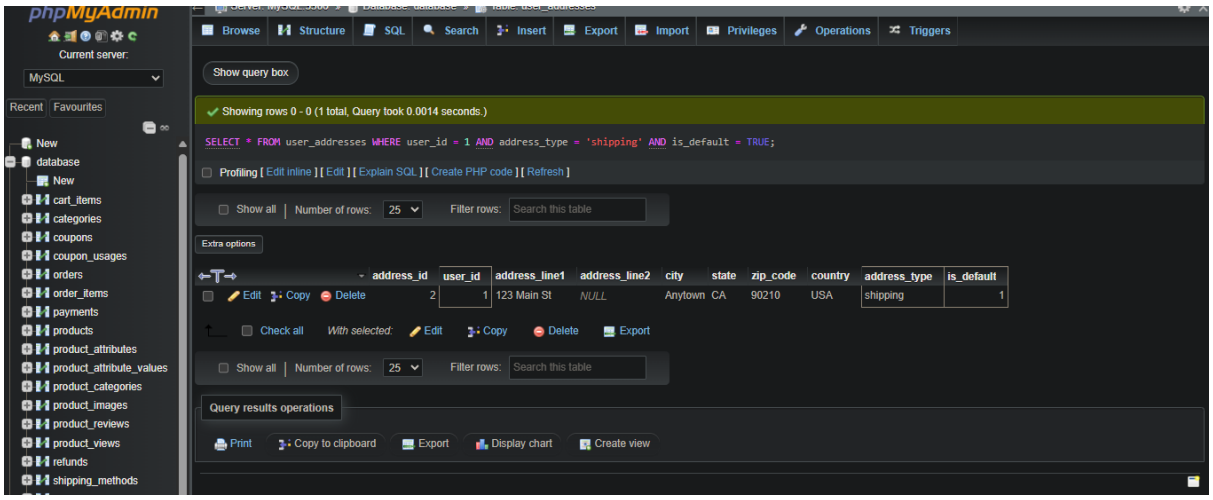
UPDATE user_addresses SET is_default = TRUE WHERE address_id = 2



34. Get default shipping address for a user:

SELECT * FROM user_addresses WHERE user_id = 1 AND address_type =

'shipping' AND is_default = TRUE



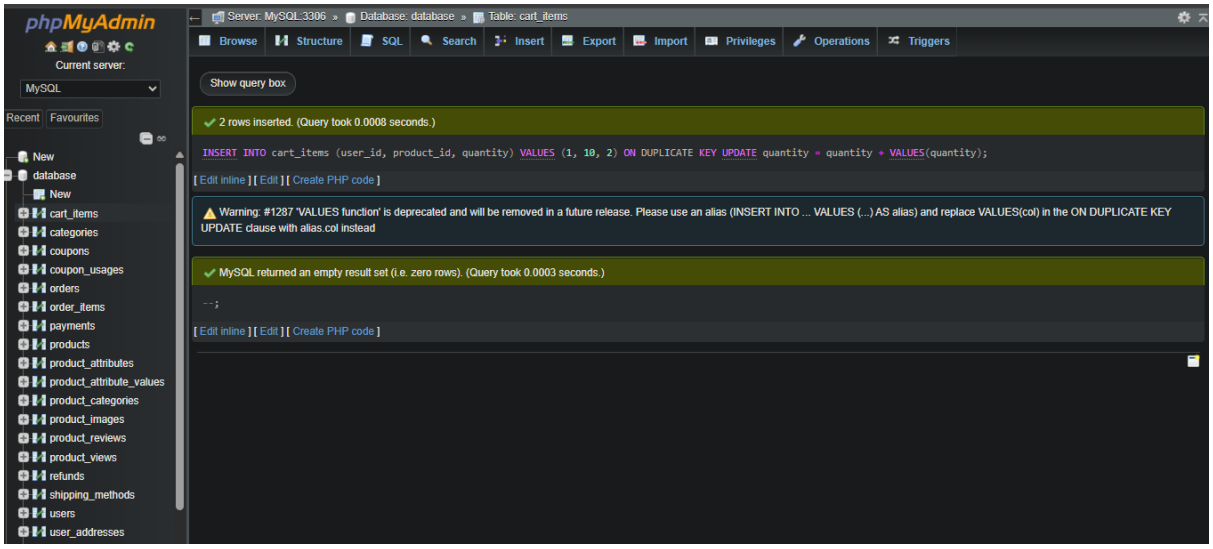
35.Shopping Cart & Wishlist

Add item to cart:

INSERT INTO cart_items (user_id, product_id, quantity)

VALUES (1, 10, 2)ON DUPLICATE KEY UPDATE quantity = quantity + VALUES(quantity); --

For updating quantity if item already exists



36.Get user's cart items:

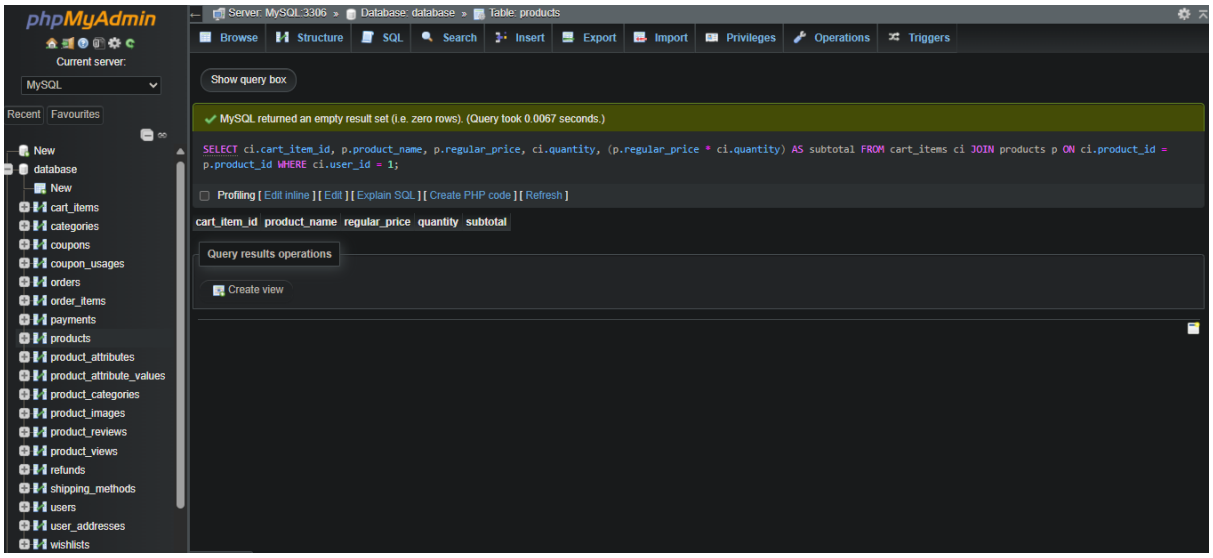
SELECT ci.cart_item_id, p.product_name, p.regular_price,

ci.quantity, (p.regular_price * ci.quantity) AS subtotal

FROM cart_items ci

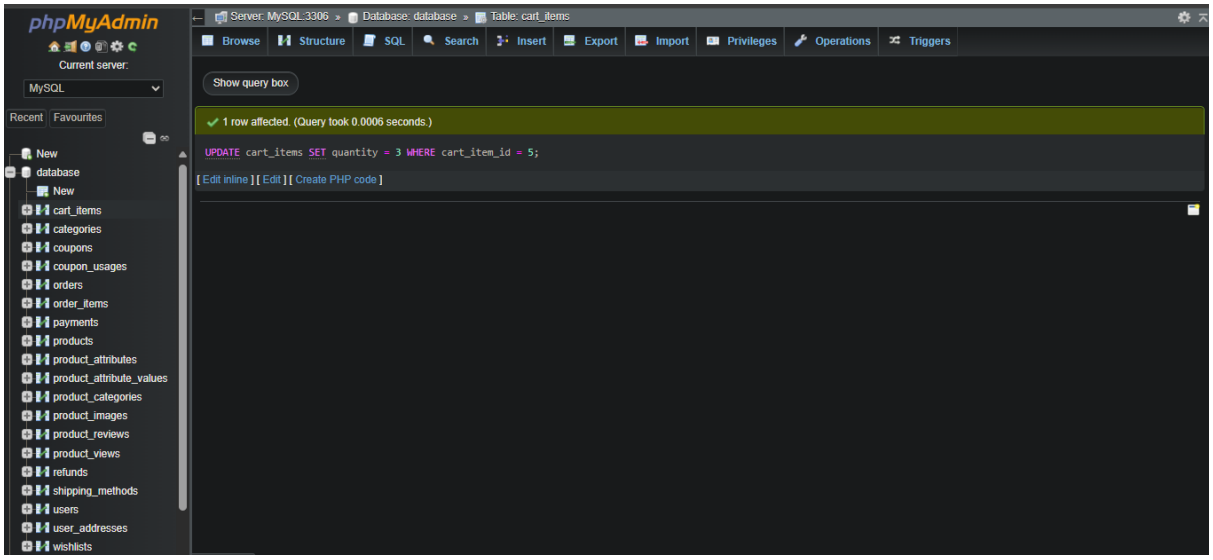
JOIN products p ON ci.product_id = p.product_id

WHERE ci.user_id = 1



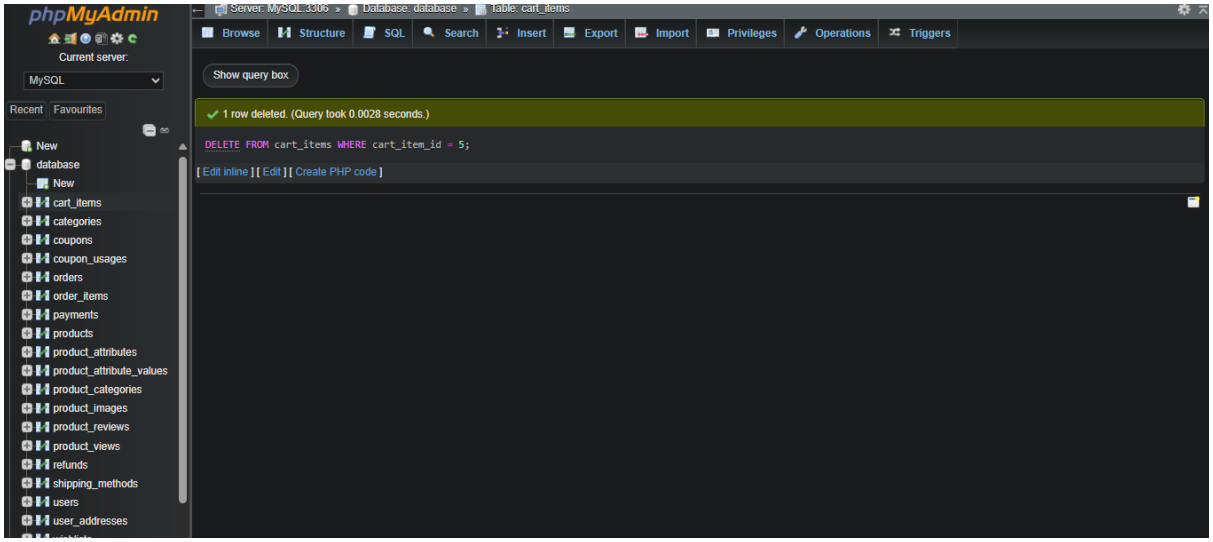
37.Update cart item quantity:

UPDATE cart_items SET quantity = 3 WHERE cart_item_id = 5



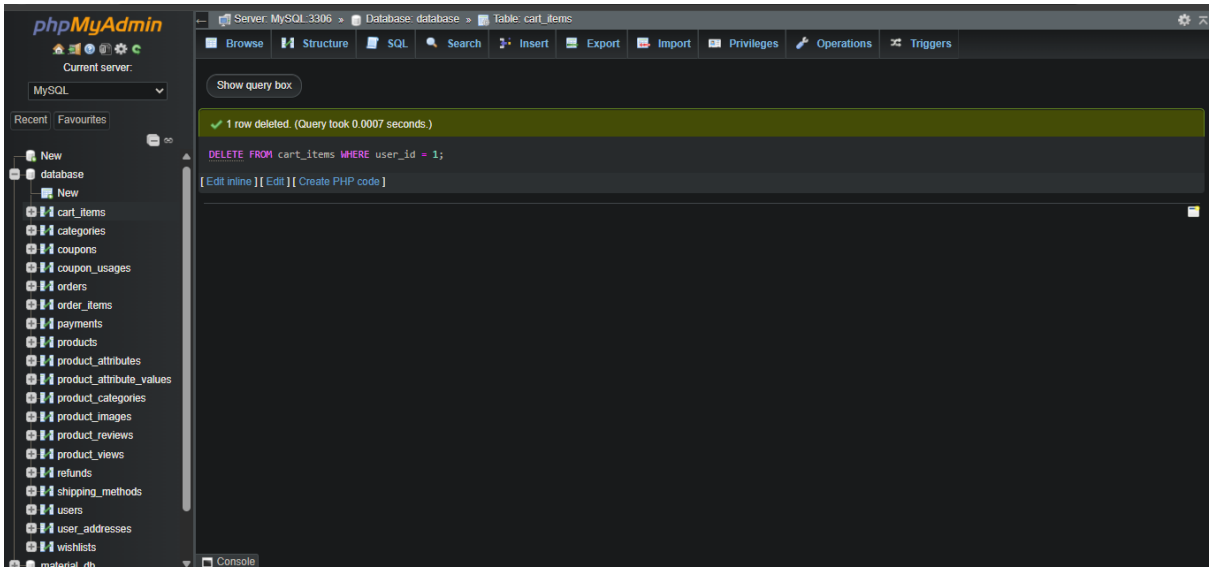
38.Remove item from cart:

DELETE FROM cart_items WHERE cart_item_id = 5;



39.Clear user's cart:

DELETE FROM cart_items WHERE user_id = 1



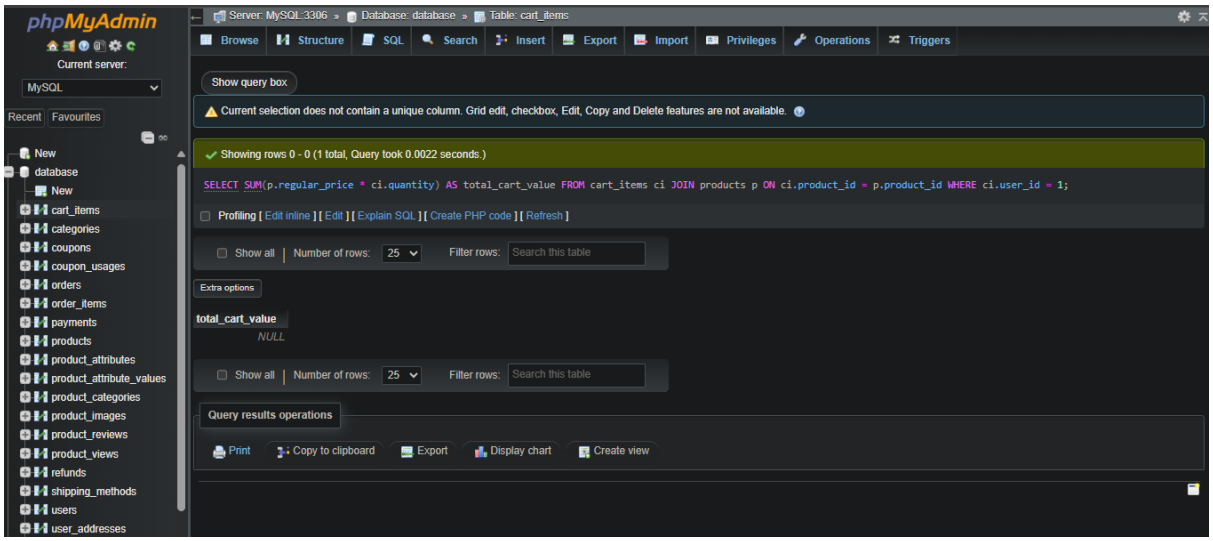
40.Get total cart value for a user:

SELECT SUM(p.regular_price * ci.quantity) AS total_cart_value

FROM cart_items ci

JOIN products p ON ci.product_id = p.product_id

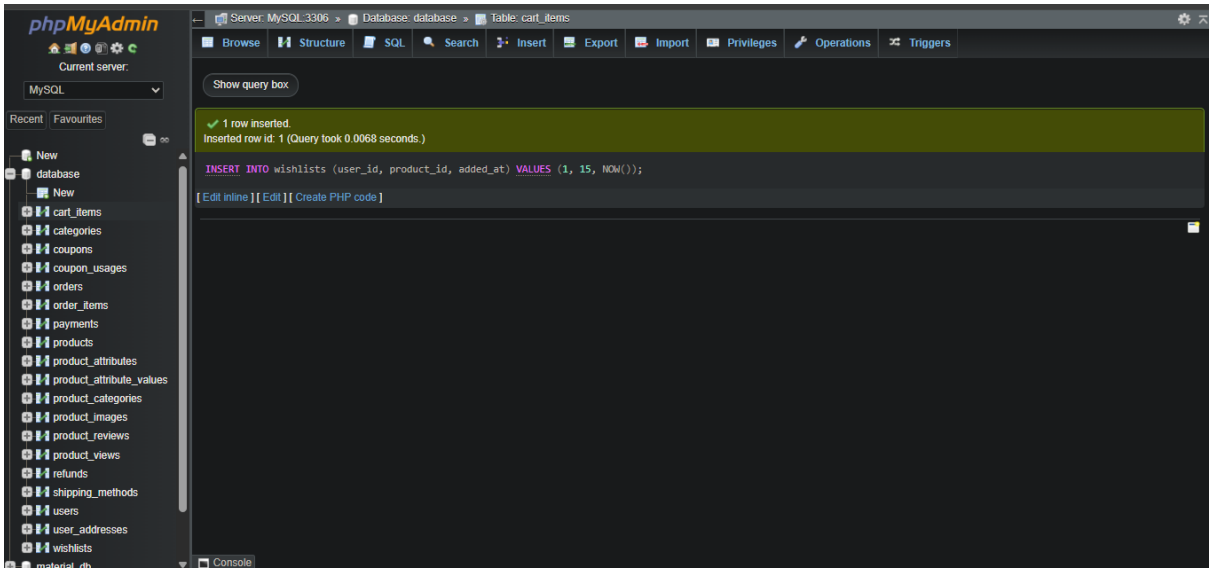
WHERE ci.user_id = 1



41.Add item to wishlist:

INSERT INTO wishlists (user_id, product_id, added_at)

VALUES (1, 15, NOW())



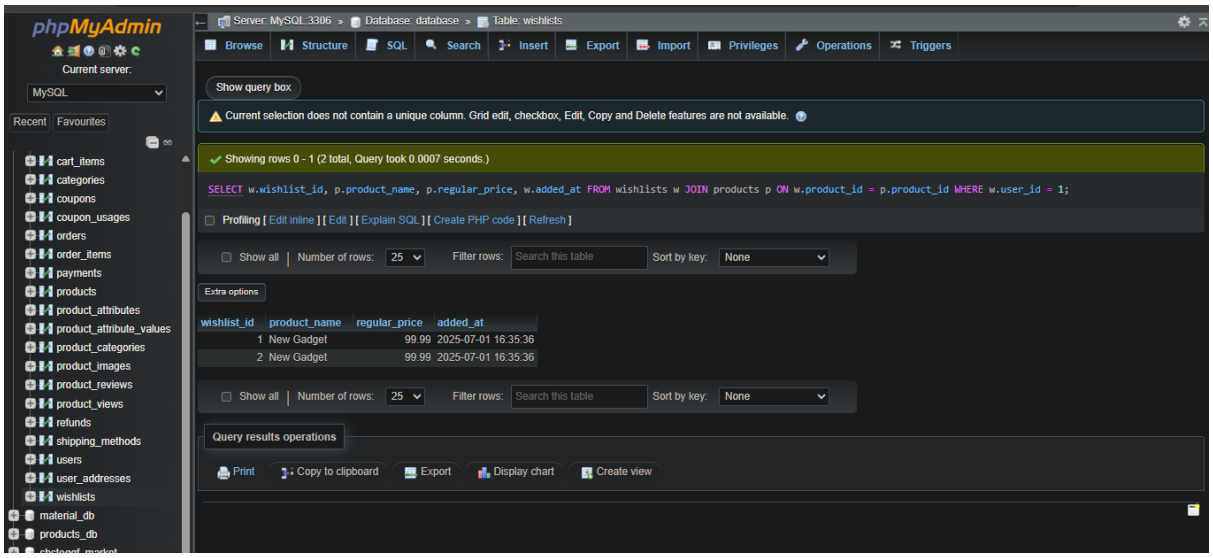
42.Get user's wishlist items:

SELECT w.wishlist_id, p.product_name, p.regular_price, w.added_at

FROM wishlists w

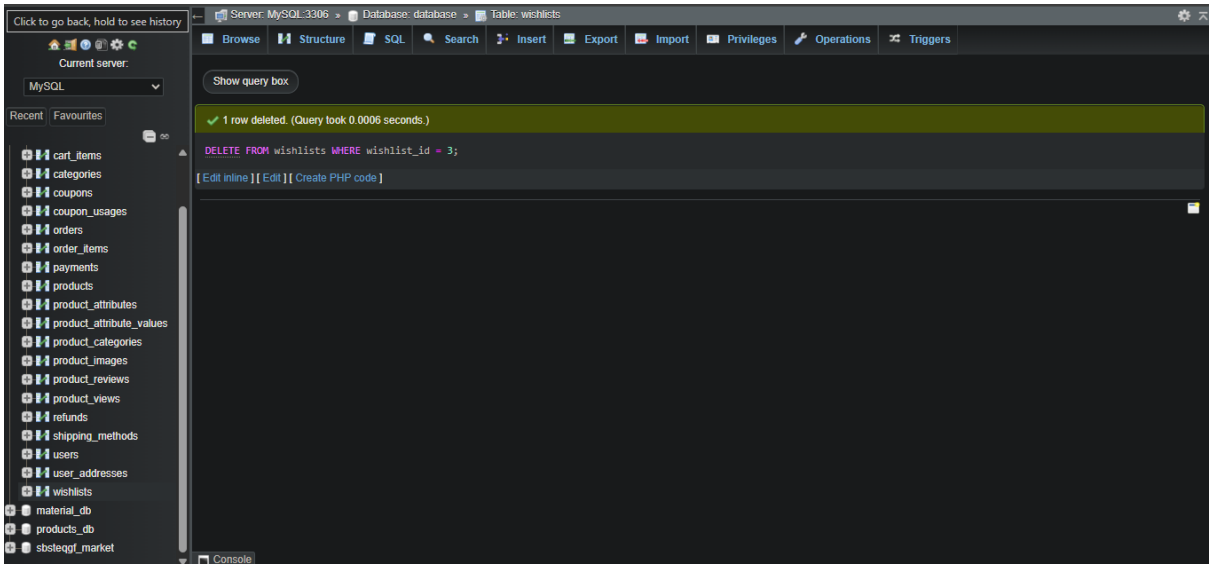
JOIN products p ON w.product_id = p.product_id

WHERE w.user_id = 1



43.Remove item from wishlist:

DELETE FROM wishlists WHERE wishlist_id = 3



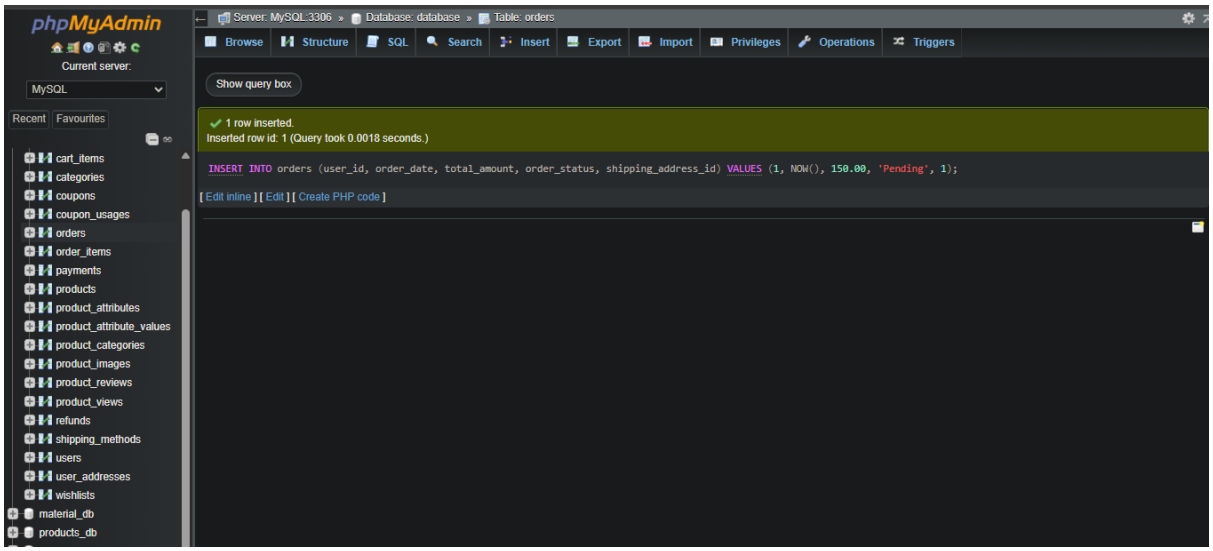
44.Order Processing

Create a new order (after successful payment):

INSERT INTO orders (user_id, order_date, total_amount,

order_status, shipping_address_id)

VALUES (1, NOW(), 150.00, 'Pending', 1)

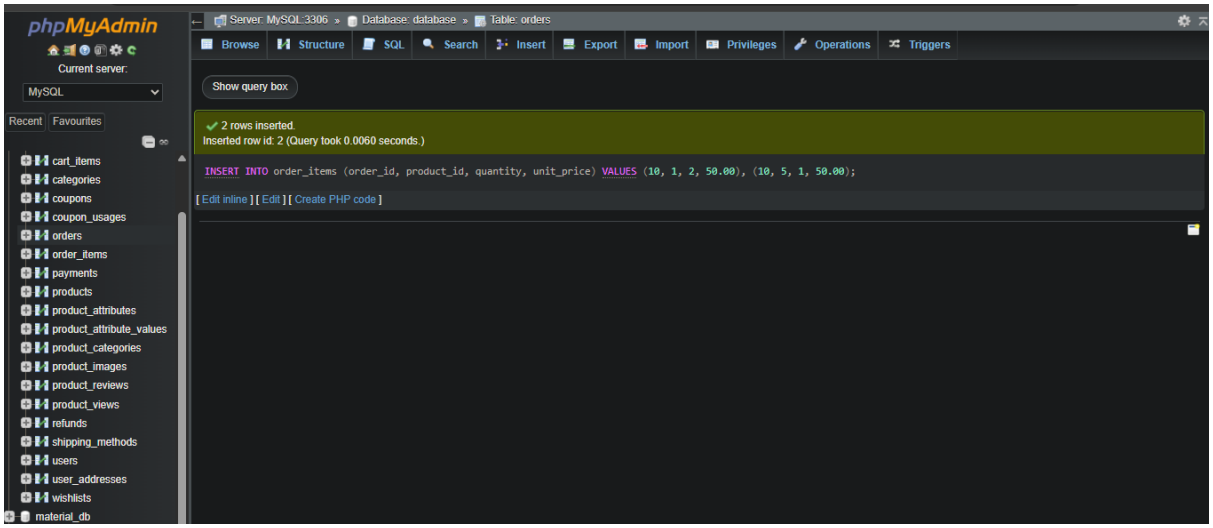


45.Add order items:

INSERT INTO order_items (order_id, product_id, quantity,

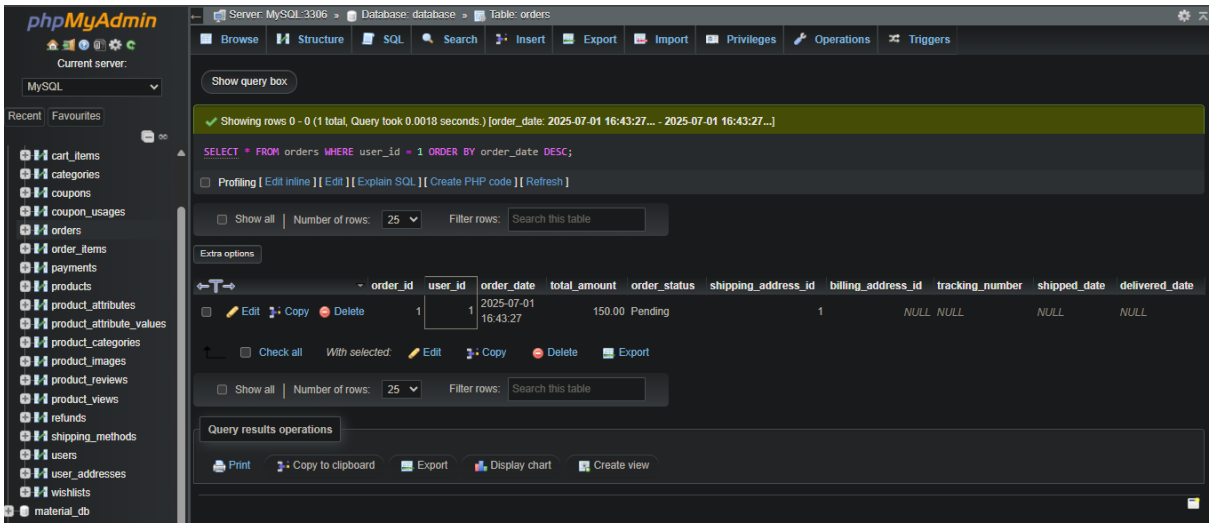
unit_price)

VALUES (10, 1, 2, 50.00), (10, 5, 1, 50.00);



46.Get all orders for a user:

SELECT * FROM orders WHERE user_id = 1 ORDER BY order_date DESC;



47.Get order details by order ID:

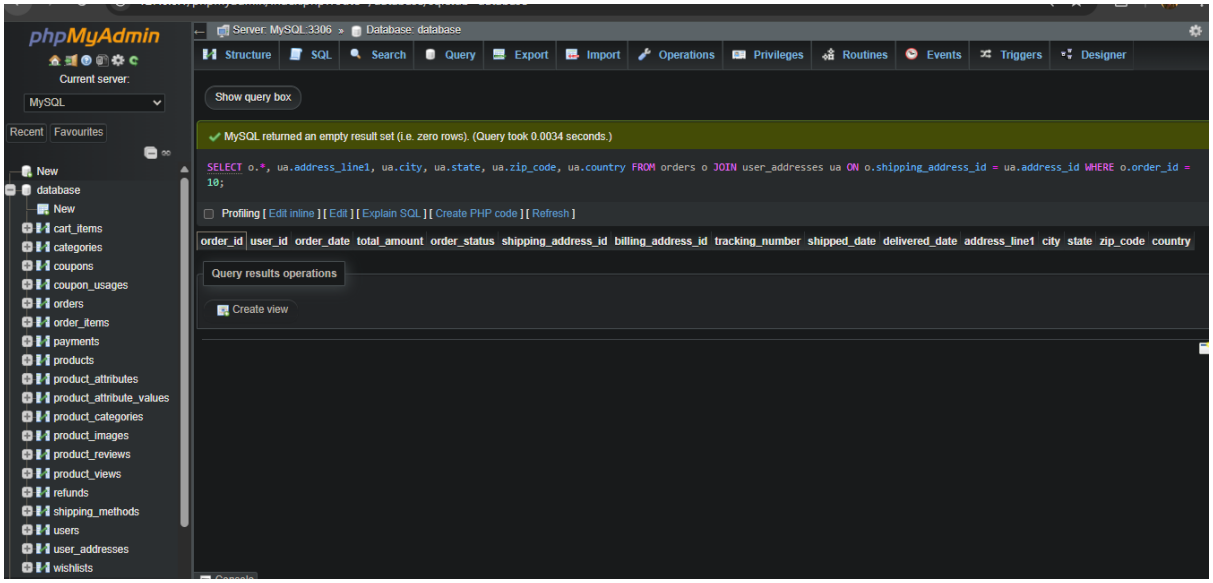
SELECT o.*, ua.address_line1, ua.city, ua.state, ua.zip_code,

ua.country

FROM orders o

JOIN user_addresses ua ON o.shipping_address_id = ua.address_id

WHERE o.order_id = 10;



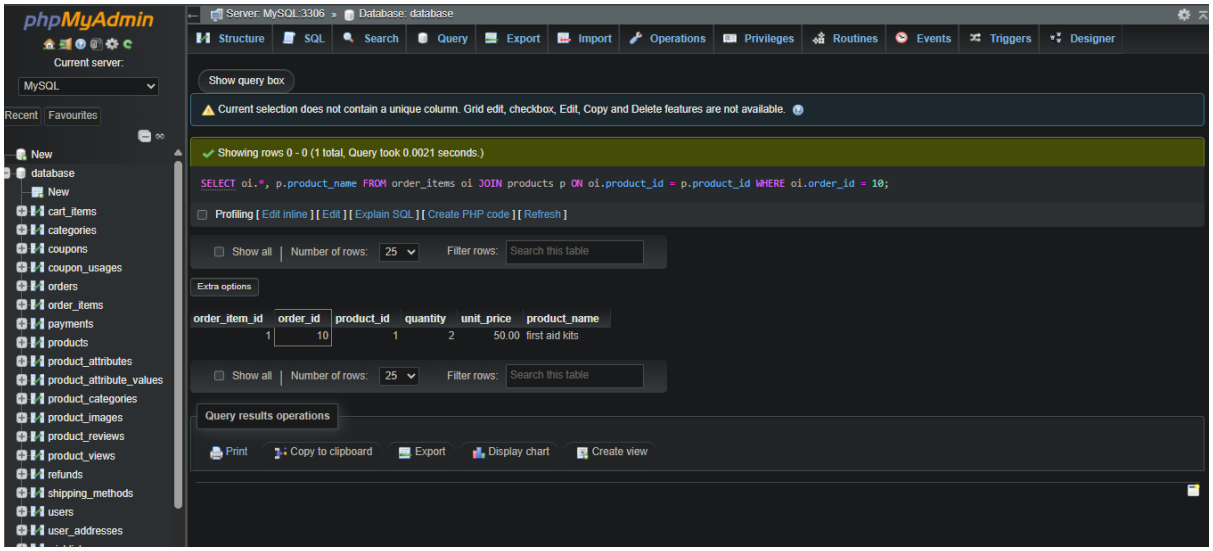
48.Get order items for a specific order:

SELECT oi.*, p.product_name

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

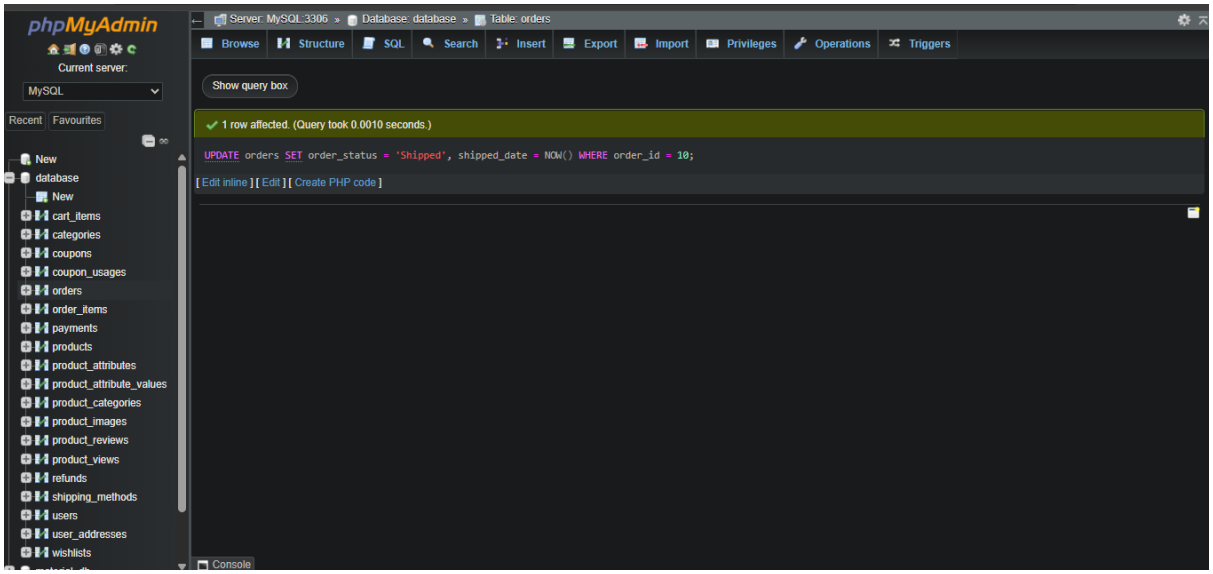
WHERE oi.order_id = 10



49.Update order status:

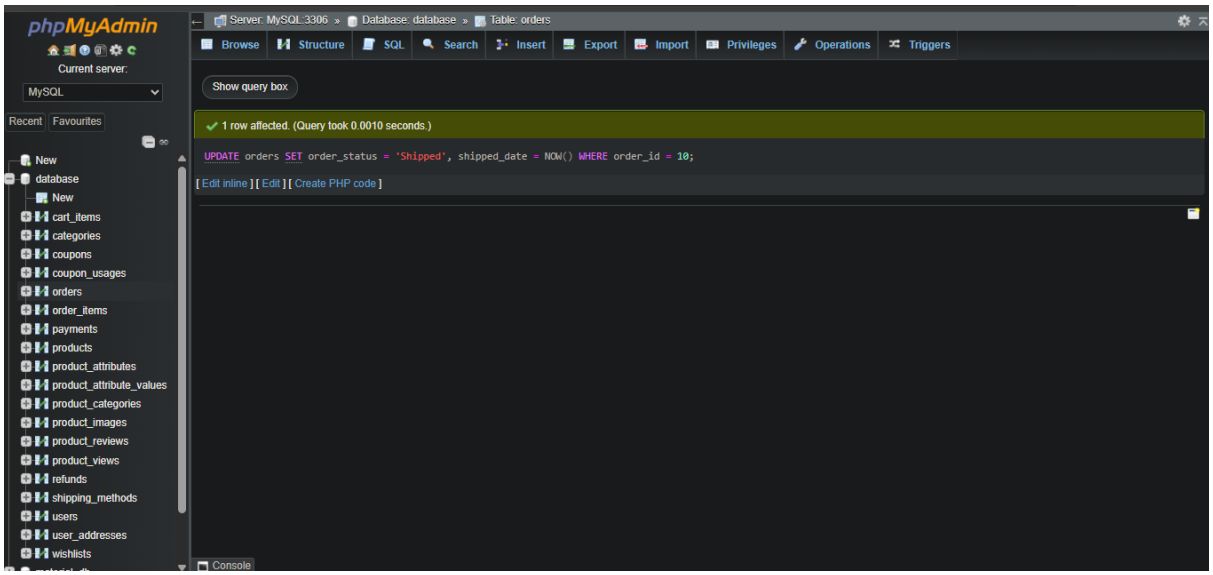
UPDATE orders SET order_status = 'Shipped', shipped_date = NOW()

WHERE order_id = 10;



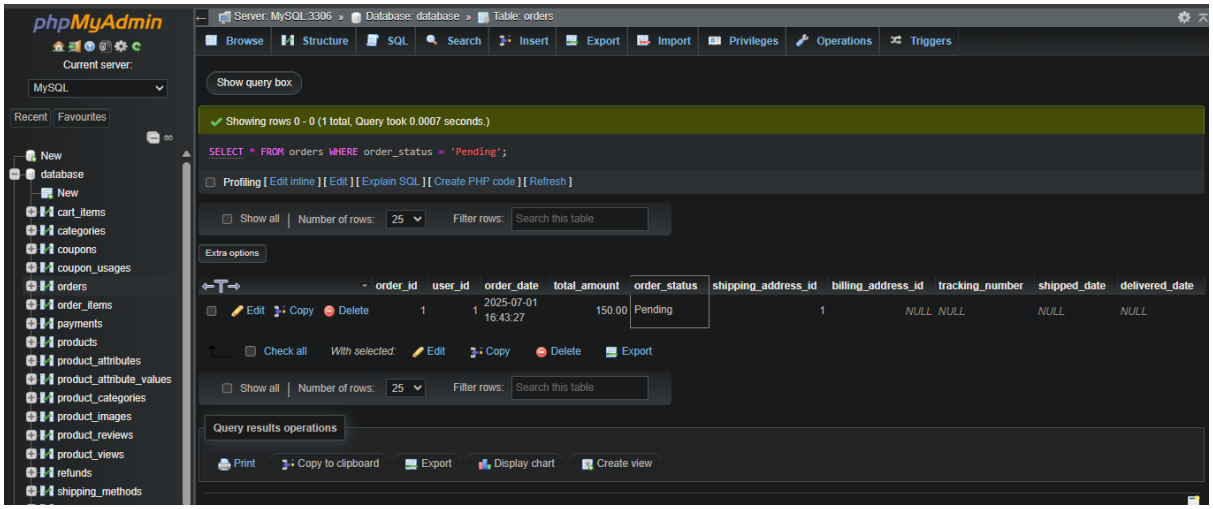
50.Cancel an order:

UPDATE orders SET order_status = 'Cancelled' WHERE order_id = 10



51.Get orders by status (e.g., 'Pending'):

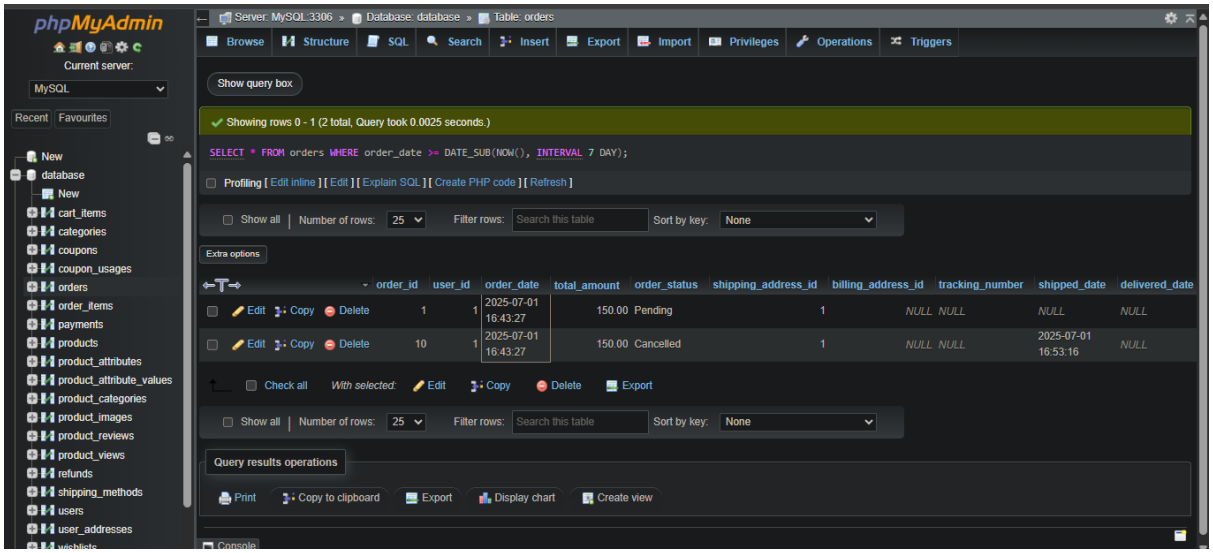
SELECT * FROM orders WHERE order_status = 'Pending'



52.Get recent orders (last 7 days):

SELECT * FROM orders WHERE order_date >= DATE_SUB(NOW(), INTERVAL

7 DAY);

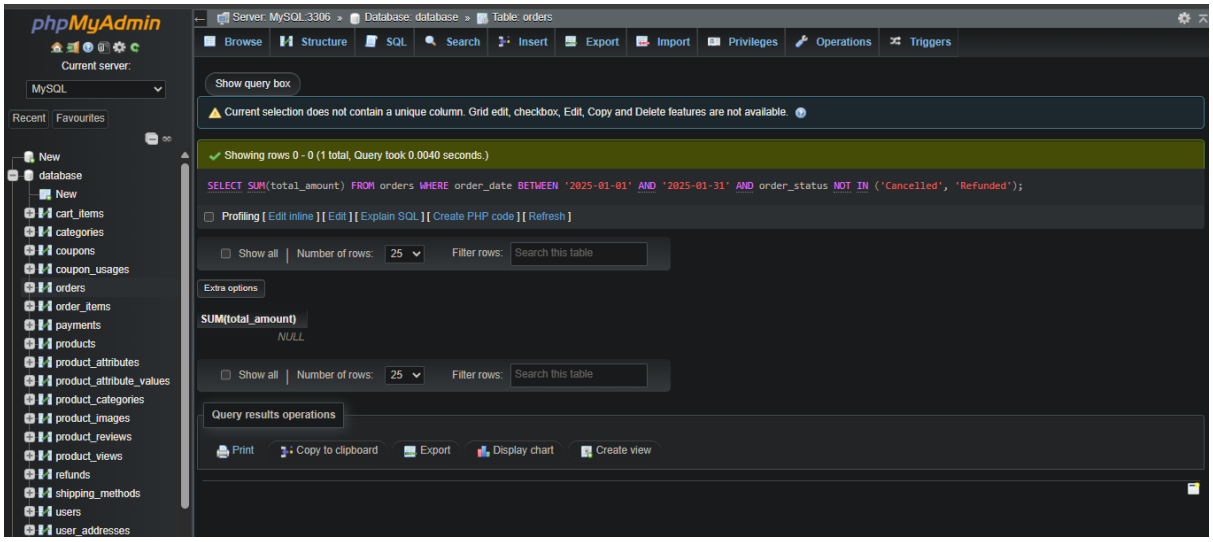


53.Get total sales for a specific period:

SELECT SUM(total_amount) FROM orders WHERE order_date BETWEEN

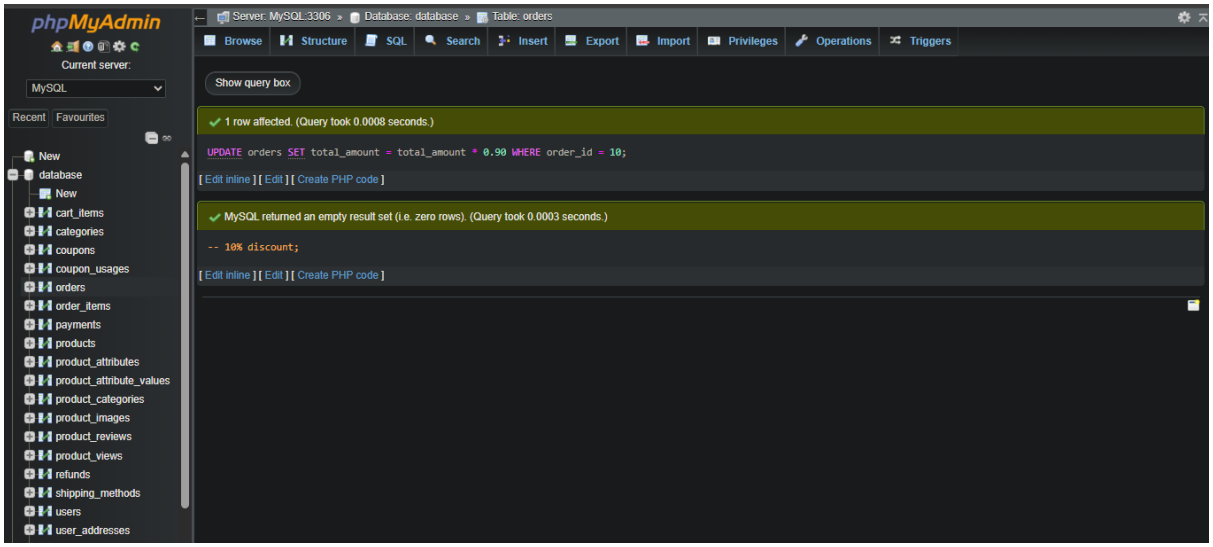
'2025-01-01' AND '2025-01-31' AND order_status NOT IN

('Cancelled', 'Refunded')



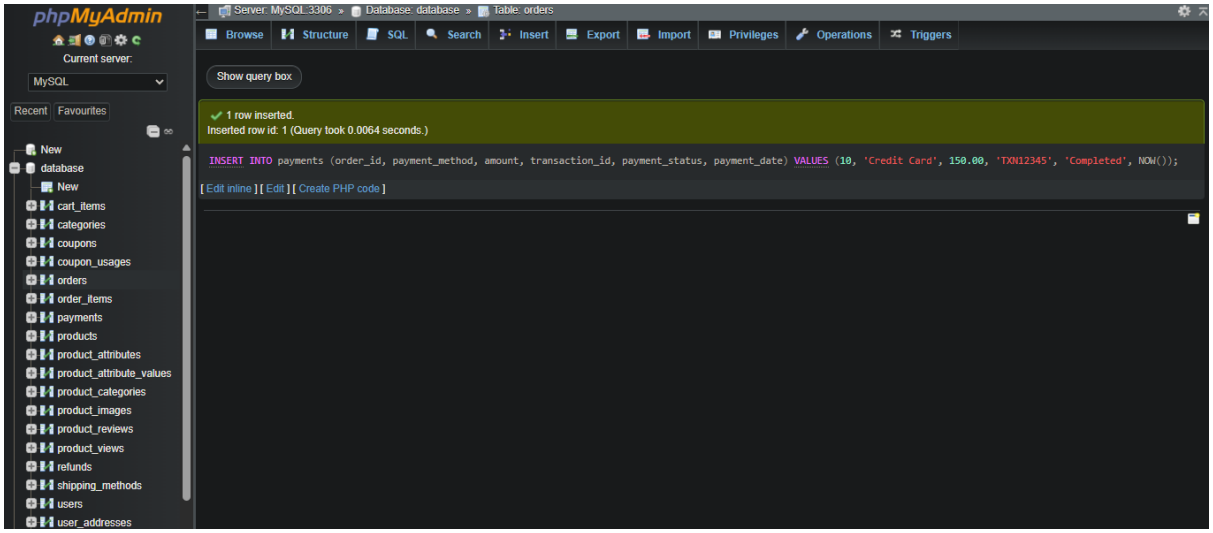
54.Apply a discount to an order (e.g., by updating total_amount or adding a discount record):

```
UPDATE orders SET total_amount = total_amount * 0.90 WHERE
order_id = 10; -- 10% discount
```



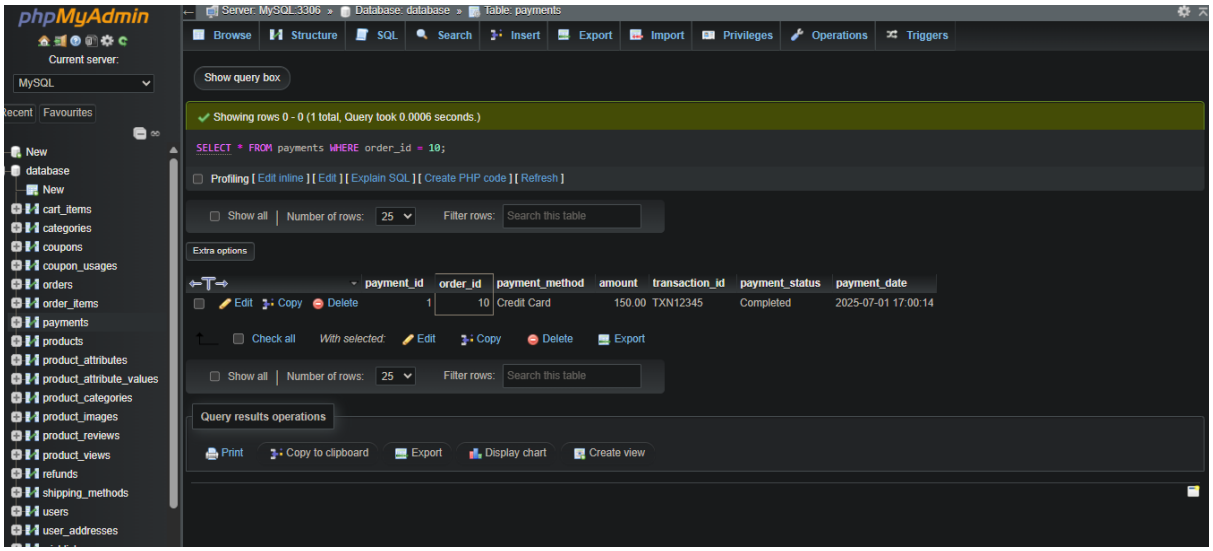
55.Record payment for an order:

```
INSERT INTO payments (order_id, payment_method, amount,
transaction_id, payment_status, payment_date)
VALUES (10, 'Credit Card', 150.00, 'TXN12345', 'Completed',
NOW());
```



56.Get payment details for an order:

SELECT * FROM payments WHERE order_id = 10

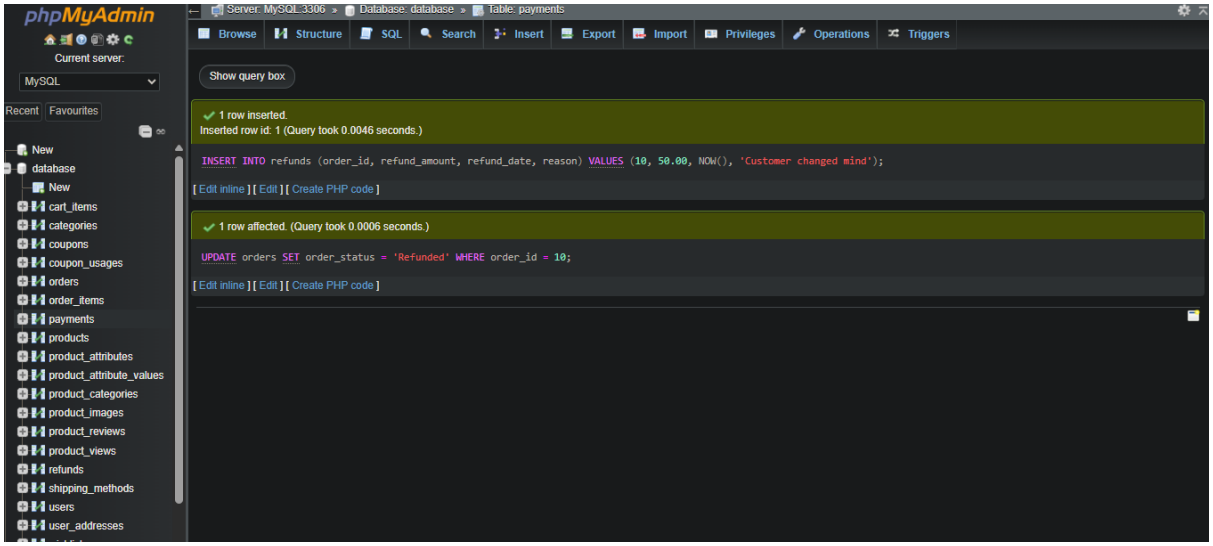


57.Process a refund:

INSERT INTO refunds (order_id, refund_amount, refund_date, reason)

VALUES (10, 50.00, NOW(), 'Customer changed mind');

UPDATE orders SET order_status = 'Refunded' WHERE order_id = 10



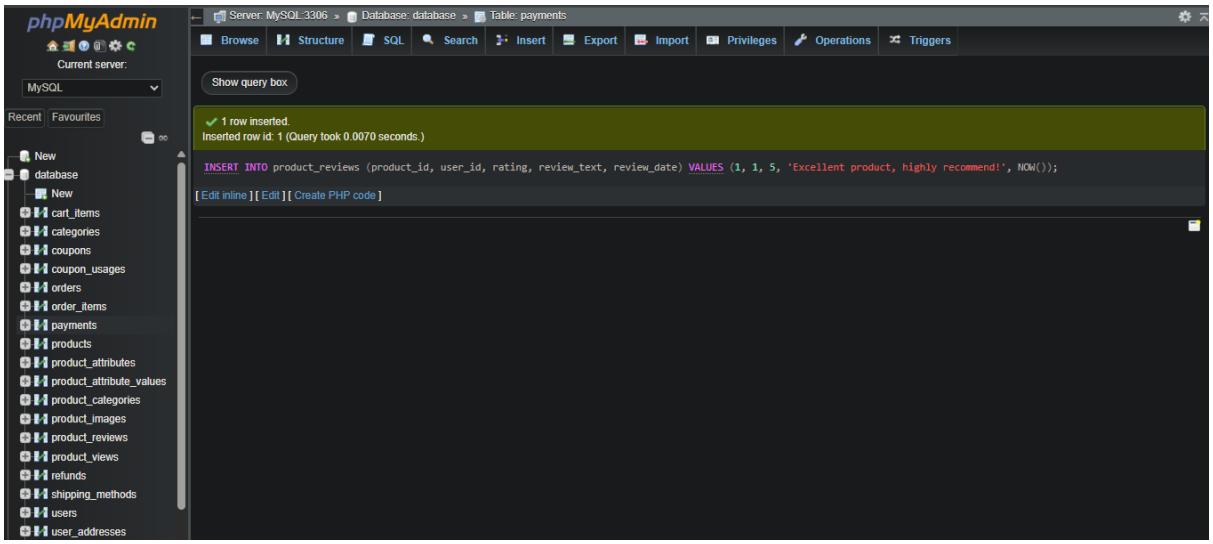
58.Reviews & Ratings

Submit a product review:

INSERT INTO product_reviews (product_id, user_id, rating,

review_text, review_date)

VALUES (1, 1, 5, 'Excellent product, highly recommend!', NOW())



59.Get all reviews for a product:

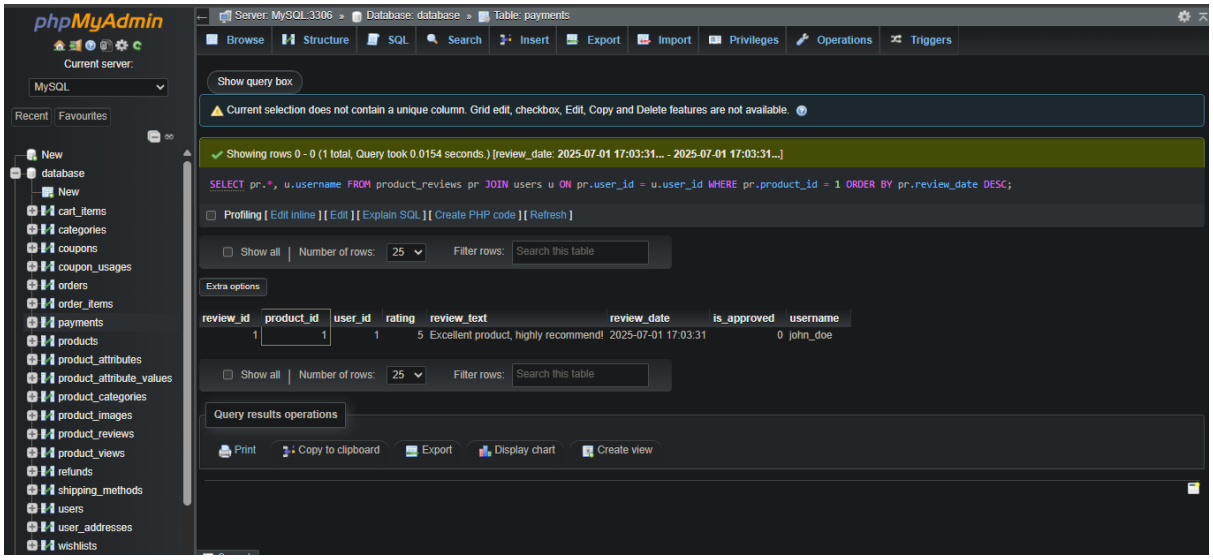
SELECT pr.*, u.username

FROM product_reviews pr

JOIN users u ON pr.user_id = u.user_id

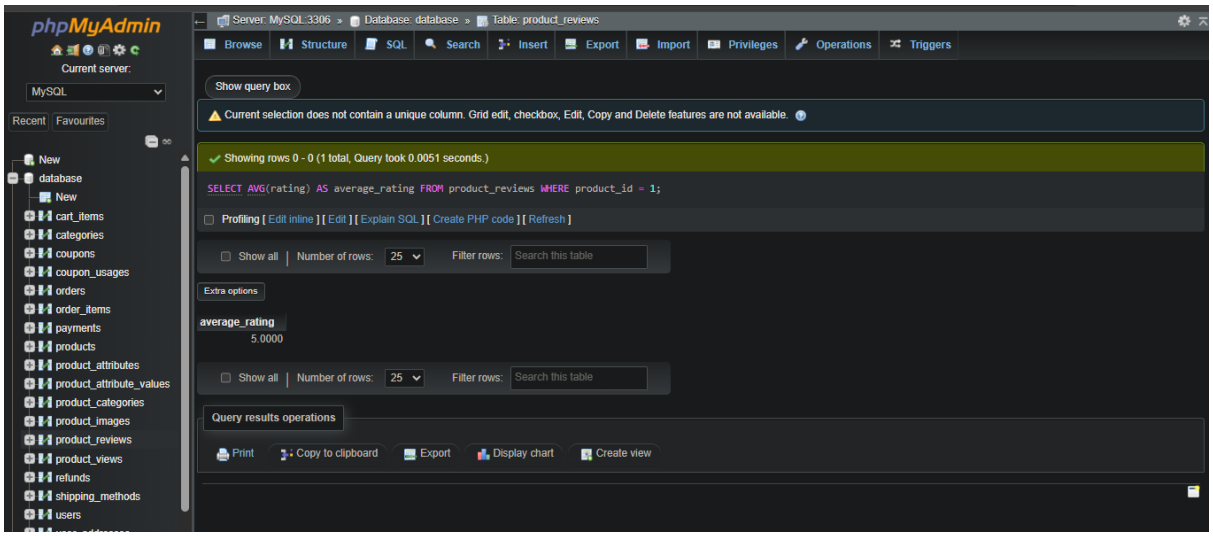
WHERE pr.product_id = 1

ORDER BY pr.review_date DESC



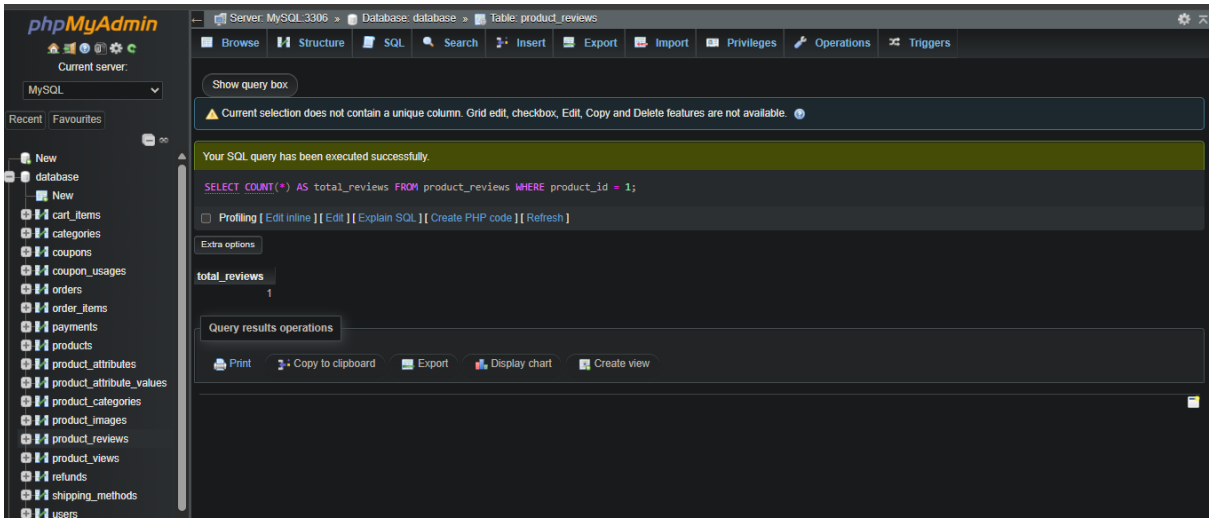
60.Get average rating for a product:

SELECT AVG(rating) AS average_rating FROM product_reviews WHERE product_id = 1;



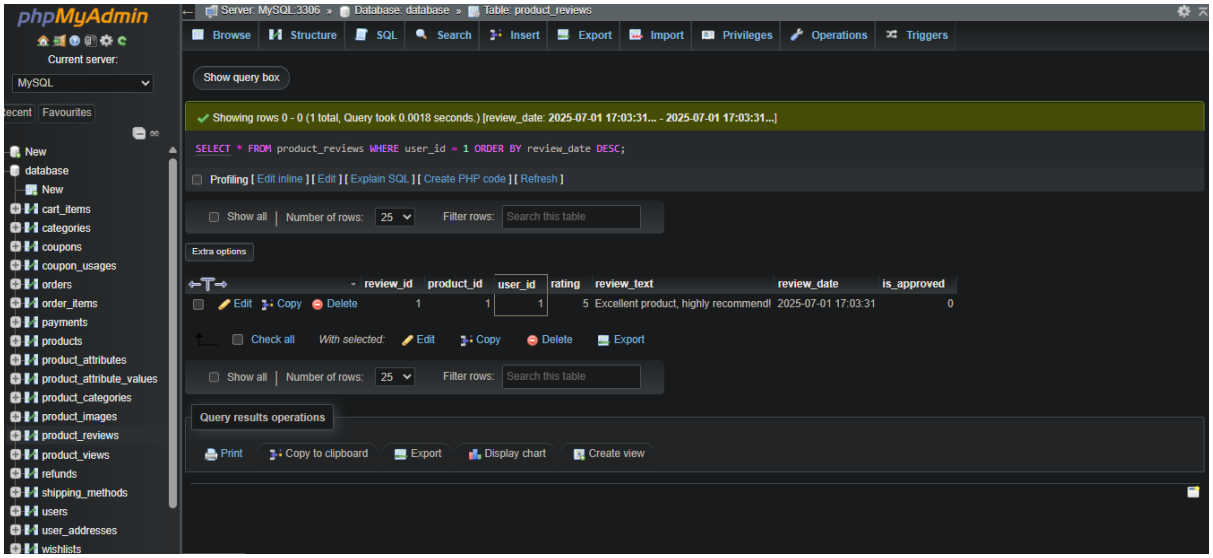
61.Get total number of reviews for a product:

SELECT COUNT(*) AS total_reviews FROM product_reviews WHERE product_id = 1;



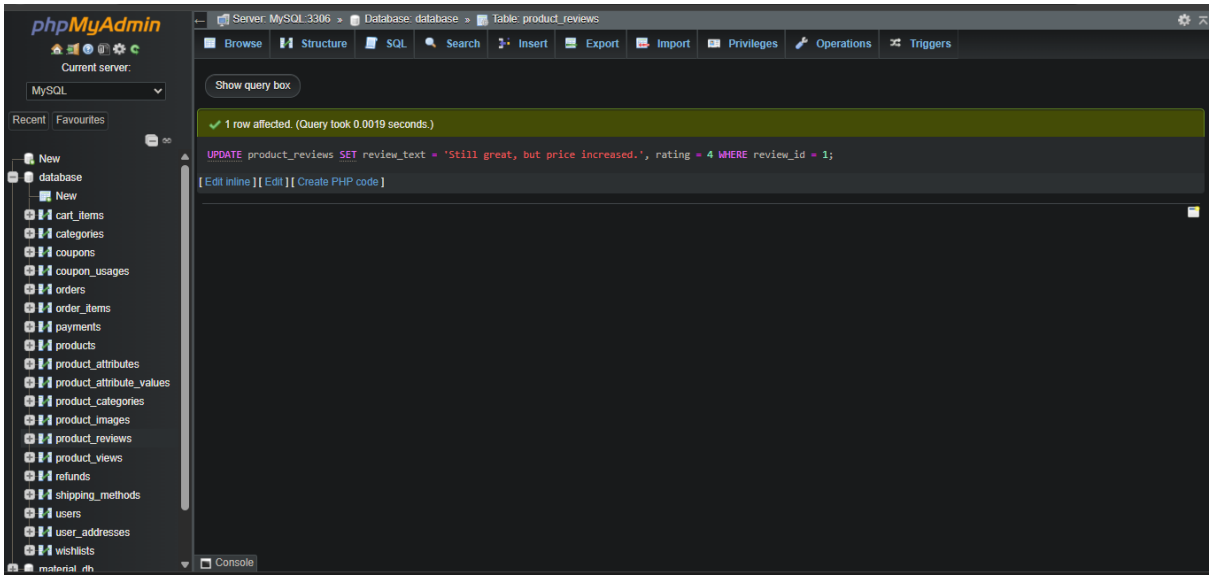
62.Get reviews by a specific user:

SELECT * FROM product_reviews WHERE user_id = 1 ORDER BY review_date DESC



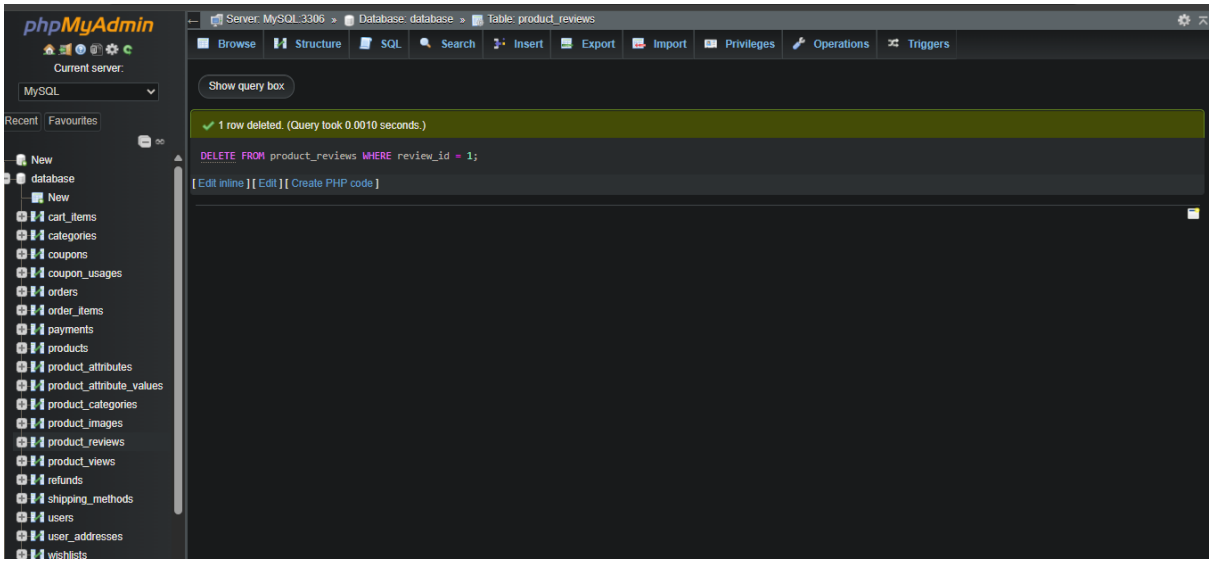
63.Update a product review:

UPDATE product_reviews SET review_text = 'Still great, but price increased.', rating = 4 WHERE review_id = 1;



64.Delete a product review (admin/user):

DELETE FROM product_reviews WHERE review_id = 1



65.Discounts & Coupons

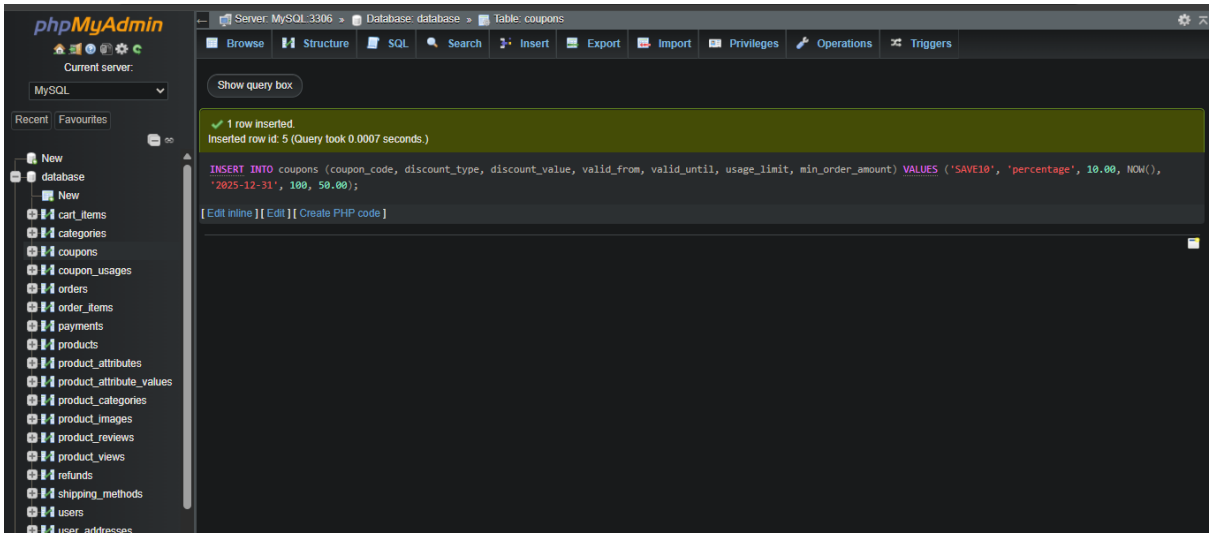
Create a new coupon:

INSERT INTO coupons (coupon_code, discount_type, discount_value,

valid_from, valid_until, usage_limit, min_order_amount)

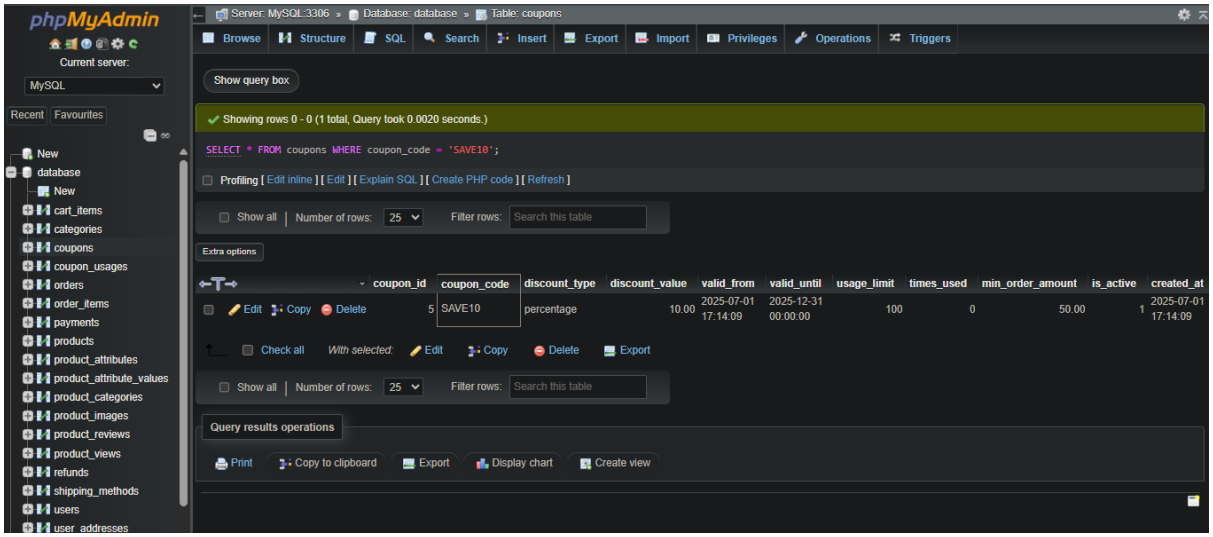
VALUES ('SAVE10', 'percentage', 10.00, NOW(), '2025-12-31', 100,

50.00);



66.Get coupon details by code:

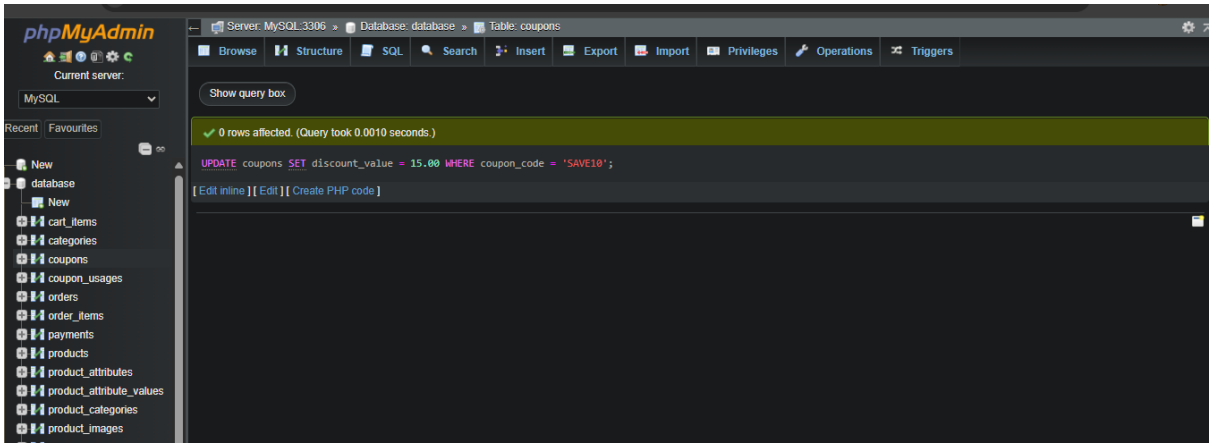
SELECT * FROM coupons WHERE coupon_code = 'SAVE10'



67.Update coupon details:

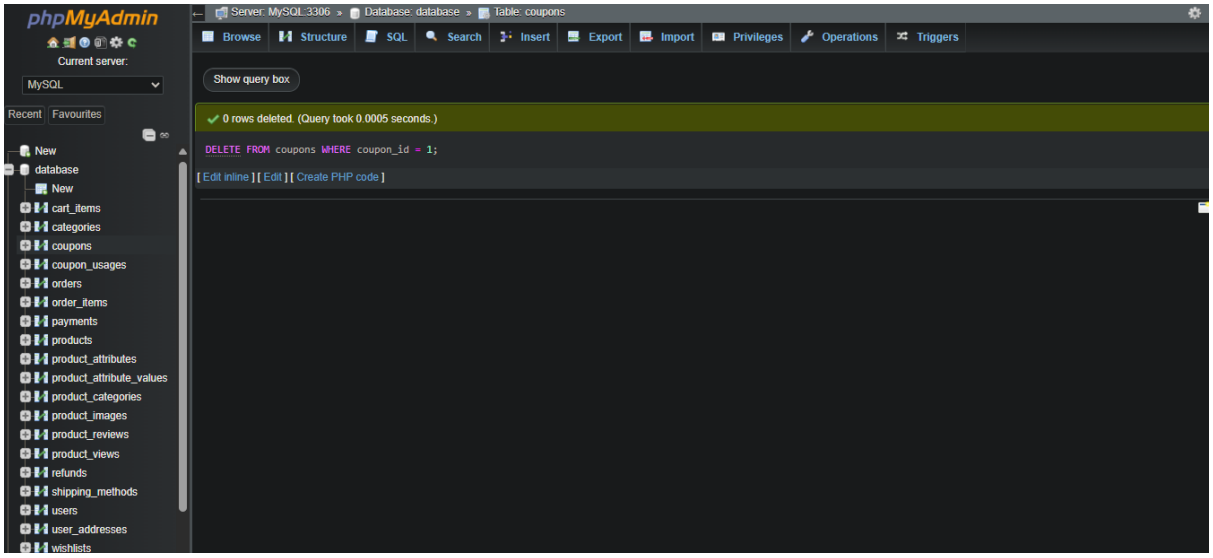
UPDATE coupons SET discount_value = 15.00 WHERE coupon_code =

'SAVE10'



68.delete a coupon:

DELETE FROM coupons WHERE coupon_id = 1



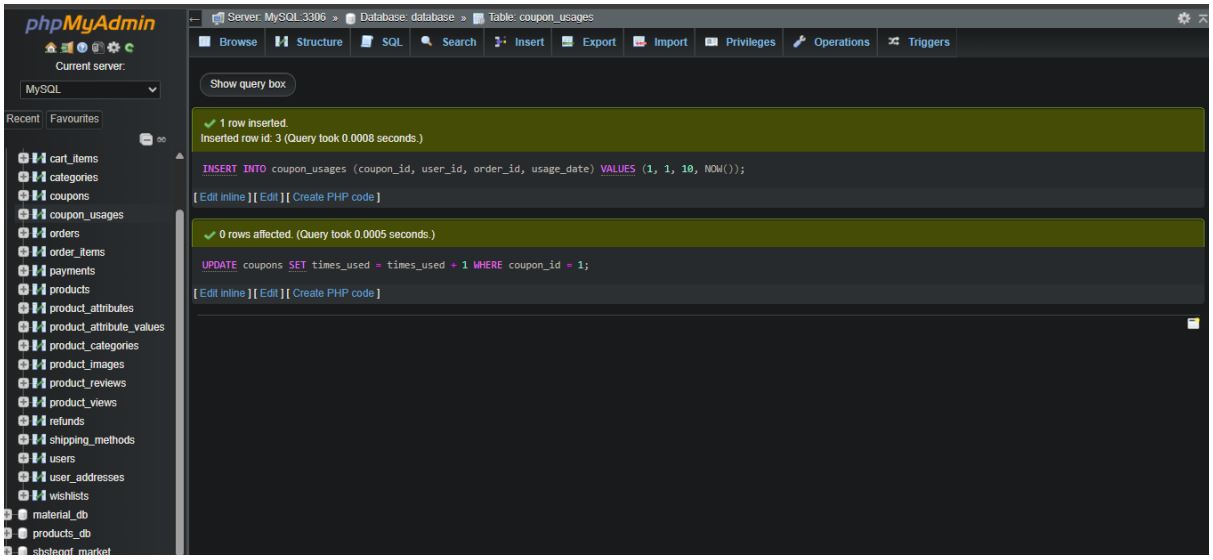
69.Record coupon usage:

INSERT INTO coupon_usages (coupon_id, user_id, order_id,

usage_date)

VALUES (1, 1, 10, NOW());

UPDATE coupons SET times_used = times_used + 1 WHERE coupon_id = 1



70.Check if a coupon is valid (date, usage limit, min order amount):

SELECT * FROM coupons

WHERE coupon_code = 'SAVE10'

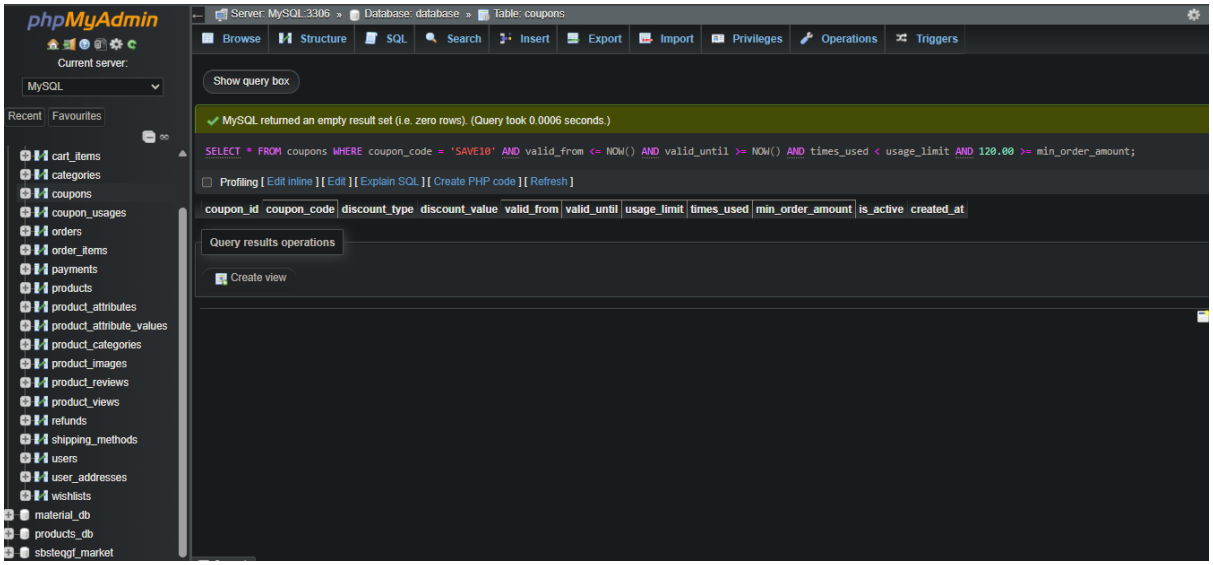
AND valid_from <= NOW()

AND valid_until >= NOW()

AND times_used < usage_limit

AND 120.00 >= min_order_amount; -- Assuming 120.00 is the current

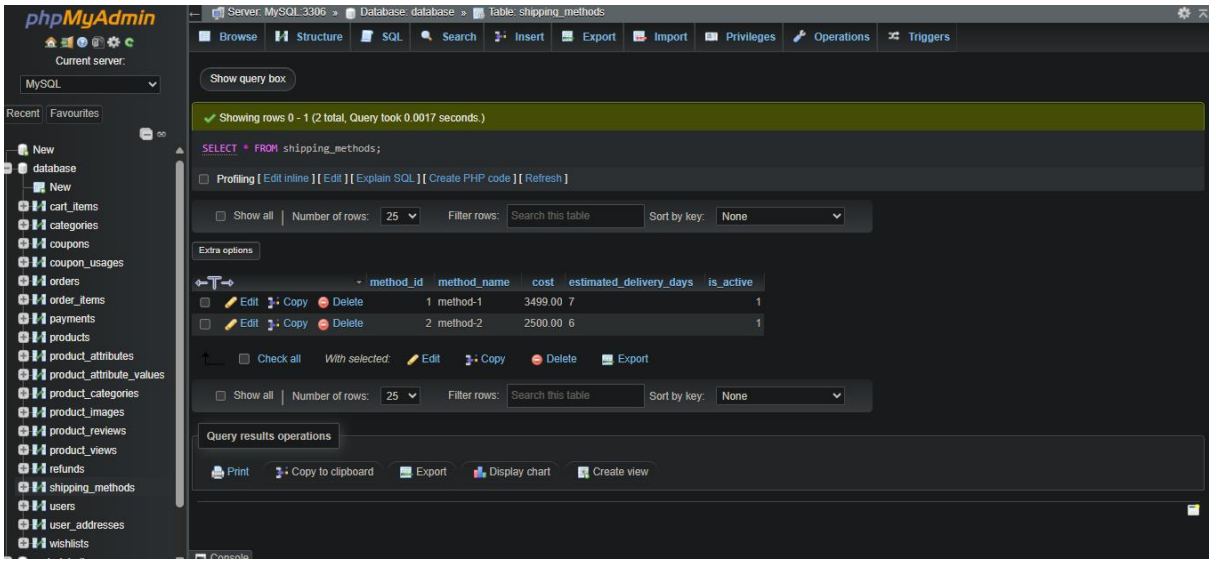
order total



71.Shipping & Inventory

Get all shipping methods:

SELECT * FROM shipping_methods

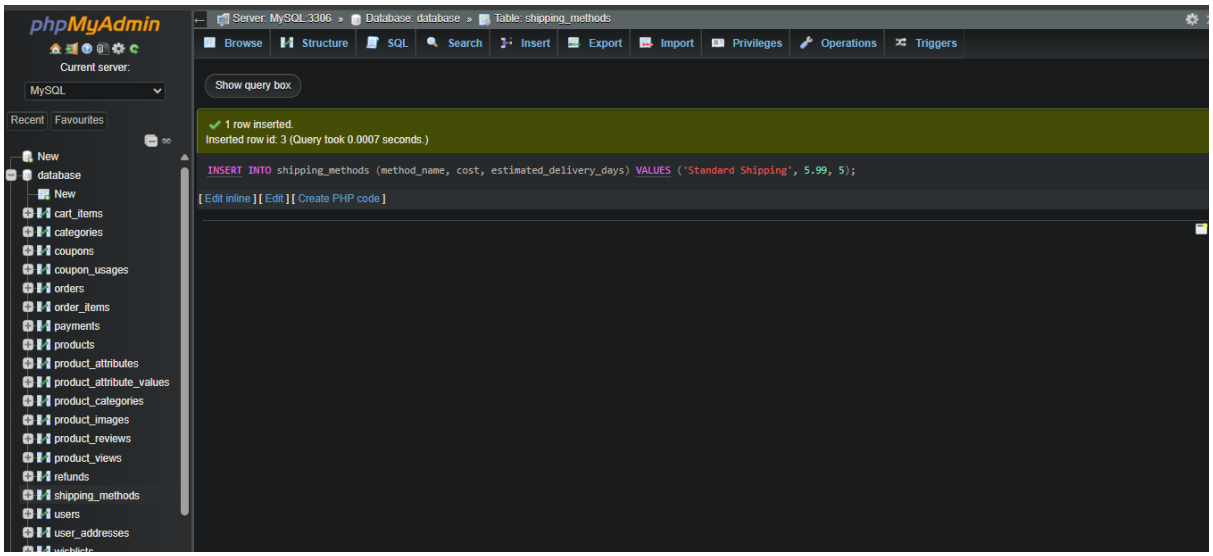


72.Add a new shipping method:

INSERT INTO shipping_methods (method_name, cost,

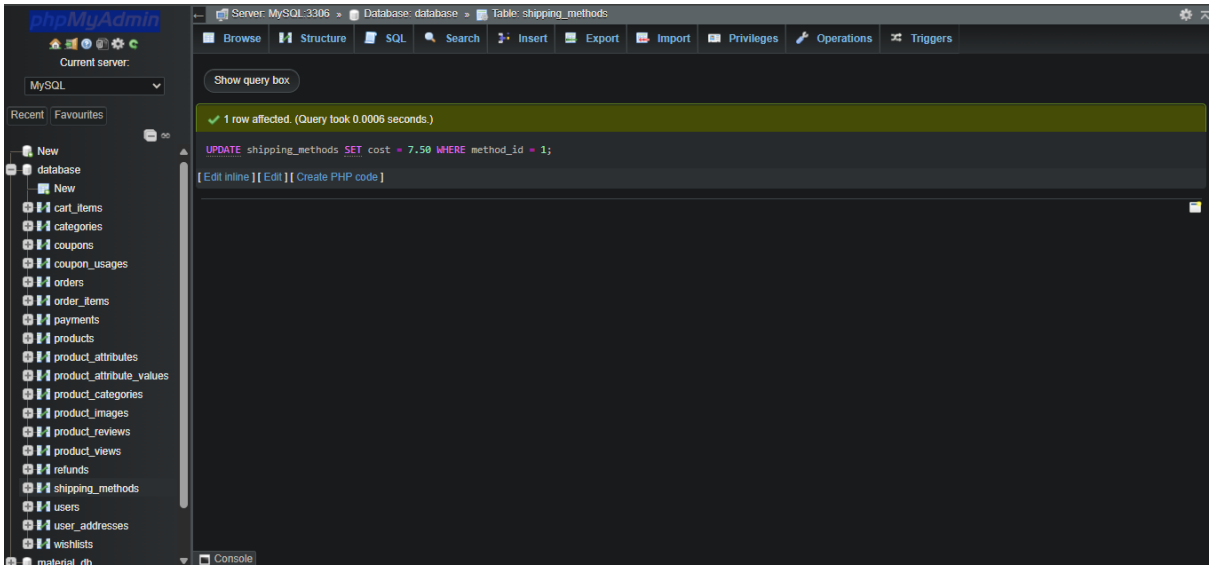
estimated_delivery_days)

VALUES ('Standard Shipping', 5.99, 5);



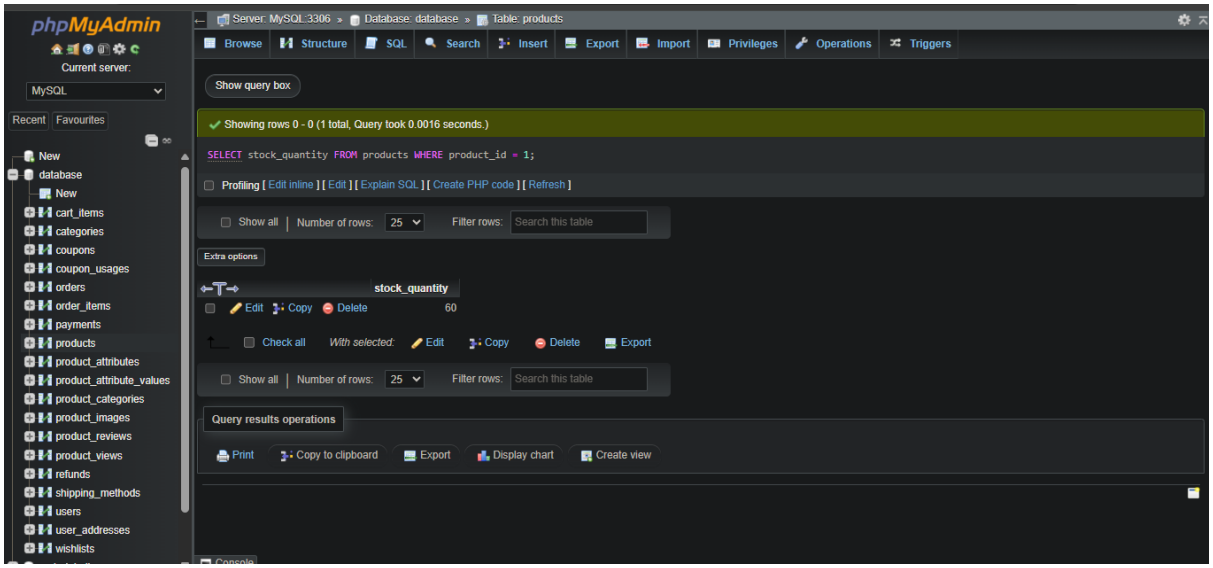
73.Update shipping method cost:

UPDATE shipping_methods SET cost = 7.50 WHERE method_id = 1



74.Get product stock quantity:

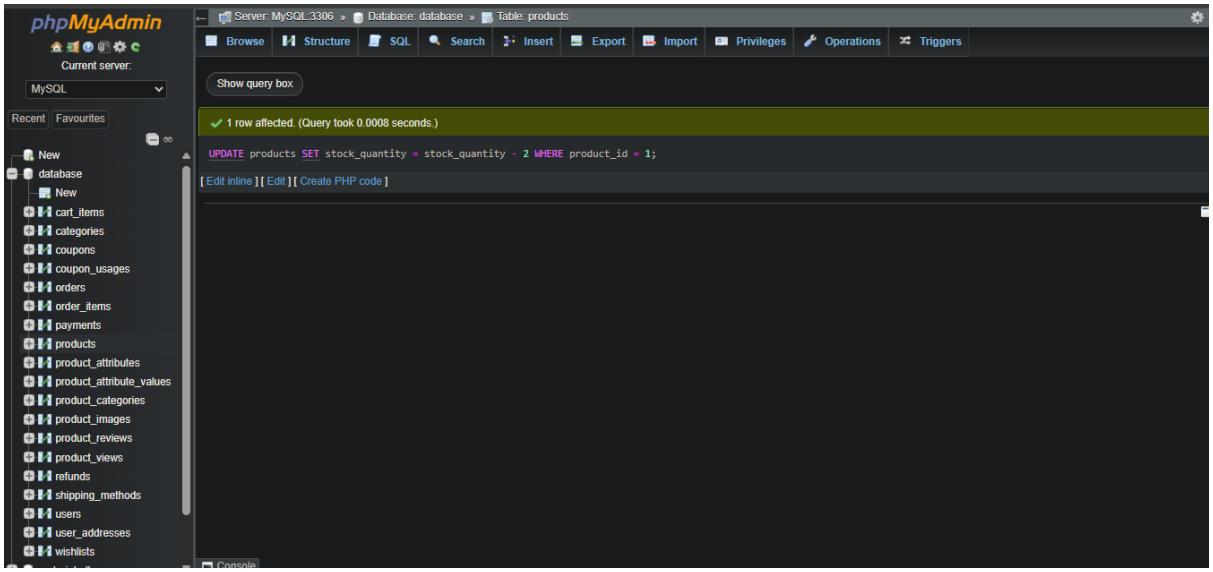
SELECT stock_quantity FROM products WHERE product_id = 1



75.Decrease product stock after an order:

UPDATE products SET stock_quantity = stock_quantity - 2 WHERE

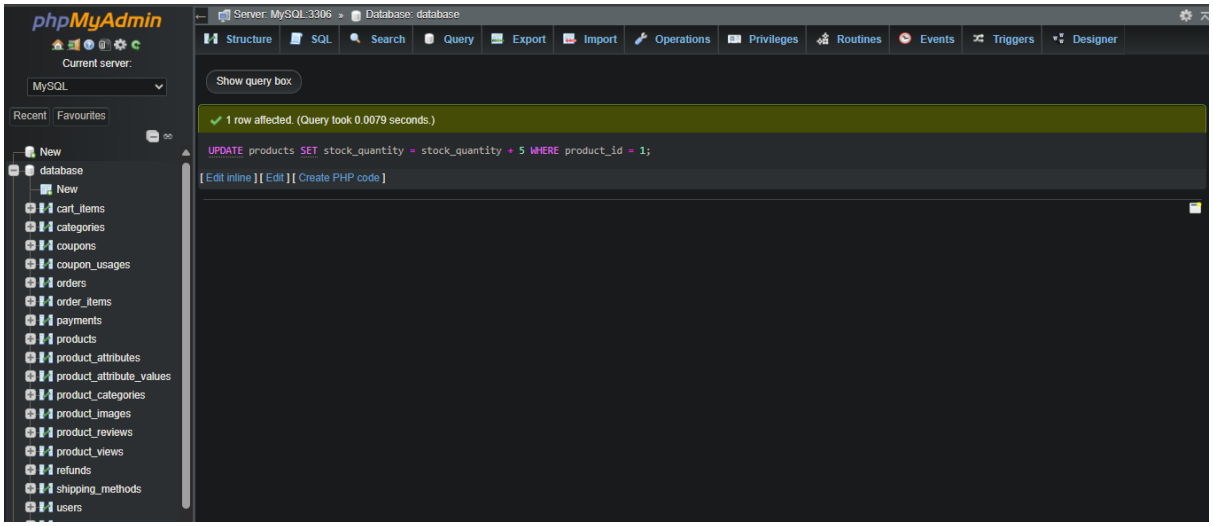
product_id = 1



76.Increase product stock (e.g., for returns or new inventory):

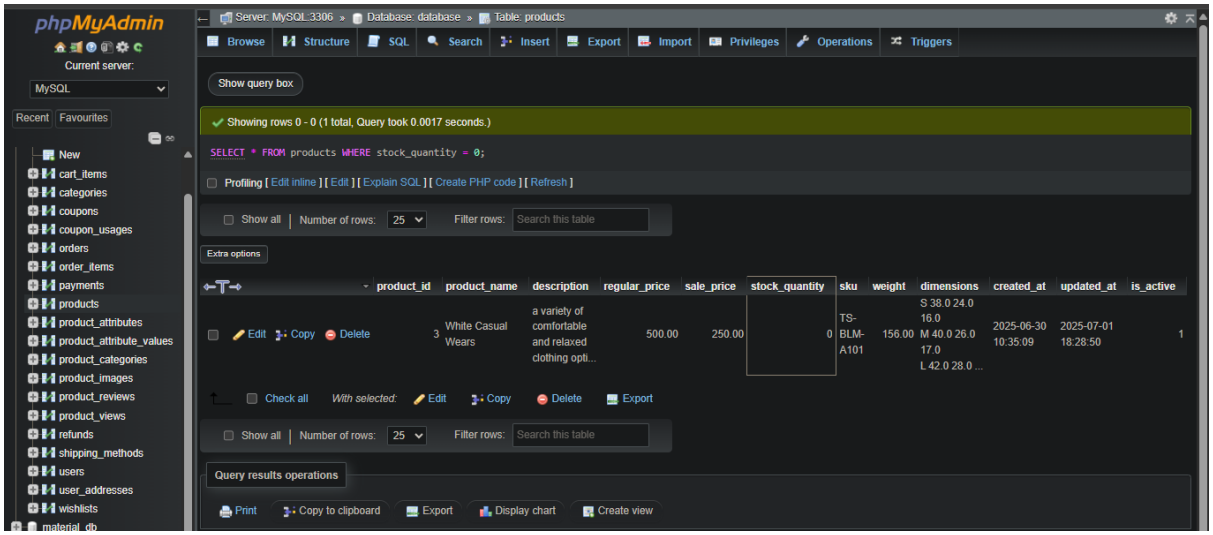
UPDATE products SET stock_quantity = stock_quantity + 5 WHERE

product_id = 1



77.Get products that are out of stock:

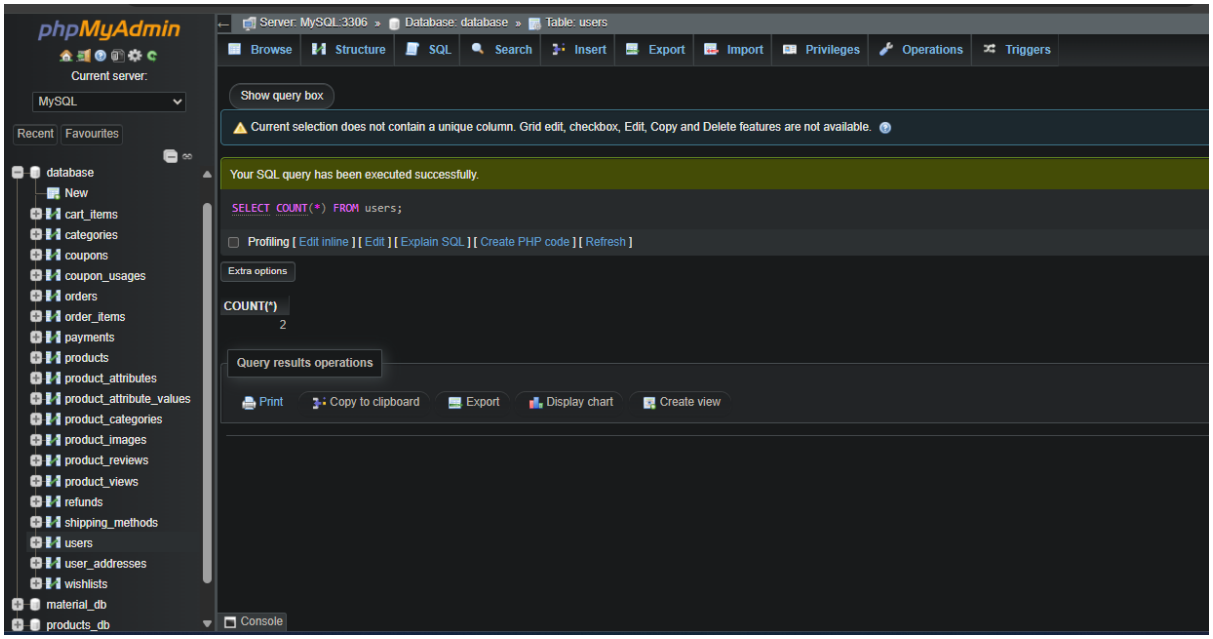
SELECT * FROM products WHERE stock_quantity = 0



78.Admin & Reporting

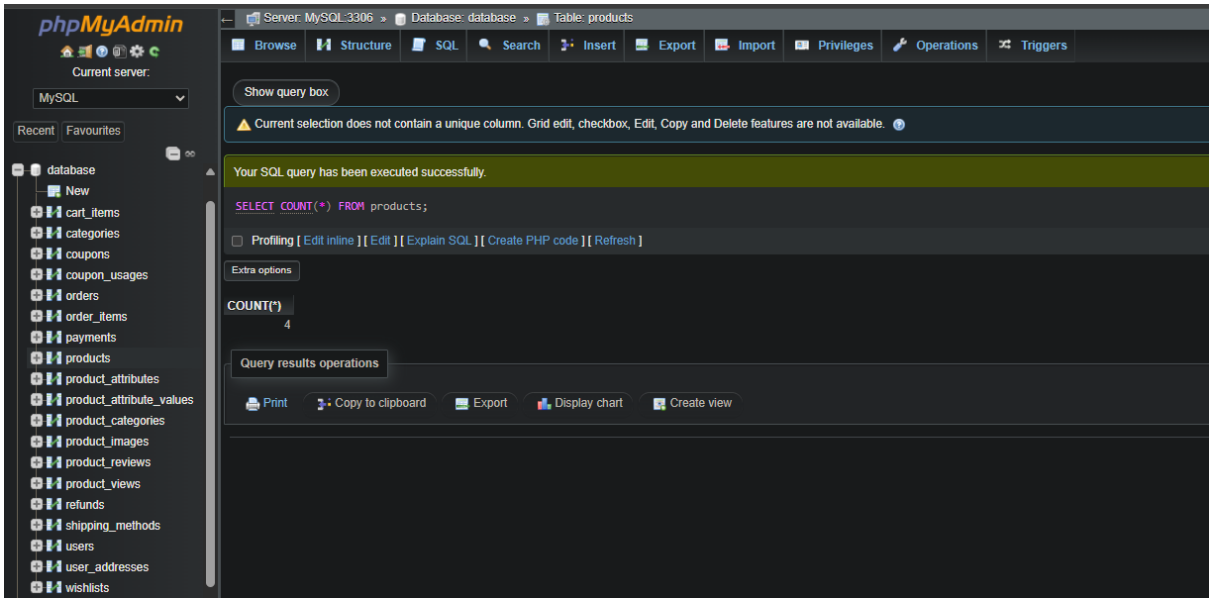
Get total number of users:

SELECT COUNT(*) FROM users



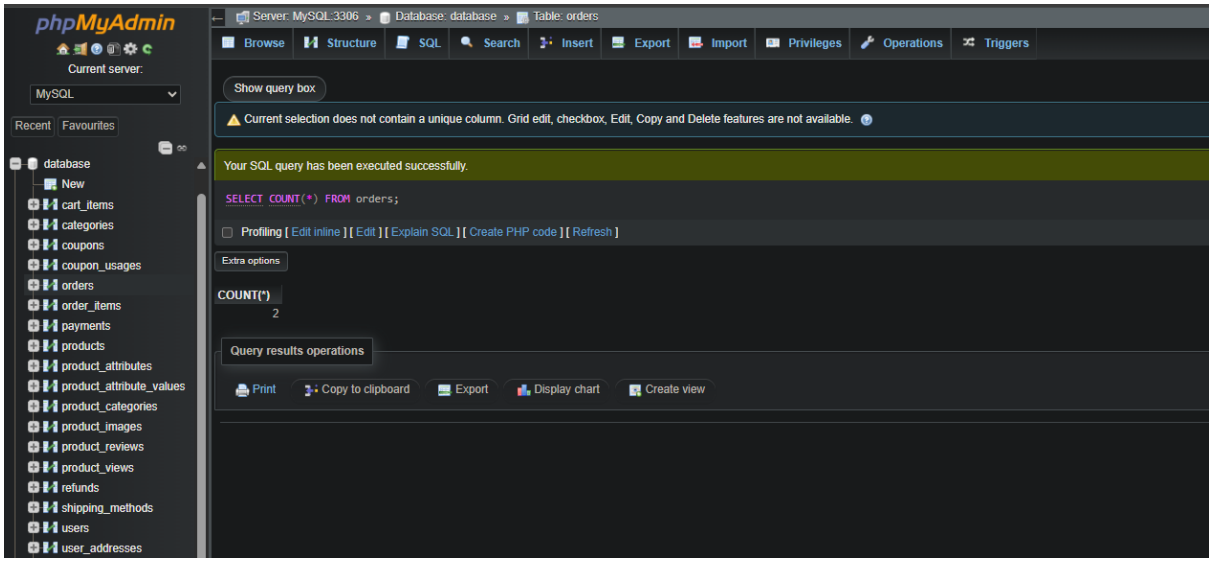
79.Get total number of products:

SELECT COUNT(*) FROM products;



80.Get total number of orders:

SELECT COUNT(*) FROM orders



81.Get revenue by month:

SELECT DATE_FORMAT(order_date, '%Y-%m') AS sales_month,

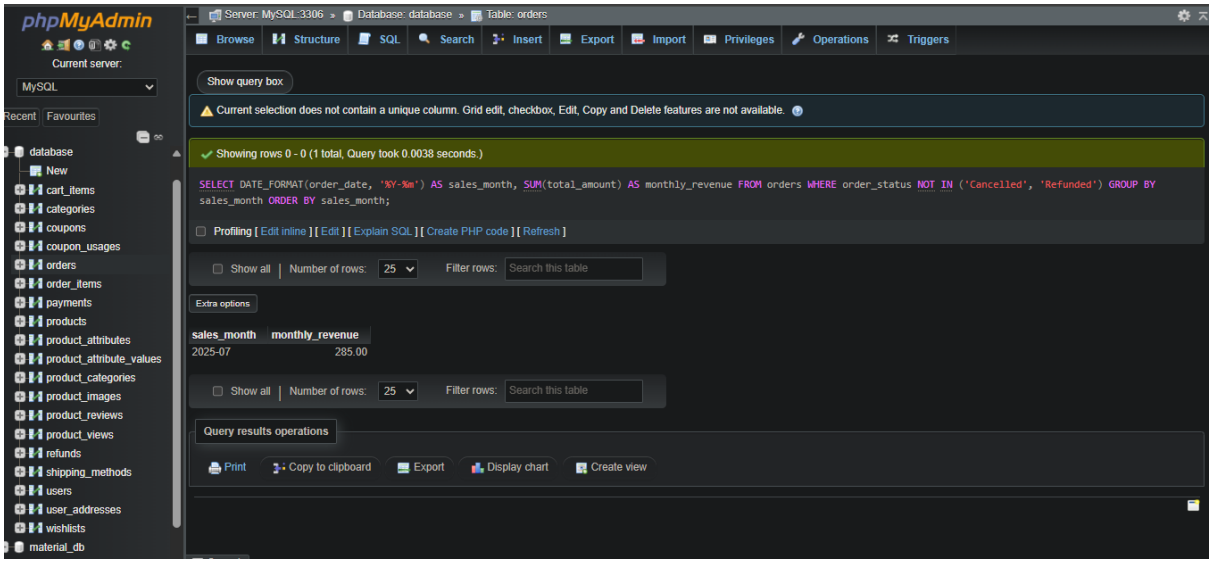
SUM(total_amount) AS monthly_revenue

FROM orders

WHERE order_status NOT IN ('Cancelled', 'Refunded')

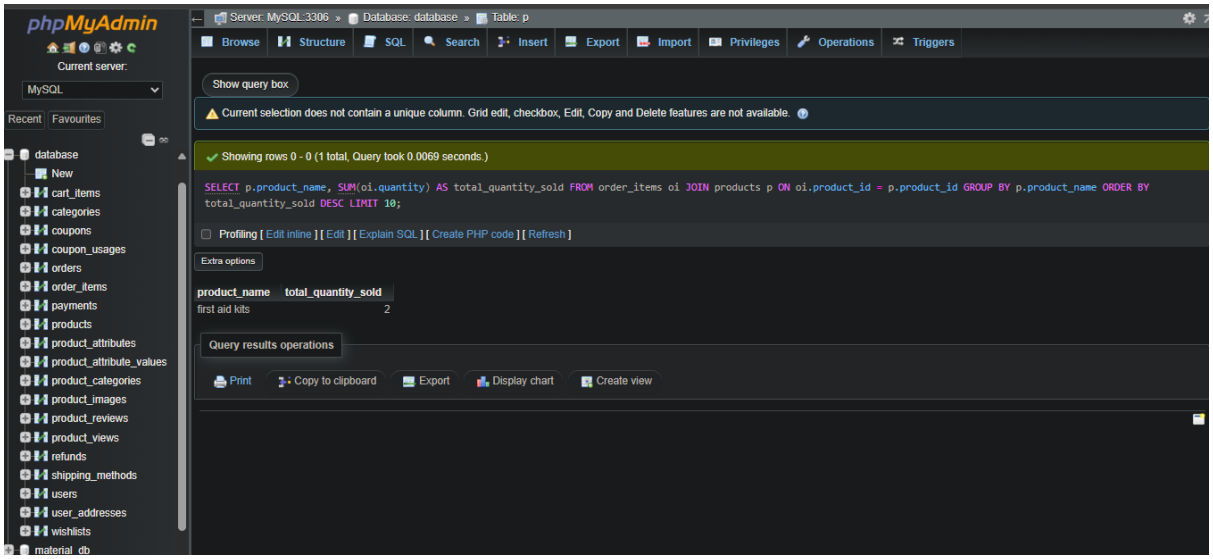
GROUP BY sales_month

ORDER BY sales_month



82.Get top 10 best-selling products (by quantity):

```
SELECT p.product_name, SUM(oi.quantity) AS total_quantity_sold
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_quantity_sold DESC
LIMIT 10
```



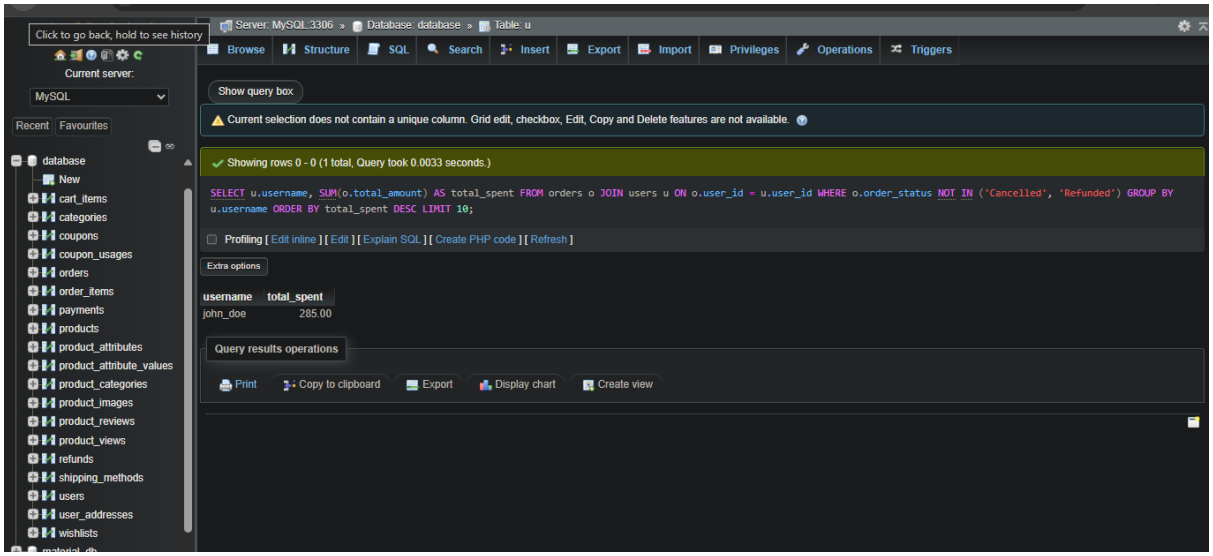
83.Get **top 10 customers (by total spend):**

```
SELECT u.username, SUM(o.total_amount) AS total_spent
FROM orders o
JOIN users u ON o.user_id = u.user_id
WHERE o.order_status NOT IN ('Cancelled', 'Refunded')
```

GROUP BY u.username

ORDER BY total_spent DESC

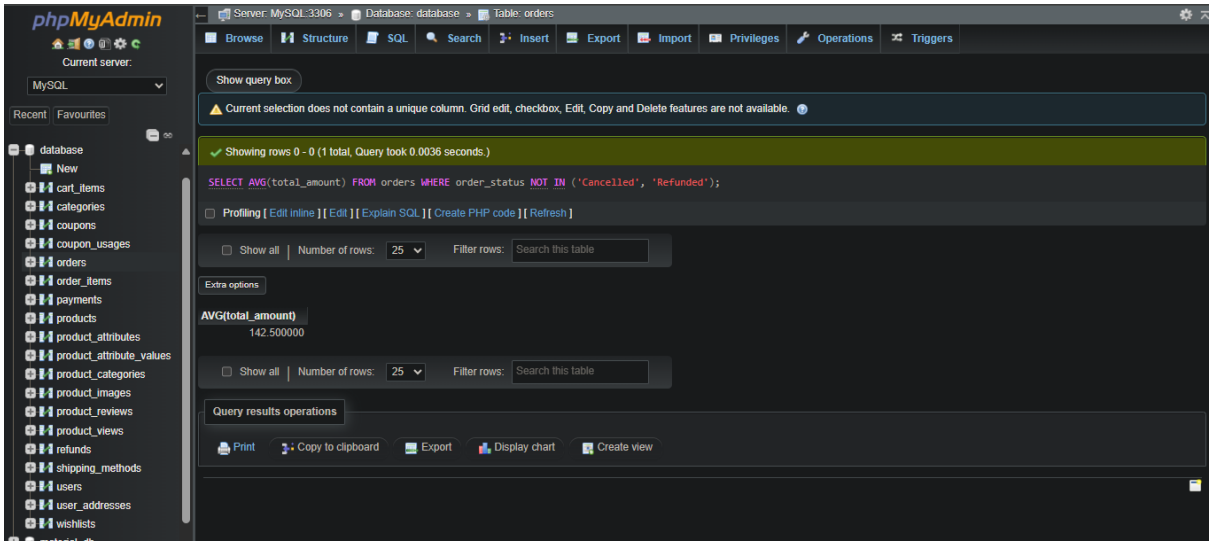
LIMIT 10;



84.Get average order value:

SELECT AVG(total_amount) FROM orders WHERE order_status NOT IN

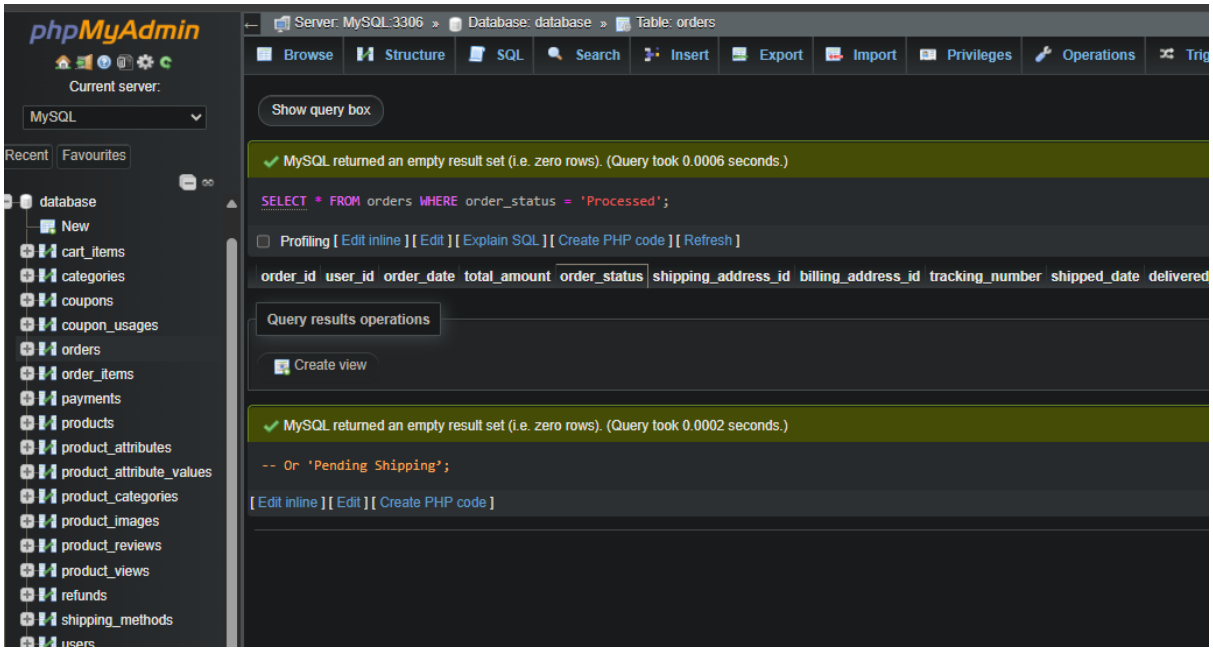
('Cancelled', 'Refunded')



85.Get orders with pending shipments:

SELECT * FROM orders WHERE order_status = 'Processed'; -- Or

'Pending Shipping'



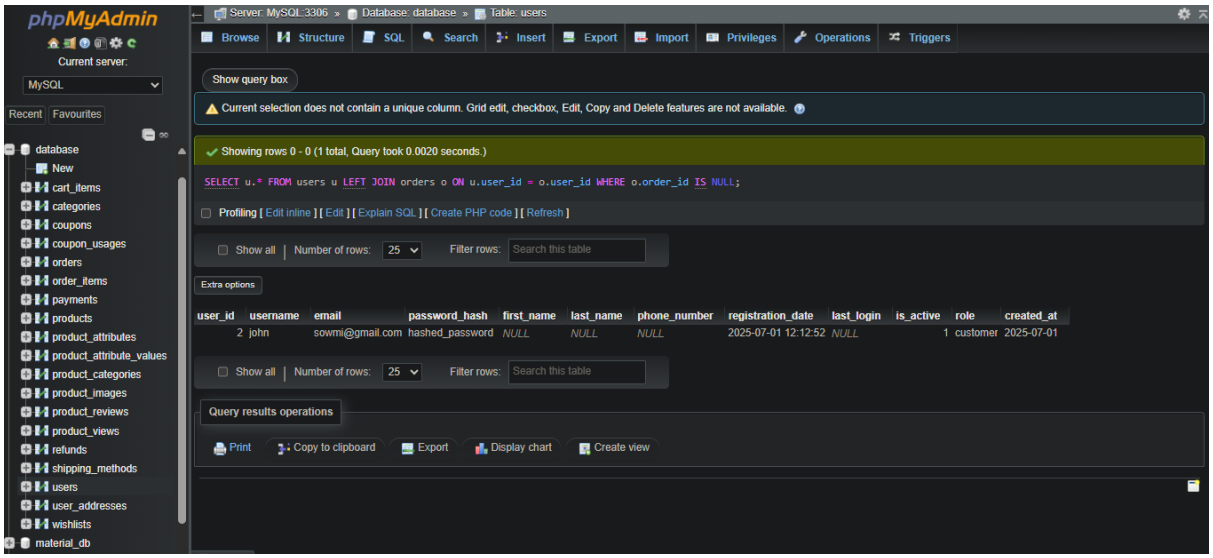
86.Get customers who haven't placed an order:

SELECT u.*

FROM users u

LEFT JOIN orders o ON u.user_id = o.user_id

WHERE o.order_id IS NULL;



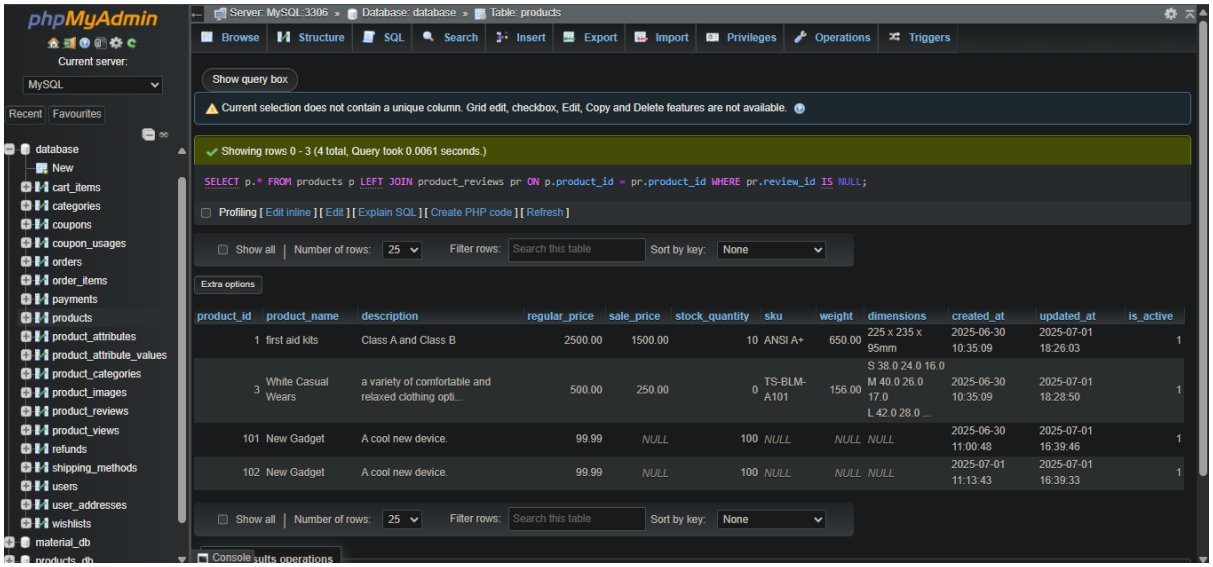
87.Get products with no reviews:

SELECT p.*

FROM products p

LEFT JOIN product_reviews pr ON p.product_id = pr.product_id

WHERE pr.review_id IS NULL



88. **Get product categories with their product count:**

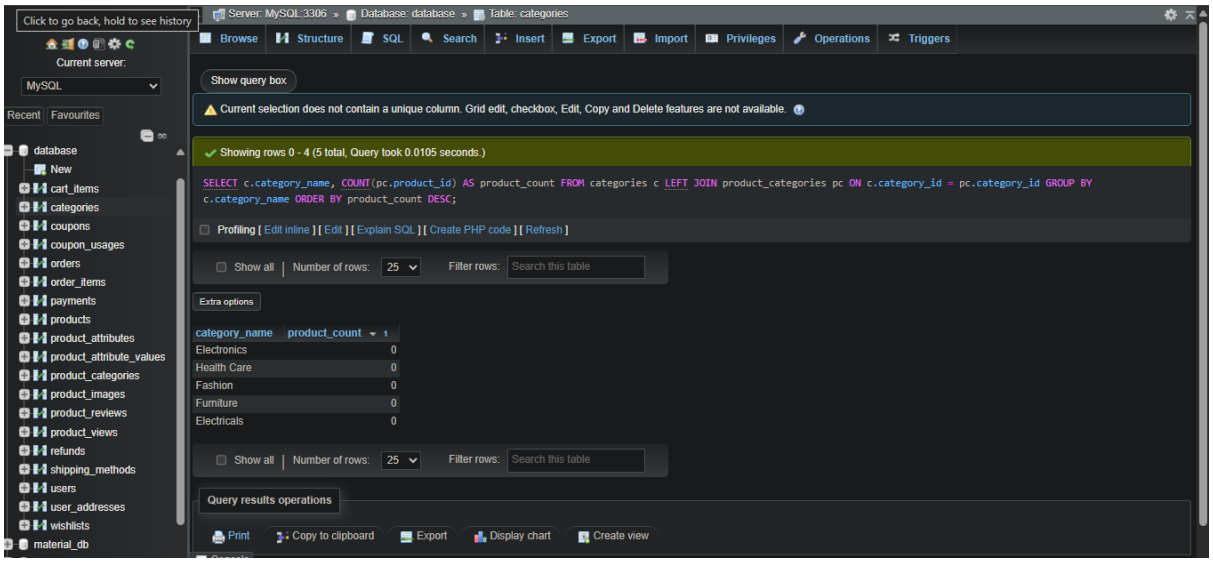
SELECT c.category_name, COUNT(pc.product_id) AS product_count

FROM categories c

LEFT JOIN product_categories pc ON c.category_id = pc.category_id

GROUP BY c.category_name

ORDER BY product_count DESC



89.Get daily sales trend:

SELECT DATE(order_date) AS sales_day, SUM(total_amount) AS

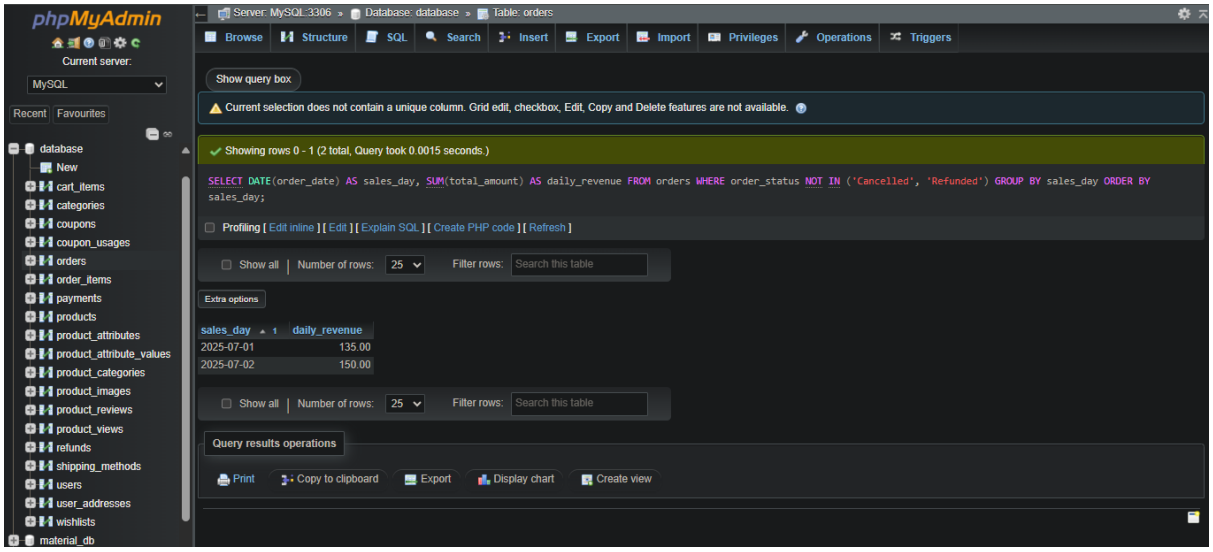
daily_revenue

FROM orders

WHERE order_status NOT IN ('Cancelled', 'Refunded')

GROUP BY sales_day

ORDER BY sales_day



90. **Get products that are frequently purchased together (simple example):**

SELECT oi1.product_id AS product1_id, oi2.product_id AS

product2_id, COUNT(*) AS co_occurrence_count

FROM order_items oi1

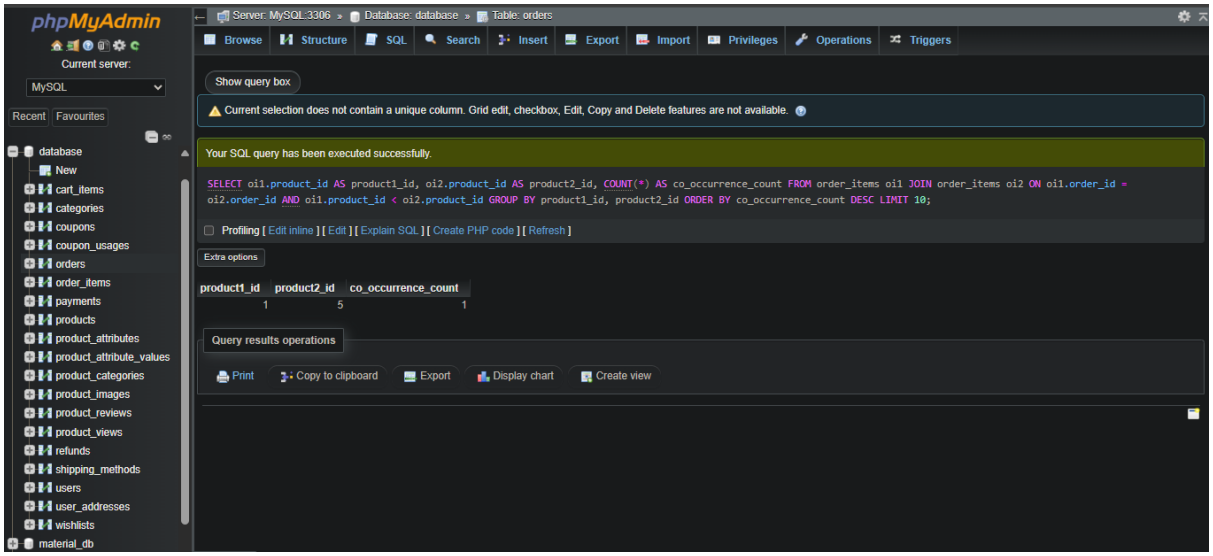
JOIN order_items oi2 ON oi1.order_id = oi2.order_id AND

oi1.product_id < oi2.product_id

GROUP BY product1_id, product2_id

ORDER BY co_occurrence_count DESC

LIMIT 10



91.Miscellaneous/Advanced

Calculate lifetime value (LTV) for each customer:

SELECT u.user_id, u.username, SUM(o.total_amount) AS

lifetime_value

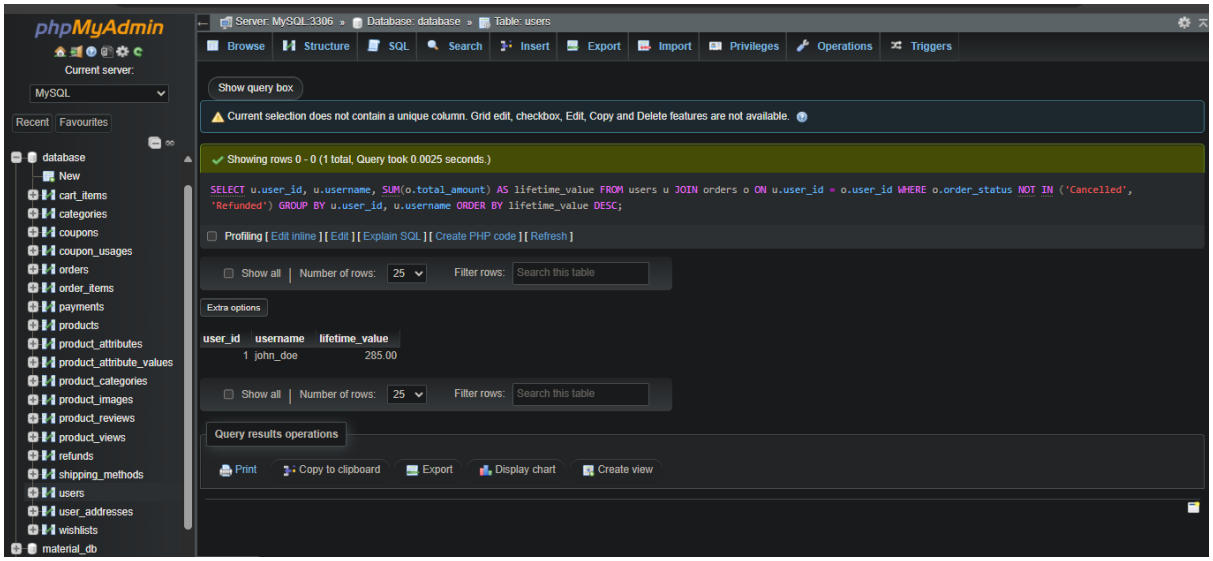
FROM users u

JOIN orders o ON u.user_id = o.user_id

WHERE o.order_status NOT IN ('Cancelled', 'Refunded')

GROUP BY u.user_id, u.username

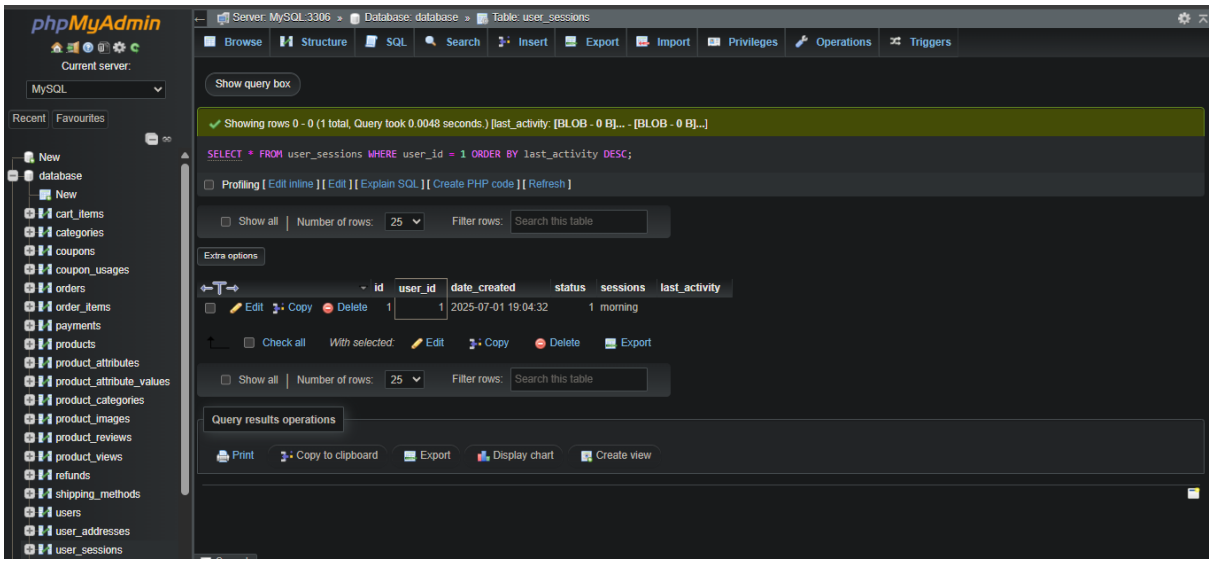
ORDER BY lifetime_value DESC;



92.Get user session activity (if you have a sessions table):

SELECT * FROM user_sessions WHERE user_id = 1 ORDER BY

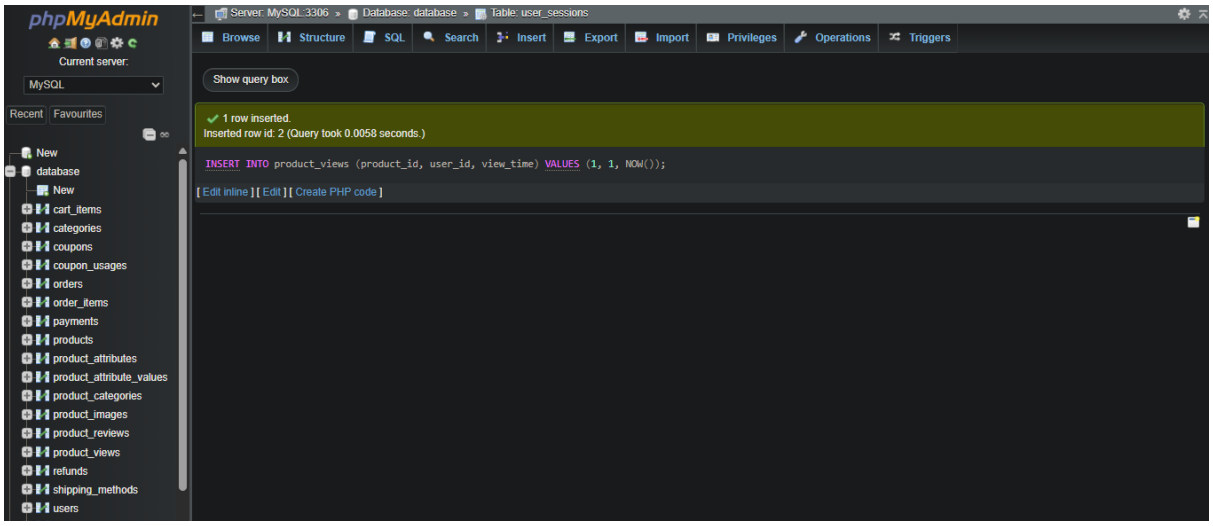
last_activity DESC



93.Record product view:

INSERT INTO product_views (product_id, user_id, view_time)

VALUES (1, 1, NOW())



94.Get products recently viewed by a user:

SELECT DISTINCT p.*

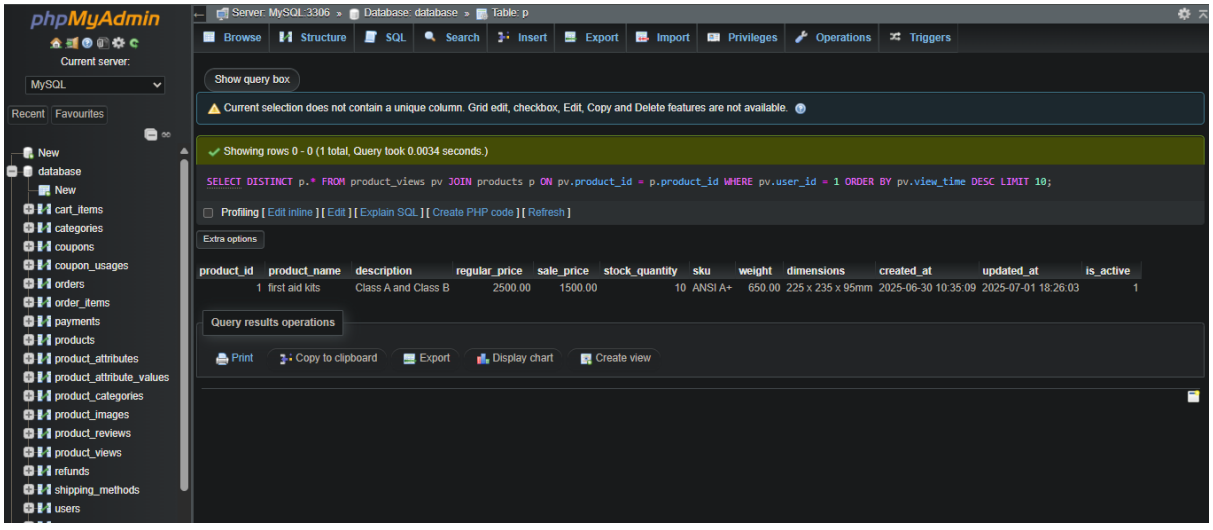
FROM product_views pv

JOIN products p ON pv.product_id = p.product_id

WHERE pv.user_id = 1

ORDER BY pv.view_time DESC

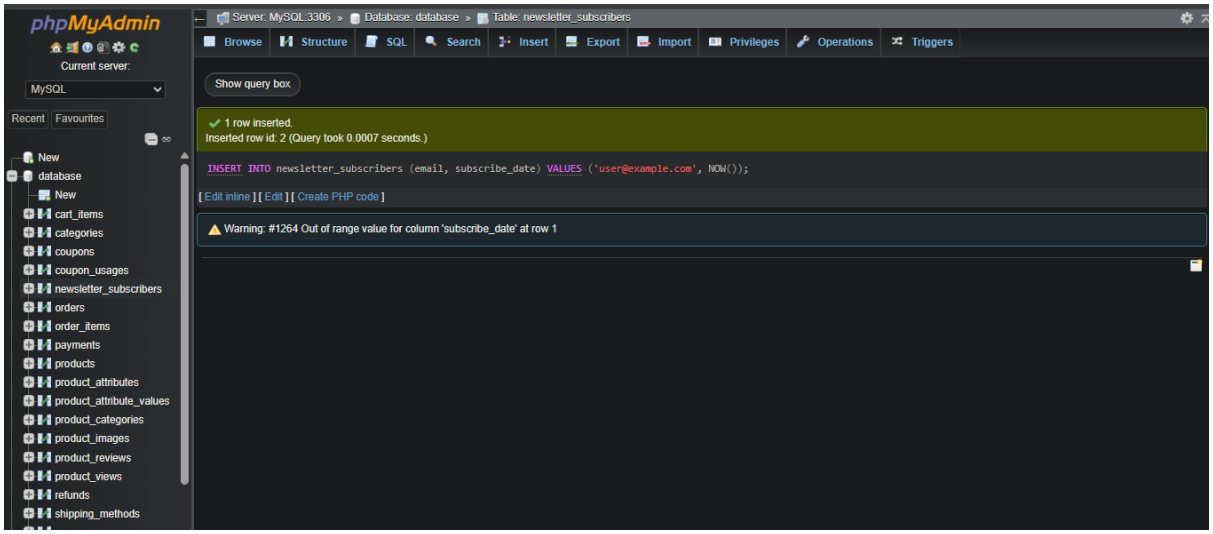
LIMIT 10;



95.Manage email subscriptions (if you have a newsletter table):

INSERT INTO newsletter_subscribers (email, subscribe_date) VALUES

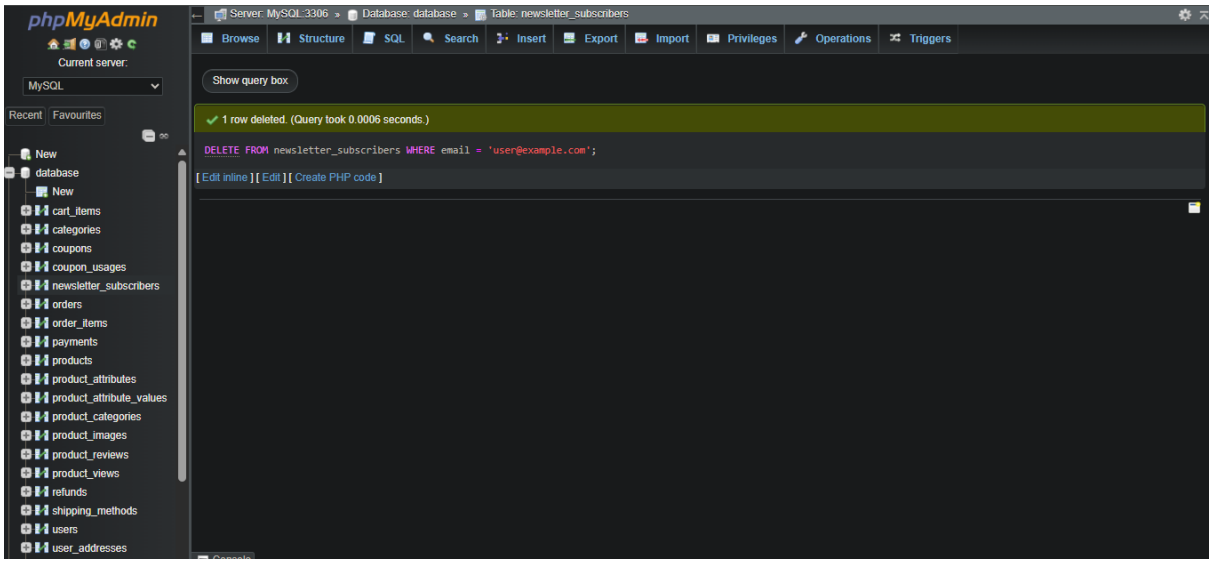
('user@example.com', NOW())



96.Unsubscribe from newsletter:

DELETE FROM newsletter_subscribers WHERE email =

'user@example.com'



97.Get all products with their associated categories:

SELECT p.product_name, GROUP_CONCAT(c.category_name SEPARATOR ',

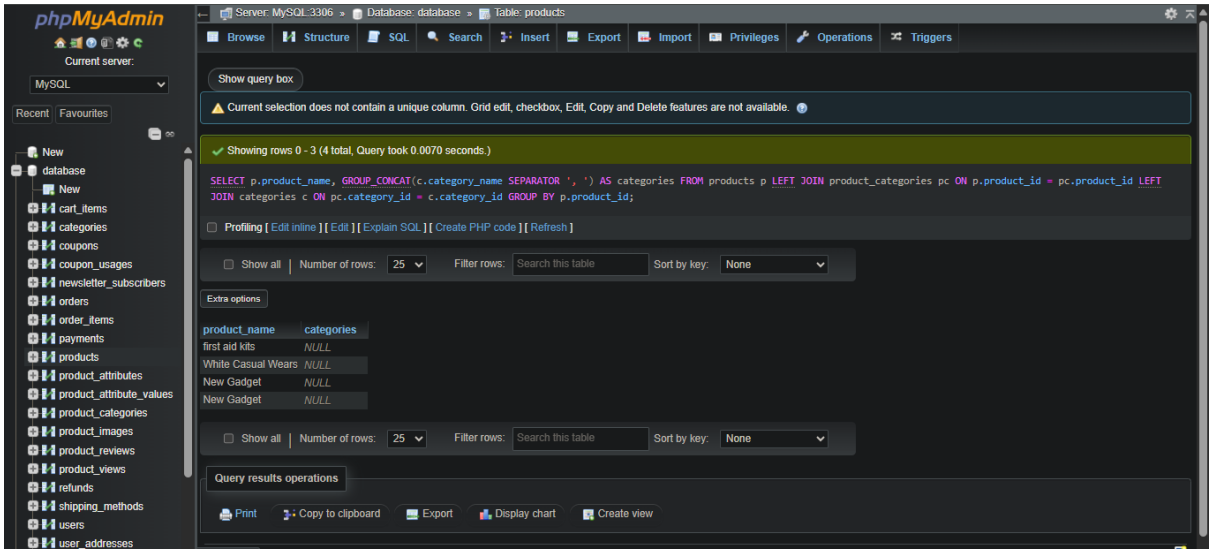
') AS categories

FROM products p

LEFT JOIN product_categories pc ON p.product_id = pc.product_id

LEFT JOIN categories c ON pc.category_id = c.category_id

GROUP BY p.product_id;



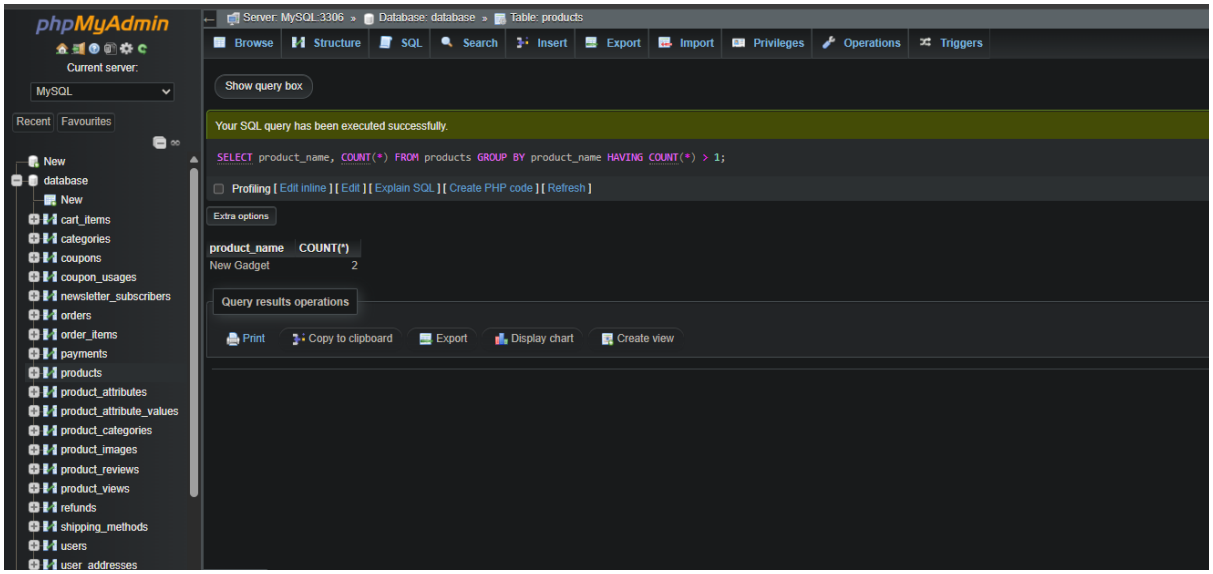
98.Find duplicate product names (for data cleaning):

SELECT product_name, COUNT(*)

FROM products

GROUP BY product_name

HAVING COUNT(*) > 1



99.Get users who have abandoned their cart:

SELECT u.*

FROM users u

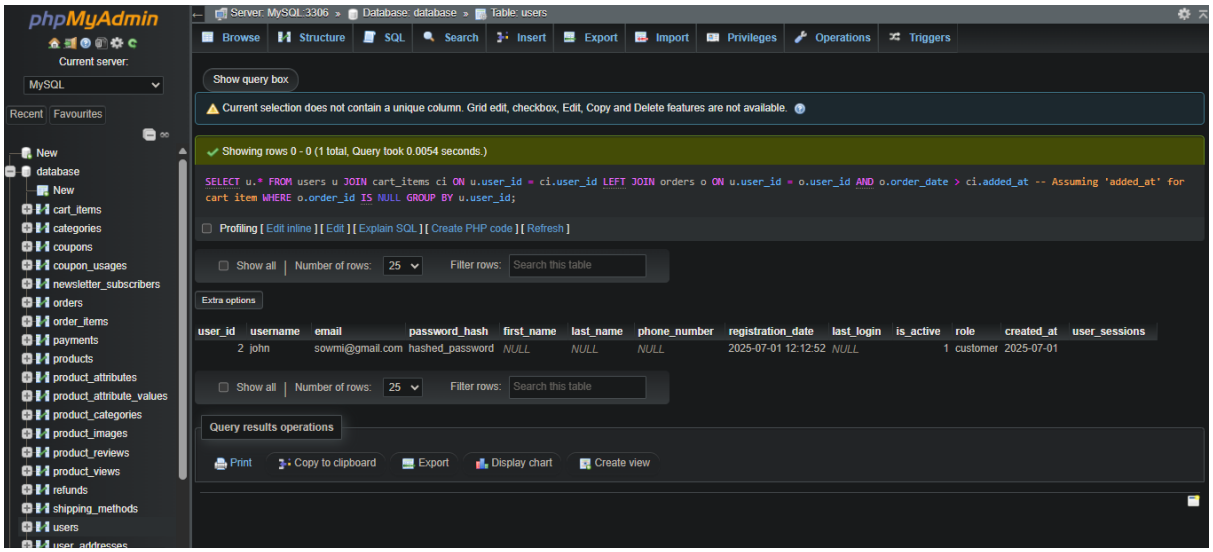
JOIN cart_items ci ON u.user_id = ci.user_id

LEFT JOIN orders o ON u.user_id = o.user_id AND o.order_date >

ci.added_at -- Assuming 'added_at' for cart item

WHERE o.order_id IS NULL

GROUP BY u.user_id;



100.**Archive old orders (moving them to an archive table for performance):**

CREATE TABLE archived_orders AS

SELECT * FROM orders

WHERE order_date < DATE_SUB(NOW(), INTERVAL 2 YEAR);