Database Management System

Unit-4

Dharmendra Kumar(Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research, Prayagraj

TRANSACTION

1 Transaction

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- A transaction is an action or sequence of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs. 800 from X's account to Y's account. This small transaction contains several low-level tasks:
X's Account:
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)

Y's Account:
Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.
Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. Read(X);
2. X = X - 500;
3. Write(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

2 Properties of Transaction

To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

1. **Atomicity:** Either all operations of the transaction are reflected properly in the database, or none are.
2. **Consistency:**
   - This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.
   - Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
3. **Isolation:**
   - In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
   - It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
4. **Durability:** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are often called the ACID properties.

Example: Let Ti be a transaction that transfers \$50 from account A to account B. This transaction can be defined as

```
Ti: read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B).
```

3 Transaction State

A transaction must be in one of the following states:

- **Active:** This is the initial state. The transaction stays in this state while it is executing.
- **Partially committed:** Transaction enter into this state after the final statement has been executed.
- **Failed:** Transaction enter into this state after the discovery that normal execution can no longer proceed.
- **Aborted:** Transaction enter into this state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- **Committed:** Transaction enter into this state after successful completion.

The state diagram corresponding to a transaction is the following:-
When transaction enters the aborted state, at this point, the system has two options:

- It can restart the transaction, but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a new transaction.
- It can kill the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.

4 Schedule

A schedule is a sequence of instructions of all the transactions in which order these instructions will execute.
There are two types of schedule. (1) Serial schedule (2) Concurrent schedule

Serial schedule: The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed. **Concurrent schedule:** If interleaving of operations is allowed, then the schedule will be concurrent schedule.

Example: Consider the following two transactions:-
Let T1 and T2 be two transactions that transfer funds from one account to another. Transaction T1 transfers $50 from account A to account B. It is defined as

```
T1: read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B).
```

Transaction T2 transfers 10 percent of the balance from account A to account B. It is defined as

```
T2: read(A);
temp := A * 0.1;
A := A - temp;
write(A);
read(B);
B := B + temp;
write(B).
```

Now we make following four schedules for these transactions.
schedule-1, schedule-2, schedule-3, and schedule-4.

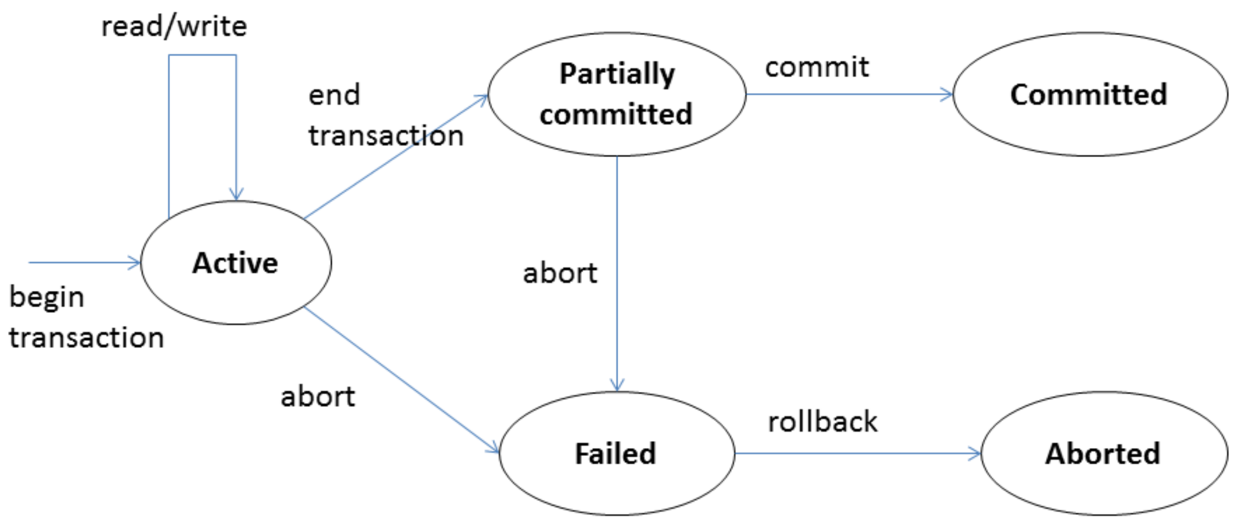
5 Serializability

To ensure consistency of database system, we must make a serializable schedule. Here, we will study two types of Serializability. These are (1) Conflict Serializability (2) View Serializability.

5.1 Conflict Serializability

Conflict Instructions:
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≠j). If I_i and I_j operates on same data item such as Q, then these instructions will be conflicting instructions if any one of the following is satisfied.

(1) I_i = read(Q), and I_j = write(Q).



State diagram for the execution of a transaction

<u>Schedule-1</u>

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

<u>Schedule-2</u>

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

Schedule-3

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Schedule-4

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

(2) I_i = write(Q), and I_j = read(Q).
(3) I_i = write(Q), and I_j = write(Q).

In all other cases, instructions I_i and I_j will be non-conflicting.

Conflict equivalent: Two schedules S and S' are said to be conflict equivalent if a schedule S can be transformed in to a schedule S' by using swapping of non-conflicting instructions.

Conflict serializable: A schedule S is said to be conflict serizaliabe if it is conflict equivalent to a serial schedule.

5.1.1 Some examples:

Example: Check schedule-1, schedule-2, schedule-3 and schedule-4 are conflict serializable or not.
Solution:
(1) Clearly schedule-1 and schedule-2 are serial schedules,therefore these schedules are conflict serializable.
(2) Consider schedule-3 i.e.

In this schedule, instruction read(B) in transaction T_1 is non-conflicting with instructions read(A) and write(A) in transaction T_2, therefore we can swap instructions read(B) in T_1 and write(A)in T_2. Similarly, we can swap instructions read(B) in T_1 and read(A)in T_2.

Similarly, instruction write(B) in transaction T_1 is non-conflicting with instructions read(A) and write(A) in transaction T_2, therefore we can swap instructions write(B) in T_1 and write(A)in T_2. Similarly, we can swap instructions write(B) in T_1 and read(A)in T_2.

Therefore, using swapping, schedule-3 can be transformed in to a serial schedule-1 i.e. T_1T_2. Therefore schedule-3 is conflict serializable.

(3) Consider schedule-4 i.e.
In this schedule, instruction write(A) in transaction T_1 and instruction write(A) in transaction T_2 are conflicting instructions, therefore, we can not swap these two instructions. Therefore, schedule-4 can not be transformed in to any serial schedule. Hence, schedule-4 is not conflict serializable.

Example: Consider the following schedule-5:-
Is this schedule conflict serializable?
Solution:
Clearly, in this schedule, instruction read(Q) in transaction T_3 and instruction write(Q) in transaction T_4 are conflicting instructions, therefore, we can not swap these two instructions.

Similarly, instruction write(Q) in transaction T_3 and instruction write(Q) in transaction T_4 are conflicting instructions, therefore, we can not swap these two instructions.

Schedule-3

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Schedule-4

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Schedule-5

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

In this situation, this schedule can not be transformed into any serial schedule. Therefore, the schedule-5 is non-conflict serializable.

Example: Consider the following schedule-6:-
Is this schedule conflict serializable?
Solution:
In this schedule, instruction read(B) in transaction T_1 and instruction write(B) in transaction T_5 are conflicting instructions, therefore, we can not swap these two instructions. Hence, this schedule can not be transformed into serial schedule T_1T_5.

Similarly, instruction write(A) in transaction T_1 and instruction read(A) in transaction T_5 are conflicting instructions, therefore, we can not swap these two instructions. Hence, this schedule can not be transformed into serial schedule T_5T_1.

Therefore, this schedule-6 is non-conflict serializable.

5.2 View Serializability

View equivalent: Consider two schedules S and S', where the same set of transactions participates in both schedules. The schedules S and S' are said to be view equivalent if following three conditions are satisfied:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.
2. For each data item Q, if transaction T_i executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction T_j , then the read(Q) operation of transaction T_i must, in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction T_j .
3. For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'.

View serializable: A schedule S is said to be view serizaliabe if it is view equivalent to a serial schedule.

Example: Consider the following schedule-3:-
Is this schedule view serializable?
Solution:
First, we check the view equivalent of schedule-3 with serial schedule-1 i.e. T_1T_2.
First consider data item A.
Clearly, in schedule-3 and schedule-1, transaction T_1 reads the initial value of A, therefore condition-1 is satisfied for data item A.
Clearly, in schedule-3, transaction T_2 reads the value of A written by T_1. In schedule-1, transaction T_2 reads the value of A written by T_1. Clearly same order of write-read occur in both schedule, therefore second condition is also satisfied.
Clearly, in schedule-3, final write(A) operation is performed by transaction T_2. In schedule-1, final write(A) operation is also performed by transaction T_2. Therefore, third condition is also satisfied.
Clearly all the three conditions are satisfied for data item A.

Schedule-6

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Schedule-3

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Now, we will check all the three conditions for data item B.
Clearly, in schedule-3 and schedule-1, transaction T_1 reads the initial value of B, therefore condition-1 is satisfied for data item B.
Clearly, in schedule-3, transaction T_2 reads the value of B written by T_1. In schedule-1, transaction T_2 reads the value of B written by T_1. Clearly same order of write-read occur in both schedule, therefore second condition is also satisfied.
Clearly, in schedule-3, final write(B) operation is performed by transaction T_2. In schedule-1, final write(B) operation is also performed by transaction T_2. Therefore, third condition is also satisfied.
Clearly all the three conditions are satisfied for data item B.
Since all the three conditions are satisfied for each data item in both schedules 1 and 3, therefore both schedule-3 is view equivalent to serial schedule-1. Therefore, schedule-3 is view serializable.‘

Example: Consider the following schedule-4:-
Is this schedule view serializable?
Solution:
First, we check the view equivalent of schedule-4 with serial schedule-1 i.e. T_1T_2.
First consider data item A.
Clearly, in schedule-4 and schedule-1, transaction T_1 reads the initial value of A, therefore condition-1 is satisfied for data item A.
Clearly, in schedule-4, there is no transaction which reads the value of A written by any transaction. Therefore second condition is also satisfied.
Clearly, in schedule-4, final write(A) operation is performed by transaction T_1. But, in schedule-1, final write(A) operation is performed by transaction T_2. Therefore, third condition is not satisfied.
Therefore, schedule-4 is not view equivalent to a serial schedule T_1T_2.

Now, we check the view equivalent of schedule-4 with serial schedule-2 i.e. T_2T_1.
First consider data item A.
Clearly, in schedule-4, transaction T_1 reads the initial value of A, but in serial schedule-2, transaction T_2 reads the initial value of A. Therefore condition-1 is not satisfied for data item A.
Therefore, schedule-4 is not view equivalent to a serial schedule T_1T_2.
Therefore, schedule-4 is not view serializable.‘

Example: Consider the following schedule-7:-
Is this schedule view serializable?
Solution:
Clearly this schedule is view equivalent to serial schedule $T_3T_4T_6$. Therefore, this schedule is view serializable.

6 Testing for Serializability

In this section we are going to study a simple and efficient method for determining conflict serializability of a schedule. This method is explained as following.

Consider a schedule S. We 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:

1. T_i executes write(Q) before T_j executes read(Q).
2. T_i executes read(Q) before T_j executes write(Q).
3. T_i executes write(Q) before T_j executes write(Q).

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

A **serializability order** of the transactions can be obtained through topological sorting, which determines a linear order consistent with the partial order of the precedence graph.

Example: Consider the following schedule-3:-
Construct precedence graph for it.
Solution:
Precedence graph for above schedule will be the following:-
Clearly this graph does not contain any cycle. Therefore, this schedule is conflict serializable.

Example: Consider the following schedule-4:-
Construct precedence graph for it.
Solution:
Precedence graph for above schedule will be the following:-
Clearly this graph contains a cycle. Therefore, this schedule is not conflict serializable.

Example: Consider the following schedule-7:-
Construct precedence graph for it.
Solution:
Precedence graph for above schedule will be the following:-
Clearly this graph contains a cycle. Therefore, this schedule is not conflict serializable.

Note: If a schedule is conflict serializable then it is also view serializable. But converse need not be true.

7 Recoverability

If a transaction T_i fails, for whatever reason, we need to undo the effect of this transaction to ensure the atomicity property of the transaction. In a system that allows concurrent execution, it is necessary also to ensure that any transaction T_j that is dependent on T_i (that is, T_j has read data written by T_i) is also aborted. To achieve this surety, we need to place restrictions on the type of schedules permitted in the system.
In the following sections, we will study two schedules which are acceptable from the viewpoint of recovery from transaction failure.

Schedule-4

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Schedule-7

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Schedule-3

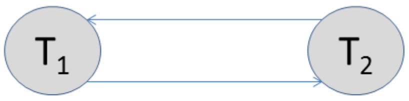
T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

$$\begin{tikzcd}T_1 \arrow[r] & T_2\end{tikzcd}$$

Precedence graph for schedule-3

Schedule-4

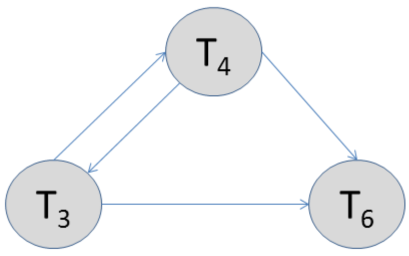
T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)



Precedence graph for schedule-4

Schedule-7

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)



Precedence graph for schedule-7

7.1 Recoverable Schedules

A schedule is said to be recoverable schedule if for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.

Example: Is the following schedule recoverable?
Solution:
Clearly, in this schedule transaction T_9 reads the value of A written by transaction T_8, but T_9 commits before T_8. Therefore this schedule is not recoverable schedule.

7.2 Cascadeless Schedules

A schedule is said to be cascadeless schedule if for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

Example: Is the following schedule cascadeless?

Solution:
Clearly, in this schedule transaction T_{11} reads the value of A written by transaction T_{10} and T_{10} commits before the read operation of T_{11}.
Similarly, transaction T_{12} reads the value of A written by transaction T_{11} and T_{11} commits before the read operation of T_{12}.
Therefore this schedule is cascadeless schedule.

Note: Every cascadeless schedule is also recoverable schedule. But converse need not be true.

8 Exercise

1. Consider the following two transactions:
```
$T_1$: read(A);
       read(B);
       if A = 0 then B := B + 1;
       write(B)
$T_2$: read(B);
       read(A);
       if B = 0 then A := A + 1;
       write(A)
```
   Let the consistency requirement be A $=$ 0 $\vee$ B $=$ 0, with A $=$ B $=$ 0 the initial values.

   (a) Show that every serial execution involving these two transactions preserves the consistency of the database.

   (b) Show a concurrent execution of T_1 and T_2 that produces a non-serializable schedule.

(c) Is there a concurrent execution of T_1 and T_2 that produces a serializable schedule?

2. Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

   (a) $r_1(X); r_3(X); w_1(X); r_2(X); w_3(X);$

   (b) $r_1(X); r_3(X); w_3(X); w_1(X); r_2(X);$

   (c) $r_3(X); r_2(X); w_3(X); r_1(X); w_1(X);$

   (d) $r_3(X); r_2(X); r_1(X); w_3(X); w_1(X);$

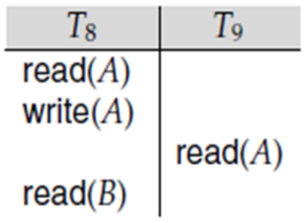
3. Consider the three transactions T_1, T_2, and T_3, and the schedules S_1 and S_2 given below. Draw the serializability (precedence) graphs for S_1 and S_2 and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s).

   T_1: $r_1(X); r_1(Z); w_1(X);$
   T_2: $r_2(Z); r_2(Y); w_2(Z); w_2(Y);$
   T_3: $r_3(X); r_3(Y); w_3(Y);$
   S_1: $r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); w_3(Y); r_2(Y); w_2(Z); w_2(Y);$
   S_2: $r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); w_2(Z); w_3(Y); w_2(Y);$

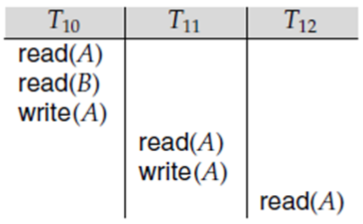
9 AKTU previous year questions

1. What do you mean by Conflict Serializable Schedule?
2. What do you understand by ACID properties of transaction ? Explain in details.
3. Define Transaction and explain its properties with suitable example.
4. What is schedule? What are its types? Explain view serializable and cascadeless schedule with suitable example of each.
5. Which of the following schedules are conflicts serializable? For each serializable schedule find the equivalent serial schedule.
   S_1: $r_1(x); r_3(x); w_3(x); w_1(x); r_2(x)$
   S_2: $r_3(x); r_2(x); w_3(x); r_1(x); w_1(x)$
   S_3: $r_1(x); r_2(x); r_3(y); w_1(x); r_2(z); r_2(y); w_2(y)$
6. Explain I in ACID Property.
7. Define schedule.
8. What do you mean by serializability? Discuss the conflict and view serialzability with example. Discuss the testing of serializability also.
9. What do you mean by Transaction? Explain transaction property with detail and suitable example.
10. What is serializability? How it is tested?

11. State the properties of transaction.
12. What is transaction? Draw a state diagram of a transaction showing its state. Explain ACID properties of a transaction with suitable examples.
13. What are the schedules? What are the differences between conflict serialzability and view serialzability ? Explain with suitable examples what are cascadeless and recoverable schedules?



T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	



T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)