**Problem Statement**

**In my e-commerce application, I encountered an issue with managing stock for products. The problem occurs when multiple users simultaneously click on the same product to add it to their respective carts. This results in the product being added to both carts, despite having limited stock available. This situation is identified as a race condition.**

**Steps Taken to Identify the Problem**

**To confirm the occurrence of a race condition, I used JMeter to simulate 100 simultaneous requests. Initially, the stock for the product was 200. When a single user added the product to their cart, the stock correctly decreased to 199. However, upon firing 100 concurrent requests, I observed that the stock reduced to only 198 instead of the expected 100. This confirmed the presence of the race condition as a critical issue in my application.**

**Steps Taken to Fix the Issue**

**1. Using Transactions in SQL Server:**

**I initially implemented the Transaction feature provided by SQL Server to handle concurrent requests and ensure accurate stock management.**

**However, this approach resulted in a deadlock, and I have not yet thoroughly investigated the root cause of the deadlock.**

**2 . Implementing Semaphores:**

**To address the issue effectively, I implemented semaphores in my code to manage concurrent access to the stock.**

**After applying semaphores, I tested the application again using JMeter with 100 concurrent requests. The stock reduced accurately by 100, indicating that the issue was resolved.**

**Learning Material :**[**https://www.geeksforgeeks.org/race-condition-vulnerability/**](https://www.geeksforgeeks.org/race-condition-vulnerability/)[**https://www.geeksforgeeks.org/semaphores-in-process-synchronization/**](https://www.geeksforgeeks.org/semaphores-in-process-synchronization/)[**https://www.youtube.com/watch?v=MqnpIwN7dz0**](https://www.youtube.com/watch?v=MqnpIwN7dz0)