

# Database Management System 20

## Concurrent Execution

Concurrent Execution

Schedules

Serial Schedule

Concurrent Schedule

Serializability

Conflict Serializability

Testing for Conflict  
Serializability

View Serializability

Chittaranjan Pradhan  
School of Computer Engineering,  
KIIT University

## Concurrent Execution

Concurrent execution of transactions means executing more than one transaction at the same time

In the serial execution, one transaction can start executing only after the completion of the previous

The advantages of using concurrent execution of transactions are:

- Improved throughput and resource utilization
- Reduced waiting time

The database system must control the interaction among the concurrent transactions to prevent them from destroying the consistency of the database. It does this through a variety of mechanisms called **concurrency control schemes**

## Concurrent Execution

### Schedules

[Serial Schedule](#)[Concurrent Schedule](#)

### Serializability

[Conflict Serializability](#)[Testing for Conflict Serializability](#)[View Serializability](#)

$T_1$  transfers Dollar \$100 from account A to account B

$T_1$
Read(A); A:=A-100; Write(A); Read(B); B:=B+100; Write(B);

$T_2$  transfers 20% of balance from account A to account B

$T_2$
Read(A); Temp=0.2*A; A:=A-Temp; Write(A); Read(B); B:=B+Temp; Write(B);

## Schedules

A **schedule** is a sequence that indicates the chronological order in which instructions of concurrent transactions are executed

A schedule for a set of transactions must consist of all instructions of those transactions

We must preserve the order in which the instructions appear in each individual transaction

## Serial Schedule

*A serial schedule is a schedule where all the instructions belonging to each transaction appear together*

There is no interleaving of transaction operations. A serial schedule has no concurrency and therefore it does not interleave the actions of different transactions

*For  $n$  transactions, there are exactly  $n!$  different serial schedules possible*

## Serial Schedule...

Schedule1 ( $T_1$  followed by  $T_2$ )

$T_1$	$T_2$
Read(A); A:=A-100; Write(A); Read(B); B:=B+100; Write(B);	Read(A); Temp=0.2*A; A:=A-Temp; Write(A); Read(B); B:=B+Temp; Write(B);

Schedule2 ( $T_2$  followed by  $T_1$ )

$T_1$	$T_2$
Read(A); A:=A-100; Write(A); Read(B); B:=B+100; Write(B);	Read(A); Temp=0.2*A; A:=A-Temp; Write(A); Read(B); B:=B+Temp; Write(B);

# Concurrent Schedule

## Concurrent Schedule

*In concurrent schedule, operations from different concurrent transactions are interleaved*

*The number of possible schedules for a set of  $n$  transactions is much larger than  $n!$*

Schedule3

$T_1$	$T_2$
Read(A); A:=A-100; Write(A);	
	Read(A); Temp=0.2*A; A:=A-Temp; Write(A);
Read(B); B:=B+100; Write(B);	
	Read(B); B:=B+Temp; Write(B);

# Concurrent Schedule...

Schedule4

$T_1$	$T_2$
Read(A); A:=A-100;	Read(A); Temp=0.2*A; A:=A-Temp; Write(A); Read(B);
Write(A); Read(B); B:=B+100; Write(B);	B:=B+Temp; Write(B);

Schedule5

$T_1$	$T_2$
Read(A); A:=A-100; Write(A);	Read(A); Temp=0.2*A; A:=A-Temp;
Read(B); B:=B+100;	Write(A); Read(B); B:=B+Temp; Write(B);
Write(B);	



## Serializability

***A concurrent schedule is serializable if it is equivalent to a serial schedule***

Serial schedules preserve consistency as we assume each transaction individually preserves consistency

The database system must control concurrent execution of transactions to ensure that the database state remains consistent

Since the modifications are done in the local buffer, we can ignore the operations other than Read and Write instructions for easier understanding of the serializability

# Serializability...

## Concurrent Execution

### Schedules

Serial Schedule

Concurrent Schedule

### Serializability

Conflict Serializability

Testing for Conflict

Serializability

View Serializability

$T_1$	$T_2$		$T_1$	$T_2$
Read(A); Write(A); Read(B); Write(B);				Read(A); Write(A); Read(B); Write(B);
	Read(A); Write(A); Read(B); Write(B);		Read(A); Write(A); Read(B); Write(B);	
Schedule1			Schedule2	

$T_1$	$T_2$		$T_1$	$T_2$		$T_1$	$T_2$
Read(A); Write(A);			Read(A);	Read(A); Write(A); Read(B);		Read(A); Write(A);	
Read(B); Write(B);	Read(A); Write(A);		Write(A); Read(B); Write(B)			Read(B);	Read(A);
	Read(B); Write(B);			Write(B);		Write(B);	Write(A); Read(B); Write(B);
Schedule3			Schedule4			Schedule5	

## Conflict Serializability

Conflict Serializability consists of conflicting operations

Let us consider a schedule  $S$  in which there are two consecutive instructions,  $I_i$  and  $I_j$  of transactions  $T_i$  and  $T_j$  respectively ( $i \neq j$ )

If  $I_i$  and  $I_j$  access different data items, then we can swap  $I_i$  and  $I_j$  without affecting the results of any transactions in the schedule. However, if  $I_i$  and  $I_j$  access the same data item  $Q$ , then the order of the two instructions may matter:

- **Case-1:  $I_i = \text{Read}(Q)$  and  $I_j = \text{Read}(Q)$ :**
  - Order of  $I_i$  and  $I_j$  does not matter
- **Case-2:  $I_i = \text{Read}(Q)$  and  $I_j = \text{Write}(Q)$ :**
  - Order of  $I_i$  and  $I_j$  matters in a schedule
- **Case-3:  $I_i = \text{Write}(Q)$  and  $I_j = \text{Read}(Q)$ :**
  - Order of  $I_i$  and  $I_j$  matters in a schedule
- **Case-1:  $I_i = \text{Write}(Q)$  and  $I_j = \text{Write}(Q)$ :**
  - Order of  $I_i$  and  $I_j$  matters in a schedule

## Conflict Serializability...

Thus,  $I_i$  and  $I_j$  **conflict** if they are the instructions by different transactions on the same data item, and at least one of these instructions is a write operation

Let  $I_i$  and  $I_j$  be consecutive instructions of a schedule  $S$ . If  $I_i$  and  $I_j$  are instructions of different transactions and they do not conflict, then we can swap the order of  $I_i$  and  $I_j$  to produce a new schedule  $S'$ . Here, we expect  $S$  to be equivalent to  $S'$

*If a schedule  $S$  can be transformed into a schedule  $S'$  by a series of swaps of non-conflicting instructions, we say that  $S$  and  $S'$  are conflict equivalent*

***A concurrent schedule  $S$  is conflict serializable if it is conflict equivalent to a serial schedule***

# Conflict Serializability...

$T_1$	$T_2$		$T_1$	$T_2$		$T_1$	$T_2$
Read(A); Write(A); Read(B); Write(B);			Read(A);  Write(A);  Read(B); Write(B);	Read(A); Write(A);  Read(B);  Write(B);		Read(A); Write(A); Read(B);  Write(B);	Read(A); Write(A); Read(B); Write(B);
Schedule3			Schedule4			Schedule5	

## Conflict Serializability...

It is possible to have two schedules that produce the same outcome, but that are not conflict equivalent

$T_1$	$T_5$
Read(A); A:=A-100; Write(A);  Read(B); B:=B+100; Write(B);	Read(B); B:=B-200; Write(B);  Read(A); A:=A+200; Write(A);
Schedule6	

$T_1$	$T_5$
Read(A); Write(A);  Read(B); Write(B);	Read(B); Write(B);  Read(A); Write(A);
Schedule6	

## Testing for Conflict Serializability

Construct a directed graph, called a **precedence graph** from  $S$ . This graph consists of a pair  $G = (V, E)$ , where  $V$  is a set of vertices and  $E$  is a set of edges

The set of vertices consists of all the transactions participating in the schedule

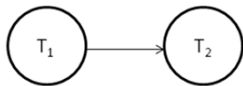
The set of edges consists of all edges  $T_i \rightarrow T_j$  for which one of three conditions holds:

- $T_i$  executes write(Q) before  $T_j$  executes read(Q)
- $T_i$  executes read(Q) before  $T_j$  executes write(Q)
- $T_i$  executes write(Q) before  $T_j$  executes write(Q)

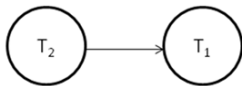
If an edge  $T_i \rightarrow T_j$  exists in the precedence graph, then in any serial schedule  $S'$  equivalent to  $S$ ,  $T_i$  must appear before  $T_j$

## Testing for Conflict Serializability...

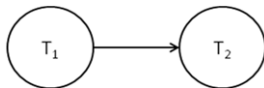
If the precedence graph for a concurrent schedule  $S$  has a **cycle**, then that schedule is not conflict serializable. If the graph contains no cycles, then the schedule  $S$  is conflict serializable



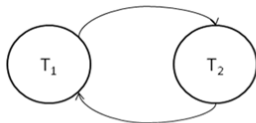
Schedule 1



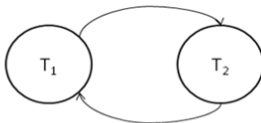
Schedule 2



Schedule 3



Schedule 4



Schedule 5



### View Serializability

The schedules  $S$  and  $S_1$  are said to be view equivalent if the following three conditions are met:

- For each data item  $Q$ , if transaction  $T_i$  reads the initial value of  $Q$  in schedule  $S$ , then it must also read the initial value of  $Q$  in schedule  $S_1$
- For each data item  $Q$ , the transaction that performs the final  $\text{Write}(Q)$  operation in schedule  $S$  must also perform the final  $\text{Write}(Q)$  operation in schedule  $S_1$
- For each data item  $Q$ , if transaction  $T_i$  executes  $\text{Read}(Q)$  in schedule  $S$ , and if that value was produced by a  $\text{Write}(Q)$  operation executed by transaction  $T_j$ ; then in schedule  $S_1$ , the  $\text{Read}(Q)$  operation of  $T_i$  must also read the value of  $Q$  that was produced by the same  $\text{Write}(Q)$  operation of transaction  $T_j$

***A schedule  $S$  is view serializable if it is view equivalent to a serial schedule***

## View Serializability...

Schedule7

$T_1$	$T_2$	$T_3$
Read(Q); Write(Q);	Write(Q);	Write(Q);

- This schedule is view serializable
- This schedule is not conflict serializable
- *Every conflict serializable schedule is also view serializable, whereas all view serializable schedules are not conflict serializable*
- Every view serializable schedule, which is not conflict serializable, has **blind writes**