DBMS

(DATABASE MANAGEMENT SYSTEM)

LESSON 7

Transactions & concurrency control:

What is Transaction?

A set of logically related operations is known as a transaction. The main operations of a transaction are:

Read(A): Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in main memory.

Write (A): Write operation Write(A) or W(A) writes the value back to the database from the buffer.

Let us take a debit transaction from an account which consists of following operations:

1. R(A);
2. A=A-1000;
3. W(A);

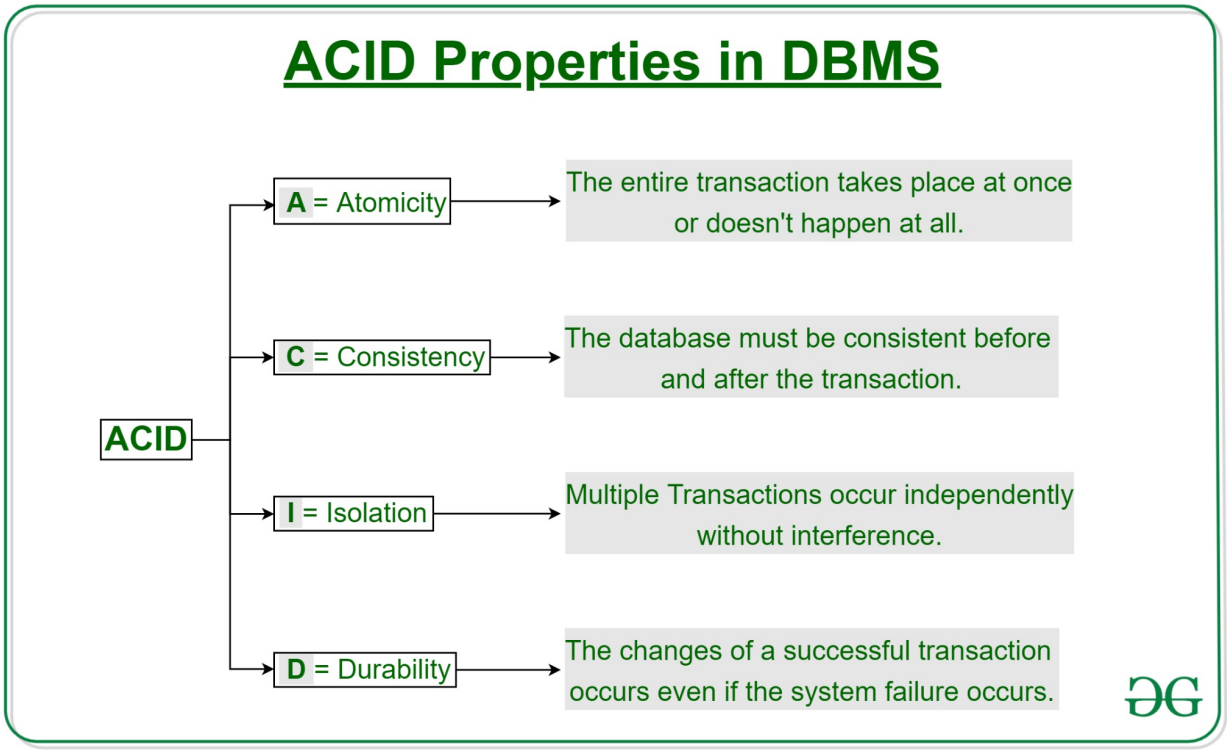
Assume A's value before starting of transaction is 5000.

- The first operation reads the value of A from database and stores it in a buffer.
- Second operation will decrease its value by 1000. So buffer will contain 4000.

- Third operation will write the value from buffer to database. So A's final value will be 4000.

ACID Properties in DBMS:

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.



Atomicity

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.
—**Abort**: If a transaction aborts, changes made to database are not visible.
—**Commit**: If a transaction commits, changes made are visible.
Atomicity is also known as the 'All or nothing rule'.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

Consistency

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,

The total amount before and after the transaction must be maintained.

Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs** = **400 + 300 = 700**.

Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write (Y)	Read (X) Read (Y) Z: = X + Y Write (Z)

Let **X**= 500, **Y** = 500.

Consider two transactions **T** and **T''**.

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result , interleaving of operations takes place due to which **T''** reads correct value of **X** but incorrect value of **Y** and sum computed by

T'': (X+Y = 50, 000+500=50, 500)

is thus not consistent with the sum at end of transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).

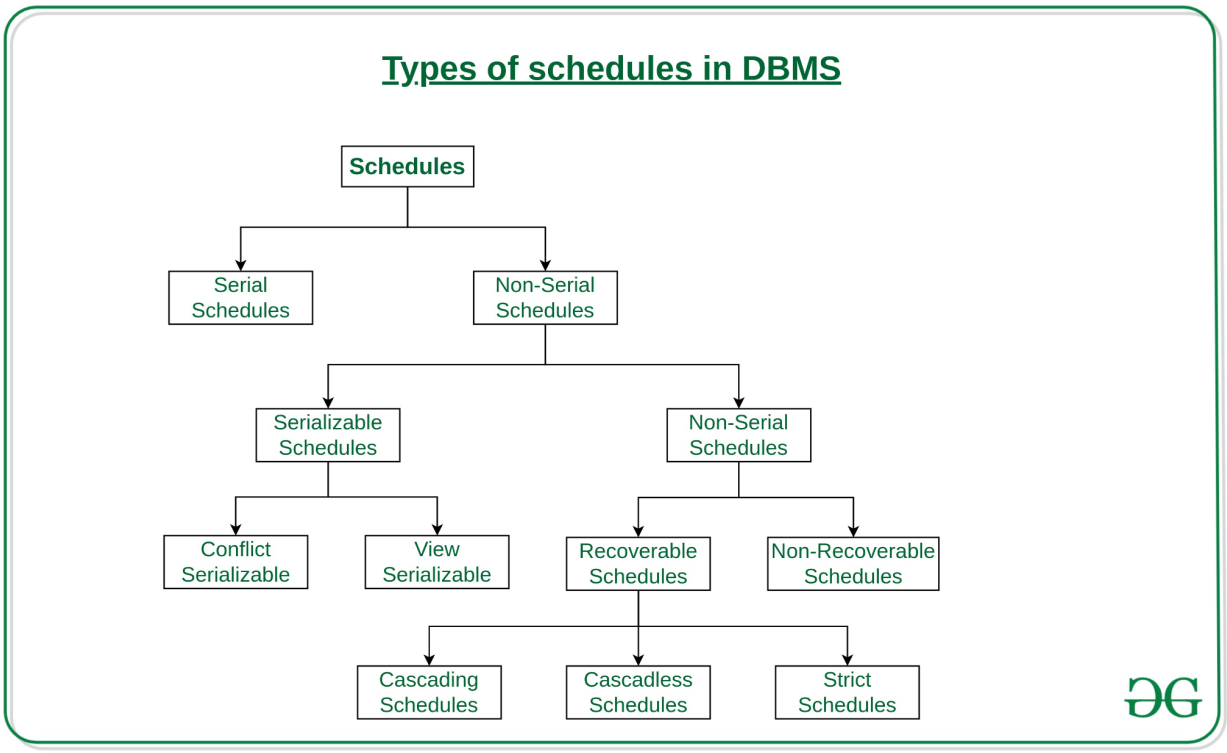
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability:
This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Types of Schedules in DBMS:

Schedule, as the name suggests, is a process of lining the transactions and executing them one by one. When there are multiple transactions that are

running in a concurrent manner and the order of operation is needed to be set so that the operations do not overlap each other, Scheduling is brought into play and the transactions are timed accordingly.



1.Serial Schedules:

Schedules in which the transactions are executed non-interleaved, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules.

Example: Consider the following schedule involving two transactions T_1 and T_2.

where R(A) denotes that a read operation is performed on some data item 'A'
This is a serial schedule since the transactions perform serially in the order $T_1 \rightarrow T_2$

2.Non-Serial Schedule:

This is a type of Scheduling where the operations of multiple transactions are interleaved. This might lead to a rise in the concurrency problem. The transactions are executed in a non-serial manner, keeping the end result correct and same as the serial schedule. Unlike the serial schedule where one transaction must wait for another to complete all its operation, in the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete. This sort of schedule does not provide any benefit of the concurrent transaction. It can be of two types namely, Serializable and Non-Serializable Schedule.
The Non-Serial Schedule can be divided further into Serializable and Non-Serializable.

1. **Serializable:**

   This is used to maintain the consistency of the database. It is

mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not. On the other hand, a serial schedule does not need the serializability because it follows a transaction only when the previous transaction is complete. The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions. Since concurrency is allowed in this case thus, multiple transactions can execute concurrently. A serializable schedule helps in improving both resource utilization and CPU throughput. These are of two types:

1. Conflict Serializable:

   A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:

   - They belong to different transactions
   - They operate on the same data item
   - At Least one of them is a write operation

2. View Serializable:

   A Schedule is called view serializable if it is view equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability

contains blind writes, then the view serializable does not conflict serializable.

2. **Non-Serializable:**

   The non-serializable schedule is divided into two types, Recoverable and Non-recoverable Schedule.

   1. Recoverable Schedule:

      Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.

      Example – Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
W(A)	
	W(A)
	R(A)
commit	
	commit

This is a recoverable schedule since T_1 commits before T_2, that makes the value read by T_2 correct.

There can be three types of recoverable schedule:

1. **Cascading Schedule:**

   Also called Avoids cascading aborts/rollbacks (ACA). When there is a failure in one transaction and this leads to the rolling back or aborting other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort. Example:

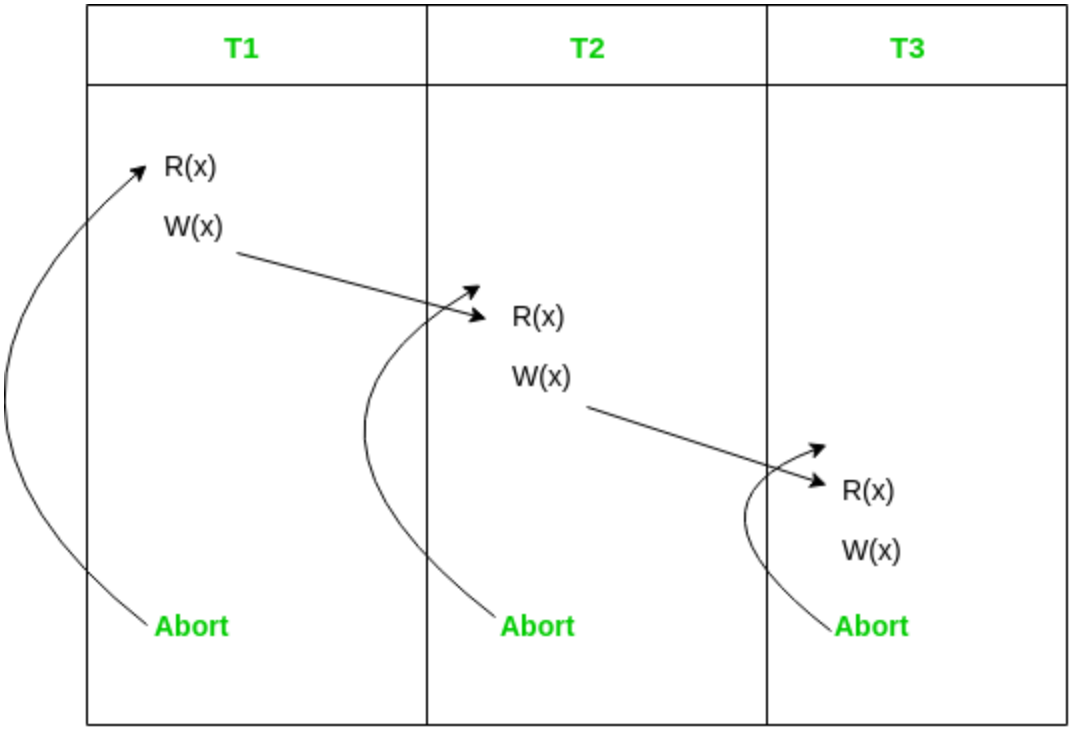


Figure - Cascading Abort

Cascadeless Schedule:

Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

In other words, if some transaction T_j wants to read value updated or written by some other transaction T_i, then the commit of T_j must read it after the commit of T_i.

Example: Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
W(A)	
	W(A)
commit	
	R(A)
	commit

This schedule is cascadeless. Since the updated value of **A** is read by T_2 only after the updating transaction i.e. T_1 commits.

Strict Schedule: A schedule is strict if for any two transactions T_i, T_j, if a write operation of T_i precedes a conflicting operation of T_j (either read or write), then the commit or abort event of T_i also precedes that conflicting operation of T_j.

In other words, T_j can read or write updated or written value of T_i only after T_i commits/aborts.

Example: Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

This is a strict schedule since T_2 reads and writes A which is written by T_1 only after the commit of T_1.

2.Non-Recoverable Schedule:

Example: Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
W(A)	
	W(A)
	R(A)
	commit
abort	

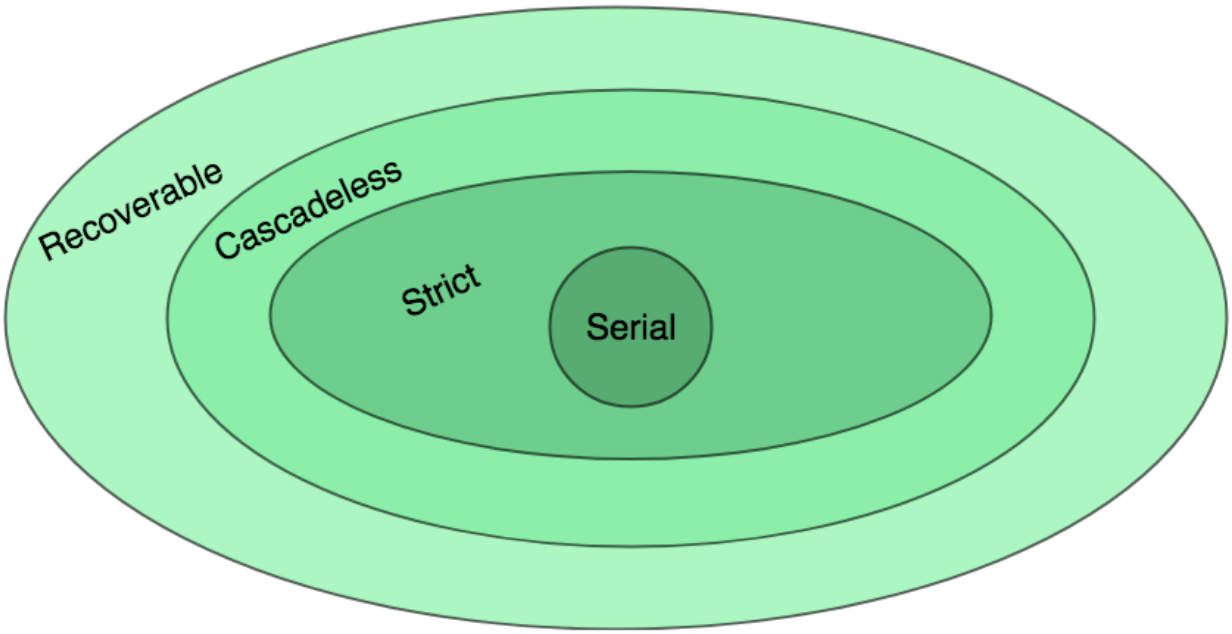
T_2 read the value of A written by T_1, and committed. T_1 later aborted, therefore the value read by T_2 is wrong, but since T_2 committed, this schedule is **non-recoverable**.

Note – It can be seen that:

1. Cascadeless schedules are stricter than recoverable schedules or are a subset of recoverable schedules.
2. Strict schedules are stricter than cascadeless schedules or are a subset of cascadeless schedules.

3. Serial schedules satisfy constraints of all recoverable, cascadeless and strict schedules and hence is a subset of strict schedules.

The relation between various types of schedules can be depicted as:



Example: Consider the following schedule:

S:R1(A), W2(A), Commit2, W1(A), W3(A), Commit3, Commit1

Which of the following is true?

(A) The schedule is view serializable schedule and strict recoverable schedule

(B) The schedule is non-serializable schedule and strict recoverable schedule

(C) The schedule is non-serializable schedule and is not strict recoverable schedule.

(D) The Schedule is serializable schedule and is not strict recoverable schedule

SOLUTION:

First of all, it is a view serializable schedule as it has view equal serial schedule $T_1 \rightarrow T_2 \rightarrow T_3$ which satisfies the initial and updated reads and final write on variable A which is required for view serializability

option (D) is correct.

Conflict Serializability in DBMS:

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

Conflicting Operations

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.

2. They have the same data item.
3. They contain at least one write operation.

Example:

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

1. T1: Read(A) T2: Read(A)

T1	T2
Read(A)	
	Read(A)

Schedule S1

Swapped

T1	T2
	Read(A)
Read(A)	

Schedule S2

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

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

Conflict Equivalent

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

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

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

Example:

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

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

T1	T2

Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Since, S1 is conflict serializable.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Result Equivalence schedule:

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

PROBLEMS with conflict execution:

https://www.geeksforgeeks.org/dirty-read-in-sql/

Concurrency-control protocols : allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and maybe even cascadeless.

These protocols do not examine the precedence graph as it is being created, instead a protocol imposes a discipline that avoids non-seralizable schedules.

Different concurrency control protocols provide different advantages between the amount of concurrency they allow and the amount of overhead that they impose.

We'll be learning some protocols which are important for GATE CS. Questions from this topic is frequently asked and it's recommended to learn this concept. (At the end of this series of articles I'll try to list all theoretical aspects of this concept for students to revise quickly and they may find the material in one place.) Now, let's get going:

Different categories of protocols:

- **Lock Based Protocol**
    - Basic 2-PL
    - Conservative 2-PL
    - Strict 2-PL
    - Rigorous 2-PL
- **Graph Based Protocol**
- **Time-Stamp Ordering Protocol**
- **Multiple Granularity Protocol**
- **Multi-version Protocol**

Lock Based Protocols –

A lock is a variable associated with a data item that describes a status of data item with respect to possible operation that can be applied to it. They synchronize the access by concurrent transactions to the database items. It is required in this protocol that all the data items must be accessed in a mutually exclusive manner. Let me introduce you to two common locks which are used and some terminology followed in this protocol.

1. **Shared Lock (S):** also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.
2. **Exclusive Lock (X):** Data item can be both read as well as written.This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction.

Lock Compatibility Matrix –

	S	X
S	✔	✗
X	✗	✗

Graph Based Concurrency Control Protocol in DBMS:

https://www.geeksforgeeks.org/graph-based-concurrency-control-protocol-in-dbms/

Time stamp based

protocol:https://www.geeksforgeeks.org/dbms-concurrency-control-protocols-timestamp-ordering-protocols/

https://www.geeksforgeeks.org/dbms-concurrency-control-protocol-thomas-write-rule/