**1. Illustrate testing of Conflict Serializability with appropriate example.**

Conflict Serializable Schedule:

* A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
* The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

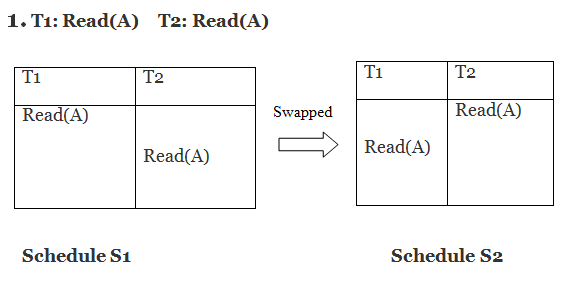
Conflicting Operations:

The two operations become conflicting if all conditions satisfy:

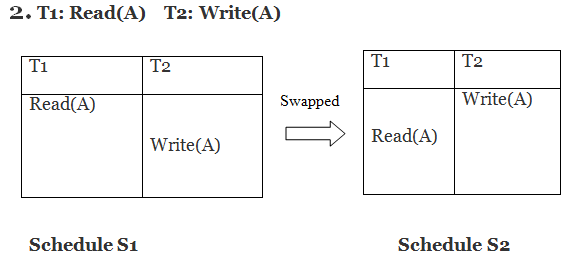
1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.



Here, S1 = S2. That means it is non-conflict.



Here, S1 ≠ S2. That means it is conflict.

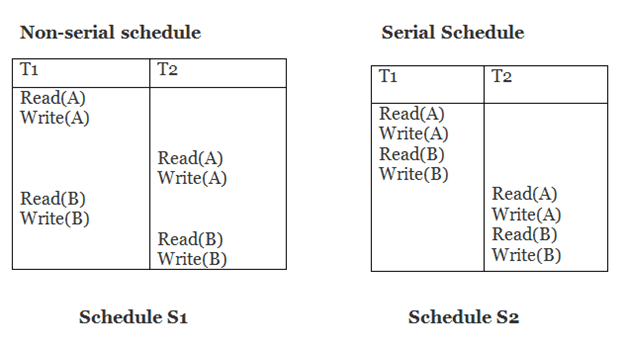
Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:



Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

**After swapping of non-conflict operations, the schedule S1 becomes:**

|  |  |
| --- | --- |
| **T1** | **T2** |
| Read(A) Write(A) Read(B) Write(B) | Read(A) Write(A) Read(B) Write(B) |

Since, S1 is conflict serializable.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

**2. Write the rules for Thomas Write Rule. Elaborate how Thomas' Write Rule allows greater potential concurrency as compared to Timestamp based protocol.**  
**Thomas Write Rule in DBMS:**

1. **Definition**:
   * Governs database changes, requiring modifications to be written to disk before user control resumes.
   * Ensures data consistency and durability, even in system failure scenarios.
2. **Importance**:
   * Integral to maintaining data integrity and reliability within database management systems.
   * Upholds principles of atomicity, consistency, isolation, and durability (ACID properties).
3. **Timestamp Ordering Protocol**:
   * States that Ri(X) precedes Wj(X) if and only if TS(Ti) < TS(Tj) for conflicting operations.
   * Non-serializable schedules are rejected, transactions rolled back.
   * Thomas Write Rule allows certain harmless operations, diverging from strict conflict serializability.
4. **Features of Thomas Write Rule**:
   * **Modification of Basic TO Protocol**:
     + Allows fewer write operation rejections.
     + Ignores outdated writes, enhancing concurrency.
   * **Operational Criteria**:
     + If R\_TS(X) > TS(T), transaction aborts and rolls back, operation rejected.
     + If W\_TS(X) > TS(T), write operation not executed, processing continues.
     + If neither condition applies, execute W\_item(X) operation and set W\_TS(X) to TS(T).
   * **Concurrency Enhancement**:
     + Achieves concurrency with View Serializable schedules, not just Conflict Serializable ones.
   * **Guarantee of Serializability Order**:
     + Improves upon Basic Timestamp Ordering Algorithm.
   * **Illustrative Example**:
     + Permits some serializable schedules that are not strictly conflict serializable.
     + Allows for more flexible scheduling without sacrificing data consistency.

**Conclusion:**

* The Thomas Write Rule stands as a cornerstone in database management, ensuring data integrity, reliability, and performance.
* It extends beyond mere transaction execution, providing a framework for concurrency, recovery, and consistency.
* By allowing certain deviations from strict serializability, it strikes a balance between efficiency and robustness in database operations.

**3. List and explain the variants of Two Phase Lock Protocol**

The Two-Phase Locking Protocol (2PL) is a fundamental concurrency control mechanism in database management systems (DBMS). It ensures serializability by requiring transactions to issue lock and unlock requests in two distinct phases:

1. **Growing Phase**:
   * Transactions can acquire locks but cannot release any lock.
   * Transactions initially enter this phase and acquire locks as needed.
2. **Shrinking Phase**:
   * Transactions can release locks but cannot obtain new locks.
   * Occurs after a transaction releases a lock, preventing further lock acquisition.

Now, let's delve into the variants of the Two-Phase Locking Protocol:

**1. Basic Two-Phase Locking Protocol:**

* **Characteristics**:
  + Guarantees serializability by enforcing lock acquisition and release phases.
  + Locks obtained by transactions are released only after the transaction commits or aborts.
* **Example**:
  + Transactions T3 and T4 follow the Two-Phase Locking Protocol.
  + T1 and T2 do not adhere to this protocol.

**2. Strict Two-Phase Locking Protocol:**

* **Enhancement**:
  + Requires exclusive-mode locks to be held until the transaction commits.
  + Prevents data written by uncommitted transactions from being read by others.
* **Purpose**:
  + Avoids cascading rollbacks and ensures consistent data state until transaction completion.

**3. Rigorous Two-Phase Locking Protocol:**

* **Characteristics**:
  + Requires all locks to be held until the transaction commits.
  + Ensures transactions can be serialized based on their commit order.
* **Advantage**:
  + Provides a strict guarantee of serializability, enhancing data consistency.

**4. Two-Phase Locking Protocol with Lock Conversion:**

* **Refinement**:
  + Allows lock conversions between shared and exclusive modes during transaction execution.
  + Enables more concurrency by allowing shared access initially, with potential later conversion to exclusive mode.
* **Mechanism**:
  + Upgrading from shared to exclusive mode occurs in the growing phase.
  + Downgrading from exclusive to shared mode occurs in the shrinking phase.

**Conclusion:**

* Various variants of the Two-Phase Locking Protocol cater to different requirements of concurrency control in DBMS.
* Each variant aims to ensure data consistency, prevent deadlocks, and optimize concurrency.
* Depending on the specific needs of the application and system, DBMS designers and administrators choose the most suitable variant to maintain an efficient and reliable database environment.

**4. Explain with appropriate example the following terms**

a. Recoverable Schedules

b. Cascadeless Schedules   
  
  
**Schedules Based on Recoverability**

* **Recoverable Schedule:**A schedule is recoverable if it allows for the recovery of the database to a consistent state after a transaction failure. In a recoverable schedule, a transaction that has updated the database must commit before any other transaction reads or writes the same data. If a transaction fails before committing, its updates must be rolled back, and any transactions that have read its uncommitted data must also be rolled back.
* **Cascadeless Schedule:**A schedule is cascaded less if it does not result in a cascading rollback of transactions after a failure. In a cascade-less schedule, a transaction that has read uncommitted data from another transaction cannot commit before that transaction commits. If a transaction fails before committing, its updates must be rolled back, but any transactions that have read its uncommitted data need not be rolled back.

**Recoverable Schedule**

A schedule is said to be recoverable if it is recoverable as the name suggests. Only reads are allowed before write operations on the same data. Only reads (Ti->Tj) are permissible.

**Example:**

S1: R1(x), **W1(x)**, R2(x), R1(y), R2(y),   
 **W2(x)**, W1(y), **C1**, **C2**;

The given schedule follows the order of **Ti->Tj => C1->C2**. Transaction T1 is executed before T2 hence there is no chance of conflict occurring. R1(x) appears before W1(x) and transaction T1 is committed before T2 i.e. completion of the first transaction performed the first update on data item x, hence given schedule is recoverable.

Let us see an example of an **unrecoverable schedule** to clear the concept more.

S2: R1(x), R2(x), R1(z), R3(x), R3(y), W1(x),   
 **W3(y)**, R2(y), W2(z), **W2(y)**, C1, **C2**, **C3**;

**Ti->Tj => C2->C3** but W3(y) executed before W2(y) which leads to conflicts thus it must be committed before the T2 transaction. So given schedule is unrecoverable. if **Ti->Tj => C3->C2** is given in the schedule then it will become a recoverable schedule.

***Note:****A committed transaction should never be rollback. It means that reading value from uncommitted transaction and commit it will enter the current transaction into inconsistent or unrecoverable state this is called Dirty Read problem.*

**Cascadeless Schedule**

When no **read** or **write-write** occurs before the execution of the transaction then the corresponding schedule is called a [cascadeless schedule.](https://www.geeksforgeeks.org/cascadeless-in-dbms/)

**Example:**

S3: R1(x), R2(z), R3(x), R1(z), R2(y), R3(y), W1(x), C1,   
 W2(z), **W3(y)**, **W2(y)**, **C3**, **C2**;

In this schedule **W3(y)** and **W2(y)** overwrite conflicts and there is no read, therefore given schedule is cascade less schedule.

***Special Case:****A committed transaction desired to abort. As given below all the transactions are reading committed data hence it’s cascadeless schedule.*

**5. What is transaction? Explain its ACID properties of transaction**   
  
**What is a transaction?**

In the context of databases and data storage systems, a **transaction** is any operation that is treated as a single unit of work, which either completes fully or does not complete at all, and leaves the storage system in a consistent state. The classic example of a transaction is what occurs when you withdraw money from your bank account. Either the money has left your bank account, or it has not — there cannot be an in-between state.

**A.C.I.D. properties: Atomicity, Consistency, Isolation, and Durability**

ACID is an acronym that refers to the set of 4 key properties that define a transaction: **Atomicity, Consistency, Isolation,** and **Durability.** If a database operation has these ACID properties, it can be called an ACID transaction, and data storage systems that apply these operations are called transactional systems. ACID transactions guarantee that each read, write, or modification of a table has the following properties:

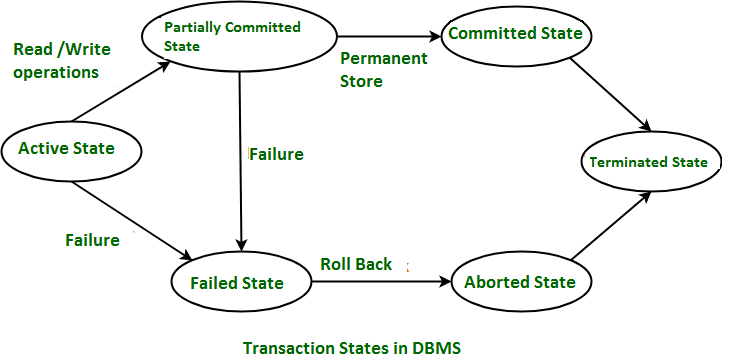
* **Atomicity** - each statement in a transaction (to read, write, update or delete data) is treated as a single unit. Either the entire statement is executed, or none of it is executed. This property prevents data loss and corruption from occurring if, for example, if your streaming data source fails mid-stream.
* **Consistency** - ensures that transactions only make changes to tables in predefined, predictable ways. Transactional consistency ensures that corruption or errors in your data do not create unintended consequences for the integrity of your table.
* **Isolation** - when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don't interfere with or affect one another. Each request can occur as though they were occurring one by one, even though they're actually occurring simultaneously.
* **Durability** - ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure.

1. **Draw and explain the Transaction State Diagram**   
     
   States through which a transaction goes during its lifetime.

These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions.

These states govern the rules which decide the fate of the transaction whether it will commit or abort.

They also use **Transaction log.** Transaction log is a file maintain by recovery management component to record all the activities of the transaction. After commit is done transaction log file is removed.



These are different types of Transaction States :

1. **Active State –**   
   When the instructions of the transaction are running then the transaction is in active state. If all the ‘read and write’ operations are performed without any error then it goes to the “partially committed state”; if any instruction fails, it goes to the “failed state”.
2. **Partially Committed –**   
   After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to “committed state” and in case of failure it will go to the “failed state”.
3. **Failed State –**   
   When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data on Data Base.
4. **Aborted State –**   
   After having any type of failure the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.
5. **Committed State –**   
   It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.
6. **Terminated State –**   
   If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.
7. **List and elaborate the Pitfalls of Lock-Based Protocols**

* Possibility of irrecoverability
* Possibility of deadlock
* Possibility of starvation.
* Possibility of Cascading rollback.

**9. List and elaborate the Intention Lock Modes in Multiple Granularity? Draw the**

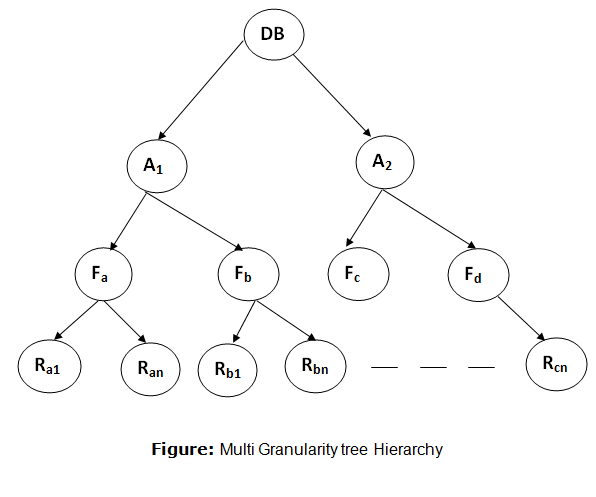
**Compatibility Matrix with Intention Lock Modes.**  
**Granularity:** It is the size of data item allowed to lock.

Multiple Granularity:

* It can be defined as hierarchically breaking up the database into blocks which can be locked.
* The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
* It maintains the track of what to lock and how to lock.
* It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

**For example:** Consider a tree which has four levels of nodes.

* The first level or higher level shows the entire database.
* The second level represents a node of type area. The higher level database consists of exactly these areas.
* The area consists of children nodes which are known as files. No file can be present in more than one area.
* Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
* Hence, the levels of the tree starting from the top level are as follows:
  1. Database
  2. Area
  3. File
  4. Record



In this example, the highest level shows the entire database. The levels below are file, record, and fields.

There are three additional lock modes with multiple granularity:

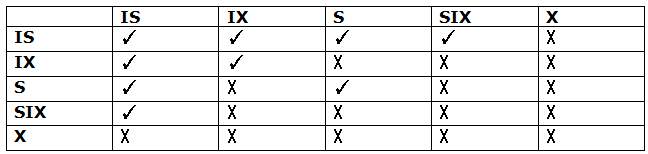
Intention Mode Lock

**Intention-shared (IS):** It contains explicit locking at a lower level of the tree but only with shared locks.

**Intention-Exclusive (IX):** It contains explicit locking at a lower level with exclusive or shared locks.

**Shared & Intention-Exclusive (SIX):** In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.

**Compatibility Matrix with Intention Lock Modes:** The below table describes the compatibility matrix for these lock modes:



It uses the intention lock modes to ensure serializability. It requires that if a transaction attempts to lock a node, then that node must follow these protocols:

* Transaction T1 should follow the lock-compatibility matrix.
* Transaction T1 firstly locks the root of the tree. It can lock it in any mode.
* If T1 currently has the parent of the node locked in either IX or IS mode, then the transaction T1 will lock a node in S or IS mode only.
* If T1 currently has the parent of the node locked in either IX or SIX modes, then the transaction T1 will lock a node in X, SIX, or IX mode only.
* If T1 has not previously unlocked any node only, then the Transaction T1 can lock a node.
* If T1 currently has none of the children of the node-locked only, then Transaction T1 will unlock a node.

Observe that in multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

* If transaction T1 reads record Ra9 in file Fa, then transaction T1 needs to lock the database, area A1 and file Fa in IX mode. Finally, it needs to lock Ra2 in S mode.
* If transaction T2 modifies record Ra9 in file Fa, then it can do so after locking the database, area A1 and file Fa in IX mode. Finally, it needs to lock the Ra9 in X mode.
* If transaction T3 reads all the records in file Fa, then transaction T3 needs to lock the database, and area A in IS mode. At last, it needs to lock Fa in S mode.
* If transaction T4 reads the entire database, then T4 needs to lock the database in S mode.