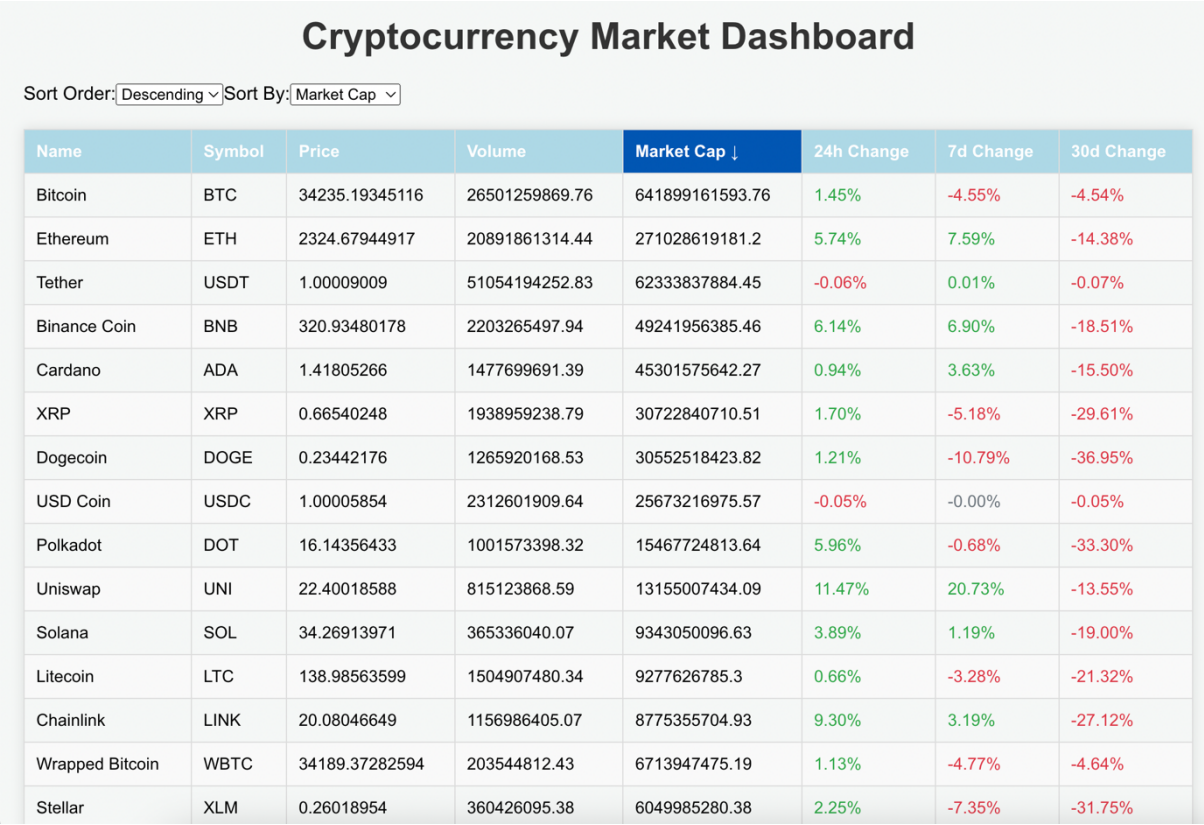
Caching strategy for Greythorn challenge

This answer is for question in code test “
- Assume you need to keep updating the original database to have the latest data, come up with a caching strategy to fit into the solution. There is no code needed to implement, but a well documented high level solution design diagram will be highly useful

”



Cryptocurrency Market Dashboard

Sort Order: Descending Sort By: Market Cap

Name	Symbol	Price	Volume	Market Cap ↓	24h Change	7d Change	30d Change
Bitcoin	BTC	34235.19345116	26501259869.76	641899161593.76	1.45%	-4.55%	-4.54%
Ethereum	ETH	2324.67944917	20891861314.44	271028619181.2	5.74%	7.59%	-14.38%
Tether	USDT	1.00009009	51054194252.83	62333837884.45	-0.06%	0.01%	-0.07%
Binance Coin	BNB	320.93480178	2203265497.94	49241956385.46	6.14%	6.90%	-18.51%
Cardano	ADA	1.41805266	1477699691.39	45301575642.27	0.94%	3.63%	-15.50%
XRP	XRP	0.66540248	1938959238.79	30722840710.51	1.70%	-5.18%	-29.61%
Dogecoin	DOGE	0.23442176	1265920168.53	30552518423.82	1.21%	-10.79%	-36.95%
USD Coin	USDC	1.00005854	2312601909.64	25673216975.57	-0.05%	-0.00%	-0.05%
Polkadot	DOT	16.14356433	1001573398.32	15467724813.64	5.96%	-0.68%	-33.30%
Uniswap	UNI	22.40018588	815123868.59	13155007434.09	11.47%	20.73%	-13.55%
Solana	SOL	34.26913971	365336040.07	9343050096.63	3.89%	1.19%	-19.00%
Litecoin	LTC	138.98563599	1504907480.34	9277626785.3	0.66%	-3.28%	-21.32%
Chainlink	LINK	20.08046649	1156986405.07	8775355704.93	9.30%	3.19%	-27.12%
Wrapped Bitcoin	WBTC	34189.37282594	203544812.43	6713947475.19	1.13%	-4.77%	-4.64%
Stellar	XLM	0.26018954	360426095.38	6049985280.38	2.25%	-7.35%	-31.75%

Here is my current website, and let us first analysis what we need briefly:

Our current database is structured to store one data(price, Volume etc) per day for each cryptocurrency. As per the new requirements, data will need to be updated every second instead of daily, This also means it requires me to calculate the fluctuations in prices over certain periods in the past.(calculate 24h 7d 30d change per second) To perform these calculations, it will be necessary to fetch price data points from 24 hours, 7 days, and 30 days prior, thereby increasing the database read operations by a factor of four.

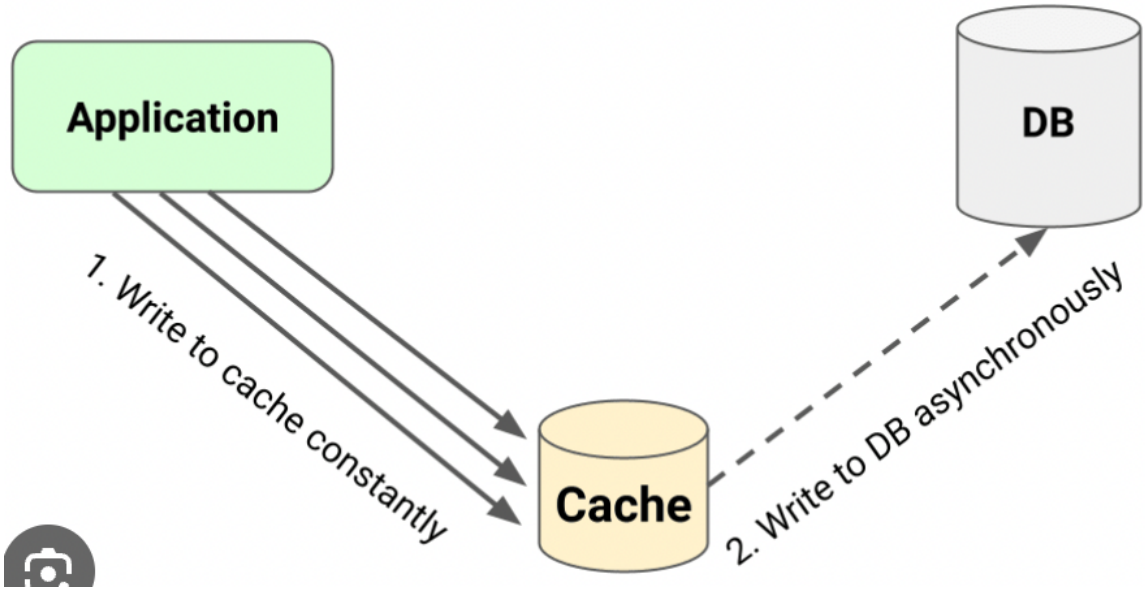
My Solution

In the proposed dashboard, every cryptocurrency's historical data(24h 7d 30d) must be readily accessible for immediate computation and display, without extra time spending on access database.

Proposed Caching Strategy

To fix these challenges, I propose a caching strategy designed to minimize the load on the database while ensuring rapid access to the necessary historical data for price change calculations. This strategy is:

1. **Write-behind Caching**: Upon each new piece of data received, I first update cache immediately, and write to DB asynchronously as figure below (figure from internet). The reason I am doing it is: In my cache strategy, I keep 30 days data in cache, so that for all data on my dashboard, I can directly access them from cache without need to access database. In this case, I can just write to DB asynchronously (as write to DB it is less important, no need to check data in DB) to let application to respond to user actions more quickly.



   Here is a figure from the internet shows this progress.
2. **Time-Based Invalidation**: For the eviction strategy, what I choose is time-based invalidation: Only keep recent 30 days data. Old data will be invalidation.

So here is my answer given to this question: Implement **Write-behind Caching with Time-Based Invalidation,** after get data from api (Binance, OKX, etc), write

them to cache directly, and also write them to database asynchronously. Data in cache keep 30 days. When user visit website, data can be get from cache directly without visit database, and also 24h, 7d, 30d data also can be extract from cache.

Solution 2(Only for Small website)

For a supporting tool website with moderate reliability requirements, based on the functionality of the site, we can rely solely on a cache system that retains data for 30 days without utilizing a traditional database. This approach simplifies the system architecture, making it easier to manage. However, it's important to note that this strategy does not guarantee long-term data persistence. In the event of a system failure, there's a potential risk of data loss. Additionally, cache systems typically do not support SQL or other complex query languages, which could limit the possibilities for data analysis.

Thus, if the website's primary function is to provide immediate, short-term insights without the necessity for historical data accumulation or complex querying capabilities, a cache-only architecture could be an appropriate choice. This design would streamline operations by removing the overhead associated with database maintenance and could offer sufficient performance for the intended use-case. However, it's crucial to implement robust error handling and data backup procedures to mitigate the risks associated with this approach.