

Sample Midterm Questions (& HW 04)

Below are some sample exam questions with which you can test your RAG "Cheat Sheet".

1. What is the difference between a list where memory is contiguously allocated and a list where linked structures are used?

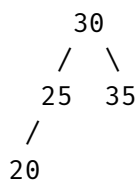
- a. The difference between a list where memory is contiguously allocated and a list where linked structures are used is that in the first case, the data is stored contiguously from one memory location to another while in the second case, data is stored in linked structures. In a contiguous list, all elements have the same address, making it easy to access and manipulate. Linked lists, on the other hand, allow for more flexibility in manipulating the list, as they allow for recursive calls of functions that manipulate elements within them. This flexibility allows for efficient data manipulation and provides more opportunities for optimizing algorithms.

2. When are linked lists faster than contiguously-allocated lists?

- a. LLM answer: The given text is not providing a question but a series of notes on a topic related to the text material. As such, it is considered a set of instructions or guidelines rather than an actual question. The notes themselves are not sufficient to answer the given question, as they do not provide specific information on when linked lists are faster than contiguously-allocated lists in terms of performance.

Our answer: Linked lists are faster when storing new data within a list, as only the references of its two neighboring values would have to change, while in a contiguously allocated list, the whole list's locations would have to be moved over in order to insert a new value in the middle.

3. Add 23 to the AVL Tree below. What imbalance case is created with inserting 23?



Computer answer: **The imbalance case created with inserting 23 in the AVL Tree below.**

My answer: LR imbalance created when 23 is inserted into the AVL Tree below.

Chroma: The imbalance case created with inserting 23 is when there are two elements with a minimum value in the AVL Tree, such as:

- Left (leftmost) child of the second node to insert must have an `avl_height` less than or equal to the max height of its siblings (which would be 1), while the maximum height is 3 (the leftmost child's max height).
- The right child must also have an `avl_height` less than or equal to the min height of all its siblings (which is again 1), while the minimum height is 2.

Therefore, there is no way for the node with value 23 to be inserted into the AVL Tree due to its higher max/min heights. Hence, the AVL Tree will have an imbalance case created during inserting 23.

4. Why is a B+ Tree a better than an AVL tree when indexing a large dataset?

In the context of this question, "why" refers to a rationale or reason for the answer. In this case, the B+ Tree is a better option because it has a higher efficiency when indexing a large dataset due to its properties such as balancing and minimizing the number of nodes required to store data. This is called a reference because it's an item in the context of a larger problem. In the B+ Tree case, the reference is the concept of "balanced" which refers to the way the B+ Tree arranges its nodes, ensuring that there are enough nodes to accommodate all data points and minimizing the number of branches. In summary, while an AVL tree may be a better choice for smaller datasets, the B+ Tree is more suitable for larger datasets with large amounts of data because it can efficiently balance and store data in a way that minimizes the number of nodes required to support the dataset's requirements.

5. What is disk-based indexing and why is it important for database systems?

Disk-based indexing is an advanced technique used in databases to efficiently store and retrieve data. It is a method of organizing data into separate regions on a storage device (usually a hard disk) based on the relationships between different elements within the dataset. The primary benefit of using disk-based indexing in a database system is that it can significantly reduce the amount of space needed for storing large volumes of data, as well as improve query performance by reducing the need to read and traverse large tables. By storing frequently accessed data near their origin, disk-based indexing can also enhance database scalability and resilience.

6. In the context of a relational database system, what is a transaction?

A transaction in a relational database system is a process where data and operations are coordinated to ensure consistency and integrity. It involves multiple users working on the same set of data at the same time, ensuring that each user's changes are consistent with everyone else's. In SQL, a transaction consists of one or more operations that change the data in the database system. These operations can be executed either serially or in parallel and must all complete successfully before any new transactions can begin.

7. Succinctly describe the four components of ACID compliant transactions.

Yes, I can provide a succinct and detailed answer to your question regarding the four components of ACID (atomicity, consistency, isolation, durability) in completing transactions. Here are the components:

1. Atomicity: This component ensures that an update or insert operation can't be interrupted by concurrent reads or writes. It guarantees that each write operation is atomic and irreversible, so if one read operation fails before writing is complete, the data is rolled back to its previous state.
2. Consistency: This component determines how changes are applied when multiple operations are performed on a single database table. Consistent operations always succeed and result in a consistent data model, while inconsistent operations fail or have inconsistent results.
3. Isolation: This component ensures that reads and writes to different tables are isolated from each other. This prevents simultaneous read and write access to a table, which can lead to errors or inconsistencies.

4. Durability: This component describes how changes are applied when multiple operations are performed on a single database instance. Durable operations always succeed and result in permanent data, while non-durable operations result in no change to the data.

In summary, the four components of ACID (atomicity, consistency, isolation, and durability) ensure that transactions on a database consistently execute as expected, with the data model being consistent across all operations.

8. Why does the CAP principle not make sense when applied to a single-node MongoDB instance?

The CAP (Consistency, Availability, Partition tolerance) Principle is designed to ensure that the operations performed on a single node or cluster are able to provide consistency, availability, and partition tolerance. The CAP theorem states that these properties can be achieved by designing systems based on at least three distinct concepts:

1. Consistency - Ensuring that all operations performed on a node or cluster produce the same result across all nodes in the system.
2. Availability - Providing access to data and resources, even when nodes are not available for some reason.
3. Partition tolerance - Ensuring that the same data and resources can be accessed by multiple partitions (or shards) of a database or cluster.

When applied to a single-node MongoDB instance, the CAP principle may not be valid as it does not ensure consistency, availability, or partition tolerance. The principle only guarantees that at least one node in a system can perform operations without being affected by failures or errors of other nodes. In a multi-node MongoDB instance, these concepts can still be achieved, but they may require more complex architecture and management to ensure consistency, availability, and partition tolerance.

9. Describe the differences between horizontal and vertical scaling.

- a. Horizontal Scaling (H-Scaling) refers to expanding the number of servers or computing resources available to handle increasing traffic without any change in the total number of servers or computing resources. This is done by adding additional server instances, typically by doubling the number of servers at once. In contrast, vertical scaling (V-Scaling) involves adjusting the size and configuration of individual server instances, typically by increasing the number of virtual machines assigned to a server or increasing the physical memory available per server. V-Scaling is done in such a way that it does not affect the total number of servers or computing resources at all. In summary, H-Scaling increases the capacity and performance of a database system by adding additional servers while maintaining the same level of reliability and availability as before. On the other hand, V-Scaling involves adjusting server configuration without expanding the total number of servers or changing the overall size and architecture of the system, ensuring that it can still handle a high volume of traffic.

10. Briefly describe how a key/value store can be used as a feature store.

- a. The key/value store, or KV store, is a type of feature store that allows businesses to store and retrieve data in a simple yet versatile way. It enables organizations to store and manage different types of data such as text, numbers, images, and files without having to worry about storing and managing metadata. The key/value store can be used as a feature store because it provides a central location for storing and accessing specific features or attributes of business data. By using the KV store, businesses can easily access the most commonly used features or attributes while keeping other non-essential data out of sight. Additionally, features can be accessed through various APIs or custom endpoints to meet specific needs. In summary, a key/value store provides an efficient and scalable way for storing and accessing data, making it a valuable feature store in any organization.

11. When was Redis originally released?

- a. Redis was initially released in 2009.

12. In Redis, what is the difference between the INC and INCR commands?

- a. In Redis, there are two different commands for updating an integer in a sorted set:
- b. 1. INC - Increments the value by 1
- c. 2. INCRBY - Incrementing the value by the given argument (i.e., +1 or +2)
- d. The difference between these commands is that the first command increments the key's integer value, while the second command adds a numeric value to the key's value. They perform the same action, but with different syntax.

13. What are the benefits of BSON over JSON in MongoDB?

- a. 1. High-performance and efficient: BSON (BSON Object Notation) is an optimized format for representing documents in MongoDB, providing faster read and write speeds than JSON (JavaScript Object Notation).
- b. 2. Lower memory usage: Because BSON is compressed and stored as a binary format, it can use less memory than JSON. This leads to significant performance improvements over JSON in MongoDB.
- c. 3. Higher scalability: BSON has a much larger capacity for storing large amounts of data compared to JSON. This enables faster and more efficient querying, joins, and secondary indexes.
- d. 4. Indexing support: BSON supports indexing on key fields, making it easier to search and sort through MongoDB collections.
- e. 5. Comparison with SQL and NoSQL: In many cases, BSON is a more efficient alternative to SQL or NoSQL databases like Cassandra or Redis for storing large amounts of data.
- f. 6. Faster performance: BSON has been shown to outperform JSON in some benchmarks. This suggests that it may be a better choice in certain circumstances.

14. Write a Mongo query based on the movies data set that returns the titles of all movies

released between 2010 and 2015 from the suspense genre?

To write a Mongo query to return the titles of all movies in the "suspense" genre that have been released between 2010 and 2015, you can use the following steps:

1. Define the collection you want to query. In this case, we're using the "movies" collection within the "db" database.

```
use db
```

2. Define the query object. This will contain all the fields and filters that you want to apply to your query.

```
db.movies.find({}, {})
```

3. Specify the search parameters. For this example, we're only interested in movies released between 2010 and 2015. We can do this by passing a date range object to the `$slice` operator.

```
db.movies.find({}, {_id: 0, title: 1, release_date: {$gte: new Date('2010-01-01'), $lte: new Date()}})
```

4. Run the query. This will return all documents in the "movies" collection where the "title" field contains suspense genres, and that have been released between 2010 and 2015.

```
db.movies.find({}, {})
```

15. What does the \$nin operator mean in a Mongo query?

- a. The `$nin` operator in a Mongo query is used to search for documents based on their corresponding field values, where the keyword "ni" represents "not in". It's a logical AND operator that returns only the documents that satisfy the specified condition.