CSI 300: Problem Set 3

For this problem set you’ll be using the Murach provided database created from

the script “create\_my\_guitar\_shop.sql” that you can find on Canvas. Run that

script before you attempt any of the exercises below.

David Thomsen did Even-Number Queries, and Micah Kezar did Odd-Number Queries but overall the work was done as a group.

**1. Write a SELECT statement that joins the Categories table to the Products**

**table and returns these columns:**

a. category\_name

b. product\_name

c. list\_price

Sort the result set by category\_name and then by product\_name in

ascending sequence.

SELECT cat.category\_name, prod.product\_name, prod.list\_price

FROM Categories cat

JOIN Products prod ON cat.category\_id = prod.category\_id

ORDER BY

cat.category\_name ASC,

prod.product\_name ASC;



**2. Write a SELECT statement that joins the Customers, Orders, Order\_Items,**

**and Products tables. This statement should return these columns:**

a. last\_name

b. first\_name

c. order\_date

d. product\_name

e. item\_price

f. discount\_amount

g. Quantity

Use aliases of your choice for the tables. Sort the final result set by

last\_name, order\_date, and product\_name.

use my\_guitar\_shop;

select

c.last\_name,

c.first\_name,

o.order\_date,

p.product\_name,

oi.item\_price,

oi.discount\_amount,

oi.quantity

from

customers c

join

orders o

on c.customer\_id = o.customer\_id

join

order\_items oi

on o.order\_id = oi.order\_id

join

products p

on oi.product\_id = p.product\_id

order by

c.last\_name,

o.order\_date,

p.product\_name;



**3. Write a SELECT statement that returns the product\_name and list\_price**

**columns from the Products table.** Return one row for each product that has

the same list price as another product. (Hint: Use a self-join to check that

the product\_id columns aren’t equal but the list\_price columns are equal).

Sort the result set by product\_name.

SELECT prod1.product\_name, prod1.list\_price

FROM Products prod1 JOIN Products prod2

ON prod1.list\_price = prod2.list\_price

AND prod1.product\_id <> prod2.product\_id

ORDER BY

prod1.product\_name;



**4. Write a SELECT statement that returns these two columns:**

a. category\_name (The category\_name column from the Categories

table)

b. product\_id (The product\_id column from the Products table)

Return one row for each category that has never been used. Hint: Use

an **outer join** and only return rows where the **product\_id column**

**contains a null value.**

(Using OUTER JOIN was throwing errors so a LEFT JOIN was used instead)

use my\_guitar\_shop;

select

c.category\_name,

p.product\_id

from categories c

left join products p

on c.category\_id = p.category\_id

where

p.product\_id IS NULL;



**5. Write a SELECT statement that returns these columns:**

a. The count of the number of orders in the Orders table

B. The sum of the tax\_amount columns in the Orders table

SELECT COUNT(\*) AS order\_amount, SUM(tax\_amount) AS total\_tax\_amount

FROM Orders;



**6. Write a SELECT statement that returns one row for each customer that has**

**orders with these columns:**

a. The email\_address from the Customers table

b. count of the number of orders

c. 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.

use my\_guitar\_shop;

select

c.email\_address,

count(distinct o.order\_id) as orders,

sum((oi.item\_price - oi.discount\_amount) \* oi.quantity) as total

from

customers c

join

orders o

on c.customer\_id = o.customer\_id

join

order\_items oi

on o.order\_id = oi.order\_id

group by

c.email\_address

having

count(distinct o.order\_id) > 1

order by

total desc;



**7. Write a SELECT statement that answers this question: Which customers**

**have ordered more than one product? Return these columns:**

a. The email address from the Customers table

b. The count of distinct products from the customer’s orders

SELECT cust.email\_address, COUNT(DISTINCT ordit.product\_id) AS distinct\_product\_count

FROM Customers cust

JOIN Orders ord ON cust.customer\_id = ord.customer\_id

JOIN Order\_Items ordit ON ord.order\_id = ordit.order\_id

GROUP BY cust.customer\_id, cust.email\_address

HAVING COUNT(DISTINCT ordit.product\_id) > 1

ORDER BY distinct\_product\_count ASC;

/\* Wasn't required, but not organized numbers bother me lol \*/



8. **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

use my\_guitar\_shop;

select distinct category\_name

from categories

where category\_id in (

select distinct category\_id

from products)

order by

category\_name;

/\*replaces the join command with a select where statement in order to gather the same information rather than a join\*/



**9. Write a SELECT statement that returns three columns:**

a. email\_address

b. order\_id

c. the order total for each customer

SELECT cust.email\_address, ordit.order\_id, SUM(ordit.quantity \* ordit.item\_price) AS order\_total

FROM Customers cust

JOIN Orders ord ON cust.customer\_id = ord.customer\_id

JOIN Order\_Items ordit ON ord.order\_id = ordit.order\_id

GROUP BY cust.email\_address, ordit.order\_id

ORDER BY ord.order\_id ASC;



**Write a second SELECT statement that uses the first SELECT statement in its**

**FROM clause. The main query should return two columns:**

a. the customer’s email address

b. the largest order for that customer.

To do this, you can group the result set by the email\_address.

/\* I was confused about whether or not you wanted the specific order\_id that was the largest, or the specific value of the largest order\_id, so you get both \*/

SELECT email\_address, MAX(order\_id) AS largest\_order\_id, SUM(order\_total)

AS largest\_order\_total

FROM (

SELECT cust.email\_address, ordit.order\_id, SUM(ordit.quantity \* ordit.item\_price)

AS order\_total

FROM Customers cust

JOIN Orders ord ON cust.customer\_id = ord.customer\_id

JOIN Order\_Items ordit ON ord.order\_id = ordit.order\_id

GROUP BY cust.email\_address, ordit.order\_id

) AS sub

GROUP BY email\_address;

**10. Use the UNION operator to generate a result set consisting of three**

**columns from the Orders table:**

a. ship\_status

b. order\_id

c. order\_date

d. A calculated column that contains a value of SHIPPED or NOT

SHIPPED.

Sort the final result set by order\_date.

use my\_guitar\_shop;

select

order\_id,

order\_date,

'SHIPPED' AS `Shipping Status`

from orders

where ship\_date IS NOT NULL

union

/\*SHIPPED^^\*/

select

order\_id,

order\_date,

'NOT SHIPPED' AS `Shipping Status`

from orders

where ship\_date IS NULL

order by

order\_date;

/\*NOT SHIPPED^^\*/

