Interview Questions - Database Performance Optimization Techniques

1. **What is database replication, and why is it important for performance?**

   Answer:
   Database replication involves creating copies of a database across multiple servers. This ensures high availability and fault tolerance. Replication improves performance by distributing read queries across multiple replicas, reducing the load on a single server. It is particularly useful in read-heavy applications where many users access the same data, such as content delivery systems or reporting applications. However, replication can add overhead for write operations because data needs to be synchronized across replicas.

2. **Can you explain the difference between sharding and partitioning in databases?**

   Answer:
   Both sharding and partitioning are techniques to split large datasets into smaller, more manageable chunks, but they differ in their approach:

   - **Sharding**: Involves distributing data across multiple databases or servers (shards), often based on a specific key (e.g., user ID). Sharding is used to scale horizontally, allowing the database to handle more traffic and data. Each shard operates independently and can be on a separate machine.

   - **Partitioning**: Refers to dividing a database table into smaller pieces (partitions) based on a range or list of values (e.g., by date). Partitioning is typically done within a single database instance, aiming to improve query performance by narrowing the search scope.

3. **How does the CAP theorem impact database performance decisions?**

   Answer:
   The **CAP theorem** states that in a distributed database system, you can achieve at most two of the following three properties:

   - **Consistency (C)**: All nodes see the same data at the same time.

   - **Availability (A)**: Every request will get a response, whether the data is the most recent or not.

   - **Partition Tolerance (P)**: The system will continue to function even if network partitions occur.

4. In performance optimization, you must decide which two of these properties to prioritize. For instance, in systems requiring high availability, you may choose eventual consistency (e.g., NoSQL databases like Cassandra), while in systems

needing strong consistency (e.g., banking), availability might be sacrificed during network partitions.

5. **What are some common types of indexes, and when would you use them?**

   Answer:

   - **B-Tree Indexes**: The most common type, ideal for equality and range queries (e.g., `SELECT * FROM users WHERE age > 30`). Use them for general-purpose indexing on columns with high cardinality (many distinct values).

   - **Hash Indexes**: Used for equality queries (e.g., `SELECT * FROM users WHERE id = 123`). They are faster for exact matches but do not support range queries.

   - **Full-Text Indexes**: Used for searching text-heavy fields for keywords or phrases. Ideal for content search applications.

   - **Bitmap Indexes**: Suitable for columns with low cardinality (few distinct values) like gender or boolean flags. They improve query performance in analytics databases.

6. **What is the difference between normalization and denormalization, and how do they affect performance?**

   Answer:

   - **Normalization**: The process of organizing a database into multiple tables to reduce redundancy and ensure data integrity. It minimizes the risk of anomalies but can increase the complexity of queries (more joins) and reduce performance for read-heavy operations.

   - **Denormalization**: Involves combining tables or adding redundant data to improve query performance by reducing the need for complex joins. However, this can lead to data inconsistency and increased storage requirements. It's useful in reporting systems where read speed is critical.

7. **How does connection pooling improve database performance?**

   Answer:
   Connection pooling involves reusing database connections from a pool of pre-established connections rather than creating and destroying connections for each request. This reduces the overhead of establishing connections repeatedly, which is expensive in terms of time and resources. Connection pooling improves throughput by allowing applications to handle a large number of database requests without the

latency of creating new connections every time.

8. **What is query optimization, and how would you approach optimizing a slow query?**

   Answer:
   Query optimization is the process of improving the performance of SQL queries by reducing their execution time and resource usage. To optimize a slow query:

   - **Analyze the execution plan**: Check if the database is performing full table scans or inefficient joins.
   - **Indexes**: Ensure that columns frequently used in WHERE, JOIN, and ORDER BY clauses are indexed.
   - **Avoid SELECT :* Select only the necessary columns.
   - **Use appropriate joins**: Use INNER JOINs instead of OUTER JOINs when possible to reduce unnecessary data retrieval.
   - **Limit Results**: Use LIMIT or pagination to reduce the size of result sets.
   - **Denormalize data** where applicable for read-heavy systems.

9. **What are materialized views, and how do they enhance performance?**

   Answer:
   A **materialized view** is a precomputed query result stored as a physical table. It improves performance by avoiding the need to recompute expensive queries each time they are needed. Instead of executing the query repeatedly, the result is retrieved from the materialized view, which is refreshed periodically. They are particularly useful in reporting and data warehousing applications where real-time data is not crucial.

10. **How would you handle large datasets in your application?**

    Answer:
    To handle large datasets:

    - **Data Partitioning and Sharding**: Split data across multiple storage systems or partitions for better scalability.
    - **Pagination**: For queries that return large result sets, implement pagination to retrieve data in chunks rather than all at once.
    - **Caching**: Use caching mechanisms like Redis or Memcached to store frequently accessed data and reduce database load.

- **Batch Processing**: For data manipulation, use batch processing to avoid overloading the database with many small operations.

11. **What are the trade-offs between consistency and performance in distributed databases?**

Answer:

The trade-off between consistency and performance arises from the CAP theorem. If a system prioritizes **consistency**, it may sacrifice availability or partition tolerance, resulting in slower performance during network partitions or high traffic (e.g., strong consistency in relational databases like MySQL). On the other hand, prioritizing **performance** (throughput) or **availability** can lead to eventual consistency, meaning data might not be immediately synchronized across all nodes, but it allows for faster reads and higher availability (e.g., eventual consistency in NoSQL databases like Cassandra).