1.Define the following cache strategies:

a) Cache Aside.

b) Read Through.

c) Write Through

d) Write behind

a) Cache-Aside

An application can emulate the functionality of read-through caching by implementing the cache-aside strategy. This strategy loads data into the cache on demand. ... When the item is next required, using the cache-aside strategy will cause the updated data to be retrieved from the data store and added back into the cache.

b) Read Through

Read-through cache sits in-line with the database When there is a cache miss, it loads missing data from database, populates the cache and returns it to the application. Both cache-aside and read-through strategies load data lazily, that is, only when it is first read.

c) Write Through

In this write strategy, data is first written to the cache and then to the database. The cache sits in-line with the database and writes always go through the cache to the main database.

d) Write behind

Write-behind is a caching strategy in which the cache layer itself connects to the backing database. This means that your applications need only ever connect to your cache layer, and the cache then reads from or updates the backing database as needed.

2) explain message queues and PUB-SUB queues. mention where they are used?

The **message queuing pattern** takes on a point-to-point approach. A message within the queue will be deleted once consumed, similar to the Post Office Protocol, where a message is deleted from the server once it is delivered. These queues enable asynchronous messaging.

In the event of a network issue that causes a delay in a message's delivery, like if a consumer is unreachable, that message will stay in the queue until it can be delivered. This means messages aren't necessarily delivered in any particular order. They are instead delivered on a first-available basis, which can boost efficiency in certain.

The **publish-subscribe pattern, often called the pub-sub pattern** involves publishers that produce ("publish") messages in different categories and subscribers who consume published messages from various categories they are subscribed to. Unlike point-to-point messaging, a message will only be deleted if it's consumed by all subscribers to the category.

Some message systems, such as Kafka, have a retention policy that ensures messages stay in the queue for a specified amount of time, even after they are consumed by all subscribers. Other providers include Google Cloud Pub/Sub messaging, which offers similar functionality. Both are examples of middleware message delivery services that offer scalability and decoupling potential for real-time data streams and workflows.

Like message queuing, publish-subscribe (commonly referred to as "pub-sub") messaging moves **information from producers to consumers**. ... Publish-subscribe messaging systems can support use cases in which multiple consumers receive each message and/or that messages are received in order by each consumer.