**Ans 1:**

Busy spin brings down the performance of the application because the thread keeps running till the condition mentioned in loop becomes false without releasing the CPU and it results in consuming resources also wastes CPU cycles.

In the code, the highlighted code is responsible for the busy wait condition.

**Graphical user interface, text, application, chat or text message

Description automatically generated**

To avoid it, we must utilize the wait() and notify() methods as ‘wait()’ will release the CPU and enables thread to wait for the notify() signal by other thread. And this will avoid the unnecessary consumption of the CPU cycles in busy spin.

**Ans 2:**

Version 1 implementation is better than version 2 due to the following reason. The main difference in both the implementations is the use of wait()[version 1] and sleep()[version 2] methods.

sleep doesn't release any lock or monitor while waiting, while wait releases the lock or monitor. Secondly, wait is more ideal to use in the inter thread communication, as compared to using sleep.

**In ‘Version-1’ implementation [Screenshot below], so ‘sharedObj.wait(100)’ will release sharedObj lock.**

Graphical user interface, text, application, email

Description automatically generated

While in version 2 [Screenshot below], ‘Thread.sleep(100)’ will not release object on a lock.

Text

Description automatically generated with low confidence

**Ans 3:**

**Select users.id, sum(Orders.volume) from Users, Orders**

**WHERE Users.id = Orders.user\_id**

**GROUP BY Orders.product**

**having sum(Orders.Volume) desc**

Indexing makes columns faster to query by creating pointers to where data is stored within a database. We can make use of a clustered indexes which are the unique index per table that uses the primary key to organize the data that is within the table. The clustered index ensures that the primary key is stored in increasing order, which is also the order the table holds in memory. Secondly, if there is an requirement to query based on the specific field, then non-clustered indexes can also be created.

To make the query fast, I will create the clustered index on the primary key column and the non-clustered index can also be created on the columns based on which the search needs to be done. They are used to increase the speed of queries on the table by creating columns that are more easily searchable.

**Ans 5:**

**5 a)** RabbitMQ is a messaging broker. It accepts messages from publishers which in this case service 1 API that has created the order, routes them and, if there were queues to route to, stores them for consumption or immediately delivers to consumers, if any. Consumers consume from queues. So, when a new order is added, assuming there are already messages ready in the queue, deliveries will start immediately.

**5 b)** To ensure that messages and broker definitions survive restarts, we need to ensure that they are on disk. The AMQP standard has a concept of durability for exchanges, queues and of persistent messages, requiring that a durable object or persistent message will survive a restart.

**5 c)** We can handle this by acknowledging the message as soon as the consumer gets it, before it starts working on it. That would avoid the RabbitMQ-related duplication issue at least. This is at-most-once delivery. But if the consumer crashes, the message is lost forever. So, you need to persist the message right before you ack it so you can recover it later and the consumer should remove it once it's complete. We can then have a single dedicated process that just works on those persisted messages. Or for that matter, handle them manually.

From the developer point of view, we can also add an attribute, say ‘**idempotencyId’** to the message body which is a guid and on the consumer side for each message this id is validated against the stored value in database, any duplicates are rejected

**Ans 6:**

One of the simplest ways to use Redis to lock a resource is to create a key in an instance. The key is usually created with a limited time to live, using the Redis expires feature, so that eventually it will get released. When the client needs to release the resource, it deletes the key. But there is a more efficient way of doing it by using the ‘Redlock’[in Redis] algorithm which implements DLM.  The algorithm claims to implement fault-tolerant distributed locks.