# Unit 4 Transaction

## Concurrent Executions:

**Definition:**

Transaction-processing systems usually allow multiple transactions to run concurrently.

Concurrent access from multiple client: We do not want to "lock out" the DBMS until one client finishes

**Advantages of concurrent execution**:

a. **Increased processor and disk utilization**-
   - A transaction consists of many steps. Some involve I/O activity; others involve CPU activity. The CPU and the disks in a computer system can operate in parallel. Therefore, I/O activity can be done in parallel with processing at the CPU.
   - The parallelism of the CPU and the I/O system can therefore be exploited to run multiple transactions in parallel.
   - While a read or write on behalf of one transaction is in progress on one disk, another transaction can be running in the CPU, while another disk may be executing a read or write on behalf of a third transaction.
   - All of this increases the throughput of the system—that is, the number of transactions executed in a given amount of time.

b. Reduced average response time for transactions-
   - There may be a mix of transactions running on a system, some short and some long.
   - If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete, which can lead to unpredictable delays in running a transaction.

**Problems:**

a. But with concurrent executions transactions can interfere with each other and produce results that could not have produced if they were executed in any serial order.
b. An execution order that produces the same results as a serial one is called a serializable schedule

**Concurrency Examples**

**DB Transactions**

TRANSACTION: A sequence of SQL statements that are executed "together" as one unit**:**

**T1. A money transfer transaction:**

S1: UPDATE Account SET balance = balance - 1000000 WHERE owner = 'Rahul' .
S2: Update Account SET balance = balance + 1000000 WHERE owner = 'Ramesh‘
S1 and S2 are the actions composing the transaction T1..

Example1: Concurrent with T1 we execute T2 that simply reports the current balances.
Example 2: T3: UPDATE Employee SET salary = salary + 1000 T4: UPDATE Emplo. SET salary = salary + 2000

Example 3: T5: UPDATE Employee SET salary = salary + 1000 T6: UPDATE Employee SET salary = salary * 0.2

Example 3 has a concurrency problem.

Example 2 does not--- because addition commutes.

But, we do not want to bother with the semantics of the operations. We want serializability criteria based only on read/write statements.

## Serializability

When multiple transactions are running concurrently then 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.

Schedules – sequences that indicate the chronological order in which instructions of concurrent transactions are executed.

A serializable schedule is the one that 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

### 1. Conflict Serializability

Which can be used to check whether a non-serial schedule is conflict serializable or not.

A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

### Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Lets see some examples to understand this:

**Example 1:** Operation W(X) of transaction T1 and operation R(X) of transaction T2 are conflicting operations, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on same data item X, one of the operation in write operation.

**Example 2:** Similarly Operations W(X) of T1 and W(X) of T2 are conflicting operations.

**Example 3:** Operations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on same data item so these operations don't satisfy the second condition.

**Example 4:** Similarly R(X) of T1 and R(X) of T2 are non-conflicting operations because none of them is write operation.

**Example 5:** Similarly W(X) of T1 and R(X) of T1 are non-conflicting operations because both the operations belong to same transaction T1.

### Conflict Equivalent Schedules

Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

### Conflict Serializable check

Lets check whether a schedule is conflict serializable or not. If a schedule is conflict Equivalent to its serial schedule then it is called Conflict Serializable schedule. Lets take few examples of schedules.

**Example of Conflict Serializability**

**Lets consider this schedule:**

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

To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable**.

**Lets take another example:**

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

Lets **swap non-conflicting operations**:

After swapping R(A) of T1 and R(A) of T2 we get:

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

After swapping R(A) of T1 and R(B) of T2 we get:

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

After swapping R(A) of T1 and W(B) of T2 we get:

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

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable**.

## 2. View Serializability:

**View Serializability** is a process to find out that a given schedule is view serializable or not.

To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule. Lets take an example to understand what I mean by that.

**Given Schedule:**

```
T1        T2
-----     ------
R(X)
W(X)
          R(X)
          W(X)
R(Y)
W(Y)
          R(Y)
          W(Y)
```

**Serial Schedule of the above given schedule:**

As we know that in Serial schedule a transaction only starts when the current running transaction is finished. So the serial schedule of the above given schedule would look like this:

```
T1        T2
-----     ------
R(X)
W(X)
R(Y)
W(Y)
          R(X)
          W(X)
          R(Y)
          W(Y)
```

If we can prove that the given schedule is **View Equivalent** to its serial schedule then the given schedule is called **view Serializable**.

**Why we need View Serializability?**

We know that a serial schedule never leaves the database in inconsistent state because there are no concurrent transactions execution. However a non-serial schedule can leave the database in inconsistent state because there are multiple transactions running concurrently. By checking that a given non-serial schedule is view serializable, we make sure that it is a consistent schedule.

You may be wondering instead of checking that a non-serial schedule is serializable or not, can't we have serial schedule all the time? The answer is no, because concurrent execution of transactions fully utilize the system resources and are considerably faster compared to serial schedules.

## View Equivalent

Lets learn how to check whether the two schedules are view equivalent.

Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. **Initial Read:** Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.

**Read vs Initial Read:** You may be confused by the term initial read. Here initial read means the first read operation on a data item, for example, a data item X can be read multiple times in a schedule but the first read operation on X is called the initial read. This will be more clear once we will get to the example in the next section of this same article.

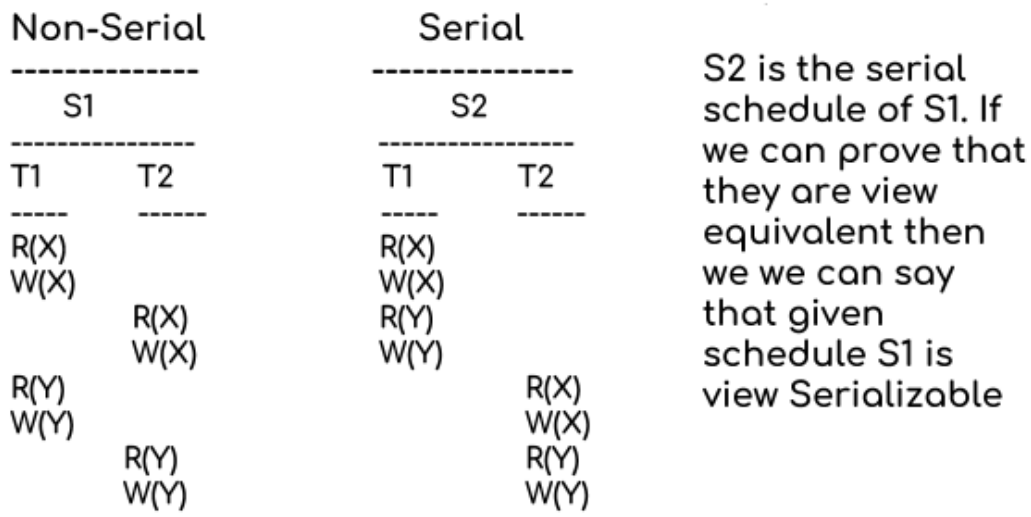
2. **Final Write:** Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

3. **Update Read:** If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

## View Serializable

If a schedule is view equivalent to its serial schedule then the given schedule is said to be View Serializable. Lets take an example.

**View Serializable Example**




Lets check the three conditions of view serializability:

**Initial Read**

In schedule S1, transaction T1 first reads the data item X. In S2 also transaction T1 first reads the data item X.

Lets check for Y. In schedule S1, transaction T1 first reads the data item Y. In S2 also the first read operation on Y is performed by T1.

We checked for both data items X & Y and the **initial read** condition is satisfied in S1 & S2.

**Final Write**

In schedule S1, the final write operation on X is done by transaction T2. In S2 also transaction T2 performs the final write on X.

Lets check for Y. In schedule S1, the final write operation on Y is done by transaction T2. In schedule S2, final write on Y is done by T2.

We checked for both data items X & Y and the **final write** condition is satisfied in S1 & S2.

**Update Read**

In S1, transaction T2 reads the value of X, written by T1. In S2, the same transaction T2 reads the X after it is written by T1.

In S1, transaction T2 reads the value of Y, written by T1. In S2, the same transaction T2 reads the value of Y after it is updated by T1.

The update read condition is also satisfied for both the schedules.

**Result:**

Since all the three conditions that checks whether the two schedules are view equivalent are satisfied in this example, which means S1 and S2 are view equivalent. Also, as we know that the schedule S2 is the serial schedule of S1, thus we can say that the schedule S1 is view serializable schedule.