# SERIALIZABILITY

**SCHEDULE:**
A schedule is a list of transactions.

**Serial Schedule:**

- Serial schedule defines each transaction is executed consecutively without any interference from other transactions.

**Non- serial Schedule:**

- Non-serial schedule defines the operations from a group of concurrent transactions that are interleaved.
- In non-serial schedule, if the schedule is not proper, then the problems can arise like multiple update, uncommitted dependency and incorrect analysis.

**SERIALIZABILITY:**

- Serializability is a concurrency scheme where the concurrent transaction is equivalent to one that executes the transactions serially.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database
- The main objective of serializability is to find non-serial schedules that allow transactions to execute concurrently without interference and produce a database state that could be produced by a serial execution.

**Serializable Schedule:**

If the schedule of concurrent transaction can be converted into an equivalent serial schedule, then it is called as serializable schedule.

**Types of Serializability:**

Serializability is mainly of two types-

1. Conflict Serializability
2. View Serializability

**CONFLICT SERIALIZABILITY:**

Conflict serializability defines two instructions of two different transactions accessing the same data item to perform a read/write operation.

It deals with detecting the instructions that are conflicting in any way and specifying the order in which the instructions should execute in case there is any conflict.

A conflict serializability arises when one of the instruction is a write operation.

**Rules for Conflict Serializability:**

The following rules are important in Conflict Serializability,

1. If two transactions are both read operation, then they are not in conflict.

2. If one transaction wants to perform a read operation and other transaction wants to perform a write operation, then they are in conflict and cannot be swapped.

3. If both the transactions are for write operation, then they are in conflict, but can be allowed to take place in any order, because the transactions do not read the value updated by each other.

**What are 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 same data item

**Conflict Pairs:**

| | |
|---|---|
| R(A) | |
| | W(A) |

| | |
|---|---|
| W(A) | |
| | R(A) |

| | |
|---|---|
| W(A) | |
| | W(A) |

**Note:** The order of the Conflict pairs cannot be swapped.

**Non Conflict Pairs:**

| | |
|---|---|
| R(A) | |
| | R(A) |

| | |
|---|---|
| R(A) | |
| | R(B) |

| | |
|---|---|
| W(A) | |
| | W(B) |

| | |
|---|---|
| R(A) | |
| | W(B) |

**Note:** The order of the Non Conflict pairs can be swapped.

**Conflict serializable schedule:**

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

**Testing for Conflict Serializability:**

**Precedence Graph** or **Serialization Graph** is used commonly to test Conflict Serializability of a schedule.

It is a directed Graph (V, E) consisting of a set of nodes V = {T1, T2, T3…….Tn} and a set of directed edges E = {e1, e2, e3……em}.

The graph contains one node for each Transaction Ti. 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 (A) before Tj executes read (A).
2. Create a node Ti → Tj if Ti executes read (A) before Tj executes write (A).
3. Create a node Ti → Tj if Ti executes write (A) before Tj executes write (A).

**Note:**

- If there are **no cycles** in the graph G, then the given schedule is **conflict-serializable**.
- If there is a **cycle** in the graph G, then the given schedule is **non-conflict-serializable**.

**Example:**

Find whether the given schedule is conflict serializable or not.

S1: R1(X), R2(Y), R2(Y), W2(X), W3(Y), R1(X)

**Step 1:**

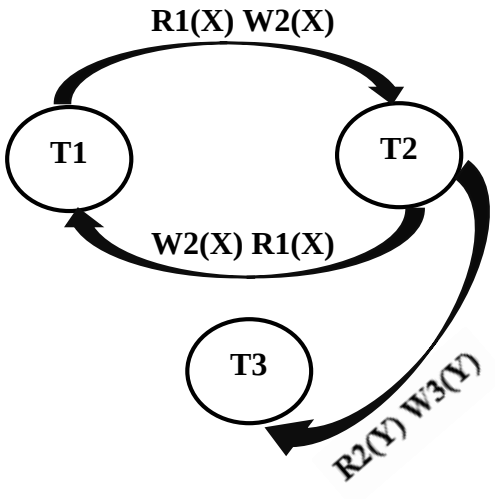
Spread it to three different transactions, it will be:

| T1 | T2 | T3 |
|---|---|---|
| R1(X) | | |
| | R2(Y) | |
| | R2(Y) | |
| | W2(X) | |
| | | W3(Y) |
| R1(X) | | |

**Step 2:**

- Check if there is a Tx that reads an item after a different Tx writes it.
  We have T1 that reads X after T2 writes it, so draw arrow from T2 -> T1
- Check if there is a Tx that writes an item after a different Tx reads it.
  We have T2 that writes X after T1 reads it, so draw arrow from T1 -> T2
  We also have T3 that writes Y after T2 reads it, so draw arrow from T2 -> T3
- Check if there is a Tx that writes an item after a different $T_X$ writes it.
  Not in this case

**So, the final graph is,**




There is a cycle between T1 and T2, so the graph is cyclic, and therefore it is not conflict serializable.

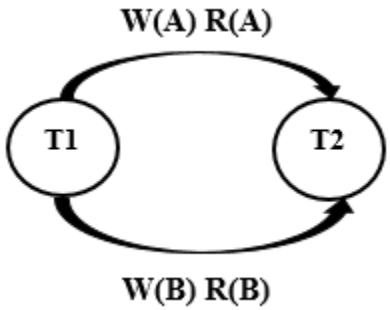
**PROBLEMS TO CONVERT A SCHEDULE INTO SERIAL SCHEDULE:**

**Example 1:**

Consider the following schedule and convert it into a serial schedule.

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| R(B) | |
| W(B) | |
| | R(B) |

**Precedence Graph:**




There is no cycle between T1 and T2, so the graph is acyclic, and therefore the given schedule is conflict serializable.

**Step 1:**

**Identifying the Conflict Pairs:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| R(B) | |
| W(B) | |
| | R(B) |

**Step 2:**

**Swapping the order of Non Conflict Pairs**

**Step 2.1:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| | R(A) |
| W(B) | |
| | R(B) |

**Step 2.2:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| W(B) | |
| | R(A) |
| | R(B) |

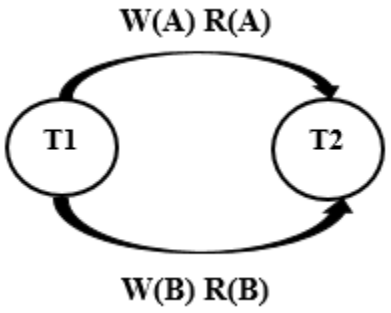
Thus the given schedule is converted into a serial schedule and it is conflict serializable.

**Example 2:**

Consider the following schedule and convert it into a serializable schedule.

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| R(B) | |
| W(B) | |
| | R(B) |
| | W(B) |

**Precedence Graph:**



There is no cycle between T1 and T2, so the graph is acyclic, and therefore the given schedule is conflict serializable.

**Step 1:**

**Identifying the Conflict Pairs:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| R(B) | |
| W(B) | |
| | R(B) |
| | W(B) |

**Step 2:**

**Swapping the order of Non Conflict Pairs**

**Step 2.1:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| R(B) | |
| | W(A) |
| W(B) | |
| | R(B) |
| | W(B) |

**Step 2.2:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| | R(A) |
| | W(A) |
| W(B) | |
| | R(B) |
| | W(B) |

**Step 2.3:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| | R(A) |
| W(B) | |
| | W(A) |
| | R(B) |
| | W(B) |

**Step 2.4:**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| W(B) | |
| | R(A) |
| | W(A) |
| | R(B) |
| | W(B) |

Thus the given schedule is converted into a serial schedule and it is conflict serializable.

**VIEW SERIALIZABILITY:**

A schedule is 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 is 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:

**Initial Read**

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

**Example:**

| T1 | T2 |
|---|---|
| R(A) | |
| | W(A) |

Schedule S1

| T1 | T2 |
|---|---|
| | W(A) |
| R(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.

**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.

**Example 1:**

| T1 | T2 | T3 |
|---|---|---|
| W(A) | | |
| | W(A) | |
| | | R(A) |

Schedule S1

| T1 | T2 | T3 |
|---|---|---|
| | W(A) | |
| W(A) | | |
| | | R(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.

**Example 2:**

| T1 | T2 | T3 |
|---|---|---|
| W(A) | | |
| | W(A) | |
| | | R(A) |

Schedule S1

| T1 | T2 | T3 |
|---|---|---|
| W(A) | | |
| | W(A) | |
| | | R(A) |

Schedule S2

Above two schedules are view equal because in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T2.

**Final Write**

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

**Example:**

| T1 | T2 | T3 |
|---|---|---|
| W(A) | | |
| | R(A) | |
| | | W(A) |

Schedule S1

| T1 | T2 | T3 |
|---|---|---|
| | R(A) | |
| W(A) | | |
| | | W(A) |

Schedule S2

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

**To Check for View Serializability:**

**Find whether the given schedule is view serializable or not.**

**Example:**

| T1 | T2 | T3 |
|---|---|---|
| R(A) | | |
| | W(A) | |
| W(A) | | |
| | | W(A) |

Schedule S

To check for view serializability of a schedule, you must create all possible combinations of the Transactions. As we have three transactions (=3! = 6), then we need to check for these combinations:

S1 < T1, T2, T3 >

S2 < T1, T3, T2 >

S3 < T2, T1, T3 >

S4 < T2, T3, T1 >

S5 < T3, T1, T2 >

S6 < T3, T2, T1 >

Consider the Schedule S1,

| T1 | T2 | T3 |
|---|---|---|
| R(A) | | |
| W(A) | | |
| | W(A) | |
| | | W(A) |

**Schedule S1**

| T1 | T2 | T3 |
|---|---|---|
| R(A) | | |
| | W(A) | |
| W(A) | | |
| | | W(A) |

**Schedule S**

Step 1: **Initial Read**

Initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 2: **Updated Read**

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 3: **Final Write**

Final write operation in S is done by T3 and in S1, it is also done by T3.

All the three conditions are satisfied. So, S and S1 are view Equivalent.

The first schedule S1 satisfies the all three conditions so, we don’t need to check other schedules.

**Hence, the view equivalent serial schedule is:**

T1 → T2 → T3