SQL Report

Stock Performance and Financial Health Analysis

Database Schema :

1. Companies CompanyID (Primary Key) Type: Integer Description: Unique identifier for each company. CompanyName Type: Varchar(255) Description: Name of the company. Industry Type: Varchar(100) Description: Industry to which the company belongs. Headquarters Type: Varchar(255) Description: Location of the company's headquarters.

2. Stocks StockID (Primary Key) Type: Integer Description: Unique identifier for each stock. CompanyID (Foreign Key) Type: Integer Description: References the associated company from the Companies table. TickerSymbol Type: Varchar(10) Description: Unique ticker symbol of the stock.

3. StockPrices PriceID (Primary Key) Type: Integer Description: Unique identifier for each stock price record. StockID (Foreign Key) Type: Integer Description: References the associated stock from the Stocks table. PriceDate Type: Date Description: The date for the stock price record. OpenPrice Type: Decimal(10, 2) Description: Opening price of the stock on that date. ClosePrice Type: Decimal(10, 2) Description: Closing price of the stock on that date. HighPrice Type: Decimal(10, 2) Description: Highest price of the stock on that date. LowPrice Type: Decimal(10, 2) Description: Lowest price of the stock on that date. Volume Type: Integer Description: Total trading volume of the stock on that date.

4. StockTransactions TransactionID (Primary Key) Type: Integer Description: Unique identifier for each stock transaction. StockID (Foreign Key) Type: Integer Description: References the associated stock from the Stocks table. TransactionDate Type: Date Description: Date of the transaction. TransactionType Type: Varchar(10) Description: Type of transaction (e.g., 'Buy', 'Sell'). TransactionVolume Type: Integer Description: Number of shares involved in the transaction. TransactionPrice Type: Decimal(10, 2) Description: Price per share for the transaction.

Scenario: Stock Performance and Company Financial Health Report Objective: Create a comprehensive report on the performance of stocks and the financial health of companies for the year 2024. The report will include various metrics such as average stock prices, trading volumes, transaction patterns, and financial ratios.

Report Sections 1. Stock Price and Volume Analysis

1. Average Daily Stock Prices: Calculate average of closeprice of each stock for all the dates from 1st till 31st.

SQL:

select stockid, tickersymbol, round(avg(closeprice),2) as avg_cp, extract(day from sp.pricedate) as day from stockprices sp natural join stocks group by stockid,tickersymbol, extract (day from sp.pricedate) order by stockid, day asc;

EXPLANATION:

The query retrieves the average closing price (`avg_cp`) for each stock on a daily basis. It groups the data by `stockid`, `tickersymbol`, and the day extracted from the `pricedate`, and calculates the rounded average closing price to two decimal places for each group. The results are then ordered by `stockid` and day in ascending order, showing how each stock's average daily closing price changes over time.

	STOCKID	TICKERSYMBOL	AVG_CP	DAY
1	1	AI	38.19	1
2	1	AI	36.51	2
3	1	AI	36.72	3
4	1	AI	40.62	4
5	1	AI	39.98	5
6	1	AI	33.09	6
7	1	AI	34.84	7
8	1	AI	39.77	8
9	1	AI	38.62	9
10	1	AI	38.87	10
11	1	AI	42.21	11
12	1	AI	40.04	12
13	1	AI	33.88	13
14	1	AI	36.49	14
15	1	AI	38.82	15
16	1	AI	39.15	16
17	1	AI	37.65	17
18	1	AI	37.87	18
19	1	AI	38.57	19
20	1	AI	34.29	20

2. Monthly Volume Trends: Analyze the trading volume trends for each stock on a monthly basis.

SQL:

select stockid, sum(volume) as vol, extract(month from sp.pricedate) as month from stockprices sp where pricedate BETWEEN TO_DATE('1990-01-01', 'YYYY-MM-DD') AND TO_DATE('2024-12-31', 'YYYY-MM-DD') group by stockid, extract(month from sp.pricedate) order by stockid, month asc;

EXPLANATION:

The query calculates the total trading volume (vol) for each stock on a monthly basis between January 1, 1990, and December 31, 2024. It groups the data by stockid and the extracted month from the pricedate, summing up the volumes for each month. The results are then ordered by stockid and month in ascending order, providing a monthly overview of trading activity for each stock over the specified period.

	STOCKID	VOL	MONTH
1	1	405733400	1
2	1	946248700	2
3	1	830858700	3
4	1	712203400	4
5	1	862433200	5
6	1	1034503500	6
7	1	513615100	7
8	1	391160700	8
9	1	322524300	9
10	1	188530300	10
11	1	277066100	11
12	1	557415100	12
13	2	19478160800	1
14	2	16026418100	2
15	2	19501914800	3
16	2	16853333000	4
17	2	18875941200	5
18	2	18032693800	6
19	2	18342718900	7
20	2	18515170200	8
21	2	17699343300	9
22	2	19326496600	10
23	2	16530727300	11
24	2	15111295500	12

3. Top Performing Stocks: Identify the top 5 stocks with the highest average closing price in 2024.

SQL:

With stocks_CTE as (select sp.stockid, round(avg(sp.closeprice),2) as avg_max_cp, Rank()over (order by avg(sp.closeprice)desc) as rank_stocks from stockprices sp where pricedate like '%-24' group by stockid) select stockid, tickersymbol, avg max cp, rank_stocks from stocks_CTE natural join stocks where rank_stocks <6 order by rank_stocks;

EXPLANATION:

The query identifies and ranks the top 5 stocks based on their average closing prices in 2024. It first calculates the average closing price (avg_max_cp) for each stock in 2024 and assigns a rank (rank_stocks) to each stock in descending order of these average prices. The results are filtered to include only the top 5 ranked stocks. Finally, the query joins this information with the stocks table to retrieve the corresponding tickersymbol and displays the stockid, tickersymbol, avg_max_cp, and rank_stocks for the top 5 stocks. The output in the image shows the ranked list of stocks with their corresponding average closing prices and rankings.

	STOCKID	TICKERSYMBOL	AVG_MAX_CP	RANK_STOCKS
1	7	NVDA	819.95	1
2	6	NOW	746.38	2
3	5	META	461.44	3
4	8	PANW	305.45	4
5	4	ANET	277.87	5

4. Most Volatile Stocks: Find the stocks with the highest price volatility (standard deviation of daily prices).

SQL:

select stockid, tickersymbol, round (STDDEV (closeprice),2) as vol_cp, round (STDDEV (openprice),2) as vol_op from stockprices sp natural join stocks s where pricedate between to_date ('1990-01-01', 'YYYY-MM-DD') and to_date ('2024-12-31', 'YYYY-MM-DD') group by stockid, tickersymbol;

EXPLANATION:

The SQL query calculates the standard deviation of the closing prices (vol_cp) and the opening prices (vol_op) for each stock from January 1, 1990, to December 31, 2024. The query uses the STDDEV function to measure the volatility of both closing and opening prices across the specified time range. The results are grouped by stockid and tickersymbol, showing the volatility metrics for each stock. The output table presents the stockid, tickersymbol, vol_cp, and vol_op for the listed stocks, indicating how much their prices have varied over time.

	STOCKID	TICKERSYMBOL	VOL_CP	VOL_OP
1	4	ANET	64.28	64.22
2	6	NOW	221.15	221.25
3	10	TSLA	101.92	102
4	1	AI	30.25	30.47
5	3	AMZN	52.75	52.77
6	2	AMD	32.23	32.24
7	5	META	106.19	106.1
8	9	PATH	18.66	18.67
9	7	NVDA	135.88	135.76
10	8	PANW	75.27	75.19

2. Transaction Patterns and Financial Ratios

1. Analyze the transaction patterns by identifying the most frequent transaction types for each stock.

SQL:

```
With TransactionCount AS (select st.StockID, st.TransactionType, extract (month from st.transactiondate) as month, count(*) AS TransactionCount from StockTransactions st group by st.StockID, st.TransactionType, extract (month from st.transactiondate))

select s.TickerSymbol, tc.TransactionType, tc.TransactionCount, month from TransactionCount tc join Stocks s on tc.StockID = s.StockID order by tc.TransactionCount desc;
```

EXPLANATION:

The SQL query calculates the count of stock transactions for each stock (StockID) by transaction type (either "BUY" or "SELL") and by month. The query first creates a Common Table Expression (CTE) named TransactionCount, which aggregates the number of transactions (TransactionCount) for each StockID, TransactionType, and month. In the final selection, the CTE is joined with the Stocks table to retrieve the TickerSymbol associated with each StockID. The result displays the TickerSymbol, TransactionType, TransactionCount, and the month of the transaction, sorted in descending order by TransactionCount. This output helps identify the frequency of buy and sell transactions for each stock on a monthly basis.

	TICKERSYMBOL	TRANSACTIONTYPE	TRANSACTIONCOUNT	MONTH
1	TSLA	SELL	4	6
2	TSLA	SELL	3	8
3	ANET	SELL	3	5
4	AMD	SELL	3	12
5	NVDA	BUY	2	5
6	TSLA	SELL	2	4
7	ANET	SELL	2	4
8	META	SELL	2	2
9	META	SELL	2	5
10	ANET	BUY	2	11
11	AMZN	BUY	2	5
12	AI	BUY	2	6
13	PATH	SELL	2	2
14	NOW	SELL	2	2
15	PATH	SELL	2	1
16	AMD	BUY	2	6
17	PANW	BUY	2	2
18	PANW	BUY	2	1
19	NVDA	SELL	2	5
20	ANET	BUY	1	6
21	NOW	SELL	1	6
22	ANET	BUY	1	8
23	PATH	BUY	1	11
24	PATH	BUY	1	2

2. Calculate the P/V ratio for each company based on the closing prices in 2024.

SQL:

```
With PV_CTE as (select c.companyid, c.companyname, round (avg(sp.closeprice) /avg(sp.volume),8) as pv  ratio from companies c join stocks s on c.companyid = s.companyid join stockprices sp on s.stockid = sp.stockid where pricedate like '%-24' group by c.companyid, c.companyname)

select companyid, companyname, pv_ratio from PV_CTE pv order by pv_ratio desc;
```

EXPLANATION:

This SQL query calculates the price-to-volume (PV) ratio for each company based on their stock prices in the year 2024. The CTE named PV_CTE is created by joining the companies, stocks, and stockprices tables. It computes the average of the closeprice divided by the average volume for each company (pv_ratio), where the pricedate is from the year 2024. The results are grouped by companyid and companyname to ensure that the PV ratio is calculated for each company independently. The query selects the companyid, companyname, and the calculated pv_ratio from the CTE, ordering the results by pv_ratio in descending order. The output provides a list of

companies with their respective PV ratios, showing how much the stock's average closing price is in relation to the trading volume for each company in 2024. This can help identify companies with a high or low price relative to their trading volume.

	COMPANYID	COMPANYNAME	PV_RATIO
1	6	ServiceNow	0.00057615
2	4	Arista Networks	0.00011091
3	8	Palo Alto Networks	0.00006064
4	5	Meta Platforms	0.0000272
5	7	NVIDIA	0.00001638
6	3	Amazon	0.00000411
7	1	C3.ai	0.00000378
8	2	Advanced Micro Devices	0.00000252
9	9	UiPath	0.00000221
10	10	Tesla	0.0000019

3. Advanced Analysis and Projections

1. Calculate the year-over-year growth rate in lowest and highest stock prices from 2014 to 2024.

SQL:

With GrowthRate_CTE as (select sp.stockid, extract(year from sp.pricedate) as extracted_year, min(sp.closeprice) as min_price, max (sp.closeprice) as max_price from stockprices spwhere pricedate between to_date ('2014-01-01', 'YYYY-MM-DD') and to_date ('2024-12-31', 'YYYY-MM_DD') group by stockid, extract(year from sp.pricedate) order by extract(year from sp.pricedate) asc)

select stockid, tickersymbol,extracted_year, min_price-lag(min_price) over (partition by stockid order by extracted_year) as lag_cp, max_price-lead(max_price) over (partition by stockid order by extracted_year) as lead_cp from GrowthRate_CTE natural join stocks s;

EXPLANATION:

This SQL query calculates the year-over-year growth rates in the minimum and maximum stock prices for each stock between 2014 and 2024.The GrowthRate_CTE is generated by extracting the stockid, the year (extracted_year), and calculating the minimum (min_price) and maximum (max_price) closing prices for each stock within the specified date range (2014-2024). The results are grouped by stockid and the extracted year. LAG and LEAD Functions: The query calculates the difference between the current year's minimum closing price and the previous year's minimum closing price using the LAG function (lag_cp). It also calculates the difference between the current year's maximum closing price and the next year's maximum closing price using the LEAD function (lead_cp). The final result selects the stockid, tickersymbol, extracted_year, and the calculated year-over-year differences in minimum and maximum prices

(lag_cp and lead_cp) for each stock.The output provides insight into how the minimum and maximum closing prices for each stock have changed on a year-over-year basis within the selected time frame. This information can help assess the growth or decline in stock prices over time.

	STOCKID	TICKERSYMBOL	EXTRACTED_YEAR	LAG_CP	LEAD_CP
1	1	AI	2020	(null)	8.55
2	1	AI	2021	-62.74	136.62
3	1	AI	2022	-19.49	-14.07
4	1	AI	2023	0.55	9.4
5	1	AI	2024	9.69	(null)
6	2	AMD	2014	(null)	1.35
7	2	AMD	2015	-0.85	-8.76
8	2	AMD	2016	0.18	-3.13
9	2	AMD	2017	7.95	-17.52
10	2	AMD	2018	-0.22	-13.91
11	2	AMD	2019	7.52	-50.49
12	2	AMD	2020	21.66	-64.79
13	2	AMD	2021	34.38	11.67
14	2	AMD	2022	-17.15	1.48
15	2	AMD	2023	6.39	-62.62
16	2	AMD	2024	72.99	(null)
17	3	AMZN	2014	(null)	-14.35
18	3	AMZN	2015	0	-7.52
19	3	AMZN	2016	9.75	-17.57
20	3	AMZN	2017	13.58	-42.19
21	3	AMZN	2018	21.77	0.93
22	3	AMZN	2019	15.56	-75.52
23	3	AMZN	2020	8.82	-10
24	3	AMZN	2021	63.77	16.17

2. Use a simple linear regression model to project stock prices for the first quarter of 2024.

SQL:

WITH PriceData AS (SELECT sp.StockID, EXTRACT(MONTH FROM sp.PriceDate) AS Month, AVG(sp.ClosePrice) AS AvgClosePrice FROM StockPrices sp WHERE EXTRACT(YEAR FROM sp.PriceDate) = 2024 GROUP BY sp.StockID, EXTRACT(MONTH FROM sp.PriceDate)

),

*LinearRegression AS (SELECT pd.StockID, pd.AvgClosePrice + (RANK() OVER (PARTITION BY pd.StockID ORDER BY pd.Month) - 1) * ((pd.AvgClosePrice) - LAG(pd.AvgClosePrice) OVER (PARTITION BY pd.StockID ORDER BY pd.Month)) AS ProjectedPrice, month FROM PriceData group by pd.stockid, pd.month, pd.AvgClosePrice)*

SELECT s.TickerSymbol, lr.ProjectedPrice, month FROM LinearRegression lr JOIN Stocks s ON lr.StockID = s.StockID where month < 4 ORDER BY month asc;

EXPLANATION:

The SQL query performs a simplified linear regression analysis to project future stock prices for the first quarter of 2024 based on monthly average closing price. The first CTE, PriceData, calculates the average closing price (AvgClosePrice) for each stock (StockID) by month for the year 2024. The results are grouped by StockID and Month. The second CTE, LinearRegression, estimates a projected price for each stock by applying a basic linear regression formula (y = mx + b). It uses the difference between the current month's average closing price and the previous month's average (calculated using the LAG function) to determine the slope (m). The rank of each month serves as the x variable, representing time progression, and the intercept (b) is set to the average closing price itself. The final query retrieves the TickerSymbol, ProjectedPrice, and Month for each stock, limited to the first quarter (months 1, 2, and 3) of 2024. This output provides a forecast of the closing prices for each stock for the first three months of 2024, enabling users to observe how the prices might trend based on the historical data available for the same year.

	TICKERSYMBOL	PROJECTEDPRICE	MONTH
1	PANW	(null)	1
2	META	(null)	1
3	ANET	(null)	1
4	AI	(null)	1
5	NOW	(null)	1
6	AMD	(null)	1
7	NVDA	(null)	1
8	TSLA	(null)	1
9	AMZN	(null)	1
10	PATH	(null)	1
11	META	566.499666666666666666666666666666666667	2
12	ANET	285.796761904761904761904761904761904762	2
13	AMZN	188.391761904761904761904761904761904762	2
14	AMD	190.014	2
15	AI	29.71638095238095238095238095238095238095	2
16	TSLA	169.455523809523809523809523809523809524	2
17	PATH	25.53014285714285714285714285714285714286	2
18	PANW	351.759857142857142857142857142857142857	2
19	NVDA	888.417095238095238095238095238095238095	2
20	NOW	821.543476190476190476190476190476190476	2
21	AMZN	188.4605	3
22	PANW	184.6835	3
23	AI	33.658	3
24	AMD	227.771	3
25	META	552.7185	3
26	NVDA	1232.2675	3
27	ANET	325.519	3
28	PATH	21.5885	3
29	NOW	742.3935	3
30	TSLA	142.903	3

3. Identify stocks that have had an increasing trend in their average monthly closing price for at least three consecutive months in 2024.

SQL:

```
WITH MonthlyAvg AS (SELECT sp.StockID, EXTRACT(MONTH FROM sp.PriceDate) AS Month, round(AVG(sp.ClosePrice),2) AS AvgPrice FROM StockPrices sp WHERE sp.PriceDate BETWEEN TO_DATE('2024-01-01', 'YYYY-MM-DD') AND TO_DATE('2024-12-31', 'YYYY-MM-DD') GROUP BY sp.StockID, EXTRACT(MONTH FROM sp.PriceDate)), IncreasingTrend AS (SELECT ma1.StockID, ma1.Month, ma1.AvgPrice,

CASE  WHEN ma1.AvgPrice < LAG(ma1.AvgPrice, 1) OVER (PARTITION BY ma1.StockID ORDER BY ma1.Month) THEN 1 ELSE  0 END AS IsIncreasing FROM MonthlyAvg ma1),

TrendAnalysis AS ( SELECT it.StockID, it.Month, it.AvgPrice,SUM(it.IsIncreasing) OVER (PARTITION BY it.StockID ORDER BY it.Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS ThreeMonthTrend FROM IncreasingTrend it)

SELECT s.TickerSymbol, ta.Month, ta.AvgPrice FROM TrendAnalysis ta JOIN Stocks s ON ta.StockID = s.StockID WHERE ta.ThreeMonthTrend = 3;
```

EXPLANATION:

The SQL query is designed to spot stocks that consistently show an upward trend in their average monthly closing prices over any three-month period in 2024.

First, the MonthlyAvg CTE calculates the average closing price for each stock by month throughout 2024. The results are grouped by stock and month.

Next, the IncreasingTrend CTE compares each month's average price with the previous month's. If the current month's average is higher, it marks it as 1 in the IsIncreasing column, otherwise, it marks it as 0.

Then, the TrendAnalysis CTE sums up these IsIncreasing values over a rolling three-month window for each stock, giving us a ThreeMonthTrend column that shows how many of the last three months had an increasing trend.

Finally, the query pulls out the stock symbols, months, and average prices for those stocks that have shown a continuous upward trend for all three months in the rolling window. The result highlights stocks that are consistently gaining value, offering a clear view of potentially strong performers throughout 2024.

	TICKERSYMBOL	MONTH	AVGPRICE
1	NOW	5	728.45
2	NOW	6	687.64
3	PATH	5	18.95
4	PATH	6	11.96
5	TSLA	4	165.87