Theoretical Questions:

What is indexing in a database, and why is it important?

Indexing refers to maintaining a lookup table for the database to increase the read performance. It usually involves a sorted list for quick discovery of a particular row. It is important because it helps in improving query performance

Explain the difference between clustered and non-clustered indexes.

A clustered index defines the order in which data is physically stored in a table whereas a non-clustered index is stored at one place and table data is stored in another place. A clustered index can have only one clustered index whereas a table can have multiple non-clustered indexes.

When should you consider creating an index on a column? Are there any scenarios where indexing can have a negative impact?

An index should be created on a column if it has frequent searches, joins, sorting and grouping to increase query performance. Indexing has negative impact when there is frequent updates to the table, the table is large and columns are rarely queried

How does indexing affect data modification operations, such as insert, update, and delete?

When using insert, update and delete, the process takes longer time with indexing because the index needs to be updated on every operation.

What are some alternative data structures or indexing techniques used in modern databases?

Some of the alternative data structures or indexing techniques used in modern databases are “In Memory Indexes” and “Column Store Indexes”.

Understanding Isolation Levels in Database

A. Write a brief summary describing the concept of isolation levels and their significance in database systems. Explain how different isolation levels help balance data consistency, concurrency, and performance.

Answer: Isolation levels are like rules that databases follow when handling multiple transactions at the same time. Imagine you're in a store. You're looking at an item, and at the same time, someone else wants to buy the same thing. Isolation levels help decide how the store handles situations like this.

There are different levels of isolation:

1. Read Uncommitted: It's like letting someone see your item before you decide to buy it. Even if another person is changing the data, you can still see it. But there's a risk because you might see incomplete or incorrect data.

2. Read Committed: Here, you can only see the item once it's been properly updated. It's like waiting for the item to be ready before you look at it. This ensures you see accurate data, but there's a chance you might see different things if you look at the same time as someone else.

3. Repeatable Read: This level ensures that once you see an item, it won't change until you're done. It's like having a special room where once you enter, no one else can change the item until you're finished. This helps keep things consistent, but it might slow down others who want to see the same item.

4. Serializable: This is like making sure nobody can touch the item until you're done with it. It's the strictest level where everything happens one after the other. This ensures perfect consistency, but it might slow down the whole process if many people want to see or buy things at the same time.

Choosing the right isolation level is crucial because it balances different things:

- Consistency: How accurate and reliable the data is.
- Concurrency: How many people can access or change the data at the same time.
- Performance: How quickly things get done without slowing down the system.

So, different isolation levels help find the right balance between making sure data is correct, allowing many people to work together, and keeping things running fast.

B. Create a comparative table or chart that presents the different isolation levels, their characteristics.

Isolation Level	Description	Consistency	Concurrency	Performance
Read Uncommitted	Lets you see data that's still being changed, so it might not be accurate.	Low	High	High
Read Committed	Only shows data that's fully updated, but if you look at the same time as others, you might see different things.	Medium	Medium	Medium
Repeatable Read	Once you see data, it won't change until you're done, so it's always consistent.	High	Low (for same data)	Low (overall)
Serializable	Keeps transactions completely separate, so everything happens one after the other, ensuring perfect consistency.	Highest	Lowest	Lowest

C. Discuss the advantages and disadvantages of each isolation level, highlighting their impact on data consistency, concurrency control, and overall database performance.

1. **Read Uncommitted**:
    - *Advantages*:
        - High concurrency: Many users can access and modify data simultaneously without waiting for locks.
        - High performance: Since there are minimal restrictions, transactions can execute quickly.
    - *Disadvantages*:
        - Low data consistency: You might see incomplete or incorrect data, leading to potential errors.
        - Risk of dirty reads: Reading uncommitted data might result in seeing changes that are later rolled back, leading to inconsistency.

2. **Read Committed**:
    - *Advantages*:
        - Better data consistency: Only committed data is visible, reducing the risk of seeing incomplete or incorrect information.
        - Moderate concurrency: Multiple users can still access and modify data simultaneously, but with fewer chances of conflicts.
    - *Disadvantages*:
        - Potential for non-repeatable reads: Data might change between reads within the same transaction, leading to inconsistency.
        - Medium performance: The need to acquire and release locks may slightly impact performance compared to Read Uncommitted.
3. **Repeatable Read**:
    - *Advantages*:
        - High data consistency: Once data is read, it remains unchanged until the transaction is complete, ensuring consistency.
        - Low risk of non-repeatable reads: Data remains consistent throughout the transaction, reducing the chance of seeing different values.
    - *Disadvantages*:
        - Reduced concurrency: Locking data to maintain consistency can lead to concurrency issues, as other transactions may need to wait.
        - Lower performance: Holding locks for the duration of transactions can decrease concurrency and impact overall performance.
4. **Serializable**:
    - *Advantages*:
        - Highest data consistency: Transactions are completely isolated, ensuring perfect consistency and preventing all concurrency issues.
        - Lowest risk of anomalies: No phenomena like dirty reads, non-repeatable reads, or phantom reads can occur.
    - *Disadvantages*:
        - Lowest concurrency: Transactions are executed serially, limiting the number of concurrent transactions and potentially slowing down the system.
        - Poor performance: Due to strict locking and isolation, transactions may take longer to complete, leading to slower overall performance.

D. Explain how the choice of isolation level can affect the behavior of database transactions and the potential challenges faced by developers in different scenarios.

Imagine you're playing a game with your friends. You all want to do different things at the same time, like move around, pick up items, or trade with each other. These actions are like transactions in a database.

Now, the game has rules, just like isolation levels in a database. These rules decide how you interact with each other. Let's see how different rules affect the game:

1. **No Rules (Read Uncommitted)**:
   - You can do anything you want, whenever you want. But this can lead to chaos. For example, someone might take an item from you while you're still using it.
2. **Some Rules (Read Committed)**:
   - You can do things, but you have to wait for others to finish before you can do something similar. For example, if your friend is using a tool, you have to wait for them to finish before you can use it.
3. **More Rules (Repeatable Read)**:
   - Now there are more restrictions. Once you start doing something, nobody else can change things until you're done. This keeps things more organized, but it might slow down the game because you have to wait for others.
4. **Strict Rules (Serializable)**:
   - Now there are very strict rules. You have to finish what you're doing before anyone else can do anything. This ensures everything is done properly, but it can make the game very slow because everyone has to wait for their turn.

Now, let's talk about challenges:

- If there are no rules (Read Uncommitted), things can get messy quickly. You might see changes that aren't final, leading to confusion.
- With some rules (Read Committed), you might end up waiting a lot because you can't do anything until others finish.
- More rules (Repeatable Read) ensure things are stable, but it slows down the game because everyone has to wait for you to finish.
- Strict rules (Serializable) ensure everything is perfect, but the game becomes slow and boring because you spend more time waiting than playing.

So, developers face a challenge in choosing the right level of rules (isolation level) for their game (database). They need to balance between letting players (transactions) do things freely and keeping the game (database) running smoothly without chaos or boredom.

E. Provide recommendations for selecting the appropriate isolation level based on specific requirements, such as transactional consistency, concurrency, and performance optimization

- **Read Uncommitted**: Use when high concurrency and performance are critical, and data consistency is less important.
- **Read Committed**: Suitable when balancing consistency and concurrency is essential, and moderate performance is acceptable.
- **Repeatable Read**: Opt for scenarios where maintaining strong consistency is crucial, even if it means sacrificing some concurrency and performance.
- **Serializable**: Choose when absolute data consistency is paramount, and performance can be sacrificed for transactional integrity.