**Department of Computer Science and Engineering Faculty of Engineering  
University of North Texas**

**MidTermExamination II CSCE5350 Fall2023 Time Allowed: 1hour 30 minutes**

**Answer All questions**

1. A transaction is a *unit* of program execution that accesses and possibly updates various data items. Answer the following questions based on the transaction processing concept.

1. Explain how a concurrent execution of transactions improves the performance of the application.

**Ans:**

* Increased Throughput: By allowing multiple transactions to execute concurrently, the DBMS can process more transactions in a given period, leading to higher throughput. This is particularly beneficial for applications with a high volume of transactions or workloads that can be parallelized.
* Efficient Resource Utilization: When transactions execute concurrently, they can utilize different system resources simultaneously, such as CPU cores, memory, and disk I/O bandwidth. This helps to maximize the utilization of available resources, reducing resource contention and improving overall system efficiency.
* Overlapped I/O and Computation: Concurrent transaction execution allows for the overlap of I/O operations (e.g., reading or writing data from/to disk) and computational operations (e.g., processing data in memory). While one transaction is waiting for I/O operations to complete, other transactions can continue executing computational tasks, effectively using the available resources more efficiently.
* Reduced Response Times: By allowing multiple transactions to execute concurrently, the DBMS can distribute the workload across multiple resources, potentially reducing the response times for individual transactions. This is particularly beneficial for applications that require low latency or have strict performance requirements.
* Scalability: Concurrent transaction execution enables the DBMS to take advantage of additional computing resources, such as multiple CPU cores or distributed computing environments. As the workload increases, more resources can be added to handle the concurrent execution of transactions, improving scalability and supporting higher transaction volumes.

1. What is the importance of preserving ACID properties of a transaction? Explain each property and its purpose.

**Ans:**

Preserving the ACID properties (Atomicity, Consistency, Isolation, and Durability) of a transaction is crucial in a database management system (DBMS) to ensure data integrity, reliability, and consistency. Here's a brief explanation of each ACID property and its importance:

* Atomicity:
  + Atomicity ensures that a transaction is treated as an indivisible unit of work, meaning that either all operations within the transaction are completed successfully, or none of them are applied.
  + If a transaction fails or is interrupted for any reason, the DBMS must roll back all changes made by the transaction, leaving the database in its original, consistent state.
  + Atomicity prevents partial updates or incomplete transactions from corrupting the database.
* Consistency:
  + Consistency ensures that a transaction transforms the database from one valid state to another valid state, according to the defined rules and constraints.
  + The DBMS enforces data integrity constraints, such as primary key constraints, foreign key constraints, and other business rules, to maintain data consistency.
  + If a transaction violates any of these constraints, the DBMS should abort the transaction and roll back any changes to preserve data consistency.
* Isolation:
  + Isolation ensures that concurrent transactions operate independently and do not interfere with each other, even if they access or modify the same data.
  + The DBMS uses concurrency control mechanisms, such as locking or versioning, to isolate transactions from each other, preventing data corruption or inconsistencies that may arise due to concurrent access.
  + Isolation guarantees that the outcome of executing a set of transactions concurrently is the same as executing them in a serial order.
* Durability:
  + Durability ensures that once a transaction is committed, its effects persist in the database, even in the event of system failures, power outages, or crashes.
  + The DBMS uses techniques like logging and checkpointing to persistently store committed transaction data on non-volatile storage (e.g., disk), ensuring that data is not lost in case of system failures.
  + Durability guarantees that committed data remain intact and recoverable, even after unexpected system disruptions.

Preserving the ACID properties is crucial for maintaining data integrity, consistency, and reliability in a DBMS. It ensures that transactions are processed correctly, without data corruption or inconsistencies, even in the presence of concurrent access, system failures, or other exceptional conditions. By adhering to these properties, the DBMS can provide a reliable and trustworthy environment for managing and processing data, which is essential for mission-critical applications and business operations.

1. Check that the following schedule is conflict serializable by using a Precedence Graph. Write down the serial schedule that the following schedule is conflict equivalent with if it is conflict-serializable.

A table of text on a white background

Description automatically generatedA diagram of a diagram

Description automatically generated

This precedence graph has a loop. Therefore, this schedule is not conflict serializable.

d. Explain why the following schedule is not recoverable.

A table with text on it

Description automatically generated

2. This question is based on query processing and optimization.

1. What is the purpose of query evaluation in the query processing steps?

**Ans:** Key purposes of query evaluation include:

* Retrieving the requested data by accessing relevant tables and applying filters/conditions.
* Executing the query plan efficiently, minimizing the time and resources required.
* Leveraging parallelism and pipelining techniques to improve performance.
* Handling complex queries involving subqueries, joins, aggregations, etc.
* Integrating with the storage engine to access and manipulate data on disk or in memory.
* Error handling and generating the final result set for presentation or further processing.

1. How do we optimize a given query?

**Ans**: In Ch 16 – Slide 16.5

1. Explain the difference in record searching in a linear file scan and index scan.

**Ans:** In a linear file scan, every record in the file is sequentially searched one by one until the desired record is found. This method can be slow for large files since it requires scanning through all records, making it inefficient for frequent searches.

On the other hand, in an index scan, a data structure called an index is used to locate records quickly. The index stores pointers to the actual records based on key values. Instead of scanning the entire file, the index is searched for the specific key, and then the corresponding record is retrieved directly. This method is much faster than a linear scan, especially for large datasets, as it reduces the number of disk accesses required to locate the desired record.

1. Compute the cost of the query if you want to select a key that is not a candidate key in the relation. However, there is a secondary index defined on the search key. Assume the index height is 10, and time it takes to transfer one block is 10ms, and the time taken for one seek is 15ms. Assume there will be 10 matches for your query.

**Ans**: Ch 15 – Slide 15.11

1. Assuming you have two relations Instructors and a department, you need to compute the cost of joining using a nested loop for the following configuration and condition.  
   Configuration: Instructor relation has 20,000 rows and is stored in 100 blocks. The department table has 20 rows and is stored in one block.  
    Condition:  
   Only one block from each relation can be accommodated in the memory.

**Ans**: Ch 15 – Slide 15.18

3. Database indexing is a vital aspect of application development.

1. Compare and contrast B+-tree file organization and B+-tree indexing.

**Ans:** B +-tree file organization and B+-tree indexing are both techniques used in database systems to efficiently store and retrieve data, particularly for range queries and searches. However, they serve different purposes and operate at different levels within the database system. Here's a comparison and contrast between the two:

* Purpose:
  + B+-tree file organization: The primary purpose of B+-tree file organization is to physically store the data records in a database file efficiently on disk. It organizes the data in a tree-like structure to minimize disk access time for queries and ensure that data can be retrieved quickly.
  + B+-tree indexing: B+-tree indexing, on the other hand, is used to create an index structure on one or more columns of a table to facilitate faster data retrieval. It creates a separate data structure that stores pointers to the actual data records, enabling the database system to locate specific records quickly based on the indexed columns.
* Structure:
  + B+-tree file organization: In B+-tree file organization, the tree structure encompasses both the index nodes and the leaf nodes. Each leaf node contains actual data records sorted by their keys, while the index nodes contain pointers to child nodes or leaf nodes.
  + B+-tree indexing: In B+-tree indexing, the tree structure consists of index nodes only. These index nodes store pointers to either other index nodes or the actual data records. Leaf nodes are typically not used in B+-tree indexing because the pointers directly lead to the data records.
* Storage Location:
  + B+-tree file organization: Data records are stored directly within the leaf nodes of the B+-tree structure. Each leaf node contains a range of consecutive data records sorted by their keys.
  + B+-tree indexing: The index structure is stored separately from the actual data records. Index nodes contain pointers to the corresponding data records or other index nodes, allowing the database system to navigate through the index to locate the desired data efficiently.
* Usage:
  + B+-tree file organization: Used to physically organize and store the data records on disk to optimize data retrieval and disk access time.
  + B+-tree indexing: Used to create efficient access paths to the data records based on specific columns or keys, speeding up search operations and query processing.

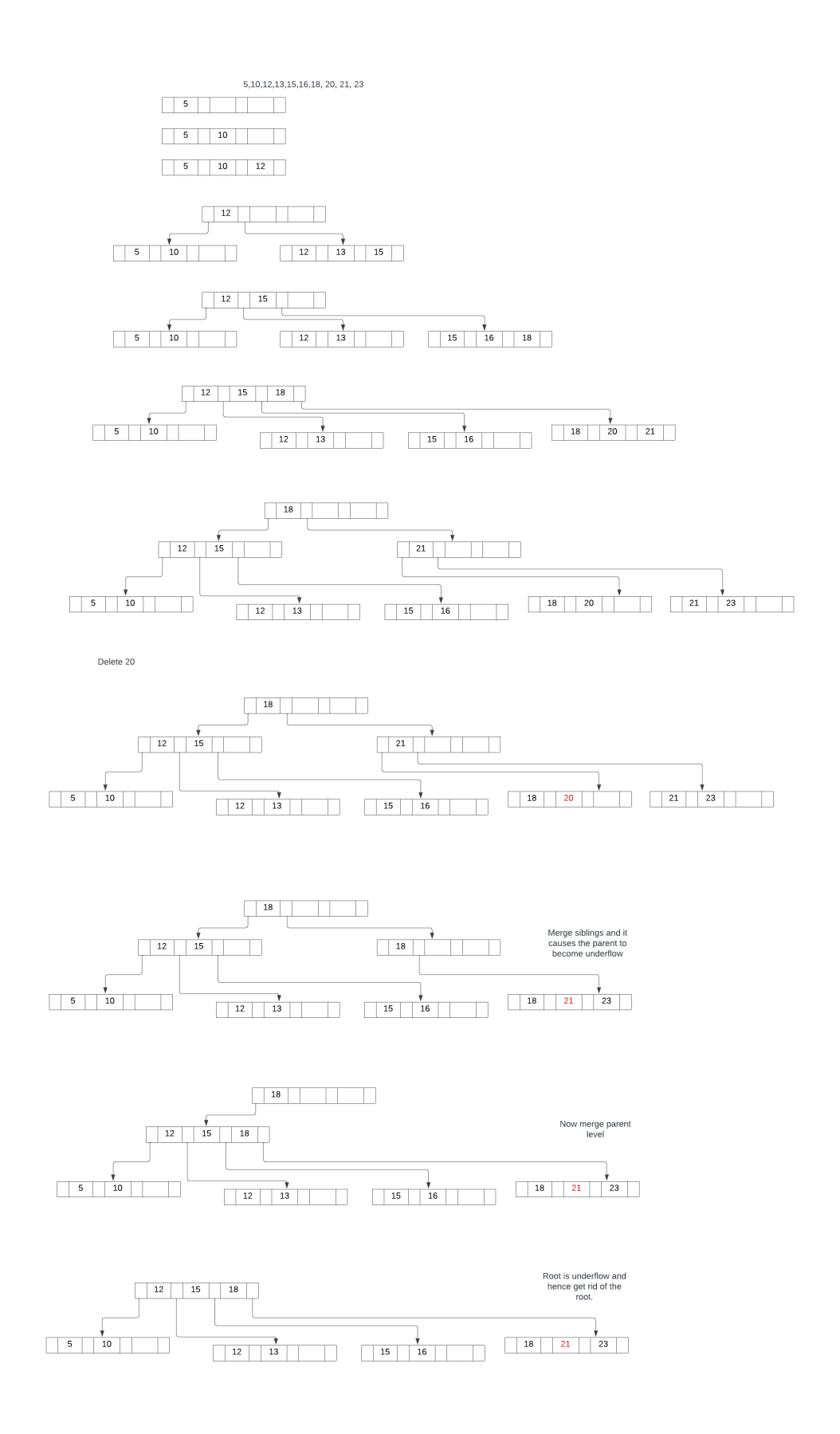
In summary, while both B+-tree file organization and B+-tree indexing utilize B+-tree structures, they serve different purposes within a database system. B+-tree file organization focuses on efficiently storing and organizing data records on disk, while B+-tree indexing is used to create index structures for faster data retrieval and query processing.

1. What is the difference between dense and dense-clustering index?

**Ans:**

* Scope of Index Entries:
  + Dense Index: Contains an index entry for every record.
  + Dense-Clustering Index: Organizes data records based on index keys, coupling index entries with actual records.
* Mapping of Index Entries to Data Records:
  + Dense Index: Each index entry directly points to a data record.
  + Dense-Clustering Index: Index entries may contain pointers to data records and actual records themselves.
* Size and Storage Overhead:
  + Dense Index: Can be large due to an index entry for every record.
  + Dense-Clustering Index: Generally more compact due to clustered storage of data records.
* Query Performance:
  + Dense Index: Effective for point queries.
  + Dense-Clustering Index: Efficient for range queries and sequential access.

1. InsertthefollowingsetofintegerstoaB+tree.Thetreenodehasafanoutof4 (which is n=4). Show each insertion in a separate tree.
   1. 5,10,12,13,15,16,18, 20, 21, 23
2. Delete item 20 from the final B+ tree generated in the above question. Show each step clearly.



A screenshot of a white sheet

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated