Sub:	Database Management Systems						Sub Code:		MMC103
Date:	18-02-2026	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

Note: Answer FIVE FULL Questions, choosing ONE full question from each Module

PART I

1 **Explain functional dependency and lossless decomposition. Illustrate with a suitable example.**

Functional Dependency (FD)

Definition:

A functional dependency $X \rightarrow Y$ exists in a relation R if for any two tuples t_1 and t_2 in R, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

It means the value of attribute(s) X uniquely determines the value of attribute(s) Y.

Lossless Decomposition

Definition:

A decomposition of relation R into R_1 and R_2 is said to be lossless if the natural join of R_1 and R_2 gives exactly the original relation R (no spurious tuples).

Condition for Lossless Join:

For decomposition $R \rightarrow R_1$ and R_2,

It is lossless if:

$R_1 \cap R_2 \rightarrow R_1$

OR

$R_1 \cap R_2 \rightarrow R_2$

(i.e., common attribute should be a super key in at least one relation.)

Example:

Given:

R(A, B, C)

FD: $A \rightarrow B$

Decompose into:

R_1(A, B)

R_2(A, C)

Here,

Common attribute = A

Since $A \rightarrow B$,

A is key for R_1

Therefore, decomposition is lossless.

Example:

Consider relation:

STUDENT(SID, Name, Dept, HOD)

Functional Dependencies:

- SID → Name, Dept
- Dept → HOD

Explanation:

- If SID is same, Name and Dept must be same.
- If Dept is same, HOD must be same.

2 **Describe Multivalued Dependency (MVD) and explain Fourth Normal Form (4NF) with an example.**

Multivalued Dependency (MVD)

Definition:

A multivalued dependency X →→ Y exists in a relation R if, for each value of X, there are multiple independent values of Y.

It means that the values of Y are independent of other attributes in the relation but depend only on X.

If two tuples t_1 and t_2 have the same X value, then we can swap their Y values and still get valid tuples in the relation.

Example:

Consider relation:

STUDENT(SID, Skill, Hobby)

Suppose:

- A student can have multiple skills.
- A student can have multiple hobbies.
- Skills and hobbies are independent of each other.

Then:

SID →→ Skill

SID →→ Hobby

Example data:

SID	Skill	Hobby
1	Java	Music
1	Python	Music
1	Java	Sports
1	Python	Sports

Here, skills and hobbies combine independently, creating redundancy.

Fourth Normal Form (4NF)

Definition:

A relation R is in Fourth Normal Form (4NF) if:

For every non-trivial multivalued dependency X →→ Y in R,

X must be a super key.

Non-trivial MVD:

X →→ Y is non-trivial if:

- Y is not a subset of X
- X ∪ Y ≠ R

Decomposition into 4NF

Given:

R(SID, Skill, Hobby)

MVDs:

SID →→ Skill

SID →→ Hobby

Since SID is not the key of entire relation (SID, Skill, Hobby),

the relation violates 4NF.

Decompose into:

R_1(SID, Skill)
R_2(SID, Hobby)

Now:

- No multivalued dependency violation.
- Both relations are in 4NF.

PART II

3 **Normalize a relation up to Third Normal Form (3NF) and justify each step.**

EMP(EmpID, EmpName, DeptID, DeptName, Location)

Step 1: Identify Functional Dependencies
Assume:

1. EmpID → EmpName, DeptID
2. DeptID → DeptName, Location

Step 2: First Normal Form (1NF)
Rule:
A relation is in 1NF if:

- All attributes contain atomic values.
- No repeating groups.

In the given relation, all attributes are atomic.
Therefore, the relation is already in 1NF.
✓ Justification: No multi-valued or composite attributes.
Step 3: Second Normal Form (2NF)
Rule:
A relation is in 2NF if:

- It is in 1NF, and
- No partial dependency exists.

Primary Key: EmpID
(Since EmpID uniquely identifies each employee)
There is no composite key.
Hence, no partial dependency.
✓ Justification: Relation is already in 2NF.
Step 4: Third Normal Form (3NF)
Rule:
A relation is in 3NF if:

- It is in 2NF, and
- No transitive dependency exists.

From FDs:
EmpID → DeptID
DeptID → DeptName, Location
Here:
EmpID → DeptID → DeptName, Location
This is a transitive dependency.
Step 5: Decomposition to Remove Transitive Dependency
Split into two relations:

1. EMP(EmpID, EmpName, DeptID)
2. DEPT(DeptID, DeptName, Location)

4 **Explain the Transaction State Diagram and discuss how Atomicity and Durability are implemented.**

A transaction passes through various states during its execution.

Transaction States:

1. Active

- Transaction is executing read/write operations.

2. Partially Committed
   - Final statement has executed.
   - Changes are not yet permanently saved.
3. Committed
   - All operations completed successfully.
   - Changes are permanently recorded in the database.
4. Failed
   - Transaction cannot proceed due to error, crash, or constraint violation.
5. Aborted
   - Database is restored to previous consistent state.
   - Transaction may restart or terminate.
6. Terminated
   - Transaction leaves the system after commit or abort.

Atomicity:

Atomicity ensures that a transaction is treated as a single unit — either all operations are executed or none."If a transaction fails, the database must be restored to its previous consistent state."

Durability

Durability ensures that once a transaction commits, its changes are permanently stored — even if system crashes later.

PART III

5 **Explain Lock-Based Protocols and show how Two-Phase Locking (2PL) ensures isolation.**

Lock-based protocols are concurrency control techniques in which a transaction must acquire a lock on a data item before accessing it.

They prevent conflicting operations from executing simultaneously.

Types of Locks

1. Shared Lock (S-lock)

- Used for read operations.
- Multiple transactions can hold shared locks on the same data item.
- Does not allow write operation.

2. Exclusive Lock (X-lock)

- Used for write operations.
- Only one transaction can hold exclusive lock on a data item.
- Prevents both read and write by other transactions.

Locking Rules

Before:

- Read(X) → Acquire Shared Lock on X.
- Write(X) → Acquire Exclusive Lock on X.

If lock is not available, transaction waits.

Two-Phase Locking (2PL)

Two-Phase Locking protocol ensures conflict serializability.

A transaction must follow two phases:

Phase 1: Growing Phase

- Transaction can acquire locks.
- Cannot release any lock.

Phase 2: Shrinking Phase

- Transaction can release locks.
- Cannot acquire new locks.

How 2PL Ensures Isolation

Isolation means:

Concurrent transactions should not interfere with each other.

Under 2PL:

- Transactions lock required data items before use.
- Conflicting operations are prevented.
- Schedules become conflict-serializable.
- Equivalent to some serial execution.

Thus, transactions appear to execute one after another logically.
Example
Transaction T1:
Lock-X(A)
Write(A)
Unlock(A)
Transaction T2:
Lock-S(A)
Read(A)
T2 cannot read until T1 releases X-lock.
Therefore:
No dirty read occurs.
Isolation is maintained.
Variants of 2PL (Optional Enrichment)

- Basic 2PL
- Strict 2PL (holds X-lock until commit)
- Rigorous 2PL (holds all locks until commit)

Strict 2PL prevents cascading rollback.

6 **Describe Timestamp-Based Protocols and compare them with lock-based protocols**

Timestamp-based protocol is a concurrency control technique that ensures serializability by assigning a unique timestamp to each transaction.
The timestamp determines the order in which transactions should be executed.
How It Works

- When a transaction Ti starts, it is assigned a unique timestamp TS(Ti).
- If TS(T1) < TS(T2), then T1 is older than T2.
- The system ensures that conflicting operations occur in timestamp order.

Each data item X maintains:

- R_TS(X) → Largest timestamp of any transaction that successfully read X.
- W_TS(X) → Largest timestamp of any transaction that successfully wrote X.

Read Rule
When transaction Ti wants to read X:

- If TS(Ti) < W_TS(X) → Rollback Ti.
- Otherwise:
  - Allow read.
  - Update R_TS(X).

Write Rule
When transaction Ti wants to write X:

- If TS(Ti) < R_TS(X) → Rollback Ti.
- If TS(Ti) < W_TS(X) → Rollback Ti.
- Otherwise:
  - Allow write.
  - Update W_TS(X).

Advantages of Timestamp Protocol

- No deadlocks.
- No waiting for locks.
- Ensures conflict serializability.

Disadvantages

- More rollbacks may occur.
- Can lead to starvation of younger transactions.

Feature	Timestamp Protocol	Lock-Based Protocol
Mechanism	Uses timestamps	Uses locks (S, X)
Waiting	No waiting	Transactions may wait
Deadlock	No deadlock	Deadlock possible
Rollback	More frequent	Less frequent
Blocking	Non-blocking	Blocking possible
Serializability	Ensured by order	Ensured by 2PL

PART IV

7 **Explain Multiple Granularity Locking with lock modes and compatibility matrix.**

Multiple Granularity Locking is a concurrency control technique that allows transactions to lock data at different levels of granularity in a hierarchy.

Hierarchy example:

Database

→ Table

→ Page

→ Record (Tuple)

It improves performance by allowing locking at coarse or fine levels as required.

Need for Multiple Granularity Locking

- Locking entire database reduces concurrency.
- Locking individual records increases overhead.
- MGL balances concurrency and overhead.

Lock Modes in MGL

1. **Shared (S)**
    - Read lock.
    - Multiple transactions can hold S-lock.
2. **Exclusive (X)**
    - Write lock.
    - Only one transaction can hold X-lock.
3. **Intention Shared (IS)**
    - Intends to acquire S-lock on lower-level nodes.
4. **Intention Exclusive (IX)**
    - Intends to acquire X-lock on lower-level nodes.
5. **Shared Intention Exclusive (SIX)**
    - Holds S-lock on current node.
    - Intends to acquire X-lock on lower nodes.

Locking Rules in MGL

- Locks must be acquired from root to leaf.
- Locks must be released from leaf to root.
- A node can be locked only if parent is locked with appropriate intention lock.

Example

If Transaction T1 wants to read a record:

- Acquire IS lock on Database
- Acquire IS lock on Table
- Acquire S lock on Record

If Transaction T2 wants to update a record:

- Acquire IX lock on Database
- Acquire IX lock on Table
- Acquire X lock on Record

Compatibility ensures proper concurrency control.

Advantages

- Higher concurrency.
- Efficient locking.
- Prevents unnecessary blocking.

8 **Explain recovery from system crash using logs with a suitable example.**

A system crash occurs due to:

- Power failure
- Software crash
- Hardware failure

In such cases:

- Main memory contents are lost.
- Disk data remains intact.
- Some transactions may be partially completed.

To restore database consistency, the DBMS uses log-based recovery.

Log-Based Recovery

The system maintains a log file on stable storage.

Each log record has the format:

<Ti, X, old value, new value>
Where:

- Ti → Transaction ID
- X → Data item
- Old value → Value before update
- New value → Value after update

Write-Ahead Logging (WAL) Rule
Before updating the database:

- The log record must be written to stable storage.

This ensures recovery is possible after crash.
Recovery Procedure After Crash
After restart:

1. Identify committed transactions.
2. REDO all committed transactions.
3. UNDO all uncommitted transactions.

Example
Initial Value:
A = 1000
Log records:
<Start T1> <T1, A, 1000, 1500> <Commit T1> <Start T2> <T2, A, 1500, 2000>
System crashes before T2 commits.
Recovery Steps:

- T1 is committed → REDO update → A = 1500
- T2 is not committed → UNDO update → Restore A = 1500

Final value after recovery:
A = 1500
Why UNDO and REDO Are Needed

- UNDO ensures Atomicity.
- REDO ensures Durability.

Together they maintain database consistency.

PART V

9 **Discuss Recovery with Concurrent Transactions and the role of UNDO and REDO.**

Recovery with Concurrent Transactions
In a multi-user database system, several transactions execute simultaneously.
If a system crash occurs, some transactions may:

- Have committed successfully.
- Be partially completed.
- Not have started writing to disk.

Recovery must ensure:

- Committed transactions are preserved.
- Incomplete transactions are rolled back.
- Database consistency is restored.

This is achieved using log-based recovery mechanisms.
Log Structure
Each log record is stored in stable storage:
<Ti, X, old value, new value>
Also includes:
<Start Ti>
<Commit Ti>
The log helps determine which transactions need UNDO or REDO.
Role of REDO
REDO is applied to:
✓ Transactions that have committed before crash.
Purpose:

- Ensure durability.
- Reapply updates if they were not written to disk.

Example:
<Start T1> <T1, A, 1000, 1500> <Commit T1>
If crash occurs after commit but before disk write:

REDO sets A = 1500.
Role of UNDO
UNDO is applied to:
✓ Transactions that were active but not committed at crash time.
Purpose:

- Maintain atomicity.
- Remove partial effects.

Example:
<Start T2> <T2, B, 2000, 2500>
Crash occurs before commit.
UNDO restores:
B = 2000
Example with Concurrent Transactions
Initial:
A = 1000
B = 2000
Log:
<Start T1> <T1, A, 1000, 1500> <Start T2> <T2, B, 2000, 2500> <Commit T1>
Crash occurs.
Recovery:

- T1 committed → REDO (A = 1500)
- T2 not committed → UNDO (B = 2000)

Final consistent state:
A = 1500
B = 2000

10 **Describe Failure with Loss of Non-Volatile Storage and explain backup and restore mechanisms.**

Failure with Loss of Non-Volatile Storage
Non-volatile storage refers to permanent storage such as:

- Hard disks
- SSDs
- Magnetic tapes

A failure with loss of non-volatile storage occurs when the physical storage device itself is damaged and database files are lost.
Causes of Such Failure

- Disk crash
- Hardware malfunction
- Fire or flood
- Accidental deletion
- Virus or ransomware attack

In this case:

- Both database and log files may be destroyed.
- Normal log-based recovery alone is not sufficient.

Backup Mechanism
Backup is a copy of the database stored separately to recover from catastrophic failure.
Types of Backups

1. Full Backup
   - Complete copy of the entire database.
   - Taken periodically.
2. Incremental Backup
   - Copies only data changed since last backup.
3. Differential Backup
   - Copies changes made since last full backup.

Backups are stored on:

- External storage
- Remote servers
- Cloud systems

Restore Mechanism
Restore is the process of recovering the database after failure.
Steps in Restore Process

1. Replace damaged disk/storage.
2. Restore latest full backup.
3. Apply incremental or archived log files.

4. Perform REDO of committed transactions.
5. Perform UNDO of incomplete transactions.

Example

Suppose:

Full back up taken at 10 AM.

Transactions T1 and T2 committed after 10 AM.

T3 was active when disk crashed.

Recovery:

- Restore 10 AM backup.
- Apply log records for T1 and T2 (REDO).
- Ignore or UNDO T3.

Database returns to consistent state.

Importance

- Protects against catastrophic failure.
- Ensures business continuity.
- Maintains durability property of transactions.