NFT DATA ANALYSIS USING MYSQL

Advanced SQL Data Analysis

SUBMITTED BY: MANSI VARSHNEY

COURSE: DATA SCIENTISTS BOOTCAMP

Introduction:

Over the past 18 months, an emerging technology has caught the attention of the world; the NFT.

What is an NFT?
They are digital assets stored on the blockchain and stands for Non - Fungible token. And over $22 billion was spent last year on purchasing NFTs.

Why? People enjoyed the art, the speculated on what they might be worth in the future, and people didn't want to miss out.

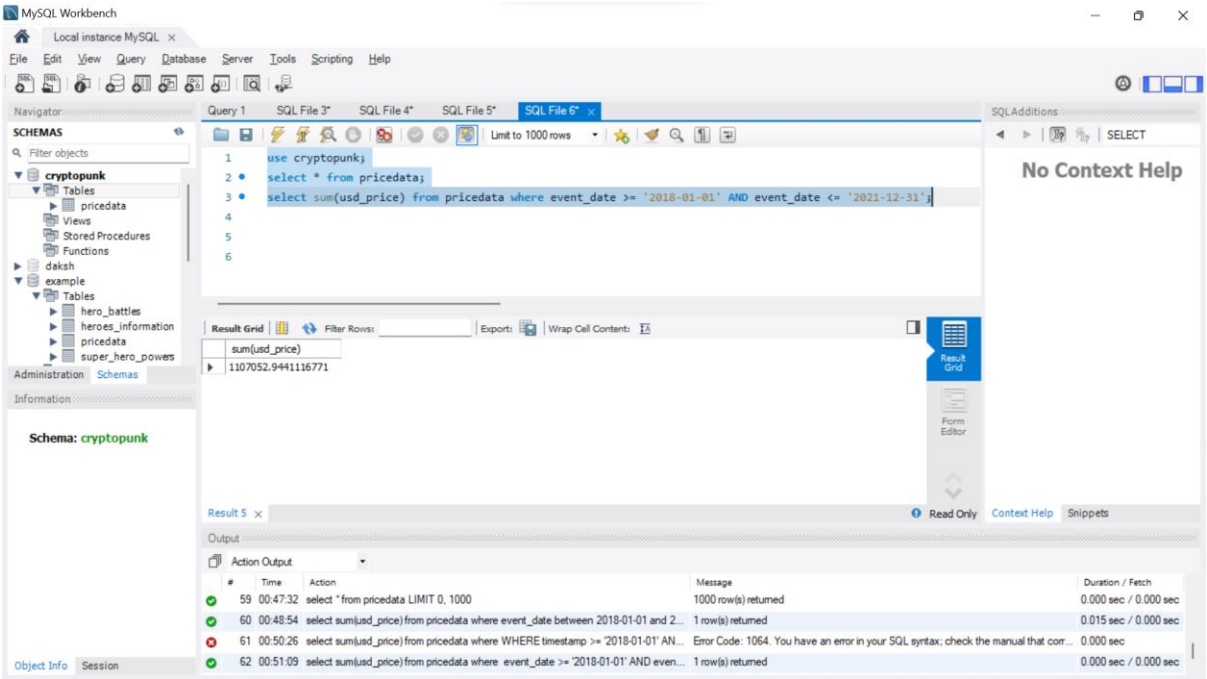
In this project, I analyses real-world of NFT.

About Dataset:

That data set is a sales data set of one of the most famous NFT projects, Cryptopunks. Meaning each row of the data set represents a sale of an NFT. The data includes sales from January 1st, 2018 to December 31st, 2021.
The table has several columns including the buyer address, the ETH price, the price in U.S. dollars, the seller's address, the date, the time, the NFT ID, the transaction hash, and the NFT name.

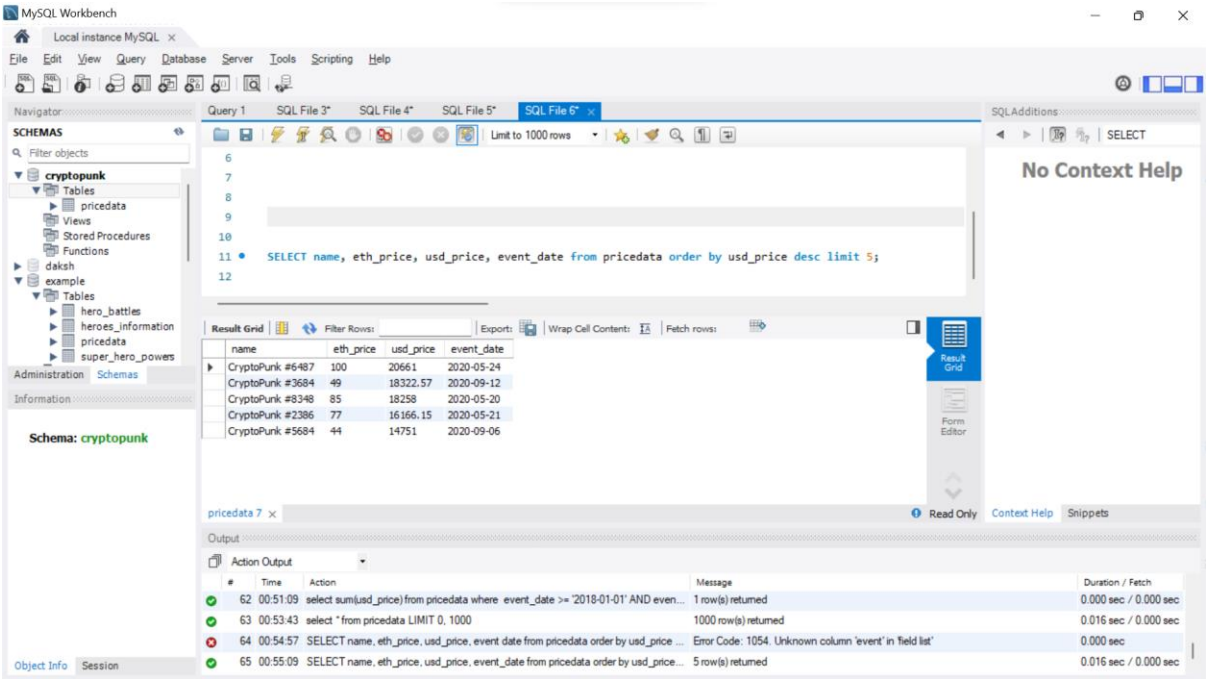
Ques 1. How many sales occurred during this time period?

```
use cryptopunk;
select * from pricedata;
select sum(usd_price) from pricedata where event_date >= '2018-01-01' AND event_date <=
'2021-12-31';
```



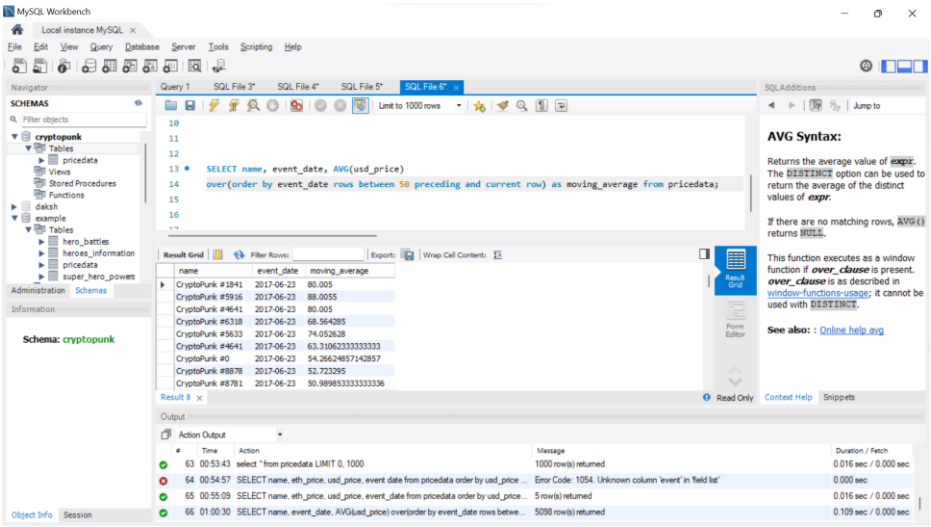
Ques 2. Return the top 5 most expensive transactions (by USD price) for this data set. Return the name, ETH price, and USD price, as well as the date.

```
SELECT name, eth_price, usd_price, event_date from pricedata order by usd_price desc limit 5;
```



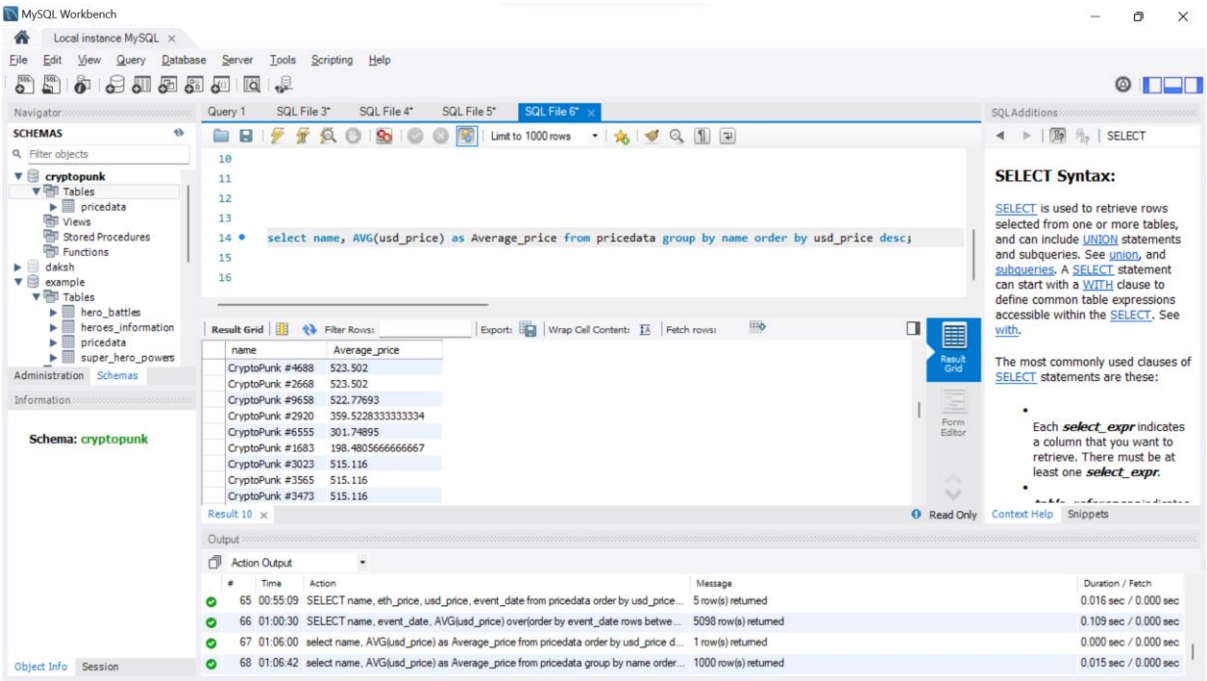
Ques 3: Return a table with a row for each transaction with an event column, a USD price column, and a moving average of USD price that averages the last 50 transactions.

```
SELECT name, event_date, AVG(usd_price)
over(order by event_date rows between 50 preceding and current row) as moving_average from pricedata;
```



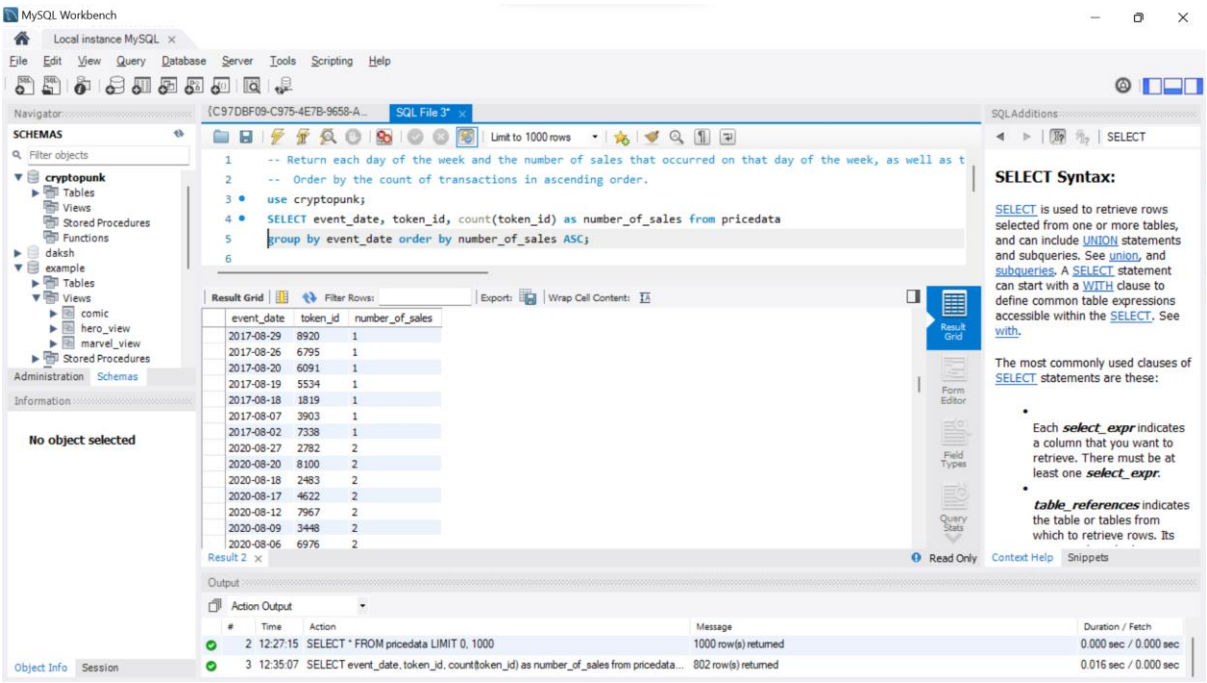
Ques 4. Return all the NFT names and their average sale price in USD. Sort descending. Name the average column as average_price.

```
select name, AVG(usd_price) as Average_price from pricedata group by name order by usd_price
desc;
```



Ques 5. Return each day of the week and the number of sales that occurred on that day of the week, as well as the average price in ETH. Order by the count of transactions in ascending order.

```
SELECT event_date, token_id, count(token_id) as number_of_sales from pricedata
group by event_date order by number_of_sales ASC;
```

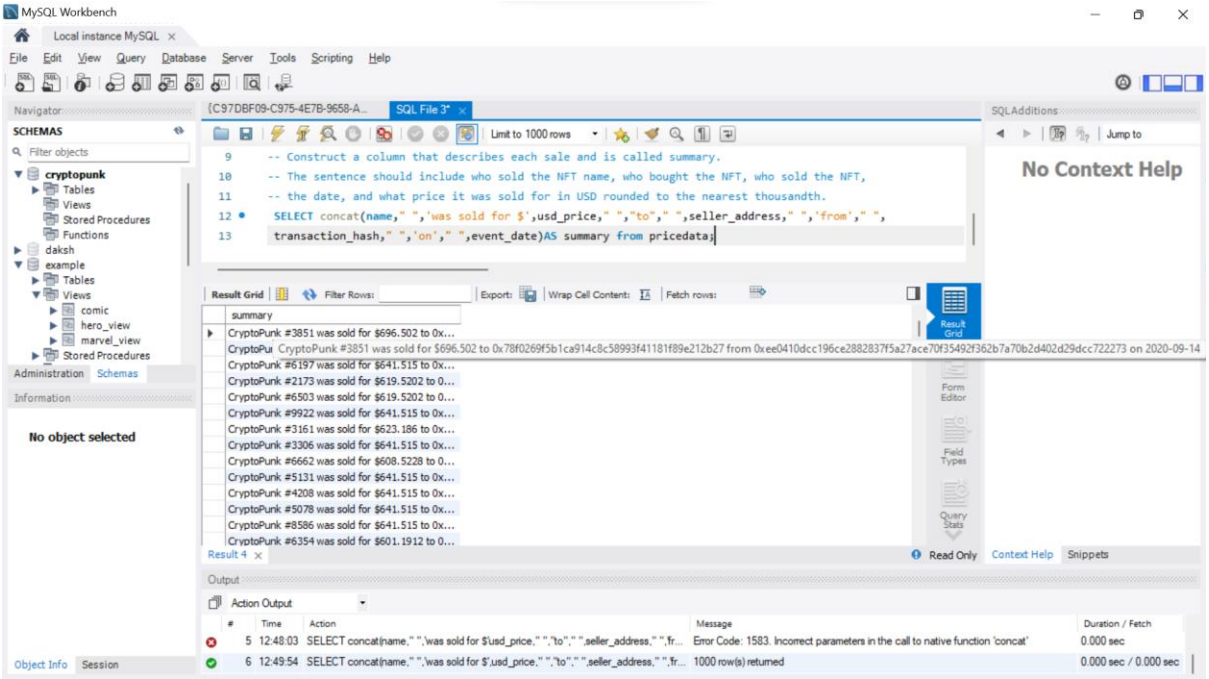


Ques 6: Construct a column that describes each sale and is called summary. The sentence should include who sold the NFT name, who bought the NFT, who sold the NFT, the date, and what price it was sold for in USD rounded to the nearest thousandth.

Here's an example summary:

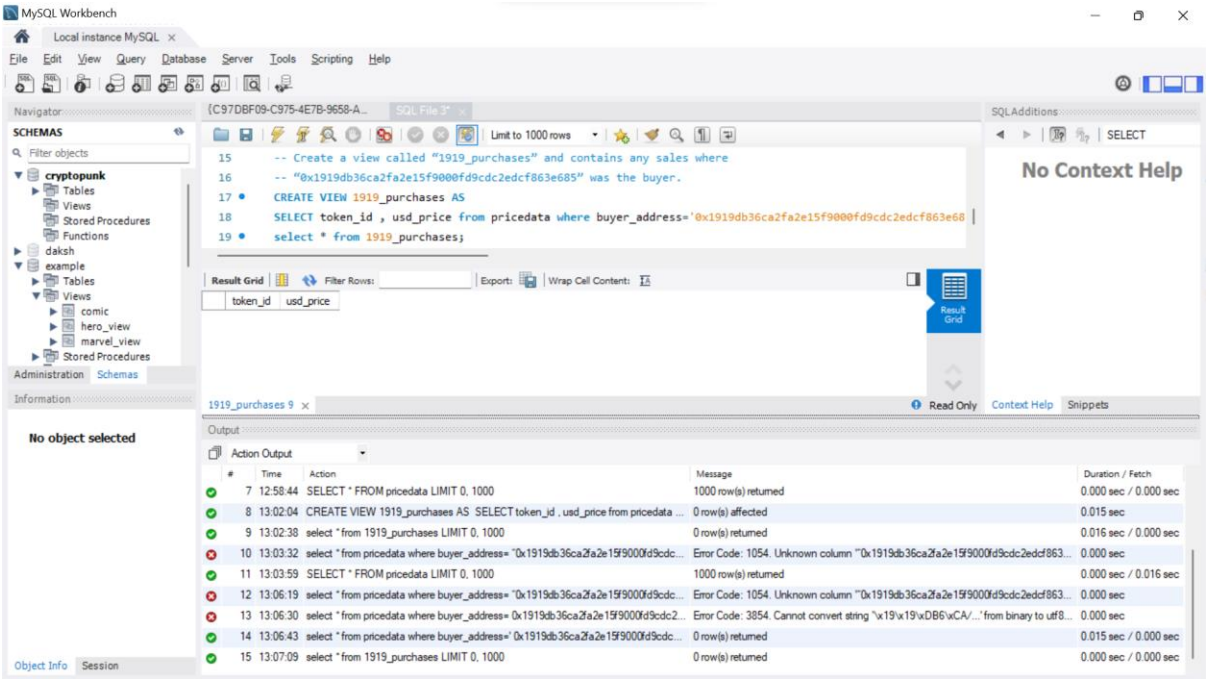
"CryptoPunk #1139 was sold for $194000 to 0x91338ccfb8c0adb7756034a82008531d7713009d from 0x1593110441ab4c5f2c133f21b0743b2b43e297cb on 2022-01-14"

```
SELECT concat(name," ",'was sold for $',usd_price," ","to"," ",seller_address," ",'from'," ",transaction_hash," ",'on'," ",event_date)AS summary from pricedata;
```



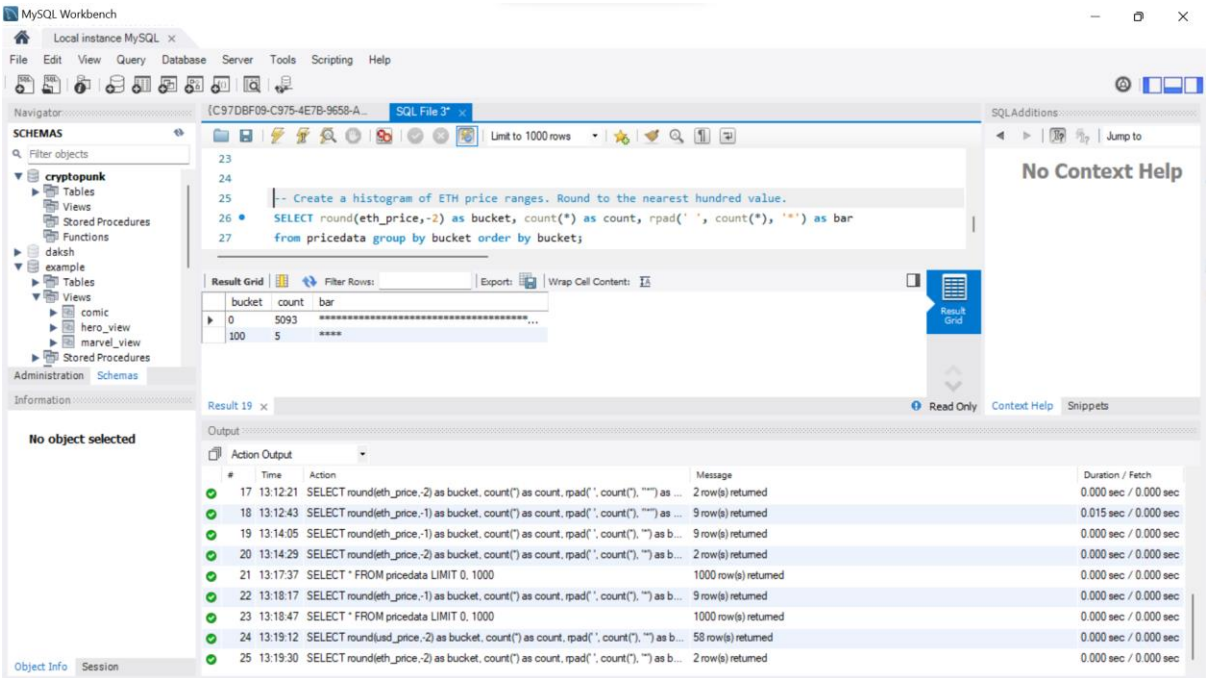
Ques 7: Create a view called "1919_purchases" and contains any sales where "0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685" was the buyer.

```
CREATE VIEW 1919_purchases AS
SELECT token_id , usd_price from pricedata where
buyer_address='0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685';
select * from 1919_purchases;
```



Ques 8: Create a histogram of ETH price ranges. Round to the nearest hundred value.

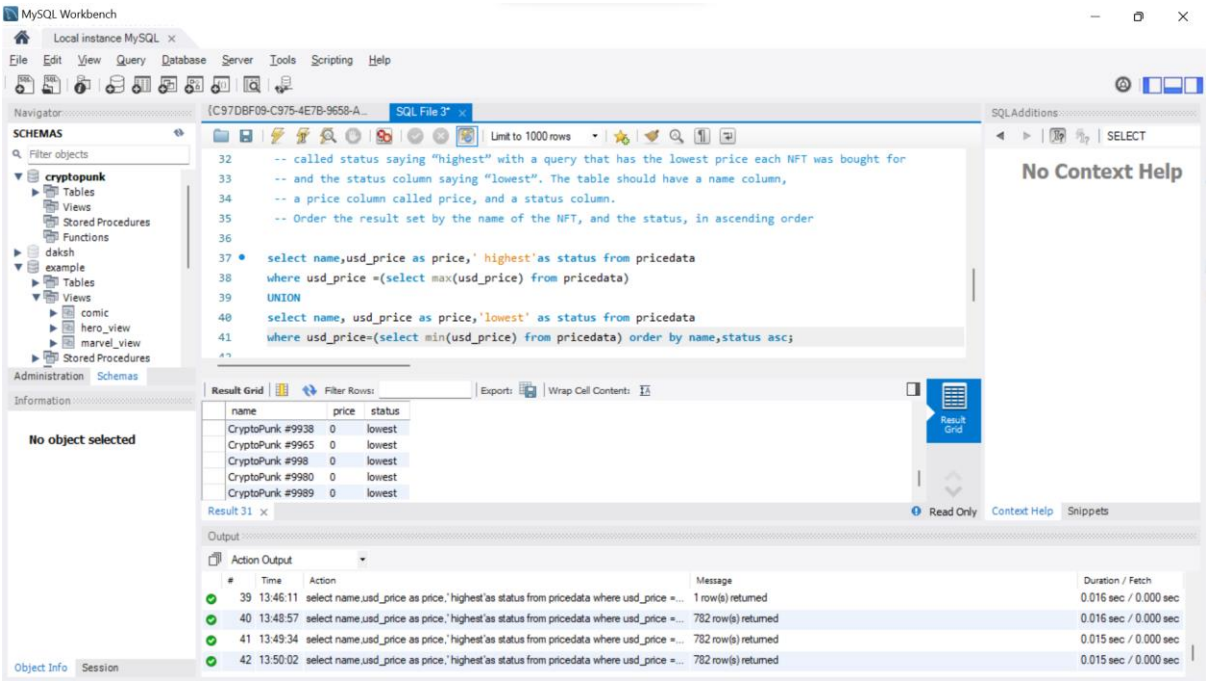
```
SELECT round(eth_price,-2) as bucket, count(*) as count, rpad(' ', count(*), '*') as bar from pricedata group by bucket order by bucket;
```



Ques 9: Return a unioned query that contains the highest price each NFT was bought for and a new column called status saying “highest” with a query that has the lowest price each NFT was bought for and the status column saying “lowest”. The table

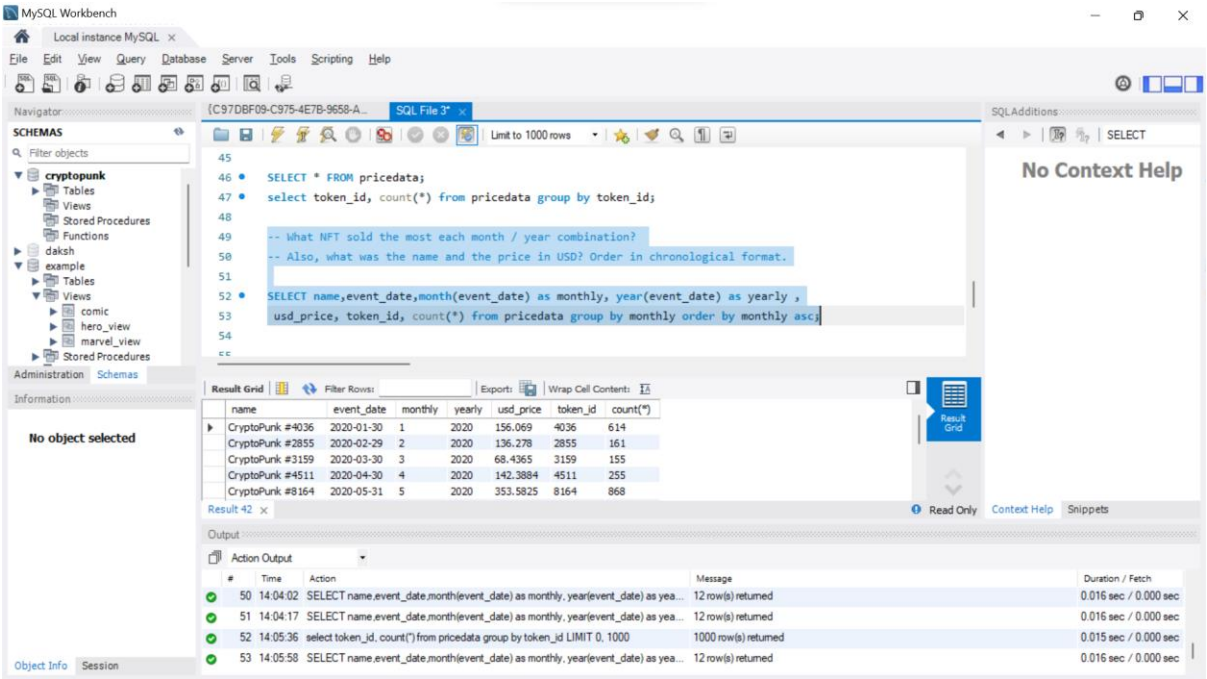
should have a name column, a price column called price, and a status column. Order the result set by the name of the NFT, and the status, in ascending order.

```
select name,usd_price as price,' highest' as status from pricedata
where usd_price =(select max(usd_price) from pricedata)
UNION
select name, usd_price as price,'lowest' as status from pricedata
where usd_price=(select min(usd_price) from pricedata) order by name asc,status
asc;
```



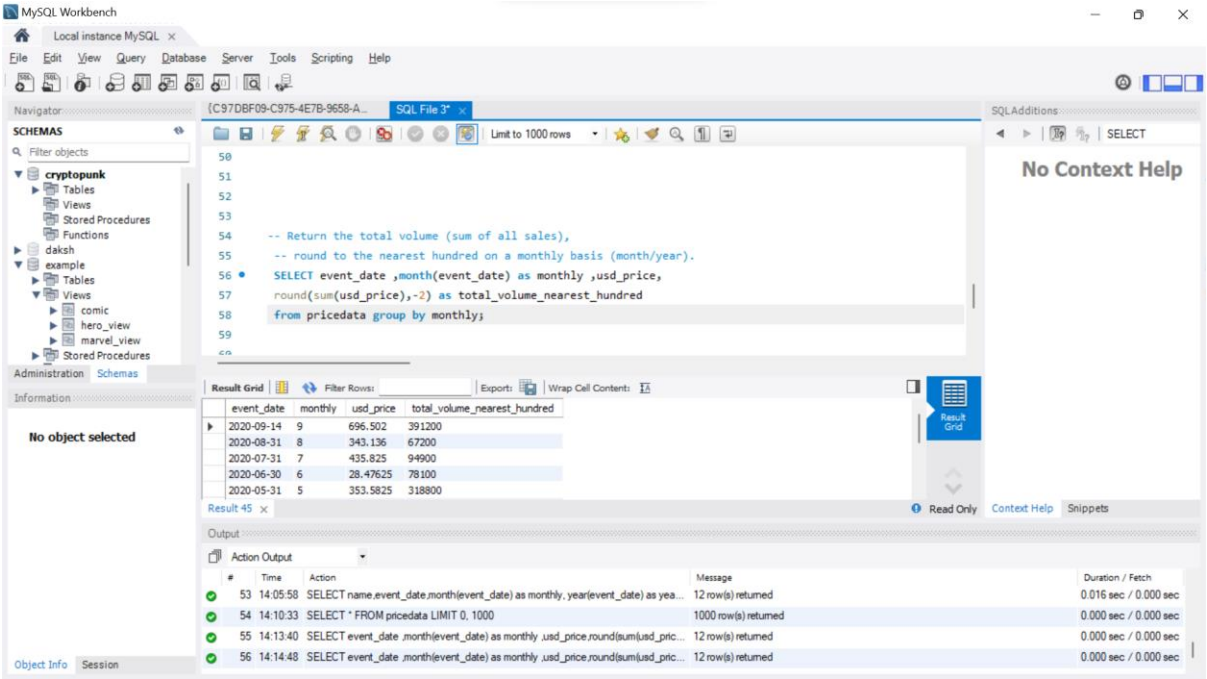
Ques 10: What NFT sold the most each month / year combination? Also, what was the name and the price in USD? Order in chronological format.

```
SELECT name,event_date,month(event_date) as monthly, year(event_date) as
yearly ,
 usd_price, token_id, count(*) from pricedata group by monthly order by monthly
asc;
```



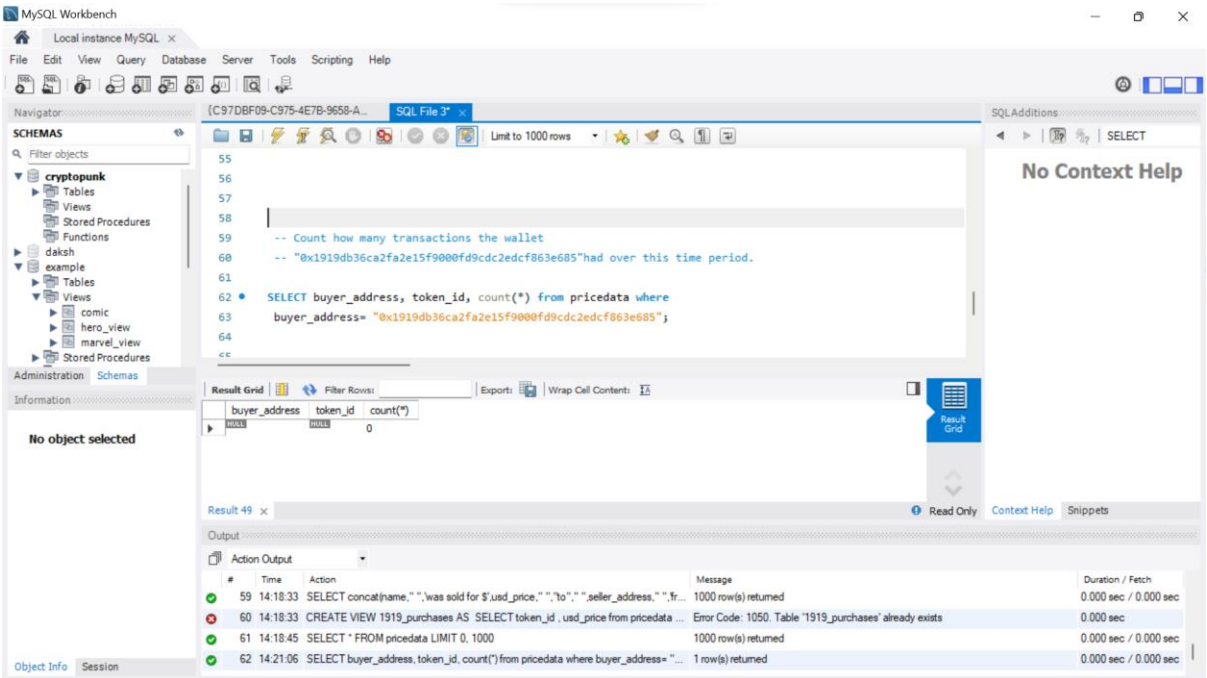
Ques 11: Return the total volume (sum of all sales), round to the nearest hundred on a monthly basis (month/year).

```
SELECT event_date ,month(event_date) as monthly ,usd_price,
 round(sum(usd_price),-2) as total_volume_nearest_hundred
 from pricedata group by monthly;
```



Ques 12: Count how many transactions the wallet "0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685"had over this time period.

```
SELECT buyer_address, token_id, count(*) from pricedata where
buyer_address= "0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685";
```



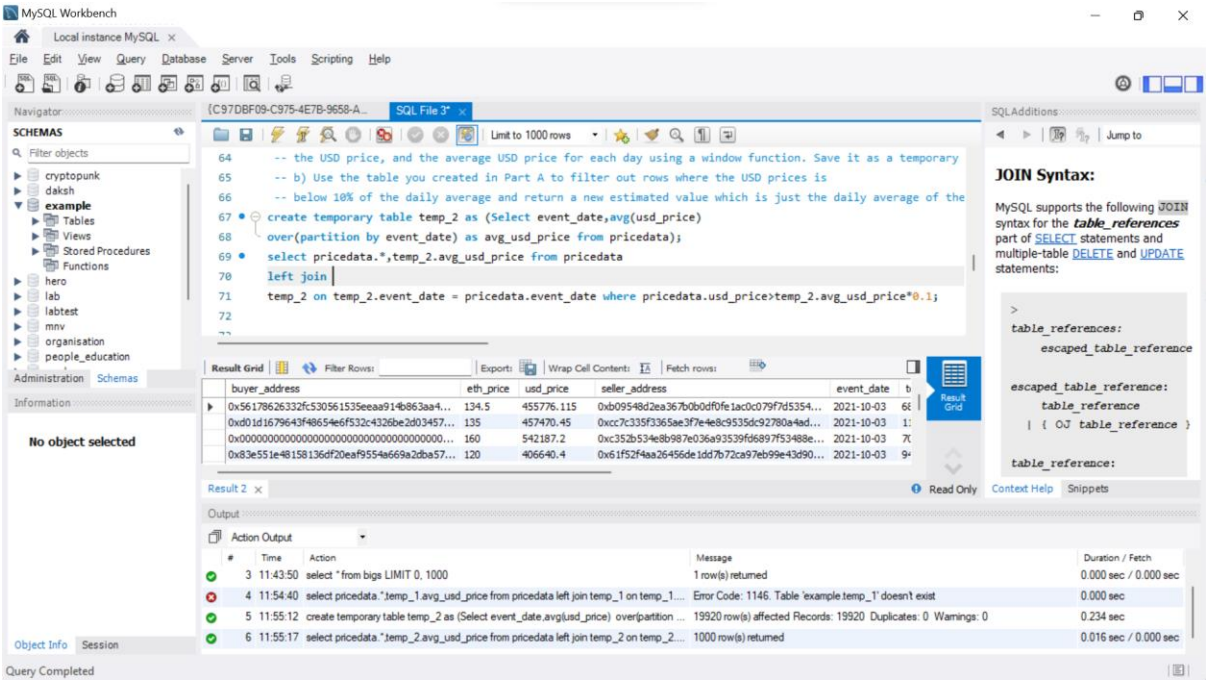
Ques 13: Create an “estimated average value calculator” that has a representative price of the collection every day based off of these criteria:

- Exclude all daily outlier sales where the purchase price is below 10% of the daily average price
- Take the daily average of remaining transactions

a) First create a query that will be used as a subquery. Select the event date, the USD price, and the average USD price for each day using a window function. Save it as a temporary table.

b) Use the table you created in Part A to filter out rows where the USD prices is below 10% of the daily average and return a new estimated value which is just the daily average of the filtered data

```
create temporary table temp_2 as (Select event_date,avg(usd_price)
over(partition by event_date) as avg_usd_price from pricedata);
select pricedata.*,temp_2.avg_usd_price from pricedata
left join
temp_2 on temp_2.event_date = pricedata.event_date where
pricedata.usd_price>temp_2.avg_usd_price*0.1;
```

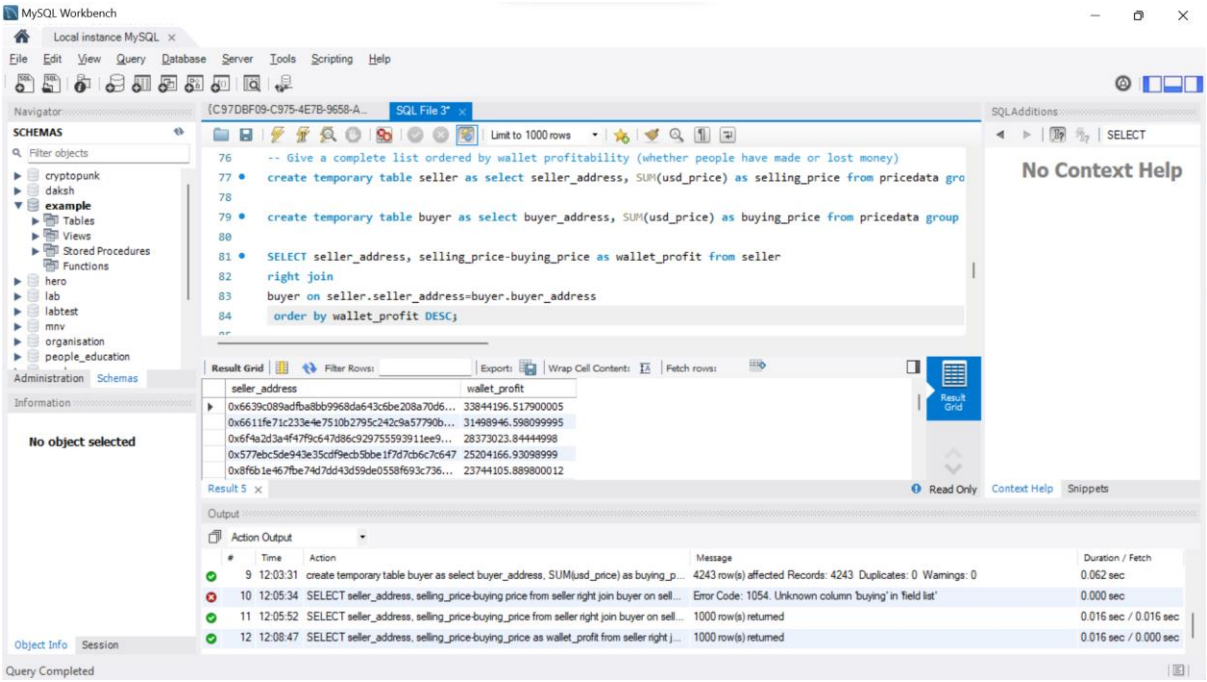


Ques 14: Give a complete list ordered by wallet profitability (whether people have made or lost money)

```
create temporary table seller as select seller_address, SUM(usd_price) as
selling_price from pricedata group by seller_address;

create temporary table buyer as select buyer_address, SUM(usd_price) as
buying_price from pricedata group by buyer_address;

SELECT seller_address, selling_price-buying_price as wallet_profit from seller
right join
buyer on seller.seller_address=buyer.buyer_address
 order by wallet_profit DESC;
```


MySQL Workbench
Local instance MySQL
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
cryptopunk
daksh
example
Tables
Views
Stored Procedures
Functions
hero
lab
labtest
mnv
organisation
people_education
Administration Schemas
Information
No object selected
Object Info Session
{C97DBF09-C975-4E7B-9658-A...
SQL File 3*
Limit to 1000 rows
-- Give a complete list ordered by wallet profitability (whether people have made or lost money)
create temporary table seller as select seller_address, SUM(usd_price) as selling_price from pricedata gro
create temporary table buyer as select buyer_address, SUM(usd_price) as buying_price from pricedata group
SELECT seller_address, selling_price-buying_price as wallet_profit from seller
right join
buyer on seller.seller_address=buyer.buyer_address
order by wallet_profit DESC;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Fetch rows:
seller_address
wallet_profit
0x6639c089adfba8bb9968da643c6be208a70d6...
33844196.517900005
0x6611fe71c233e4e7510b2795c242c9a57790b...
31498946.598099995
0x6f4a2d3a4f47f9c647d86c929755593911ee9...
28373023.84444998
0x577ebc5de943e35cdf9ecb5bbe1f7d7cb6c7c647
25204166.93098999
0x8f6b1e467fbe74d7dd43d59de0558f693c736...
23744105.889800012
Result 5
Read Only
Output
Action Output
Time Action Message Duration / Fetch
9 12:03:31 create temporary table buyer as select buyer_address, SUM(usd_price) as buying_p... 4243 row(s) affected Records: 4243 Duplicates: 0 Warnings: 0 0.062 sec
10 12:05:34 SELECT seller_address, selling_price-buying price from seller right join buyer on sell... Error Code: 1054. Unknown column 'buying' in 'field list' 0.000 sec
11 12:05:52 SELECT seller_address, selling_price-buying_price from seller right join buyer on sell... 1000 row(s) returned 0.016 sec / 0.016 sec
12 12:08:47 SELECT seller_address, selling_price-buying_price as wallet_profit from seller right j... 1000 row(s) returned 0.016 sec / 0.000 sec
Query Completed
SQLAdditions
SELECT
No Context Help
Context Help Snippets