CHAPTER 2

Transaction Concepts and Concurrency Control

Contents...

Objectives...

- To understand Concepts of Transaction Processing and Concurrency Control
- To learn Schedule and their Types
- To know Concepts of Timestamping and Locking
- To learn Deadlocks and its Detection, Prevention and Recovery

2.0 INTRODUCTION

- Transaction processing and concurrency control are the important activities of any database system. A transaction is a set of database operations that performs a particular task/operation. Concurrency control is the process of managing simultaneous (concurrent) execution of transactions.
- Number of transactions can be executed at the same time and they may be accessing the same database. Such concurrent access to the database may result in some inconsistent state of database.
- Concurrency control in transaction management preserves the consistency of database in case of concurrent accesses.
- This chapter deals with the topics like basic concepts of transaction, transaction management and concurrency control.

2.1 INTRODUCTION TO TRANSACTION (April 16)

- A transaction can be considered as a unit of program execution that accesses and updates various data items of a database.
- Collection of operations that forms a single logical unit of work is called transaction.
- A transaction accesses and possibly updates various data items. After every transaction, the database should be in a consistent state.

2.1.1 Transaction Concepts (April 16, 18; Oct. 17)

- A transaction is a program unit whose execution accesses and possibly updates the contents of a database. A transaction can be defined as, "a logical unit of database processing that includes one or more database access operations".
- If the database was inconsistent state before a transaction, then on execution of transaction, the database will be in a consistent state.
- Fig. 2.1 shows concept of transaction in database. Every transaction is delimited by statements or function calls of the form begin transaction and end transaction.
- A transaction is a sequence of READ and WRITE actions that are grouped together to from a database access. Whenever we Read from and/or Write to (update) the database, a transaction is created.

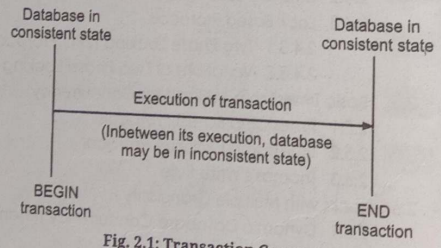


Fig. 2.1: Transaction Concept

2.1.2 Transaction Properties (April 16, 18; Oct. 17)

- To ensure accuracy, completeness and the integrity of data, database system maintains following properties of transaction.

1. **Atomicity:** Atomicity property ensures that at the end of the transaction, either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of a transaction, the updates made by the transaction will be accessible to other transactions and processes outside the transaction.
2. **Consistency:** Consistency property of transaction implies that if the database was in consistent state before the start of a transaction, then on termination of a transaction, the database will also be in a consistent state.

3. **Isolation:** Isolation property of transaction indicates that action performed by a transaction will be hidden from outside the transaction until the transaction terminates. Thus each transaction is unaware of other transactions executing concurrently in the system.
4. **Durability:** Durability property of a transaction ensures that once a transaction completes successfully (commits), the changes it has made to the database persist, even if there are system failures.

- These four properties are often called ACID (Atomicity, Consistency, Isolation, and Durability) properties of transaction.
- ACID is a set of properties that guarantee the reliability of processing of database transactions.

Significance of ACID Properties:

- Consider a banking system consisting of several accounts and a set of transactions that accesses and updates those accounts. Here consider that the database resides on disk, but some portion of database is temporarily stored in main memory.
- Following are the functions to access the database:
  1. **read (X):** Which transfers the data item X from the database to a local buffer, belonging to the transaction that executed the read operation.
  2. **write (X):** Which transfers the data item X from the local buffer of the transaction that executed the write back to the database.
- Let T_1 be a transaction that transfers ₹ 50 from account A to account B. This transaction can be defined as:

```
T1 :  read (A);
      A := A – 50;
      write (A);
      read (B);
      B:= B + 50;
      write (B).
```

- Let us now consider the significance of ACID properties.

Atomicity:

- This property ensures that either all the operations of a transaction reflect in database or none.
- For example, suppose in a banking system Account A has a balance of ₹ 400 and B has ₹ 700. Account A is transferring ₹ 100 to Account B. This is a transaction that has two operations i.e., Withdrawal of ₹ 100 from A's balance and Depositing ₹ 100 to B's balance. Let's say first operation passed successfully while second failed, in this case A's balance would be ₹ 300 while B would be having ₹ 700 instead of ₹ 800. This is unacceptable in a banking system. Either the transaction should fail without executing any of the operation or it should process both the operations. Thus, the Atomicity property ensures that a transaction is said to be atomic if a transaction always executes all its actions in one step or not executes any actions at all.

Consistency:

- To preserve the consistency of database, the execution of transaction should take place in isolation that means no other transaction should run concurrently when there is a transaction already running.
- For example, account A is having a balance of ₹ 400 and it is transferring ₹ 100 to account B and C both. So we have two transactions here. Let's say these transactions run concurrently and both the

transactions read ₹ 400 balance, in that case the final balance of A would be ₹ 300 instead of ₹ 200. This is wrong. If the transaction were to run in isolation then the second transaction would have read the correct balance ₹ 300 (before debiting ₹ 100) once the first transaction went successful.

Isolation:

- Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way.
- The isolation property ensures that for every pair of transactions, one transaction should start execution only when the other finished execution.

Durability:

- The durability assures that once transaction completes successfully, all updates that it carried out on the database persist even if there is a system failure.
- We can guarantee durability by ensuring that either:
  1. The updates carried out by a transaction have been written to disk before the transaction completes.
  2. Information about updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the database system is restarted after failure.

2.1.3 States of Transaction (April 17)

- Fig. 2.2 shows a state transition diagram of a transaction.
- A transaction is a sequence of operations that transforms the database from one state to the other state. Fig. 2.2 illustrates how a transaction moves through its execution states.

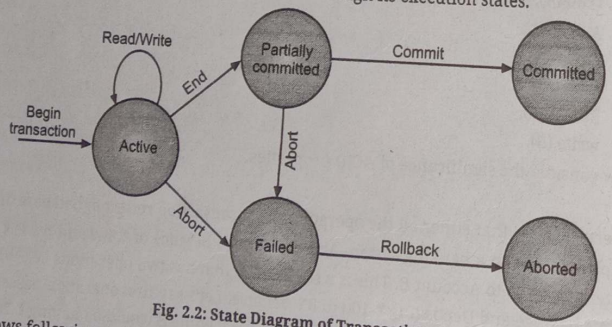


Fig. 2.2: State Diagram of Transaction

- Fig. 2.2 shows following states of a transaction:
  1. **Active State:** This is the initial state of transaction. A transaction is active when it is executing. A transaction always starts with active state. It remains in active state till all commands of that transaction are executed.
  2. **Partially Committed:** When a transaction completes its last statement (command), it enters in partially committed state.
  3. **Failed:** If the system decides that the normal execution of the transaction can no longer proceed, then transaction is termed as failed. If some failure occurs in active state or partially committed state, transaction enters in failed state.
  4. **Committed:** When the transaction completes its execution successfully it enters committed state from partially committed state.

  5. **Aborted:** To ensure the atomicity property, changes made by failed transaction are undone i.e. the transaction is rolled back. After rollback, that transaction enters in aborted state. When the transaction is in failed state, it rollbacks that transaction and enters in aborted state.

- When the transaction is in aborted state, the system has two options:
  1. If the transaction was aborted as a result of some hardware or software error (software error which is not created because of some internal logic of transaction), then such transaction can be restarted and that transaction is considered to be a new transaction.
  2. If the transaction was aborted because of some internal logical error which can be corrected only by rewriting of the application program or because of the input was bad or the desired data were not found in the database, then system can kill such transactions.

2.2 EXECUTING TRANSACTIONS CONCURRENTLY

- Concurrent execution of transaction means executing more than one transaction at the same time or parallel.
- In concurrent execution, the DBMS controls the execution of two or more transactions in parallel however, it allows only one operation of any transaction to occur at any given time within the system.
- The DBMS interleaves the actions of different transactions to improve performance, in terms of increased throughput or improved response times for short transactions, but not all interleaving should be allowed.
- Ensuring transaction isolation while permitting such concurrent execution is difficult but is necessary for performance reasons.
  1. While one transaction is waiting for a page to be read in from disk, the CPU can process another transaction. This is because I/O activity can be done in parallel with CPU activity in a computer. Overlapping I/O and CPU activity reduces the amount of time disks and processors are idle, and increases system throughput (the average number of transactions completed in a given time).
  2. Interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly. In serial execution, a short transaction could get stuck behind a long transaction leading to unpredictable delays in response time or average time taken to complete a transaction.

     Following table gives concept of interleaving, firstly the transaction T_1 executes and then transaction T_2 and then again transaction T_1, this condition is known as interleaving. This way multiple transactions are allowed to run concurrently in the system.

T_1	T_2
read (A)	
write (A)	
	read (B)
	write (B)
read (C)	
write (C)	

2.2.1 Problem Associated in Concurrent Execution (Oct. 17)

- When concurrent transactions are executed in an uncontrolled manner, several problems can occur. The concurrency control has the three main problems/issues namely, Lost updates, Dirty read (or uncommitted data) and Unrepeatable read (or inconsistent retrievals).

1. **Lost Update Problem/Issue:**
- A lost update problem occurs when two transactions that access the same database items have their operations in a way that makes the value of some database item incorrect.
- In other words, if transactions T_1 and T_2 both read a record and then update it, the effects of the first update will be overwritten by the second update. **(Oct. 17)**

2. **Dirty Read (or Uncommitted Data) Problem/Issue:**
- A dirty read problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated database item is accessed by another transaction before it is changed back to the original value.
- In other words, a transaction T_1 updates a record, which is read by the transaction T_2. Then T_1 aborts and T_2 now has values which have never formed part of the stable database.

3. **Unrepeatable Read (or Inconsistent Retrievals) Problem/Issue:**
- Unrepeatable read (or inconsistent retrievals) issue occurs when a transaction calculates some summary (aggregate) function over a set of data while other transactions are updating the data.
- The problem/issue is that the bail action might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.
- In an unrepeatable read, the transaction T_1 reads a record and then does some other processing during which the transaction T_2 updates the record. Now, if T_1 rereads the record, the new value will be inconsistent with the previous value.

2.2.2 Introduction to Concurrency Control

- The ability of a database system which handles simultaneously or a number of transactions by interleaving parts of the actions or the overlapping is called concurrency. Concurrency control is the management of concurrent transaction execution.
- Concurrency can be defined as, "the ability for multiple transactions to access or change shared data at the same time".

Goals/Objectives of Concurrency Control:

1. To ensure that current transactions operate correctly without any loss of database consistency.
2. To ensure that concurrent transaction do not interfere with each other operation.
3. To ensure isolation of transaction in order to guard the integrity of the database.
4. To ensure the atomicity (or serialisability) of the execution of transactions in a multi-user database environment.

Need/Purpose of Concurrency Control:

1. **Preserve Database Consistency:** To maintain the consistency of shared data access during concurrent execution.
2. **Minimize Transaction Size:** The smaller a transaction is the less likely it is to interact with other transactions.
3. **Run Resource-Intensive Operations:** Whenever, possible, schedule operations that include long-running transactions to run after hours or during off-peak times.
4. **Limit Transaction Operations:** The transaction in a database should accomplish one task and should include only those statements needed to accomplish that task.
5. **Data Isolation:** To enforce isolation among conflicting transactions.
6. **Access Resources in a Consistent Order:** To minimize the possibility of two transactions interfering with or mutually blocking each other.

7. **Control over Database:** Control is needed to co-ordinate concurrent accesses to a DBMS so that the overall correctness of the database is maintained.
8. **Minimize Resources Access Time:** To keep the time that hold a resource open for access to a minimum by going in doing what need to do, and then getting back out as quickly as possible.
9. **Resolve Conflicts:** To resolve read-write, write-read and write-write conflicts.

2.3 SCHEDULE **(April 16, 18; Oct. 16)**

- Schedule represents the chronological order in which instructions are executed in the system. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- A schedule can be defined as, "a sequence of operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions".
- For example, consider two transactions T_1 and T_2 are executing concurrently as given in following table:

T_1	T_2
read (X)	read (Y)
X=X+5	Y=Y+100
write (X)	read (X)
commit	X=X+Y
	write (X)
	commit

- Then a schedule S of $\{T_1, T_2\}$ is given as:

T_1	T_2
read (X)	read (X)
X=X+5	X=X+Y
	read (Y)
	Y=Y+100
write (X)	
commit	
	read (X)
	X=X+Y
	write (X)
	commit

2.3.1 Types of Schedules **(April 15, 16)**

- When two or more transactions are running concurrently, the steps of the transactions would normally be interleaved. The interleaved execution of transactions is decided by the database scheduler, which receives a stream of user requests that arise from the active transactions.
- A particular sequencing (usually interleaved) of the actions of a set of transactions is called a schedule. It is the sequential order in which instructions in a transaction is executed.
- There are two types of schedules as explained below:

(April 15, 18)

1. **Serial Schedule:**

- It consists of a sequence of instructions from various transactions where the instructions belonging to one single transaction appear together in that schedule.
- **Definition:** "If the actions (operations) of different transactions are not interleaved i.e., transactions are executed one-by-one from start to finish, the schedule is called as a serial schedule".
- **For example:** Consider the banking system of several accounts and a set of transactions that accesses and updates those accounts. Let T_1 and T_2 be two transactions. Assume initial balances of A and B are 1000 and 2000 respectively.

  T_1: Transfers ₹ 50 from account A to account B.

  T_1 : read (A);
  A = A – 50;
  write (A);
  read (B);
  B = B + 50;
  write (B).

  T_2: Transfers 10% of balance from account A to B.

  T_2: read (A);
  temp:= A * 0.1;
  A:= A – temp;
  write (A);
  read (B);
  B:= B + temp;
  write (B).

- For T_1 and T_2, two serial schedules are possible as explained below:

(i) <T_1, T_2> Serial Schedule: It will execute transaction T_1 first and then T_2.

After executing both transactions, account balance of A is ₹ 855 and account balance of B is ₹ 2145 i.e., <T_1 T_2> preserves A + B i.e. the database are consistent.

Table 2.1: Schedule 1

T_1	T_2
read (A)	
A:= A – 50	
write (A)	
read (B)	
B:= B + 50	
write (B)	
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
	read (B)
	B:= B + temp
	write (B)

(ii) <T_2, T_1> Serial Schedule: It executes T_2 first and then T_1.

Table 2.2: Schedule 2

T_1	T_2
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
	read (B)
	B:= B + temp
	write (B)
read (A)	
A:= A – 50	
write (A)	
read (B)	
B:= B + 50	
write (B)	

Here, after executing both the transactions, values of A and B are ₹ 850 and ₹ 2150 respectively. In this case also A + B is constant and it preserves the consistency. Thus, for a set of n transactions, there exist n ! different valid serial schedules.

2. **Concurrent Schedule (Non-Serial Schedule):** **(April 16)**

- When several transactions are executed concurrently, the corresponding schedule is called concurrent schedule.
- Several execution sequences are possible since the various instructions from both transactions may now be interleaved.
- In general it is not possible to predict exactly how many instructions of a transaction will be executed before CPU switches to other transaction. Thus, the number of possible concurrent schedules for a set of n transactions is much larger than n!.
- **For example:** A concurrent schedule for T_1 and T_2 is given in Table 2.3.

  (i) Consistent Concurrent Schedule for T_1 and T_2: This concurrent schedule preserves the consistency of database and A + B is constant.

Table 2.3: Schedule 3

T_1	T_2
read (A)	
A:= A – 50	
write (A)	
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
read (B)	
B:= B + 50	
write (B)	
	read (B)
	B:= B + temp
	write (B)

(ii) Inconsistent concurrent schedule for T_1 and T_2:

Table 2.4: Schedule 4 (Concurrent Schedule)

T_1	T_2
read (A)	
A:= A – 50	
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
	read (B)
write (A)	
read (B)	
B:= B + 50	
write (B)	
	B:= B + temp
	write (B)

2.3.2 Concept of Serializability

- For a transaction, a serial schedule always results in a consistent database and not every concurrent schedule can result in consistent database.
- But a concurrent schedule results in a consistent state if its result is equivalent to a serial schedule of that transaction. Such concurrent schedule is known as **serializable**.
- A serialisable schedule is a schedule that follows a set of transactions to execute in some order such that the effects are equivalent to executing them in some serial order like a serial order.
- A non-serial schedule that is equivalent to some serial execution of transactions is known as a serializable schedule.
- The most widely accepted notion of correctness for concurrent execution of transactions is serializability which is the property that an (possibly interleaved) execution of a group of transactions has the same effect on the database, and produces the same output, as some serial (i.e., non-interleaved) execution of those transactions.
- Serialization refers to, the transactions running serially, one after the other, rather than concurrently.
- A serializable schedule is defined as,

  "given (an interleaved execution) a concurrent schedule for n transactions; the following conditions hold for each transaction in the set.

  1. All transactions are correct i.e. if any one of the transactions is executed on a consistent database, the resulting database is also consistent.
  2. Any serial execution of the transactions is also correct and preserves the consistency of the database."
- The objective/goal of serialisability is to find the non-serial schedules that allow transactions to execute concurrently without interfering with one another and thereby producing a database state that could be produced by a serial execution.
- There are two forms of serializability conflict serializability and view serializability as explained below.

1. Conflict Serializability:

- When the schedule (S) is conflict equivalent to some serial schedule (S'), then that schedule is called as conflict serializable schedule.
- Conflict serializability can be defined as, "a concurrent execution of n transactions say T_1, T_2, ... T_n (call this execution S) is called conflict serializable if the execution is conflict equivalent (if two schedules have the same set of operations and the same conflict relations) to a serial execution of the n transaction".
- Consider that T_1 and T_2 are two transactions and S is a schedule for T_1 and T_2. I_i and I_j are two instructions. If I_i and I_j refer to different data items, then I_i and I_j can be executed in any sequence.
- But, if I_i and I_j refer to same data items then the order of two instructions may matter. Here, I_i and I_j can be a read or write operation only. Hence, following three conditions are possible:

  (i) I_i = read (A)
  I_j = read (A)
  The order of I_i and I_j does not matter because both are reading the data.

  (ii) I_i = read (A) I_j = write (A)
  I_i = write (A) I_j = read (A)
  Here, if read (A) is executed before write (A) then it will read the original value of A otherwise it will read that value of A which is written by write (A). Hence, the order of I_i and I_j matters.

  (iii) I_i = write (A) I_j = write (A)
  Here, order of I_i and I_j does not affect either T_i or T_j. But the database is changed, and it makes difference for next read.
- We say that I_i and I_j conflict if they are operated by different transactions on the same data item and at least one of them is write operation. i.e. only in case 1, I_i and I_j do not conflict.
- Consider an example of concurrent schedule 5.

Table 2.5: Concurrent Schedule 5

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

- Here, write (A) of T_1 conflicts with read (A) of T_2, similarly write (B) of T_1 conflicts with read (B) of T_2. But write (A) of T_2 does not conflict with read (B) of T_1 because both are accessing different data items.
- If I_i and I_j are two consecutive instructions of schedule S and if they do not conflict, then we can swap the order of I_i and I_j, to produce new schedule S'. We say that S and S' are equivalent since all instructions appear in the same order except for I_i and I_j whose order does not matter.

- Equivalent schedule for a schedule given in Table 2.5 can be obtained by Swap write (A) of T_2 with read (B) of T_1.

Table 2.6: Schedule 6 (Schedule after Swapping Instructions)

T_1	T_2
read (A)	
write (A)	
	read (A)
read (B)	
	write (A)
write (B)	
	read (B)
	write (B)

- Similarly swap the following in schedule given in Table 2.6.
  - (i) read (B) instruction and read (A) instruction of T_1 and T_2 respectively.
  - (ii) write (B) instruction and write (A) instruction of T_1 and T_2 respectively.
  - (iii) write (B) instruction and read (A) instruction of T_1 and T_2 respectively.
- The final schedule S' after these swapping is given in Table 2.7.

Table 2.7: Schedule 7

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

- Which is a serial schedule of T_1 and T_2. Thus concurrent schedule S is transferred to serial schedule S' by a series of swaps of non-conflicting instructions and schedules S and S' are conflict equivalent.
- We say that a schedule S is conflict serializable, if it is conflict equivalent to a serial schedule. Consider the schedule shown in Table 2.8.

Table 2.8: Schedule 8

T_1	T_2
read (Q)	
	write (Q)
read (Q)	

- This schedule is not conflict serializable, since it is not conflict equivalent to any serial schedule $<T_1, T_2>$ or $<T_2, T_1>$.

- There may be any two schedules which are not conflict equivalent but produce same outcome. Schedule given in Table 2.9 is not conflict serializable.

Table 2.9: Schedule 9 (Concurrent schedule)

T_1	T_5
read (A)	
A:= A – 50	
write (A)	
	read (B)
	B:= B – 10
	write (B)
read (B)	
B:= B + 50	
write (B)	
	read (A)
	A:= A + 10
	write (A)

- Result of above schedule is same as serial schedule $<T_1, T_5>$, but this is not conflict serializable, since in above schedule write (B) of T_5 conflicts with read (B) of T_1. Thus, we cannot move all instructions of T_1 before those of T_5 by swapping consecutive non-conflicting instructions.

2. View Serializability:

- A schedule is view serializable, if it is view equivalent to some serial schedule. Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.
- View serializability can be defined as, "a concurrent execution of n transactions can be defined as, T_1, T_2, ... T_n (call this execution S) is called view serializable if the execution say is view equivalent to a serial execution of the n transaction".
- Consider two schedules S and S', where same set of transactions participate in both schedules. The schedules S and S' are said to be view equivalent if the following three conditions are satisfied:
  - (i) For each data item Q if transaction T_i reads the initial value of Q in schedule S, then transaction T_i in schedule S' must also read the initial value of Q.
  - (ii) For each data item Q if transaction T_i executes read (Q) in schedule S, and that value was produced by transaction T_j (if any), then transaction T_i in schedule S', must also read the value of Q that was produced by T_j transaction.
  - (iii) For each data item Q, the transaction that performs the final write (Q) operation in schedule S must perform the final write (Q) operation in schedule S'.
- A schedule S is view serializable if it is view equivalent to a serial schedule.
- Example of view equivalence: Schedule 1 is not view equivalent to schedule 2 since, in schedule 1 the value of account A read by transaction T_2 was produced by T_1, whereas this is not the case in schedule 2. Schedule 1 is view equivalent to schedule 3, because values of account A and B read by transaction T_2 were produced by T_1 in both schedules.

- Example of view serializable schedule, A schedule given in Table 2.10 is view serializable schedule.

Table 2.10: Schedule 10

T_3	T_4	T_6
read (A)		
	write (A)	
write (A)		
		write (A)

- This schedule is view equivalent to serial schedule $<T_3, T_4, T_6>$.

Note: Transactions T_4 and T_6 perform write (A) operations without having performed a read (A) operation. Writes of this form are called blind writes.

- Every conflict serializable schedule is view serializable, but there are view serializable schedules that are not conflict serializable.
- A view serializable schedule in which blind write appear is not a conflict serializable. Schedule 10 in Table 2.10 is view serializable but it is not conflict serializable.

2.3.3 Precedence Graph for Serializability

- A serializability schedule gives same result as some serial schedule. A serial schedule always gives correct result. i.e. a schedule that is serializable schedule is always correct. Hence, we must show that schedules generated by concurrency control scheme are serializable.
- A schedule can easily be tested for serializability through the use of a precedence graph. A precedence graph is a directed graph that contains a vertex for each committed transaction execution in a schedule (non-committed executions can be ignored).
- The precedence graph contains an edge from transaction execution T_i, to transaction execution T_j ($i \neq j$) if there is an operation in T_i that is constrained to precede an operation of T_j in the schedule.
- A schedule is serializable if and only if its precedence graph is acydic.

1. **Testing of Conflict Serializability:**

- There is an algorithm to establish the serializability of a given schedule for a set of transactions. This algorithm uses a directed graph called precedence graph, constructed from given schedule.
- **Precedence Graph:** It consists of a pair G = (V, E), where, V is the set of vertices and E is the set of edges.
- The set of vertices consists of all transactions participating in the schedule. The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of the following three conditions hold:
  - (i) T_i executes write (Q) before: T_j executes read (Q)
  - (ii) T_i executes read (Q) before: T_j executes write (Q)
  - (iii) T_i executes write (Q) before: T_j executes write (Q)
- A precedence graph is said to be acyclic if there are no cycles in the graph otherwise it is a cyclic graph.

Algorithm: Conflict serializability:

Step 1: Construct a precedence graph G for given schedule S.

Step 2: If the graph G has a cycle, schedule S is not conflict serializable.

- If the graph is acyclic, then find, using the topological sort given below, a linear ordering of transactions, so that if there is arc from T_i to T_j in G, T_i precedes T_j.
- Find a serial schedule as follows:
  - (i) Initialize the serial schedule as empty.
  - (ii) Find a transaction T_i, such that there are no arcs entering T_i, T_i is the next transaction in the serial schedule.
  - (iii) Remove T_i and all edges emitting from T_i. If the remaining set is non-empty, return to (ii), otherwise the serial schedule is complete.

Example:

(i) Given schedule is:

Table 2.11

T_{11}	T_{12}	T_{13}
read (A)		
	read (B)	
A:= f_1 (A)		
		read (C)
	B:= f_2 (B)	
	write (B)	
		C:= f_3 (C)
		write (C)
write (A)		
	read (A)	
	A:= f_4 (A)	
read (C)		
	Write (A)	
C:= f_5 (C)		
write (A)		
		B:= f_6 (B)
		write (B)

- The precedence graph for given schedule is shown in Fig. 2.3 (a). This graph contains cycle. Hence, the schedule is not conflict serializable.

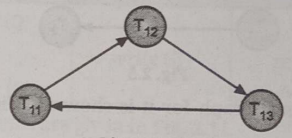


Fig. 2.3 (a)

(ii) The given schedule is:

Table 2.12

T_{14}	T_{15}	T_{16}
read (A)		
A:= f_1 (A)		
read (C)		
write (A)		
A:= f_2 (C)		
	read (B)	
write (C)		
	read (A)	
		read (C)
	B:= f_3 (B)	
	write (B)	
		C:= f_4 (C)
		read (B)
		write (C)
	A:= f_5 (A)	
	write (A)	
		B:= f_6 (B)
		write (B)

- The precedence graph for given schedule is shown in Fig. 2.3 (b).

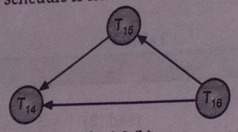


Fig. 2.3 (b)

- The graph in Fig. 2.3 (b) is acyclic. The conflict equivalent serial schedule for given schedule can be obtained using step 2 of algorithm.
- T_{14} is the transaction with no arcs entering in T_{14}. Hence, T_{14} is the first transaction in serial schedule. Remove T_{14} and all edges emitting from T_{14}.
- T_{15} is the next schedule, since it has no incoming edges. Remove T_{15} and edges emitting from T_{15}. T_{16} is the last schedule. Hence, the serial schedule which is conflict equivalent to given schedule is shown in Fig. 2.4. Hence, the schedule is conflict serializable.

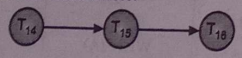


Fig. 2.4

(iii) Consider schedule 3, the precedence graph for it is given as in Fig. 2.5. Since, T_1 executes write (A) and write (B) before T_2 executes read (A) and read (B).

- This graph is acyclic and the conflict equivalent serial schedule is $T_1 \rightarrow T_2$. Hence, schedule 3 is conflict serializable.

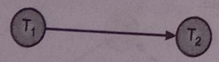


Fig. 2.5

(iv) Consider schedule 4, the precedence graph for it is shown in Fig. 2.6, which is a cyclic graph and hence it is not conflict serializable.

Fig. 2.6

(v) Consider the precedence graph given in Fig. 2.7.

- The graph is acyclic. The conflict equivalent serial schedule is equivalent to this are shown in Fig. 2.8.

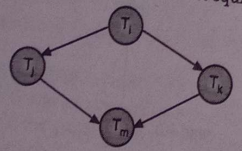
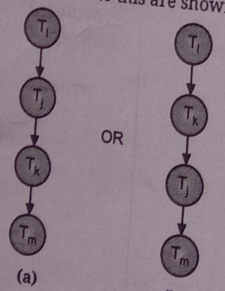


Fig. 2.7



(a) (b)

Fig. 2.8

Hence, the schedule corresponding to precedence graph in Fig. 2.8 is conflict serializable.

(vi) Consider the precedence graph in Fig. 2.9. The graph is acyclic. The serial schedules that are conflict equivalent to given schedule are shown in Fig. 2.10.

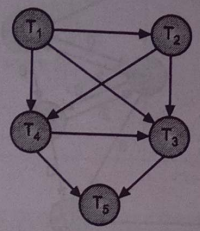
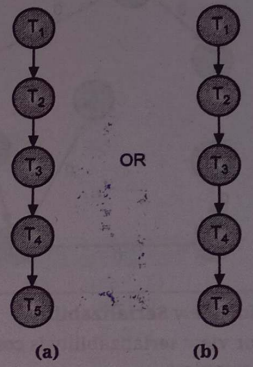


Fig. 2.9



(a) (b)

Fig. 2.10

- Hence, the schedule corresponding to given precedence graph is conflict serializable.

Examples:

Examples 1: Consider the following schedule:

T_3	T_4	T_7
read (Q)		
	write (Q)	read (Q)
write (Q)		write (Q)

Sol.: The labelled precedence graphs are given in Fig. 2.11.

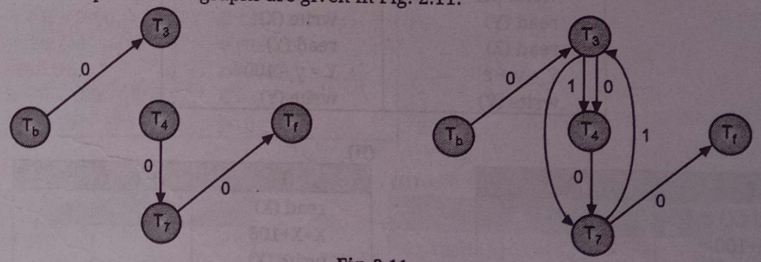


Fig. 2.11

Examples 2: Consider the following schedule:

T_3	T_4	T_7	T_8	T_9	T_{10}
read (Q)					
	write (Q)			read (A)	
		read (Q)	write (A)		
write (Q)		write (Q)		write (A)	
			write (B)		
					read (A)
					write (A)

Sol.: The labelled precedence graphs are shown in Fig. 2.12.

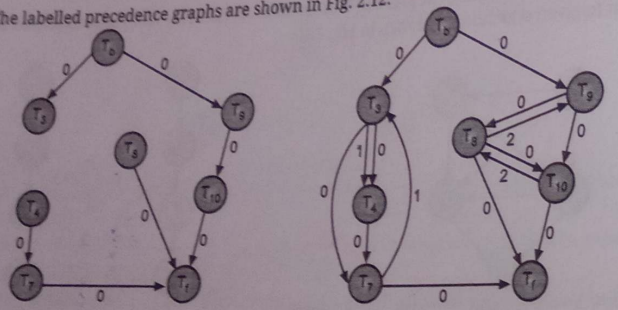


Fig. 2.12

2. Testing for View Serializability:

- Testing for view serializability is complicated. It has been shown that testing for view serializability is itself NP-complete.
- Thus there exists no algorithm to test for view serializability. However concurrency control schemes can still use sufficient conditions for view serializability.
- That is if sufficient conditions are satisfied, the schedule is view serializable schedule. But there may be view serializable schedules that do not satisfy the sufficient conditions.

Solved Examples:

1. Consider the following Transaction. Give two non-serial schedules that the serializable:

T_1	T_2
read (X)	read (Z)
X = x + 100	read (X)
write (X)	X = x – z
read (Y)	write (X)
read (Z)	read (Y)
Y = y + z	Y = y – 100
write (Y)	write (Y)

Sol.:

(i)

T_1	T_2
read (X)	
X=X+100	
write (X)	
	read (Z)
	read (X)
	X=X-Z
	write (X)
read (Y)	
read (Z)	
Y=Y+Z	
write (Y)	
	read (Y)
	Y=Y–100
	write (Y)

(ii)

T_1	T_2
read (X)	
X=X+100	
write (X)	
	read (Z)
read (Y)	
read (Z)	
	read (X)
	X=X-Z
	write (X)
Y=Y+Z	
write (Y)	
	read (Y)
	Y=Y–100
	write (Y)

2. Consider the following transactions. Give two non-serial schedules that are serializable.

T_1	T_2
read (Y)	read (X)
read (a)	read (a)
Y = y + a	X = x + a
write (Y)	write (X)
	read (Y)
	Y = y + a
	write (Y)

Sol.:

(i)

T_1	T_2
read (Y)	
read (a)	
	read (X)
	read (a)
Y=Y+a	
write (Y)	
	X=X+a
	write (X)
	read (Y)
	Y=Y+a
	write (Y)

(ii)

T_1	T_2
	read (X)
	read (a)
	X=X+a
	write(X)
read (Y)	
read (a)	
Y=Y+a	
write (Y)	
	read (Y)
	Y=Y+a
	write (Y)

3. Consider the following Transaction. Give two non-serial schedules that the serializable:

T_1	T_2
read (A)	read (B)
A = A – 1000	B = B + 100
write (A)	write (B)
read (B)	read (C)
B = B – 100	C = C + 100
write (B)	write (C)

Sol.:

(i)

T_1	T_2
read (A)	
A=A-1000	
write (A)	
	read (B)
	B=B+100
	write (B)
read (B)	
B=B-100	
write (B)	
	read (C)
	C=C+100
	write (C)

(ii)

T_1	T_2
read (A)	
A=A-1000	
	read (B)
write (A)	
	B=B+100
	write (B)
read (B)	
B=B-100	
	read (C)
write (B)	
	C=C+100
	write (C)

4. Consider the following transactions. Give two non-serial schedules that are serializable.

T_1	T_2
read (x)	read (x)
x=x+10	x=x-10
write (x)	write (x)
read (y)	read (y)
y=y+20	y=y-20
write (y)	write (y)
read (z)	
z=z+30	
write (z)	

Sol.:

(i)

T_1	T_2
read (x)	
x=x+10	
write (x)	
	read (x)
	x=x-10
	write (x)
read (y)	
y=y+20	
write (y)	
	read (y)
	y=y-20
	write (y)
read (z)	
z=z+30	
write (z)	

(ii)

T_1	T_2
read (x)	
x=x+10	
write(x)	
	read(x)
read(y)	
	x=x-10
	write (x)
y=y+20	
write (y)	
	read (y)
read (z)	
	y=y-20
	write (y)
z=z+30	
write (z)	

5. Consider the following Transaction. Give two non-serial schedules that are serializable to serial schedule $<T_1, T_2, T_3>$.

T_1	T_2	T_3
read (a)	read (c)	read (b)
a = a + 100	read (b)	b = b + 100
write (a)	b = b + c	write (b)
read (b)	write (b)	read (a)
b = b – 100	read (a)	c = c – 100
write (b)	a = a – c	write (c)
	write (a)	

Sol.:

(i)

T_1	T_2	T_3
read (a)		
a=a+100		
write (a)		
read (b)		
	read (c)	
b=b-100		
write (b)		
	read (b)	
	b=b+c	
	write (b)	
		read (b)
		b=b+100
		write (b)
	read (a)	
	a=a-c	
	write (a)	
		read (c)
		c=c-100
		write (c)

(ii)

T_1	T_2	T_3
read (a)		
a=a+100		
write(a)		
	read (c)	
read(b)		
b=b-100		
write(b)		
	read (b)	
	b=b+c	
	write (b)	
		read (b)
		b=b+100
		write (b)
	read (a)	
		read (c)
	a=a-c	
	write (a)	
		c=c-100
		write (c)

6. Consider the following transactions. Give two non-serial schedules that are serializable to serial schedule $<T_1, T_2>$.

T_1	T_2
read (c)	read (c)
read (a)	read (a)
a = a – c	a = a + c
write (a)	write (a)
read (b)	
b = b – c	
write (b)	

Sol.:

(i)

T_1	T_2
read (c)	
read (a)	
a=a-c	
write (a)	
	read (c)
	read (a)
read (b)	
	a=a+c
	write (a)
b=b-c	
write (b)	

(ii)

T_1	T_2
read (c)	
read (a)	
	read (c)
a=a-c	
write (a)	
	read (a)
read (b)	
b=b-c	
write (b)	
	a=a+c
	write (a)

2.4 ENSURING SERIALIZABILITY BY LOCKS

- One way to ensure serializability in a database is to require that one transaction is accessing a data item, no other transaction can modify/alter that data item.
- The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that data item.
- A lock describes the status of the item with respect to possible operations that can be applied to it. Generally, there is one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database item.
- The DBMS uses a locking mechanism to enable multiple, concurrent users to access and modify data in the database. Locking is necessary to enable the DBMS to facilitate the ACID properties of transaction processing.
- One way to ensure serializability is to associate with each data item a lock and to require that each transaction follow a locking protocol that governs how locks are acquired and released. There are various modes in which a data item can be locked.
- This section deals with some concurrency control schemes/algorithms like Lock-based protocol, Timestamp based protocol, Multiple granularity, Multiversion schemes etc., which ensure the serializability property of the database schedules.

2.4.1 Concept of Lock/Locking (April 17)

- Locking data items is one of the main techniques for controlling the concurrent execution of transactions. A lock is a variable, associated with the data item, which controls the access of that data item.
- The data item involved in transaction has a lock associated with it and when a transaction intends to access it, has to examine the associated lock first. If no other transaction holds the lock, the scheduler locks the data item for that transaction.
- Generally, there is a lock for each data item in the database. A lock describes the status of the data item with respect to possible operations that can be applied to that item used for synchronizing the access by concurrent transactions to the database items.
- A transaction locks an object before using it. When an object is locked by another transaction, the requesting transaction must wait.
- A lock is a variable associated with each data item in a database. When updated by a transaction, DBMS locks the data item. Serializability could be maintained by this.
- Consider the database is made up of data items. A lock is a variable associated with each data item. Locks are granted and released by a lock manager.
- The principle data structure of a lock manager is the lock table. In the lock table, an entry consists of a transaction identifier, a granule identifier and lock type.
- Manipulating the value of lock is called locking. The value of lock is used in locking schemes to control the concurrent access and to manipulate the associated data items.
- The size of the data unit that is locked is called lock granularity. Granularity is a lockable unit in a lock-based concurrency control scheme. Lock granularity indicates that level of lock use.

2.4.2 Different Lock Modes (Oct. 17)

- A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- Locks prevents access to a database record by a second transaction until the first transaction has completed all of its actions. Generally, there is one lock for each data item in the database.

- Locks are used as means of synchronizing the access by concurrent transactions to the database items. Thus, locking schemes aim to allow the concurrent execution of compatible operations.
- Following are the two types or modes of locks:
  1. **Shared:** If a transaction T_i has obtained a shared mode lock (denoted by S) on item A, then T_i can read but it cannot write A. It is also called a read-locked item, since multiple transactions are allowed to read a database item concurrently.
  2. **Exclusive:** If a transaction T_i, has obtained an exclusive mode lock (denoted by X) on item A, then T_i can both read and write A. It is also called a write-locked item since a transaction exclusively holds the lock on an item, until it finishes updating the item.
- Depending on the type of the operation, the transaction requests a lock in an appropriate mode on data item.
- The request is made to the concurrency-control manager, and the transaction can proceed with operation only after the concurrency-control manager grants the lock to that transaction.

Compatibility Function:

- Given a set of lock modes, the compatibility function can be defined as:
  - Let A and B represent arbitrary lock modes. Suppose that a transaction T_i requests a lock of mode A on item Q on which transaction T_j currently holds a lock of mode B.
  - If transaction T_i can be granted a lock on Q immediately, in spite of presence of the lock of mode B, then we say that mode A is compatible with B. Such a function can be represented conveniently by a matrix. Compatibility of two modes of lock is given in compatibility matrix.
- A compatibility matrix has a row and column for each lock mode. Compatibility of two modes of lock is given in matrix 'comp' compatibility matrix given as follows:

	S	X
S	True	False
X	False	False

'comp' Lock-compatibility Matrix:

- If comp (A, B) is true it means that A is compatible with B. Notice that shared mode (S) is compatible with shared mode (S). At any time, several shared-mode locks can be held simultaneously on a particular data item.
- A transaction requests a shared lock on data item Q by executing the lock S(Q) instruction and an exclusive lock is requested through lock X(Q) instruction. A data item Q can be unlocked via the unlock (Q) instruction.
- Transaction T_i can unlock the data item Q by executing unlock (Q) instruction. But transaction must hold a lock on a data item, as long as it accesses the data item.
- Locking protocols indicate when a transaction may lock and unlock each of the data items. Each transaction must follow the set of rules specified by locking protocols.
- Locking protocols restrict the number of possible serializable schedules. This section deals with only those locking protocols which allow conflict serializable schedules.

Starvation of Locks:

- Transaction starvation usually happens to large or long transactions that need to lock many data items.
- Suppose that transaction T_2 has a shared mode lock on data item and T_1 requests an exclusive mode lock on same data item. Clearly T_1 has to wait for T_2 to release the shared mode lock.

- Meanwhile, suppose that, T_3 request a shared mode lock on same data item.
- This lock request is compatible with the lock granted to T_2, so T_3 may be granted the shared mode lock.
- At this point, T_2 may release the lock but still T_1 has to wait for T_3. But again there may be a new transaction T_4 requesting a shared mode lock on the same data item.
- It is possible that there is a sequence of transactions that each requests a shared mode lock on same data item and T_1 never gets the exclusive mode lock on the data item. The transaction T_1 may never make progress and is said to be starved.
- Starvation of transactions can be avoided by granting locks as follows. When a transaction T_1 requests a lock on a data item Q in a particular mode M, the lock is granted provided that:
  1. There is no other transaction holding a lock on Q in a mode that conflicts with M.
  2. There is no other transaction that is waiting for a lock on Q and that made its lock request before T_1.

2.4.3 Lock Based Protocols

- Database systems equipped with lock based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it.
- A locking protocol is set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

2.4.3.1 Two Phase Locking (2PL) Protocol (April 15, 16, 18)

- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks. To guarantee serializability, the locking and releasing operations on data items by transactions need to be synchronized for this problem the solution is Two Phase Locking (2PL) protocol.
- Two phase locking guarantees serialisability, which means that transactions can be executed in such a way that their results are the same as if each transaction's actions were executed in sequence without interruption.
- 2PL protocol requires that each transaction issue a lock and unlock requests in following two phases:
  1. **Growing Phase:** A transaction may obtain locks, but may not release any lock.
  2. **Shrinking Phase:** A transaction may release locks, but may not obtain any new locks. Initially the transaction is in growing phase. In this it acquires locks as needed. Once, the transaction releases a lock, it enters the shrinking phase and it can issue no more lock requests.
- The point in the schedule where the transaction has obtained its final lock, (the end of its growing phase) is called the lock point of the transaction. The transactions can be ordered according to lock points.
- This ordering gives the serializability ordering for transaction. This serial schedule is conflict equivalent i.e. the two-phase locking protocol ensures conflict serializability.
- **Example:** Following two transactions are two phase transactions:

T_3 :	T_4 :
lock-X (B);	lock-S (A);
read (B);	read (A);
B : = B – 50;	lock-S (B);
write (B);	read (B);
lock-X(A);	display (A + B);
read (A);	Unlock (A);
A : = A + 50;	Unlock (B).
write (A);	
Unlock (B);	
Unlock (A).	

- The unlock instructions do not need to appear at the end of the transaction. Two phase locking does not ensure freedom from deadlock. Transactions T_3 and T_4 are two phase, but they are deadlocked in the schedule.

Table 2.13

T_3	T_4
lock-X (B)	
read (B)	
B := B – 50	
write (B)	
	lock- S (A)
	read (A)
	lock-S (B)
lock-X (A)	
read (A)	
A := A + 50	
write (A)	
unlock (B)	
unlock (A)	
	read (B);
	display (A + B);
	unlock (A);
	unlock (B)

- Cascading rollback may occur under two phase locking. A single transaction failure leads to a series of transaction rollbacks, known as cascading rollback.
- The cascading rollbacks problem can be solved by using Strict 2PL protocol and Rigorous 2PL protocol.

2.4.3.2 Variations of Two Phase Locking

- Variations of 2PL locking are explained below:

1. **Strict Two Phase Locking Protocol (Strict 2PL):**

- Cascading rollbacks can be avoided by a modification of two phase locking called strict two phase locking protocol.
- It requires that in addition to locking being two phase, all exclusive-mode locks taken by a transaction must be held until that transaction commits.
- This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

2. **Rigorous Two Phase Locking Protocol (Rigorous 2PL):**

- It requires that all locks to be held until the transaction commits.
- With rigorous two-phase locking, transactions can be serialized in the order in which they commit. Most database system implements strict or rigorous two-phase locking.

3. **Conservative 2PL:**

- Conservative 2PL also called a static 2PL, which requires a transaction to lock all the items it accesses before the transaction begins execution by pre-declaring its read set and write set.

- Read set of a transaction is the set of all items that the transaction reads and write set of a transaction is the set of all items that the transaction writes. If any of the predeclared items needed can't be locked, the transaction doesn't lock any item. Instead, it waits until all the items are available for locking.
- Conservative 2PL is a deadlock free protocol.
- For example, consider the two transactions in two-phase locking protocol:

T_1	T_2
lock X (B)	lock S (A)
read (B)	read (A)
B = B – 50;	lock S (B)
write (B);	read (B)
lock X (A)	display (A + B)
read (A)	unlock (A)
A = A + 50	unlock (B)
write (A)	
unlock (B)	
unlock (A)	

- The solution is, two phase locking does not ensure freedom from deadlock T_1 and T_2 are two phase transmission but they are deadlocked.

T_1	T_2
lock X (B)	
read (B)	
B = B – 50	
write (B)	
	lock S (A)
	read (A)
	lock (B)
lock X (A)	
read (A)	
A = A + 50	
write (A)	
unlock (B)	
unlock (A)	
	read (B)
	display (A + B)
	unlock (A)
	unlock (B)

- Here, T_1 has a X lock and T_2 wants a S on B and T_2 has a S lock and T_1 wants a X on A.

2.5 BASIC TIMESTAMP METHOD FOR CONCURRENCY (April 18)

- In timestamps based concurrency control mechanism, all individual transaction is assigned a time-stamp when it enters the system.

- If an old transaction T_i and new transaction T_j is assigned timestamp like TS (T_i) and TS (T_j) respectively then in such case we can say that TS (T_i) < TS (T_j).
- In the concurrency control, a Timestamp (a unique identifier) created by the DBMS to identify the relative starting time of a transaction.
- Typically, timestamp values are assigned in the order in which the transactions are submitted to the system. So, a timestamp can be thought of as the transaction start time.
- Therefore, time stamping is a method of concurrency control in which each transaction is assigned a transaction timestamp.
- Timestamps must have two properties namely uniqueness, (assures that no equal timestamp values can exist) and monotonicity, (assures that timestamp values always increase).

2.5.1 Timestamp Based Protocols (April 15, 17, 18)

- Timestamp-based protocol decides the ordering of transactions in advance to determine the serializability order. For this it uses time-stamp ordering scheme.
- With each transaction T_i in the system, a fixed value called timestamp is associated, denoted by TS (T_i). This timestamp is assigned by database system before T_i starts execution.
- If transaction T_i is assigned a timestamp TS (T_i) and a new transaction T_j enters the system, then TS (T_i) < T_s (T_j). There are two simple methods for implementing timestamp scheme.
  1. Use the value of system clock as the timestamp i.e. a transaction's timestamp is equal to the value of system clock, when the transaction enters the system.
  2. Use a logical counter that is incremented after a new timestamp has been assigned. i.e. a transaction's timestamp is equal to the value of the counter when the transaction enters the system.
- Timestamp of transaction determines the serializability order. If TS (T_i) < TS (T_j) then the system must ensure that the resultant schedule is equivalent to serial schedule in which T_i appears before T_j.
- To implement this scheme, two timestamp values are associated with each data item Q.
  1. **W-Timestamp (Q):** It denotes the largest timestamp of any transaction that executed write (Q) successfully.
  2. **R-Timestamp (Q):** It denotes the largest timestamp of any transaction that executed read (Q) successfully.
- Whenever, a new read (Q) or write (Q) instruction is executed, these timestamps are updated.

Example: Consider the given transactions T_1 and T_2.

T_1	T_2
read (B)	read (B)
read (A)	B:=B-50
display (A+B)	write (B)
	read (A)
	A:=A+50
	write (A)
	display (A+B)

- Concurrent schedule for the two transactions is given below. Here we consider that $Ts(T_1) < Ts(T_2)$. The schedule given below is possible under the timestamp protocol.

T_1	T_2
read (B)	
	read (B)
	B:=B-50
	Write (B)
read (A)	
	read (A)
display (A+B)	
	A:=A+50
	write (A)
	display (A+B)

- Read (B) of T_1 is executed because TS (T_1) ≥ W-Timestamp (B) and it sets R-Timestamp (B) = TS (T_1).
- Read (B) of T_2 is executed because TS (T_2) ≥ W-Timestamp (B) and it sets R-Timestamp (B) = TS (T_2).
- Write (B) of T_2 is executed because TS (T_2) = R-Timestamp (B). Similarly, read (A) of T_1 and Write (A) of T_2 are executed.

2.5.2 Timestamp Ordering Protocol

- The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- Timestamp ordering in DBMS is a schedule-based concurrency management (control) method that preserves the transaction order sequence and it uses a timestamp protocol that manages transaction processing to ensure that the transaction order is persevered.
- A schedule in which the transactions participate is the serial table, and the evident serial schedule has the transactions in order of their timestamp values, this is called Timestamp Ordering (TO).
- Following are the three basic variants of timestamp-based methods of concurrency control:
  1. In a **partial timestamp ordering** only non-permutable actions are ordered to improve upon the total timestamp ordering.
  2. The **multiversion timestamp ordering** stores several versions of an update granule, allowing transactions to see a consistent set of versions for all granules it accesses.
  3. The **total timestamp ordering** depends on maintaining access to granules in timestamp order by aborting one of the transactions involved in any conflicting access.
- Timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- This protocol works as follows:

1. Suppose that transaction T_i issues read (Q).

(i) If TS (T_i) < W-Timestamp (Q), then T_i needs to read a value of Q that was already overwritten. Hence, the read operation is cancelled, and T_i is rolledback.

(ii) If TS (T_i) ≥ W-Timestamp (Q), then the read operation is executed and R-Timestamp (Q) is set to maximum of R-Timestamp (Q) and TS (T_i).

2. Suppose that T_i issues write (Q).

(i) If TS (T_i) < R-Timestamp (Q) then the value of Q that T_i is producing was needed previously and the system assumed that value would never be produced. Hence, the write operation is cancelled, and T_i is rolled back.

(ii) If TS (T_i) < W-timestamp (Q), then T_i is attempting to write an outdated value of Q. Hence, this write operation is cancelled, T_i is rolled back.

(iii) Otherwise the write operation is executed and W-timestamp is set to TS (T_i).

- A transaction T_i, that is rolled back by the concurrency control scheme as a result of either a read or write operation being issued, is assigned a new timestamp and is restarted.

Example: Consider the given transactions T_1 and T_2.

T_1: read (B);
read (A);
display (A + B)

T_2: read (B);
B := B – 50;
write (B);
read (A);
A := A + 50;
write (A);
display (A + B).

- Concurrent schedule for the two transactions is given below. Here, we consider that TS (T_1) < TS (T_2). The schedule given in Table 2.14 is possible under the timestamp protocol.

Table 2.14

T_1	T_2
read (B)	
	read (B)
	B : = B – 50
	write (B)
read (A)	
	read (A)
display (A + B)	
	A : = A + 50
	write (A)
	display (A + B)

- read (B) of T_1 is executed because TS (T_1) ≥ W-timestamp (B) and it sets R-Timestamp (B) = TS (T_1).
- read (B) of T_2 is executed because TS (T_2) ≥ W-timestamp (B) and it sets R-Timestamp (B) = TS (T_2).
- write (B) of T_2 is executed because TS (T_2) = R-Timestamp (B). Similarly, read (A) of T_1 and write (A) of T_2 are executed.

2.5.3 Thomas Write Rule (April 18)

- A modification to the basic timestamp ordering protocol that relaxes conflict serializability can be used to provide greater concurrency by rejecting obsolete write operations and the extension known as Thomas's write rule.
- Thomas proposed a new rule named as Thomas write rule which states that, "in order to increase the concurrency and to reduce the rollback of timestamp-based protocols, obsolete write operations can be ignored under certain circumstances."

- So, the protocol rules for read operations remain unchanged whereas for the write operations, they are slightly different from our timestamp ordering protocol.
- To understand the concept of Thomas write rule let us consider the schedule in Table 2.15.

Table 2.15

T_1	T_2
read (Q)	
	write (Q)
write (Q)	

- Apply timestamp ordering protocol to given schedule. Since, T_1 starts before T_2, TS (T_1) < TS (T_2), read (Q) of T_1 and write (Q) operations succeed.
- When T_1 attempts its write (Q) operation, we observe that, TS (T_1) < W-Timestamp (Q), since W_timestamp (Q) = TS (T_2), according to Timestamp protocol, write (Q) must be rejected, T_1 will be rolled back.
- Timestamp ordering protocol rolls back the transaction T_1, but the value of write (Q) Operation of T_1 is already written by write (Q) of T_2, and the value that write (Q) of T_1 is attempting to write will never be read, i.e. we can ignore the write (Q) of T_1.
- This leads to a modification of timestamp ordering protocol. This modified protocol operates as follows:

1. **Suppose that transaction T_i issues read (Q):**
   (i) If TS (T_i) < W-Timestamp (Q), then T_i needs to read a value of Q that was already over written. Hence, the read operation is rejected, and T_i is rolled back.
   (ii) If TS (T_i) ≥ W-timestamp (Q), then read operation is executed, and R-timestamp (Q) is set to the maximum of R-timestamp (Q) and TS (T_i).
2. **Suppose that transaction T_i issues write (Q):**
   (i) If TS (T_i) < R-Timestamp (Q) then the value of Q that T_i is producing was previously needed, and it was assumed that the value would never be produced. Hence, the write operation is cancelled, and T_i is rolled back.
   (ii) If TS (T_i) < W-timestamp (Q), then T_i is attempting write an outdated value of Q. Hence, the write operation can be ignored.
   (iii) Otherwise, the write operation is executed, and W-timestamp (Q) is set to TS (T_i). i.e., here if TS (T_i) < W-Timestamp (Q), then we ignore the obsolute write operation. This modification to timestamp ordering protocol is called Thomas write rule.

- Thomas write rule for schedule given in Table 2.15 results in a serial schedule < T_1, T_2 > which is view equivalent to given schedule.

2.6 LOCKS WITH MULTIPLE GRANULARITY

- Locking granularity is the size of the data item that the lock protects. Locking granularity is important for the performance of a database system.
- In the concurrency control schemes described so far, we have used each individual data item as the unit on which synchronization is performed.
- But in some cases it would be advantageous to group several data items and to treat them as one individual synchronization unit.

- **For example:** If a transaction T_i needs to access the entire database, and locking protocol is used, then it must lock each data item in the database.
- It would be better if T_i could issue a single lock request to lock the entire database. Similarly, if transaction T_i needs to access only one record in the database, it should not be required to lock the entire database. That is we need a mechanism to allow the system to define multiple levels of granularity.
- We can make one by allowing data items to be of various sizes and defining a hierarchy of data granularities where the small granularities are nested within larger ones. Such a hierarchy can be represented by tree. As an illustration consider the tree of Fig. 2.13.

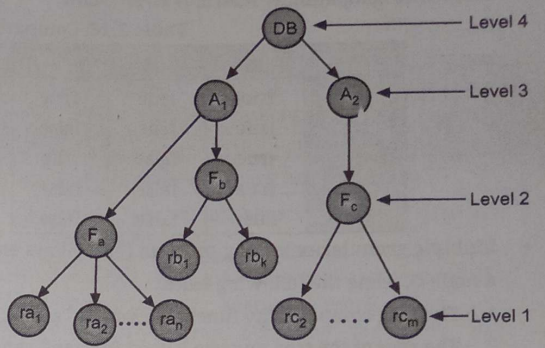


Fig. 2.13: Hierarchy which is represented by Tree

- The highest level represents the entire database. Below that are nodes of type area. Each area in turn has nodes of type file as its children. Each area contains exactly those files that are its child nodes. No file is in more than one area.
- Each file has nodes of type record. The file consists of those exactly records that are its child nodes and no record can be present in more than one file. Each node in the tree can be locked individually.
- A transaction in a database requires lock granularity on the basis of the operation being performed. An attempt to modify a particular tuple (data item) requires only that tuple to be locked.
- At the same time, an attempt to modify/alter or delete multiple tuples requires an entire relation to be locked. Since different transactions have dissimilar requests, it is desirable that the database system provides a range of locking granules called multiple-granularity locking.
- The efficiency of the locking mechanism can be improved by considering the lock granularity to be applied. However, the overheads associated with multiple-granularity locking can outweigh the performance gains of the transactions.
- Multiple-granularity locking permits each transaction to use levels of locking that are most suitable for its operations. This implies that long transactions use coarse granularity and short transactions use fine granularity.
- In this way, long transactions can save time by using few locks on a large data item and short transactions do not block the large data item, when its requirement can be satisfied by the small data item.
- Following are modes of lock used in **multiple granularity** scheme.
  1. Shared mode lock (S).
  2. Exclusive mode lock (X).
  3. Intension-shared mode lock (IS).
  4. Intension-exclusive mode lock (IX).
  5. Shared and Intension-exclusive mode lock (SIX).
- When transaction locks a node in shared or exclusive mode, the transaction also has locked all the descendents of that node in the same lock mode.
- If a node is locked in intension-shared mode, then explicit locking is being done at lower level of the tree, but with only shared mode locks.

- If a node is locked in shared and intension-exclusive mode, the sub-tree rooted by that node is locked explicitly in shared mode and that explicit locking is being done at a lower level with exclusive-mode locks. The compatibility matrix is given below:

Table 2.16: Compatibility Matrix

	IS	IX	S	SIX	X
IS	true	true	True	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

- Multiple granularity locking protocol that follows ensures serializability. Each transaction T_i can lock a node Q, using the following rules:
  1. The lock compatibility function given in Table 2.16 must be observed.
  2. The root of the tree must be locked first, and can be locked in any mode.
  3. A node Q can be locked by T_i in S or IS mode only if the parent Q is currently locked by T_i in either IX or IS mode.
  4. A node Q can be locked by T_i in X, SIX or IX mode only if the parent of Q is currently locked by T_i in either IX or SIX mode.
  5. T_i can lock a node only if it has not previously unlocked any node.
  6. T_i can unlock a node Q only if none of the children of Q are currently locked by T_i.

Example: Consider the tree given in Fig. 2.13 and following transactions:

1. Suppose that transaction T_{18} reads record r_{a2} in file F_a. Then T_{18} needs to lock, the database area A_1 and file F_a in 'IS' mode and to lock r_{a2} in S mode.
2. Suppose that transaction T_{19} modifies record r_{a9} in file F_a. Then T_{19} needs to lock the database area A_1 and file F_a in 'IX' mode and finally to lock r_{a9} in 'X' mode.
3. Suppose that transaction T_{20} reads all records in file F_a. Then T_{20} needs to lock the database area A_1 in 'IS' mode and finally to lock file F_a in 'S' mode.
4. Suppose that transaction T_{21} reads the entire database. It can do so after locking the database in 'S' mode.

2.6.1 Dynamic Database Concurrency (Phantom Problem)

- A static view of a database has been considered for locking but in reality, a database is dynamic since the size of database changes over time. Due to the dynamic nature of database, the phantom problem may arise.
- If the concurrency control is performed at the tuple granularity, the conflict may be undetected. This is also called as Phantom Problem.
- **For example:** We search for a dept-name = 'Physics'. Say T_1 requires to access all the records (tuples) pertaining Physics.

```
SELECT COUNT(*) FROM Department WHERE dept-name='Physics';
```

- And if T_2 is a transaction which is executed.

```
INSERT INTO department values (14,'XYZ','Physics',12300);
```

- Let us consider S is the schedule having T_1 and T_2 transactions.
- If T_1 uses the newly added tuple by T_2 while computing count(*) then T_1 reads a value written by T_2. So in serial schedule T_2 must come first before T_1.

2.7 TIMESTAMPING VERSUS LOCKING

- The difference timestamping and locking is given in Table 2.17.

Table 2.17

Sr. No.	Time Stamping	Locking
1.	Used to decide whether transaction should wait or rollback.	Used for concurrency control.
2.	It is used for deadlock prevention.	It is used to improve performance.
3.	Timestamp is a unique identifier to identify a transaction.	A lock is a variable associated with a data item in a database.
4.	Timestamping methods assign timestamps to each transaction and enforced serializability by ensuring that the transaction timestamps match the schedule for the transactions.	Locking methods prevents unsearializable schedules by keeping more than one transaction from accessing the same data elements.
5.	Timestamping methods may have causing more transaction abort than a locking protocol.	Locking methods does not have to abort transaction because they prevent potentially conflicting transactions from interacting with other transactions.
6.	Space is needed for read and write-times with every database element, whether or not it is currently accessed.	Space in the lock table is proportional to the number of database elements locked.

2.8 DEADLOCK AND DEADLOCK HANDLING

- A system is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. If $\{T_0, T_1, ..., T_n\}$ is the set of transactions such that T_0 is waiting for a data item that is held by T_1 and T_1 is waiting for a data item that is held by T_2 ... and T_{n-1} is waiting for a data item that is held by T_n. T_n is waiting for a data item that is held by T_0. Hence, none of the transactions can make progress in such situation. That is the system is in deadlock state.
- A deadlock can be defined as, "a situation (condition) in which each transaction in a set of two or more concurrently executing transactions is blocked waiting for another transaction in the set."
- In a database environment, a deadlock is a situation when transactions are endlessly waiting for one another. Any lock based concurrency control algorithm and some timestamp based concurrency control algorithms may result in deadlocks, as these algorithms require transactions to wait for one another.
- A deadlock is a condition in which two (or more) transactions in a set are waiting simultaneously for locks held by some other transaction in the set.
- **For example:** Consider following two transactions. T_1 transaction first read and then write on data item A that performs read and write on data item B.

T_1
read (A)
write (A)
read (B)
write (B)

- The T_2 transaction first read and then write data on data item B, then after that performs read and write on data item A.

T_2
read (B)
write (B)
read (A)
write (A)

- Consider above two transactions are executing using locking protocol as given below:

T_1	T_2	
lock-A (A)		
read (A)		
write (A)		
	lock-A (B)	
	read (B)	
	write (B)	
lock-A (B)		Wait for transaction T_2 to Unlock B
read (B)		
write (B)		
	lock-A (A)	Wait for transaction T_1 to Unlock A
	read (A)	
	write (A)	

- In above schedule transaction T_1 is waiting for transaction T_2 to Unlock data item B and transaction T_2 is waiting for transaction T_1 to Unlock data item A.
- So the system is in a deadlock state as there are set of transactions T_1 and T_2 such that, both are waiting for each other transaction to complete. This state is called as deadlock state.
- There are two methods for dealing with deadlock problems:
  1. Deadlock detection and deadlock recovery, (Wait for Graph) and
  2. Deadlock prevention, (Wound-Wait and Wait-die).

2.8.1 Deadlock Avoidance (Wound-Wait, Wait-Die) (April 17)

- The deadlock avoidance approach handles deadlocks before they occur. It analyzes the transactions and the locks to determine whether or not waiting leads to a deadlock.
- Transactions start executing and request data items that they need to lock. The lock manager checks whether the lock is available. If it is available, the lock manager allocates the data item and the transaction acquires the lock.
- However, if the item is locked by some other transaction in incompatible mode, the lock manager runs an algorithm to test whether keeping the transaction in waiting state will cause a deadlock or not. Accordingly, the schemes decide whether the transaction can wait or one of the transactions should be aborted.
- There are two schemes for this purpose, namely wait-die and wound-wait. Deadlock avoidance schemes never result in deadlocks of require potential deadlock situations are detected in advance and steps are taken such that they will no occur.

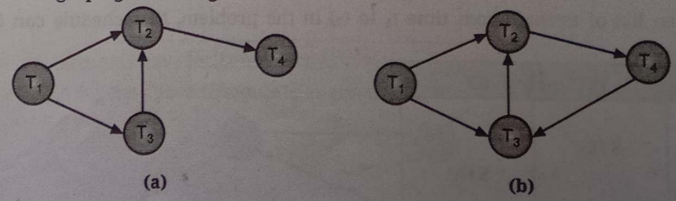
- Let us assume that there are two transactions, T_1 and T_2, where T_1 tries to lock a data item which is already locked by T_2. The algorithms are as follows:
  1. **Wait-Die:** If T_1 is older than T_2, T_1 is allowed to wait. Otherwise, if T_1 is younger than T_2, T_1 is aborted and later restarted.
  2. **Wound-Wait:** If T_1 is older than T_2, T_2 is aborted and later restarted. Otherwise, if T_1 is younger than T_2, T_1 is allowed to wait.

2.8.2 Deadlock Detection and Recovery (Wait for Graph) (April 15, 16, 17; Oct. 17)

- This is one method for dealing with deadlock and it allows the system to enter a deadlock state and then try to recover using deadlock detection and deadlock recovery scheme. If the probability that system enters deadlock state is relatively low, this method is efficient.
- Deadlock detection is defined as, a periodic check mechanism by DBMS to determine whether a deadlock has occurred or not.
- An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred. If it has occurred then the system must attempt to recover from the deadlock.
- Following are the requirements for deadlock detection and recovery:
  1. Maintain information about the current allocation of data items to transactions, as well as any outstanding data item requests.
  2. Provide an algorithm that uses this information to determine whether the system has entered a deadlock state.
  3. Recover from the deadlock when detection algorithm determines that a deadlock exists.
- Let us see deadlock detection and recovery in detail:

1. **Deadlock Detection:** (April 16, 17)

- Deadlock detection is periodic check by the DBMS to determine if the waiting line for some resource exceeds a predetermined limit.
- Deadlocks can be detected using a directed graph called wait for graph. In a wait-for graph, nodes of the graph represent transactions and edges of the graph represent the waiting-for relationships among transactions.
- The wait for graph consists of a pair G = (V, E) where, V is a set of vertices and E is a set of edges. The set of vertices consists of all transactions in the system. Each element in the set E of edges is an ordered pair $T_i \rightarrow T_j$.
- A directed edge $T_i \rightarrow T_j$ in graph imply that T_i is waiting for transaction T_j to release the data item.
- The edge $T_i \rightarrow T_j$ is removed when T_j is no longer holding a data item needed by transaction T_i.
- A deadlock exist in the system if the wait for graph for that system contain a cycle (or loop).
- Consider the wait graph given in Fig. 2.14.



(a) (b)

Fig. 2.14: Wait Graph

Transaction T_1 is waiting for T_2 and T_3.
Transaction T_2 is waiting for T_4.
Transaction T_3 is waiting for T_2.

- The graph contains no cycle. Hence, the system is not in deadlock state.
  But now consider the Fig. 2.12 (b). It contains a loop, $T_2 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$.
- Hence, the system is in deadlock state. In this way using this algorithm deadlocks can be detected. Depending on the frequency of deadlock occurrence, the deadlock detection algorithm should be executed.

(Oct. 17)

2. Deadlock Recovery:

- The most common solution to recover from deadlock is to rollback one or more transactions to break the deadlock.
- A transaction can be recovered using following three actions:

(i) Selection of Victim: Determine which transaction to rollback. Those transactions that will incur the minimum cost will be rolled back. The cost of rollback can be decided by following factors.

(a) How long the transaction has computed and how much longer the transaction will compute before it completes the assigned task?
(b) How many data items it has used?
(c) How many more data items it needs to complete?
(d) How many transactions will be involved in the rollback?

(ii) Rollback: One method to rollback a transaction is to abort transaction and restart it. The other more effective method is to rollback the transaction only as far as necessary to break the deadlock. But this require additional information about all the running transactions.

(iii)Starvation: It may happen that the same transaction is always selected as victim. This results in starvation. The most common solution is to include the number of rollbacks in the cost factor.

Solved Examples:

1. Following is the list of events in an interleaved execution of set T_1, T_2, and T_3 assuming 2PL (Two Phase Lock). Is there a Deadlock? If yes, which transactions are involved in Deadlock?

Time	Transaction	Code
t_1	T_1	lock (A, X)
t_2	T_2	lock (B, S)
t_3	T_3	lock (A, S)
t_4	T_1	lock (C, X)
t_5	T_2	lock (D, X)
t_6	T_1	lock (D, S)
t_7	T_2	lock (C, S)

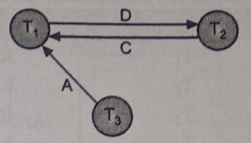
Sol.:

As per the given list of events (from time t_1 to t_7) in the problem, the schedule can be written as follows:

T_1	T_2	T_3
X (A)		
	S (B)	
X (C)		S (A)
	X (D)	
S (D)		
	S (C)	

Where X (A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:



From the above waits-for graph, there is cycle between T_1, T_2. Hence, there is deadlock.

2. Following is the list of events in an interleaved execution of set T_1, T_2, T_3 and T_4 assuming 2PL. Is there a Deadlock? If yes, which transactions are involved in Deadlock?

Time	Transaction	Code
t_1	T_1	lock (A, X)
t_2	T_2	lock (B, X)
t_3	T_3	lock (A, S)
t_4	T_4	lock (B, S)
t_5	T_1	lock (B, S)
t_6	T_2	lock (D, S)
t_7	T_3	lock (C, S)
t_8	T_4	lock (C, X)

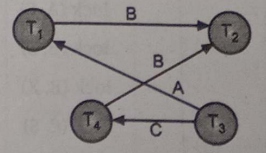
Sol.:

As per the given list of events (from time t_1 to t_8) in the problem, the schedule can be written as follows:

T_1	T_2	T_3	T_4
X (A)			
	X (B)		
		S (A)	
			S (B)
S (B)			
	S (D)		
		S (C)	
			X (C)

Where, X (A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:



From the above waits-for graph, there is no cycle. Hence, there is no deadlock.

3. Following is the list of events in an interleaved execution of set T_1, T_2, T_3 and T_4 assuming 2PL (Two Phase Lock). Is there a Deadlock? If yes, which transactions are involved in Deadlock?

Time	Transaction	Code
t_1	T_1	lock (A, X)
t_2	T_2	lock (B, X)
t_3	T_3	lock (A, S)
t_4	T_4	lock (B, S)
t_5	T_1	lock (B, S)
t_6	T_3	lock (D, X)
t_7	T_2	lock (D, S)
t_8	T_4	lock (C, X)

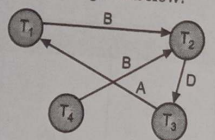
Sol.:

As per the given list of events (from time t_1 to t_8) in the problem, the schedule can be written as follows:

T_1	T_2	T_3	T_4
X (A)			
	X (B)		
		S (A)	
S (B)			S (B)
		X (D)	
	S (D)		
			X (C)

Where, X (A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:



From the above waits-for graph, there is cycle among T_1, T_2 and T_3. Hence, there is deadlock.

4. Following is the list of events in an interleaved execution of set T_1, T_2, T_3 and T_4 assuming 2PL. Is there a Deadlock? If yes, which transactions are involved in Deadlock?

Time	Transaction	Code
t_1	T_1	lock (A, X)
t_2	T_2	lock (B, S)
t_3	T_3	lock (A, S)
t_4	T_4	lock (B, S)
t_5	T_1	lock (B, X)
t_6	T_2	lock (C, S)
t_7	T_3	lock (D, S)
t_8	T_4	lock (D, X)

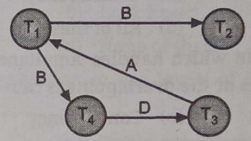
Sol.:

As per the given list of events (from time t_1 to t_8) in the problem, the schedule can be written as follows:

T_1	T_2	T_3	T_4
X (A)			
	S (B)		
		S (A)	
			S (B)
X (B)			
	S (C)		
		S (D)	
			X (D)

Where X (A); indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:



From the above waits-for graph, there is cycle among T_1, T_4 and T_3. Hence, there is deadlock.

5. Following is the list of events in an interleaved execution of set of transaction T_1, T_2, T_3 and T_4 assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock?

Time	Transaction	Code
t_1	T_1	lock (A, X)
t_2	T_2	lock (A, S)
t_3	T_3	lock (A, S)
t_4	T_4	lock (B, S)
t_5	T_1	lock (B, X)
t_6	T_2	lock (C, X)
t_7	T_3	lock (D, S)
t_8	T_4	lock (D, X)

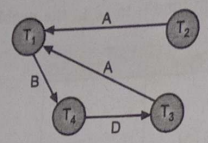
Sol.:

As per the given list of events (from time t_1 to t_7) in the problem, the schedule can be written as follows:

T_1	T_2	T_3	T_4
X (A)			
	S (A)		
		S (A)	
			S (B)
X (B)			
	X (C)		
		S (D)	
			X (D)

Where, X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:



From the above waits-for graph, there is cycle among T_1, T_4 and T_3. Hence, there is deadlock.

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. A set of logically related operations is called as?
   (a) Transaction (b) Concurrency
   (c) Both (a) and (b) (d) All of these
2. The ability of a database system which handles simultaneously or a number of transactions by interleaving parts of the actions or the overlapping is called as?
   (a) Transaction (b) Concurrency
   (c) Both (a) and (b) (d) All of these
3. To ensure the integrity of data, database system maintains which properties of transaction?
   (a) Atomicity: (All or Nothing)
   (b) Consistency: (No Violation of Integrity Constraints)
   (c) Isolation: (Concurrent Changes Invisibles) and Durability: (Committed Update Persist)
   (d) All of these
4. Which is also called history and a sequence of actions or operations?
   (a) Transaction (b) Lock
   (c) Schedule (d) None of these
5. Which is the schedule in which all the operations are completed before another transaction begin?
   (a) Serial (b) Concurrent
   (c) serializable (d) None of these
6. Which is a concurrency scheme ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order?
   (a) Atomicity (b) Availability
   (c) Serializability (d) All of these
7. Which is a variable, associated with the data item, which controls the access of that data item?
   (a) Schedule (b) Lock
   (c) Both (a) and (b) (d) None of these
8. One of the main techniques for controlling the concurrent execution of transactions on data items is known as?
   (a) Scheduling (b) Locking
   (c) Granting (d) All of these

9. The size of the data unit that is locked is called lock?
   (a) atomicity (b) granularity
   (c) consistency (d) None of these
10. The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the _____ of the transaction.
    (a) lock point (b) transaction point
    (c) concurrency point (d) All of these
11. A variant of 2PL protocol is _____.
    (a) Graph-based protocol (b) Strict Two-phase Locking Protocol
    (c) Timestamp-based protocol (d) All of these
12. Which protocol ensures that any conflicting read and write operations are executed in timestamp order?
    (a) 2PL (b) Lock based
    (c) Timestamp-ordering (d) None of these
13. Which locking permits each transaction to use levels of locking that are most suitable for its operations?
    (a) Multiple-granularity (b) Single-granularity
    (c) Both (a) and (b) (d) None of these
14. A condition in which two (or more) transactions in a set are waiting simultaneously for locks held by some other transaction in the set is called as?
    (a) Transaction (b) Deadlock
    (c) Concurrency (d) None of these

Answers

1. (a)	2. (b)	3. (d)	4. (c)	5. (a)	6. (c)	7. (b)	8. (b)	9. (b)	10. (a)
11. (b)	12. (c)	13. (a)	14. (b)						

Q.II Fill in the Blanks:

1. Collection of operations that forms a single logical unit of work is called _____.
2. _____ represents the chronological order in which instructions are executed in the system.
3. _____ control deals with interleaved execution of more than one transaction.
4. _____ is a set of properties that guarantee the reliability of processing of database transactions.
5. When several transactions are executed concurrently, the corresponding schedule is called _____ schedule.
6. _____ processing allows multiple transactions to run concurrently.
7. A in which the operations from a set of concurrent transactions are interleaved is called as _____ schedule.
8. The given concurrent schedule is said to be if _____ it produces the same result as some serial schedule of the transaction.

9. A schedule can easily be tested for serializability through the use of a _____ graph.
10. The _____ consists of a pair G = (V, E) where V is a set of vertices and E is a set of edges.
11. _____ protocol requires that each transaction issues a lock and unlock requests in two phases i.e., growing phase (a transaction may obtain locks, but may not release any lock) and shrinking phase (a transaction may release locks, but may not obtain any new locks.
12. _____ 2PL also called a static 2PL, which requires a transaction to lock all the items it accesses before the transaction begins execution by pre-declaring its read set and write set..
13. _____ can be detected using a directed graph called wait for graph.

Answers

1. Transaction	2. Schedule	3. Concurrency	4. ACID	5. concurrent
6. Transaction	7. non-serial	8. serializable	9. precedence	10. wait for graph
11. 2PL	12. Conservative	13. Deadlocks		

Q.III State True or False:

1. A transaction is an action or series of actions that is carried out by a single user or application program to perform operations/tasks for accessing the contents of the database.
2. A transaction is a sequence of READ and WRITE actions that arc grouped together to from a database access.
3. The acronym ACID properties of transaction presented as Atomicity, Consistency, Isolation and Durability.
4. The concurrency control has the three main problems/issues namely, Lost updates, Dirty read (or uncommitted data) and Unrepeatable read (or inconsistent retrievals).
5. A schedule is a list of actions from asset of transactions which must consist of all the instructions of those transactions.
6. A lock is a sequence of actions or operations such as, reading writing, aborting or committing that is constructed by merging the actions of a set of transactions, respecting the sequence of actions within each transaction.
7. A serialisable schedule is a schedule that follows a set of transactions to execute in some order such that the effects are equivalent to executing them in some serial order like a serial order.
8. Deadlock detection is defined as, a periodic check mechanism by DBMS to determine whether a deadlock has occurred or not.
9. A transaction can be recovered using following three actions namely, Selection of victim, Rollback and Starvation.
10. The wait-die and wait-wound schemes provide starvation.
11. A lock is a variable associated with each data item while manipulating the value of lock is called locking.

Answers

1. (T)	2. (T)	3. (T)	4. (T)	5. (T)	6. (F)	7. (T)	8. (T)	9. (T)	10. (F)	11. (T)

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. What is transaction?
2. What is concurrency?
3. What is schedule? Enlist its types.
4. What is the purpose of transaction management?
5. Define the term lock.
6. What is serializability?
7. What is granularity? Define it.
8. What is timstamp?
9. What is concurrency control?
10. Which operations performed by a transaction?
11. What is deadlock?
12. Define the schedule.
13. List techniques for deadlock recovery.
14. Compare time-stamping and locking, (any two points).
15. Enlist properties of transaction.

(B) Long Answer Questions:

1. Give the importance of transaction processing and concurrency control in a database..
2. Explain the term transaction concept in detail.
3. With suitable diagram explain different states of transaction.
4. Explain ACID properties of transaction in detail.
5. Explain the term schedule. Explain any two types of schedules. Also compare them.
6. What is serializability? Enlist its types.
7. What are the main difference between view serializability and conflict serializability.
8. Describe following schedules:
   (i) Cascadeless schedule
   (ii) Recoverable schedule
   (iii) Non-Recoverable schedule.
9. What is the need for concurrent execution of transactions?
10. Explain when schedule is conflict serializable when it is view serializable? Test if the given schedule is view serializable.

T_3	T_4	T_5
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

11. Differentiate between serial and serializable schedule.
12. Explain:
    (1) View serializability
    (2) Conflict serializability.
13. What is the test view serializability?
14. Explain two-phase lock protocol with example.
15. Explain timestamp-based and lock-based protocols.
16. When do deadlocks happen, how to prevent them, and how to recover if deadlock takes place?
17. State the rules for a timestamp based concurrency control protocol and demonstrate its working.
18. Explain any one locking protocol.
19. How to detect a deadlock? Explain in detail.
20. Describe procedure for deadlock recovery.
21. What is meant by deadlock handling?
22. What is a lock? Explain its different modes.
23. State and Explain Thomas write rule with example.
24. What is timestamp? Define it.
25. Write a note on phantom problem.
26. Compare between timestamping and locking.
27. Write short note on:
    (i) Multiple granularity.
    (ii) Transaction.
    (iii) Concurrency Control.

UNIVERSITY QUESTIONS AND ANSWERS

April 2015

1. Define serial schedule.

Ans. Refer to Section 2.3.1 Point (1). (1 M)

2. Define cascading rollback.

Ans. Refer to Page 2.25. (1 M)

3. Explain timestamp based protocol with read write conflicting conditions.

Ans. Refer to Section 2.5.1. (5 M)

4. The following is a list of events in an interleaved execution of set of transactions T_1, T_2, T_3, T_4 with two phase locking protocol.

Time	Transaction	Code
t_1	T_1	lock (B, X)
t_2	T_2	lock (A, X)
t_3	T_3	lock (C, S)

Contd...

t_4	T_4	lock (B, X)
t_5	T_1	lock (D, S)
t_6	T_2	lock (C, X)
t_7	T_3	lock (A, X)
t_8	T_4	lock (C, S)

Check if there is a deadlock using wait-for graph. If yes, which transactions are involved in deadlock? (5 M)

Ans. Refer to Section 2.8.2.

5. Explain two-phase locking protocol. (5 M)

Ans. Refer to Section 2.4.3.1.

April 2016

1. State the phases of 2PL protocol. (1 M)

Ans. Refer to Section 2.4.3.1.

2. Give the disadvantage of a concurrent schedule. (1 M)

Ans. Refer to Section 2.3.1, Point (2).

3. State how to detect a deadlock. (1 M)

Ans. Refer to Section 2.8.2.

4. Define a transaction. Explain its properties. (5 M)

Ans. Refer to Sections 2.1.1 and 2.1.2.

5. Consider the following non-serial schedule:

S : T1	T2
read(p)	
p:=p+50;	
	read(q);
	q:q–100;
write(p);	
	write(q);
read(q);	
	read(p);
	p:=p–10;
q:=q+100;	
write(q);	
	write(p);

Is this schedule is a conflict serializable to a serial schedule <T_1, T_2> ? (5 M)

Ans. Refer to Section 2.3.

April 2017

1. Draw the state diagram of transaction. (1 M)

Ans. Refer to Section 2.1.3.

2. What is locking? (1 M)

Ans. Refer to Section 2.4.1.

3. Explain wait-die and wound-wait deadlock prevention scheme. (5 M)

Ans. Refer to Section 2.8.1.

4. The following is a list of events in an interleaved execution of set of transactions T1, T2, T3, T4 with two phase locking protocol.

Time	Transaction	Code
t1	T1	LOCK (A, S)
t2	T2	LOCK (B, X)
t3	T3	LOCK (C, X)
t4	T4	LOCK (A, S)
t5	T1	LOCk (C, X)
t6	T2	LOCK (A, S)
t7	T3	LOCK (D, X)
t8	T4	LOCK (B, S)

Construct wait for graph according to above request. Is there deadlock at any instance? Justify.

Ans. Refer to Solved Example Page 2.37. (5 M)

5. Explain the stamp based protocol with read-write conflicting conditions.

Ans. Refer to Section 2.5.1. (3 M)

October 2017

1. What are different types of locks?

Ans. Refer to Section 2.4.2. (1 M)

2. List the actions of transaction recovered from deadlock.

Ans. Refer to Section 2.8.2, Point (2). (1 M)

3. Define transaction. Explain ACID properties of transaction.

Ans. Refer to Sections 2.1.1 and 2.1.2. (5 M)

4. Consider the following transaction:

T_2	T_1
read (x)	
x:=x–70;	
	read (y)
	y:=y+10;

Contd...

write (x);	
	write (y);
read (y)	
b:=b+70;	
	read (z)
	z=z–5;
write (y);	
	read (x);
	write (z);
	x=x–15;
	write (x);

Give at least two non-serial schedule that are serializable. (5 M)

Ans. Refer to Section 2.3.

5. Define throughput and dirty read problem. (2 M)

Ans. Refer to Section 2.2.1, Point (2).

April 2018

1. Define term serial schedule. (1 M)

Ans. Refer to Section 2.3.1, Point (1).

2. What is timestamp? (1 M)

Ans. Refer to Section 2.5.

3. Define cascading rollback. (1 M)

Ans. Refer to Page 2.25.

4. What is transaction? Explain ACID property of transaction. (5 M)

Ans. Refer to Sections 2.1.1 and 2.1.2.

5. The following is a list of event in an interleaved execution of set of transaction T_1, T_2, T_3, T_4, T_5 with two phase locking protocol: (5 M)

Time	Transaction	Code
t_1	T_1	lock (A, X)
t_2	T_2	lock (B, X)
t_3	T_3	lock (E, S)
t_4	T_4	lock (B, X)
t_5	T_5	lock (A, X)
t_6	T_4	lock (A, X)
t_7	T_1	lock (B, X)
t_8	T_5	lock (D, X)
t_9	T_3	lock (A, S)
t_{10}	T_2	lock (D, X)

Ans. Refer to Solved Examples page 2.37.

6. Consider the following non-serial schedule:

T_1	T_2	T_3
read (A)		
	write (A)	
write (A)		write (A)

Is this schedule conflict serializable?

Is this schedule view serializable?

Justify your answer.

If it is serializable, give its equivalent serial schedule. (5 M)

Ans. Refer to Section 2.3.

7. Write a short note on Thomas Write Rule. (5 M)

Ans. Refer to Section 2.5.3.

8. Explain timestamp based protocol. (5 M)

Ans. Refer to Section 2.5.1.

Syllabus ...

1. Relational Database Design Using PLSQL **(8 Hrs.)**

1.1 Introduction to PLSQL.
1.2 PL/pgSqL: Datatypes, Language Structure.
1.3 Controlling the Program Flow, Conditional Statements, Loops.
1.4 Stored Procedures.
1.5 Stored Functions.
1.6 Handling Errors and Exceptions.
1.7 Cursors.
1.8 Triggers.

2. Transaction Concepts and Concurrency Control **(10 Hrs.)**

2.1 Describe a Transaction, Properties of Transaction, State of the Transaction.
2.2 Executing Transactions Concurrently Associated Problem in Concurrent Execution.
2.3 Schedules, Types of Schedules, Concept of Serializability, Precedence Graph for Serializability.
2.4 Ensuring Serializability by Locks, Different Lock Modes, 2PL and its Variations.
2.5 Basic Timestamp Method for Concurrency, Thomas Write Rule.
2.6 Locks with Multiple Granularity, Dynamic Database Concurrency (Phantom Problem).
2.7 Timestamps versus Locking.
2.8 Deadlock and Deadlock Handling - Deadlock Avoidance (Wait-Die, Wound-Wait), Deadlock Detection and Recovery (Wait for Graph).

3. Database Integrity and Security Concepts **(6 Hrs.)**

3.1 Domain Constraints.
3.2 Referential Integrity.
3.3 Introduction to Database Security Concepts.
3.4 Methods for Database Security.
  3.4.1 Discretionary Access Control Method.
  3.4.2 Mandatory Access Control.
  3.4.3 Role Base Access Control for Multilevel Security.
3.5 Use of Views in Security Enforcement.
3.6 Overview of Encryption Technique for Security.
3.7 Statistical Database Security.

4. Crash Recovery **(4 Hrs.)**

4.1 Failure Classification.
4.2 Recovery Concepts.
4.3 Log base recovery Techniques (Deferred and Immediate Update).
4.4 Checkpoints, Relationship between Database Manager and Buffer Cache. ARIES Recovery Algorithm.
4.5 Recovery with Concurrent Transactions (Rollback, Checkpoints, Commit).
4.6 Database Backup and Recovery from Catastrophic Failure.

5. Other Databases **(2 Hrs.)**

5.1 Introduction to Parallel and Distributed Databases.
5.2 Introduction to Object Based Databases.
5.3 XML Databases.
5.4 NoSQL Database.
5.5 Multimedia Databases.
5.6 Big Data Databases.