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

```
select count(order_id), sum(tax_amount) from orders
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	count(order_id)	sum(tax_amount)
▶	9	122.24

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.

```
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
group by c.category_name
order by count desc
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	category_name	count	max(p.list_price)
▶	Guitars	6	2517.00
	Basses	2	799.99
	Drums	2	799.99

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.

```
select
    c.email_address, sum(o.item_price) * count(o.item_id) as first_sum_by_count,
    sum(o.discount_amount) * count(o.item_id) as second_sum_by_count
from customers as c
    inner join orders as ord on c.customer_id = ord.customer_id
    inner join order_items as o on o.order_id = ord.order_id
group by c.customer_id
order by sum(o.item_price) desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	first_sum_by_count	second_sum_by_count
allan.sherwood@yahoo.com	12393.00	5491.17
frankwilson@sbcglobal.net	6596.94	1979.10
christineb@solarone.com	1199.00	359.70
david.goldstein@hotmail.com	1996.00	419.40
gary_hernandez@yahoo.com	799.99	120.00
barryz@gmail.com	489.99	186.20
erinv@gmail.com	299.00	0.00

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.

```
select c.email_address, count(o.order_id) as count, sum(o.item_price - o.discount_amount) * count(o.order_id) as total_amount
from customers as c
    inner join orders as ord on c.customer_id = ord.customer_id
    inner join order_items as o on o.order_id = ord.order_id
group by c.customer_id having count > 1
order by total_amount desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	count	total_amount
allan.sherwood@yahoo.com	3	6901.83
frankwilson@sbcglobal.net	3	4617.84
david.goldstein@hotmail.com	2	1576.60

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.

```
select c.email_address, count(o.order_id) as count, sum(o.item_price - o.discount_amount) * count(o.order_id) as total_amount
from customers as c
    inner join orders as ord on c.customer_id = ord.customer_id
    inner join order_items as o on o.order_id = ord.order_id
where o.item_price>400
group by c.customer_id having count > 1
order by total_amount desc
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	count	total_amount
allan.sherwood@yahoo.com	3	6901.83
frankwilson@sbcglobal.net	3	4617.84

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.

```
select
    coalesce(p.product_name, 'grand total') as 'product name',
    sum(o.item_price - o.discount_amount) * count(o.item_id) as 'total amount'
from products p left join order_items o on p.product_id = o.product_id
group by p.product_name with rollup
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product name	total amount
Fender Precision	559.99
Fender Stratocaster	1957.20
Gibson Les Paul	3357.20
Gibson SG	1208.16
Hofner Icon	NULL
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	1196.00
Yamaha FG700S	303.79
grand total	81003.24

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
from customers as 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 email_address asc
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

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.

```
select
    if(c.category_name is null, 'Grand Total', c.category_name) as 'category name',
    if(p.product_name is null, if(c.category_name is null, '', 'Subtotal'), p.product_name) as 'product name',
    sum(if(o.quantity is null, 0, o.quantity)) as 'total quantity'
from categories c
    left join products p on p.category_id = c.category_id
    left join order_items o on p.product_id = o.product_id
group by c.category_name, p.product_name with rollup
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category name	product name	total quantity
Basses	Fender Precision	1
Basses	Hofner Icon	0
Basses	Subtotal	1
Drums	Ludwig 5-piece Drum Set with Cymbals	1
Drums	Tama 5-Piece Drum Set with Cymbals	1
Drums	Subtotal	2
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
Guitars	Subtotal	10
Keyboards	Subtotal	0
Keyboards	Subtotal	0
Grand Total		13

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
```

```
select distinct category_name
from categories c
where c.category_id in(select p.category_id from products p)
order by category_name
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category_name
Basses
Drums
Guitars

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.

```
select product_name,list_price
from products
where list_price > (select avg(list_price) from products)
order by list_price desc
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	list_price
Gibson SG	2517.00
Gibson Les Paul	1199.00

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

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

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category_name
Keyboards

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.

```
select
    customers.email_address, orders.order_id,
    sum((order_items.item_price - order_items.discount_amount)*order_items.quantity) as order_total
from order_items
    join orders on order_items.order_id = orders.order_id
    join customers on orders.customer_id = customers.customer_id
group by customers.email_address, orders.order_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	order_id	order_total
allan.sherwood@yahoo.com	1	839.30
allan.sherwood@yahoo.com	3	1461.31
barryz@gmail.com	2	303.79
christineb@solarone.com	4	1678.60
david.goldstein@hotmail.com	5	299.00
david.goldstein@hotmail.com	9	489.30
erinv@gmail.com	6	299.00
frankwilson@sbcglobal.net	7	1539.28
gary_hernandez@yahoo.com	8	679.99

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.

```
select customers.email_address, max(ordertotal) as largest_order
from
    (select customers.email_address, orders.order_id, sum((order_items.item_price - order_items.discount_amount)*order_items.quantity) as ordertotal
from order_items
    join orders on order_items.order_id = orders.order_id
    join customers on orders.customer_id = customers.customer_id
group by customers.email_address, orders.order_id) as customers
group by customers.email_address
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	largest_order
allan.sherwood@yahoo.com	1461.31
barryz@gmail.com	303.79
christineb@solarone.com	1678.60
david.goldstein@hotmail.com	489.30
erinv@gmail.com	299.00
frankwilson@sbcglobal.net	1539.28
gary_hernandez@yahoo.com	679.99

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.

```
select product_name, discount_percent
from products
where discount_percent in (select discount_percent from products group by discount_percent having count(*) = 1)
order by product_name
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	discount_percent
Gibson SG	52.00
Hofner Icon	25.00
Rodriguez Caballero 11	39.00
Tama 5-Piece Drum Set with Cymbals	15.00
Washburn D10S	0.00
Yamaha FG700S	38.00

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.

```
select
    email_address, order_id, order_date
from customers c join orders o on c.customer_id = o.customer_id
where
    order_date = (select min(order_date) from orders where customer_id = o.customer_id)
order by order_date, order_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	order_id	order_date
allan.sherwood@yahoo.com	1	2018-03-28 09:40:28
barryz@gmail.com	2	2018-03-28 11:23:20
christineb@solarone.com	4	2018-03-30 15:22:31
david.goldstein@hotmail.com	5	2018-03-31 05:43:11
erinv@gmail.com	6	2018-03-31 18:37:22
frankwilson@sbcglobal.net	7	2018-04-01 23:11:12
gary_hernandez@yahoo.com	8	2018-04-02 11:26:38

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.

```
select
    list_price,
    format(list_price, 1),
    convert( list_price, unsigned),
    cast(list_price as unsigned)
from products
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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).

```sql
select
    date_added,
    cast(date_added as date),
    cast(date_added as char (7)),
    cast(date_added as time)
from products
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

date_added	cast(date_added as date)	cast(date_added as char (7))	cast(date_added as 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
2018-07-30 13:14:15	2018-07-30	2018-07	13:14:15

Use functions

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

    The list_price column
    The discount_percent column
    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.

```
select
    list_price,
    discount_percent,
    round(list_price * discount_percent / 100, 2) as discount_amount
from products
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

list_price	discount_percent	discount_amount
699.00	30.00	209.70
1199.00	30.00	359.70
2517.00	52.00	1308.84
489.99	38.00	186.20
299.00	0.00	0.00
415.00	39.00	161.85
799.99	30.00	240.00
499.99	25.00	125.00
699.99	30.00	210.00
799.99	15.00	120.00

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

The order_date column
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.

```
select
    date_format(order_date, '%Y') as first_format,
    date_format(order_date, '%b-%e-%y') as second_format,
    date_format(order_date, '%l:%i %p') as third_format,
    date_format(order_date, '%m/%e/%y %h:%i') as fourth_format
from orders
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_format	second_format	third_format	fourth_format
2018	Mar-28-18	9:40 AM	03/28/18 09:40
2018	Mar-28-18	11:23 AM	03/28/18 11:23
2018	Mar-29-18	9:44 AM	03/29/18 09:44
2018	Mar-30-18	3:22 PM	03/30/18 03:22
2018	Mar-31-18	5:43 AM	03/31/18 05:43
2018	Mar-31-18	6:37 PM	03/31/18 06:37
2018	Apr-1-18	11:11 PM	04/1/18 11:11
2018	Apr-2-18	11:26 AM	04/2/18 11:26
2018	Apr-3-18	12:22 PM	04/3/18 12:22

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
A column that displays an X for each digit of the card_number column except for the last four digits. If the card number contains 16 digits, it should be displayed in this format: XXXX-XXXX-XXXX-1234, where 1234 are the actual last four digits of the number. If the card number contains 15 digits, it should be displayed in this format: XXXX-XXXXXX-X1234. (Hint: Use an IF function to determine which format to use.)

```
select
    o.card_number,
    length(o.card_number),
    right(o.card_number, 4) as last_four_digits,
    insert(right(o.card_number, 4), 1, 0, if(length(o.card_number)=16, 'xxxx-xxxx-xxxx-', 'xxx-xxxxxx-x')) as xx_format
from orders o
    inner join order_items oi on o.order_id = oi.order_id
    inner join products p on p.product_id = oi.product_id
order by o.card_number
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

card_number	length(o.card_number)	last_four_digits	xx_format
378282246310005	15	0005	xxx-xxxxxx-x0005
4012888888881881	16	1881	xxxx-xxxx-xxxx-1881
4012888888881881	16	1881	xxxx-xxxx-xxxx-1881
4111111111111111	16	1111	xxxx-xxxx-xxxx-1111
4111111111111111	16	1111	xxxx-xxxx-xxxx-1111
4111111111111111	16	1111	xxxx-xxxx-xxxx-1111
4111111111111111	16	1111	xxxx-xxxx-xxxx-1111
4111111111111111	16	1111	xxxx-xxxx-xxxx-1111
5555555555554444	16	4444	xxxx-xxxx-xxxx-4444
5555555555554444	16	4444	xxxx-xxxx-xxxx-4444
5555555555554444	16	4444	xxxx-xxxx-xxxx-4444
6011111111111117	16	1117	xxxx-xxxx-xxxx-1117

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'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	order_date	approx_ship_date	ship_date	days_to_ship
1	2018-03-28 09:40:28	2018-03-30 09:40:28	2018-03-30 15:32:51	2
2	2018-03-28 11:23:20	2018-03-30 11:23:20	2018-03-29 12:52:14	1
3	2018-03-29 09:44:58	2018-03-31 09:44:58	2018-03-31 09:11:41	2
4	2018-03-30 15:22:31	2018-04-01 15:22:31	2018-04-03 16:32:21	4
5	2018-03-31 05:43:11	2018-04-02 05:43:11	2018-04-02 14:21:12	2

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
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_address	user_name	domain_name
admin@myguitarshop.com	admin	myguitarshop.com
joel@murach.com	joel	murach.com
mike@murach.com	mike	murach.com

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

```sql
select products.product_name, sum(quantity) as total_quantity,
rank() over (order by sum(quantity) desc) 'rank',
dense_rank() over (order by sum(quantity) desc) 'dense_rank'
from products inner join order_items on products.product_id=order_items.product_id
group by products.product_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	total_quantity	rank	dense_rank
Gibson Les Paul	3	1	1
Washburn D10S	2	2	2
Fender Stratocaster	2	2	2
Yamaha FG700S	1	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	4	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.

```sql
select c.category_name, p.product_name,
    sum((oi.item_price+oi.discount_amount)*oi.quantity) as total_sales,
    first_value(p.product_name) over(partition by c.category_name order by sum((oi.item_price+oi.discount_amount)*oi.quantity) desc) as highest_sales,
    last_value(p.product_name) over(partition by c.category_name order by sum((oi.item_price+oi.discount_amount)*oi.quantity) desc) as lowest_sales
from (categories c join products p on c.category_id = p.category_id)
    join order_items oi on oi.product_id = p.product_id
group by c.category_id, c.category_name, p.product_id, p.product_name
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category_name	product_name	total_sales	highest_sales	lowest_sales
Basses	Fender Precision	1039.99	Fender Precision	Fender Precision
Drums	Tama 5-Piece Drum Set with Cymbals	919.99	Tama 5-Piece Drum Set with Cymbals	Tama 5-Piece Drum Set with Cymbals
Drums	Ludwig 5-piece Drum Set with Cymbals	909.99	Tama 5-Piece Drum Set with Cymbals	Ludwig 5-piece Drum Set with Cymbals
Guitars	Gibson Les Paul	4676.10	Gibson Les Paul	Gibson Les Paul
Guitars	Gibson SG	3825.84	Gibson Les Paul	Gibson SG
Guitars	Fender Stratocaster	1817.40	Gibson Les Paul	Fender Stratocaster
Guitars	Yamaha FG700S	676.19	Gibson Les Paul	Yamaha FG700S
Guitars	Washburn D10S	598.00	Gibson Les Paul	Washburn D10S
Guitars	Rodriguez Caballero 11	576.85	Gibson Les Paul	Rodriguez Caballero 11