# Paper Name: Database Management Systems

# Topic: Serializability

# Paper Code: BCA CC410

# Semester IV


## By Ms. Manisha Prasad

## Head, Assistant Professor,

## Department of Computer Science

## Patna Women's College

E – mail : manisha_prasad@yahoo.com

## Serializability

We know that in Real Time Applications, execution of multiple transactions concurrently is a must.

It is obvious that when multiple transactions execute serially, then there is no problem of **inconsistency** because a serial schedule doesn't allow concurrency, i.e. only one transaction executes at a time and the other starts when the already running transaction finishes.

When multiple transactions are running concurrently then there is a possibility that the database may be left in an **inconsistent** state

The coordination of the simultaneous execution of transactions in a multiuser database system is known as Concurrency control. Concurrency control is important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems. The three main problems are lost updates, uncommitted data, and inconsistent retrievals.

**These problems have been discussed in detail in the previous notes.**

The objective of concurrency control is to ensure the serializability of transactions in a multiuser database environment. A serializable schedule always leaves the database in consistent state.

### What is a serializable schedule?

A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finishes execution. However a Non-serial schedule needs to be checked for Serializability. A Non-serial schedule with N number of transactions is said to be serializable schedule, **if it is equivalent to the serial schedule of those N transactions.**

The following examples we will explain the concept of Serializability

**First we will see that there are two transactions T1 and T2 which are a part of schedule S1, S2, S3 and S4. A transaction is a group of instructions performing a logical task and a schedule is a group of all the executing transactions taken together. Since all the transactions will read some values from database , manipulate them and finally write the changes in the database, it is imperative there must be some mechanism to ensure that these transactions do the manipulations correctly in a proper order , so that database changes from one consistent state to another.**

**In S1 and S2 the Transactions T1 and T2 are executed serially where as n S3 and S4 they will be executed in an interleaved manner(concurrently).**
**It has been observed that when the transactions are executed serially, whatever may be the order of execution of transaction, they produce same results, so we can say that serial schedule are always keeping the database in consistent state.**
**However when the transactions execute concurrently, they may or may not maintain the consistency. The Concurrency Control Mechanism of DBMS is responsible to check this.**

**It means that we cannot say that all Non serial schedules lead to inconsistency. There are Non serial schedules which are equivalent to their serial schedule. These Non serial schedules which are equivalent to their serial schedule counterparts are called serializable schedule and such schedules do not cause any inconsistency .**

**Example: Schedule S1**

| T1 | T2 |
|---|---|
| R(A) | |
| A:=A-100 | |
| W(A) | |
| R(B) | |
| B:=B+500 | |
| W(B) | |
| | R(A) |
| | A=A+200 |
| | W(A) |

**Let us assume the value of A is 1000 and value of B is 2000.**

Now if we consider **Serial Schedule S1** in which T1 executes first and then T2 will execute. The final values will be **A = 1100 and B = 2500.**

Because value of A will be read , its value will decrease by 100 so it becomes 900 and writes this value in the Local buffer, then the value of B will be read and its value will be increased by 500, so B becomes 2500 and writes this value in the Local buffer. Now transaction T1 finishes.

Then transaction T2 starts , it will read the value of A which is now 900. It will increase the value of A by 200, so the value becomes 1100 which is the written into the local buffer. Now transaction T2 finishes.

**Example: Schedule S2**

| T1 | T2 |
|---|---|
| | R(A)<br>A=A+200<br>W(A) |
| R(A)<br>A:=A-100<br>W(A)<br>R(B)<br>B:=B+500<br>W(B) | |

Now if we consider **Serial Schedule S2** in which T2 executes first and then T1 will execute. The final values will be **A = 1100 and B = 2500.**

The transaction T2 starts , value of A will be read , its value will be increased by 200 so it becomes 1200 and writes this value in the Local buffer. Now transaction T2 finishes.

Now transaction T1 starts, it will read the value of A which is now 1200. It will be decreased by 100, so the value becomes 1100 which is then written into the Local buffer. Then the value of B will be read and its value will be increased by 500, so B becomes 2500 and the transaction writes this value in the Local buffer. Now transaction T1 finishes.

**So in both the cases, since the transactions were executed serially irrespective of the order of execution of participating transactions, the results were same, hence there was no inconsistency.**

**Example: Schedule S3**

| T1 | T2 |
|---|---|
| R(A) | |
| A:=A-100 | |
| | R(A) |
| | A:= A + 200 |
| W(A) | |
| R(B) | |
| | W(A) |
| B:=B+500 | |
| W(B) | |

This schedule is now a **Non serial schedule** because here context switching **or** interleaving of the intruction execution between particiapting transactions with in the schedule has been allowed.

In this schedulr Transaction T1 will start and read the value of A form Local buffer which was 1000 and decrement its value by 100, making its value 900. But before writing the value in local buffer, context switch occurs and Transaction T2 begins, it also reads the value of A as 1000 and increases the value by 200 making it to 1200 but it has also not written this value in the Local buffer.

Again a context switch occurs and transaction T1 resumes. It writes the value of A as 900 and then it reads the value of B from Local buffer as 2000.

Again a context switch occurs and the transaction T2 will write the value of A as 1200 in the local buffer overwriting the value 900(as T2 had added 200 to the initial value of A).

Again a context switch will occur, Transaction T1 will increment the value of B by 500 making it to 2500 and then will write the value of B as 2500.

So we see that the final value of **A= 1200** and **B = 2500** after the execution of this Schedule S3. Since this schedule did not produced the same result as its serial conterparts Schedule S1 and scheduleS2, **so S3 is not a Serializable Schedule.**

Let us take another example of a Non serial Schedule S4.

**Example: Schedule S4**

| T1 | T2 |
|---|---|
| R(A) | |
| A:=A-100 | |
| W(A) | |
| | R(A) |
| | A:= A + 200 |
| | W(A) |
| R(B) | |
| B:=B+500 | |
| W(B) | |

This schedule is also a **Non serial schedule** because here context switching or interleaving of the intruction execution between different transactions with in the schedule has been allowed.

In this schedule Transaction T1 will start and read the value of A form Local buffer which was 1000 and decrement its value by 100, making its value 900 and writes the value of A as 900 in the Local buffer.

Then a context switch occurs and Transaction T2 begins, it reads the value of A from the local buffer which has now become 900 increases the value by 200 making it to 1100, and writes it to the Local buffer.

Again a context switch occurs and transaction T1 resumes. and then it reads the value of B from local buffer as 2000 and increment the value of B by 500 making it to 2500 and then will write the value of B as 2500 in the buffer.

So we see that the final value of **A= 1100** and **B = 2500** after the execution of this Schedule S4. Since this schedule produced the same result as its serial conterparts Schedule S1 and scheduleS2, **so S4 is a Serializable Schedule.**