ISOM 671: Managing Big Data (Assignment 1)

Sean Jung

There are 20 numbered questions. Please answer them all and submit your assignment as a single PDF or Word file by uploading it to course canvas page. You should provide: SQL statements, results of the SQL statement (typically copy first 10 rows), and answers to questions, if any.

Pets Stackexchange (http://pets.stackexchange.com/) is a Q&A forum for pets. This assignment is based on data of this forum. As other Stackexchange-branded forums, the pet exchange lists questions by the number of votes, answers, and views. Questions can be tagged so that users can easily explore related questions. You can use the StackExchange data explorer (http://data.stackexchange.com/) to explore the data. For the first five questions, please use the data explorer for the Pets Stackexchange (http://data.stackexchange.com/pets/query/new). Alternatively, if you prefer, you can download a data dump and import the database to your local installation of MySQL.

1. Using the posts table, find out the number of posts, the minimum creation date (as min_date), the maximum creation date (as max_date), and average score (as avg_score).

```
SELECT COUNT(DISTINCT ID),
       min(CreationDate) as min_date,
       max(CreationDate) as max_date,
       AVG(Score) as avg_score
FROM Posts
```

	min_date	max_date	avg_score
17869	2013-10-08 21:29:51	2020-09-26 15:36:17	3

2. We want to get some ideas of how many posts were written each month. Use SQL to count the number of posts by year-month. Note that by year-month, we mean that May 2013 and May 2014 should be considered as different year-months. The result table should show year-month and count and order results by year-month.

```
SELECT
    count(*) as count,
    year(CreationDate) as year,
    month(CreationDate) as month
FROM Posts
GROUP BY year(CreationDate), month(CreationDate);
```

count	year	mont...
202	2017	5
245	2014	1
306	2014	7
164	2016	12
151	2020	3
155	2019	4
133	2018	9

3. We know that there are different types of posts, as reflected by the PostTypeID: the original posts (i.e., 1), follow up posts (i.e., 2), survey questions (i.e., 4), and unknown (i.e., 5). Please use SQL to get the number of posts by year-month and by post type. The results table should show: year, month, posttypeid, and count (label: cnt). Use this result to answer the following question: which year-month has the most original posts?

```
SELECT
      Year(CreationDate) AS year,
      Month(CreationDate) AS month,
      PostTypeId,
      COUNT(PostTypeId) as 'cnt'
FROM Posts
GROUP BY Year(CreationDate), Month(CreationDate), PostTypeId
ORDER BY Year(CreationDate), Month(CreationDate), PostTypeId;
```

year ▲	mont…	PostTypeI…	cn…
2013	10	1	350
2013	10	2	510
2013	10	4	73
2013	10	5	73
2013	11	1	114
2013	11	2	174

4. Popular badges. Pets Exchange has an elaborate badging system. Use the badges table to find the most common type of badges. Specifically, show badge name (label name) and the number of people who won it (label cnt), limiting results to badges with at least 20 winners and sort the results by cnt in a descending order.

```
SELECT
      Name AS label_name,
      COUNT(Id) AS label_count
FROM Badges
GROUP BY Name
HAVING count(Id) >= 20
ORDER BY count(Id) DESC;
```

label_name	label_count
Autobiographer	4639
Student	3878
Supporter	3238
Popular Question	2139
Teacher	1989
Editor	1469

5. (use the users table) Find the number of users who report being located in the New York state. These include people who report themselves in New York city or in the NY state. Be as accurate as possible as the self-reported location may take different forms. Report id, displayname, and location in your result set.

```
SELECT Id, DisplayName, Location

FROM Users

WHERE Location like 'N%Y%';
```

Id	DisplayName	Location
6448	Batbro	New York
6584	JesseTG	New York, NY, United States
6998	DVK	New York, NY
7031	Matthew Read	New York, NY, United States
7065	DIMMSum	New York, NY, United States
9037	Alan Thomas	NY, United States
9170	briantist	New York, United States

Please use the my_guitar_shop database provided on canvas to explore the data in your local MySQL instance and answer the next questions.

6. Write a SELECT statement that joins the Categories table to the Products table and returns these columns: category_name, product_name, list_price.
Sort the result set by category_name and then by product_name in ascending sequence.

```
SELECT a.category_name, b.product_name, b.list_price

FROM categories as a

INNER JOIN products as b

ON a.category_id = b.category_id

ORDER BY 1, 2 ASC;
```

category_name	product_name	list_price
Basses	Fender Precision	799.99
Basses	Hofner Icon	499.99
Drums	Ludwig 5-piece Drum Set with Cymbals	699.99
Drums	Tama 5-Piece Drum Set with Cymbals	799.99
Guitars	Fender Stratocaster	699.00
Guitars	Gibson Les Paul	1199.00

7. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first_name, last_name, line1, city, state, zip_code. Return one row for each address for the customer with an email address of allan.sherwood@yahoo.com.

```
SELECT b.first_name, b.last_name, a.line1, a.city, a.state, a.zip_code

FROM addresses as a

RIGHT JOIN customers as b

ON a.customer_id = b.customer_id

WHERE b.email_address = 'allan.sherwood@yahoo.com';
```

first_name	last_name	line1	city	state	zip_code
Allan	Sherwood	100 East Ridgewood Ave.	Paramus	NJ	07652
Allan	Sherwood	21 Rosewood Rd.	Woodcliff Lake	NJ	07677

8. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first_name, last_name, line1, city, state, zip_code. Return one row for each customer, but only return addresses that are the shipping address for a customer.

```
SELECT b.first_name, b.last_name, a.line1, a.city, a.state, a.zip_code

FROM addresses as a

JOIN customers as b

ON a.customer_id = b.customer_id

WHERE a.address_id = b.shipping_address_id

GROUP BY 1,2;
```

first_name	last_name	line1	city	state	zip_code
Allan	Sherwood	100 East Ridgewood Ave.	Paramus	NJ	07652
Barry	Zimmer	16285 Wendell St.	Omaha	NE	68135
Christine	Brown	19270 NW Cornell Rd.	Beaverton	OR	97006
David	Goldstein	186 Vermont St.	San Francisco	CA	94110
Erin	Valentino	6982 Palm Ave.	Fresno	CA	93711
Frank Lee	Wilson	23 Mountain View St.	Denver	CO	80208

9. Write a SELECT statement that joins the Customers, Orders, Order_Items, and Products tables. This statement should return these columns: last_name, first_name, order_date, product_name, item_price, discount_amount, and quantity. Use aliases for the tables. Sort the final result set by last_name, order_date, and product_name.

```
SELECT a.last_name, a.first_name, c.order_date, d.product_name, d.list_price,
b.discount_amount , b.quantity

FROM customers as a

INNER JOIN orders as c

ON a.customer_id = c.customer_id

INNER JOIN order_items as b

ON b.order_id = c.order_id

INNER JOIN products as d

ON b.product_id = d.product_id

ORDER BY 1, 3, 4;
```

last_name	first_name	order_date	product_name	list_price	discount_amount	quantity
Brown	Christine	2015-03-30 15:22:31	Gibson Les Paul	1199.00	359.70	2
Goldstein	David	2015-03-31 05:43:11	Washburn D10S	299.00	0.00	1
Goldstein	David	2015-04-03 12:22:31	Fender Stratocaster	699.00	209.70	1
Hernandez	Gary	2015-04-02 11:26:38	Tama 5-Piece Drum Set with Cymbals	799.99	120.00	1
Sherwood	Allan	2015-03-28 09:40:28	Gibson Les Paul	1199.00	359.70	1
Sherwood	Allan	2015-03-29 09:44:58	Gibson SG	2517.00	1308.84	1

10. 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. Sort the result set by product_name.

```
SELECT p.product_name, p.list_price

FROM products as p

INNER JOIN products as p1

ON p.list_price = p1.list_price

WHERE p.product_id <> p1.product_id

ORDER BY p.product_name;
```

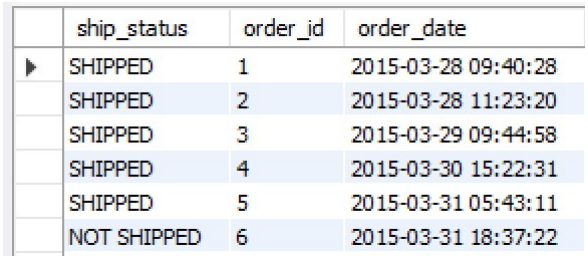
product_name	list_price
Fender Precision	799.99
Tama 5-Piece Drum Set with Cymbals	799.99

11. Use the UNION operator to generate a result set consisting of three columns from the Orders table:

- ship_status A calculated column that contains a value of SHIPPED or NOT SHIPPED
- order_id The order_id column
- order_date The order_date column

If the order has a value in the ship_date column, the ship_status column should contain a value of SHIPPED. Otherwise, it should contain a value of NOT SHIPPED. Sort the final result set by order_date.

```
SELECT 'SHIPPED' as ship_status, order_id, order_date

FROM orders

WHERE ship_date IS NOT NULL

UNION

SELECT 'NOT SHIPPED', order_id, order_date

FROM orders

WHERE ship_date IS NULL

ORDER BY order_date;
```



ship_status	order_id	order_date
SHIPPED	1	2015-03-28 09:40:28
SHIPPED	2	2015-03-28 11:23:20
SHIPPED	3	2015-03-29 09:44:58
SHIPPED	4	2015-03-30 15:22:31
SHIPPED	5	2015-03-31 05:43:11
NOT SHIPPED	6	2015-03-31 18:37:22

12. 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 a.category_name, count(b.product_id) as count, b.list_price

FROM categories as a

JOIN products as b

ON a.category_id = b.category_id

GROUP BY 1

ORDER BY 3 DESC;
```

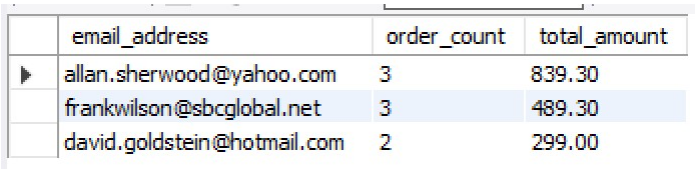
	category_name	count	list_price
▶	Basses	2	799.99
	Drums	2	699.99
	Guitars	6	699.00

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

- The email_address 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 a.email_address, count(b.order_id) as order_count, ((d.list_price - b.discount_amount)*b.quantity) as total_amount

FROM customers as a

JOIN orders as c

ON a.customer_id = c.customer_id

JOIN order_items as b

ON b.order_id = c.order_id

JOIN products as d

ON b.product_id = d.product_id

GROUP BY 1

HAVING order_count > 1

ORDER BY total_amount DESC;
```



email_address	order_count	total_amount
allan.sherwood@yahoo.com	3	839.30
frankwilson@sbcglobal.net	3	489.30
david.goldstein@hotmail.com	2	299.00

14. Write a SELECT statement that answers this question: Which customers have ordered more than one product? Return these columns:

- The email address from the Customers table
- The count of distinct products from the customer's orders

```
SELECT a.email_address, count(b.order_id) as count

FROM customers as a

JOIN orders as b

ON a.customer_id = b.customer_id

GROUP BY 1

HAVING count > 1

ORDER BY 1 DESC;
```

email_address	count
david.goldstein@hotmail.com	2
allan.sherwood@yahoo.com	2

15. 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 results by the list_price column in descending sequence.

```
SELECT product_name, list_price

FROM products AS a

WHERE list_price > (SELECT avg(list_price) FROM products)

ORDER BY 2 DESC;
```

	product_name	list_price
▶	Gibson SG	2517.00
	Gibson Les Paul	1199.00

16. 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 category_name

FROM categories as a

WHERE NOT EXISTS

(SELECT *

FROM products as b

WHERE a.category_id = b.category_id);
```

	category_name
▶	Keyboards

17. Write a SELECT statement that returns three columns: email_address, order_id, and the order total for each customer and order; 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.

```
Select email_address, max(order_total) as largest_order

FROM (SELECT c.email_address, o.order_id, sum((item_price-
discount_amount)*quantity) as order_total

FROM customers as c

join orders as o

ON c.customer_id = o.customer_id

Join order_items as oi

ON o.order_id = oi.order_id

GROUP by c.email_address, oi.order_id)  A

Group by email_address;
```

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

18. 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 results by the product_name column.

```
SELECT DISTINCT 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;
```

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

19. 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
- 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)

```
SELECT list_price, FORMAT(list_price,1) as price_1,

CONVERT(list_price, SIGNED) as price_2,

CAST(list_price AS SIGNED) as price_3, date_added,

CAST(date_added as date) as yyyy_mm_dd,

CAST(date_added as char(7)) as yyyy_mm_dd,

CAST(date_added as time) as time

FROM products

ORDER BY 1 ASC;
```

list_price	price_1	price_2	price_3	date_added	yyyy_mm_dd	yyyy_mm_dd	time
299.00	299.0	299	299	2015-07-30 13:58:35	2015-07-30	2015-07	13:58:35
415.00	415.0	415	415	2015-07-30 14:12:41	2015-07-30	2015-07	14:12:41
489.99	490.0	490	490	2015-06-01 11:12:59	2015-06-01	2015-06	11:12:59
499.99	500.0	500	500	2015-07-30 14:18:33	2015-07-30	2015-07	14:18:33
699.00	699.0	699	699	2014-10-30 09:32:40	2014-10-30	2014-10	09:32:40
699.99	700.0	700	700	2015-07-30 12:46:40	2015-07-30	2015-07	12:46:40

20. 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 the last four digits of the card_number column in this format: XXXX-XXXX-XXXX-1234. In other words, use Xs for the first 12 digits of the card number and actual numbers for the last four digits of the number.

```
SELECT card_number,
       CONCAT('XXXX-XXXX-XXXX-',RIGHT(card_number ,4)) as last_four_digits
FROM orders;
```

card_number	last_four_digits
4111111111111111	XXXX-XXXX-XXXX-1111
4012888888881881	XXXX-XXXX-XXXX-1881
4111111111111111	XXXX-XXXX-XXXX-1111
378282246310005	XXXX-XXXX-XXXX-0005
4111111111111111	XXXX-XXXX-XXXX-1111
6011111111111117	XXXX-XXXX-XXXX-1117