Student Name: Alex Student
Course: Computer Science - Advanced Topics
Date: December 18, 2025
Subject: Final Exam Submission

Question 1

Answer:

Database transactions are governed by the ACID properties to ensure reliability and data integrity. These properties are Atomicity, Consistency, Isolation, and Durability.

1. Atomicity: This property ensures that a transaction is treated as a single unit of work. Either all steps in the transaction complete successfully, or none of them do. If a failure occurs during the transaction, the database is rolled back to its state before the transaction started.
Real-world violation: Consider a bank transfer where money is deducted from Account A but the system crashes before adding it to Account B. Without atomicity, the money simply vanishes. Atomicity ensures that if the credit fails, the debit is rolled back.

2. Consistency: This ensures that a transaction brings the database from one valid state to another, maintaining all defined rules, constraints, and cascades.
Real-world violation: If a database requires every employee to belong to a valid department, a violation occurs if a transaction adds an employee with a non-existent department ID. Consistency prevents this transaction from committing.

3. Isolation: This property ensures that concurrent transactions occur independently without interference. The intermediate state of a transaction should not be visible to other transactions until it is committed.
Real-world violation: In an e-commerce inventory system, if two customers buy the last item simultaneously, a lack of isolation could allow both to "purchase" the item, leading to a negative inventory count (a race condition).

4. Durability: Once a transaction is committed, it remains committed even in the event of a system failure (e.g., power outage). The changes are permanently saved.
Real-world violation: A user receives a "Purchase Successful" message, but the server reboots immediately after. If the data wasn't written to non-volatile storage (disk), the purchase record is lost, yet the user believes it was successful.

Question 2

```
SELECT d.dept_name, AVG(e.salary) AS avg_salary
FROM departments d
INNER JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_id, d.dept_name
HAVING COUNT(e.emp_id) >= 10
ORDER BY avg_salary DESC
LIMIT 5;
```

Question 3

Answer:

Differences between Supervised and Unsupervised Learning
The fundamental difference between these two paradigms lies in the nature of the data and the goal of the training. **Supervised learning** utilizes labeled data, where the algorithm is trained on input-output pairs. The model learns a function that maps inputs to desired outputs. In contrast, **unsupervised learning** deals with unlabeled data. The algorithm must explore the data structure on its own to find hidden patterns or groupings without explicit instructions on what the output should look like.

Examples and Problem Suitability

1. Supervised Learning:
* *Examples:* Classification (Email Spam Detection) and Regression (Predicting House Prices).
* *Suitability:* This approach is best suited for problems where historical data is available with known outcomes, and the goal is to predict outcomes for new, unforeseen data.

2. Unsupervised Learning:
* *Examples:* Clustering (Customer Segmentation) and Association (Market Basket Analysis).
* *Suitability:* This approach is ideal for exploratory analysis when the ground truth is unknown. It works best for discovering structure within data, such as anomaly detection or reducing the dimensionality of complex datasets.

Question 4

Answer:
To solve this in linear time O(n), I will use a hash set to store cumulative sums. If the difference between the current cumulative sum and the target value exists in the set, a subarray with the specific sum exists.

```
def find_subarray_sum(arr, target):
    # Set to track cumulative sums, initialized with 0
    seen_sums = {0}
    current_sum = 0

    for num in arr:
        current_sum += num

        # If (current_sum - target) is in the set, a subarray exists
        if (current_sum - target) in seen_sums:
            return True

        seen_sums.add(current_sum)

    return False
```

Analysis:
* **Time Complexity:** O(n) because we pass through the array once, and set lookups are O(1).
* **Space Complexity:** O(n) to store the cumulative sums in the set.

Question 5

Answer:

Impact of Network Latency and Bandwidth
In distributed systems, network latency introduces delays in communication between nodes, directly affecting the response time perceived by the user and the speed of data replication. Bandwidth constraints limit the volume of data that can be transmitted per unit of time. High latency can lead to consistency issues (stale data) if replication is slow, while low bandwidth can cause bottlenecks during high-traffic periods.

Architectural Strategies
1. **Caching:** Implementing aggressive caching strategies (e.g., Redis or CDNs) closer to the user to reduce network calls.
2. **Data Compression:** Compressing data before transmission to optimize available pipe capacity.
3. **Asynchronous Communication:** Using message queues (like Kafka) so the UI remains responsive even if backend synchronization is slow.
4. **Local Processing (Edge Computing):** Moving computation to the client side or edge servers to reduce round-trip times.

Trade-offs
These optimizations often come with trade-offs. For instance, aggressive caching can lead to eventual consistency rather than strong consistency. We accept that a user might see slightly outdated data (Performance over Consistency) to ensure the system remains usable in a poor network environment.

Question 6

```
db.orders.aggregate([
    {
        $group: {
            _id: {
                category: "$product.category",
                month: { $dateToString: { format: "%Y-%m", date: "$order_date" } }
            },
            monthly_revenue: { $sum: "$total_amount" }
        }
    },
    {
        $sort: { monthly_revenue: -1 }
    }
])
```

Explanation:
1. **$group:** Groups documents by a composite ID containing both the product category and the formatted month.
2. **$sum:** Calculates the total revenue for each group.
3. **$sort:** Orders the results by monthly_revenue in descending order (-1).