Coffee Ordering System Phase 2

(Emre Kaan Usta - Bahar Abit - Hüseyin Eren Yıldız - Ayşe Efdal Erdem)

1) Trigger Implementation

Ayşe Efdal ERDEM :

//Trigger 1 :
Trigger that updates the inventory after an order is placed.

```
DELIMITER //
CREATE TRIGGER update_inventory_after_order
AFTER INSERT ON OrderCoffeeItem
FOR EACH ROW
BEGIN
   DECLARE shop_id INT;

   -- Get shop ID from the order
   SELECT o.sID INTO shop_id
   FROM ORDERS o
   WHERE o.oID = NEW.oID;

   -- Update inventory
   UPDATE INVENTORY i
   SET i.remainingMilk = i.remainingMilk - (NEW.quantity * 30),
      i.remainingCoffee = i.remainingCoffee - (NEW.quantity * 20)
   WHERE i.sID = shop_id;
END;//
DELIMITER ;
```

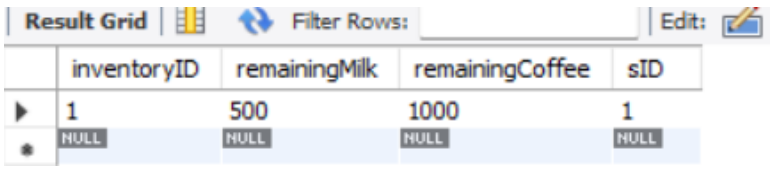
//Trigger 1 test

```
// Check Inventory Status
SELECT * FROM INVENTORY WHERE sID = 1;
// Add a New Order
INSERT INTO ORDERS (oID, sID, cID, oDate)
VALUES (12466, 1, 1, CURDATE());
```

```
// Add Order Details
INSERT INTO OrderCoffeeItem (oID, coffeeID, quantity)
VALUES (12466, 1, 2);
// Recheck Inventory Status
SELECT * FROM INVENTORY WHERE sID = 1;
```

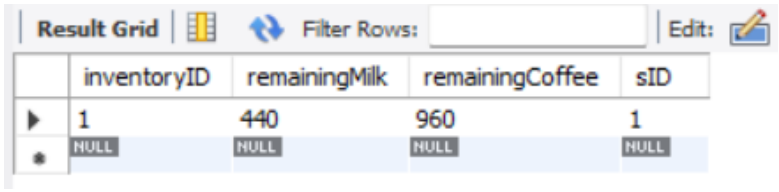
Trigger 1 Results

// Before Trigger



inventoryID	remainingMilk	remainingCoffee	sID
1	500	1000	1
NULL	NULL	NULL	NULL

//After Trigger



inventoryID	remainingMilk	remainingCoffee	sID
1	440	960	1
NULL	NULL	NULL	NULL

Trigger Functionality

The *update_inventory_after_order* trigger operates AFTER INSERT operations on the OrderCoffeeItem table. Its primary functions are:

1. Retrieve the shop ID associated with the new order
2. Automatically decrease inventory levels based on standardized recipe requirements:
    - 30 units of milk per coffee drink
    - 20 units of coffee per coffee drink

After the trigger execution, the inventory was automatically updated:

- Remaining Milk: 440 units (Decreased by 60 units: 2 drinks × 30 units of milk)
- Remaining Coffee: 960 units (Decreased by 40 units: 2 drinks × 20 units of coffee)

Emre Kaan Usta:

//Trigger:

```
DELIMITER $$
CREATE TRIGGER UpdateAverageReview
AFTER INSERT ON REVIEW
FOR EACH ROW
BEGIN
   -- Declare a variable to store the new average review
   DECLARE new_avg DECIMAL(3, 2);
   -- Calculate the new average review for the coffee shop
   SELECT AVG(rating) INTO new_avg
   FROM REVIEW
   WHERE sID = NEW.sID;
   -- Update the average_review column in the COFFESHOP table
   UPDATE COFFESHOP
   SET average_review = new_avg
   WHERE sID = NEW.sID;
END;
$$
DELIMITER ;
```

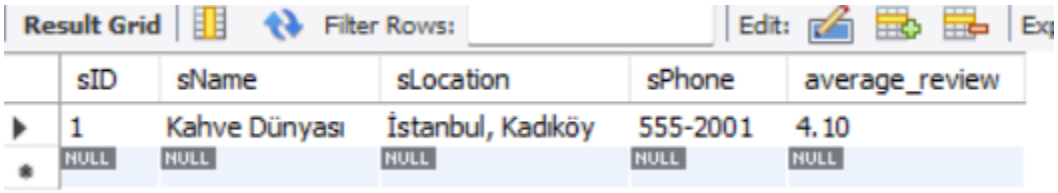
Trigger: `UpdateAverageReview`

- **Purpose: Automatically updates the `average_review` column in the `COFFESHOP` table whenever a new review is added to the `REVIEW` table.**
- **Logic:**
    1. **Declare Variable: `new_avg` is declared to hold the calculated average rating.**
    2. **Calculate Average Rating:**
        - **Computes the average rating (`AVG(rating)`) for all reviews associated with the coffee shop (`sID = NEW.sID`).**
        - **Stores the result in `new_avg`.**
    3. **Update Coffee Shop:**
        - **Updates the `average_review` column in the `COFFESHOP` table for the corresponding coffee shop.**

//Test:

```
SELECT * FROM COFFESHOP WHERE sID = 1;

INSERT INTO REVIEW (rID, sID, cID, rating)
VALUES (11, 1, 2, 5.0);

SELECT * FROM COFFESHOP WHERE sID = 1;
```

//Before

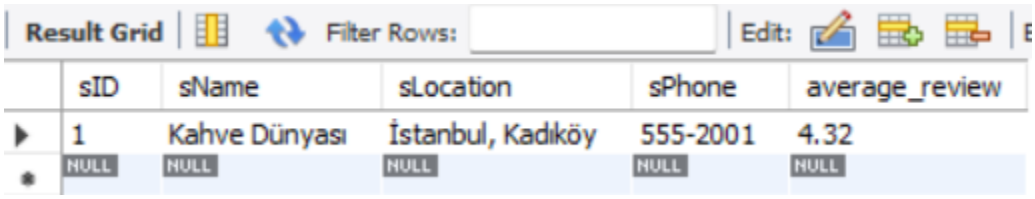


Result Grid | Filter Rows: | Edit:

sID	sName	sLocation	sPhone	average_review
1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.10
NULL	NULL	NULL	NULL	NULL

$$\text{New Average} = \frac{4.5 + 2.8 + 5.0}{3} = 4.10$$

//After



Result Grid | Filter Rows: | Edit:

sID	sName	sLocation	sPhone	average_review
1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.32
NULL	NULL	NULL	NULL	NULL

$$\frac{4.5 + 2.8 + 5.0 + 5.0}{4} = \frac{17.3}{4} = 4.325$$

Bahar ABİT

//Trigger:

```
-- Create Trigger to Update item_count in MENU Table
DELIMITER //
CREATE TRIGGER update_menu_coffee_item_count
AFTER INSERT ON MenuCoffeeItem
FOR EACH ROW
BEGIN
   DECLARE menu_item_count INT;

   -- Get the total count of coffee items for the menu
   SELECT COUNT(*) INTO menu_item_count
   FROM MenuCoffeeItem
   WHERE menuID = NEW.menuID;

   -- Update the item_count in MENU table
   UPDATE MENU
   SET item_count = menu_item_count
   WHERE menuID = NEW.menuID;
END;
DELIMITER ;
```

-- Trigger Test:

```
Insert a new MenuCoffeeItem and check the item_count
INSERT INTO MenuCoffeeItem (menuID, coffeeID) VALUES (1, 10);  -- Add a coffee item to
'Sabancı Menüsü'
SELECT * FROM MenuCoffeeItem WHERE menuID = 1;
```

Trigger Functionality

The `update_menu_coffee_item_count` trigger operates **AFTER INSERT** operations on the `MenuCoffeeItem` table. Its primary functions are:

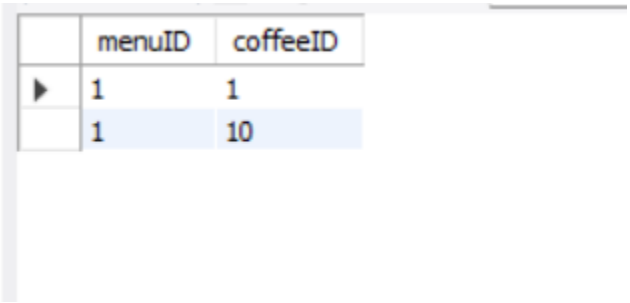
1. **Recalculate the total coffee items**:
    - Retrieve the `menuID` of the newly inserted coffee item.
    - Count the total number of coffee items (`coffeeID`) associated with the affected `menuID` by querying the `MenuCoffeeItem` table.
2. **Automatically update the `item_count` field** in the `MENU` table:
    - Reflect the updated count of coffee items for the specific menu.

After the trigger execution, the `item_count` field in the `MENU` table was automatically updated to reflect the accurate number of coffee items associated with the corresponding menu.

Before Trigger:

	menuID	coffeeID
▶	1	1

After Trigger:



	menuID	coffeeID
▶	1	1
	1	10

Hüseyin Eren YILDIZ:

Trigger that updates the bartender counts after a new bartender is added to a coffee shop.

Trigger Implementation:

```
DELIMITER //

CREATE TRIGGER update_bartender_count
AFTER INSERT ON BARTENDER
FOR EACH ROW
BEGIN
   DECLARE bartender_count INT;

   -- Calculate the number of bartenders in the associated coffee shop
   SELECT COUNT(*)
   INTO bartender_count
   FROM BARTENDER
   WHERE sID = NEW.sID;

   -- Update the number of bartenders in the coffee shop
   UPDATE COFFESHOP
   SET bartender_count = bartender_count + 1 -- Yeni bir barmen eklendiği için 1 artırıyoruz
   WHERE sID = NEW.sID;
END //

DELIMITER ;
```

Trigger Test:

```
-- Add a new bartender
INSERT INTO BARTENDER (bID, sID, bName)
VALUES (21, 2, 'Ahmet');

-- SELECT * FROM BARTENDER WHERE sID = 2;

SELECT sID, sName, sLocation, bartender_count
FROM COFFESHOP
WHERE sID = 2; -- To select a specific coffee shop
```

Before Trigger:

	sID	sName	sLocation	bartender_count
▶	2	Starbucks	İstanbul, Beşiktaş	2
*	NULL	NULL	NULL	NULL

	sID	sName	sLocation	sPhone	average_review	bartender_count
▶	1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.03	2
	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99	2
	3	Punto	İstanbul, Nişantaşı	555-2003	1.31	1
	4	MOC	İstanbul, Taksim	555-2004	2.36	1
	5	Kahve Durağı	Ankara, Çankaya	555-2005	3.65	2
	6	Cafe Nero	İzmir, Alsancak	555-2006	4.56	1
	7	V60 Coffee	Bursa, Osmangazi	555-2007	2.43	1
	8	Cafe Gusto	Antalya, Kaleiçi	555-2008	3.76	0

After Trigger:

Result Grid | Filter Rows: | Edit:

	sID	sName	sLocation	bartender_count
▶	2	Starbucks	İstanbul, Beşiktaş	3
*	NULL	NULL	NULL	NULL

	sID	sName	sLocation	sPhone	average_review	bartender_count
▶	1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.03	2
	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99	3
	3	Punto	İstanbul, Nişantaşı	555-2003	1.31	1
	4	MOC	İstanbul, Taksim	555-2004	2.36	1
	5	Kahve Durağı	Ankara, Çankaya	555-2005	3.65	2
	6	Cafe Nero	İzmir, Alsancak	555-2006	4.56	1
	7	V60 Coffee	Bursa, Osmangazi	555-2007	2.43	1
	8	Cafe Gusto	Antalya, Kaleiçi	555-2008	3.76	0

Trigger Functionality:

The update_bartender_count trigger operates AFTER INSERT operations on the BARTENDER table. Its primary functionality is to automatically update the bartender_count column in the COFFESHOP table when a new bartender is added to a coffee shop.

Workflow:

- **Retrieve the Current Bartender Count**

The trigger calculates the total number of bartenders currently assigned to the coffee shop associated with the new bartender (NEW.sID).

- **Update the Coffee Shop's Bartender Count**

The bartender_count in the COFFESHOP table is incremented by 1 to reflect the addition of the new bartender.

Testing the Trigger:

Step 1: Add a New Bartender

A new bartender with the name 'Ahmet' is added to the coffee shop with sID = 2:

Step 2: Verify the Updated Bartender Count

Verify that the bartender_count in the COFFESHOP table has been incremented.

Example Scenario

Before adding the new bartender:

- bartender_count: 2 (for sID=2). The BARTENDER table will record the new entry.

After the new bartender is added:

- bartender_count: 3 (for sID=2). The COFFESHOP table will automatically update the bartender_count for the Starbucks location, incrementing it by 1.

This change confirms the trigger was executed successfully and updated the coffee shop's bartender count. This ensures consistency between the BARTENDER and COFFESHOP tables.

2) Stored Procedure Implementation

Ayşe Efdal ERDEM :

// stored procedure for placing an order and managing payments:

```
DELIMITER //
CREATE PROCEDURE place_coffee_order(
    IN p_customer_id INT,
    IN p_shop_id INT,
    IN p_coffee_id INT,
    IN p_quantity INT,
    IN p_payment_type VARCHAR(200),
    OUT p_order_id INT
)
BEGIN
    DECLARE coffee_price INT;
    DECLARE total_amount INT;
    DECLARE payment_id INT;

    // Get coffee price
    SELECT price INTO coffee_price
    FROM COFFEEITEMS
    WHERE coffeeID = p_coffee_id;

    // Calculate total
    SET total_amount = coffee_price * p_quantity;
    SET payment_id = FLOOR(11 + RAND() * 89);  // Random payment ID between 11-99
    SET p_order_id = FLOOR(12466 + RAND() * 87533);  // Random order ID

    // Insert order
    INSERT INTO ORDERS (oID, sID, cID, oDate)
    VALUES (p_order_id, p_shop_id, p_customer_id, CURDATE());

   //  Insert order items
    INSERT INTO OrderCoffeeItem (oID, coffeeID, quantity)
    VALUES (p_order_id, p_coffee_id, p_quantity);

    // Create payment record
```

```
    INSERT INTO PAYMENT (amount, pID, oID, PType, PStatus)
    VALUES (total_amount, payment_id, p_order_id, p_payment_type, 1);

END//
DELIMITER ;

// stored  procedure test 1
SET @order_id = 0;
CALL place_coffee_order(1, 1, 1, 2, 'Credit Card', @order_id);
SELECT @order_id;

// Check new order
SELECT * FROM ORDERS WHERE oID = @order_id;
// Check order items
SELECT * FROM OrderCoffeeItem WHERE oID = @order_id;
// Check payment
SELECT * FROM PAYMENT WHERE oID = @order_id;

//Test results

// ORDERS Table Result
```

Result Grid | Filter Rows:

oID	sID	cID	oDate
12473	1	1	2024-12-05
NULL	NULL	NULL	NULL

// OrderCoffeeItem Table Result

Result Grid | Filter Rows:

oID	coffeeID	quantity
12473	1	2

// PAYMENT Table Result

Result Grid | Filter Rows:

amount	pID	oID	PType	PStatus
160	41	12473	Credit Card	1
NULL	NULL	NULL	NULL	NULL

Process Flow

1. Price Retrieval
    - Fetches the unit price from COFFEEITEMS table
    - Calculates total amount based on quantity
2. ID Generation
    - Generates unique payment ID (range: 11-99)
    - Generates unique order ID (range: 12466-99999)
3. Transaction Processing
    - Creates order record in ORDERS table
    - Inserts order details in OrderCoffeeItem table
    - Records payment information in PAYMENT table

The test results showcase a unified transaction where a customer (ID: 1) placed an order at a coffee shop (Shop ID: 1) for 2 units of a coffee product (Coffee ID: 1). The system successfully:

- Generated order #12473 on December 5, 2024 (ORDERS table)
- Recorded the order details with the specified quantity (OrderCoffeeItem table)
- Processed a credit card payment of 160 units with payment ID 41 and active status (PAYMENT table)

Emre Kaan USTA :

//Stored Procedure For Updating Review

```
DELIMITER //
CREATE PROCEDURE add_review(
  IN p_rID INT,
  IN p_sID INT,
  IN p_cID INT,
  IN p_rating DECIMAL(2,1)
)
BEGIN
  -- Validate rating range
  IF p_rating BETWEEN 1.0 AND 5.0 THEN
    -- Insert review
    INSERT INTO REVIEW (rID, sID, cID, rating)
    VALUES (p_rID, p_sID, p_cID, p_rating);

    -- Get updated average
    SELECT average_review
    FROM COFFESHOP
    WHERE sID = p_sID;
  ELSE
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Rating must be between 1.0 and 5.0';
  END IF;
END //
DELIMITER ;
```

Stored Procedure: `add_review`

- **Purpose:** Adds a review to the `REVIEW` table and retrieves the updated average rating of a coffee shop.
- **Parameters:**
  - `p_rID`: Review ID
  - `p_sID`: Coffee shop ID
  - `p_cID`: Customer ID
  - `p_rating`: Rating (1.0 to 5.0)
-

- **Logic:**
  - Validates `p_rating` (must be between 1.0 and 5.0). Throws an error if invalid.
  - Inserts the review into the `REVIEW` table.
  - Retrieves the updated `average_review` for the specified coffee shop (`sID = p_sID`).

//Test

```
SELECT * FROM COFFESHOP WHERE sID = 1;
CALL add_review(12, 1, 3, 4.5);
SELECT * FROM COFFESHOP WHERE sID = 1;
```

//Before

Result Grid | Filter Rows: | Edit:

sID	sName	sLocation	sPhone	average_review
1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.32
NULL	NULL	NULL	NULL	NULL

//After

Result Grid | Filter Rows: | Edit:

sID	sName	sLocation	sPhone	average_review
1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.36
NULL	NULL	NULL	NULL	NULL

BAHAR ABİT

// A stored procedure to handle order placement and payment management

```
DELIMITER //

CREATE PROCEDURE GetMenuDetails(
   IN input_menuID INT
)
BEGIN
  -- --Display menu details (check correct column names)
    SELECT menuID, name AS menu_name, item_count
    FROM MENU
    WHERE menuID = input_menuID;

    -- -- List coffee items associated with the menu


    SELECT coffeeID
    FROM MenuCoffeeItem
    WHERE menuID = input_menuID;
END //
```

//stored procedure test

```
DELIMITER ;
INSERT INTO COFFEEITEMS (coffeeID, coffeeName) VALUES (16, 'Espresso');
INSERT INTO COFFEEITEMS (coffeeID, coffeeName) VALUES (11, 'Latte');

INSERT INTO MenuCoffeeItem (menuID, coffeeID) VALUES (1, 16);
INSERT INTO MenuCoffeeItem (menuID, coffeeID) VALUES (1, 11);

CALL GetMenuDetails(1);

SELECT * FROM MenuCoffeeItem WHERE menuID = 1;
SELECT * FROM MENU WHERE menuID = 1;
```

Purpose

The `GetMenuDetails` stored procedure retrieves detailed information about a menu from the `MENU` table and lists all associated coffee items from the `MenuCoffeeItem` table. This ensures that the menu details and associated coffee items are displayed in an organized and efficient manner.

Parameters

- `input_menuID` (INT): The ID of the menu for which details and associated coffee items are to be retrieved.

Logic

1. Menu Details Retrieval
    - Fetches the details of the menu with the given `menuID` from the `MENU` table.
    - Retrieves:
        - `menuID`: Unique identifier for the menu.
        - `menu_name`: Name of the menu (retrieved as `mName`).
        - `item_count`: The total number of coffee items associated with the menu.
2. Associated Coffee Items Retrieval
    - Queries the `MenuCoffeeItem` table to fetch all `coffeeID` values linked to the provided `menuID`.
    - Organizes the list of associated coffee items for clear output.

Before Procedure:

menuID	coffeeID
1	1
1	10

After Procedure:

menuID	coffeeID
1	1
1	10
1	16
1	11

Hüseyin Eren YILDIZ

This procedure handles the insertion of a new bartender into the BARTENDER table and automatically updates the corresponding bartender count in the COFFESHOP table.

Process Flow:

1. Adding a New Bartender

The procedure accepts the following inputs:

- bartenderID: A unique identifier for the bartender.
- shopID: The ID of the coffee shop where the bartender will work.
- bartenderName: The name of the bartender being added.

It inserts these details into the BARTENDER table.

2. Updating Bartender Count

After successfully adding the bartender, the procedure calculates the total number of bartenders working at the specified coffee shop (shopID) by counting the rows in the BARTENDER table. Then, it automatically updates the bartender count for the specified coffee shop in the COFFESHOP table.

Summary:

Bartender Insertion: Inserts the bartender details into the BARTENDER table.

Count Calculation: Calculates the total number of bartenders for the specified coffee shop.

Database Update: Updates the bartender_count column in the COFFESHOP table to ensure consistency.

Testing Procedure:

Step 1: Add a New Bartender (Procedure CALL)

A new bartender is added with bartenderID: 24, shopID: 2 (Coffee shop ID: Starbucks), bartenderName: Dries Mertens (CALL AddBartenderAndUpdateCount(24, 2, 'Dries Mertens');)

Step 2: Output Verification

1) *Check Bartender Record*

The new bartender is added to the BARTENDER table. A record for Dries Mertens with bID = 24 appears in the table

2) *Check Bartender Count*

The bartender_count for the coffee shop (sID = 2) is updated. The count is incremented by 1.

Notes:

- I ensure that both the BARTENDER and COFFESHOP tables are properly defined and that `sID` (shopID) is a valid foreign key in the BARTENDER table.

- The procedure assumes that the `bartender_count` column exists in the COFFESHOP table.

// *Stored Procedure for Adding a Bartender and Updating the Bartender Count:*

```
DELIMITER //

CREATE PROCEDURE AddBartenderAndUpdateCount (
    IN bartenderID INT,
    IN shopID INT,
    IN bartenderName VARCHAR(200)
)
BEGIN
    -- Add a new bartender (First Step)
    INSERT INTO BARTENDER (bID, sID, bName)
    VALUES (bartenderID, shopID, bartenderName);

   -- Calculate the current number of bartenders for the specified coffee shop (Second Step)
    SELECT COUNT(*)
    FROM BARTENDER
    WHERE sID = shopID;

    -- Update the number of bartenders in the associated coffee shop (Second Step)
    UPDATE COFFESHOP
    SET bartender_count = (
        SELECT COUNT(*)
        FROM BARTENDER
```

```
        WHERE sID = shopID
    )
    WHERE sID = shopID;
END;
//

DELIMITER ;
```

Stored Procedure Test

```
-- Call the procedure to add a new bartender and update the count
CALL AddBartenderAndUpdateCount(24, 2, 'Dries Mertens');

-- Retrieve bartenders for a specific coffee shop (Checking)
SELECT * FROM BARTENDER WHERE sID = 2;

-- Retrieve the bartender count for a specific coffee shop (Checking)
SELECT sID, bartender_count FROM COFFESHOP WHERE sID = 2;
```

Before Procedure:

sID	bartender_count
2	3
NULL	NULL

bID	sID	bName
3	2	Barış Alper
4	2	Kaan Ayhan
21	2	Ahmet
NULL	NULL	NULL

After Procedure:

sID	bartender_count
2	4
NULL	NULL

bID	sID	bName
3	2	Barış Alper
4	2	Kaan Ayhan
21	2	Ahmet
24	2	Dries Mertens
NULL	NULL	NULL

3) Web Access Module

Ayşe Efdal ERDEM

// HTML Form (index.html)

// The HTML form efficiently captures all required order information through a simple interface with validated inputs and a payment type dropdown menu.

```
<!DOCTYPE html>
<html>
<head>
   <title>Coffee Order</title>
</head>
<body>
   <h2>Place Coffee Order</h2>
   <form action="process.php" method="POST">
      <p>
         <label>Customer ID:</label>
         <input type="number" name="customer_id" required>
      </p>
      <p>
         <label>Shop ID:</label>
         <input type="number" name="shop_id" required>
      </p>
      <p>
         <label>Coffee ID:</label>
         <input type="number" name="coffee_id" required>
      </p>
      <p>
         <label>Quantity:</label>
         <input type="number" name="quantity" required>
      </p>
     <p>
         <label>Payment Type:</label>
         <select name="payment_type">
            <option value="Kredi Kartı">Kredi Kartı</option>
            <option value="Nakit">Nakit</option>
            <option value="Havale">Havale</option>
         </select>
      </p>
```

```
        <button type="submit">Order</button>
    </form>
</body>
</html>
```

//Database Connection (db.php)

//The database connection component securely establishes and manages MySQL connectivity with proper error handling for connection issues.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "phase3";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

// Order Processing (process.php)

// The order processing script manages the complete workflow from form submission through stored procedure execution to displaying order confirmation, ensuring successful transaction completion.

```
<?php
include 'db.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $customer_id = $_POST['customer_id'];
    $shop_id = $_POST['shop_id'];
    $coffee_id = $_POST['coffee_id'];
    $quantity = $_POST['quantity'];
    $payment_type = $_POST['payment_type'];

    // Use the stored procedure
    $stmt = $conn->prepare("CALL place_coffee_order(?, ?, ?, ?, ?, @order_id)");
    $stmt->bind_param("iiisi", $customer_id, $shop_id, $coffee_id, $quantity, $payment_type);
```

```
    if ($stmt->execute()) {
        // Retrieve the generated order ID
        $result = $conn->query("SELECT @order_id AS order_id");
        $row = $result->fetch_assoc();
        $order_id = $row['order_id'];

        echo "<h2>Order Successfully Placed!</h2>";
        echo "Order ID: " . $order_id . "<br>";
        echo "Customer ID: " . $customer_id . "<br>";
        echo "Shop ID: " . $shop_id . "<br>";
        echo "Coffee ID: " . $coffee_id . "<br>";
        echo "Quantity: " . $quantity . "<br>";
        echo "Payment Type: " . $payment_type . "<br>";
    } else {
        echo "Error: " . $stmt->error;
    }
    $conn->commit();

    $stmt->close();
    $conn->close();
}
?>
```

// INTERFACE SCREENSHOTS

// Order Input Form

// Order Confirmation

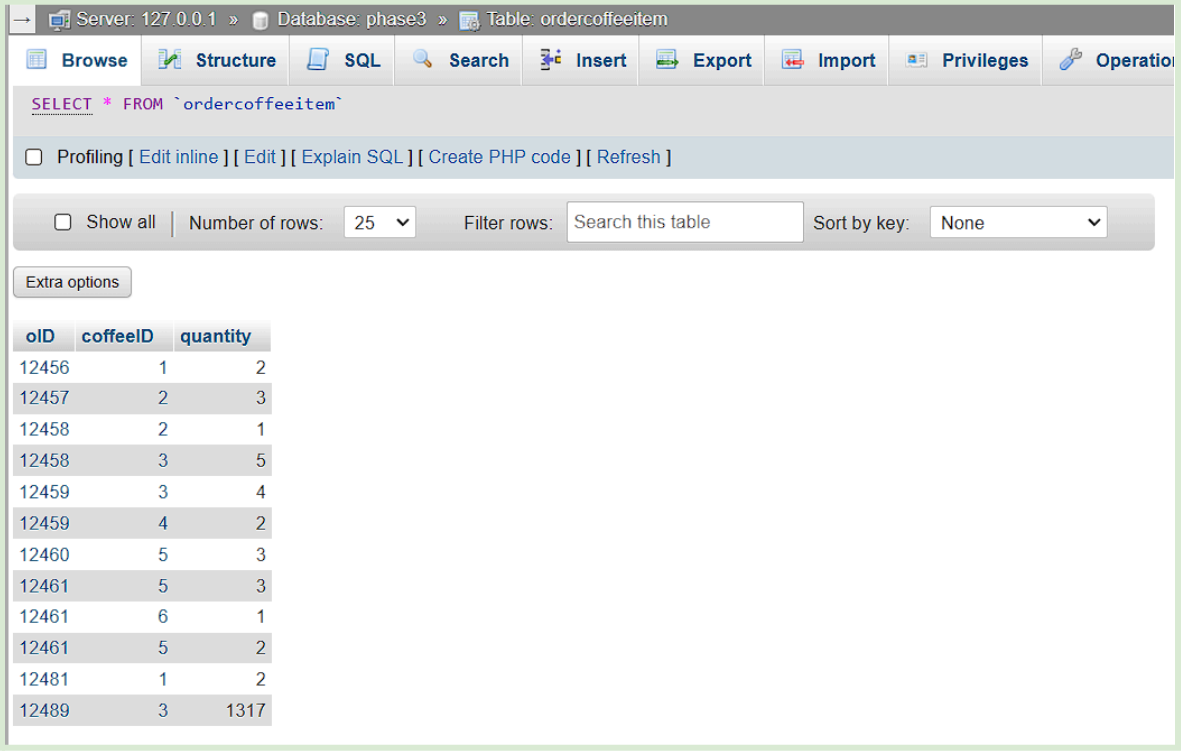
//The confirmation page displays a complete summary of the successful order including the generated order ID and all entered details.

Order Successfully Placed!

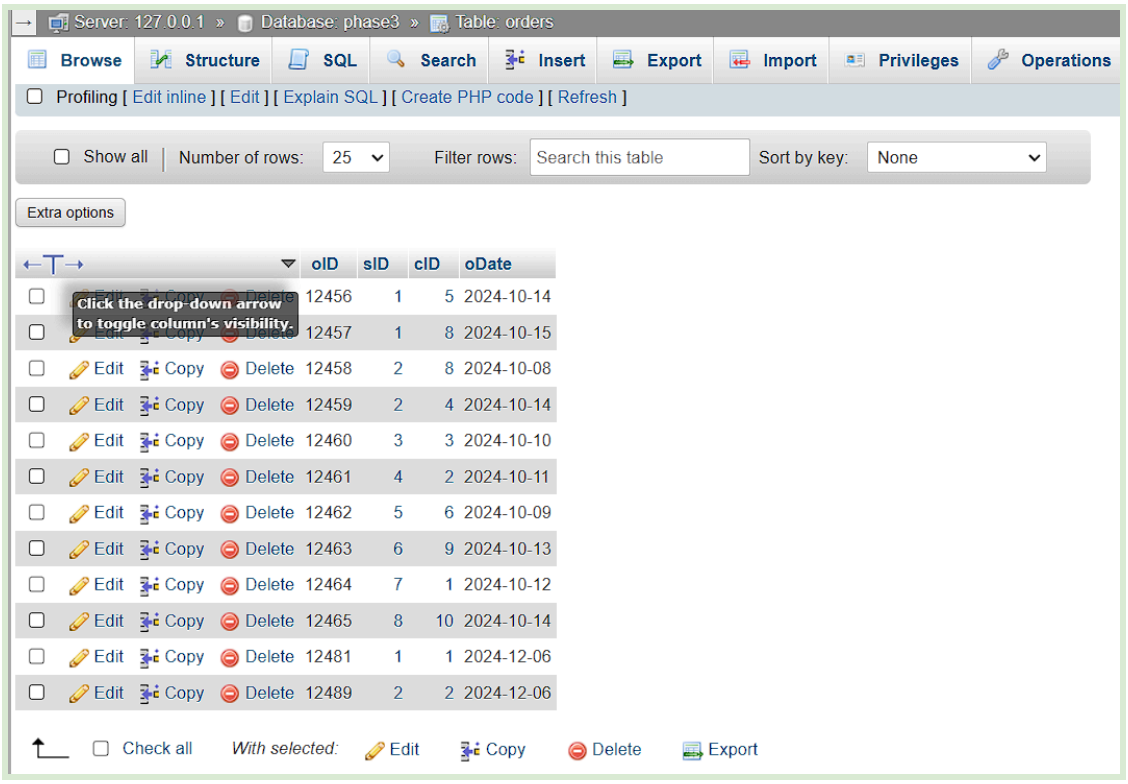
Order ID: 12466
Customer ID: 1
Shop ID: 1
Coffee ID: 1
Quantity: 17
Payment Type: Nakit

// DATABASE STATE CHANGES

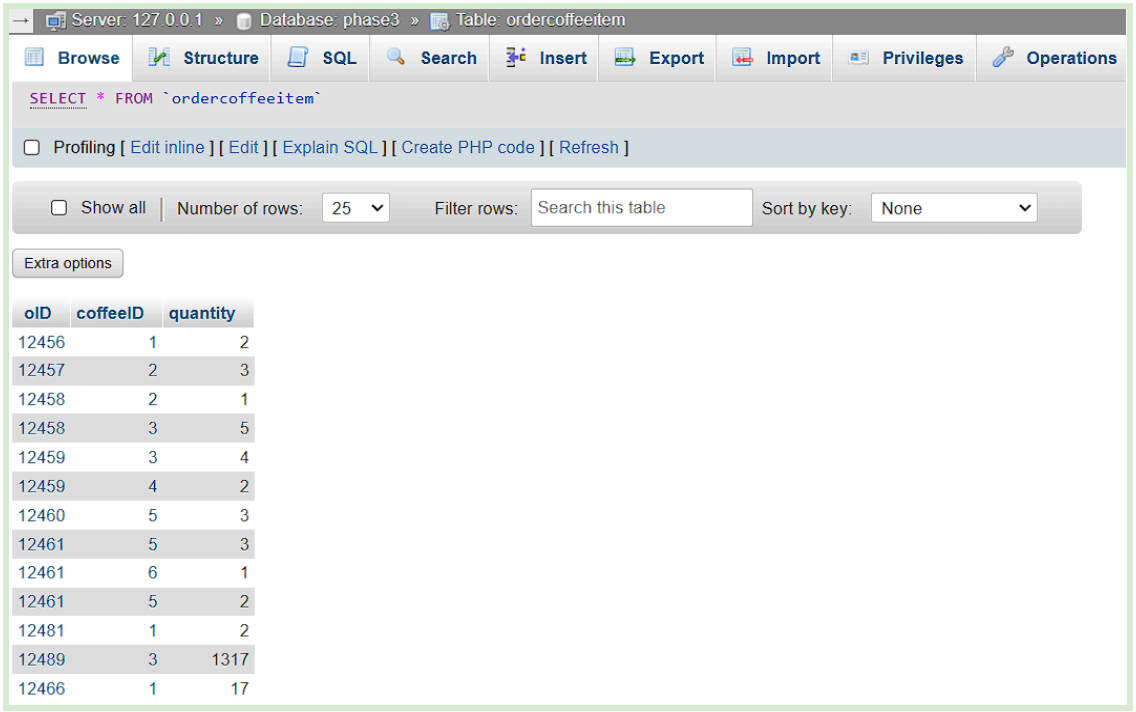
//Before Insertion
//phpMyAdmin displays the initial state of all relevant tables before the new order insertion.

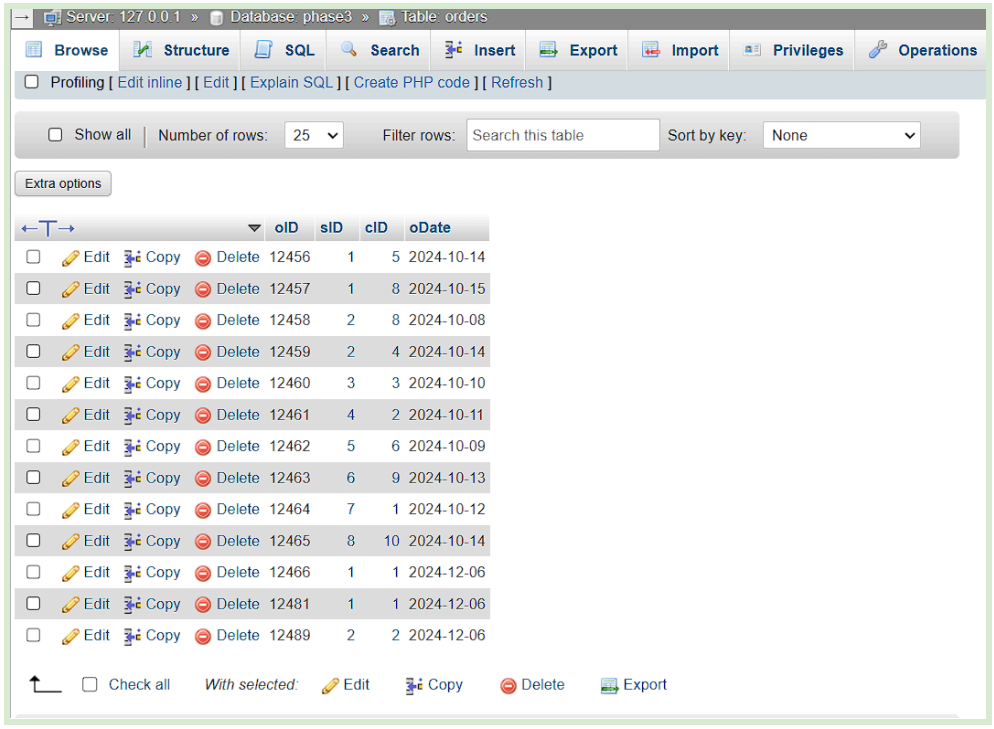


oID	coffeeID	quantity
12456	1	2
12457	2	3
12458	2	1
12458	3	5
12459	3	4
12459	4	2
12460	5	3
12461	5	3
12461	6	1
12461	5	2
12481	1	2
12489	3	1317



// After Insertion
// phpMyAdmin confirms successful insertion of the new order with all specified details in the database tables.




Server: 127.0.0.1 » Database: phase3 » Table: orders
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Number of rows: 25
Filter rows: Search this table
Sort by key: None
Extra options
oID sID cID oDate
Edit Copy Delete 12456 1 5 2024-10-14
Edit Copy Delete 12457 1 8 2024-10-15
Edit Copy Delete 12458 2 8 2024-10-08
Edit Copy Delete 12459 2 4 2024-10-14
Edit Copy Delete 12460 3 3 2024-10-10
Edit Copy Delete 12461 4 2 2024-10-11
Edit Copy Delete 12462 5 6 2024-10-09
Edit Copy Delete 12463 6 9 2024-10-13
Edit Copy Delete 12464 7 1 2024-10-12
Edit Copy Delete 12465 8 10 2024-10-14
Edit Copy Delete 12466 1 1 2024-12-06
Edit Copy Delete 12481 1 1 2024-12-06
Edit Copy Delete 12489 2 2 2024-12-06
Check all
With selected:
Edit
Copy
Delete
Export

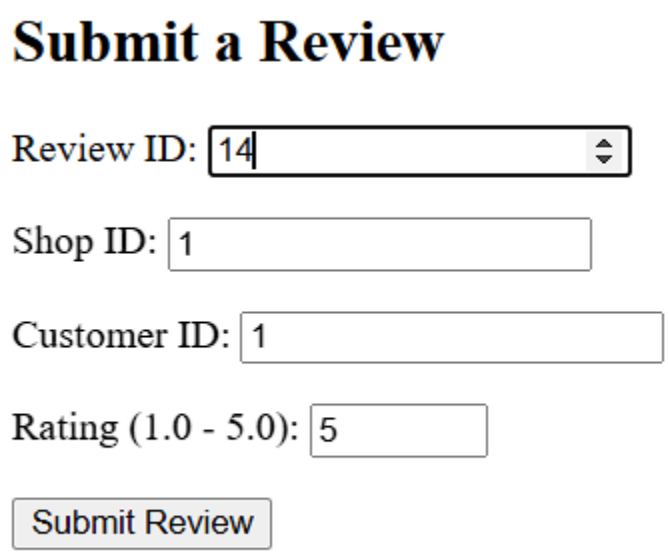
Emre Kaan Usta

// HTML file

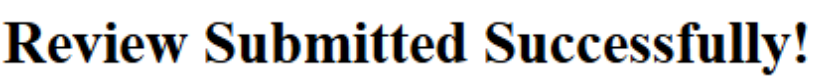
```
<!DOCTYPE html>
<html>
<head>
   <title>Submit Coffee Shop Review</title>
</head>
<body>
   <h2>Submit a Review</h2>
   <form action="add_review.php" method="POST">
      <p><label>Review ID:</label>
         <input type="number" name="rID" required></p>
      <p><label>Shop ID:</label>
         <input type="number" name="sID" required></p>
      <p><label>Customer ID:</label>
         <input type="number" name="cID" required></p>
      <p><label>Rating (1.0 - 5.0):</label>
         <input type="number" step="0.1" min="1.0" max="5.0" name="rating" required></p>
      <button type="submit">Submit Review</button>
   </form>
</body>
</html>
```

// Web Data Insertion Page

A web page to enter attributes of the stored procedure. After clicking submit button, it takes us to add_review.php where review is submitted and both review table and shop's average review is updated accordingly.



After submit button, confirmation message is displayed if the review ID is unique and successfully submission is made.



// Database Before Insertion

As you can see here, review with ID 14 does not exist. Also average review is 4.36 for "Kahve Dünyası" with shop ID = 1. Review table with arbitrary rating IDs are given below before insertion.

	rID	sID	cID	rating
Edit Copy Delete	1	1	1	4.5
Edit Copy Delete	2	2	2	5
Edit Copy Delete	3	3	3	3.7
Edit Copy Delete	4	1	4	2.8
Edit Copy Delete	5	4	5	4.2
Edit Copy Delete	6	2	6	3
Edit Copy Delete	7	5	7	4.9
Edit Copy Delete	8	3	8	1.5
Edit Copy Delete	9	6	9	4.1
Edit Copy Delete	10	7	10	3.1
Edit Copy Delete	11	1	2	5
Edit Copy Delete	12	1	3	4.5
Edit Copy Delete	13	1	1	5

Also the coffeshop table is given below. This table shows the average reviews of the shops. See that "Kahve Dünyası" has a 4.36 average review before the insertion.

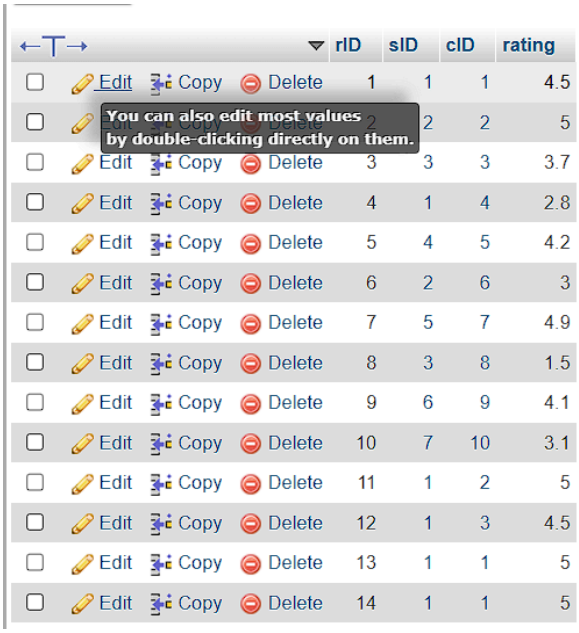
Server: 127.0.0.1 » Database: phase3 » Table: coffeshop

Browse Structure SQL Search Insert Export Import

	sID	sName	sLocation	sPhone	average_review
Edit Copy Delete	1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.36
Edit Copy Delete	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99
Edit Copy Delete	3	Punto	İstanbul, Nişantaşı	555-2003	1.31
Edit Copy Delete	4	MOC	İstanbul, Taksim	555-2004	2.36
Edit Copy Delete	5	Kahve Durağı	Ankara, Çankaya	555-2005	3.65
Edit Copy Delete	6	Cafe Nero	İzmir, Alsancak	555-2006	4.56
Edit Copy Delete	7	V60 Coffee	Bursa, Osmangazi	555-2007	2.43
Edit Copy Delete	8	Cafe Gusto	Antalya, Kaleiçi	555-2008	3.76
Edit Copy Delete	9	Kahve Molası	Konya, Selçuklu	555-2009	2.69
Edit Copy Delete	10	Çaycı	Trabzon, Ortahisar	555-2010	1.70

// Database After Insertion

After the insertion according to the values given above screenshots, review with ID = 14 is also added to review table at the bottom. It's rating = 5 and for the shop with ID = 1 which is "Kahve Dünyası".



	rID	sID	cID	rating
Edit Copy Delete	1	1	1	4.5
Edit Copy Delete	2	2	2	5
Edit Copy Delete	3	3	3	3.7
Edit Copy Delete	4	1	4	2.8
Edit Copy Delete	5	4	5	4.2
Edit Copy Delete	6	2	6	3
Edit Copy Delete	7	5	7	4.9
Edit Copy Delete	8	3	8	1.5
Edit Copy Delete	9	6	9	4.1
Edit Copy Delete	10	7	10	3.1
Edit Copy Delete	11	1	2	5
Edit Copy Delete	12	1	3	4.5
Edit Copy Delete	13	1	1	5
Edit Copy Delete	14	1	1	5

Additionally, as you can see below, average review of shop with ID = 1 (Kahve Dünyası) is 4.47 which is updated from 4.36. We therefore know that this table is also updated with correct values.

	sID	sName	sLocation	sPhone	average_review
Edit Copy Delete	1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.47
Edit Copy Delete	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99
Edit Copy Delete	3	Punto	İstanbul, Nişantaşı	555-2003	1.31
Edit Copy Delete	4	MOC	İstanbul, Taksim	555-2004	2.36
Edit Copy Delete	5	Kahve Durağı	Ankara, Çankaya	555-2005	3.65
Edit Copy Delete	6	Cafe Nero	İzmir, Alsancak	555-2006	4.56
Edit Copy Delete	7	V60 Coffee	Bursa, Osmangazi	555-2007	2.43
Edit Copy Delete	8	Cafe Gusto	Antalya, Kaleiçi	555-2008	3.76
Edit Copy Delete	9	Kahve Molası	Konya, Selçuklu	555-2009	2.69
Edit Copy Delete	10	Çaycı	Trabzon, Ortahisar	555-2010	1.70

Bahar ABİT:

HTML file (index3.html)

```
<!DOCTYPE html>
<html>
<head>
   <title>Place Coffee Item</title>
</head>
<body>
   <h2>Add a Coffee Item to a Menu</h2>
   <form action="process2.php" method="POST">
      <p>
         <label>Menu ID:</label>
         <input type="number" name="menuID" required>
      </p>
      <p>
         <label>Coffee ID:</label>
         <input type="number" name="coffeeID" required>
      </p>
      <button type="submit">Add Coffee Item</button>
   </form>
</body>
</html>
```

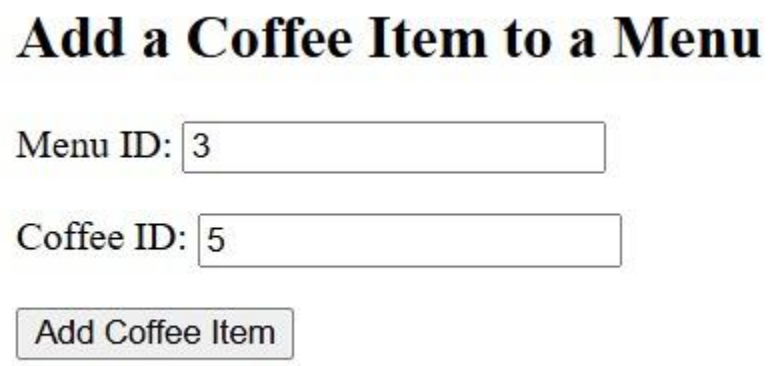
process2.php

```
<?php
include 'db.php'; //
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // Receive data from the form.
   $menuID = $_POST['menuID'];
   $coffeeID = $_POST['coffeeID'];

   // Insert data into the MenuCoffeeItem table.

   $stmt = $conn->prepare("INSERT INTO MenuCoffeeItem (menuID, coffeeID) VALUES (?,
?)");
   $stmt->bind_param("ii", $menuID, $coffeeID);

   if ($stmt->execute()) {
      echo "<p>Yeni kahve öğesi başarıyla eklendi!</p>";
```

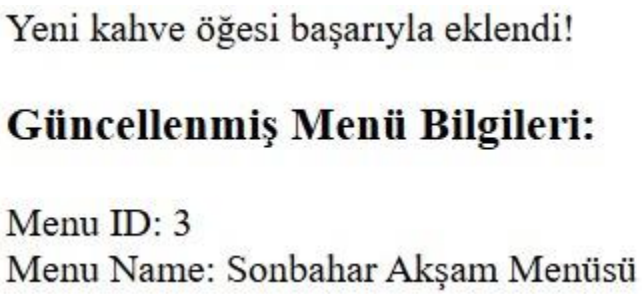
```
        // Retrieve the updated information from the MENU table

        $result = $conn->query("SELECT * FROM MENU WHERE menuID = $menuID");
        if ($result && $result->num_rows > 0) {
            $menu = $result->fetch_assoc();
            echo "<h3>Güncellenmiş Menü Bilgileri:</h3>";
            echo "Menu ID: " . $menu['menuID'] . "<br>";
            echo "Menu Name: " . $menu['mName'] . "<br>";
            echo "Item Count: " . $menu['item_count'] . "<br>";
        }

        // Call the stored procedure
        $call = $conn->prepare("CALL GetMenuDetails(?)");
        $call->bind_param("i", $menuID);
        if ($call->execute()) {
            // Process results using a loop

            do {
                if ($result = $conn->store_result()) {
                    echo "<h3>Menü Detayları:</h3>";
                    while ($row = $result->fetch_assoc()) {
                        echo "Menu Name: " . $row['mName'] . "<br>";
                        echo "Item Count: " . $row['item_count'] . "<br>";
                    }
                    $result->free(); // // Clear the result set

                }
            } while ($conn->more_results() && $conn->next_result());
        }
    } else {
        echo "Hata: " . $stmt->error;
    }

    $stmt->close();
    $conn->close();
}
?>
```

Web Data Insertion Page:

This is a web page designed to input the necessary attributes for adding a coffee item to a menu. After clicking the "Add Coffee Item" button, the form redirects to **process2.php**, where the new coffee item is added to the specified menu. Upon submission, the MenuCoffeeItem table is updated with the new entry, and the item_count column in the MENU table is automatically updated using the defined trigger and stored procedure.



This page verifies that a new coffee item has been successfully added to the menu. It shows the updated menu details, including the Menu ID and Menu Name, highlighting the recent update.



Database Before Insertion:

This table displays the current associations between `menuID` and `coffeeID` in the `MenuCoffeeItem` table prior to any new entries being added. Each row signifies a specific relationship where a menu (`menuID`) contains a coffee item (`coffeeID`).

Purpose of the MenuCoffeeItem Table:

This table serves as a junction between the `MENU` and `COFFEEITEMS` tables, linking menus to their corresponding coffee items. It ensures a many-to-many relationship, allowing multiple coffee items to be associated with a single menu, and vice versa.

Preparation for Insertion:

Before adding a new coffee item, the database is in its initial state. Adding a new entry to this table will activate the trigger, updating the `item_count` field in the `MENU` table, ensuring consistency between linked records.

menuID	coffeeID
1	1
2	1
2	2
3	3
3	4
4	4
5	5
6	6
4	7
7	8
1	10
1	16
1	11
3	6

This table is critical for understanding how the database evolves after the execution of insertion operations and validates the effectiveness of triggers and stored procedures in maintaining data consistency.

Database After Insertion:

The table represents the updated state of the MenuCoffeeItem table following the insertion of a new coffee item. It highlights the changes made to the relationships between menus (menuID) and coffee items (coffeeID) in the database.

A new row has been added with menuID = 3 and coffeeID = 5, establishing a new relationship between the menu and the coffee item.

The data reflects accurate updates, ensuring the consistency of the MenuCoffeeItem table. Existing associations between menus and coffee items remain intact, with no unintended modifications.

The insertion was designed to link an additional coffee item to menuID = 3.
This process demonstrates the functionality of the trigger, which dynamically updates the item_count field in the MENU table, ensuring synchronization.

menuID	coffeeID
1	1
2	1
2	2
3	3
3	4
4	4
5	5
6	6
4	7
7	8
1	10
1	16
1	11
3	6
3	5

This updated view of the `MenuCoffeeItem` table serves as verification of the successful operation of the trigger and the integrity of the database after the insertion.

Hüseyin Eren YILDIZ

HTML File (processBartender.php)

The HTML form is designed to efficiently capture all necessary information required to add a new bartender to the database. It provides a user-friendly interface with validated input fields for essential details. This form simplifies the process of adding a bartender, ensuring accurate data entry and smooth integration with the backend system.

```
<!DOCTYPE html>
<html>
<head>
    <title>Add Bartender</title>
</head>
<body>
    <h2>Add Bartender</h2>
    <form action="/phase2/processBartender.php" method="POST">
        <p>
            <label>Bartender ID:</label>
            <input type="number" name="bartender_id" required>
        </p>
        <p>
            <label>Shop ID:</label>
            <input type="number" name="shop_id" required>
        </p>
        <p>
            <label>Bartender Name:</label>
            <input type="text" name="bartender_name" required>
        </p>
        <button type="submit">Add Bartender</button>
    </form>
</body>
</html>
```

//Database Connection (db.php)

This component establishes a secure connection to the MySQL database, ensuring reliability and incorporating effective error handling to manage any connection issues gracefully.

```
<?php
$servername = "localhost:3307";
$username = "root";
$password = "";
$dbname = "phase2";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

echo "Database connection successful!";
?>
```

// Bartender Processing (processBartender.php)

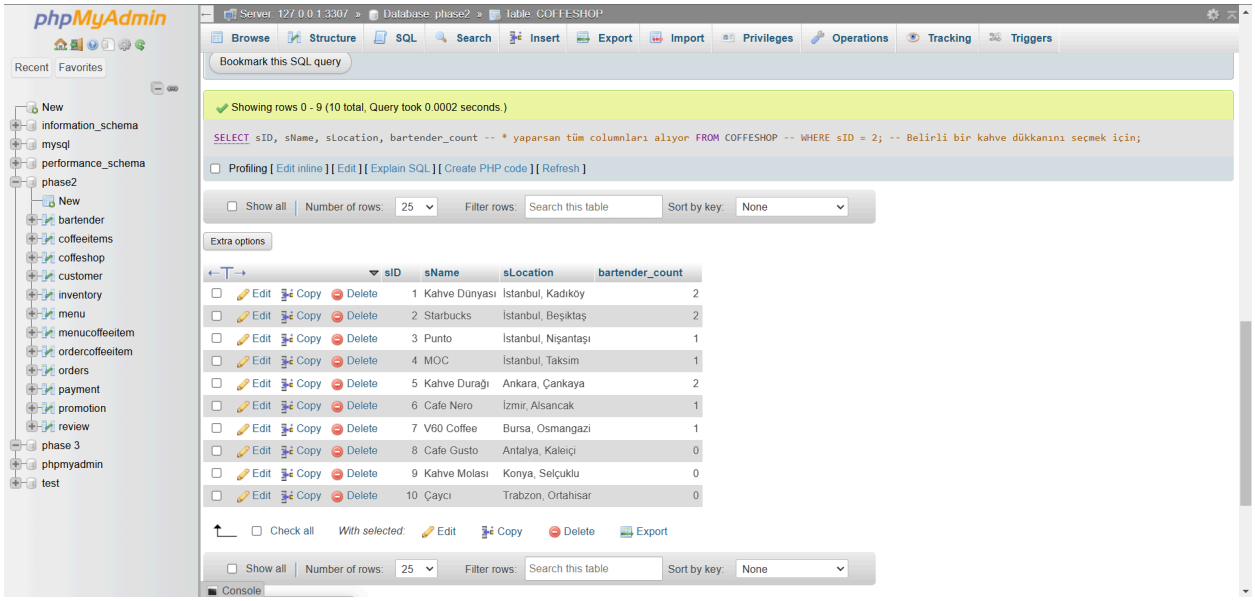
The bartender processing script handles the entire workflow for adding a new bartender. It starts by receiving data from the HTML form, executes the stored procedure to add the bartender and update the count, and finally displays a confirmation message. The script ensures seamless operation and proper error handling.

```
<?php
include 'db.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $bartender_id = $_POST['bartender_id'];
    $shop_id = $_POST['shop_id'];
    $bartender_name = $_POST['bartender_name'];

    $stmt = $conn->prepare("CALL AddBartenderAndUpdateCount(?, ?, ?)");
    $stmt->bind_param("iis", $bartender_id, $shop_id, $bartender_name);
```

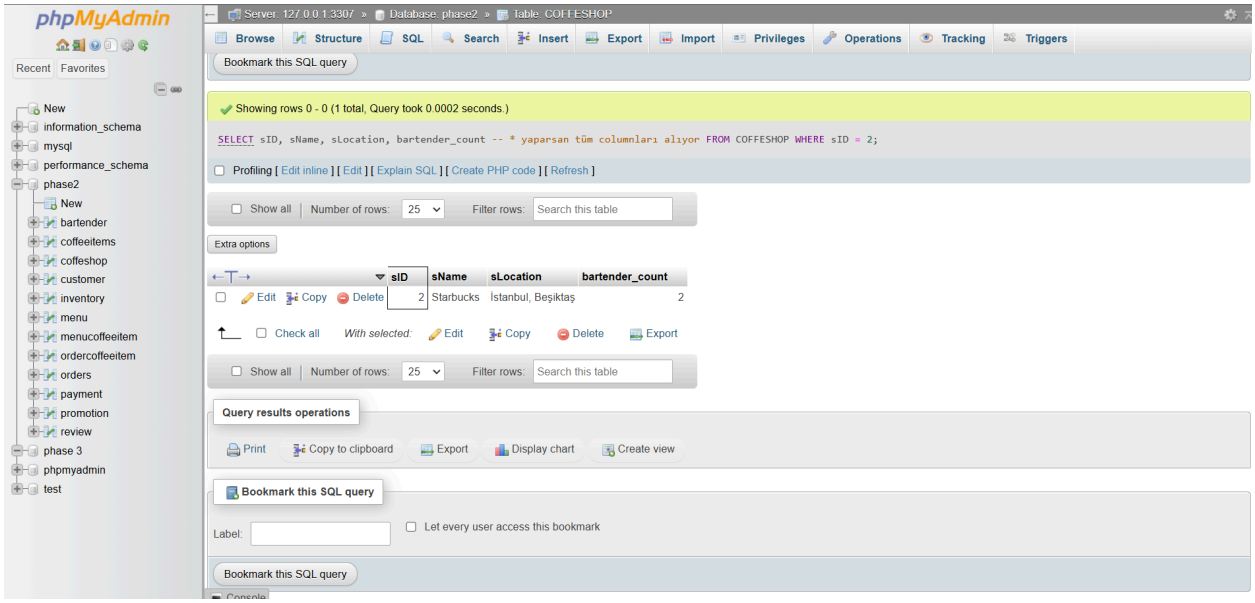
```
    if ($stmt->execute()) {
        echo "<h2>Bartender Added Successfully!</h2>";
        echo "Bartender ID: " . htmlspecialchars($bartender_id) . "<br>";
        echo "Shop ID: " . htmlspecialchars($shop_id) . "<br>";
        echo "Bartender Name: " . htmlspecialchars($bartender_name) . "<br>";
    } else {
        echo "<h2>Error Adding Bartender</h2>";
        echo "Error: " . htmlspecialchars($stmt->error);
    }

    $stmt->close();
    $conn->close();
}
?>
```

BEFORE WEB DATA INSERTION

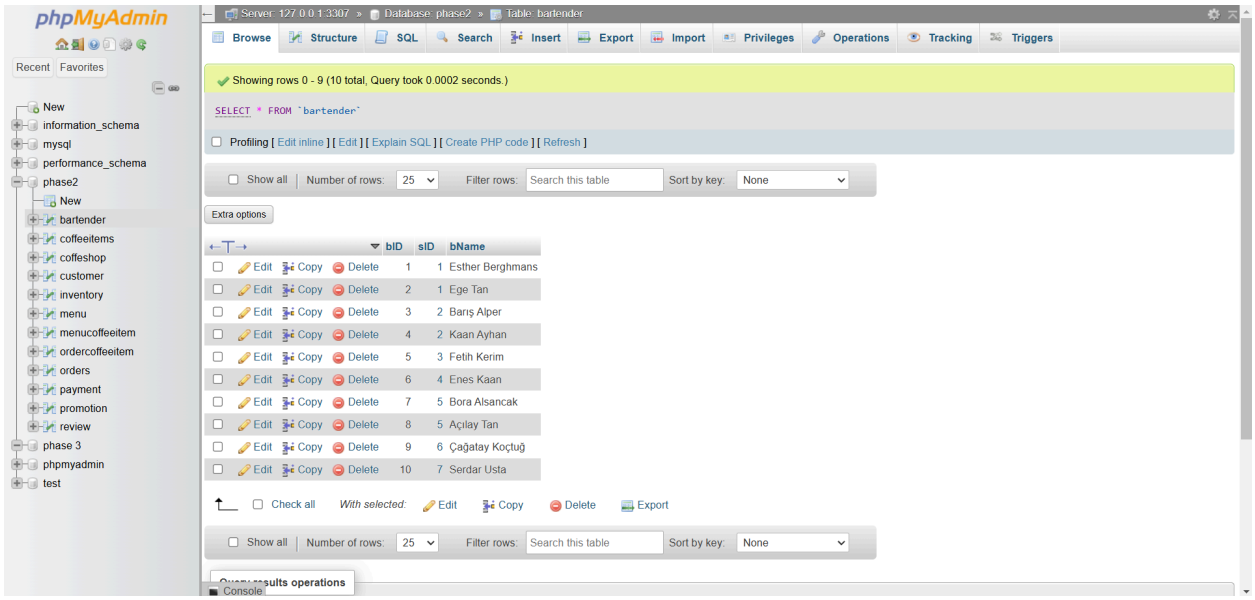
This screenshot shows all coffee shops and their respective bartender counts before insertion.



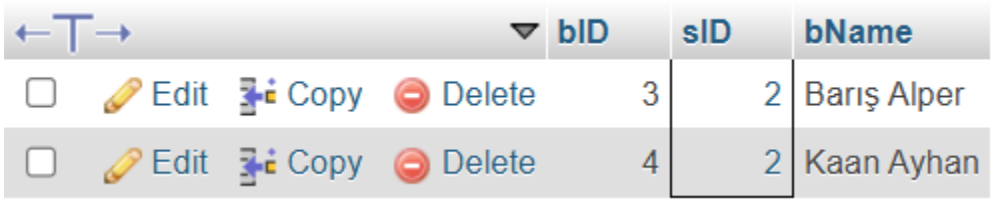
This screenshot shows a specific coffee shop (sID = 2) with respective bartender counts before input insertion.



This screenshot shows all bartenders (with bID and sID) and their respective names.



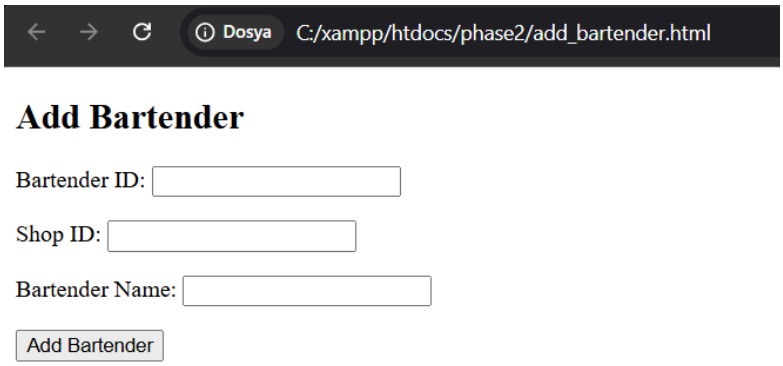
This table shows all bartenders working in a specific coffee shop. It displays all bartenders' names working at Starbucks (sID=2).

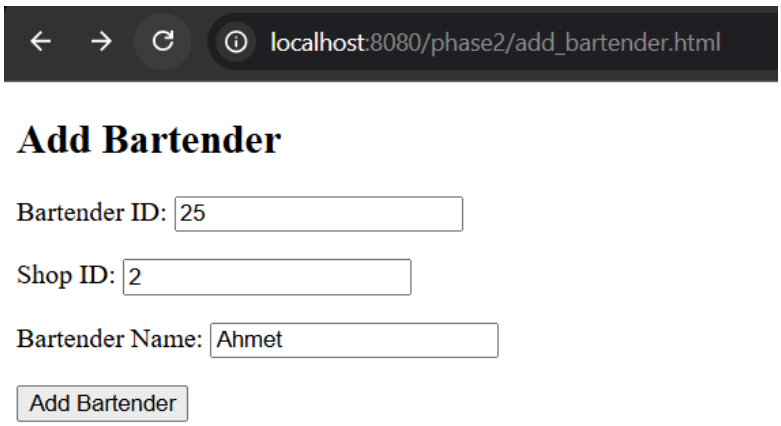


	bID	sID	bName
Edit Copy Delete	3	2	Barış Alper
Edit Copy Delete	4	2	Kaan Ayhan

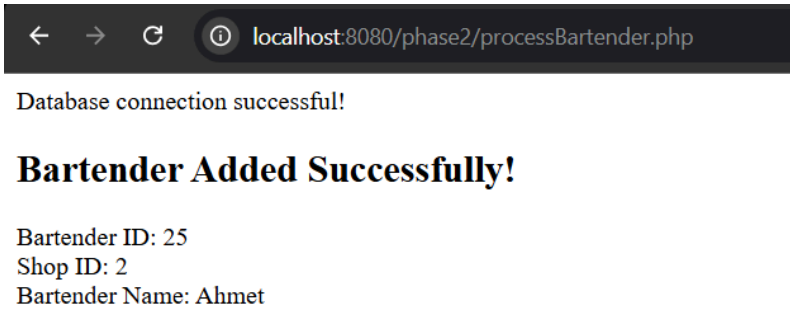
// WEB DATA INSERTION PAGE

This web page provides a simple interface to input details for adding a new bartender. Once the required fields are filled and the "Add Bartender" button is clicked, the form submits the data to processBartender.php. There, the bartender is added to the database, and the corresponding coffee shop's bartender count is automatically updated to reflect the new addition.



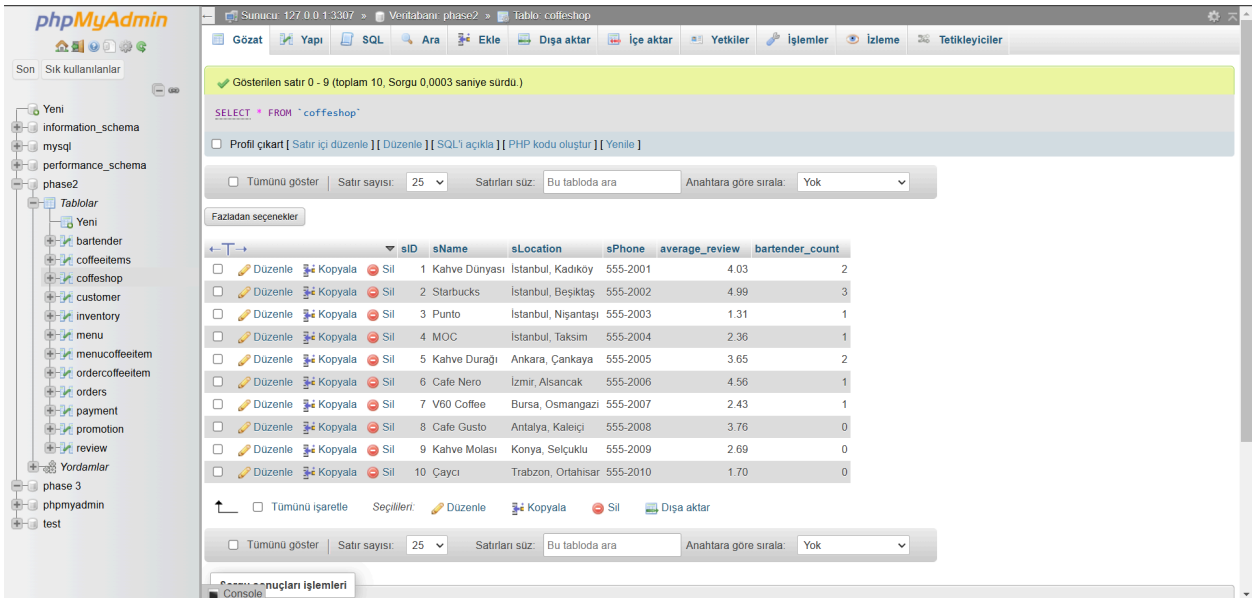


This confirmation page displays a detailed summary of the successful bartender addition. It confirms that the database connection was established and shows the submitted information, including the Bartender ID, Shop ID, and Bartender Name, ensuring that the operation was completed successfully.



AFTER WEB DATA INSERTION

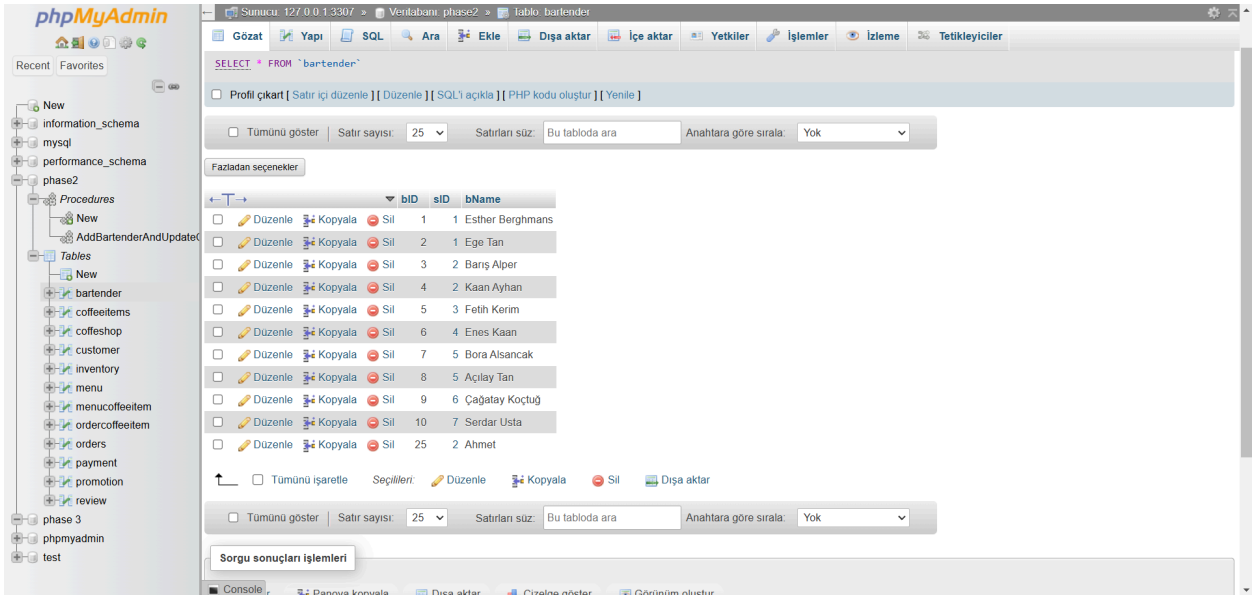
This screenshot shows all coffee shops and their respective bartender counts after insertion. The bartender count for the coffee shop with the specified shopID entered by the user in the Web Data Form has increased. (In this example sID = 2 → Starbucks) Number of bartenders increased from 2 to 3.

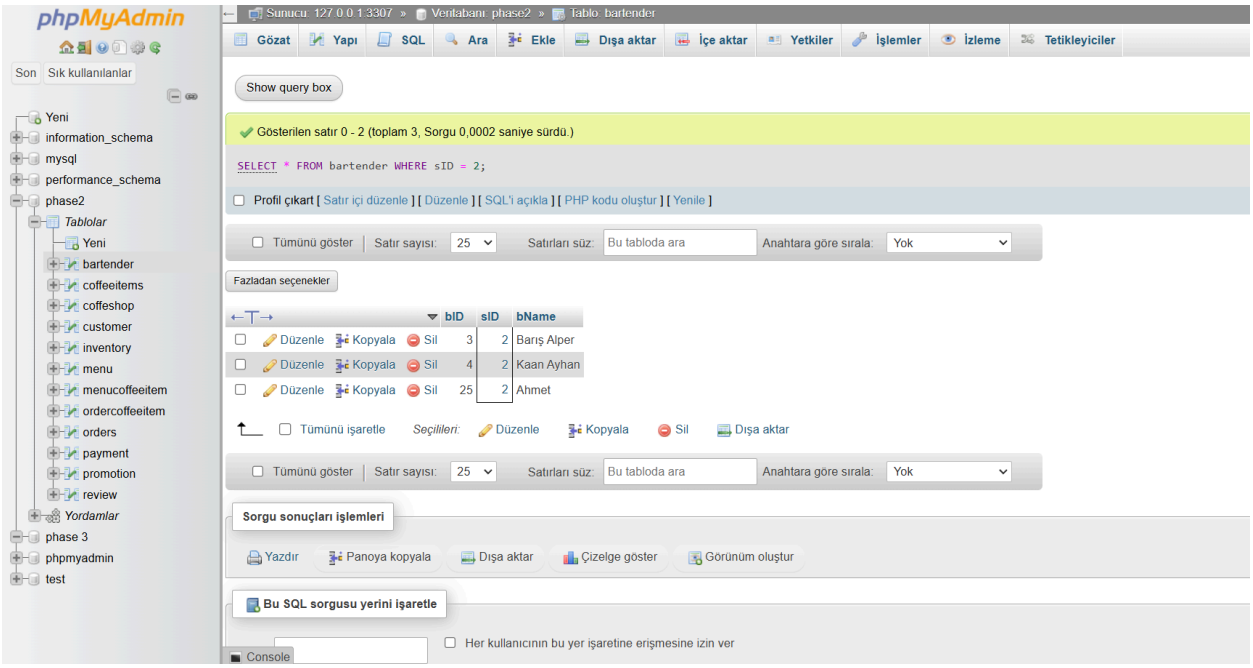


This screenshot shows a specific coffee shop (sID = 2) with respective bartender counts before input insertion. At the start, there were 2 bartenders but after the provided input, we successfully increased the number to 3.

	sID	sName	sLocation	sPhone	average_review	bartender_count
Düzenle Kopyala Sil	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99	3

The newly added bartender is now visible in all bartenders and specific coffee shop bartender lists (sID: 2).





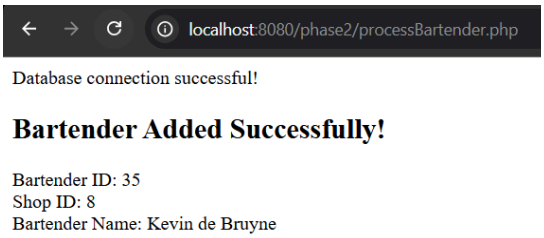
MORE EXAMPLE:

In the image on the left, the coffee shop with sID=8 has a bartender_count of 0, and in the image on the right, the list of all bartenders before adding the new bartender is displayed.

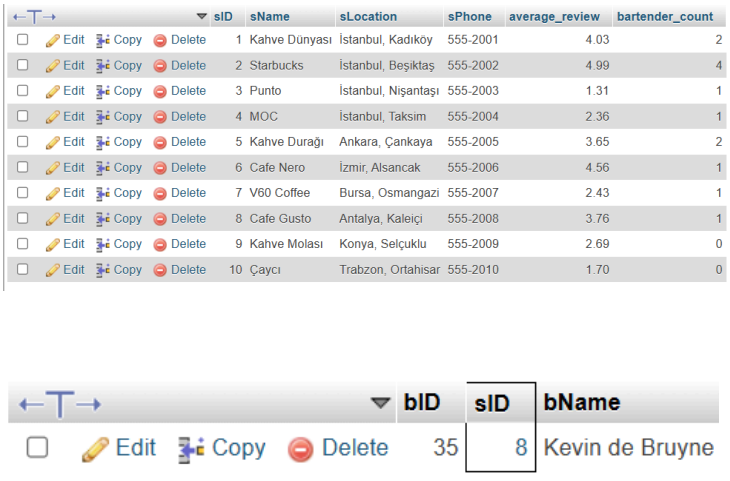
	sID	sName	sLocation	sPhone	average_review	bartender_count
Edit Copy Delete	1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.03	2
Edit Copy Delete	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99	4
Edit Copy Delete	3	Punto	İstanbul, Nişantaşı	555-2003	1.31	1
Edit Copy Delete	4	MOC	İstanbul, Taksim	555-2004	2.36	1
Edit Copy Delete	5	Kahve Durağı	Ankara, Çankaya	555-2005	3.65	2
Edit Copy Delete	6	Cafe Nero	İzmir, Alsancak	555-2006	4.56	1
Edit Copy Delete	7	V60 Coffee	Bursa, Osmangazi	555-2007	2.43	1
Edit Copy Delete	8	Cafe Gusto	Antalya, Kaleiçi	555-2008	3.76	0
Edit Copy Delete	9	Kahve Molası	Konya, Selçuklu	555-2009	2.69	0
Edit Copy Delete	10	Çaycı	Trabzon, Ortahisar	555-2010	1.70	0

	bID	sID	bName
Edit Copy Delete	1	1	Esther Berghmans
Edit Copy Delete	2	1	Ege Tan
Edit Copy Delete	3	2	Barış Alper
Edit Copy Delete	4	2	Kaan Ayhan
Edit Copy Delete	5	3	Fetih Kerim
Edit Copy Delete	6	4	Enes Kaan
Edit Copy Delete	7	5	Bora Alsancak
Edit Copy Delete	8	5	Açılay Tan
Edit Copy Delete	9	6	Çağatay Koçtuğ
Edit Copy Delete	10	7	Serdar Usta
Edit Copy Delete	25	2	Ahmet
Edit Copy Delete	30	2	Lionel Messi

Successful Web Data Insertion



After the successful addition of a bartender, it is observed in the left image that the bartender_count for the coffee shop with sID=8 has increased, and in the right image, the newly added bartender (bID: 35, bName: Kevin de Bruyne) is included in the list of all bartenders.



	sID	sName	sLocation	sPhone	average_review	bartender_count
Edit Copy Delete	1	Kahve Dünyası	İstanbul, Kadıköy	555-2001	4.03	2
Edit Copy Delete	2	Starbucks	İstanbul, Beşiktaş	555-2002	4.99	4
Edit Copy Delete	3	Punto	İstanbul, Nişantaşı	555-2003	1.31	1
Edit Copy Delete	4	MOC	İstanbul, Taksim	555-2004	2.36	1
Edit Copy Delete	5	Kahve Durağı	Ankara, Çankaya	555-2005	3.65	2
Edit Copy Delete	6	Cafe Nero	İzmir, Alsancak	555-2006	4.56	1
Edit Copy Delete	7	V60 Coffee	Bursa, Osmangazi	555-2007	2.43	1
Edit Copy Delete	8	Cafe Gusto	Antalya, Kaleiçi	555-2008	3.76	1
Edit Copy Delete	9	Kahve Molası	Konya, Selçuklu	555-2009	2.69	0
Edit Copy Delete	10	Çaycı	Trabzon, Ortahisar	555-2010	1.70	0

	bID	sID	bName
Edit Copy Delete	35	8	Kevin de Bruyne

	bID	sID	bName
Edit Copy Delete	1	1	Esther Berghmans
Edit Copy Delete	2	1	Ege Tan
Edit Copy Delete	3	2	Barış Alper
Edit Copy Delete	4	2	Kaan Ayhan
Edit Copy Delete	5	3	Fetih Kerim
Edit Copy Delete	6	4	Enes Kaan
Edit Copy Delete	7	5	Bora Alsancak
Edit Copy Delete	8	5	Açılay Tan
Edit Copy Delete	9	6	Çağatay Koçtuğ
Edit Copy Delete	10	7	Serdar Usta
Edit Copy Delete	25	2	Ahmet
Edit Copy Delete	30	2	Lionel Messi
Edit Copy Delete	35	8	Kevin de Bruyne