**MySQL: Stored Functions and Procedures**

Stephan Peters

Colorado State University Global

23WC-ITS410-2: Database Management

Jamia Mills

25 February 2024

## MySQL: Stored Functions and Procedures

Full resolution images of the figures shown may be viewed on my public GitHub repository here: <https://github.com/speters33w/CSUGlobal_ITS410/blob/main/CriticalThinking6/Module_6_Critical_Thinking.md>, or with CSU Global Login via Google Drive here:

## Storing and Calling a Procedure

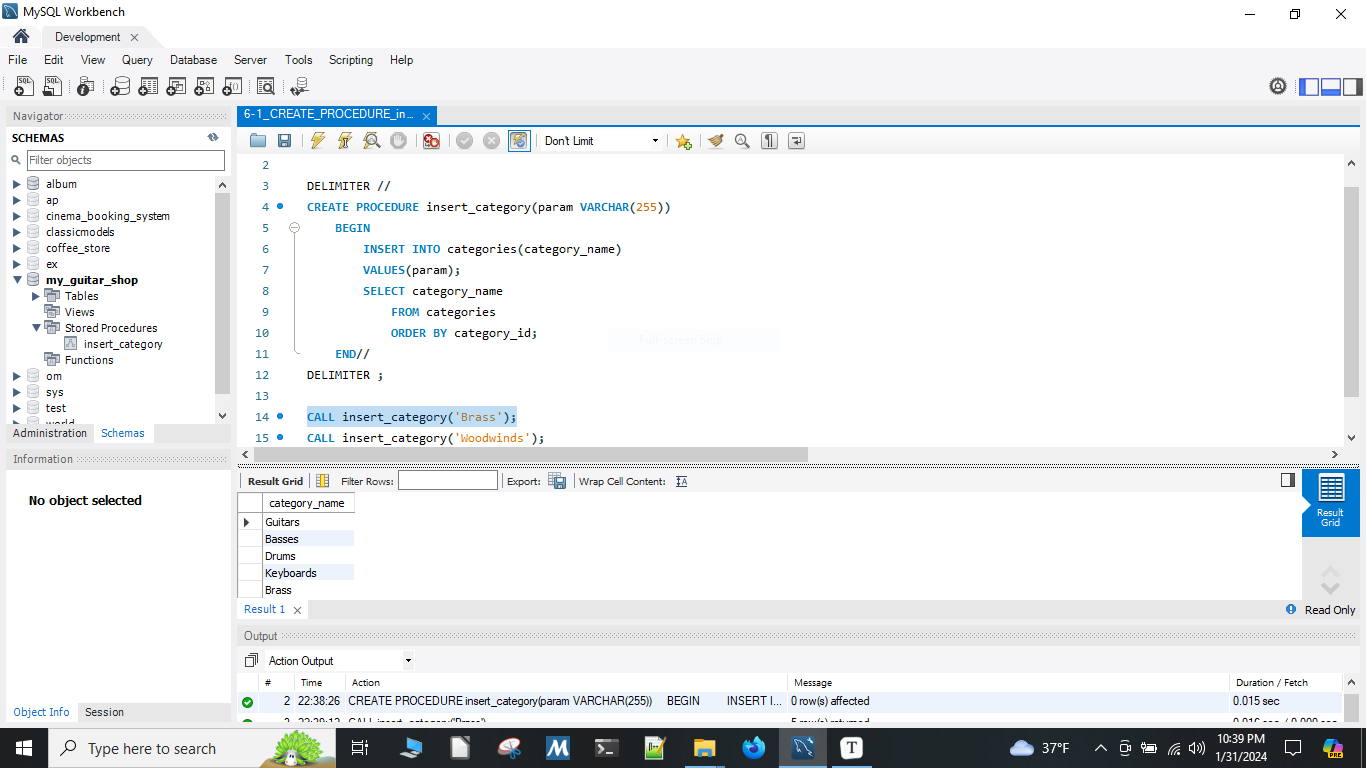
This writes a stored procedure named insert\_category that adds a new row to the category table using a parameter for the category name. The next query (Script 1a / Figure 1a) calls the function and adds “Brass” as a category. The following (Script 1b / Figure 1b) calls the function and adds “Woodwinds” as a category. The function uses a SELECT statement to verify insertion of the new category while running.

Script 1a

|  |
| --- |
| USE my\_guitar\_shop;  DROP PROCEDURE IF EXISTS *insert\_category*;  DELIMITER // CREATE PROCEDURE *insert\_category*(param VARCHAR(255))  BEGIN  INSERT INTO categories(category\_name)  VALUES(param);  SELECT category\_name  FROM categories  ORDER BY category\_id DESC;  END// DELIMITER ;  CALL *insert\_category*('Brass');  SHOW PROCEDURE STATUS WHERE db = 'my\_guitar\_shop'; |

Figure 1a

CREATE PROCEDURE insert\_category

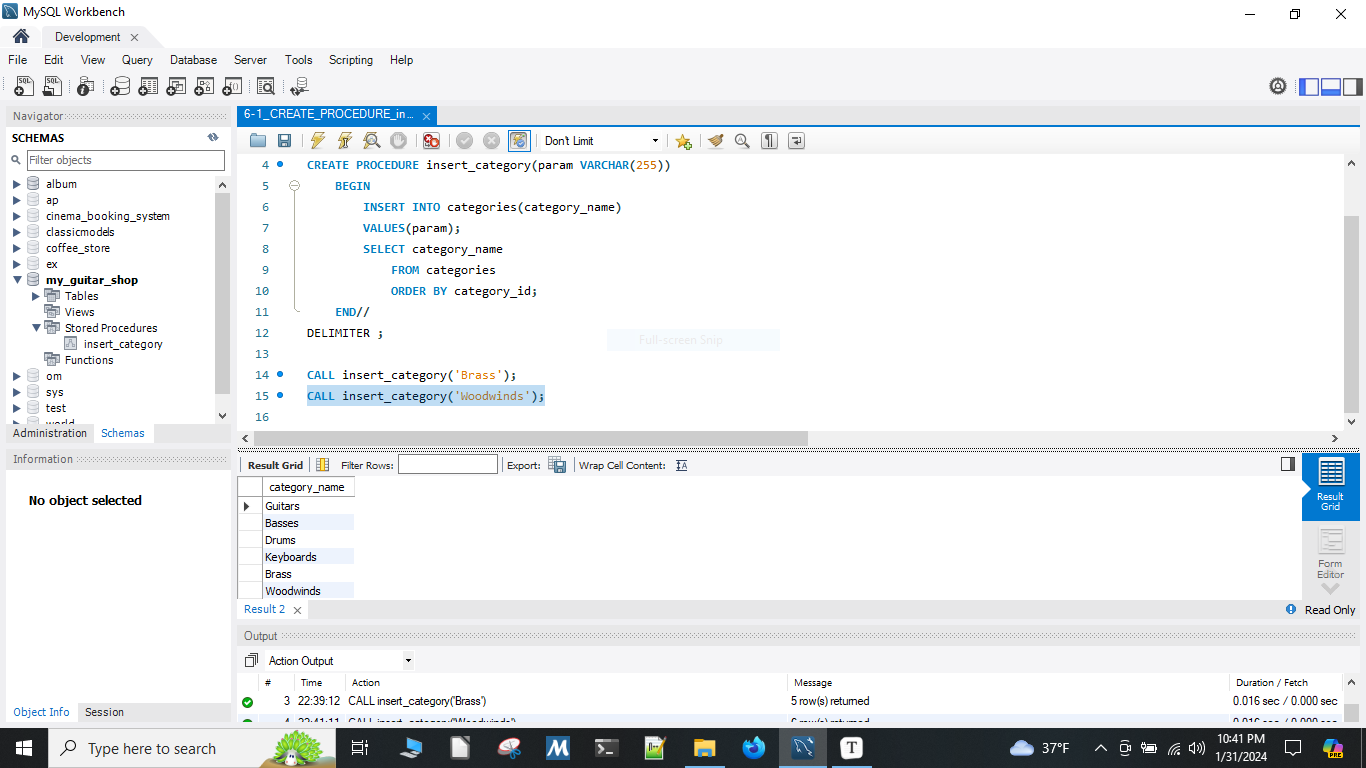
****

Script 1b

|  |
| --- |
| CALL *insert\_category*('Woodwinds'); |

Figure 1b

CALL insert\_category('Woodwinds');

****

## Storing and Calling a Stored Function in MySQL

This function returns the actual price of an order item after the discount amount has been applied. The query returns the order item, the product name, the list price, the discount amount, and the discount price using the stored discount\_price function ordered by order id.

Script 2

|  |
| --- |
| USE my\_guitar\_shop;  SELECT *\** FROM order\_items;  DROP FUNCTION IF EXISTS *discount\_price*;  DELIMITER // CREATE FUNCTION *discount\_price*(order\_item\_id INT) RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN  DECLARE actual\_price DECIMAL(10,2);  SELECT (item\_price - discount\_amount) INTO actual\_price  FROM order\_items WHERE item\_id = order\_item\_id;  RETURN actual\_price; END// DELIMITER ;  SELECT o.item\_id AS order\_item\_id, p.product\_name, o.item\_price,  o.discount\_amount, *discount\_price*(item\_id) AS discount\_price FROM order\_items o JOIN products p USING (product\_id) ORDER BY o.item\_id;  SHOW FUNCTION STATUS WHERE db = 'my\_guitar\_shop'; |

Figure 2a

CREATE FUNCTION discount\_price;

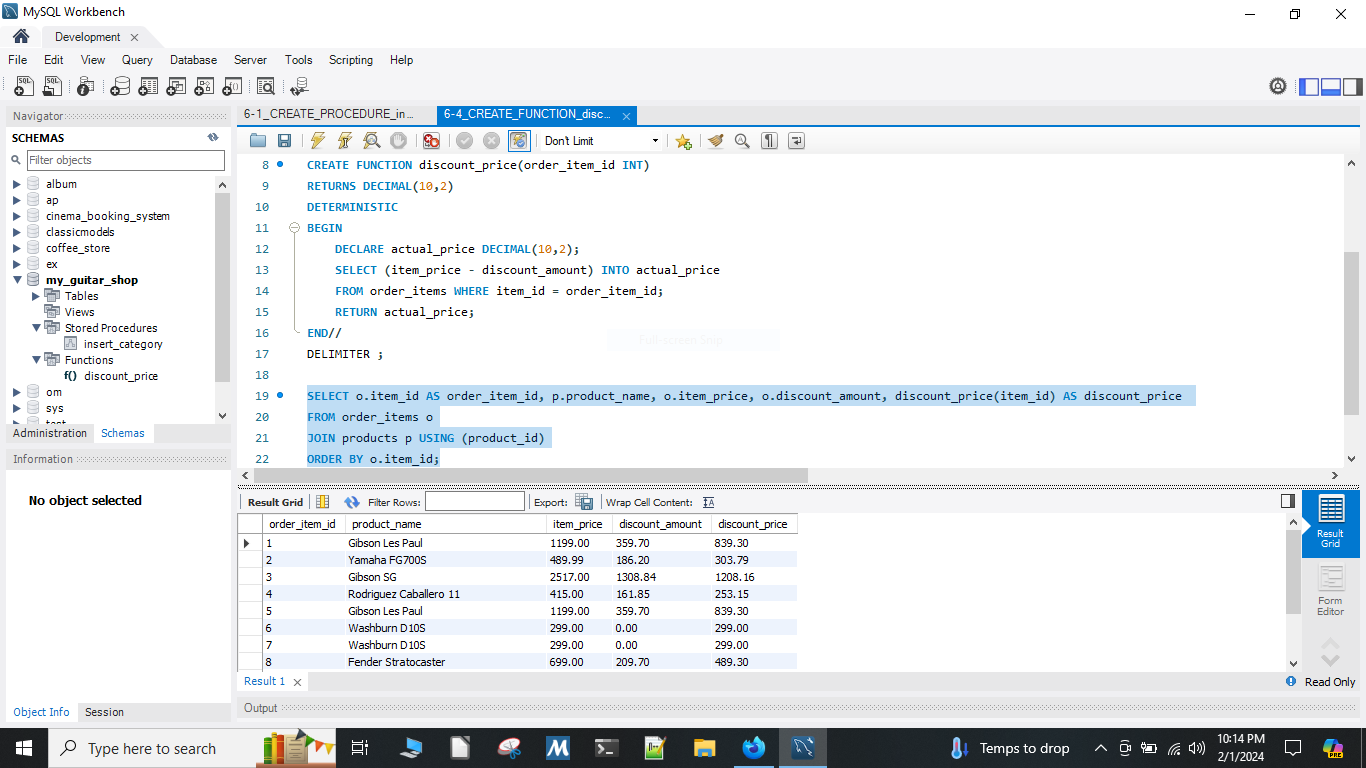
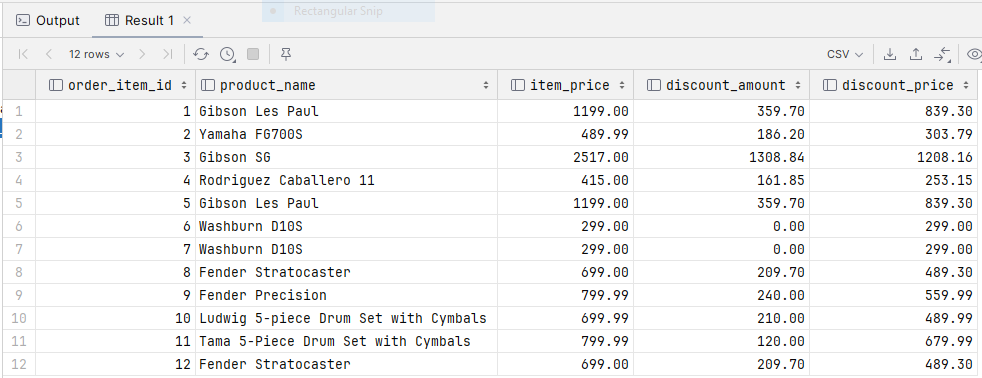
****

Figure 2b

Result of SELECT discount\_price query.

****

## Using sum() in a MySQL Query

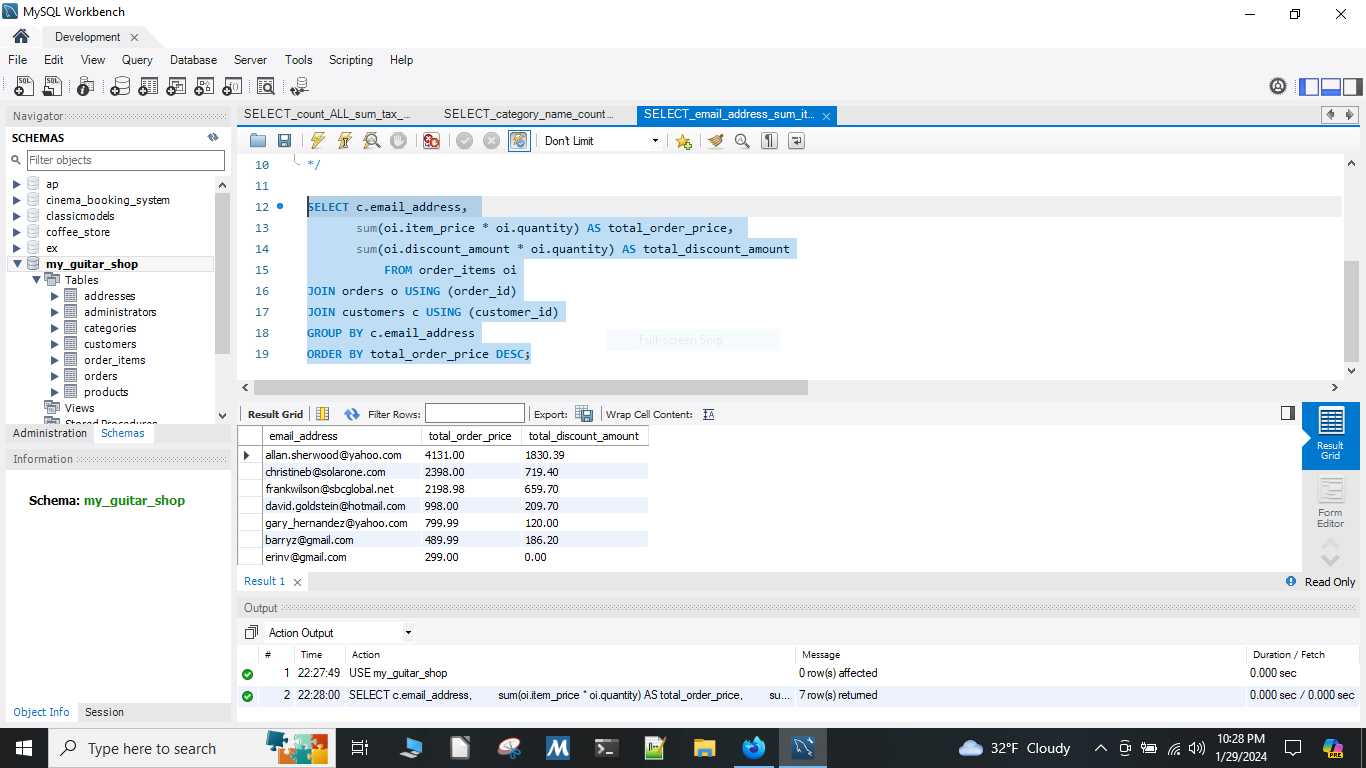
This query returns one row for each customer that has orders with these columns:

* The email address column from the customers table
* The sum of the item price in the order items table multiplied by the quantity in the order items table
* The sum of the discount amount column in the order items table multiplied by the quantity in the order items table

sorted in descending sequence by the item price total for each customer.

Figure 3

SELECT c.email\_address, sum(oi.item\_price \* oi.quantity) AS total\_order\_price, sum(oi.discount\_amount \* oi.quantity) AS total\_discount\_amount FROM order\_items oi JOIN orders o USING (order\_id) JOIN customers c USING (customer\_id) GROUP BY c.email\_address ORDER BY total\_order\_price DESC;

****

## Using sum() and count(DISTINCT) in a MySQL Query

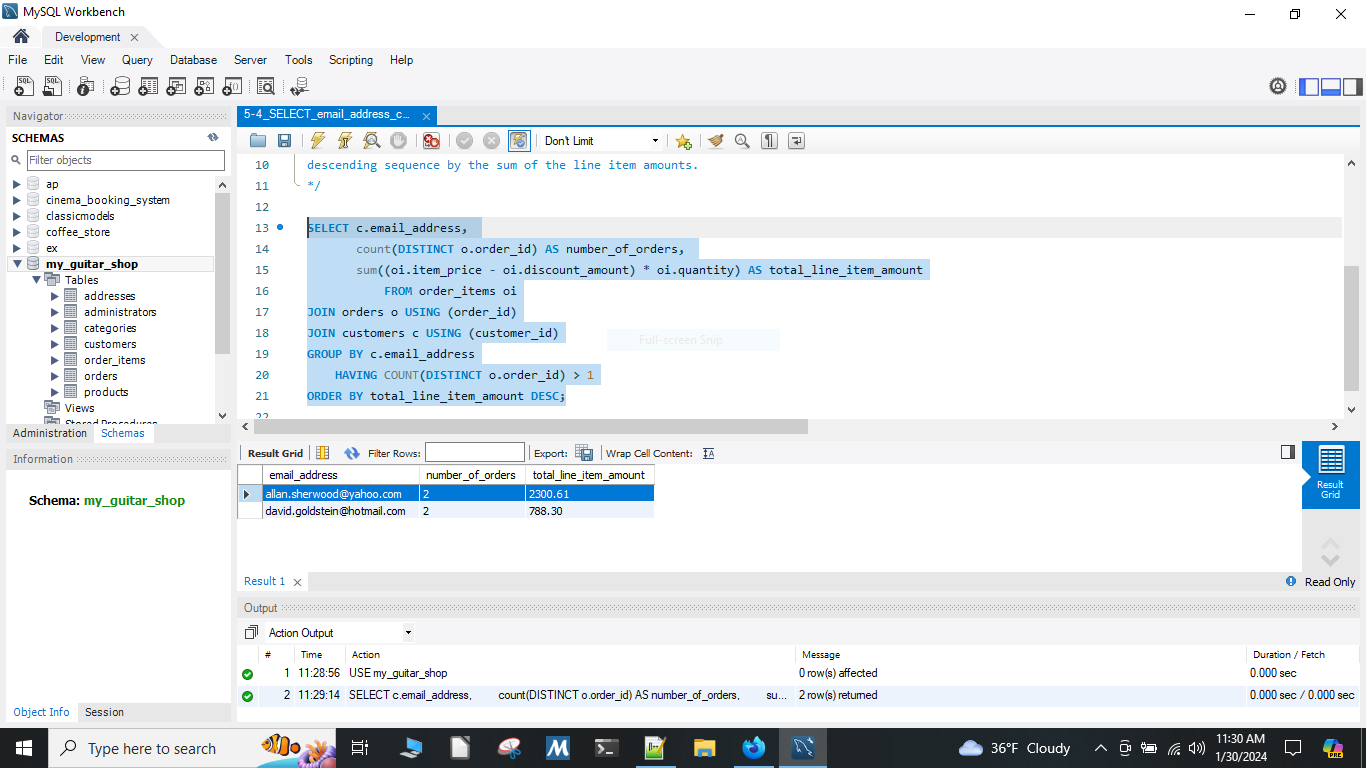
This query returns one row for each customer that has orders with these columns:

* The email address column from the customers table
* A count of the number of orders
* The total amount for each order

It returns only those rows where the customer has more than one order, sorted in descending sequence by the sum of the line-item amounts.

Figure 4

SELECT c.email\_address, count(DISTINCT o.order\_id) AS number\_of\_orders, sum((oi.item\_price - oi.discount\_amount) \* oi.quantity) AS total\_line\_item\_amount FROM order\_items oi JOIN orders o USING (order\_id) JOIN customers c USING (customer\_id) GROUP BY c.email\_address HAVING COUNT(DISTINCT o.order\_id) > 1 ORDER BY total\_line\_item\_amount DESC;

****