Unit 6: Transaction Processing and Concurrency Control, and Recovery

Unit 6: Transaction Processing and Concurrency Control, and Recovery (8 Hours) Introduction to Transaction Processing; *(Single User vs. Multi User, Read/ Write Operations, Need for Concurrency Control: Lost Update Problem, Temporary Update (or Dirty Read) Problem, Incorrect Summary Problem, Unrepeatable Read Problem, Need for Recovery);*	**2 Hours**
Transaction and System Concepts *(Transaction States, System Log, Commit Point of Transaction)*; Desirable Properties of Transactions (*Atomicity, Consistency, Isolation, Durability)*;	**2 Hours**
Schedules, Conflicting Operations in Schedule, Characterizing Schedules based on Recoverability, Characterizing Schedules based on Seriablizility, Serial, Nonserial, Conflict Serializable Schedules, Testing for serializability of Schedule, Using Serializability for Concurrency Control;	**2 Hours**
Concurrency Control Techniques; Two-Phase Locking *(Types of Lock, Basic, Conservative, Strict, and Rigorous Two-Phase Locking, Deadlock and Starvation, Deadlock Prevention, Deadlock Detection)* and Timestamp Ordering *(Timestamp, Read Timestamp, Write Timestamp, Basic Timestamp Ordering, Strict Timestamp Ordering);*	**2 Hours**

Unit-06 Introduction to Transaction Processing Concepts and Theory

Introduction to Transaction Processing:

A transaction is a unit of program execution that accesses and possibly updates various data items. A transaction must see a consistent database. During transaction execution, the database may be inconsistent.

Examples of fund transfer:

Transaction to transfer Rs 50 from account A to account B.

1. read(A)
2. A := A-50
3. write(A)
4. read(B)
5. B:= B + 50
6. write(B)

What happens when transaction fail at step 5?

Database become inconsistence state.

Introduction to Transaction Processing:

Two main issues to deal with transaction:

1. How to recover transactions with various kind of failures of transaction such as hardware failures, power supply, system crashes etc.
2. How to control the concurrent execution of multiple transactions to maintain the database consistency.

T_1	T_2
X:=X-N	
	X:=X+M
Read(Y)	
Y:=Y+N	

Introduction to Transaction Processing:

Database System classifying according to the number of users who can use the system concurrently:

1. Single-User System - A DBMS is single-user if at most one user at a time can use the system. Single-user DBMSs are mostly restricted to personal computer systems;
2. Multiuser System - It is multiuser system if many users can use the system - and hence access the database -Concurrently. Many users can access the system concurrently. For example, an airline reservations system is used by hundreds of users and travel agents concurrently. Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems. In these systems, hundreds or thousands of users are typically operating on the database by submitting transactions concurrently to the system.

Introduction to Transaction Processing:

Multiple users can access databases and use computer systems simultaneously because of the concept of multiprogramming, which allows the operating system of the computer to execute multiple program or processes at the same time.

A single central processing unit (CPU) can only execute at most one process at a time. However, multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again.

Hence, concurrent execution of processes are of two types:

Interleaved processing: Concurrent execution of processes is interleaved in a single CPU

Parallel processing: Processes are concurrently executed in multiple CPUs.

Unit-06 Introduction to Transaction Processing Concepts and Theory

Transactions, Database Items, Read and Write Operations and DBMS Buffers:

Transactions:

A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations - these can include insertion, deletion, modification (update), or retrieval operations.

The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program; in this case, all database access operations between the two are considered as forming one transaction.

Transactions, Database Items, Read and Write Operations and DBMS Buffers:

Read and Write Operations and DBMS Buffers:

If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction; otherwise it is known as a read-write transaction.

The basic database access operations that a transaction can include are as follows:

- read_item(X). Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
- write_item(X). Writes the value of program variable X into the database item named X.

Unit-06 Introduction to Transaction Processing Concepts and Theory

Transactions, Database Items, Read and Write Operations and DBMS Buffers:

READ AND WRITE OPERATIONS - Basic unit of data transfer from the disk to the computer main memory(buffer) is one block. In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole block.

Read_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Transactions, Database Items, Read and Write Operations and DBMS Buffers:

Write_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Find Block in Disk of item X
⬇
Copy Block that holds item X from Disk into Memory
⬇
Copy item X into Memory and Update
⬇
Store back updated X value in Disk immediate or later

Unit-06 Introduction to Transaction Processing Concepts and Theory

Chances of transaction failure & inconsistent data:

1. The Lost Update Problem

This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

2. The Temporary Update (or Dirty Read) Problem

This occurs when one transaction updates a database item and then the transaction fails for some reason.
The updated item is accessed by another transaction before it is changed back to its original value.

3. The Incorrect Summary Problem

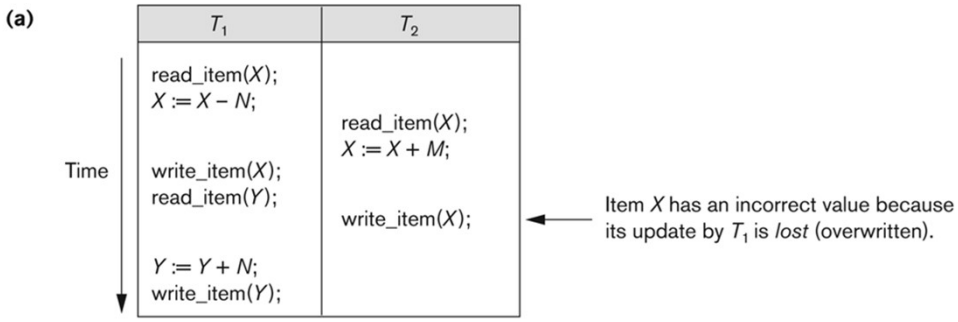
If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Chances of transaction failure & inconsistent data:

1. The Lost Update Problem

This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.



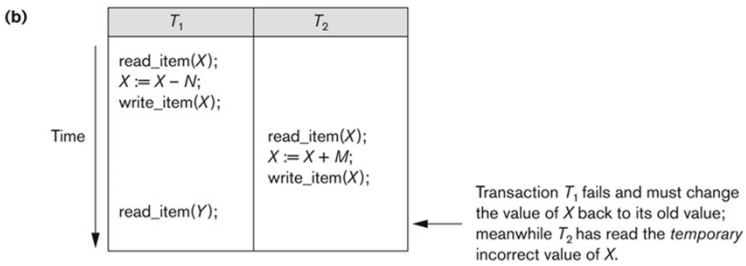
Unit-06 Introduction to Transaction Processing Concepts and Theory

Chances of transaction failure & inconsistent data:

2. The Temporary Update (or Dirty Read) Problem

- This occurs when one transaction updates a database item and then the transaction fails for some reason. (Note: if transaction fail, it revert back to original state and create inconsistent data)
- The updated item is accessed by another transaction before it is changed back to its original value.

Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.



Chances of transaction failure & inconsistent data:

3. The Incorrect Summary Problem

- If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

(c)

T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	. . .

read_item(*X*);
X := *X* − *N*;
write_item(*X*);	

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

← T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).

What causes a Transaction to fail:

1. A computer failure (system crash):

A hardware or software error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.

2. A transaction or system error:

Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

3. Local errors or exception conditions detected by the transaction:

Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal from that account, to be cancelled.

A programmed abort in the transaction causes it to fail.

What causes a Transaction to fail:

4. Concurrency control enforcement:

The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

5. Disk failure:

Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

6. Physical problems and catastrophes:

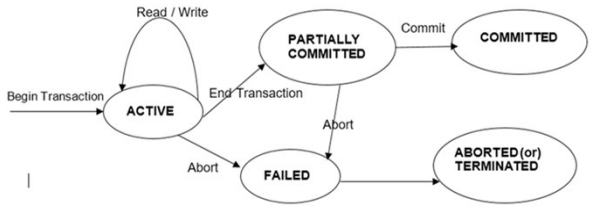
This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Transaction and system concepts:

- A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.

Therefore, the recovery manager of the DBMS needs to keep track of the following operations:

- BEGIN_TRANSACTION
- READ or WRITE
- END_TRANSACTION
- COMMIT_TRANSACTION
- ROLLBACK (or ABORT)



Introduction to Transaction Processing Concepts and Theory

Transaction and system concepts:

Therefore, the recovery manager of the DBMS needs to keep track of the following operations:

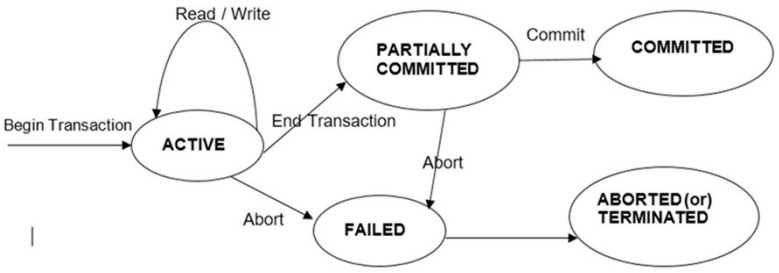
1. BEGIN_TRANSACTION - This marks the beginning of transaction execution.
2. READ or WRITE - These specify read or write operations on the database items that are executed as part of a transaction.
3. END_TRANSACTION - This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason.
4. COMMIT_TRANSACTION -This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
5. ROLLBACK (or ABORT) - This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Introduction to Transaction Processing Concepts and Theory

Transaction States:

A state transition diagram that illustrates how a transaction moves through its execution states.

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Terminated State



The System Log:

- To be able to recover from failures that affect transactions, the system maintains a log to keep track of all transaction operations that affect the values of database items, as well as other transaction information that may be needed to permit recovery from failures.
- The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.

The System Log:

The following are the types of entries (called log records) that are written to the log file and the corresponding action for each log record. In these entries, T refers to a unique transaction-id that is generated automatically by the system for each transaction and that is used to identify each transaction:

1. [start_transaction, T]. Indicates that transaction T has started execution.
2. [write_item, T, X, old_value, new_value]. Indicates that transaction T has changed the value of database item X from old_value to new_value.
3. [read_item, T, X]. Indicates that transaction T has read the value of database item X.
4. [commit, T]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [abort, T]. Indicates that transaction T has been aborted.

Commit Point of a Transaction:

- A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log. Beyond the commit point, the transaction is said to be committed, and its effect must be permanently recorded in the database. The transaction then writes a commit record [commit, T] into the log.
- If a system failure occurs, we can search back in the log for all transactions T that have written a [start_transaction, T] record into the log but have not written their [commit, T] record yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process.
- Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, so their effect on the database can be redone from the log records.

Desirable properties of transactions:

Transactions should possesses several properties, called ACID properties and they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties.

1. Atomicity: Either all operations of the transactions are properly reflected in the database or none are.
2. Consistency: Execution of a transaction in isolation preserves the consistency of the database. A transaction is consistency preserving if its completes execution takes the database from one consistent state to another.
3. Isolation: Although multiple transactions may execute concurrently, each transaction must be unaware of another concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
4. Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Introduction to Transaction Processing Concepts and Theory

Desirable properties of transactions:

Example: Transaction to transfer Rs 50 from account A to account B.

1. read(A)
2. A := A-50
3. write(A)
4. read(B)
5. B:= B + 50
6. write(B)

Consistency Requirement: The sum of A and B is unchanged by the execution of the transaction.

Atomicity Requirement: If the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.

Introduction to Transaction Processing Concepts and Theory

Desirable properties of transactions:

Durability: Once the user has been noticed that the transaction has completed (i. e transfer of the 50 has taken place), the updates to the database by the transaction must persist despite failures.

Isolation Requirement: If between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database. (the sum A + B will be less than it should be).

Concurrency Control

Concurrency Control:

Multiple transactions are allowed to run concurrently in the system.
Advantages are:

- Increased processor and disk utilization, leading to better transaction throughput: one transaction can be using the CPU while another is reading from or writing to the disk
- Reduced average response time for transactions: short transactions need not wait behind long ones.

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

Concurrency Control:

Example: Suppose a transaction T1 that transfers N(10) reservations from one flight whose number of reserved seats is stored in the database item X to another flight whose number of reserved seats is stored in the database item named Y. Another transaction T2 add value M (5) to the item X.

Concurrency Control

T_1	T_2	Remarks (i.e X =100;Y=50; N=10 and M=5)
X:=X-N		X=100-10=90
	X:=X+M	X=105
Read(Y)		Y=50
Y:=Y+N		Y=50+10 = 60

Concurrency Control:

Concurrency control schemes: is the mechanisms to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database.

So to prevent transaction from destroying the consistency of the database, we have to understand following terms:

1. Schedules
   - Serial Schedule
   - Non serial schedule
2. Conflict schedule
3. Serializability of schedule
4. Testing of serializability

Unit-06:Transaction Processing and Concurrency Control and Recovery

Schedule:

When transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from various transaction is known as a schedule (or history).

T_1	T_2
Read(X)	
X:=X-N	
	Read(X)
	X:=X+M
Wrtie(X)	
Read(Y)	
	Write(X)
Y:=Y+N	
Write(Y)	

Types of Schedule: 1. Serial Schedule
2. Non-Serial Schedule

- For a set of n transactions, there exists n! different valid serial schedules.
- Transaction processing system allow multiple transaction to run concurrently. So there will be many possible schedules for non-serial schedules.

Serial Schedule

Schedule: **X=100, N=10, Y=50, M=5**

T1	T2	
read_item(X);		100
X:=X-N;		90
write_item(X);		X= 90
read_item(Y);		50
Y:=Y+N;		**Y= 50+10=60**
write_item(Y);		
	read_item(X);	90
	X:=X+M;	X=90+5=95
	write_item(X);	**X= 95**

T1	T2	
	read_item(X);	100
	X:=X+M;	X=100+5=105
	write_item(X);	X=105
read_item(X);		105
X:=X-N;		X=105-10=95
write_item(X);		**X= 95**
read_item(Y);		50
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

Result should be X=95 and Y=60 after T1 and T2 execution

Non Serial Schedule

Schedule: **X=100, N=10, Y=50, M=5**

T1	T2
read_item(X);	
X:=X-N;	
	read_item(X);
	X:=X+M;
write_item(X);	
read_item(Y);	
	write_item(X);
Y:=Y+N;	
write_item(Y);	

T1	T2
read_item(X);	
X:=X-N;	
write_item(X);	
	read_item(X);
	X:=X+M;
	write_item(X);
read_item(Y);	
Y:=Y+N;	
write_item(Y);	

Non Serial Schedule

Schedule: X=100, N=10, Y=50, M=5

T1	T2	
read_item(X);		100
X:=X-N;		X=100-10=90
	read_item(X);	100
	X:=X+M;	X=100+5=105
write_item(X);		X=105
read_item(Y);		50
	write_item(X);	**X=105**
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

T1	T2	
read_item(X);		100
X:=X-N;		X=100-10=90
write_item(X);		X=90
	read_item(X);	90
	X:=X+M;	X=90+5=95
	write_item(X);	**X=95**
read_item(Y);		50
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

Conflict Schedule:

Two operations in a schedule are said to conflict if they satisfy all three of the following conditions;

1. They belong to different transactions
2. They access the same item X
3. At least one of the operations is a write_item(X)

Example:

T1=read(X), T2 = read(X) - T1 and T2 don't conflict
T1=read(X), T2 = write(X) - They conflict
T1=write(X), T2 = read(X) - They conflict
T1=read(X), T1 = write(X) - They don't conflict
T1=write(X), T2 = write(X) - They conflict

Serializability of Schedule:

- The schedules which gives correct answer is called serializable schedule.
- Serializability will characterize the types of schedules that are considered correct when concurrent transactions are executing. So schedules that gives correct result is called serializable schedule.

Serializability of Schedule

Schedule: X=100, N=10, Y=50, M=5

T1	T2	
read_item(X);		100
X:=X-N;		X=100-10=90
	read_item(X);	100
	X:=X+M;	X=100+5=105
write_item(X);		X=105
read_item(Y);		50
	write_item(X);	**X=105**
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

T1	T2	
read_item(X);		100
X:=X-N;		X=100-10=90
write_item(X);		X=90
	read_item(X);	90
	X:=X+M;	X=90+5=95
	write_item(X);	**X=95**
read_item(Y);		50
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

Note: We would like to determine which of the non-serial schedules always give a correct result and which may give erroneous results. The concept used to characterize schedules in this manner is that of serializability of a schedule.

Serializability of Schedule:

We would like to determine which of the non-serial schedules always give a correct result and which may give erroneous results. The concept used to characterize schedules in this manner is that of serializability of a schedule.

Unit-06:Transaction Processing and Concurrency Control and Recovery

Conflict Serializability:

Two schedules are said to be conflict equivalence if the order of any two conflicting operations is the same in both schedules.

Recall again that two operations in a schedule are said to conflict if they belong to different transactions, access the same database item, and at least one of the two operations is a write_item operation. If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on other transactions in the schedule, and hence the schedules are not conflict equivalent.

View equivalence:

- A less restrictive definition of equivalence of schedules

View serializability:

- Definition of serializability based on view equivalence.
- A schedule is view serializable if it is view equivalent to a serial schedule.

Unit-06:Transaction Processing and Concurrency Control and Recovery

View equivalence:

Two schedules are said to be view equivalent if the following three conditions hold:

1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
2. For any operation $R_i(X)$ of T_i in S, if the value of X read by the operation has been written by an operation $W_j(X)$ of T_j (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation $R_i(X)$ of T_i in S'.
3. If the operation $W_k(Y)$ of T_k is the last operation to write item Y in S, then $W_k(Y)$ of T_k must also be the last operation to write item Y in S'.

View equivalence:

The premise behind view equivalence:

1. As long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results.
2. "The view": the read operations are said to see the same view in both schedules.

TESTING OF SERIALIZABILITY OF SCHEDULE:

ALGORITHM:

1. For each transaction T_i participating in schedule S, create a node labelled T_i, in the preceding graph.
2. For each case in S where T_J executes a read_item(X) after T_i executes a write-item(X), create an edge ($T_i \rightarrow T_J$)in the precedence graph.
3. For each case in S where T_j executes a write_item (X) after T_i executes a read_Item(X), create an edge ($T_i \rightarrow T_J$) in the precedence graph.
4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_J$) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

Testing of Serializability

Schedule: X=100, N=10, Y=50, M=5

Sc= r1(X), r2(X), w1(X), r1(Y), W2(X), W1(Y)

Sd= r1(X), W1(X), r2(X), W2(X), r1(Y), W1(Y)

(c) Incorrect Schedule

T1	T2	
read_item(X);		100
X:=X-N;		X=100-10=90
	read_item(X);	100
	X:=X+M;	X=100+5=105
write_item(X);		X=105
read_item(Y);		50
	write_item(X);	**X=105**
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

(d) Correct Schedule

T1	T2	
read_item(X);		100
X:=X-N;		X=100-10=90
write_item(X);		X=90
	read_item(X);	90
	X:=X+M;	X=90+5=95
	write_item(X);	**X=95**
read_item(Y);		50
Y:=Y+N;		Y=50+10=60
write_item(Y);		**Y=60**

Testing of Serializability

Sc= r1(X), r2(X), w1(X), r1(Y), W2(X), W1(Y)

Sd= r1(X), W1(X), r2(X), W2(X), r1(Y), W1(Y)

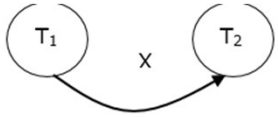
T1	T2	
read_item(X);		100
X:=X-N;		90
	read_item(X);	100
	X:=X+M;	105
write_item(X);		105
read_item(Y);		50
	write_item(X);	**X=105**
Y:=Y+N;		60
write_item(Y);		**Y=60**

T1	T2	
read_item(X);		100
X:=X-N;		90
write_item(X);		90
	read_item(X);	90
	X:=X+M;	95
	write_item(X);	**X=95**
read_item(Y);		50
Y:=Y+N;		55
write_item(Y);		**Y=55**

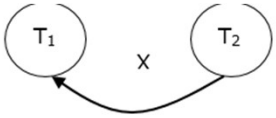
TESTING OF SERIALIZABILITY OF SCHEDULE

Sc= r1(X), r2(X), w1(X), r1(Y), W2(X), W1(Y)
Sd= r1(X), W1(X), r2(X), W2(X), r1(Y), W1(Y)

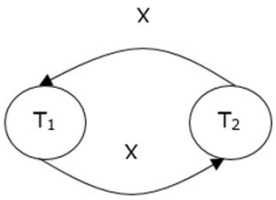
(a)



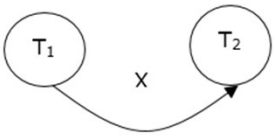
(b)



(c)



(d)



Unit-06: Concurrency Control Techniques

Unit-06: Concurrency Control Techniques

Concurrency Control Techniques; Two-Phase Locking *(Types of Lock, Basic, Conservative, Strict, and Rigorous Two-Phase Locking, Deadlock and Starvation, Deadlock Prevention, Deadlock Detection)* and Timestamp Ordering *(Timestamp, Read Timestamp, Write Timestamp, Basic Timestamp Ordering, Strict Timestamp Ordering);*	**2 Hours**

Concurrency Control Techniques

Overview of Concurrency Control:

- In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.
- So, concurrency control is a technique that are used to control/ensure the non-inteference or isolation property of concurrently executing transactions.
- Most of these techniques ensure serializability of schedules using protocols (set of rules) that guarantee serializability.

Concurrency Control Techniques

Purpose of Concurrency Control

- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Example: In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Concurrency Control Techniques

What is Concurrency Control? Why is it required in multiprogramming environment?

Concurrency Control Techniques

Various Methods of Database Concurrency Control

1. Lock based protocols
   a) Binary locks
   b) Shared/exclusive (Read/Write) locks
   c) Two phase locking
      1. Basic
      2. Conservative
      3. Strict 2PL
      4. Rigorous 2PL
2. Time stamped based protocols
   a) Basic Timestamp ordering
   b) Strict Timestamp Ordering

Concurrency Control Techniques

1. Lock based protocols

a) Binary Locks:

- Locking is an operation which secures
  (a) permission to Read
  (b) permission to Write a data item for a transaction.

Example: Lock (X): Data item X is locked in behalf of the requesting transaction.

- Unlocking is an operation which removes these permissions from the data item.

Example: Unlock (X): Data item X is made available to all other transactions.

- Lock and Unlock are Atomic operations.

Each lock can be a record with 3 fields: <data_item, Lock, Locking_transaction>

Concurrency Control Techniques

1. Lock based protocols

a) Binary Locks:

In binary locking scheme, every transaction must obey the following rules:

1. A transaction T must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T.
2. A transaction T must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T.
3. A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X.
4. A transaction T will not issue an unlock_item(X) operation unless it already holds the lock on item X.
5. At the most, only one transaction can hold the lock on a particular item. No two transaction can access the same item concurrently.

The lock manager modules of DBMS can enforce these rules.

Concurrency Control Techniques

1. Lock based protocols

Binary Locks: Advantages and Disadvantages:

Binary locking is easy to implement but it is restrictive to yield optimal concurrency conditions. DBMS will not allow two transactions to read the same item even if neither of the transaction updates the database. At most one transaction can hold a lock on a given item. So this scheme is not used for practical purposes.

Concurrency Control Techniques

1. Lock based protocols

b) Shared/exclusive (Read/Write) locks modes:

(a) shared (read) (b) exclusive (write).

- Shared mode: shared lock (X)
  - More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.
- Exclusive mode: Write lock (X)
  - Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.
- Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

Concurrency Control Techniques

1. Lock based protocols

b) Shared/exclusive (Read/Write) locks modes:

Three locking operations are

1) **Lock_R(X) ; R = shared lock**
2) **Lock_W(X) ; W= Exclusive lock**
3) **Unlock(X)**

Thus, a lock associated with an item X, Lock (X) now has 3 possible states:

1) **Read_locked**
2) **Write_locked**
3) **Unlock**

A shared lock is used when a transaction wants to read data from the database and no exclusive lock is held on that data item. It allows several read transactions to concurrently read the same data item.

Concurrency Control Techniques

1. Lock based protocols

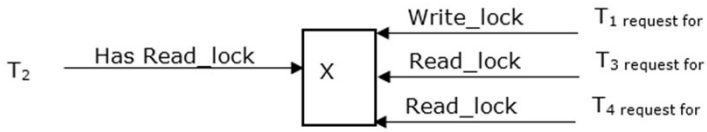
b) Shared/exclusive (Read/Write) locks modes:

- Lock Manager:
  - Managing locks on data items.
- Lock table:
  - Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Concurrency Control Techniques

Starvation (wait infinity):



Rule: A shared lock is used when a transaction wants to read data from the database and no exclusive lock is held on that data item. It allows several read transactions to concurrently read the same data item.

Problem of shared & exclusive locking is starvation, which occurs when a transaction cannot proceed for an infinity period of time while other transactions in the system continue normally. Here T_1 can't proceed.

Concurrency Control Techniques

c) Two phase locking:

Rule: A transaction is said to follow the "two phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction"

Such a transaction can be divided into two phases: an expanding or growing (first) phase, during which new locks on items can be acquired but none can be released and a shrinking (second) phase, during which existing locks can be released but no new locks can be acquired.

Concurrency Control Techniques

c) Two phase locking:

Example:

T1	T2
read_lock(Y)	read_lock(X)
read_item(Y)	read_item(X)
unlock(Y)	unlock(X)
write_lock(X)	write_lock(Y)
read_item(X)	read_item(Y)
X:= X+Y	Y:= X+Y
write_item(X)	write_item(Y)
unlock(X)	unlock(Y)

Re-scheduling T1 and T2 according as two phase locking should be as:

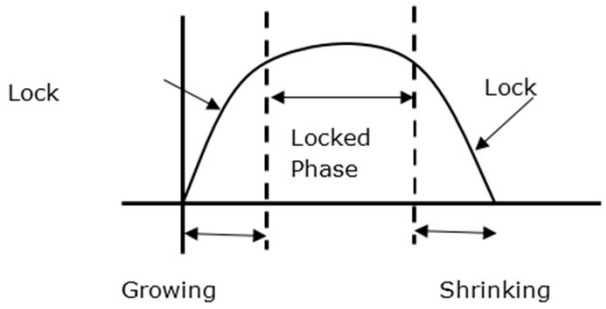
T1	T2
read_lock(Y)	read_lock(X)
read_item(Y)	read_item(X)
write_lock(X)	write_lock(Y)
unlock(Y)	unlock(X)
read_item(X)	read_item(Y)
X:= X+Y	Y:= X+Y
write_item(X)	write_item(Y)
unlock(X)	unlock(Y)

Concurrency Control Techniques

c) Two phase locking:

If lock conversion is allowed, then upgrading of locks (from read_lock to write_lock) must be done during the expanding phase, and downgrading of locks (from write_lock to read_lock) must be done in the shrinking phase.

Note: the transaction cannot acquire a new lock after it has unlocked any of its existing locked items.

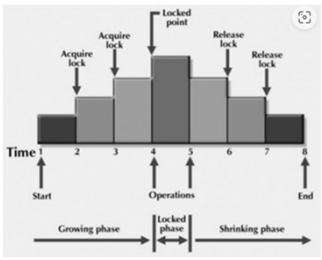


Concurrency Control Techniques

c) Two phase locking:

Two phase locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.

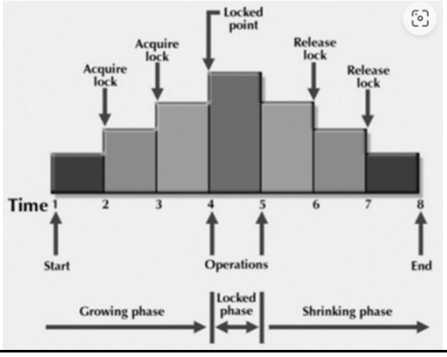


Concurrency Control Techniques

c) Two phase locking:

The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

- Growing Phase: In this phase transaction may obtain locks but may not release any locks.
- Shrinking Phase: In this phase, a transaction may release locks but not obtain any new lock



Concurrency Control Techniques

Types of Two-Phase Locking:

1. Basic
2. Conservative
3. Strict 2PL
4. Rigorous 2PL

Basic: The technique just described above is known as basic 2PL.

Conservative: A variation known as conservative 2PL (or static 2PL) requires a transaction to lock all the items it accesses before the transaction begins execution, by pre-declaring its read-set and write-set. The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that it writes. If any of the pre-declared items needed cannot be locked, the transaction does not lock any item; instead, it waits until all the items are available for locking.

However, it is difficult to use in practice because of the need to pre-declare the read-set and write set, which is not possible in some situations.

Concurrency Control Techniques

Types of Two-Phase Locking:

Strict 2PL: In this variation, a transaction T does not release any of its exclusive (write) locks until after it commits or aborts. Hence, no other transaction can read or write an item that is written by T unless T has committed, leading to a strict schedule for recoverability.

In this specific variation of transaction handling, a transaction (referred to as T) follows a unique locking strategy. In Strict 2PL, T retains all of its exclusive (write) locks until it reaches a commitment or abort decision. This means that other transactions are unable to access or modify any data item that T has written until T completes its transaction by either committing or aborting. This approach creates a rigid schedule that ensures the recoverability of data, as no conflicting actions can take place until T's final outcome is determined.

Concurrency Control Techniques

Types of Two-Phase Locking:

Rigorous 2PL: In this variation, a transaction T does not release any of its locks read and write lock (exclusive or shared) until after it commits or aborts, and so it is easier to implement than strict 2PL.

Concurrency Control Techniques

Basic, Conservative, Strict, and Rigorous Two-Phase Locking:

Notice the difference between strict and rigorous 2PL: the former holds write-locks until it commits, whereas the latter holds all locks (read and write).

Also, the difference between conservative and rigorous 2PL is that the former must lock all its items before it starts, so once the transaction starts it is in its shrinking phase; the latter does not unlock any of its items until after it terminates (by committing or aborting), so the transaction is in its expanding phase until it ends.

Concurrency Control Techniques

Deadlock:

Deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

Example, assume a set of transactions {T0, T1, T2, ...,Tn}. T0 needs a resource X to complete its task. Resource X is held by T1, and T1 is waiting for a resource Y, which is held by T2. T2 is waiting for resource Z, which is held by T0. Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

(T0 need X which is held by T1; T1 need Y which is held by T2; T2 need Z which is held by T0. All T0, T1 and T3 are in deadlock position)

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

Concurrency Control Techniques

Deadlock Prevention:

- To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyses if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.
- There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Concurrency Control Techniques

Deadlock Prevention methods:

1. Wait-Die Scheme
2. Wound-Wait Scheme
3. Deadlock Avoidance
4. Wait-for Graph

Note: Read yourself on your own interest.

Methods of Database Concurrency Control

2) Time stamped based protocols

- Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.
- The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.
- Lock-based protocols help to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

Methods of Database Concurrency Control

2) Time stamped based protocols

Example: Suppose there are there transactions T1, T2, and T3.

- T1 has entered the system at time 00:10
- T2 has entered the system at 00:20
- T3 has entered the system at 00:30
- Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

Methods of Database Concurrency Control

2) Basic Timestamp ordering protocols

- In this protocol, any conflicting read and write operations are executed in timestamp order.
- Each transaction is assigned a timestamp when it start. The idea for this scheme is to order the transactions based on their timestamps.
- If an old transaction T1 has time_stamp TS(T1), a new transaction T2 is assigned time stamp TS(T2) such that TS(T1) < TS(T2)

The protocol manages concurrent execution such that the timestamps determine the serializability order by re-ordering the conflicting operations in the schedule.

Methods of Database Concurrency Control

2) Time stamped based protocols

The protocol maintains for each data item X, two time stamp values:

1. Read timestamp R_TS(X): is the largest(latest) timestamp of any transaction that executed read(X) successfully.
2. Write timestamp W_TS(X): is the largest(latest) timestamp of any transaction that executed write(X) successfully.

Whenever some transaction T tries to issue a R_item(X) or a W_item(X) operation, the basic Timestamp ordering (TO) algorithm compares the timestamp of T i.e. TS(T) with R_TS(X) and W_TS(X) to ensure that the timestamp order of transaction execution is not violated.

Methods of Database Concurrency Control

2) Time stamped based protocols - Example

- Timestamp of any transaction: is denoted by TS(Ti) Where TS is timestamp and Ti is transaction number.

i.e. TS(T1)=100, TS(T2)=200, TS(T3)=300

T1 (Timestamp =100)	T2 (Timestamp =200)	T3 (Timestamp =300)
R(A)		
	R(A)	
		R(A)

- Read Timestamp of Data: Timestamp of last (latest) transaction number which performs the read operation on data (say "A") successfully is called Read timestamp. i.e. So Read timestamp of data "A" will be the Read timestamp of latest transaction. i.e. R_TS (A) = 300
- Write Timestamp of Data: Timestamp of last (latest) transaction number which performs the write operation on same data successfully is called Write timestamp. T2 is the latest transaction which perform the Write operation on same data "A". So write timestamp of some data (Say "A)" will be W_TS (A) = 200.

T1 (Timestamp =100)	T2 (Timestamp =200)	T3 (Timestamp =300)
W(A)		
		W(A)
	W(A)	

Methods of Database Concurrency Control

2) Time stamped based protocols - Example

T1 (Timestamp =100)	T2 (Timestamp =200)	T3 (Timestamp =300)
R(A)		R(B)
	R(A)	W(B)
R(B)	W(A)	R(A)
W(B)		

If there are multiple data items in the schedule (i.e. A, B, C…) then each data item holds its own Read and Write timestamp as given below

- R_TS(A) = 300
- W_TS(A) =200
- R_TS(B) =100
- W_TS(B) =100

Methods of Database Concurrency Control

2) Time stamped based protocols - Timestamp follow some rules to perform read or write operation.

1. When any transaction wants to perform Read(A) operation
   - If W_TS(A) > TS (Ti), then Ti Rollback //some **younger** trans. has written A
   - Else (otherwise) execute R(A) operation and SET R_TS (A) = MAX {RTS(A), TS(Ti)}
2. When a transaction needs to perform Write (A)
   - If R_TS(A) > TS (Ti), then Ti Rollback. //some **younger** trans. has read A
   - If W_TS(A) > TS (Ti), then Ti Rollback. //some **younger** trans. has written A
   - Else (otherwise) execute W(A) operation and SET W_TS (A) = TS(Ti).

   Where "A" is some data.

Methods of Database Concurrency Control

2) Time stamped based protocols - Advantages of Timestamp Protocol

Timestamp protocol always ensure Serializabilty and Deadlock removal because Transaction with smaller timestamp (TS) will came first in execution sequence than Transaction with higher TS.

Disadvantages of Timestamp Protocol

- Schedule may not be recoverable.
- Schedule may not be cascading free.

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

Solution: Draw the following table

In above table A,B,C are data values. And Read and Write timestamp values are given "0". As in the example table, time0 to time7 are given, let discuss one by one all.

—	A	B	C
RTS	0	0	0
WTS	0	0	0

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	0	0	0
WTS	0	0	0

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 1, the transaction 1 wants to perform read operation on data "A" then according to **Rule No 01,**

- WTS(A) > TS(T1) = 0>100 // condition false
- Go to else part and **SET** RTS (A) = MAX {RTS(A), TS(T1)} So,
- RTS (A) = MAX{0,100}= 100.

So, finally RTS(A) is updated with 100

Updated table will be appear as following,

—	A	B	C
RTS	**100**	0	0
WTS	0	0	0

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	0	0	0
WTS	0	0	0

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 2, the transaction 2 wants to perform read operation on data "B" then according to Rule No 01,

- WTS(B) > TS(T2) = 0>200 // condition false
- Go to else part and SET RTS (B) = MAX {RTS(B), TS(T2)} So,
- RTS (B) = MAX{0,200} = 200.

So, finally RTS(B) is updated with 200

Updated table will be appear as following,

—	A	B	C
RTS	100	200	0
WTS	0	0	0

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	100	200	0
WTS	0	0	0

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 3, the transaction 1 wants to perform write operation on data "C" then according to Rule No 02,

- RTS(C) > TS(T1) = 0>100 // condition false
- Go to second condition, WTS(C) > TS(T1) = 0>100 // again condition false
- Go to else part and SET WTS (C) = TS(T1) So,
- WTS (C) = TS(T1) = 100.

So, finally WTS(C) is updated with 100

Updated table will be appear as following,

—	A	B	C
RTS	100	200	0
WTS	0	0	100

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	100	200	0
WTS	0	0	100

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 4, the transaction 3 wants to perform read operation on data "B" then according to Rule No 01,

- WTS(B) > TS(T3) = 0>300 // condition false
- Go to else part and SET RTS (B) = MAX {RTS(B), TS(T3)} So,
- RTS (B) = MAX{200,300} = 300.

So, finally RTS(B) replace 200 and updated with 300.

Updated table will be appear as following,

—	A	B	C
RTS	100	300	0
WTS	0	0	100

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	100	300	0
WTS	0	0	100

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 5, the transaction T1 wants to perform read operation on data "C" then according to Rule No 01,

- WTS(C) > TS(T1) = 100>100 // condition false
- Go to else part and SET RTS (C) = MAX {RTS(C), TS(T1)} So,
- RTS (A) = MAX{0,100}= 100.
- So, finally RTS(C) is updated with 100

Updated table will be appear as following,

—	A	B	C
RTS	100	300	100
WTS	0	0	100

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	100	300	100
WTS	0	0	100

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 6, the transaction 2 wants to perform write operation on data **"B"** then according to **Rule No 02,** RTS(B) > TS(T2) = 300>200 // condition True

According to Rule 2: if condition true then Rollback T2.

- When T2 rollback then it never be resume, it will restart with new timestamp value. Keep in mind T2 restart after completion of all running transactions, so in this example T2 will restart after completion of T3.

It happens due to conflict where the older transaction (T2) want to perform write operation on data "B" but younger transaction (T3) already Read the same data "B"

- Table will remain the same

—	A	B	C
RTS	100	300	100
WTS	0	0	100

Methods of Database Concurrency Control

Example of Timestamp ordering Protocol:

—	A	B	C
RTS	100	300	100
WTS	0	0	100

Time of Transaction	T1 (Timestamp = 100)	T2 (Timestamp = 200)	T3 (Timestamp = 300)
Time 1	R (A)		
Time 2		R (B)	
Time 3	W (C)		
Time 4			R (B)
Time 5	R (C)		
Time 6		W (B)	
Time 7			W (A)

At time 7, the transaction 3 wants to perform write operation on data "A" then according to Rule No 02,

- RTS(A) > TS(T3) = 100>300 // condition false
- Go to second condition, WTS(A) > TS(T3) = 100>300 // again condition false
- Go to else part and SET WTS (A) = TS(T3) So,
- WTS (A) = 300.

So, finally WTS(A) is updated with 300
Updated table will be appear as following,

—	A	B	C
RTS	100	300	100
WTS	300	0	100

Methods of Database Concurrency Control

Strict Timestamp Ordering:

In Basic Timestamp Order(BTO), does not wait for commit or abort but in strict, it wait for commit or abort.

Each active database item will have the following two timestamps:

- R_TS(X): the sequence number of the youngest transaction that read item X.
- W_TS(X): the sequence number of the youngest transaction that wrote item X.

Every transaction is assigned with a unique integer sequence number. TS(T9) is the timestamp of T9 which is the sequence number of the transaction = 9

Consider two transactions T5 and T8, where T8 has started after T5. Hence, T8 is younger to T5. Also, TS(T8) > TS(T5).

Methods of Database Concurrency Control

Strict Timestamp Ordering:

T9 requests write_item(X):

If ((R_TS(X) > TS(T9)) OR (W_TS(X) > TS(T9)))
{
T9 will abort; //some younger trans. has read/written X
}
Else
{
T9 will wait till the youngest transaction T (W_TS(X)) that has written X, is committed/aborted.
T9 performs write_item(X);
W_TS(X) = TS(T9);
}

Note: in BTO, does not wait for commit or abort but in strict, it wait for commit or abort.

Methods of Database Concurrency Control

Strict Timestamp Ordering:

T9 requests read_item(X):

If (W_TS(X) > TS(T9))

{

T9 will abort; //some younger trans. has written X

}

Else

{

T9 will wait till the youngest transaction T (W_TS(X)) that has written X, is committed/aborted.

T9 performs read_item(X);

read_TS(X) = max(read_TS(X),TS(T9));

}

Note: in BTO, does not wait for commit or abort but in strict, it wait for commit or abort.

Methods of Database Concurrency Control

Strict Timestamp Ordering:

A variation of Basic TO is called **Strict TO** ensures that the schedules are both Strict and Conflict Serializable. In this variation, a Transaction T that issues a R_item(X) or W_item(X) such that TS(T) > W_TS(X) has its read or write operation delayed until the Transaction **T**' that wrote the values of X has committed or aborted.