SET – 1

2)a) Explain about Timestamp based protocol.

Timestamp-Based Protocol in DBMS:

Timestamp-based protocol is a concurrency control method that uses **timestamps** to manage the order of transactions. Each transaction is assigned a **unique timestamp** when it starts, which determines its order of execution.

Key Points:

- Ensures **serializability** by ordering transactions based on timestamps.
- Each data item maintains:
  - **Read Timestamp (RTS)** – latest time it was read.
  - **Write Timestamp (WTS)** – latest time it was written.
- A transaction T is allowed to:
  - **Read(X)** only if `TS(T) ≥ WTS(X)`
  - **Write(X)** only if `TS(T) ≥ RTS(X)` and `TS(T) ≥ WTS(X)`
- If rules are violated, the transaction is **rolled back**.

Advantage:

- No deadlocks (no locking mechanism used).

Disadvantage:

- More rollbacks compared to locking protocols.

2)b) Write about Transaction Properties.

Transaction:

A **transaction** is a sequence of database operations performed as a single logical unit of work.

---

1. Atomicity

- Ensures all operations in a transaction are completed or none.
- If any operation fails, changes are rolled back.
- Maintains database in a consistent state.

---

2. Consistency

- Preserves database rules (e.g., constraints).
- Transforms data from one valid state to another.
- No violation of integrity occurs after the transaction.

---

3. Isolation

- Transactions execute independently.
- Intermediate results are hidden from other transactions.
- Prevents data conflicts in concurrent execution.

4. Durability

- Committed changes are saved permanently.
- Survives system crashes or power failures.
- Ensures data is stored in non-volatile memory.

5)a) Write about ISAM.

ISAM (Indexed Sequential Access Method) is a database storage technique optimized for fast reads via indexed and sequential access.

Structure:

- **Data**: Sorted sequentially by key in fixed blocks.
- **Indexes**: Primary index (maps keys to block addresses); optional secondary indexes.

Pros:

- Fast read access for static/semi-static data (e.g., archives, reports).
- Simple, predictable performance.

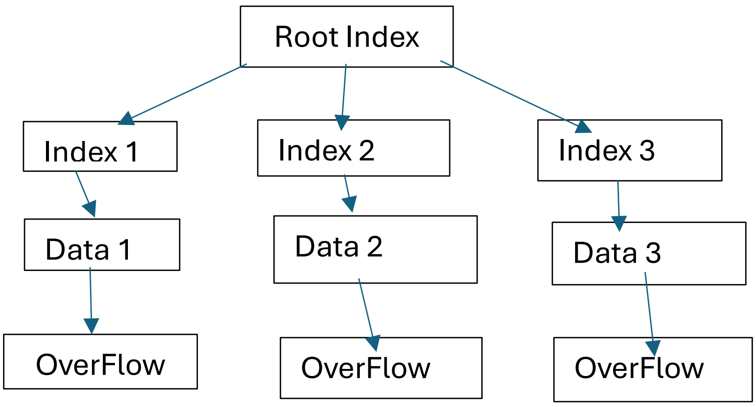
Cons:

- Slow writes due to overflow management.
- Fragmentation over time, needing maintenance.

Use Cases: Legacy systems, read-heavy environments with minimal data changes.

Modern Alternatives:

- **B+ Trees**: Auto-balancing, better for dynamic data.
- **VSAM**: Enhanced IBM version with improved space management.



5)B) Compare any two File Organizations.

No.	Aspect	Sequential File Organization	ISAM (Indexed Sequential Access Method)
1.	**Storage Structure**	Records are stored **one after another** in a fixed order, usually based on a key field.	Stores data **sequentially** and uses an **index** with pointers to access records.
2.	**Access Type**	Supports only **sequential access**; suitable for batch processing.	Supports **both sequential and direct access** using index pointers.
3.	**Insertion & Deletion**	Inserting or deleting requires **rearranging the entire file**, which is time-consuming.	Uses an **overflow area** for insertions, making it easier but requires **periodic reorg**.
4.	**Performance**	Efficient only for full scans; **slow for searches** and updates in large files.	**Fast for reads**, especially in **read-heavy workloads**; less efficient for frequent writes.
5.	**Suitability**	Best for **static data** and **simple applications** like log files or backups.	Suitable for **static or read-mostly environments** like report generation systems.

6) Write the problems related to Decomposition.

Problems Related to Decomposition in DBMS:

Decomposition is the process of breaking a relation into two or more sub-relations. While it helps remove redundancy and anomalies, it may also lead to the following problems:

---

1. **Loss of Information (Lossy Decomposition)**
   - Some original data may be **lost** if the decomposition is not done properly.
   - The original relation **cannot be recovered** through natural joins.
2. **Dependency Preservation Problem**
   - All **functional dependencies** from the original relation **may not be preserved** in the decomposed relations.
   - Makes **constraint enforcement** difficult.
3. **Join Dependency Issues**
   - Rejoining decomposed relations may produce **extra or missing tuples**.
   - Leads to **incorrect query results**.
4. **Increased Query Complexity**
   - Queries may need to **join multiple tables**, increasing **execution time** and **processing cost**.

Set – 2

2)a) Explain ACID properties.

1. Atomicity

- Ensures all operations in a transaction are completed or none.
- If any operation fails, changes are rolled back.
- Maintains database in a consistent state.

---

2. Consistency

- Preserves database rules (e.g., constraints).
- Transforms data from one valid state to another.
- No violation of integrity occurs after the transaction.

---

3. Isolation

- Transactions execute independently.
- Intermediate results are hidden from other transactions.
- Prevents data conflicts in concurrent execution.

---

4. Durability

- Committed changes are saved permanently.
- Survives system crashes or power failures.
- Ensures data is stored in non-volatile memory.

2)b) Explain conflict serializability with an example?

Conflict Serializability in DBMS:

Conflict serializability ensures that a schedule of concurrent transactions is equivalent to some serial schedule, based on conflicting operations.

Conflicting Operations:

Two operations conflict if:

1. **They belong to different transactions,**
2. **They access the same data item,**
3. **At least one of them is a write.**

Example:

Consider the following schedule:

T1	T2
Read(A)	
	Read(A)
Write(A)	
	Write(B)

Conflicts:

- **T1.Write(A) conflicts with T2.Read(A) ⇒ T2 → T1**
- **T2.Write(B) has no conflict with T1 (they access different items)**

Precedence Graph:

- **Nodes: T1, T2**
- **Edge: T2 → T1**

Since the graph has no cycle, the schedule is conflict serializable.

Conclusion:
The schedule is equivalent to the serial order: T2 followed by T1.

5)a) Explain about inserting node in B+trees

Inserting a Node in B+ Trees:

Insertion in a B+ tree maintains the sorted order and balance of the tree. B+ trees store data only in leaf nodes, while internal nodes hold keys for navigation.

Steps for Insertion:

1. **Locate the Leaf Node**
   - **Traverse the tree from the root to the appropriate leaf node where the key should be inserted.**
2. **Insert into Leaf Node**
   - **If there is space, insert the key in sorted order.**
3. **Split if Overflow Occurs**
   - **If the node overflows (i.e., exceeds maximum capacity):**
     - **Split the node into two.**
     - **Move the middle key to the parent node for redirection.**
4. **Repeat Splitting if Needed**
   - **If the parent also overflows, repeat the split up to the root.**
   - **If the root splits, a new root is created, increasing the tree height.**

Example:

Suppose the order of B+ tree is 3 (max 2 keys per node):

Insert keys: 10, 20, 5, 6

- **Insert 10 → fits in leaf**
- **Insert 20 → fits**
- **Insert 5 → causes overflow → split: [5], [10, 20] → promote 10**
- **Tree structure updates with 10 in root and two child leaves**

This maintains balance, sorted order, and efficient access.

5)b) Write about ISAM.

ISAM (Indexed Sequential Access Method) is a database storage technique optimized for fast reads via indexed and sequential access.

Structure:

- **Data**: Sorted sequentially by key in fixed blocks.
- **Indexes**: Primary index (maps keys to block addresses); optional secondary indexes.

Pros:

- Fast read access for static/semi-static data (e.g., archives, reports).
- Simple, predictable performance.

Cons:

- Slow writes due to overflow management.
- Fragmentation over time, needing maintenance.

Use Cases: Legacy systems, read-heavy environments with minimal data changes.

Modern Alternatives:

- **B+ Trees**: Auto-balancing, better for dynamic data.
- **VSAM**: Enhanced IBM version with improved space management.

7) Describe about 1st Normal form?

First Normal Form (1NF):

1NF is the basic level of normalization in relational databases. A relation is in 1NF if:

Rules of 1NF:

1. **Atomic Values Only**
   - **Each cell must contain only a single (indivisible) value.**
   - **No sets, arrays, or lists.**
2. **Unique Column Names**
   - **Each column must have a unique name.**
3. **No Repeating Groups**
   - **There should be no multiple columns for the same type of data.**

Example (Before 1NF):

StudentID	**Name**	**Courses**

1	**Raju**	**DBMS, OS**
2	**Meena**	**DBMS**

Here, "Courses" has multiple values → violates 1NF.

Converted to 1NF:

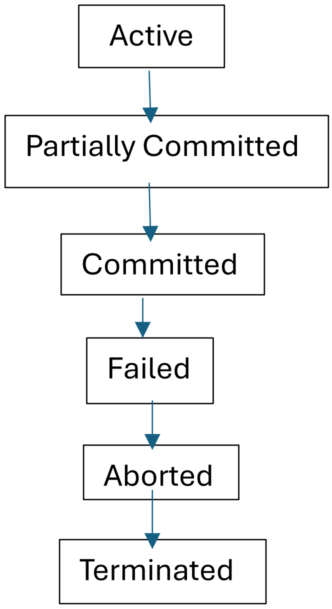
StudentID	**Name**	**Course**
1	**Raju**	**DBMS**
1	**Raju**	**OS**
2	**Meena**	**DBMS**

Conclusion:

1NF removes multi-valued attributes and ensures each field contains atomic data, making the table easier to manage.

Set – 3

2)a) Draw transaction state diagram and describe each state that a transaction goes through during its execution.



Transaction States Explained:

1. **Active**
    - **The transaction is currently executing its operations (read/write).**
2. **Partially Committed**
    - **All operations are done, and the transaction is ready to save changes.**
3. **Committed**
    - **Changes made by the transaction are permanently saved in the database.**
4. **Failed**
    - **Some error occurred, so the transaction cannot proceed.**
5. **Aborted**
    - **The transaction is rolled back, undoing all changes.**
6. **Terminated**
    - **The transaction has either committed or aborted, and its execution is complete.**

2)b) Explain ACID properties.

1. Atomicity

- Ensures all operations in a transaction are completed or none.
- If any operation fails, changes are rolled back.
- Maintains database in a consistent state.

---

2. Consistency

- Preserves database rules (e.g., constraints).
- Transforms data from one valid state to another.
- No violation of integrity occurs after the transaction.

---

3. Isolation

- Transactions execute independently.
- Intermediate results are hidden from other transactions.
- Prevents data conflicts in concurrent execution.

---

4. Durability

- Committed changes are saved permanently.
- Survives system crashes or power failures.
- Ensures data is stored in non-volatile memory.

4)a) Explain in detail about Data on External Storage in DBMS.

Data on External Storage in DBMS (Short Notes):

External storage refers to storing data on non-volatile devices like hard disks or SSDs, used when data is too large for main memory.

Key Concepts:

- **Block: Unit of data transfer between disk and RAM.**
- **File: Contains multiple blocks, holds tables, indexes, etc.**
- **Record/Tuple: A row in a table, stored inside blocks.**

Types of File Organization:

1. **Heap: No order; fast insertions.**
2. **Sequential: Sorted by key; good for range queries.**
3. **Hashed: Uses hash function; fast for exact lookups.**
4. **Indexed: Uses indexes (e.g., B+ Trees) for fast search.**

Uses:

- **Stores large data permanently.**
- **Supports efficient retrieval and updates.**
- **Essential for big databases.**

4)b) Write about ISAM.

ISAM (Indexed Sequential Access Method) is a database storage technique optimized for fast reads via indexed and sequential access.

Structure:

- **Data**: Sorted sequentially by key in fixed blocks.
- **Indexes**: Primary index (maps keys to block addresses); optional secondary indexes.

Pros:

- Fast read access for static/semi-static data (e.g., archives, reports).
- Simple, predictable performance.

Cons:

- Slow writes due to overflow management.
- Fragmentation over time, needing maintenance.

Use Cases: Legacy systems, read-heavy environments with minimal data changes.

Modern Alternatives:

- **B+ Trees**: Auto-balancing, better for dynamic data.
- **VSAM**: Enhanced IBM version with improved space management.

7)Write about Decomposition & its problems in DBMS.

Problems Related to Decomposition in DBMS:

Decomposition is the process of breaking a relation into two or more sub-relations. While it helps remove redundancy and anomalies, it may also lead to the following problems:

---

1. **Loss of Information (Lossy Decomposition)**
   - Some original data may be **lost** if the decomposition is not done properly.
   - The original relation **cannot be recovered** through natural joins.
2. **Dependency Preservation Problem**
   - All **functional dependencies** from the original relation **may not be preserved** in the decomposed relations.
   - Makes **constraint enforcement** difficult.
3. **Join Dependency Issues**
   - Rejoining decomposed relations may produce **extra or missing tuples**.
   - Leads to **incorrect query results**.
4. **Increased Query Complexity**
   - Queries may need to **join multiple tables**, increasing **execution time** and **processing cost**.