Caching Strategies

In this lesson, we will discuss some of the commonly used caching strategies.

We'll cover the following

There are different kinds of caching strategies that serve specific use cases. Those are *cache aside*, *read-through cache*, *write-through cache* & *write-back cache*

Let's find out what they are and why we need different strategies when implementing caching.

Cache aside#

This is the most common caching strategy. In this approach, the cache works along with the database trying to reduce the hits on it as much as possible.

The data is *lazy-loaded* in the cache. When the user sends a request for particular data, the system first looks for it in the cache. If present, it is simply returned. If not, the data is fetched from the database, and the cache is updated and returned to the user.

This kind of strategy works best with *read-heavy* workloads. This includes the kind of data that is not frequently updated, like user profile data in a portal. User's name, account number, etc.

The data in this strategy is written directly to the database. This means that the data present in the cache and the database could become inconsistent. To avoid this, data on the cache has a TTL. After the stipulated period, the data is invalidated from the cache.

Read-through#

This strategy is pretty similar to the *cache aside* strategy. A subtle difference from the *cache aside* strategy is that the cache always stays consistent with the database in the read-through strategy.

The cache library, or the framework, takes the onus of maintaining consistency with the backend. The information in this strategy, is also *lazy-loaded* in the cache, only when the user requests it.

So, for the first time, when information is requested, it results in a cache miss. Then, the backend has to update the cache while returning the response to the user.

However, the developers can always preload the cache with the information that is expected to be requested most by the users.

Write-through#

In this strategy, each and every information written to the database even goes through the cache. Before the data is written to the DB, the cache is updated with it

This maintains high consistency between the cache and the database even though it adds a little latency during the write operations because the data is updated in the cache separately. This works well for write-heavy workloads like online massive multiplayer games.

This strategy is generally used with other caching strategies to achieve optimized performance.

Write-back#

This strategy helps optimize costs significantly. In the *write-back* caching strategy the data is directly written to the cache instead of the database, and the cache after some delay, as per the business logic, writes data to the database.

If there are quite a heavy number of writes in the application, developers can reduce the frequency of database writes to cut down the load and the associated costs.

This is the strategy I talked about in the previous lesson.

A risk in this approach is if the cache fails before the DB is updated, the data might get lost. Again, this strategy is used with other caching strategies to make the most out of them.

With this, we are done with the caching mechanism of web applications. Now let's move on to the world of message queues.

← Back Next →

Reducing the Application Deployment...

Caching Quiz

✓ Mark as Completed

Report an Issue