

Lab 6: How to Code Summary Queries

Step 1: Write a SELECT statement that returns these columns: The count of the number of orders in the Orders table. The sum of the tax_amount columns in the Orders table.

The screenshot shows the MySQL Workbench interface. The left sidebar contains the 'Navigation' pane with sections for 'MANAGEMENT' (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), 'INSTANCE' (Startup / Shutdown, Server Logs, Options File), 'PERFORMANCE' (Dashboard, Performance Reports, Performance Schema Setup), and 'Administration' (Schemas). The main editor window displays a SQL query:

```
1 SELECT
2   COUNT(*) AS order_count,
3   SUM(tax_amount) AS total_tax_amount
4 FROM
5   orders;
```

Below the query editor, the 'Result Grid' shows the following data:

order_count	total_tax_amount
9	122.24

The bottom pane shows the 'Output' tab with a table of actions and messages:

#	Time	Action	Message	Duration / Fetch
34	17:41:00	INSERT INTO orders (order_id, customer_id, order_date, ship_amount, tax_amount, ship_dia...	9 row(s) affected Records: 9 Duplicates: 0 Warnings: 0	0.031 sec
35	17:41:00	INSERT INTO order_items (item_id, order_id, product_id, item_price, discount_amount, quan...	12 row(s) affected Records: 12 Duplicates: 0 Warnings: 0	0.031 sec
36	17:41:00	INSERT INTO administrators (admin_id, email_address, password, first_name, last_name) VA...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.032 sec
37	17:41:00	GRANT SELECT, INSERT, UPDATE, DELETE ON * TO mgs_user@localhost IDENTIFIED BY ...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds ...	0.015 sec
38	17:41:10	SELECT COUNT(*) AS order_count, SUM(tax_amount) AS total_tax_amount FROM ...	Error Code: 1146. Table 'my_guitar_shop.Orders' doesn't exist	0.032 sec
39	17:43:09	SELECT COUNT(*) AS order_count, SUM(tax_amount) AS total_tax_amount FROM ...	1 row(s) returned	0.015 sec / 0.000 sec

Step 2: Write a SELECT statement that returns one row for each category that has products with these columns: The category_name column from the Categories table. The count of the products in the Products table. The list price of the most expensive product in the Products table Sort the result set so the category with the most products appears first.

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

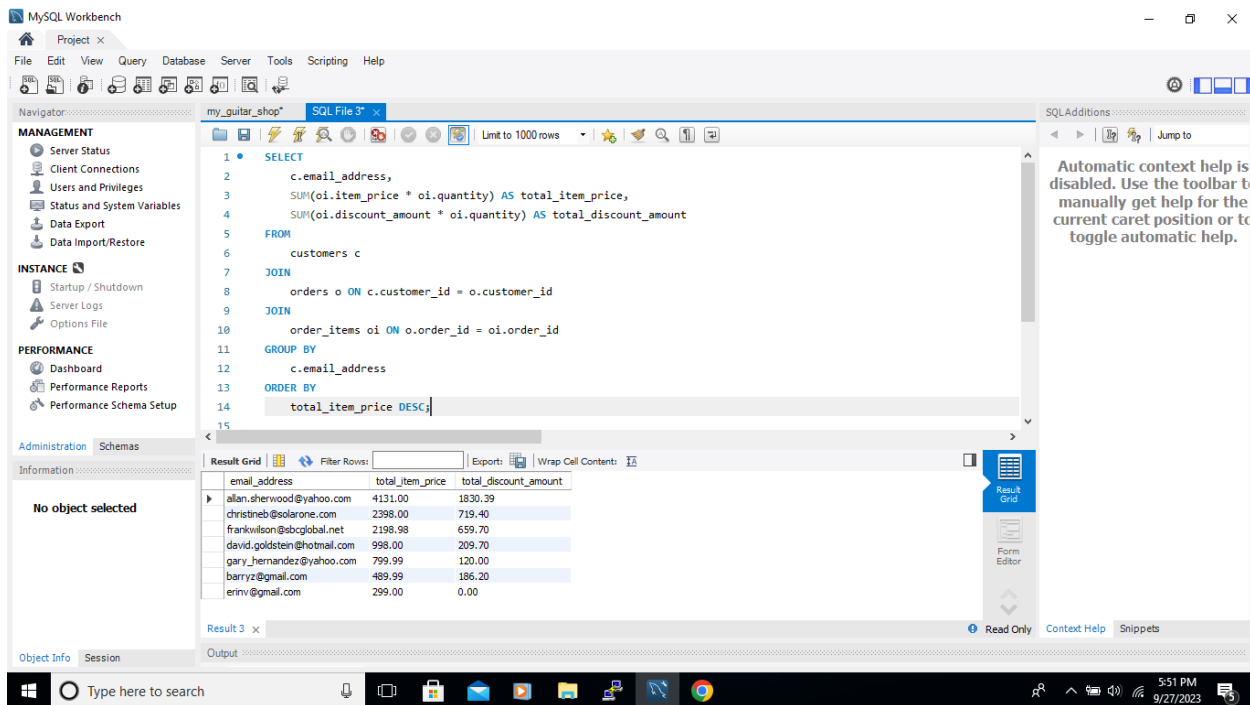
```
1 SELECT
2   c.category_name,
3   COUNT(p.product_id) AS product_count,
4   MAX(p.list_price) AS max_list_price
5 FROM
6   categories c
7 JOIN
8   products p ON c.category_id = p.category_id
9 GROUP BY
10  c.category_name
11 ORDER BY
12  product_count DESC;
```

The Results window displays the following data:

category_name	product_count	max_list_price
Guitars	6	2517.00
Basses	2	799.99
Drums	2	799.99

The interface also shows the left-hand sidebar with various toolboxes (MANAGEMENT, INSTANCE, PERFORMANCE, Administration, Information) and the bottom status bar with system information.

Step 3: Write a SELECT statement that 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. Sort the result set in descending sequence by the item price total for each customer.



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

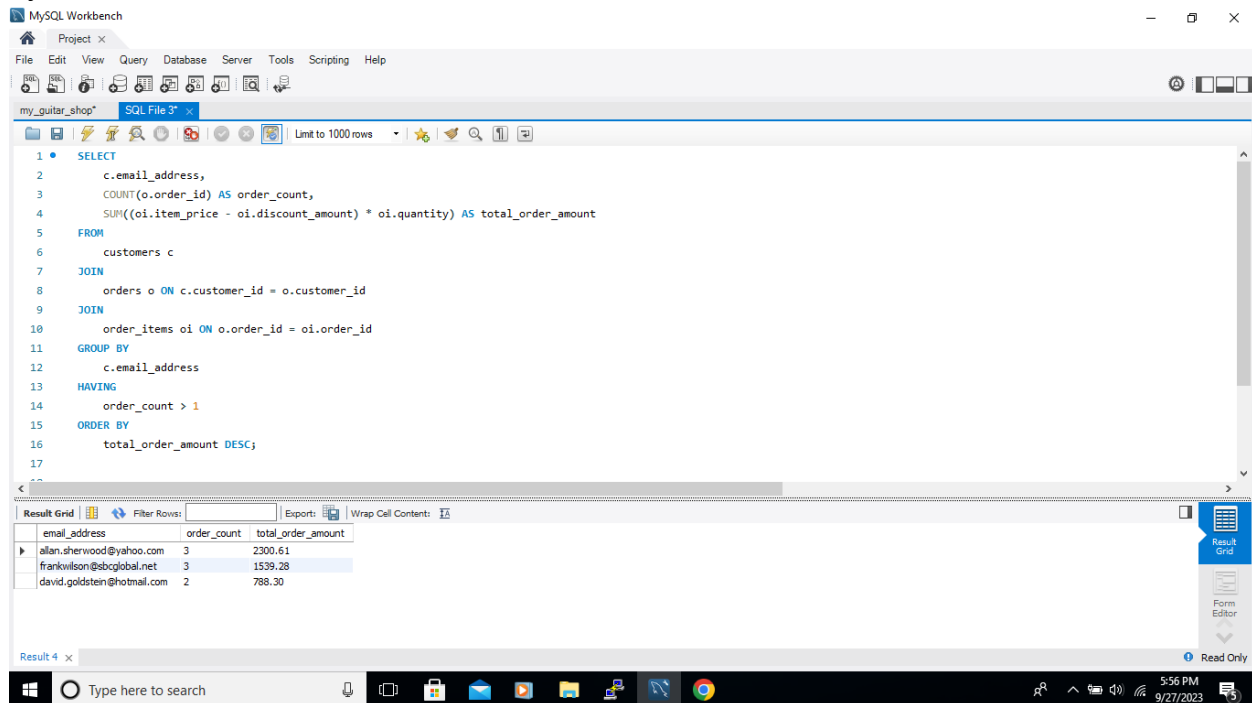
```
1 SELECT
2   c.email_address,
3   SUM(o.item_price * oi.quantity) AS total_item_price,
4   SUM(oi.discount_amount * oi.quantity) AS total_discount_amount
5 FROM
6   customers c
7 JOIN
8   orders o ON c.customer_id = o.customer_id
9 JOIN
10  order_items oi ON o.order_id = oi.order_id
11 GROUP BY
12   c.email_address
13 ORDER BY
14   total_item_price DESC;
```

The Results window displays the following data:

email_address	total_item_price	total_discount_amount
alan.shenwood@yahoo.com	4131.00	1830.39
christineb@solarone.com	2398.00	719.40
frankwilson@ibcglobal.net	2198.98	659.70
david.goldstein@hotmail.com	998.00	209.70
gary_hernandez@yahoo.com	799.99	120.00
berryz@gmail.com	489.99	186.20
eriniv@gmail.com	299.00	0.00

The interface also shows a sidebar with navigation options like MANAGEMENT, INSTANCE, and PERFORMANCE. The bottom status bar indicates the time is 5:51 PM on 9/27/2023.

Step 4: Write a SELECT statement that 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 (Hint: First, subtract the discount amount from the price. Then, multiply by the quantity.) Return only those rows where the customer has more than 1 order. Sort the result set in descending sequence by the sum of the line item amounts.



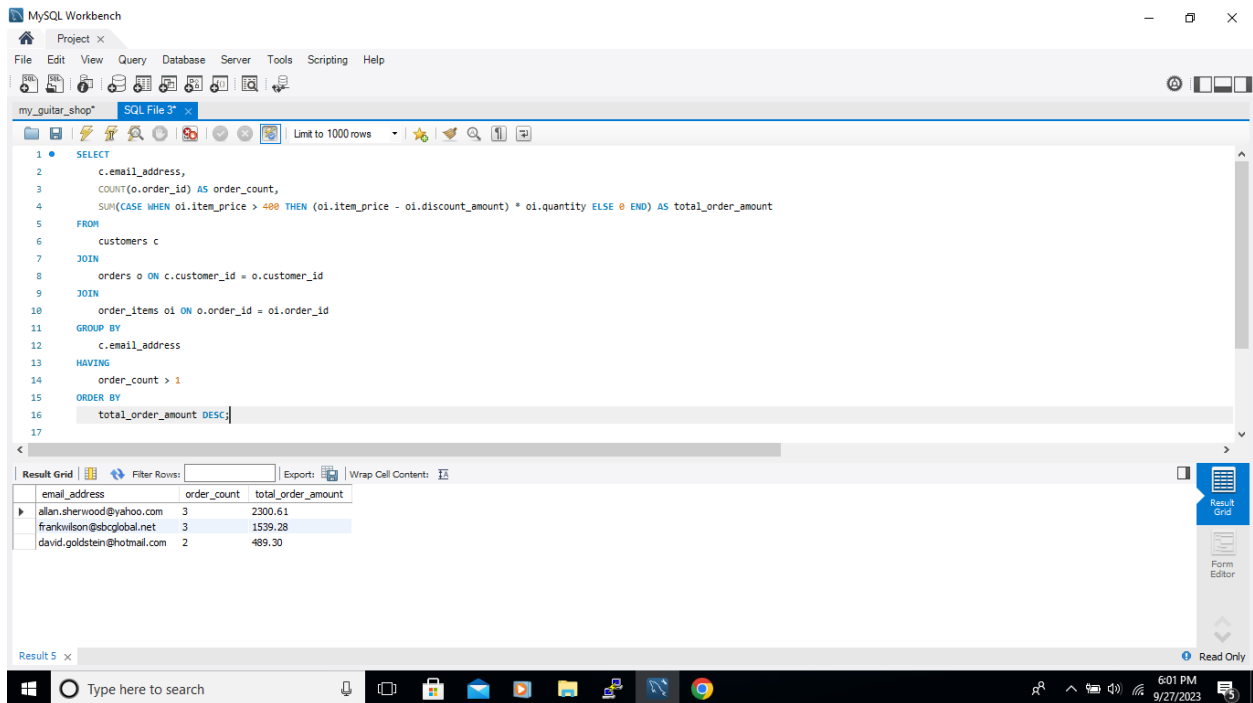
The screenshot shows the MySQL Workbench interface. The SQL Editor window displays a query that selects customer email addresses, order counts, and total order amounts, filtered by customers with more than one order, and sorted by total order amount in descending order.

```
1 SELECT
2   c.email_address,
3   COUNT(o.order_id) AS order_count,
4   SUM((oi.item_price - oi.discount_amount) * oi.quantity) AS total_order_amount
5 FROM
6   customers c
7 JOIN
8   orders o ON c.customer_id = o.customer_id
9 JOIN
10  order_items oi ON o.order_id = oi.order_id
11 GROUP BY
12  c.email_address
13 HAVING
14  order_count > 1
15 ORDER BY
16  total_order_amount DESC;
```

The Results window shows the output of the query, displaying three columns: email_address, order_count, and total_order_amount. The results are sorted by total_order_amount in descending order.

email_address	order_count	total_order_amount
allan.sherwood@yahoo.com	3	2300.61
frankwilson@sbcglobal.net	3	1539.28
david.goldstein@hotmail.com	2	788.30

Step 5: Modify the solution to exercise 4 so it only counts and totals line items that have an item_price value that's greater than 400.



The screenshot shows the MySQL Workbench interface. The SQL Editor contains a query that filters for items with a price greater than 400. The query is as follows:

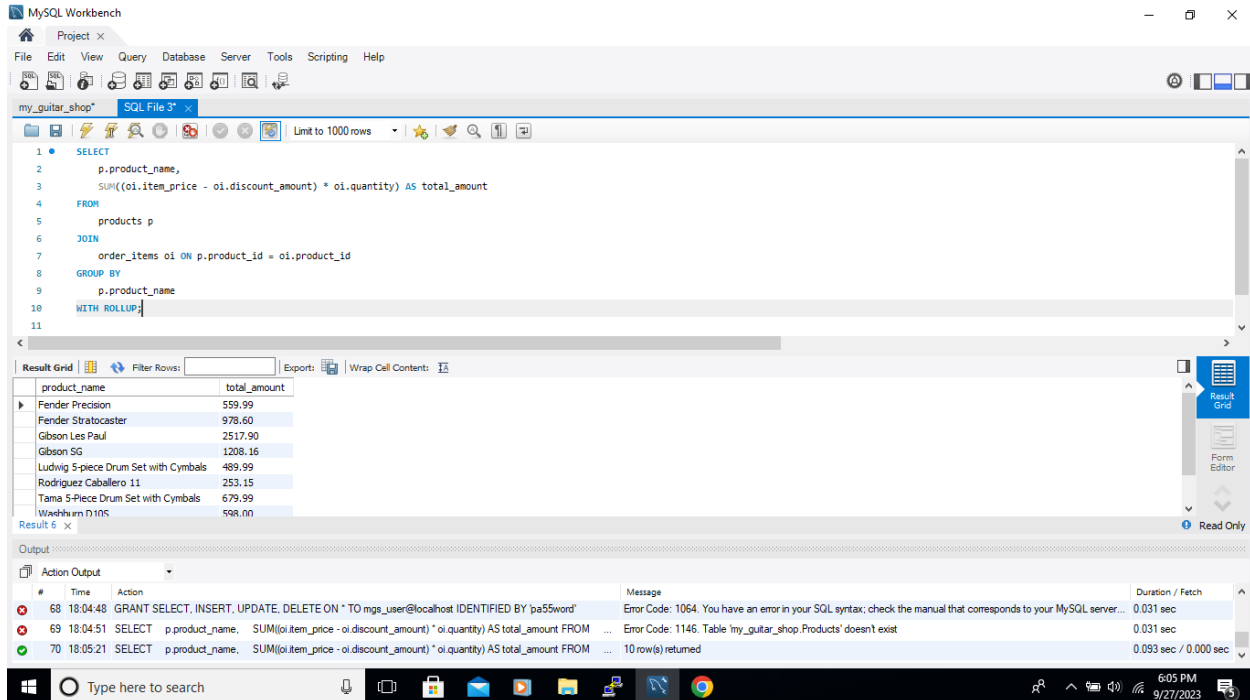
```
1 SELECT
2   c.email_address,
3   COUNT(o.order_id) AS order_count,
4   SUM(CASE WHEN oi.item_price > 400 THEN (oi.item_price - oi.discount_amount) * oi.quantity ELSE 0 END) AS total_order_amount
5 FROM
6   customers c
7 JOIN
8   orders o ON c.customer_id = o.customer_id
9 JOIN
10  order_items oi ON o.order_id = oi.order_id
11 GROUP BY
12  c.email_address
13 HAVING
14  order_count > 1
15 ORDER BY
16  total_order_amount DESC
17
```

The Results window shows the output of the query, which is a table with three columns: email_address, order_count, and total_order_amount. The results are sorted by total_order_amount in descending order.

email_address	order_count	total_order_amount
allan.sherwood@yahoo.com	3	2300.61
frankwilson@sbglobal.net	3	1539.28
david.goldstein@hotmail.com	2	489.30

The bottom of the screenshot shows the Windows taskbar with the search bar and system clock indicating 6:01 PM on 9/27/2023.

Step 6: Write a SELECT statement that answers this question: What is the total amount ordered for each product? Return these columns: The product_name column from the Products table. The total amount for each product in the Order_Items table (Hint: You can calculate the total amount by subtracting the discount amount from the item price and then multiplying it by the quantity) Use the WITH ROLLUP operator to include a row that gives the grand total. Note: Once you add the WITH ROLLUP operator, you may need to use MySQL Workbench's Execute SQL Script button instead of its Execute Current Statement button to execute this statement.



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

```

1 SELECT
2   p.product_name,
3   SUM((oi.item_price - oi.discount_amount) * oi.quantity) AS total_amount
4 FROM
5   products p
6 JOIN
7   order_items oi ON p.product_id = oi.product_id
8 GROUP BY
9   p.product_name
10 WITH ROLLUP;
11

```

The Result Grid shows the following data:

product_name	total_amount
Fender Precision	559.99
Fender Stratocaster	978.60
Gibson Les Paul	2517.90
Gibson SG	1208.16
Ludwig 5-piece Drum Set with Cymbals	489.99
Rodriguez Caballero 11	253.15
Tama 5-Piece Drum Set with Cymbals	679.99
Washburn D10S	598.00

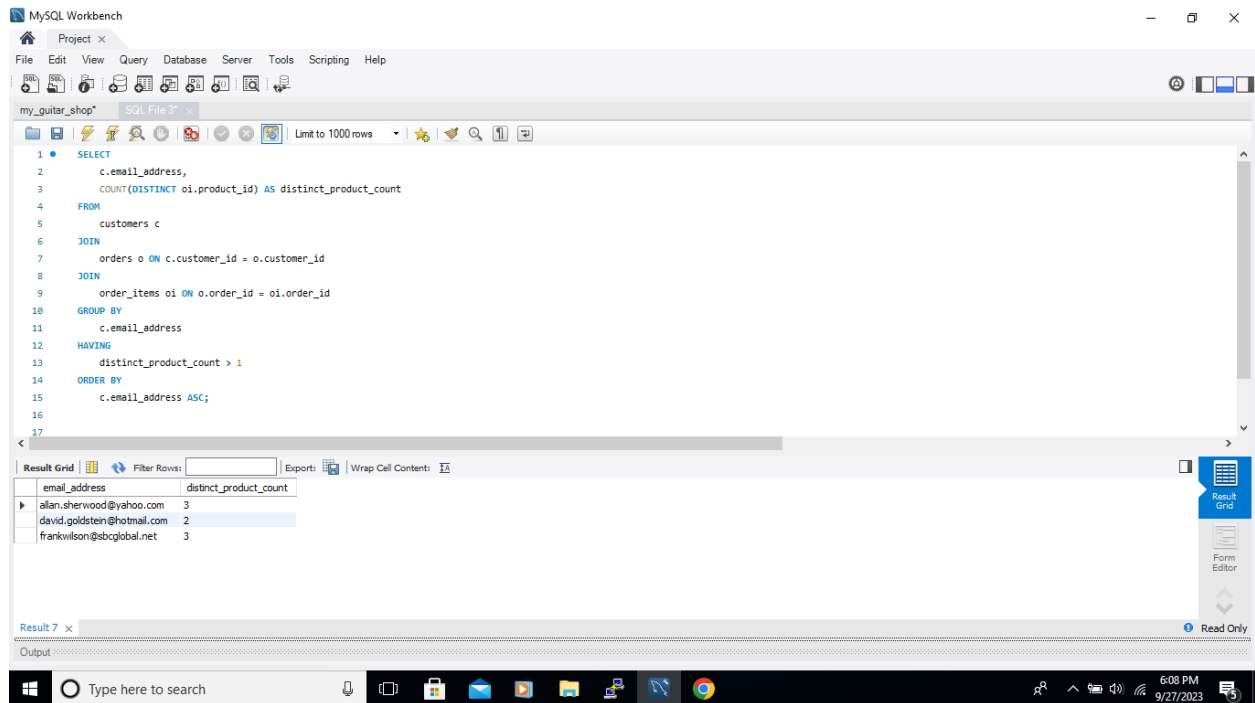
The Output pane shows the following messages:

```

68 18:04:48 GRANT SELECT, INSERT, UPDATE, DELETE ON * TO mgs_user@localhost IDENTIFIED BY 'pa55word' ... Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server... 0.031 sec
69 18:04:51 SELECT p.product_name, SUM((oi.item_price - oi.discount_amount) * oi.quantity) AS total_amount FROM ... Error Code: 1146. Table 'my_guitar_shop.Products' doesn't exist 0.031 sec
70 18:05:21 SELECT p.product_name, SUM((oi.item_price - oi.discount_amount) * oi.quantity) AS total_amount FROM ... 10 row(s) returned 0.093 sec / 0.000 sec

```

Step 7: Write a SELECT statement that answers this question: Which customers have ordered more than one product? Return these columns: The email_address column from the Customers table. The count of distinct products from the customer's orders. Sort the result set in ascending sequence by the email_address column.



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

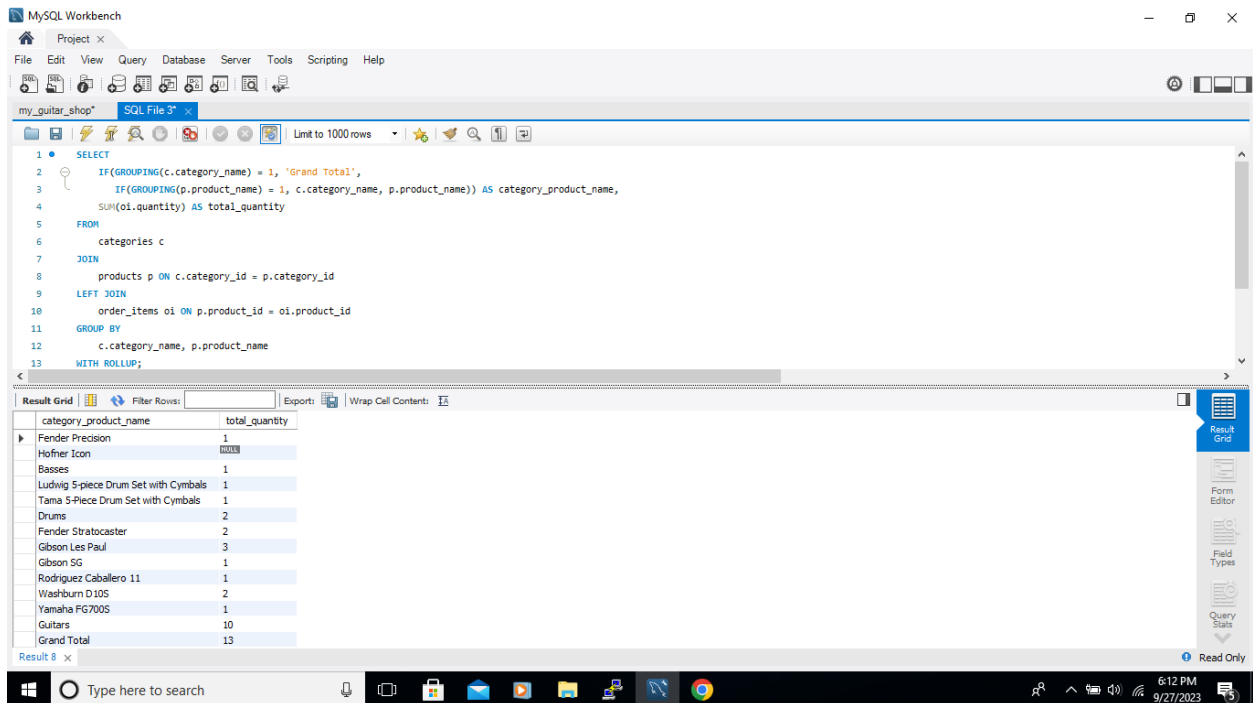
```
1 SELECT
2   c.email_address,
3   COUNT(DISTINCT oi.product_id) AS distinct_product_count
4 FROM
5   customers c
6 JOIN
7   orders o ON c.customer_id = o.customer_id
8 JOIN
9   order_items oi ON o.order_id = oi.order_id
10 GROUP BY
11   c.email_address
12 HAVING
13   distinct_product_count > 1
14 ORDER BY
15   c.email_address ASC;
16
17
```

The Results pane displays the following data:

email_address	distinct_product_count
allan.sherwood@yahoo.com	3
david.goldstein@hotmail.com	2
frankwilson@sbcglobal.net	3

The Windows taskbar at the bottom shows the time as 6:08 PM on 9/27/2023.

Step 8: Write a SELECT statement that answers this question: What is the total quantity purchased for each product within each category? Return these columns: The category_name column from the category table. The product_name column from the products table. The total quantity purchased for each product with orders in the Order_Items table. Use the WITH ROLLUP operator to include rows that give a summary for each category name as well as a row that gives the grand total. Use the IF and GROUPING functions to replace null values in the category_name and product_name columns with literal values if they're for summary rows.



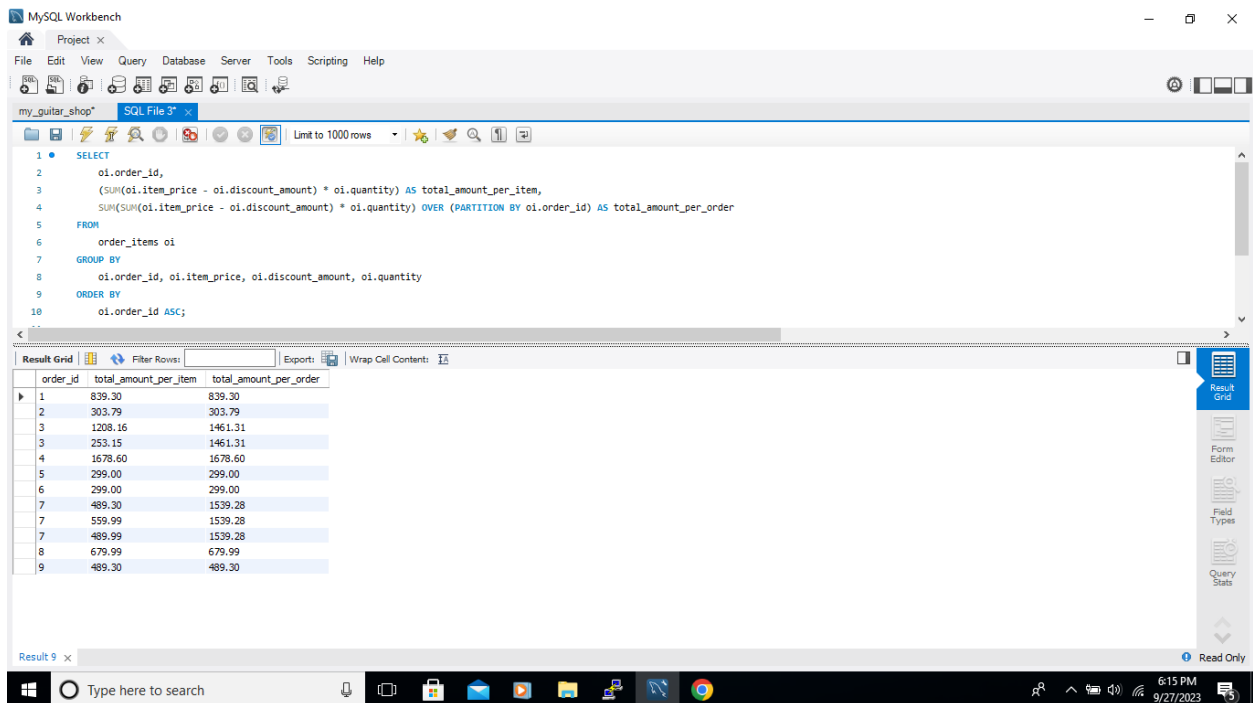
The screenshot shows the MySQL Workbench interface. The SQL Editor at the top contains a query that uses the WITH ROLLUP operator to calculate the total quantity for each product within each category, including a grand total row. The query uses the IF and GROUPING functions to handle null values in the category_name and product_name columns for summary rows.

```
1 SELECT
2     IF(GROUPING(c.category_name) = 1, 'Grand Total',
3     IF(GROUPING(p.product_name) = 1, c.category_name, p.product_name)) AS category_product_name,
4     SUM(oi.quantity) AS total_quantity
5 FROM
6     categories c
7 JOIN
8     products p ON c.category_id = p.category_id
9 LEFT JOIN
10    order_items oi ON p.product_id = oi.product_id
11 GROUP BY
12     c.category_name, p.product_name
13 WITH ROLLUP;
```

The Results window at the bottom displays the query results in a table with two columns: category_product_name and total_quantity. The table lists the total quantity for each product within each category, including a grand total row.

category_product_name	total_quantity
Fender Precision	1
Hofner Icon	1
Basses	1
Ludwig 5-piece Drum Set with Cymbals	1
Tama 5-Piece Drum Set with Cymbals	1
Drums	2
Fender Stratocaster	2
Gibson Les Paul	3
Gibson SG	1
Rodriguez Caballero 11	1
Washburn D10S	2
Yamaha FG700S	1
Guitars	10
Grand Total	13

Step 9: Write a SELECT statement that uses an aggregate window function to get the total amount of each order. Return these columns: The order_id column from the Order_Items table. The total amount for each order item in the Order_Items table (Hint: You can calculate the total amount by subtracting the discount amount from the item price and then multiplying it by the quantity). The total amount for each order Sort the result set in ascending sequence by the order_id column.



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

```
1 SELECT
2   oi.order_id,
3   (SUM(oi.item_price - oi.discount_amount) * oi.quantity) AS total_amount_per_item,
4   SUM(SUM(oi.item_price - oi.discount_amount) * oi.quantity) OVER (PARTITION BY oi.order_id) AS total_amount_per_order
5 FROM
6   order_items oi
7 GROUP BY
8   oi.order_id, oi.item_price, oi.discount_amount, oi.quantity
9 ORDER BY
10  oi.order_id ASC;
```

The Results tab displays the following data:

order_id	total_amount_per_item	total_amount_per_order
1	839.30	839.30
2	303.79	303.79
3	1208.16	1461.31
3	253.15	1461.31
4	1678.60	1678.60
5	299.00	299.00
6	299.00	299.00
7	489.30	1539.28
7	559.99	1539.28
7	489.99	1539.28
8	679.99	679.99
9	489.30	489.30

Step 10: Modify the solution to exercise 9 so the column that contains the total amount for each order contains a cumulative total by item amount. Add another column to the SELECT statement that uses an aggregate window function to get the average item amount for each order. Modify the SELECT statement so it uses a named window for the two aggregate functions.

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

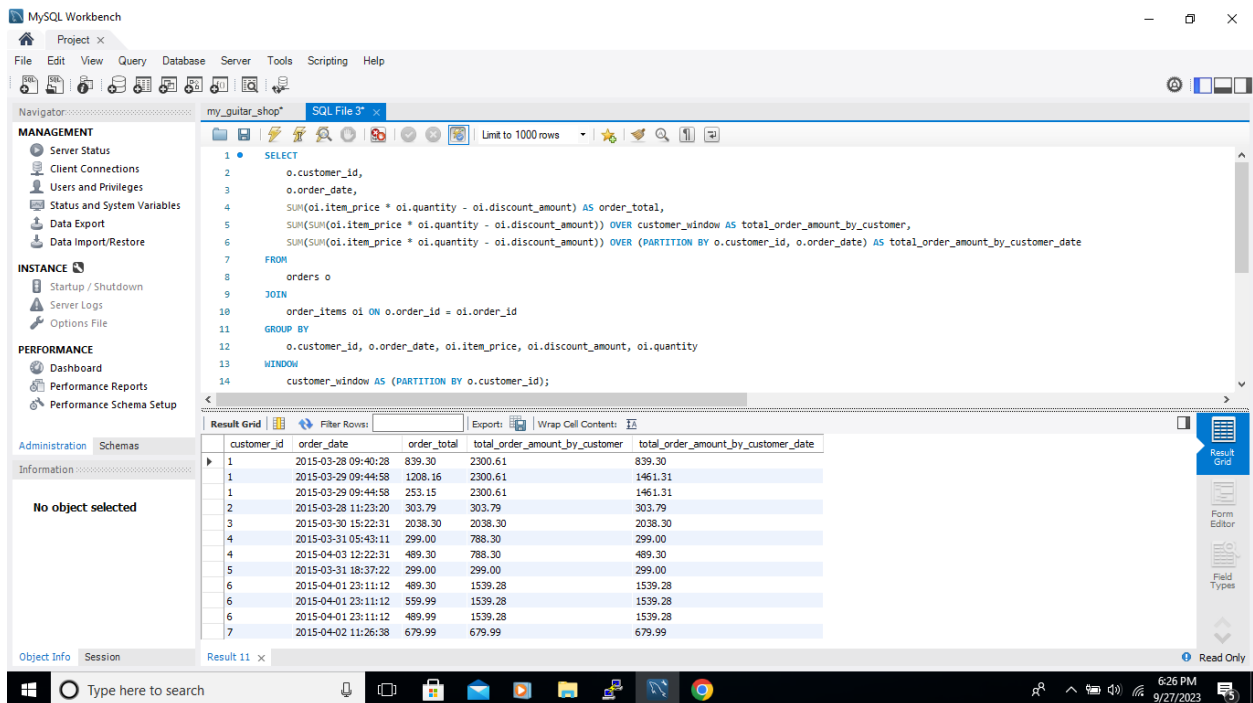
```
1 SELECT
2   oi.order_id,
3   (SUM(oi.item_price - oi.discount_amount) * oi.quantity) AS total_amount_per_item,
4   SUM(SUM(oi.item_price - oi.discount_amount) * oi.quantity) OVER total_amount_window AS cumulative_total_per_order,
5   AVG(SUM(oi.item_price - oi.discount_amount) * oi.quantity) OVER total_amount_window AS average_amount_per_order
6 FROM
7   order_items oi
8 GROUP BY
9   oi.order_id
10 WINDOW
11   total_amount_window AS (PARTITION BY oi.order_id)
12 ORDER BY
13   oi.order_id ASC
```

The Results Grid shows the following data:

order_id	total_amount_per_item	cumulative_total_per_order	average_amount_per_order
1	839.30	839.30	839.300000
2	303.79	303.79	303.790000
3	1461.31	1461.31	1461.310000
4	1678.60	1678.60	1678.600000
5	299.00	299.00	299.000000
6	299.00	299.00	299.000000
7	1539.28	1539.28	1539.280000
8	679.99	679.99	679.990000
9	489.30	489.30	489.300000

The interface also shows the left-hand navigation pane with sections for MANAGEMENT, INSTANCE, and PERFORMANCE. The bottom status bar indicates the session is 'Result 10' and the window is 'Read Only'.

Step 11: Write a SELECT statement that uses aggregate window functions to calculate the order total for each customer and the order total for each customer by date. Return these columns: The customer_id column from the Orders table. The order_date column from the Orders table. The total amount for each order item in the Order_Items table. The sum of the order totals for each customer. The sum of the order totals for each customer by date (Hint: You can create a peer group to get these values).



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

```

1  SELECT
2      o.customer_id,
3      o.order_date,
4      SUM(oi.item_price * oi.quantity - oi.discount_amount) AS order_total,
5      SUM(SUM(oi.item_price * oi.quantity - oi.discount_amount)) OVER customer_window AS total_order_amount_by_customer,
6      SUM(SUM(oi.item_price * oi.quantity - oi.discount_amount)) OVER (PARTITION BY o.customer_id, o.order_date) AS total_order_amount_by_customer_date
7  FROM
8      orders o
9  JOIN
10     order_items oi ON o.order_id = oi.order_id
11  GROUP BY
12     o.customer_id, o.order_date, oi.item_price, oi.discount_amount, oi.quantity
13  WINDOW
14     customer_window AS (PARTITION BY o.customer_id);

```

The Results window displays the following data:

	customer_id	order_date	order_total	total_order_amount_by_customer	total_order_amount_by_customer_date
1	1	2015-03-28 09:40:28	839.30	2300.61	839.30
1	1	2015-03-29 09:44:58	1208.16	2300.61	1461.31
1	1	2015-03-29 09:44:58	253.15	2300.61	1461.31
2	2	2015-03-28 11:23:20	303.79	303.79	303.79
3	3	2015-03-30 15:22:31	2038.30	2038.30	2038.30
4	4	2015-03-31 05:43:11	299.00	788.30	299.00
4	4	2015-04-03 12:22:31	489.30	788.30	489.30
5	5	2015-03-31 18:37:22	299.00	299.00	299.00
6	6	2015-04-01 23:11:12	489.30	1539.28	1539.28
6	6	2015-04-01 23:11:12	559.99	1539.28	1539.28
6	6	2015-04-01 23:11:12	489.99	1539.28	1539.28
7	7	2015-04-02 11:26:38	679.99	679.99	679.99