CHAPTER 13

Module VI

Concurrency Control

Syllabus

Concurrency Control : Lock-based-protocols, Deadlock handling Timestamp-based protocols, Self-learning Topics: Study the various deadlock situation which may occur for a database designed in module V.

13.1 Concept of Concurrency Control

Q. Explain the concept of concurrency control (2 Marks)

- In a single user database only one user is accessing the data at any time.
- This means that the DBMS does not have to be concerned with how changes made to the database will affect other users.
- In a multi-user database many users may be accessing the data at the same time. The operations of one user may interfere with other users of the database.
- The DBMS uses concurrency control to manage multi-user databases.
- Concurrency control is concerned with preventing loss of data integrity due to interference between users in a multi-user environment.

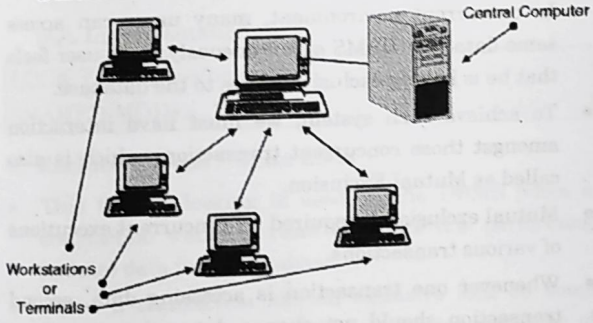

Fig. 13.1.1 : Concurrent database access

- Concurrency control should provide a mechanism for avoiding and managing conflict between various database users operating same data item at same time.

13.2 Problems Caused by Concurrency

Q. Explain various problems of concurrent executions. (4 Marks)

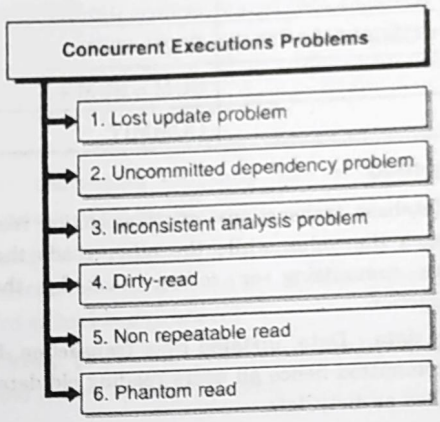

Fig. 13.2.1 : Concurrency execution problems

1. Lost update problem

- Update made by one transaction is overwritten by other transaction or other user.
- This may loss updates of one transaction.

2. Uncommitted dependency problem

This problem occurs when user sees data coming from intermediate step of another ongoing transaction which is yet uncommitted.

Time	X	Y	A
t_1	–	begin transaction	100
t_2		read (A)	100
t_3		A=A + 100	100
t_4	begin transaction	Write (A)	200
t_5	read (A)		200
t_6	A=A – 10	Rollback	100
t_7	write (A)		190
t_8	Commit		190

3. Inconsistent analysis problem

Transaction reads partial results of incomplete transaction update made by other transaction.

T_1	T_2
–	BEGIN Transaction
BEGIN Transaction	SUM=0
R (X)	R (X)
X = X – 20	SUM = SUM + X
W (X)	R (Y)
R (Z)	SUM = SUM + Y
Z = Z + 10	
W (Z)	
COMMIT	R (Z)
	SUM = SUM + Z
	COMMIT

4. Dirty-read

- In database transactions, one transaction reads and changes the value while the other reads the value before committing or rolling back by the first transaction.
- Dirty data : Data, updated by a transaction, but not yet committed hence all users reading old data which is called as dirty data.
- Dirty read : A transaction reading dirty data is called as 'dirty read'.
- Because there is always a chance that the first transaction might rollback the change which causes the second transaction reads an invalid value.
- In short, dirty read is changes made during a Transaction are 'visible' to other parties.
- Whether dirty reads are actually avoided or not depends on the database backend used and/or its configuration.

5. Non repeatable read

- A non-repeatable read occurs when a persistent object is read twice within a same transaction
- It is possible that between the reads, data is modified by another transaction, therefore the second read returns different values as compared to the first;
- If Transaction T_1 reads a row and Transaction T_2 changes the same row, when T_1 re-reads and sees the changes made by T_2,then this is non - repeatable read.

6. Phantom read

- Phantom reads means insert or delete action is performed on a table row which is referred by another transaction.
- Transaction T_2 inserts a row, T_1 re-
- reads the query and if T_1 see the additional row, it is a ghost row to T_1 then this is called as **Phantom read**.

13.3 Concurrency Control Schemes

- Fundamental property of transaction is isolation.
- Isolation property ensures that each transaction must remain unaware of other concurrently executing transactions.
- In case of concurrent transactions, when several transactions are executing simultaneously on a database may not preserve isolation property for long time.
- To implement concurrent system there must be interaction among various concurrent transactions.
- This can be done by using one of the concurrency control schemes.
- All the following schemes are based on serializability of schedules.

13.4 Lock Based Protocols

Q.	Explain concept of locking.	**(4 Marks)**
Q.	Explain algorithm of lock-based protocol.	**(4 Marks)**

1. Introduction (concept of locking) - implementation of isolation

- In concurrent environment, many users can access same data in a DBMS simultaneously; each user feels that he is having exclusive access to the database.
- To achieve such system, we must have interaction amongst those concurrent transactions which is also called as Mutual Exclusion.
- Mutual exclusion is required for concurrent executions of various transactions.
- Whenever one transaction is accessing data, second transaction should not change data otherwise there may be dirty read problem. This can be done with help of locking concept.
- Transaction can access data if it is locked by that transaction.
- Locking is necessary in a concurrent environment to assure that one process should not retrieve or update a record which another process is updating.
- Failure to this would result in inconsistent and corrupted data.

- For example, in case of traffic signals only one lane (line) is allowed to pass at a time and other lanes are locked. Similarly, in data transaction only one transaction can perform operations at a time other transitions are locked.

2. Types of locks

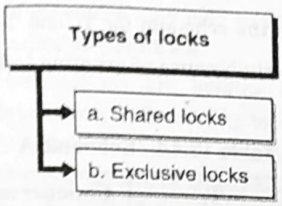

Fig. 13.4.1 : Types of locks

There are two types of locking to control concurrent access :

a. Shared locks / Read Lock

- This type of locking is used by the DBMS when a transaction wants to only read data without performing modification to it from the database.
- Another concurrent transaction can also acquire a shared lock on the same data, allowing the other transaction to read the data.
- Shared locks are represented by S.
- If a transaction T1 has obtained a shared lock on data item X, then transaction T1 can only read data item X, but cannot write on data item X.

SQL Implementation :

```
LOCK TABLE customer IN
SHARED MODE;
```

b. Exclusive locks / Write Lock

- This type of locking is used by the DBMS when a transaction wants to **read or write** (i.e. performing update) data in the database.
- When a transaction has an exclusive lock on some data, other transactions cannot acquire any type of lock (shared or exclusive) on the data.
- Exclusive locks are represented by X.
- If a transaction T_1 has obtained a exclusive lock on data item X, then transaction T_1 can read data item X and also can write on data item X.

SQL Implementation :

```
LOCK TABLE customer IN
EXCLUSIVE MODE;
```

c. Lock compatibility matrix

		Lock by Transaction T_j	
		S	X
Lock by Transaction T_i	S	No Conflict	Conflict
	X	Conflict	Conflict

- Transaction can acquire shared lock although there is other transaction which currently has a shared or an exclusive lock on the data. As many transactions can READ data without any conflict.
- Transaction can acquire an exclusive lock only if no other transaction currently has a shared or an exclusive lock on the data.
- To avoid these kinds of problems, every transaction in the system should follow a set of rules which are also called as **locking protocol**.
- **Locking protocols** tells when every transaction should lock or unlock data item.

3. Locking Levels

There are two locking levels to achieve concurrency :

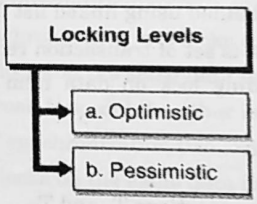

Fig. 13.4.2 : Locking levels

a. Optimistic

- Transactions with optimistic concurrency, work on the assumption that resource conflicts when more than one transaction works on the same set of data are unlikely (though possible).
- Optimistic transactions check for potential conflicts when committing changes to a database and conflicts are resolved by resubmitting data.

b. Pessimistic

- Pessimistic transactions expect conflicts from beginning and lock all resources.
- User can select either of the locking techniques.

13.4.1 Working of Locking Protocol / Locking Scheduler / Locking Manager

- Whenever any user transaction want to access any data item (D) then he will issue a locking request to concurrency control manager.
- A lock manager is a process that receives **locking request messages** and sends responsibility accordingly.
- Now based on some test criteria lock request can be granted or rejected.
- The response given may be
  - Lock grant - On granting to request.
  - If there is deadlock then rollback transaction.
  - If lock is already held by other transaction then asking to wait.
  - On unlock just an acknowledgement is sent.
- Transaction can access data if it is locked by that transaction.
- Lock manager uses linked list and hash tables to store information about all locking requests.
- The table used to store locking information is called as **lock-table**. Each record store transaction as locked data, locking mode, request grant response.
- Records are chained using linked list for a data item.
- The linked list is set of transaction requesting for that item and holding lock on data item and others are waiting.

Example

Consider two transactions T_1 and T_2,

T_1 :	LOCK – X(Q)
	READ (Q)
	Q = Q – 100
	WRITE (Q)
	UNLOCK (Q)
	LOCK-X(P)
	READ (P)
	P = P + 100
	WRITE(P)
	UNLOCK(P)
T_2 :	LOCK-S(P)
	READ(P)
	UNLOCK(P)
	LOCK – S(Q)
	READ (Q)
	UNLOCK (Q)
	DISPLAY(P + Q)

Following is the schedule for T_1 and T_2, which is not scheduled properly because of granting locks at improper timings.

Table 13.4.1 : Schedule A

T_1	T_2	Concurrency Control Manager
LOCK – X(Q)		Grant - X(Q, T_1)
READ (Q)		
Q = Q – 100		
WRITE (Q)		
UNLOCK (Q)		
	LOCK-S(P)	Grant - S(P,T_2)
	READ(P)	
	UNLOCK(P)	
	LOCK – S(Q)	Grant - S(Q,T_2)
	READ (Q)	
	UNLOCK (Q)	
	DISPLAY(P + Q)	
LOCK-X(P)		Grant - X(P,T_1)
READ(P)		
P = P + 100		
WRITE(P)		
UNLOCK(P)		

- In above schedule, T_1 unlocks Q very early, because of which the inconsistent state is exposed to transaction T_2 which results in wrong value of P + Q.
- Assume P = 400, Q = 600 then serial schedule < T_1, T_2 > will show 1000 but above given schedule will result

in 900 as T_1 has not updated P and before that only P + Q displayed by T_2.

- Every time transaction requests for lock and concurrency control manager grants it (if it is not conflicting) then only next instructions are executed. Always this sequence is followed.
- Such kind of schedules may lead to some undesirable conditions, called as **deadlock**.
- Both the transactions are holding a lock on two different data items, and waiting to acquire lock on each other's data item, this will never end up **waiting** of them and schedule cannot be executed further. This situation can be solved by roll back of one of the transactions. But this may lead to rollback of another transaction (such situation is called as cascaded rollback occurs in dependent transactions).

Locking protocol

- To avoid these kinds of problems, every transaction in the system should follow a set of rules called as 'locking protocol'.
- It tells when every transaction should lock and unlock data item.
- It restricts set of possible schedules and all those are serializable schedule.

13.4.2 Granting Locks

- Lock is granted only when no other conflicting type of lock on it, is held by other transactions.
- Lock compatibility matrix shows which locks will be granted and which will be rejected.
- It may happen that a series of transactions holding shared lock on data item X one by one so transaction T has to wait for exclusive lock on data item X forever, this situation is called as starvation of T.
- To avoid starvation, Concurrency Control Manager should consider two things,
  a. No other transaction is holding conflicting lock on it.
  b. No other transaction is waiting on that data item for locking it.

13.4.3 Rejecting Locks

- Lock is rejected when other conflicting type of lock is held by other transactions.
- If any transaction has exclusive lock for writing on data item D then no other transaction can acquire any type of lock on data item D.
- Lock compatibility matrix will show the locks that will be granted or rejected.

13.5 Two-Phase Locking (2PL) Protocol

Q. Explain advance locking concept using 2P locking. **(4 Marks)**

1. Introduction

- Two-Phase Locking (2PL) synchronizes read and write by explicitly detecting and preventing conflicts between concurrent operations.
- Before reading data item X, a transaction must "own" a read lock on X. Before writing into X, transaction must "own" a write lock on X.

The ownership of locks is governed by two rules :

a. Different transactions cannot simultaneously own conflicting locks (i.e. WR).
b. Once a transaction surrenders ownership of a lock, it may never obtain additional locks.

- The definition of conflicting lock depends on the type of synchronization being performed:
  (Refer lock compatibility matrix)
- For **'RW'** synchronization two locks conflict if
  - Both are locks on the same data item.
  - One is a read lock and the other is a write lock.
  - For **'WW'** synchronization two locks conflict if.
  - Both are locks on the same data item.
  - Both are write locks.
- The second lock ownership rule causes every transaction to obtain locks in a two phase manner.

Growing phase

In this phase, transaction may obtain n number of new locks but may not release any lock.

Shrinking phase

Transaction may release locks but cannot obtain any new Lock.

2. Working of 2P protocols

- Initially transaction is in 'growing phase', it acquires locks as needed. Once it starts releasing locks it enters into the 'Shrinking Phase' and now it may not acquire any lock after shrinking phase starts.

- The point at which transaction obtains final (last) lock is called as **lock point of transaction.**

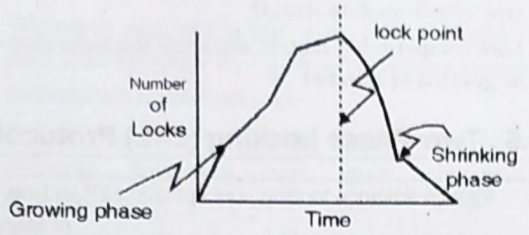

Fig. 13.5.1 : Locking point

- **Lock point** is end of growing phase for that transaction and start of shrinking phase.
- The schedule to be serializable, transactions should be arranged according to their lock points.
- When the transaction terminates (or aborts), all remaining locks are automatically released. Some systems also require that transactions hold all locks until termination.

3. Advantages

- Two-phase locking protocol ensures conflict serializability.
- Two-phase locking protocol is simple to implement and understand.

4. Drawbacks

- Deadlock may occur in two phase locked schedule.
- Cascaded rollback may occur under two phase locking. Cascaded rollbacks do mean by if one transaction is rolling back all transactions depends on this transaction will be rolled back.

Example

(A) No 2P (B) with 2P locking.

13.5.1 Modified Versions of Two-phase Locking Protocol

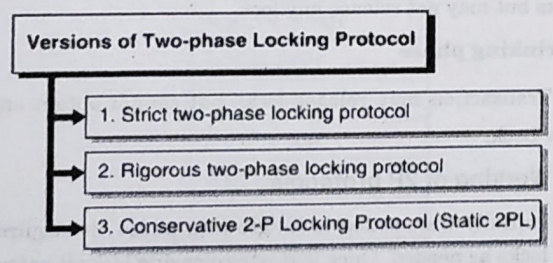

Fig. 13.5.2 : Versions of Two-phase Locking Protocol

1. Strict two-phase locking protocol

- Avoids cascaded rollbacks.
- It requires not only two-phase locking but also that all exclusive-locks held by transaction should be held until that transaction commits or abort.
- This property ensures that if data is being modified by one transaction (holding lock-X) then other transaction can't read it until first transaction commits.
- Strict schedule recoverability.
- Not deadlock free.

2. Rigorous two-phase locking protocol

- It also avoids cascading rollbacks.
- It requires that all share and exclusive locks to be held until the transaction commits.
- So transactions can be serialized in the sequence they commit.
- Most of the database systems implement strict or rigorous two phase locking protocol.

3. Conservative 2-P Locking Protocol (Static 2PL)

- It is also called as static 2P locking protocol.
- This scheme requires locking all items needed to access before the transactions starts.
- It begins execution by declaration about read set and write Set of all data items needed in advance.
  - **Read set :** Set of all data transactions lock.
  - **Write set :** Set of all data than n-calls transactions lock.
- If any one item of above list is currently not available for locking then lock will not be granted it waits till all items are ready for locking.
- It is almost free from deadlocks as all required items are listed in advanced.

13.6 Timestamp Based Protocols

Q. What do you mean by time stamp-based protocol? Explain timestamp modeling protocol. **(4 Marks)**

1. Introduction

- To achieve serializability order of transactions for execution can be decided in advance using its time at which transaction entered in system.
- A general method to achieve this is using time stamp ordering protocol.

2. Concept of Timestamp

- A fixed timestamp is assigned at start of execution of the transaction. Every transaction T_j has been assigned a timestamp by database system denoted as TS (T_j).
- A transaction which has entered in system recently will have greater timestamp. If transaction T_j starts after T_i then,
  TS (T_i) < TS (T_j)
- **System clock** is used as timestamp i.e. system time when transaction $T_i$enters system or a **logical counter** can be used as timestamp and incremented after every assignment.
- If TS(T_i) < TS(T_j), then system must ensure that serializable schedule is equivalent to a serial schedule as < T_i, T_j>
- Every data item X is with two timestamp values :

a. W-timestamp (X)

- It denotes the largest timestamp of any transaction that executed WRITE (X) successfully on given data item X.
- That mean it is timestamp of recent WRITE (X) operation.
- On execution of next WRITE (X) operation (O_1), W-timestamps will be updated to new timestamp of O_1 i.e. TS (O_1).

b. R-timestamp (X)

- Denotes the largest timestamp of any transaction that executed READ (X) successfully on data item (X).
- That mean it is timestamp of recent READ (X) operation.
- On execution of next READ (X) operation (O_1), R-timestamps will be updated to new timestamp of O_1 i.e. TS (O_1).

3. Timestamp - ordering Protocol

- This protocol ensures any conflicting READ or WRITE is executed in order of timestamp.
- If by any reasons, if transaction is aborted then on restarting, new timestamp is assigned.

Working

a. For transaction T_i to execute READ (X) Operation,

If TS (T_i) < W - timestamp (X)

Then T_i is trying to read value of X that is overwritten by other transaction.

W - timestamp (X) = 149 (Recent Write)			
TS	**T_i**	**T_x**	**Operations**
148	READ (X)	....	TS (T_i) < W - timestamp (X) i.e. 148 < 149 (W-Timestamp)
149	Unlock (X)	WRITE (X)	Record W - timestamp (X) = 149

So this READ is rejected (as T_x is already performing write operation on data so no other operation can be performed by any other transaction) and T_i is rolled back.

If TS (T_i) ≥ W - timestamp (X)

Then READ executed and set

R-timestamp(X) = max {TS (T_i), R-timestamp}			
TS	**T_i**	**T_x**	**Operations**
148	....	WRITE (X)	Record W - timestamp (X) = 148 (recent write)
149	READ (X)	....	TS (T_i) ≥ W - timestamp (X) i.e. 149 ≥ 148 (W-Timestamp) Record R-timestamp(X) = 149
150	....	....	....
151	READ (X)	....	TS (T_1) ≥ W - timestamp (X) i.e. 151 ≥ 148 (W-Timestamp) Record R-timestamp(X) = max{149,151} = 151

b. If transaction $T_i$execute WRITE (X) Operation,

If TS (T_i) < R - timestamp (X)

Then T_i has produced value of X which is not needed now so rollback T_i.

TS	**T_i**	**T_x**	**Operations**
148	....	...	...
149	WRITE(X)	....	TS (T_1) < R - timestamp (X) i.e. 149 < 151 (R-Timestamp) Reject WRITE roll back T_i
150	....	....	....
151		READ (X)	Record R-timestamp(X) = 151 (Recent READ Operation)

If TS (T_i) < W - timestamp (X)

Then T_i is trying to write obsolete value of X, So rollback T_i

TS	T_i	T_x	Operations
148	....	...	...
149	WRITE (X)	....	TS (T_1) < W - timestamp (X) i.e. 149 < 151 (W-Timestamp) Cannot write as other transaction using data X for writing.
150	....	....	....
151		WRITE (X)	Record W-timestamp(X) = 151

c. Otherwise, system executes WRITE (X) and sets W-timestamp (X) = TS (T_i)

TS	T_i	T_x	Operations
150	....	....	....
151	WRITE (X)		Record W-timestamp(X) = 151

4. Advantages

- This protocol ensures conflict serializability, as conflicting operations are processed in order of timestamp of operation.
- Ensures that it is free from deadlock.

5. Disadvantages

- Starvation is possible for long transaction if short transaction conflicts with it causes restorative of long transaction again and again.
- It may not give recoverable schedules.

13.7 Thomas' Write Rule

Q. Explain Thomas write rule with algorithm. **(4 Marks)**

1. Introduction

- Thomas Write Rule is also known as Modified timestamp protocol.
- Thomas Write Rule uses view serializability.
- It generates schedules which are not possible by other protocols.
- Generated schedules that are view equivalent to the serial schedule.

2. Example

T_1	T_2
READ (D)	
	WRITE (D)
WRITE (D)	

- The above schedule with 2 transactions T_1 and T_2.
- As T_1 starts before T_2 so timestamp TS (T_1) < TS (T_2). T_1 executes READ (D), and then T_2 executes WRITE (D). Now W-timestamp (D) = TS (T_2) so when T_1 executes WRITE (D), but TS (T_1) <W- timestamp (D), So WRITE (D) of T_1 is rejected and T_1 is rolled back.
- But this rollback is unnecessary. Since T_2 has already written value of D and T_1 is trying to write value which will never be read.
- So any time if T_i with TS (T_i) < TS (T_2) trying to READ (D) will be rollback as TS (T_i) < W - timestamp (D). Any transaction T_j with TS (T_j) > TS (T_2) should read value of D by T_2.
- So such WRITE (D) operations (as in T_2) are obsolete and should be ignored.

3. Thomas' Write Rule

- To achieve this functionality timestamp ordering protocol is modified which is called as **Thomas Write Rule**.
- Rejects few write (D) operations by modifying check for WRITE(D).
- Suppose, T_i issues WRITE (D)

a. If TS (T_i) < R - timestamp (D)

Then T_i is producing value of D, which needed previously not now Assuming it is never produced the system rejects WRITE and T_i is rollback.

TS	T_i	T_x	Operations
148	....	...	...
149	WRITE(X)	....	TS (T_1) < R - timestamp (X) i.e. 149 < 151 (R-timestamp) Reject WRITE roll back T_i
150	....	....	....
151		READ (X)	Record R-timestamp(X) = 151 (Recent READ Operation)

b. If TS (T_i) < W-timestamp (D)

Then T_i is writing obsolete or outdated value of D so WRITE is ignored.

TS	T_i	T_x	Operations
148	....	...	...
149	WRITE (X)	....	TS (T_1) < W - timestamp (X) i.e. 149 < 151 (W-timestamp) Cannot write as other transaction using data X for writing.
150	....	....	....
151		WRITE (X)	Record W-timestamp(X) = 151

c. Otherwise

WRITE (D) is executed and W-timestamp (D) = TS (T_j)

TS	T_i	T_x	Operations
150	....	....	....
151	WRITE (X)	....	Record W-timestamp(X) = 151

Compare to Basic Timestamp ordering

Unlike, timestamp protocol, Thomas write rule asks to ignore write operation if T_i issues WRITE (D) and TS (T_i) < W - timestamp (D).

13.8 Multiple Granularity Locking

1. Data Item Granularity

- Multiple granularities allow data items can be of various sizes and define a hierarchy of data granularities.
- In data granularity the small granularities are nested within larger granularity.
- Multiple granularity can be represented graphically as a tree as shown below in diagram.
- When a transaction locks a node in the tree explicitly, it automatically locks all the nodes below that node.
- Granularity of locking / Granularity of Data Items
  - **Fine granularity (lower in tree) :** high concurrency, high locking overhead
  - **Coarse granularity (higher in tree) :** low locking overhead, low concurrency

2. Example

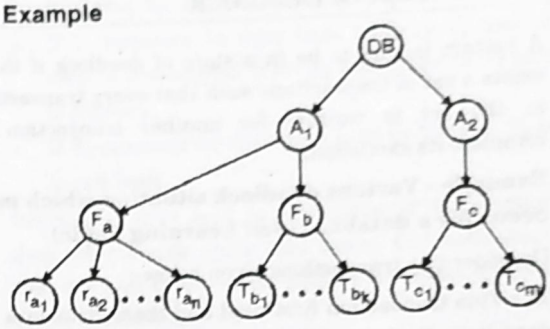

Fig. 13.8.1

The levels, starting from the coarsest (top) level are :

1. Database
2. Area
3. File
4. Record

3. Multiple Granularity Locking

- There are three additional lock modes with multiple granularities.
- Intention-shared :
  - Explicit locking at a lowest level of tree
  - It works with shared lock mode.
- Intention-exclusive (IX) :
  - Explicit locking at a lower level
  - It can works with exclusive or shared locks
- Shared and intention-exclusive (SIX):
  - Node is locked explicitly in shared mode
  - Explicit locking is being done to a lower level with exclusive-mode
- Intention locks allow a higher level node to be locked in Shared or Exclusive mode without having to check all descendent nodes.
- Lock Compatibility Matrix

	IS	IX	S	S IX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	×	✓	×	×
S IX	✓	×	×	×	×
X	×	×	×	×	×

13.9 Concept of Deadlock

1. A system is said to be in a state of deadlock if there exists a set of transactions such that every transaction in the set is waiting for another transaction to complete its execution.
2. **Example - Various deadlock situation which may occur for a database (Self Learning Topic)**

   Consider two transactions given below :

   T_1 : This transaction first read and then write data on data item X, then after that performs read and write on data item Y.

T_1
Read (X)
Write (X)
Read (Y)
Write (Y)

   T_2 : This transaction first read and then write data on data item Y, then after that performs read and write on data item X.

T_2
Read (Y)
Write (Y)
Read (X)
Write (X)

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

T_1	T_2	
Lock-X(X)		
Read (X)		
Write (X)		
	Lock-X(Y)	
	Read (Y)	
	Write (Y)	
Lock-X(Y)		Wait for transaction T_2 to Unlock Y
Read (Y)		
Write (Y)		
	Lock-X(X)	Wait for transaction T_1 to Unlock X
	Read (X)	
	Write (X)	

   - So in above schedule consider two transactions given below transaction T_1 is waiting for transaction T_2 to Unlock data item Y and transaction T_2 is waiting for transaction T_1 to Unlock data item X.
   - So 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**.
3. There are two principle methods to handle deadlock in system,
   a. Deadlock prevention

      We can use deadlock prevention techniques to ensure that the system will never enter a deadlock state.

   b. Deadlock detection and recovery

      We can allow the system to enter a deadlock state and then we can detect such state by **deadlock detection techniques** and try to recover from deadlock state by using and **deadlock recovery scheme**.
4. Both of above methods may result in transaction rollback.
5. Deadlock Prevention is commonly used if the probability that the deadlock occurs in system is high else detection and recovery are more efficient if probability of deadlock occurrence is less.

13.9.1 Approaches for Deadlock Prevention

1. Approach 1

A simplest form, in which a transaction acquires lock on all data items which will be required by transaction at the start of execution. It is effective as other transactions can't hold lock on those data items till first unlocks data item.

Disadvantages

a. It is difficult to know in advance which data items need to be locked.

b. Data utilization is very low as may be unused data items locked by transaction.

2. Approach 2

This approach uses timestamp ordering of data items. It is something like tree protocol and every transaction have to access data item in given sequence only. The variation of this approach with two phase protocol assures deadlock prevention. Order of data items must be known to every transaction.

1. To have concurrency control, two phase locking protocol is used which will ensure locks are requested in right order.

2. Timestamp ordering is determined by validation (T_i) i.e. TS (T_i) = validation (T_i) to achieve serializability.
3. The validation test for transaction T_j requires that for all transaction T_i with TS (T_i) < TS (T_j) then one of the following conditions must be hold.
   a. Finish (T_i) <start (T_j) as T_i finishes before T_j starts the serializability should be maintained.
   b. T_i completes its write phase before T_j starts its validation phase (Start (T_j) < finish (T_i) <Validation (T_j)).
4. This ensures writes of both transactions do not take place at same time.
5. Read of T_j not affected by writes of T_j and T_j can't affect read of T_i so serializability is maintained.

3. Approach 3 : Prevention and transaction rollbacks

1. Pre-emptive Technique
2. Pre-emption means, if transaction T_i wants to hold lock on data item hold by T_j. then system may preempt (UNLOCK all previous locks) T_i by rolling it back and granting lock to T_j on that data item.
3. To control Pre-Emption, Transaction can be assigned a unique timestamp.
4. System will use this timestamp to decide whether to wait or rollback transaction.
5. The transaction keeps its old time stamp if it is rollback and restarted.
6. Various deadlock prevention techniques using timestamps,

A. Wait-Die

- This is Non pre-emptive technique of deadlock prevention.
- When transaction T_i wants to hold data item, currently hold by T_j, then T_i is allowed to wait if and only if it is older than T_j otherwise T_i is rolled back (die).
- If T_1 requests for data item hold by T_2 then as TS (T_1) < TS (T_2) so T_1 should wait.

B. Wound-Wait

- This is Pre-emptive Technique of deadlock prevention.
- When T_i wants to hold data item, currently hold by T_j, then T_i is allowed to wait if and only if T_i is younger to T_j otherwise T_j is rollback (wounded).
- Consider T_1, T_2, T_3 transactions.
- If T_1 requests for data item hold by T_2 then data item is pre-empted from T_2 and given to T_1 and T_2 is rollback.
- If T_3 requests for data item hold by T_2 then T_3 need to roll back.

7. Prevention may lead to starvation of transaction, if they are rollback again and again. But both schemes wait-die and wound-wait avoid starvation by making use of timestamps.

13.9.2 Deadlock Detection and Recovery

1. Introduction

1. When system is having deadlock detection and recovery techniques, the detection is done periodically to check whether the system is in deadlock, if yes then the recovery techniques are used to resolve this deadlock.
2. This can be done with help of some deadlock detection algorithms.
3. To achieve this, system must do the following :
   a. System should maintain information about current data items allocation to transactions and requests to be satisfied.
   b. Algorithm that uses this information to detect deadlock.
   c. Recovery techniques to be applied after detection of deadlock.

2. Deadlock detection

1. Deadlock can be detected using directed graph called as **wait-for-graph.**
2. The graph G = (V,E) can be seen as V is of vertices i.e. set of transaction in execution concurrently and E is set of edges.
3. Such that edge $T_i \rightarrow T_j$ if $T_i \rightarrow T_j$ is present in graph it shows that, transaction T_j is waiting for transaction T_i to release a data item it needs.
4. If cycle is present in wait-for-graph then deadlock is present and transactions in cycle are deadlock.

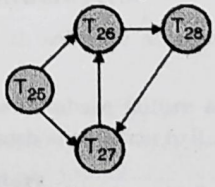

Fig. 13.9.1 : Wait-for graph with a cycle

5. To detect deadlock, system must maintain wait-for-graph and periodically invoke an algorithm that searches cycle in the graph.

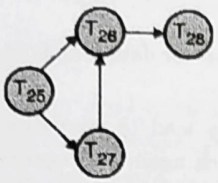

Fig. 13.9.2 : Wait-for graph with no cycle

6. If no cycle is present it mean no deadlock in system.
7. If deadlock occurs frequently, then detection algorithm should be invoked more frequently problem in this scenario is the data items locked by deadlock transaction will be unavailable until the deadlock is problem.
8. This may tend to more cycles in graph degrading capacity of granting lock requests.

3. Recovery from deadlock

- When deadlock is detected in the system, then system should be recovered from deadlock using recovery schemes.
- A most common solution is rollback one or more transaction to break the deadlock.

Methods for recover from deadlock :

a. Selection of victim

- If deadlock is detected then a transaction one or more transactions (victims) to be selected to break the deadlock.
- Transactions with minimum cost should be selected for rollbacks.
- Cost can be detected by following factors.
  - For how much time transaction has computed and how much time it needs to do.
  - How many data items it has locked ?
  - How many data items it may need ahead ?
  - Number of transaction to be rollback.

b. Rollback

- Once victims are decided then there are two ways to do so.
- **Total rollback :** The transaction is aborted and then restart it.
- **Partial rollback :** Rollback the only transaction which is needed to break the deadlock.
- But this approach needs to record some more information like state of running transactions, locks on data item held by them, and deadlock detection algorithm specifies points up to which transaction to be rollback and recovery method has to rollback.
- After sometime transaction resume; partial transaction.

c. Starvation

- It may happen every time same transaction is selected as victim and this may lead to starvation of that transaction (minimum cost transaction it selected every time).
- System should take care that every time same transaction should not be selected as victim. So it will not be starved.

Review Questions

1. What do you understand by concurrency control? Explain view serializability and conflict serializability with proper example.
2. What is concurrency ? If not controlled where it can lead to? What are the methods to control concurrency ?
3. Explain Two phase locking.
4. Explain time stamp based protocols.
5. List and explain various issues while transactions are running concurrently in DBMS.
6. Explain two phase locking protocol. Also list advantages of this protocol.

❑❑❑