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 which serve specific use cases. Those are *Cache Aside*, *Read-through cache*, *Write-through cache* & *Write-back cache*

Let’s find out what they are & why do 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, then it is simply returned from it. If not, the data is fetched from the database, the cache is updated and is returned to the user.

This kind of strategy works best with *read-heavy* workloads. This includes the kind of data which is not much frequently updated, for instance, 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 get inconsistent. To avoid this data on the cache has a TTL “Time to Live”. After that 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 in the *Read-through* strategy, the cache always stays consistent with the database.

The cache library or the framework takes the onus of maintaining the consistency with the backend; The information in this strategy too is *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 pre-load the cache with the information which is expected to be requested most by the users.

Write-Through

In this strategy, each & every information written to the database 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 though it adds a little latency during the write operations as data is to be updated in the cache additionally. 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 & the associated costs.

This is the strategy which 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 these.

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