**Write-Through Cache:**Any new information is first written into the cache before the main memory/database.If it is combined with anyother cache implementation data consistency is reduced.

### Read-Through Cache:It gets the data from database if there is any data miss in the cache.For instance, a news website would be serving the same stories for a day over and over.The downside of this strategy is if the data is requested for the first time, it is always a cache miss, and thereby it will be slower than a normal request.

### Write-Back

### whenever there is a write operation, the application writes the information to the cache that immediately acknowledges the changes and after some delay, it writes back the data to the databasethe downside is that if there is a cache failure, the data might get lost forever. In most relational databases, the write-back caching mechanism is enabled by default.

### Write-Around

### In this case, the data is directly written to the database and only the data which is read is stored in the cache. It can be combined with the read-through cache and could be a good choice in situations where the data is written once and is read only a few times. For example, when there is a need for real-time logs or chats.

### Hazelcast:How to add property in springboot

* add org.springframework.boot:spring-boot-starter-cache dependency
* add @EnableCaching annotation to your main class
* add @Cacheable("books") annotation to every method you want to cache
* add hazelcast yml in resources mention hazelcast parameter

 You only need to auto-wire the HazelcastInstance bean in the CommandController and use it to access to Hazelcast data structures and add get and put http methods to update or retrieve infro from cache.

@Bean  
public Config configure() {  
 return new Config().setInstanceName("hazlecast-insatnce")  
 .addMapConfig(new MapConfig().setName("userCache")  
 .setMaxSizeConfig(new MaxSizeConfig(200, MaxSizeConfig.MaxSizePolicy.FREE\_HEAP\_SIZE))

.setEvictionPolicy(EvictionPolicy.LRU).setTimeToLiveSeconds(2000));

@Cacheable(value = "userCache", key = "#id", unless = "#result==null")