

**Portfolio Project**

Stephan Peters

Colorado State University Global

ITS410-2 23WC: Database Management

Jamia Mills

10 March 2024

## **Part Two: Queries**

This query returns the product name, the list price, the discount percent, the discount amount, and the discount price from the my guitar shop products table and limits the results to five lines and sorts the results in descending order:

### Source Code 1

*Source code for step 1.*

```
USE my_guitar_shop;

SELECT product_name, list_price, discount_percent,
       ROUND((list_price * discount_percent * .01), 2) AS discount_amount,
       ROUND((list_price - (list_price * discount_percent * .01)), 2) AS
discount_price
       FROM products
ORDER BY discount_price DESC
LIMIT 5;
```

### Screenshot 1

*Screenshot of Execution for step 1.*

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
USE my_guitar_shop;

SELECT product_name, list_price, discount_percent,
       ROUND((list_price * discount_percent * .01), 2) AS discount_amount,
       ROUND((list_price - (list_price * discount_percent * .01)), 2) AS discount_price
       FROM products
ORDER BY discount_price DESC
LIMIT 5;
```

The results are displayed in the Result Grid below the query editor:

product_name	list_price	discount_percent	discount_amount	discount_price
Gibson SG	2517.00	52.00	1308.84	1208.16
Gibson Les Paul	1199.00	30.00	359.70	839.30
Yamaha DGX 640 88-Key Digital Piano	799.99	0.00	0.00	799.99
Tama 5-Piece Drum Set with Cymbals	799.99	15.00	120.00	679.99
Fender Precision	799.99	30.00	240.00	559.99

The bottom of the screenshot shows the Action Output pane with the following entries:

#	Time	Action	Message	Duration / Fetch
1	22:09:57	USE my_guitar_shop	0 row(s) affected	0.000 sec
2	22:10:03	SELECT product_name, list_price, discount_percent, ROUND(list_price * discount_perc...	5 row(s) returned	0.000 sec / 0.000 sec

This query returns the item id, the item price, the discount amount, the quantity, the total price, the discount total, and the item total from the my guitar shop products table and limits the results to where the item total is over \$500, sorted in descending order:

## Source Code 2

*Source code for step 2.*

```
USE my_guitar_shop;

SELECT item_id, item_price, discount_amount, quantity,
       (item_price * quantity) AS price_total,
       (discount_amount * quantity) AS discount_total,
       item_total(item_id) AS item_total
FROM order_items
WHERE item_total(item_id) > 500
ORDER BY item_total DESC;
```

## Screenshot 2

*Screenshot of Execution for step 2.*

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with 'my\_guitar\_shop' selected. The main editor window contains the following SQL query:

```
1 USE my_guitar_shop;
2
3 SELECT item_id, item_price, discount_amount, quantity,
4        (item_price * quantity) AS price_total,
5        (discount_amount * quantity) AS discount_total,
6        item_total(item_id) AS item_total
7 FROM order_items
8 WHERE item_total(item_id) > 500
9 ORDER BY item_total DESC;
```

The 'Result Grid' at the bottom shows the following data:

item_id	item_price	discount_amount	quantity	price_total	discount_total	item_total
5	1199.00	359.70	2	2398.00	719.40	1678.60
3	2517.00	1308.84	1	2517.00	1308.84	1208.16
1	1199.00	359.70	1	1199.00	359.70	839.30
11	799.99	120.00	1	799.99	120.00	679.99
9	799.99	240.00	1	799.99	240.00	559.99

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

#	Time	Action	Message	Duration / Fetch
1	22:24:37	USE my_guitar_shop	0 row(s) affected	0.000 sec
2	22:24:42	SELECT item_id, item_price, discount_amount, quantity, (item_price * quantity) AS price_total, (discount_amount * quantity) AS discount_total, item_total(item_id) AS item_total	5 row(s) returned	0.016 sec / 0.000 sec

This query returns the product name and list price from the my guitar shop products table where the list price is identical to the list price of another product:

### Source Code 3

*Source code for step 3.*

```
USE my_guitar_shop;

SELECT product_name, list_price FROM products
WHERE list_price IN (
    SELECT list_price FROM products
    GROUP BY list_price
    HAVING count(list_price) > 1
);
```

### Screenshot 3

*Screenshot of Execution for step 3.*

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with 'my\_guitar\_shop' selected. The main editor shows the following SQL query:

```
1 USE my_guitar_shop;
2
3 SELECT product_name, list_price FROM products
4 WHERE list_price IN (
5     SELECT list_price FROM products
6     GROUP BY list_price
7     HAVING count(list_price) > 1
8 );
9
```

The 'Result Grid' shows the following data:

product_name	list_price
Fender Precision	799.99
Tama 5-Piece Drum Set with Cymbals	799.99
Yamaha DGX 640 88-Key Digital Piano	799.99

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

#	Time	Action	Message	Duration / Fetch
1	09:53:37	USE my_guitar_shop	0 row(s) affected	0.000 sec
2	09:53:37	SELECT product_name, list_price FROM products WHERE list_price IN ( SELECT list_price FROM products GROUP BY list_price HAVING count(list_price) > 1 );	3 row(s) returned	0.000 sec / 0.000 sec

This query returns the category name from the categories table, and the product id from the products table of the my guitar workshop database where the category name is not used:

#### Source Code 4

*Source code for step 4.*

```
USE my_guitar_shop;

SELECT c.category_name, p.product_id FROM categories c
LEFT JOIN products p USING (category_id)
WHERE p.product_id IS NULL
UNION ALL
SELECT c.category_name, p.product_id FROM categories c
RIGHT JOIN products p USING (category_id)
WHERE p.product_id IS NULL;
```

#### Screenshot 4

*Screenshot of Execution for step 4.*

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'my\_guitar\_shop' database schema with tables 'categories' and 'products'. The main editor window contains the SQL query from Source Code 4. The 'Result Grid' at the bottom shows the output of the query, which consists of two rows: 'Brass' and 'Woodwinds', both with a 'product\_id' of 'null'. The 'Action Output' panel at the bottom right shows the execution progress, including the query text and the number of rows returned (2 rows).

category_name	product_id
Brass	null
Woodwinds	null

These queries add the following customer to the customers table in the my guitar workshop database Rick Raven, rick@raven.com, and verify the insertion:

### Source Code 5

*Source code for step 5.*

```
USE my_guitar_shop;

INSERT INTO customers(email_address, password, first_name, last_name)
VALUES ('rick@raven.com', '', 'Rick', 'Raven');

SELECT * FROM customers
WHERE email_address = 'rick@raven.com';
```

### Screenshot 5

*Screenshot of Execution for step 5.*

The screenshot displays the MySQL Workbench interface. The Navigator pane on the left shows the 'my\_guitar\_shop' schema selected. The central query editor contains the following SQL code:

```
1 USE my_guitar_shop;
2
3 INSERT INTO customers(email_address, password, first_name, last_name)
4 VALUES ('rick@raven.com', '', 'Rick', 'Raven');
5
6 SELECT * FROM customers
7 WHERE email_address = 'rick@raven.com';
8
```

Below the query editor, the Result Grid shows the output of the SELECT query:

customer_id	email_address	password	first_name	last_name	shipping_address_id	billing_address_id
9	rick@raven.com		Rick	Raven		

The Output pane at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
3	11:42:09	INSERT INTO customers(email_address, password, first_name, last_name) VALUES ('rick@raven.com', '', 'Rick', 'Raven');	1 row(s) affected	0.031 sec
4	11:42:09	SELECT * FROM customers WHERE email_address = 'rick@raven.com'	1 row(s) returned	0.000 sec / 0.000 sec

This query returns the email address and the number of distinct products the customer ordered for only customers that have ordered more than one product, sorted by email address:

### Source Code 6

*Source code for step 6.*

```
USE my_guitar_shop;

SELECT c.email_address, COUNT(DISTINCT oi.product_id) AS count
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
INNER JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY c.email_address
HAVING count > 1
ORDER BY c.email_address;
```

### Screenshot 6

*Screenshot of Execution for step 6.*

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view of databases, with 'my\_guitar\_shop' selected. The main 'Query Editor' window contains the SQL query from Source Code 6. Below the editor, the 'Result Grid' shows the output of the query, which is a table with two columns: 'email\_address' and 'count'. The data rows are as follows:

email_address	count
alan.sherwood@yahoo.com	3
david.goldstein@hotmail.com	2
frankwilson@sbcglobal.net	3

At the bottom, the 'Output' pane shows the execution log. It contains two entries:

#	Time	Action	Message	Duration / Fetch
1	16:58:40	USE my_guitar_shop	0 row(s) affected	0.000 sec
2	16:58:40	SELECT c.email_address, COUNT(DISTINCT oi.product_id) AS count FROM customers c...	3 row(s) returned	0.047 sec / 0.000 sec

The Windows taskbar at the bottom shows the system time as 5:01 PM on 2/13/2024.



This script returns the category name from the categories table, the product name from the products table, and the total quantity purchased from the order items table that displays the total quantity purchased for each product within each category with all null values replaced with literal values:

### Source Code 7

*Source code for step 7.*

```
USE my_guitar_shop;

DROP TABLE IF EXISTS total_quantity_purchased;
CREATE TEMPORARY TABLE total_quantity_purchased
SELECT c.category_name, p.product_name, sum(oi.quantity) AS total_quantity
FROM products p
JOIN order_items oi USING (product_id)
JOIN categories c USING (category_id)
GROUP BY oi.product_id;

ALTER TABLE total_quantity_purchased
ADD PRIMARY KEY(product_name);

INSERT IGNORE INTO total_quantity_purchased (
    product_name,
    category_name,
    total_quantity
)
SELECT p.product_name, c.category_name, 0 FROM products p
JOIN categories c USING (category_id);

SELECT * FROM total_quantity_purchased
ORDER BY category_name;
```

Screenshot 7a shows the resulting table from the first query in the script, which returns the category name, product name, and total number ordered of each product in inventory that has had any sales in order items. Screenshot 7b shows the total number of products sold in each category that has products in inventory.

## Screenshot 7a

*Screenshot of number of categories and products sold for each product.*

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'my\_guitar\_shop' database schema with tables like addresses, administrators, categories, customers, discounts, order\_items, orders, products, and views. The main editor window shows a SQL query (Query 1) with the following code:

```

12 ADD PRIMARY KEY(product_name);
13
14 INSERT IGNORE INTO total_quantity_purchased (product_name, category_name, total_quantity_sold)
15 SELECT p.product_name, c.category_name, 0
16 FROM products p
17 JOIN categories c USING (category_id);
18
19 SELECT * FROM total_quantity_purchased
20 ORDER BY category_name;
21

```

The 'Result Grid' at the bottom displays the results of the query, showing a list of products and their total quantity sold, grouped by category. The data is as follows:

category_name	product_name	total_quantity_sold
Basses	Fender Precision	1
Basses	Hofner Icon	0
Drums	Ludwig 5-piece Drum Set with Cymbals	1
Drums	Tama 5-Piece Drum Set with Cymbals	1
Guitars	Fender Stratocaster	2
Guitars	Gibson Les Paul	3
Guitars	Gibson SG	1
Guitars	Rodriguez Caballero 11	1
Guitars	Washburn D10S	2
Guitars	Yamaha FG700S	1
Keyboards	Yamaha DGX 640 88-Key Digital Piano	0

## Screenshot 7b

*Screenshot of number products sold in each category.*

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'my\_guitar\_shop' database schema. The main editor window shows a SQL query (Query 1) with the following code:

```

10
11 ALTER TABLE total_quantity_purchased
12 ADD PRIMARY KEY(product_name);
13
14 INSERT IGNORE INTO total_quantity_purchased (product name, category name, total quantity_sold)
15 SELECT p.product_name, c.category_name, 0
16 FROM products p
17 JOIN categories c USING (category_id);
18
19 SELECT * FROM total_quantity_purchased
20 ORDER BY category_name;
21
22 SELECT category_name, sum(total_quantity_sold) as total_quantity_sold
23 FROM total_quantity_purchased
24 GROUP BY category_name;

```

The 'Result Grid' at the bottom displays the results of the query, showing the total quantity sold for each category. The data is as follows:

category_name	total_quantity_sold
Basses	1
Guitars	10
Drums	2
Keyboards	0

The script in Source Code 8a creates an order manager role that grants the following privileges for the my guitar shop database to a user with this role:

- SELECT, INSERT, UPDATE, and DELETE privileges for the customers, addresses, orders, and order\_items tables.
- SELECT privileges for the products and categories tables.
- No GRANT privileges to other users.

### Source Code 8a

*Source code for step 8a.*

```
USE my_guitar_shop;

DROP ROLE IF EXISTS 'order_manager';
CREATE ROLE IF NOT EXISTS 'order_manager';

REVOKE USAGE ON *.* FROM order_manager;

GRANT SELECT, INSERT, UPDATE, DELETE
ON my_guitar_shop.customers
TO 'order_manager';

GRANT SELECT, INSERT, UPDATE, DELETE
ON my_guitar_shop.addresses
TO 'order_manager';

GRANT SELECT, INSERT, UPDATE, DELETE
ON my_guitar_shop.orders
TO 'order_manager';

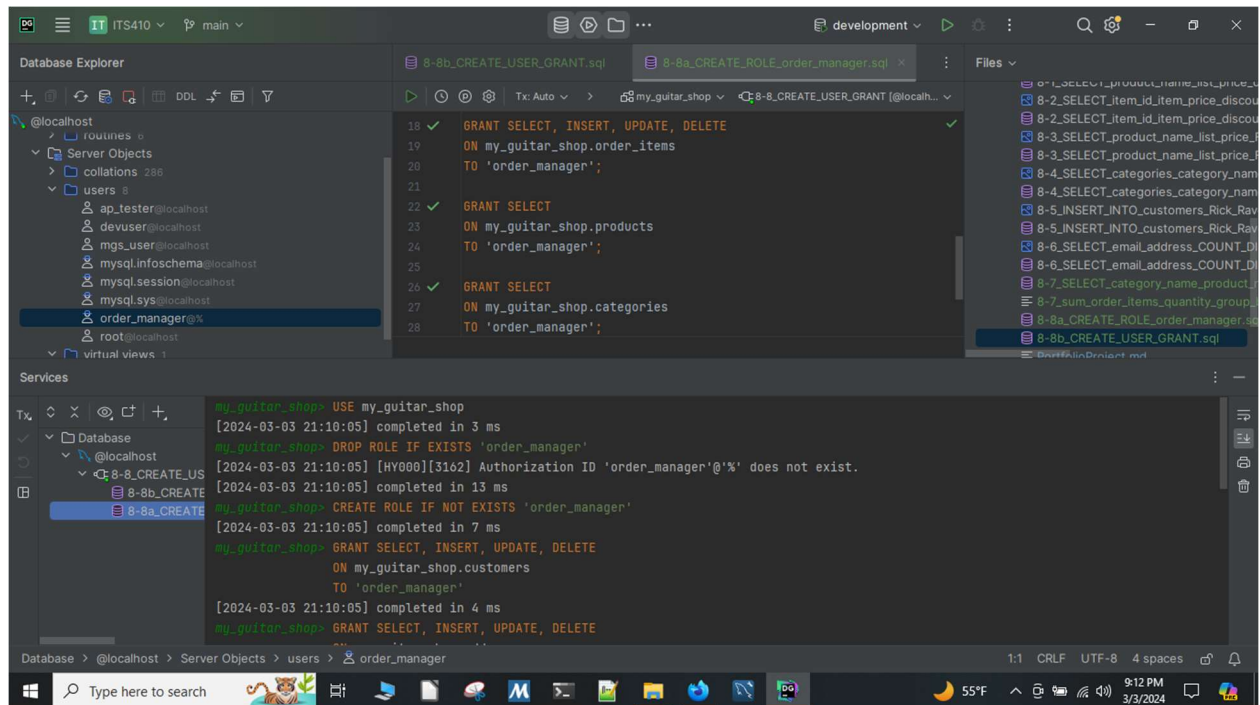
GRANT SELECT, INSERT, UPDATE, DELETE
ON my_guitar_shop.order_items
TO 'order_manager';

GRANT SELECT
ON my_guitar_shop.products
TO 'order_manager';

GRANT SELECT
ON my_guitar_shop.categories
TO 'order_manager';
```

## Screenshot 8a

*Screenshot of creating a user role for step 8.*



The script in Source Code 8b creates a user named speters with the order manager role and all privileges and restrictions of the order manager role:

## Source Code 8b

*Source code for step 8b.*

```
USE my_guitar_shop;
DROP USER IF EXISTS speters;

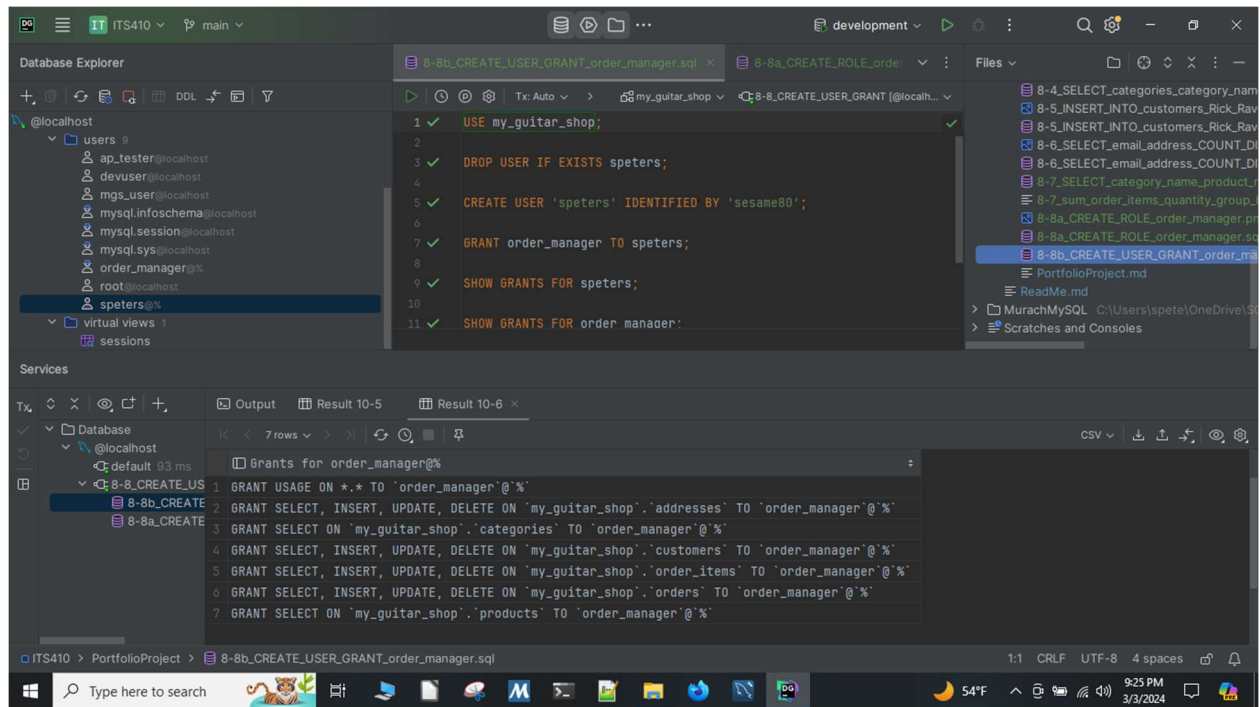
CREATE USER 'speters' IDENTIFIED BY 'sesame80';

GRANT order_manager TO speters;

SHOW GRANTS FOR speters;
SHOW GRANTS FOR order_manager;
```

## Screenshot 8b

*Screenshot of creating a user with the order manager role for step 8.*



## GitHub Repository

In addition to the solutions above, I created a GitHub repository for the project. This repository is located at

[https://github.com/speters33w/CSUGlobal\\_ITS410/blob/main/PortfolioProject/PortfolioProject.md](https://github.com/speters33w/CSUGlobal_ITS410/blob/main/PortfolioProject/PortfolioProject.md)

Figure 9 shows a screenshot of the markdown file for the portfolio project of this repository.

**Figure 9**

*Screenshot of the portfolio project page of my ITS 410 GitHub repository*

