

Critical Thinking Assignment 6: Stored Procedures

Alexander Ricciardi

Colorado State University Global

ITS410: Database Management

Dr. Murthy Rallapalli

May 25, 2025

Critical Thinking Assignment 6: Stored Procedures

This documentation is part of the Critical Thinking 6 Assignment from ITS410: Database Management at Colorado State University Global. The documentation provides screenshots showcasing stored procedures using MySQL and the My Guitar Shop database.

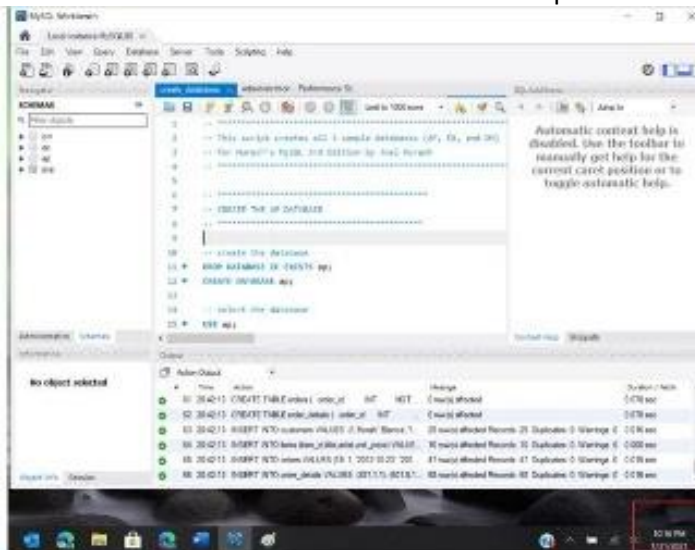
The Assignment Direction:

Stored Procedures

Using the My Guitar Shop database you installed in Module 1, develop the following queries.

1. Write a script that creates and *calls* a stored procedure named `insert_category`.
2. First, code a statement that creates a procedure that adds a new row to the `Categories` table.
 - To do that, this procedure should have one parameter for the category name.
3. Code at least two `CALL` statements that test this procedure. (Note that this table doesn't allow duplicate category names.)
 - Execute the query and take a screenshot of the query and the results.
4. Write a script that creates and calls a stored function named `discount_price` that calculates the discount price of an item in the `Order_Items` table (discount amount subtracted from item price).
 - To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.
 - Execute the query and take a screenshot of the query and the results.
5. Write a script that creates and calls a stored function named `item_total` that calculates the total amount of an item in the `Order_Items` table (discount price multiplied by quantity).
 - To do that, this function should accept one parameter for the item ID, it should use the `discount_price` function that you created in exercise 2, and it should return the value of the total for that item.
 - Execute the query and take a screenshot of the query and the results.

All the screenshots should show current date. Example of screenshot.



Submit your labeled results screenshots in a Word file.

Steps 1 through 3: Write a script that creates and *calls* a stored procedure named `insert_category`.

Figure 1

Assignment Steps 1 through 3

The figure consists of three screenshots of the MySQL Workbench interface, illustrating the steps to create and call a stored procedure.

Top Screenshot: Shows the SQL Editor with the script for creating the `insert_category` stored procedure. The script includes error handling logic using `DECLARE`, `IF`, and `SELECT` statements. The procedure is designed to insert a new category into the `categories` table and return the status message, category ID, MySQL error code, error message, and SQL state code.

Middle Screenshot: Shows the SQL Editor with the script for calling the `insert_category` stored procedure. The script includes a `CALL insert_category('Saxophone');` statement. The Output window shows the results of the execution, including the status message, category ID, MySQL error code, error message, and SQL state code.

Bottom Screenshot: Shows the SQL Editor with the script for calling the `insert_category` stored procedure. The script includes a `CALL insert_category('Guitars');` statement. The Output window shows the results of the execution, including the status message, category ID, MySQL error code, error message, and SQL state code.

Note: The figure illustrates the MySQL Workbench result after performing steps 1 through 3.

Step 4: Write a script that creates and calls a stored function named `discount_price` that calculates the discount price of an item in the `Order_Items` table.

Figure 2
Assignment Steps 4

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view of the database structure, including tables like `addresses`, `administrators`, `categories`, `customers`, `order_items`, `orders`, `products`, `views`, `stored_procedures`, and `functions`. The `discount_price` function is highlighted under the `functions` folder.

The main editor window shows the SQL script for creating and using the `discount_price` function. The script is as follows:

```

1 DROP FUNCTION IF EXISTS discount_price;
2
3 DELIMITER //
4
5 -- Create the stored function named discount_price
6 CREATE FUNCTION discount_price (
7     order_item_id INT -- The ID of the order item
8 )
9 RETURNS DECIMAL(10,2) -- Returns a decimal value
10 DETERMINISTIC -- the function returns the same result for the same input
11 READS SQL DATA -- the function reads data from the database
12
13 BEGIN
14
15     DECLARE discount_price_var DECIMAL(10,2);
16
17     -- Fetches the item_price and discount_amount for the given item_id from the Order_Items table
18     -- Compute (item_price - discount_amount) and returns the discounted price
19     SELECT
20         item_price - discount_amount
21     INTO
22         discount_price_var
23     FROM
24         order_items -- Referencing the Order_Items table from your schema
25     WHERE
26         item_id = order_item_id;
27
28     -- Returns the calculated discount price.
29     -- If the order_item_id is not found in the table,
30     -- the discount_price_var will compute a NULL value, which will then be returned
31     RETURN discount_price_var;
32 END //
33
34 DELIMITER ;
35
36 -- Example of use in query statement
37 -- Computes the discount price for ALL the items in order_item table
38 SELECT
39     item_id,
40     item_price,
41     discount_amount,
42     quantity,
43     discount_price(item_id) AS customer_amount
44 FROM
45     order_items
46

```

The bottom panel shows the 'Result Grid' with 12 rows of data. The columns are `item_id`, `item_price`, `discount_amount`, `quantity`, and `customer_amount`. The data is as follows:

item_id	item_price	discount_amount	quantity	customer_amount
1	1199.00	359.70	1	839.30
2	489.99	186.20	1	303.79
3	2517.00	1308.84	1	1208.16
4	415.00	161.85	1	253.15
5	1199.00	359.70	2	839.30
6	299.00	0.00	1	299.00
7	299.00	0.00	1	299.00
8	699.00	209.70	1	489.30
9	799.99	240.00	1	559.99
10	699.99	210.00	1	489.99
11	799.99	120.00	1	679.99
12	699.00	209.70	1	489.30

The bottom panel also shows the 'Output' section with a table of actions and their results:

#	Time	Action	Message	Duration / Fetch
1	15:15:27	DROP FUNCTION IF EXISTS discount_price	0 row(s) affected, 1 warning(s): 1305 FUNCTION my_guit...	0.000 sec
2	15:15:27	-- Create the stored function named discount_price CREA...	0 row(s) affected	0.000 sec
3	15:15:27	-- Example of use in query statement -- Computes the disco...	12 row(s) returned	0.000 sec / 0.000 sec
4	15:18:00	DROP PROCEDURE 'my_guitar_shop'.insert_category'	0 row(s) affected	0.015 sec

Note: The figure illustrates the MySQL Workbench result after performing steps 4.

Step 5: Write a script that creates and calls a stored function named `item_total` that calculates the total amount of an item in the `Order_Items` table.

Figure 3
Assignment Step 5

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'Schemas' panel with the 'my_guitar_shop' database selected. The main editor window contains the following SQL script:

```

1 DROP FUNCTION IF EXISTS item_total;
2
3 DELIMITER //
4
5 -- Create the stored function named item_total
6 CREATE FUNCTION item_total (
7     order_item_id_var INT -- ID of the order item
8 )
9 RETURNS DECIMAL(10,2) -- Returns a decimal value
10 DETERMINISTIC -- the function returns the same result for the same input
11 READS SQL DATA -- the function reads data from the database
12 BEGIN
13     -- Declare variables with new names
14     DECLARE calculated_total_var DECIMAL(10,2);
15     DECLARE discounted_price_var DECIMAL(10,2);
16     DECLARE quantity_var INT;
17
18     -- Computes the discounted price of the item using the discount_price function
19     SET discounted_price_var = discount_price(order_item_id_var);
20
21     -- The quantity of the item from the order_items table
22     SELECT
23         quantity
24     INTO
25         quantity_var
26     FROM
27         order_items
28     WHERE
29         item_id = order_item_id_var;
30
31     -- Computes the total amount for the item
32     IF discounted_price_var IS NULL OR quantity_var IS NULL THEN
33         SET calculated_total_var = NULL;
34     ELSE
35         SET calculated_total_var = discounted_price_var * quantity_var;
36     END IF;
37
38     -- Returns the total amount
39     RETURN calculated_total_var;
40 END //
41
42 DELIMITER ;
43
44 -- Calls the function with a query statement
45 -- Computes the item total for each of the item quantity in the order_item table
46 SELECT
47     oi.item_id,
48     oi.item_price,
49     oi.discount_amount,
50     discount_price(oi.item_id) AS discounted_price,
51     oi.quantity,
52     item_total(oi.item_id) AS items_total_price -- Calls function
53 FROM
54     order_items oi
55 
```

The 'Result Grid' at the bottom shows the output of the query, with the following data:

Item Id	Item Price	Discount Amount	Discounted Price	Quantity	Items Total Price
1	1199.00	359.70	839.30	1	839.30
2	489.99	186.20	303.79	1	303.79
3	2517.00	1308.84	1208.16	1	1208.16
4	415.00	161.85	253.15	1	253.15
5	1199.00	359.70	839.30	2	1678.60
6	299.00	0.00	299.00	1	299.00
7	299.00	0.00	299.00	1	299.00
8	699.00	209.70	489.30	1	489.30
9	799.99	240.00	559.99	1	559.99
10	699.99	210.00	489.99	1	489.99
11	799.99	120.00	679.99	1	679.99
12	699.00	209.70	489.30	1	489.30

The 'Output' pane at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	17:16:29	DROP FUNCTION IF EXISTS item_total	0 row(s) affected, 1 warning...	0.015 sec
2	17:16:29	CREATE the stored function named item_total	0 row(s) affected	0.016 sec
3	17:16:29	CALLS the function with a query statement	12 row(s) returned	0.000 sec / 0.000 sec

Note: The figure illustrates the MySQL Workbench result after performing steps 5.