MYSQL PROJECT ON REAL WORLD NFT (CRYPTOPUNKS) DATA

By Mohammad Asad

https://github.com/Asad19971/MySQL_NFT_Cryptopinks_Project.git

Instruction

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 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.

The future of NFT's is unclear as much of the NFT's turned out to be scams of sorts since the field is wildly unregulated. They're also contested heavily for their impact on the environment.

Regardless of these controversies, it is clear that there is money to be made in NFT's. And one cool part about NFT's is that all of the data is recorded on the blockchain, meaning anytime something happens to an NFT, it is logged in this database.

In this project, you'll be tasked to analyze real-world NFT data.
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.
You might not understand all the jargon around the NFT space, but you should be able to infer enough to answer the following prompts.

DATA SNAPSHOT

	A	B	C	D	E	F	G	H	I	J	K
1	buyer_ad	eth_price	usd_price	seller_adc	day	utc_times	token_id	transactio	name	wrapped_punk	
2	0x91338cc	59.9	194171.8	0x159311C	01/14/22	01/14/22	1139	0x5f9f302	CryptoPur	FALSE	
3	0xafa796c	63.95	207300.3	0x1919db3	01/14/22	01/14/22	3874	0x06d789	CryptoPur	FALSE	
4	0x000000C	50	162080	0xad4fa8a	01/14/22	01/14/22	7969	0x443a2ef	CryptoPur	FALSE	
5	0x3e8faf5	67.95	220266.7	0x44a3ccd	01/14/22	01/14/22	5231	0x41d380	CryptoPur	FALSE	
6	0x000000C	59	191254.4	0xfe31364	01/14/22	01/14/22	3193	0x5b6a81a	CryptoPur	FALSE	
7	0x000000C	82	265811.2	0x6f4a2d3	01/14/22	01/14/22	3961	0x99efb4b	CryptoPur	FALSE	
8	0x1919db3	68.88	232349.5	0x6611fe7	01/13/22	01/13/22	9056	0x9940791	CryptoPur	FALSE	
9	0x000000C	60	202395	0x74dacd8	01/13/22	01/13/22	8335	0xfc1a9b9	CryptoPur	FALSE	
10	0xc66d5be	62.97	204075.1	0x60e23da	########	########	2354	0xa1cb365	CryptoPur	FALSE	
11	0x91338cc	60.95	197528.6	0x1919db3	########	########	1915	0x9ec598e	CryptoPur	FALSE	
12	0x91338cc	59.69	193445.1	0x1919db3	########	########	1482	0x2e04dd	CryptoPur	FALSE	
13	0x12039d9	82.95	268826.8	0x6639c08	########	########	4965	0x9c14698	CryptoPur	FALSE	
14	0x65b1b9(	66.66	216033.7	0xcc7c335	########	########	9504	0xc52f062	CryptoPur	FALSE	
15	0xa8d31c4	79.9	258942.3	0x2891a44	########	########	6928	0x098a164	CryptoPur	FALSE	
16	0x1919db3	60	194449.8	0x7a01064	########	########	3080	0xce457c5	CryptoPur	FALSE	
17	0x23d23d(	65	200428.2	0xfe8d12b	########	########	6050	0x287dd8(	CryptoPur	FALSE	
18	0xe2ca739	63.63	196203.7	0x6f4a2d3	########	########	3993	0x064713b	CryptoPur	FALSE	
19	0x000000C	0	0	0xb0161d9	########	########	3842	0x2e768cb	CryptoPur	FALSE	
20	0x79330aa	63.97	197252.1	0x60e23da	########	########	6558	0x83bd59	CryptoPur	FALSE	

1.HOW MANY SALES OCCURRED DURING THIS TIME PERIOD?

```
Select count(*)
From pricedata;
```

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;
```

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 event_date, usd_price, AVG(usd_price)
OVER(ORDER BY event_date ROWS BETWEEN 50
PRECEDING AND CURRENT ROW)
FROM pricedata;
```

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 average_price desc;
```

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 dayname(event_date) as Weekday, count(*) as
NumberOfSales, Avg(eth_price)
From pricedata
Group by weekday
Order by NumberOfSales;
```

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.

```
SELECT CONCAT(name, " was sold for $", ROUND(usd_price,-3), " to ", buyer_address, " from ", seller_address, " on ", event_date) FROM pricedata;
```

7.CREATE A VIEW CALLED “1919_PURCHASES” AND CONTAINS ANY SALES WHERE “0X1919DB36CA2FA2E15F9000FD9CDC2EDCF863E685” WAS THE BUYER.

```
Create view 1919_purchases AS
Select *
From Pricedata
where buyer_address =
'0x1919db36ca2fa2e15f9000fd
9cdc2edcf863e685';
```

8. CREATE A HISTOGRAM OF ETH PRICE RANGES. ROUND TO THE NEAREST HUNDRED VALUE.

```
Select Round(eth_price,-2) as eth_price_range, COUNT(*) AS COUNT
From pricedata
Group BY eth_price_range
Order BY eth_price_range;
```

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, MAX(usd_price) AS price,
'highest' AS status FROM pricedata
GROUP BY name)
UNION
(SELECT name, MIN(usd_price) AS price,
'lowest' AS status FROM pricedata GROUP
BY name)
ORDER BY name, status ASC;
```

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, date_format(event_date,'%Y/%M') AS
month_year, COUNT(*) FROM pricedata
GROUP BY name, month_year
ORDER BY month_year, COUNT(*) desc;
```

11. RETURN THE TOTAL VOLUME (SUM OF ALL SALES), ROUND TO THE NEAREST HUNDRED ON A MONTHLY BASIS (MONTH/YEAR).

```
Select CONCAT(Month(event_date), '/', YEAR(event_date)) AS
month_year,
ROUND(SUM(usd_price), -2) AS total_volume
From pricedata
GROUP BY month_year
ORDER BY total_volume ASC;
```

12. COUNT HOW MANY TRANSACTIONS THE WALLET "0X1919DB36CA2FA2E15F9000FD9CDC2EDCF863E685"HAD OVER THIS TIME PERIOD.

```
Select count(*) as Transaction_Count
From Pricedata
where buyer_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685'
seller_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685';
```

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_table as
select event_date, usd_price, Avg(usd_price)
Over (Partition by event_date) as Avg_USD_Price
from pricedata;

SELECT event_date, AVG(usd_price) AS estimated_value
FROM temp_table
WHERE usd_price >= 0.1 * avg_usd_price
GROUP BY event_date;
```

THANK YOU