Lab Assignment 3

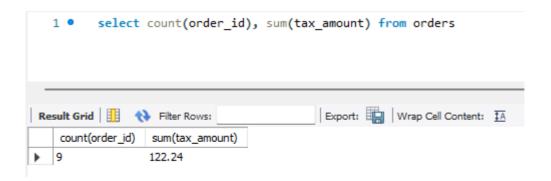
Exercises

We assume you have already installed MySQL server and Workbench, as well as the My Guitar Shop database with original data. In these exercises, you'll use MySQL Workbench to work on the My Guitar Shop database and submit queries to this database.

Code summary queries

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



2. Write a SELECT statement that returns one row for each category that has products with these columns:

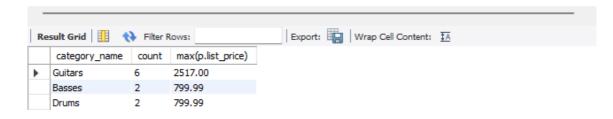
The category name column from the Categories table

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.

- 1 select c.category_name, count(p.product_id) as count, max(p.list_price)
- from categories as c inner join products as p on c.category_id = p.category_id
- 3 group by c.category_name
- 4 order by count desc



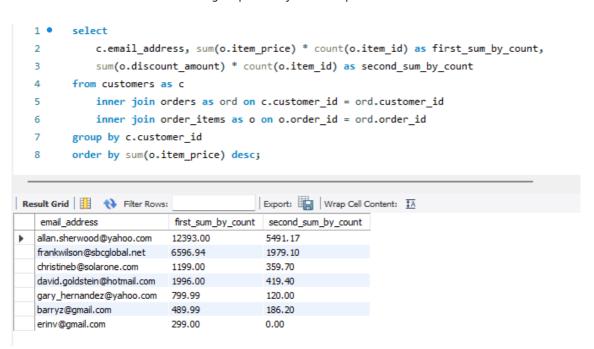
3. Write a SELECT statement that returns one row for each customer that has orders with these columns:

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.



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

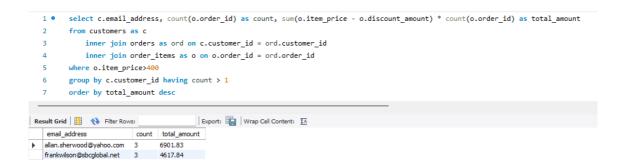
The total amount for each order (Hint: First, subtract the discount amount from the price.

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.

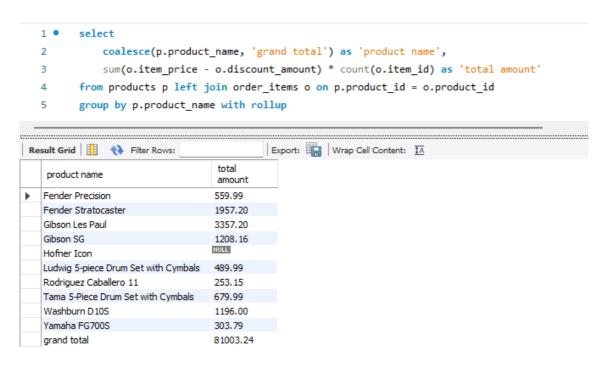
5. Modify the solution to exercise 4 so it only counts and totals line items that have an item_price value that's greater than 400.



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 Use the WITH ROLLUP operator to include a row that gives the grand total.



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.

```
select c.email_address, count(distinct oi.product_id) as count
  2
        from customers as c
             inner join orders o on c.customer_id = o.customer_id
  3
             inner join order_items oi on o.order_id = oi.order_id
  4
        group by c.email address having count > 1
  5
        order by email address asc
                                         Export: Wrap Cell Content: $\overline{1}{4}
email_address
                          count
  allan.sherwood@yahoo.com
  david.goldstein@hotmail.com
                        2
  frankwilson@sbcglobal.net
```

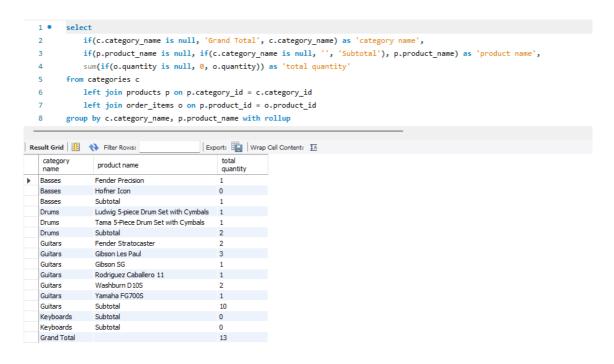
8. Write a SELECT statement that answers this question: What is the total quantity purchased for each product within each category? Return these columns:

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.



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

The total amount for each order

Sort the result set in ascending sequence by the order id column.

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.

Code subqueries

11. Write a SELECT statement that returns the same result set as this SELECT statement, but don't use a join. Instead, use a subquery in a WHERE clause that uses the IN keyword.

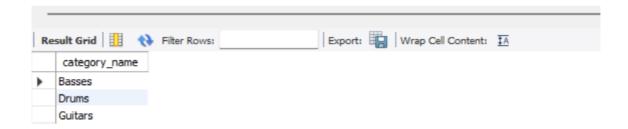
```
SELECT DISTINCT category_name

FROM categories c JOIN products p

ON c.category_id = p.category_id

ORDER BY category_name
```

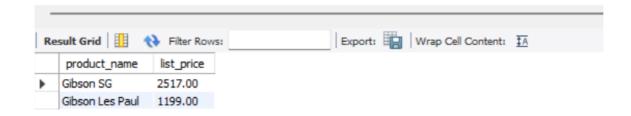
- 1 select distinct category_name
- 2 from categories c
- 3 where c.category_id in(select p.category_id from products p)
- 4 order by category_name



12. Write a SELECT statement that answers this question: Which products have a list price that's greater than the average list price for all products?

Return the product_name and list_price columns for each product. Sort the result set by the list price column in descending sequence.

- 1 select product_name,list_price
- 2 from products
- 3 where list_price > (select avg(list_price) from products)
- 4 order by list price desc



13. Write a SELECT statement that returns the category_name column from the Categories table.

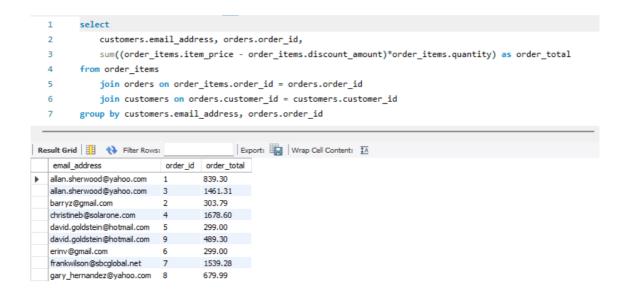
Return one row for each category that has never been assigned to any product in the Products

To do that, use a subquery introduced with the NOT EXISTS operator.

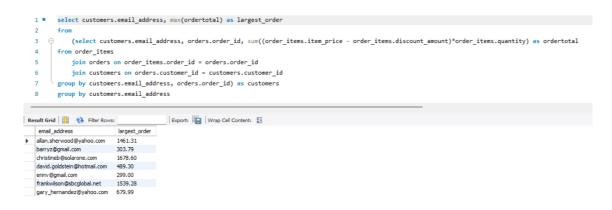
- 1 select c.category_name
- 2 from categories c
- 3 where not exists (select 1 from products as p where p.category_id = c.category_id)



14. Write a SELECT statement that returns three columns: email_address, order_id, and the order total for each customer. To do this, you can group the result set by the email_address and order_id columns. In addition, you must calculate the order total from the columns in the Order_Items table.

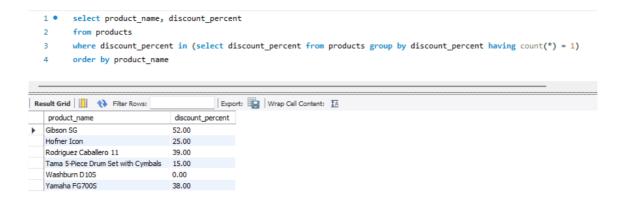


Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the email_address. Sort the result set by the largest order in descending sequence.



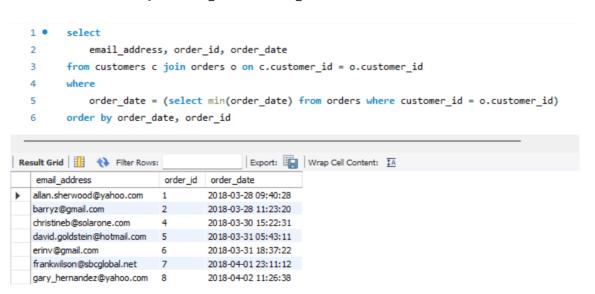
15. Write a SELECT statement that returns the name and discount percent of each product that has a unique discount percent. In other words, don't include products that have the same discount percent as another product.

Sort the result set by the product_name column.



16. Use a correlated subquery to return one row per customer, representing the customer's oldest order (the one with the earliest date). Each row should include these three columns: email_address, order_id, and order_date.

Sort the result set by the order date and order id columns.



Process data types

17. Write a SELECT statement that returns these columns from the Products table:

The list price column

A column that uses the FORMAT function to return the list_price column with 1 digit to the right of the decimal point

A column that uses the CONVERT function to return the list_price column as an integer A column that uses the CAST function to return the list_price column as an integer.

```
1 • select
2     list_price,
3     format(list_price, 1),
4     convert( list_price, unsigned),
5     cast(list_price as unsigned)
6     from products
```

Result Grid				
	list_price	format(list_price, 1)	convert(list_price, unsigned)	cast(list_price as unsigned)
•	699.00	699.0	699	699
	1199.00	1,199.0	1199	1199
	2517.00	2,517.0	2517	2517
	489.99	490.0	490	490
	299.00	299.0	299	299
	415.00	415.0	415	415
	799.99	800.0	800	800
	499.99	500.0	500	500
	699.99	700.0	700	700
	799.99	800.0	800	800

18. Write a SELECT statement that returns these columns from the Products table:

The date_added column

A column that uses the CAST function to return the date_added column with its date only (year, month, and day)

A column that uses the CAST function to return the date_added column with just the year and the month

A column that uses the CAST function to return the date_added column with its full time only (hour, minutes, and seconds).

```
1 •
        select
  2
              date_added,
              cast(date_added as date),
  3
              cast(date_added as char (7)),
  4
              cast(date_added as time)
         from products
Export: Wrap Cell Content: 1A
                       cast(date_added as
                                              cast(date_added as char
                                                                        cast(date_added as
   date_added
                       date)
                                              (7))
                                                                        time)
  2017-10-30 09:32:40
                      2017-10-30
                                             2017-10
                                                                       09:32:40
   2017-12-05 16:33:13 2017-12-05
                                             2017-12
                                                                       16:33:13
   2018-02-04 11:04:31 2018-02-04
                                             2018-02
                                                                       11:04:31
   2018-06-01 11:12:59 2018-06-01
                                             2018-06
                                                                       11:12:59
   2018-07-30 13:58:35 2018-07-30
                                             2018-07
                                                                       13:58:35
   2018-07-30 14:12:41 2018-07-30
                                             2018-07
                                                                       14:12:41
   2018-06-01 11:29:35 2018-06-01
                                             2018-06
                                                                       11:29:35
   2018-07-30 14:18:33 2018-07-30
                                             2018-07
                                                                       14:18:33
   2018-07-30 12:46:40 2018-07-30
                                             2018-07
                                                                       12:46:40
```

Use functions

19. Write a SELECT statement that returns these columns from the Products table:

The list_price column

The discount percent column

2018-07-30 13:14:15 2018-07-30

A column named discount_amount that uses the previous two columns to calculate the discount amount and uses the ROUND function to round the result so it has 2 decimal digits.

2018-07

13:14:15

```
1 •
        select
            list price,
  2
            discount percent,
  3
            round(list_price * discount_percent / 100, 2) as discount_amount
  4
  5
        from products
                                        Export: Wrap Cell Content: 1A
list_price discount_percent
                         discount_amount
  699.00
          30.00
                         209.70
  1199.00
          30.00
                         359.70
  2517.00
                         1308.84
          52.00
```

20. Write a SELECT statement that returns these columns from the Orders table:

186.20

161.85

240.00

125.00

210.00

120.00

0.00

The order_date column

38.00

0.00

39.00

30.00

25.00

30.00

15.00

489.99

299.00

415.00

799.99

499.99

699.99

799.99

A column that uses the DATE_FORMAT function to return the four-digit year that's stored in the order_date column

A column that uses the DATE_FORMAT function to return the order_date column in this format: Mon-DD-YYYY. In other words, use abbreviated months and separate each date component with dashes.

A column that uses the DATE_FORMAT function to return the order_date column with only the hours and minutes on a 12-hour clock with an am/pm indicator A column that uses the DATE_FORMAT function to return the order_date column in this format: MM/DD/YY HH:SS. In other words, use two-digit months, days, and years and separate them by slashes. Use 2-digit hours and minutes on a 24-hour clock. And use leading zeros for all date/time components.

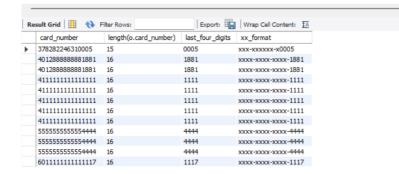
```
1 •
         select
              date_format(order_date, '%Y') as first_format,
  2
              date_format(order_date, '%b-%e-%y') as second_format,
  3
              date format(order date, '%1:%i %p') as third format,
  4
              date_format(order_date, '%m/%e/%y %h:%i') as fourth_format
  5
  6
          from orders
                                               Export: Wrap Cell Content: IA
Result Grid
               Filter Rows:
   first_format
               second_format
                                           fourth_format
                              third_format
                                           03/28/18 09:40
  2018
               Mar-28-18
                              9:40 AM
  2018
               Mar-28-18
                              11:23 AM
                                           03/28/18 11:23
  2018
               Mar-29-18
                                           03/29/18 09:44
                              9:44 AM
  2018
                                           03/30/18 03:22
               Mar-30-18
                              3:22 PM
  2018
                                           03/31/18 05:43
               Mar-31-18
                              5:43 AM
  2018
               Mar-31-18
                              6:37 PM
                                           03/31/18 06:37
  2018
                                           04/1/18 11:11
               Apr-1-18
                              11:11 PM
  2018
               Apr-2-18
                              11:26 AM
                                           04/2/18 11:26
                                           04/3/18 12:22
  2018
               Apr-3-18
                              12:22 PM
```

21. Write a SELECT statement that returns these columns from the Orders table:

The card_number column

The length of the card_number column

The last four digits of the card_number column



22. Write a SELECT statement that returns these columns from the Orders table:

The order_id column

The order date column

A column named approx_ship_date that's calculated by adding 2 days to the order_date column

The ship date column if it doesn't contain a null value

A column named days_to_ship that shows the number of days between the order date and the ship date

When you have this working, add a WHERE clause that retrieves just the orders for March 2018.

```
select order_id, order_date,

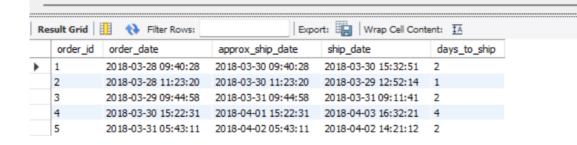
date_add(order_date, interval 2 day) as 'approx_ship_date', ship_date,

datediff(ship_date, order_date) as 'days_to_ship'

from orders where ship_date is not null

and month(order_date) = '3'

and year(order_date) = '2018'
```



23. Write a SELECT statement that uses regular expression functions to get the username and domain name parts of the email addresses in the Administrators table. Return these columns:

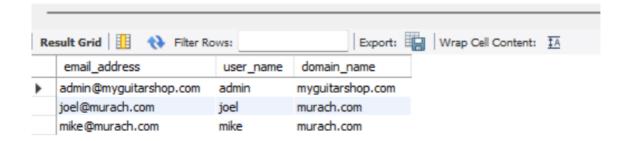
The email_address column

A column named user_name that contains the username part of the email_address column (the part before the @ symbol)

A column named domain_name that contains the domain name part of the email_address column (the part after the @ symbol)

Note: The username part of the email addresses contains only letters, and the domain name part contains only letters and a period.

```
select email_address,
substring_index(email_address, '@', 1) as user_name,
substring_index(email_address, '@', -1) as domain_name
from administrators
```



24. Write a SELECT statement that uses the ranking functions to rank products by the total quantity sold. Return these columns:

The product_name column from the Products table

A column named total_quantity that shows the sum of the quantity for each product in the Order_Items table

A column named rank that uses the RANK function to rank the total quantity in descending sequence

A column named dense_rank that uses the DENSE_RANK function to rank the total quantity in descending sequence

```
select products.product_name, sum(quantity) as total_quantity,
  1
  2
         rank() over (order by sum(quantity) desc) 'rank',
  3
         dense_rank() over (order by sum(quantity) desc) 'dense_rank'
         from products inner join order_items on products.product_id=order_items.product_id
  4
  5
         group by products.product_id
Export: Wrap Cell Content: IA
                                  total_quantity rank dense_rank
   product_name
  Gibson Les Paul
                                 3
                                               1
                                                     1
  Washburn D10S
  Fender Stratocaster
                                               2
  Yamaha FG700S
                                              4
                                                     3
  Gibson SG
                                 1
                                              4
                                                     3
  Rodriguez Caballero 11
                                 1
                                              4
                                                    3
  Fender Precision
                                 1
                                              4
                                                     3
  Ludwig 5-piece Drum Set with Cymbals
                                 1
                                              4
                                                    3
  Tama 5-Piece Drum Set with Cymbals
                                 1
                                                     3
```

25. Write a SELECT statement that uses the analytic functions to get the highest and lowest sales by product within each category. Return these columns:

The category name column from the Categories table

The product_name column from the Products table

A column named total_sales that shows the sum of the sales for each product with sales in the Order_Items table

A column named highest_sales that uses the FIRST_VALUE function to show the name of the product with the highest sales within each category

A column named lowest_sales that uses the LAST_VALUE function to show the name of the product with the lowest sales within each category.

