

DBMS QB – UNIT – 4

[4-marks]

1. What is a transaction in DBMS? Give one example.

- A transaction in DBMS is a sequence of one or more database operations such as read, write, update, or delete that are executed as a single logical unit of work. All operations inside a transaction are treated together, meaning they must all succeed or all fail.
- A transaction ensures data integrity and consistency by making sure that the database moves from one consistent state to another only after successful completion. If any operation fails in between, the entire transaction is rolled back.
- During the execution of a transaction, the database may temporarily become inconsistent. The database becomes consistent again only after the transaction is either committed or rolled back.

Example:

- In an online banking system, when money is transferred from one account to another, multiple operations occur. These include reading the sender's balance, deducting the amount, and adding the same amount to the receiver's account, all of which together form a single transaction.

2. Define ACID properties of a transaction.

- Atomicity means that a transaction is treated as an all-or-nothing operation. If any part of the transaction fails, the entire transaction is rolled back and no changes are saved in the database.
- Consistency ensures that a transaction brings the database from one valid state to another valid state. It makes sure that database rules, constraints, and integrity conditions are not violated.
- Isolation means that each transaction is executed independently of other transactions. The intermediate results of one transaction should not be visible to other transactions running at the same time.
- Durability guarantees that once a transaction is committed, its changes remain permanent. Even if there is a system crash or power failure, the committed data is not lost.

3. List the different states of a transaction.

- Active State is the initial state of a transaction where it is executing its operations such as read or write. The transaction remains active until all operations are performed or an error occurs.
- Partially Committed State occurs after the transaction has executed its last operation. At this point, the changes are not yet permanently saved in the database.
- Committed State is reached when the transaction successfully completes and the commit operation is executed. Once committed, all changes made by the transaction become permanent.
- Failed State happens when an error occurs during execution or system checks fail. In this state, the transaction cannot continue further.
- Aborted State occurs after a transaction is rolled back. The database is restored to the state it was in before the transaction began.

4. What is Concurrency Control?

- Concurrency Control is a mechanism in DBMS that allows multiple transactions to execute simultaneously while maintaining database correctness. It ensures that transactions do not interfere with each other in a harmful way.
- The main goal of concurrency control is to maintain the ACID properties, especially isolation and consistency, when multiple users access the database at the same time.
- Without concurrency control, problems such as lost updates, dirty reads, and inconsistent data may occur. These issues can lead to incorrect results and data corruption.
- Concurrency control uses techniques such as locking mechanisms and timestamp ordering to manage simultaneous access. This improves system efficiency while keeping data reliable and accurate.

5. Name any three problems that occur due to concurrent execution of transactions.

- Dirty Read (Temporary Update Problem) occurs when a transaction reads data that has been modified by another transaction but not yet committed. If the other transaction later rolls back, the first transaction has used incorrect data.
- Lost Update Problem happens when two transactions update the same data item at the same time. The update made by one transaction is overwritten by another, causing one update to be lost.
- Unrepeatable Read Problem occurs when a transaction reads the same data item more than once and gets different values each time. This happens because another transaction modifies the data between the two read operations.

These problems highlight the importance of proper concurrency control in DBMS.

6. What is a Serial Schedule?

- A **Serial Schedule** is a type of transaction schedule in which transactions are executed **one after another**, without any overlapping of operations. This means one transaction must complete fully before the next transaction starts.
- In a serial schedule, there is **no interleaving** of read or write operations between transactions. Because of this, conflicts between transactions do not occur.
- Serial schedules are considered **safe and consistent** since they naturally preserve database correctness. However, they may reduce system performance because other transactions must wait until the current one finishes.
- Although serial schedules guarantee correctness, they are **not efficient for multi-user systems**. Therefore, databases prefer concurrent execution with rules that behave like serial execution.

7. Define Serializability.

- **Serializability** is a concept used to ensure correctness in concurrent transaction execution. It checks whether a non-serial (interleaved) schedule produces the same result as some serial schedule.
- A schedule is serializable if its **final database state is the same** as that of a serial execution of the same transactions. This ensures that concurrency does not affect correctness.
- Serializability allows transactions to run concurrently while still maintaining **data consistency and integrity**. It helps improve system performance without sacrificing correctness.
- By enforcing serializability, DBMS avoids problems like lost updates and dirty reads. Thus, it is a key goal of concurrency control mechanisms.

8. What is a Conflict Serializable Schedule?

- A **Conflict Serializable Schedule** is a schedule that can be converted into a serial schedule by **swapping non-conflicting operations**. These swaps do not change the final result of the schedule.
- Two operations are said to be **conflicting** if they belong to different transactions, operate on the same data item, and at least one of them is a write operation. Only non-conflicting operations can be swapped.
- Conflict serializability is checked using a **precedence graph**. If the graph contains no cycle, the schedule is conflict serializable.
- Conflict serializability is a **stricter condition** than view serializability. Every conflict

9. What is a Deadlock in DBMS?

- A **deadlock** in DBMS occurs when two or more transactions are waiting for each other indefinitely. Each transaction holds a resource that the other transaction needs.
- Deadlock results in a **circular wait condition**, where no transaction can proceed. As a result, the system becomes stuck and transactions cannot complete.
- Deadlocks usually occur in multi-user environments where **locking mechanisms** are used. Improper ordering of resource requests increases the chance of deadlocks.
- To handle deadlocks, DBMS uses methods like **deadlock avoidance, detection, and prevention**. These techniques help ensure smooth transaction execution.

10. Define Lock and explain Shared Lock.

- A **lock** in DBMS is a mechanism used to control access to data items when multiple transactions are executing concurrently. It ensures that only permitted operations are performed on data at a time.
- Locks help maintain **data integrity and serializability** by preventing conflicting operations. A transaction must acquire a lock before reading or writing data.
- A **Shared Lock (S Lock)** is also known as a **read lock**. It allows a transaction to read a data item but does not allow it to modify the data.
- Multiple transactions can hold a shared lock on the same data item at the same time. However, no transaction can acquire a write lock on that item until all shared locks are released.

11. What is Timestamp Ordering in concurrency control?

- **Timestamp Ordering** is a concurrency control technique used in DBMS to ensure that transactions are executed in a correct serial order. It does not use locks and instead relies on timestamps to manage transaction execution.
- In this method, **each transaction is assigned a unique timestamp** when it enters the system. The timestamp represents the transaction's age, and older transactions are given higher priority.
- The DBMS ensures that all conflicting read and write operations follow the **timestamp order**. If a transaction violates this order, it is aborted or delayed.
- Timestamp ordering helps in maintaining **serializability and consistency** of the database. It also avoids deadlocks because no locking mechanism is involved.

12. Define Cascading Rollback.

- **Cascading Rollback** occurs when the failure of one transaction causes other dependent transactions to also roll back. This happens because those transactions have read uncommitted data.
- When a transaction reads data written by another transaction that has not yet committed, it becomes dependent on that transaction. If the original transaction aborts, all dependent transactions must also abort.
- Cascading rollback creates a **chain reaction of aborts**, similar to falling dominoes. This can reduce system performance and increase recovery time.
- To avoid cascading rollback, DBMS uses **cascadeless or strict schedules**, where transactions are not allowed to read uncommitted data.

13. What is a Two-Phase Locking (2PL) protocol?

- **Two-Phase Locking (2PL)** is a lock-based concurrency control protocol that ensures serializability of transactions. It divides the locking process into two distinct phases.
- The first phase is the **Growing Phase**, during which a transaction can acquire new locks but cannot release any lock. This phase allows the transaction to gather all required locks.
- The second phase is the **Shrinking Phase**, where a transaction can release locks but cannot acquire any new locks. This ensures that no new conflicts are introduced.
- 2PL guarantees **conflict serializability**, but it may lead to deadlocks. To overcome this, strict 2PL is often used in practical systems.

14. What is a Read-Write conflict? Give an example.

- A **Read-Write conflict** occurs when two transactions access the same data item and one transaction performs a read operation while the other performs a write operation. These operations belong to different transactions.
- This conflict happens because the read operation may get an incorrect or inconsistent value if the write operation is not properly controlled. It can affect database consistency.
- Read-write conflicts are important in determining **conflict serializability**. Such conflicts influence the order in which transactions should be executed.
- **Example:**
Transaction T1 reads data item A, and transaction T2 writes data item A. Since both access the same item and one operation is a write, a read-write conflict occurs.

[5- marks]

1. Explain the ACID properties with suitable examples.

- **Atomicity** means that a transaction is treated as a single unit of work. Either all the operations of a transaction are completed successfully or none of them are applied to the database.
Example: In a money transfer, if the debit from one account succeeds but the credit to another fails, the entire transaction is rolled back.
- **Consistency** ensures that the database remains in a valid state before and after the transaction. All database rules and constraints must be maintained during execution.
Example: If an account has ₹1000 and ₹200 is withdrawn, the final balance must be ₹800.
- **Isolation** means that multiple transactions execute independently without affecting each other's intermediate results. One transaction should not see partial results of another transaction.
Example: Two users withdrawing money from the same account should not interfere with each other.
- **Durability** guarantees that once a transaction is committed, its changes are permanent. Even system crashes or power failures will not remove committed data.

2. Describe the different states of a transaction with a neat diagram (explanation form).

- **Active State** is the initial state where the transaction starts execution. In this state, read and write operations are being performed on the database.
- **Partially Committed State** occurs after the transaction has executed its last operation. At this point, changes are made but not yet permanently stored.
- **Committed State** is reached when the commit operation is successfully completed. All changes become permanent and visible to other transactions.
- **Failed State** happens when an error occurs or system checks fail. The transaction can no longer continue execution.
- **Aborted State** occurs when a failed transaction is rolled back. The database is restored to the state it was in before the transaction began.

Diagram explanation:

Active → Partially Committed → Committed

Active/Partially Committed → Failed → Aborted

3. Explain concurrency problems: Lost Update, Dirty Read, Unrepeatable Read.

- **Lost Update Problem** occurs when two transactions update the same data item concurrently. The update made by one transaction is overwritten by another transaction, causing loss of data.
- **Dirty Read Problem** happens when a transaction reads data written by another transaction that has not yet committed. If the writer transaction aborts later, the read data becomes invalid.
- **Unrepeatable Read Problem** occurs when a transaction reads the same data item more than once and gets different values each time. This happens because another transaction modifies the data between reads.

These problems arise due to lack of proper concurrency control. DBMS uses locking and scheduling techniques to avoid such issues.

4. Explain Serial and Non-Serial schedules with examples.

- A **Serial Schedule** is one in which transactions execute one after another without interleaving. One transaction must complete before another transaction begins.
Example: Transaction T1 completes all operations before Transaction T2 starts.
- Serial schedules are easy to manage and always maintain consistency. However, they reduce system efficiency due to waiting time.
- A **Non-Serial Schedule** allows operations of multiple transactions to interleave. Transactions are executed concurrently to improve performance.
- Non-serial schedules increase system throughput but may cause conflicts. Therefore, concurrency control techniques are required to maintain correctness.
Example: T1 reads A, then T2 writes A, followed by T1 writing A.

5. What is Conflict Serializability? Explain with an example.

- **Conflict Serializability** ensures that a non-serial schedule produces the same result as a serial schedule. This is done by rearranging non-conflicting operations.
- Two operations conflict if they belong to different transactions, operate on the same data item, and at least one of them is a write operation.
- If a schedule can be converted into a serial schedule by swapping non-conflicting operations, it is conflict serializable.
- Conflict serializability is checked using a **precedence graph**. If the graph has no cycle, the schedule is conflict serializable.

Example:

If T1 writes A and T2 reads A, the order of these operations matters and creates a conflict. If no cycle exists, the schedule is conflict serializable.

6. Explain Deadlock and mention its necessary conditions.

- A **deadlock** in DBMS occurs when two or more transactions are waiting for each other indefinitely to release resources. Each transaction holds a resource that the other transaction needs, so none of them can proceed.
- Deadlock usually happens in a multi-user environment where **locking mechanisms** are used for concurrency control. Due to improper order of resource requests, transactions block each other permanently.
- Deadlocks reduce system performance because transactions remain stuck and cannot complete. The DBMS must detect or prevent deadlocks to keep the system running smoothly.

Necessary conditions for deadlock:

- **Mutual Exclusion:** Only one transaction can hold a resource at a time. Other transactions must wait until the resource is released.
- **Hold and Wait:** A transaction holding one resource can request additional resources that are held by other transactions.
- **No Preemption:** Resources cannot be forcibly taken from a transaction. They must be released voluntarily.
- **Circular Wait:** A circular chain of transactions exists, where each transaction waits for a resource held by the next one.

All four conditions must occur together for a deadlock to happen.

7. Describe Two-Phase Locking (2PL) protocol and its types.

- **Two-Phase Locking (2PL)** is a lock-based concurrency control protocol that ensures serializability of transactions. It divides the locking process into two distinct phases.
- In the **Growing Phase**, a transaction can acquire new locks but cannot release any lock. This phase allows the transaction to collect all the locks it needs.
- In the **Shrinking Phase**, a transaction can release locks but cannot acquire any new locks. Once a lock is released, the transaction moves permanently into the shrinking phase.

Types of 2PL:

- **Basic Two-Phase Locking:** Follows the growing and shrinking phases strictly. It ensures conflict serializability but may lead to deadlocks.
- **Strict Two-Phase Locking (Strict 2PL):** In this type, all exclusive (write) locks are held until the transaction commits or aborts. This avoids cascading rollbacks and ensures strict schedules.

2PL is widely used because it guarantees correctness, though deadlocks must be handled separately.

8. Explain Timestamp-based Concurrency Control mechanism.

- **Timestamp-based concurrency control** is a technique used to manage concurrent transactions without using locks. Each transaction is assigned a unique timestamp when it enters the system.
- The timestamp represents the **order of execution**, where older transactions (smaller timestamps) are given higher priority over newer ones.
- Each data item maintains two timestamps: **Read Timestamp (R_TS)** and **Write Timestamp (W_TS)**. These help the system decide whether a read or write operation is allowed.
- When a transaction tries to read or write a data item, its timestamp is compared with the data item's timestamps. If the order is violated, the transaction is aborted or delayed.
- This protocol ensures **serializability and deadlock-free execution**. However, it may cause transaction starvation and higher overhead due to frequent timestamp checks.

9. Explain how locking prevents concurrency problems.

- **Locking** is a mechanism used in DBMS to control access to shared data items. A transaction must obtain a lock before reading or writing any data.
- By using locks, the DBMS ensures that **conflicting operations** do not occur at the same time. This prevents problems such as dirty reads and lost updates.
- **Shared locks** allow multiple transactions to read data simultaneously but prevent updates. **Exclusive locks** allow a transaction to read and write data but block all other accesses.
- Locking ensures **isolation** among transactions by preventing one transaction from seeing partial updates of another.
- Protocols like **Two-Phase Locking** further ensure serializability and consistency. Thus, locking plays a vital role in maintaining correctness in concurrent environments.

10. Explain View Serializability with a simple example.

- **View Serializability** is a type of serializability that checks whether a non-serial schedule is equivalent to a serial schedule based on how data is read and written.
- Two schedules are view serializable if they produce the same final database state and each read operation reads the same value in both schedules.
- View serializability considers **initial reads**, **reads from writes**, and **final writes** on data items. If all these match a serial schedule, the schedule is view serializable.
- View serializability is more general than conflict serializability. Some schedules that are not conflict serializable may still be view serializable.

Example:

If Transaction T1 writes A and Transaction T2 later overwrites A without reading it (blind write), the schedule may still be view serializable if the final value of A matches a serial order.

11. Explain how Deadlock can be prevented.

- **Deadlock prevention** is a technique used in DBMS to ensure that deadlocks never occur in the system. The DBMS analyzes transactions in advance and applies rules so that the necessary conditions for deadlock are never satisfied.
- One common method is **resource ordering**, where all transactions are forced to request resources in a fixed order. When every transaction follows the same order, circular wait conditions are avoided.
- Another approach is **Wait-Die Scheme**, which is a non-preemptive method. In this method, older transactions are allowed to wait for resources, while younger transactions are aborted and restarted if they request a resource held by an older transaction.
- **Wound-Wait Scheme** is a preemptive deadlock prevention technique. In this scheme, older transactions can force younger transactions to abort, while younger transactions must wait if the resource is held by an older one.
- By using these prevention schemes, DBMS ensures that **circular wait does not occur**, thereby preventing deadlocks completely and maintaining smooth transaction execution.

12. What is the difference between Shared and Exclusive Locks?

- A **Shared Lock (S Lock)** is also known as a read lock. It allows a transaction to read a data item but does not allow it to modify the data.
- When a transaction holds a shared lock on a data item, **other transactions can also acquire shared locks** on the same data item. However, no transaction can obtain an exclusive lock at that time.
- An **Exclusive Lock (X Lock)** allows a transaction to both read and write a data item. This lock provides complete control over the data item to the transaction holding it.
- When a transaction holds an exclusive lock, **no other transaction can acquire either a shared lock or an exclusive lock** on that data item. This ensures that no conflicting operations occur.
- Shared locks improve concurrency by allowing multiple reads, while exclusive locks ensure data consistency during updates. Both locks together help maintain isolation and correctness in concurrent transaction execution.

[10- marks]

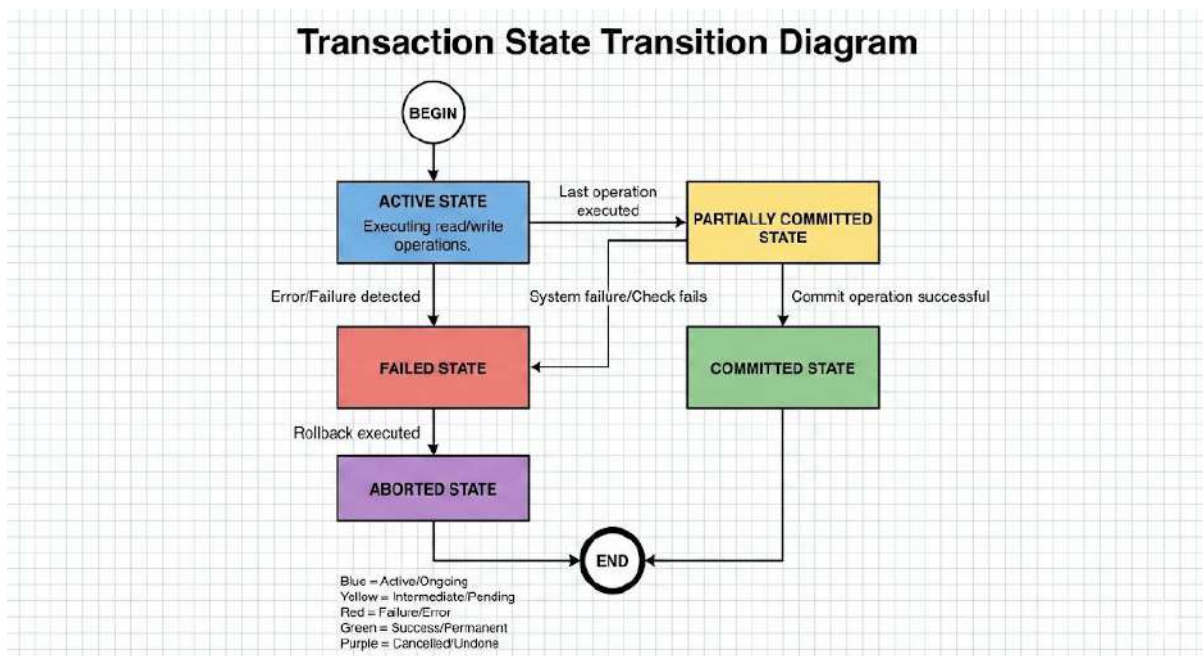
1. Discuss ACID properties in detail and explain why they are essential for reliable transaction processing.

- **ACID properties** are a set of rules that ensure every transaction in a DBMS is processed reliably. These properties protect the database from errors, failures, and inconsistent states during transaction execution.
- **Atomicity** means that a transaction is treated as a single unit of work. Either all operations of the transaction are completed successfully, or none of them are applied to the database. This is essential because partial execution can corrupt data.
Example: If money is debited from one account but not credited to another, atomicity ensures the entire transaction is rolled back.
- **Consistency** ensures that a transaction moves the database from one valid state to another valid state. All database rules and constraints must be satisfied before and after execution.
This property is essential to maintain correctness of stored data.
- **Isolation** ensures that multiple transactions executing at the same time do not interfere with each other. Intermediate results of one transaction are not visible to others.
This prevents problems like dirty reads and lost updates.
- **Durability** guarantees that once a transaction is committed, its changes are permanent. Even system crashes or power failures will not remove committed data.

Together, ACID properties ensure **accuracy, safety, and reliability** of transaction processing in real-world applications like banking systems.

2. Explain the states of a transaction with a clear diagram and describe what happens in each state.

- A transaction in DBMS passes through several **states** from its start until completion. These states help the system manage execution and recovery effectively.
- The **Active State** is the first state of a transaction. In this state, the transaction is executing read and write operations on the database.
- The **Partially Committed State** occurs after the transaction has executed its last operation. At this point, changes are made but not yet permanently stored.
- The **Committed State** is reached when the transaction successfully completes and the commit operation is performed. All changes become permanent and visible to other transactions.
- The **Failed State** occurs when an error is detected during execution or system checks fail. The transaction cannot proceed further in this state.
- The **Aborted State** occurs when a failed transaction is rolled back. All changes made by the transaction are undone, and the database is restored to its previous consistent state.



3. Explain concurrency control and discuss all concurrency-related problems with examples.

- **Concurrency control** is a mechanism in DBMS that allows multiple transactions to execute at the same time while maintaining data correctness and consistency. Its main purpose is to ensure that concurrent execution does not violate ACID properties, especially **isolation and consistency**.
- In a multi-user database system, many users may read or update the same data simultaneously. Without proper concurrency control, this can lead to incorrect results, data inconsistency, and loss of important updates.
- **Lost Update Problem** occurs when two transactions update the same data item at the same time. The update made by one transaction is overwritten by another transaction, causing loss of data.
Example: Two users withdraw money from the same account, and the final balance reflects only one withdrawal.
- **Dirty Read (Temporary Update) Problem** happens when a transaction reads data written by another transaction that has not yet committed. If the writing transaction later aborts, the read value becomes invalid.
Example: A transaction reads an updated salary value that is later rolled back.
- **Unrepeatable Read Problem** occurs when a transaction reads the same data item more than once and gets different values each time. This happens because another transaction modifies the data between the reads.
- **Phantom Read Problem** occurs when a transaction executes a query twice and finds new or missing records due to another transaction inserting or deleting rows.
- To avoid these problems, DBMS uses **locking mechanisms and timestamp ordering protocols**. These techniques ensure safe concurrent execution and reliable transaction processing.

4. Explain Serializability in detail. Distinguish between Conflict Serializability and View Serializability with examples.

- **Serializability** is a concept used to ensure correctness in concurrent transaction execution. A schedule is said to be serializable if its final result is the same as that of a serial execution of the same transactions.
- Serializability allows transactions to execute concurrently while ensuring that the database remains consistent. This improves system performance without sacrificing correctness.
- **Conflict Serializability** checks whether a non-serial schedule can be converted into a serial schedule by swapping **non-conflicting operations**.
Two operations are conflicting if they belong to different transactions, access the same data item, and at least one of them is a write operation.
Example: If Transaction T1 writes A and Transaction T2 reads A, the order of these operations matters and cannot be changed.
- Conflict serializability is usually checked using a **precedence graph**. If the graph does not contain a cycle, the schedule is conflict serializable.
- **View Serializability** checks whether a schedule is equivalent to a serial schedule based on how data values are read and written. It considers initial reads, read-from relationships, and final writes.
Example: A schedule containing **blind writes** may not be conflict serializable but can still be view serializable.
- Conflict serializability is **stricter** than view serializability. Every conflict-serializable schedule is view-serializable, but not every view-serializable schedule is conflict-serializable.

5. What is a schedule? Explain Serial, Non-Serial, Conflict Serializable, and View Serializable schedules.

Check whether the given schedule S is conflict serializable:

S = R1(A), R2(A), R1(B), R2(B), R3(B), W1(A), W2(B)

- A **schedule** is the chronological order in which operations of multiple transactions are executed in a database. It shows how read and write operations from different transactions are interleaved during execution.
- A **Serial Schedule** is a schedule in which transactions are executed one after another without interleaving. One transaction completes all its operations before the next transaction starts. Serial schedules are always correct but inefficient.
- A **Non-Serial Schedule** allows operations of multiple transactions to interleave. This improves system performance but may cause conflicts if not properly controlled.
- **Conflict Serializability** checks whether a non-serial schedule can be converted into a serial schedule by swapping non-conflicting operations.
Two operations conflict if they belong to different transactions, access the same data item, and at least one is a write.
- **View Serializability** checks correctness based on read-from relationships and final writes. It is more general than conflict serializability and allows some schedules with blind writes.

Checking Conflict Serializability of Schedule S

Conflicting operations in S:

- $R2(A) \rightarrow W1(A) \Rightarrow$ conflict between T2 and T1
- $R1(B) \rightarrow W2(B) \Rightarrow$ conflict between T1 and T2
- $R3(B) \rightarrow W2(B) \Rightarrow$ conflict between T3 and T2

Precedence Graph edges:

- $T2 \rightarrow T1$ (due to A)
- $T1 \rightarrow T2$ (due to B)
- $T3 \rightarrow T2$ (due to B)

Since there is a **cycle between T1 and T2**, the schedule is **NOT conflict serializable**.

6. Discuss Deadlock in DBMS. Explain its detection, prevention, and recovery techniques.

- A **deadlock** in DBMS occurs when two or more transactions wait indefinitely for resources held by each other. Due to this circular waiting condition, none of the transactions can proceed.
- Deadlocks usually occur in environments where **locking mechanisms** are used and multiple transactions compete for the same resources. This situation reduces system performance and blocks transaction execution.

Deadlock Detection

- Deadlock detection allows transactions to execute freely and checks periodically for deadlocks.
- The DBMS uses a **Wait-For Graph**, where nodes represent transactions and edges show waiting relationships.
- If the graph contains a **cycle**, a deadlock exists and must be resolved.

Deadlock Prevention

- Deadlock prevention ensures that deadlocks never occur by breaking one of the necessary deadlock conditions.
- **Wait-Die Scheme:** Older transactions wait, while younger ones are aborted if they request a resource held by an older transaction.
- **Wound-Wait Scheme:** Older transactions can force younger ones to abort, while younger ones must wait for older ones.
- Resource ordering is also used to avoid circular wait.

Deadlock Recovery

- Once a deadlock is detected, the DBMS must recover by aborting one or more transactions.
- The aborted transaction releases its resources, allowing other transactions to continue.
- The aborted transaction is later restarted to ensure correctness.

Deadlock handling is essential to maintain smooth and efficient database operations.

7. Explain Two-Phase Locking (2PL) protocol. Discuss its working, types (Basic, Strict, Rigorous), and advantages.

- **Two-Phase Locking (2PL)** is a lock-based concurrency control protocol used in DBMS to ensure **serializability** of transactions. It controls how locks are acquired and released during transaction execution.
- The working of 2PL is divided into **two phases**:
 - **Growing Phase:**
In this phase, a transaction can acquire new locks (shared or exclusive) but **cannot release any lock**. The transaction collects all the locks it needs in this phase.
 - **Shrinking Phase:**
In this phase, a transaction can release locks but **cannot acquire any new locks**. Once the first lock is released, the transaction enters the shrinking phase permanently.
- **Types of Two-Phase Locking:**
 - **Basic 2PL:**
Follows the growing and shrinking phases strictly. It guarantees conflict serializability but may lead to deadlocks.
 - **Strict 2PL:**
In this type, all **exclusive (write) locks are held until the transaction commits or aborts**. This prevents cascading rollbacks and ensures strict schedules.
 - **Rigorous 2PL:**
In rigorous 2PL, **both shared and exclusive locks are held until commit or abort**. This is the most restrictive form and provides very strong consistency.
- **Advantages of 2PL:**
 - Ensures **conflict serializability** of transactions.
 - Prevents problems like dirty reads and lost updates.
 - Strict and rigorous 2PL avoid cascading rollbacks.
 - Widely used in real-world database systems due to reliability.

Thus, Two-Phase Locking plays an important role in maintaining correctness and consistency in concurrent transaction execution.

8. Explain Timestamp-based Concurrency Control with examples and cases (Read–Write conflict, Write–Read conflict).

- **Timestamp-based Concurrency Control** is a method used in DBMS to manage concurrent transactions **without using locks**. Each transaction is assigned a unique timestamp when it enters the system.
- The timestamp represents the **execution order** of transactions. Older transactions (smaller timestamp) are given higher priority than newer transactions.
- Each data item maintains two timestamps:
 - **R_TS(X)**: Timestamp of the last transaction that read data item X.
 - **W_TS(X)**: Timestamp of the last transaction that wrote data item X.
- The DBMS compares transaction timestamps with these values to decide whether a read or write operation is allowed.

Read–Write Conflict (R–W Conflict)

- This occurs when a transaction tries to read a data item that was written by a newer transaction.
- If $W_TS(X) > TS(T)$, the read operation is not allowed and the transaction is aborted.
- *Example:*
T1 (older) tries to read X after T2 (newer) has written X. This violates timestamp order, so T1 is aborted.

Write–Read Conflict (W–R Conflict)

- This occurs when a transaction tries to write a data item that has already been read by a newer transaction.
- If $R_TS(X) > TS(T)$, the write operation is not allowed.
- *Example:*
T1 (older) tries to write X after T2 (newer) has already read X. This violates the timestamp order.
- **Advantages:**
 - Ensures serializability.
 - Deadlock-free since no locks are used.
- **Disadvantages:**
 - May cause frequent transaction aborts.
 - Can lead to starvation of older or newer transactions.

Timestamp-based protocols provide a structured and deadlock-free approach to concurrency control in DBMS.

9. Discuss the need for concurrency control in DBMS. Explain various concurrency control techniques.

- In a **DBMS**, multiple users often access and modify the database at the same time. **Concurrency control** is required to manage simultaneous execution of transactions so that the database remains correct and consistent.
- Without concurrency control, transactions may interfere with each other and cause problems such as dirty reads, lost updates, and inconsistent data. These issues violate the **ACID properties**, especially isolation and consistency.
- Concurrency control ensures that the final result of concurrent transaction execution is the same as if the transactions were executed serially. This improves **data reliability** while still allowing better system performance.

Concurrency Control Techniques:

- **Lock-Based Concurrency Control:**
In this technique, transactions use locks to control access to data items.
 - **Shared Lock (Read Lock)** allows multiple transactions to read data but not modify it.
 - **Exclusive Lock (Write Lock)** allows a transaction to read and write data, blocking all other access.
Protocols like **Two-Phase Locking (2PL)** ensure serializability.
- **Timestamp-Based Concurrency Control:**
Each transaction is assigned a unique timestamp. Transactions are ordered based on timestamps, and operations are allowed only if they follow this order. This method is deadlock-free.
- **Strict Protocols:**
Strict 2PL and Strict Timestamp Ordering prevent cascading rollbacks by ensuring that uncommitted data is never read.

These techniques ensure **safe concurrent execution**, high throughput, and data consistency in multi-user database systems.

10. Explain the concept of Serializability Testing using precedence graphs with examples.

- **Serializability testing** is used to check whether a concurrent (non-serial) schedule is correct. A schedule is correct if it is **serializable**, meaning it produces the same result as some serial schedule.
- One common method of testing serializability is using a **precedence graph**, also called a **serialization graph**.

Precedence Graph Concept:

- Each **node** in the graph represents a transaction.
- A **directed edge** from transaction T_i to T_j indicates that T_i must be executed before T_j .
- An edge is added when two operations conflict and the operation of T_i occurs before the operation of T_j in the schedule.

Conflicting Operations:

- Operations belong to different transactions.
- They access the same data item.
- At least one of the operations is a write.

Testing Rule:

- If the precedence graph has **no cycle**, the schedule is **conflict serializable**.
- If the graph contains a **cycle**, the schedule is **not conflict serializable**.

Example:

Schedule: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$

- Conflict: $W_1(A) \rightarrow R_2(A)$
- Edge: $T_1 \rightarrow T_2$
- Since no cycle exists, the schedule is conflict serializable.

Precedence graphs provide a **clear and systematic method** for checking correctness of concurrent schedules.

11. Describe the entire transaction processing cycle, including concurrency control, recovery, and commit/rollback.

- The **transaction processing cycle** in DBMS describes how a transaction moves from start to completion while maintaining correctness and reliability.
- A transaction begins in the **Active State**, where it performs read and write operations. During this phase, **concurrency control mechanisms** such as locking or timestamp ordering are applied to prevent conflicts with other transactions.
- While executing, the DBMS ensures isolation by controlling access to shared data. Locks or timestamps ensure that concurrent transactions do not violate consistency.
- After executing its last operation, the transaction enters the **Partially Committed State**. At this point, system checks are performed to ensure there are no errors.
- If all operations are successful, the **Commit operation** is executed. This makes all changes permanent in the database, satisfying durability.
- If an error occurs due to system failure, deadlock, or rule violation, the transaction enters the **Failed State**.
- The DBMS then performs a **Rollback**, undoing all changes made by the transaction. The transaction moves to the **Aborted State**, and the database is restored to its previous consistent state.
- **Recovery mechanisms** ensure that committed transactions remain permanent and aborted transactions leave no partial effects.

This complete cycle ensures **reliability, consistency, and safety** of transaction execution in