Submission Guidelines: Load and Stress Testing.

General Rules for All Submissions:

1. Context: Before any table, briefly describe your system and what constitutes a “user”, “transaction”, or “load” for your system.
2. Define your metrics: Clearly state what you are measuring (e.g., page load time, API response time, frames per second).
3. Explain Your Environment: Note if testing was done on a local machine.

Sample Testing Submission (SRS Format)

(Note: Do not include the text colored in red)

Load Test Results

System context: Campus Connect is a web-based portal that allows students to register for courses, view their grades, and access academic resources. Our load test simulated concurrent student access during peak online registration periods.

Table 1: Load Test Performance Metrics

Database load (record)	**Average Response Time (milliseconds)**	**System Performance Observation**
10,000 records	500	Instantaneous loading and search. UI is responsive
50,000 records	1800	Slight delay when applying complex filters
250,000 records	4200	Noticeable 2-3 second delay when loading the page; search takes 4+ seconds.
1,000,000 records	12,000+	Page load times are unacceptable. Search queries time out 30% of the time. The system is at its functional limit.

(Note: The specific data volumes you test should reflect your system's real-world scope. For example:

- *A student club membership system might test with 100 to 5,000 records.*
- *An e-commerce prototype might test with 5,000 to 50,000 product records.*
- *A social media-like platform might test with the population of the region or city, 100,000 to 200,000 record).*

The load test demonstrates that system performance degrades predictably as both the database size and user concurrency increase. With a moderate database of 50,000 records and light user traffic, the system performs well within our 2-second target. However, the core issue is revealed as the database scales to 250,000+ records; even with a stable number of concurrent users, the response time for loading and searching the product catalog degrades significantly to over 4 seconds, indicating that database query efficiency is the primary bottleneck, not the application server itself.

At the maximum test load of 1 million records, the system becomes practically unusable, with frequent timeouts confirming that the current database indexing and query design cannot support large-scale inventory operations. Therefore, optimization efforts must focus on implementing advanced database indexing, query caching, and pagination strategies to ensure performance remains acceptable as the company's product catalog grows.

Stress Test Results

System Context: InventoryPro is a web application for warehouse management. The stress test evaluates system behavior when both database size and concurrent user activity are pushed beyond normal operational limits simultaneously. This simulates worst-case scenarios in a local machine, such as inventory audits conducted during peak sales periods.

Table 1: Stress Test Performance and Breaking Point Analysis

Database Load	Concurrent Users & Actions	System Response	Observation & Failure Mode
500,000 records	10 Users performing: • Product Searches • Stock Updates • Report Generation	Response Time: 8-10 seconds	System is severely degraded but operational.
500,000 records	20 Users performing: • Product Searches • Stock Updates • Report Generation	Response Time: >15	Critical failures begin.
500,000 records	30 Users performing: • Product Searches • Stock Updates • Report Generation	Response Time: N/A (System Unresponsive)	Breaking Point Reached. Transaction log is full.
Recovery Test (Return to 50,000 records & 25 users)	10 Users performing: • Product Searches • Stock Updates • Report Generation	Response Time: < 3 seconds	Back to normal operation

(Note: To simulate concurrent users, you can use 2–3 laptops or PCs connected to your web server. Open multiple browser tabs or windows on each machine (each representing one user) and execute the defined actions simultaneously. This method effectively creates the necessary load for stress testing without requiring specialized load-testing software.)

The stress test reveals a significant vulnerability in the system when operating under a large database load. With the database containing 500,000 records, performance degrades severely, resulting in an 8-10 second response time, even with only 10 concurrent users performing standard operations. The situation escalates to critical failure with 20 users, where response times exceed 15 seconds, indicating that the database transaction layer is failing to keep pace.

The system's breaking point is reached at just 30 concurrent users, at which point it becomes completely unresponsive due to the transaction log filling up, demonstrating that the primary bottleneck is the database's capacity to handle concurrent write operations under heavy data volume.

The successful recovery test, where response times returned to under 3 seconds after reducing both the database load and user concurrency, confirms that the failure was due to load and not data corruption. This result underscores an urgent need for database optimization, including transaction log management, query tuning, and connection pooling, as the system in its current state cannot support moderate multi-user operations on a large dataset.