**DBMS Serializability**

When there is a possibility that the database may be left in an inconsistent state. Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.

**What is a serializable schedule?**

A serializable schedule always leaves the database in consistent state. A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution. However a non-serial schedule needs to be checked for Serializability.

A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions. A serial schedule doesn’t allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

**Types of Serializability**

There are two types of Serializability.

1. [Conflict Serializability](https://beginnersbook.com/2018/12/dbms-conflict-serializability/)

2. [View Serializability](https://beginnersbook.com/2018/12/dbms-view-serializability/)

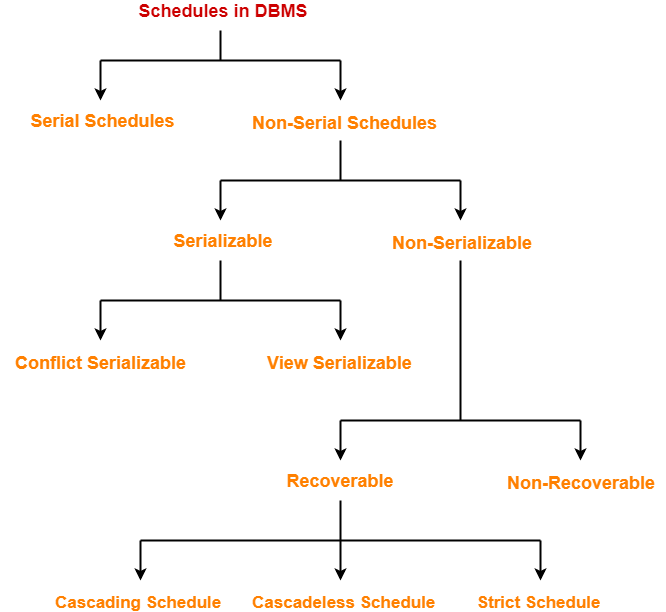
**Schedules in DBMS-**

Before you go through this article, make sure that you have gone through the previous article on [**Schedules in DBMS**](https://www.gatevidyalay.com/schedules-in-dbms/).

We have discussed-

* A schedule is the order in which the operations of multiple transactions appear for execution.
* Serial schedules are always consistent.
* Non-serial schedules are not always consistent.

In DBMS, schedules may be classified as-



In this article, we will discuss about Serializability in DBMS.

**Serializability in DBMS-**

* Some non-serial schedules may lead to inconsistency of the database.
* Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

**Serializable Schedules-**

 If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a **serializable schedule**.

**Characteristics-**

 Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

* Consistent
* Recoverable
* Cascadeless
* Strict

**Serial Schedules Vs Serializable Schedules-**

Serial Schedules Serializable Schedules

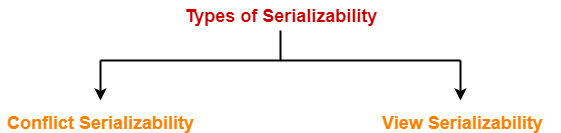
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other. Concurrency is allowed. Thus, multiple transactions can execute concurrently.

Serial schedules lead to less resource utilization and CPU throughput. Serializable schedules improve both resource utilization and CPU throughput.

Serial Schedules are less efficient as compared to serializable schedules.(due to above reason) Serializable Schedules are always better than serial schedules.(due to above reason)

**Types of Serializability-**

Serializability is mainly of two types-



1. Conflict Serializability
2. View Serializability

**Conflict Serializability-**

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

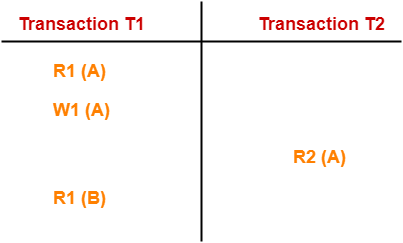
**Conflicting Operations-**

Two operations are called as **conflicting operations** if all the following conditions hold true for them-

* Both the operations belong to different transactions
* Both the operations are on the same data item
* At least one of the two operations is a write operation

**Example-**

Consider the following schedule-



In this schedule,

* W1 (A) and R2 (A) are called as conflicting operations.
* This is because all the above conditions hold true for them.

**Checking Whether a Schedule is Conflict Serializable Or Not-**

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

**Step-01:**

 Find and list all the conflicting operations.

**Step-02:**

 Start creating a precedence graph by drawing one node for each transaction.

**Step-03:**

* Draw an edge for each conflict pair such that if Xi (V) and Yj (V) forms a conflict pair then draw an edge from Ti to Tj.
* This ensures that Ti gets executed before Tj.

**Step-04:**

* Check if there is any cycle formed in the graph.
* If there is no cycle found, then the schedule is conflict serializable otherwise not.

**NOTE-**

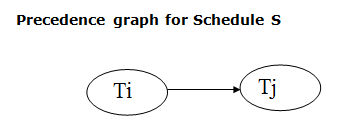
* By performing the Topological Sort of the Directed Acyclic graph so obtained, the corresponding serial schedule(s) can be found.
* Such schedules can be more than 1.

**Testing of Serializability**

Serialization Graph is used to test the Serializability of a schedule.

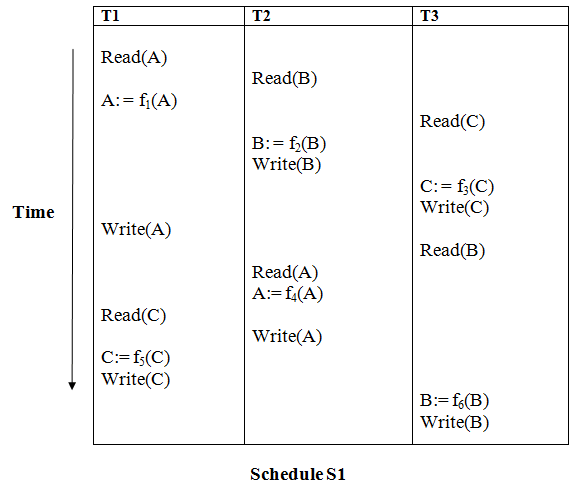
Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).



* If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
* If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

**For example:**



**Explanation:**

**Read(A):** In T1, no subsequent writes to A, so no new edges

**Read(B):** In T2, no subsequent writes to B, so no new edges

**Read(C):** In T3, no subsequent writes to C, so no new edges

**Write(B):** B is subsequently read by T3, so add edge T2 → T3

**Write(C):** C is subsequently read by T1, so add edge T3 → T1

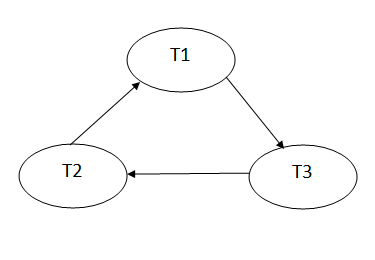
**Write(A):** A is subsequently read by T2, so add edge T1 → T2

**Write(A):** In T2, no subsequent reads to A, so no new edges

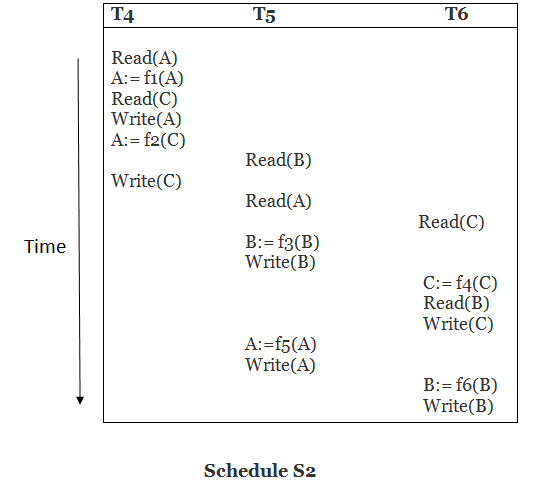
**Write(C):** In T1, no subsequent reads to C, so no new edges

**Write(B):** In T3, no subsequent reads to B, so no new edges

**Precedence graph for schedule S1:**



The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.



**Explanation:**

**Read(A):** In T4,no subsequent writes to A, so no new edges

**Read(C):** In T4, no subsequent writes to C, so no new edges

**Write(A):** A is subsequently read by T5, so add edge T4 → T5

**Read(B):** In T5,no subsequent writes to B, so no new edges

**Write(C):** C is subsequently read by T6, so add edge T4 → T6

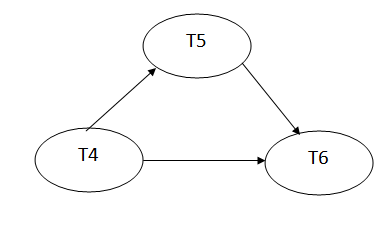
**Write(B):** A is subsequently read by T6, so add edge T5 → T6

**Write(C):** In T6, no subsequent reads to C, so no new edges

**Write(A):** In T5, no subsequent reads to A, so no new edges

**Write(B):** In T6, no subsequent reads to B, so no new edges

**Precedence graph for schedule S2:**



The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

**OTHER REFRENCES**

* [DBMS Testing of Serializability - javatpoint](https://www.javatpoint.com/dbms-testing-of-serializability)
* [Serializability in DBMS | Conflict Serializability | Gate Vidyalay](https://www.gatevidyalay.com/serializability-in-dbms-conflict-serializability/)
* [DBMS Serializability (beginnersbook.com)](https://beginnersbook.com/2018/12/dbms-serializability/)

**SUGGESTED BOOK REFERENCES**

* Ramez Elmasri and Shamkant B. Navathe,“Fundamentals of Database System”, The Benjamin / Cummings Publishing Co.
* Korth and Silberschatz Abraham, “DatabaseSystem Concepts”, McGraw Hall.
* Pratt,”DBMS”, Cengage Learning.