SE240: Introduction to Database Systems

Overview of Transaction Management and Concurrent Control

Outline

- Transaction
- Serial Schedule
- Conflict Serializability
- View Serializability
- Strict Two-Phase Locking

Transactions

- A user's program may carry out many operations on the data from the database, but the DBMS is only concerned about what data is read/written from/to the database.
- A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes.

Transactions

- Sequence of operations
- E.g. T1: Read(A), A=A+100, Write(A)

  : R(A), W(A) single transaction

  Can also be denoted by $R_1(A)$, $W_1(A)$
- E.g. T2 : W(B), R(A) two transactions

Example of User Program and Transaction

- Assume A is a tuple in relation Account
- User program

```
CalculateInterests(Account A, Rate r)
{
    float x, y;
    x := A;    // read from database
    if (x <= 1000) y := 1.05*x
    else if (x <= 10000) y := 1.06*x
    else y := 1.065*x;
    A := y;    // write to database
}
```

Corresponding Transaction T

```
T: x := A, A := y        or        T: R(A), W(A)
```

ACID Property

- Atomicity
  - In a transaction, either all operations are carried out or none are.
- Consistency
  - Regardless of other transactions, each transaction must preserve the consistency of the database
- Isolation
  - User can understand a transaction without considering the effect of other transactions
- Durability
  - The effect of transaction should persist forever whenever the transaction is completed/committed.

Outline

- Transaction
- ACID Property
- Serial Schedule
- Conflict Serializability
- View Serializability
- Strict Two-Phase Locking

Schedules

- Transaction
  - E.g. T_1 : R(A), W(A) → transaction
  - E.g. T_2 : W(B), R(A)
- Schedule
  - A sequence of operations in a set of transactions $\{T_1, T_2, \ldots, T_n\}$
  - E.g. If a set of transactions is $\{T_1, T_2\}$,

For a intuitive understanding, we can list the operations in T1 and T2 in a line.
Thus, you will find out that the transactions perform serially.(One after another)

	t_1	t_2	t_3	t_4
H_1: T_1:	R(A),	W(A),		
H_1: T_2:			W(B),	R(A)
H_2: T_1:	R(A),		W(A),	
H_2: T_2:		W(B),		R(A)
H_3: T_1:		R(A),		W(A)
H_3: T_2:	W(B),		R(A),	

⇔ T_1: R(A), W(A) T_2: W(B), R(A)

→ serial schedule

NOT serial schedule

Serial Schedules

- Serial schedule
  - A schedule which the operations belonging to one single transaction appear together
  - E.g. H_1 is a serial schedule

    H_1: T_1T_2

    H_2 and H_3 are not serial schedule
- Serializable schedules
  - Equivalent to some serial schedule
  - E.g. H_1 and H_2 are serializable schedules (to T_1T_2)
  - H_3 is a serializable schedule (to T_2T_1).

H_1:	T_1:	R(A),	W(A),		
	T_2:			W(B),	R(A)
H_2:	T_1:	R(A),		W(A),	
	T_2:		W(B),		R(A)
H_3:	T_1:		R(A),		W(A)
	T_2:	W(B),		R(A),	

serial schedule

NOT!

W(B), R(A), R(A), W(A)

but this is a serializable schedule

Likewise, we can list transactions in T2 and T1 in a line

Anomalies with Interleaved Execution

- Reading Uncommitted Data (WR Conflicts, “dirty reads”)
- Read an object modified by uncommitted transaction

T1:	R(A),	W(A),					R(B),	W(B), About
T2:			R(A),	W(A),	R(B),	W(B),	C	

- T1 transfers $100 from A to B, T2 add 6% to A and B

Anomalies with Interleaved Execution

- Unrepeatable Reads (RW Conflicts):
- T1 tries to read a data object again after T2 modified it. The data object may have a different value.

Violate the consistency of database

T1:	R(A),			R(A), W(A), C
T2:		R(A), W(A), C		

- Imagine T1 verifies first that fund is sufficient, then withdraw.
- T2 is a simple withdraw

Anomalies with Interleaved Execution

- Overwriting Uncommitted Data (WW Conflicts):

T1:	W(A),			W(B), C
T2:		W(A),	W(B), C	

  - Both A and B must be initialized to 1 or 0.
  - T1 writes 1 to both, and T2 writes 0 to both.
  - A = 0, B = 1
  - (T1 writes 1 to A, but overwritten by T2)

  Now A and B have got different values (lost update)

- ***None of these anomalies is a serializable schedule!!***

used to maintain the consistency of database

Serializable Schedule and Serializability

- A schedule that is equivalent to some serial execution of the transactions is called a serializable schedule.
- Every serializable schedule preserves consistency
  - In the serial schedule, transactions are executed one after another
  - Every transaction itself preserves consistency (by consistency property)
  - So the serial schedule preserves consistency
  - So the serializable schedule preserves consistency

Serializable Schedule and Serializability

- The objective of serializability to find non-serial schedules
  - Allow transactions to execute concurrently without interfering with one another
  - Produce a database state that could be produced by a serial execution
- It is important to guarantee serializability of concurrent transactions in order to prevent inconsistency
- In serializability, the ordering of read and write operations in important

Outline

- Transaction
- ACID Property
- Serial Schedule
- Conflict Serializability
- View Serializability
- Strict Two-Phase Locking

Serializable Schedule

- A schedule (history) of transactions is serial if

  $\forall i \forall j, i \neq j \Rightarrow$ either

  - all operations in T_i appear before all operations in T_j, or
  - all operations in T_j appear before all operations in T_i.
- Assumption: Each T_i is correct when executed individually, i.e., all serial schedules are valid.
- Objective: Accept schedules "equivalent" to a serial schedule (serializable schedules).
- What do we mean by "*equivalent*".

Conflict Serializability

- Two operations are **conflict** if
  1. They are operations of different transactions on the **same** data object
  2. At least one of them is a **Write** operation
  - E.g. ① W(X), R(X) or R(X), W(X) ② W(X)-twice

T_i: W(X),	T_i: R(X),	T_i: W (X),
T_j: R(X)	T_j: W(X)	T_j: W(X)

conflict equivalent

- Two operations are **non-conflict**
  - E.g.

T_i: R(X),		
T_j: R(X)		
T_i: W(X),	T_i: R(X),	T_i: W (X),
T_j: R(Y)	T_j: W(Y)	T_j: W(Y)

Conflict Equivalent

- Two schedules S_1 and S_2 are **conflict equivalent** if
  - S_1 and S_2 involve the *same operations* of the *same* transaction
  - Every pair of conflicting *operations* is *ordered* in the *same* way in S_1 and S_2

Conflict Equivalent

- E.g.1. Is H_8 and H_9 conflict equivalent?

H_8:

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

H_9:

T_1:	W (X),	R(Y),		
T_2:			R(Y),	R(X)

Conflict Equivalent

- E.g.1. Is H_8 and H_9 conflict equivalent?

in terms of object X

X

H_8:

T_1:	W (X),		R(Y),
T_2:		R(Y),	R(X)

H_9:

T_1:	W (X),	R(Y),	
T_2:		R(Y),	R(X)

Conflict Operations?

YES

Same Order?

YES

Y

H_8:

T_1:	W (X),		R(Y),
T_2:		R(Y),	R(X)

H_9:

T_1:	W (X),	R(Y),	
T_2:		R(Y),	R(X)

Conflict Operations?

NO

in terms of object Y

H_8 and H_9 are conflict equivalent

Conflict Equivalent

- E.g.2. Is H_8 and H_{10} conflict equivalent?

H_8:

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

H_{10}:

T_1:			W(X),	R(Y)
T_2:	R(Y),	R(X),		

Conflict Equivalent

- E.g.2. Is H_8 and H_{10} conflict equivalent?

X

t_1 t_2 t_3 t_4

H_8:

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

H_{10}:

T_1:			W(X),	R(Y)
T_2:	R(Y),	R(X),		

from the left to right

Conflict Operations?

YES

Same Order?

NO

H_8 and H_{10} are NOT conflict equivalent

Conflict Serializability

- S is **conflict serializable** if it is conflict equivalent to a serial schedule
- E.g.

H_8:

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

H_9:

T_1:	W (X),	R(Y),		
T_2:			R(Y),	R(X)

- H_8 and H_9 are conflict equivalent
- H_9 is a serial schedule
- H_8 is conflict serializable

Conflict-serializability - Both schedules have the same sets of respective chronologically ordered pairs of conflicting operations.

Precedence Graph

- Test for conflict serializability
- A directed graph G=(V,E), where
  - V includes all transactions involved in the schedule
  - E consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:

Conflict Operations

    - T_i executes write(X) before T_j executes read(X)
    - T_i executes read(X) before T_j executes write(X)
    - T_i executes write(X) before T_j executes write(X)

T_i:	W(X),	
T_j:		R(X)

T_i:	R(X),	
T_j:		W(X)

T_i:	W (X),	
T_j:		W(X)

conflict w(x) twice

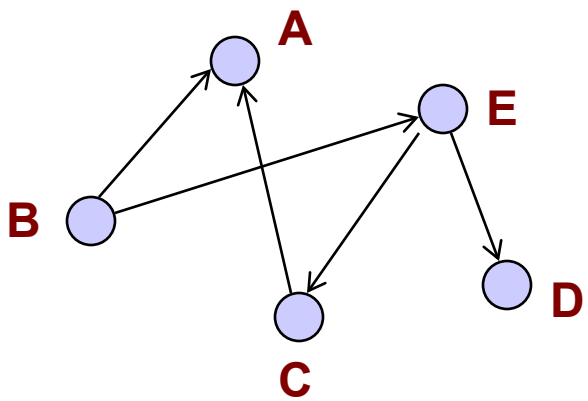
Precedence Graph

if and only if

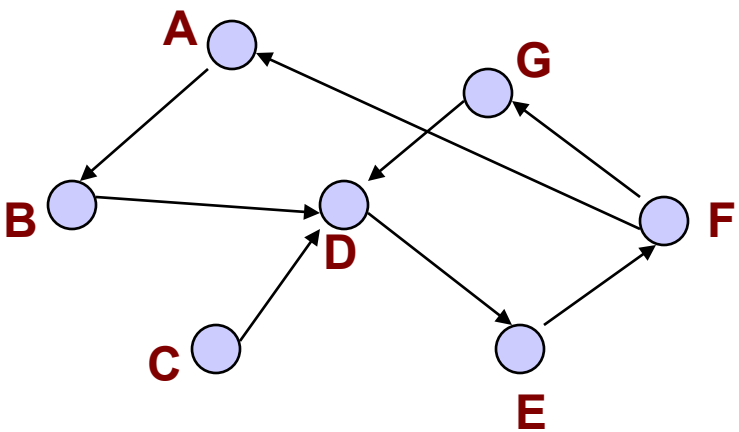
- ***Theorem:*** A schedule S is conflict serializable iff G(S) is **acyclic (i.e. no cycle)**
- The serialization order can be obtained through ***topological sorting***, which determines a linear order consistent with the partial order of the dependence graph

Precedence Graph

- Cycle in a graph
  - A cycle is a path that starts and terminates at the same node
- Examples



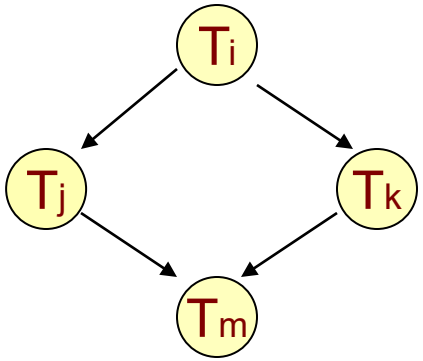
Does not contain cycle



Contains cycles

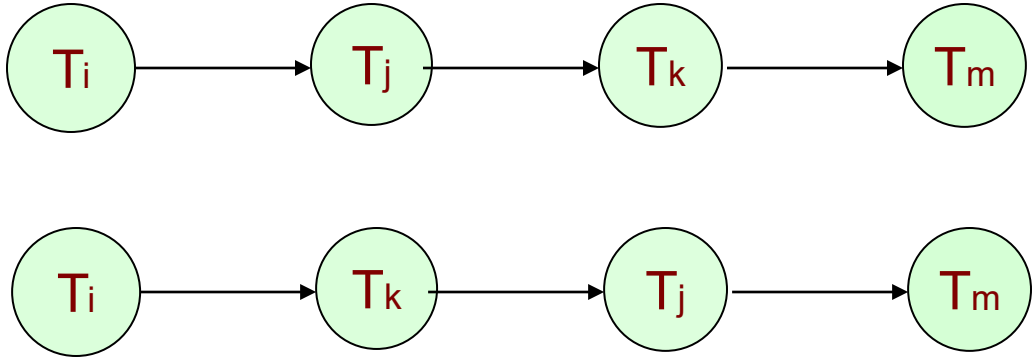
Precedence Graph

- $w_i[a]\ r_k[a]\ r_i[b]\ w_j[b]\ w_k[c]\ r_m[c]\ w_j[d]\ r_m[d]$



Precedence graph

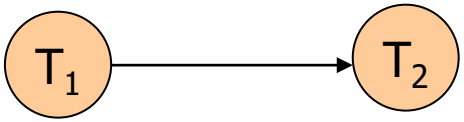
Two possible serialization orders



Precedence Graph

- E.g. Consider again the schedule H_8:

The Precedence graph is:



T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

X

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

Conflict Operation

Y

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

No Conflict Operation

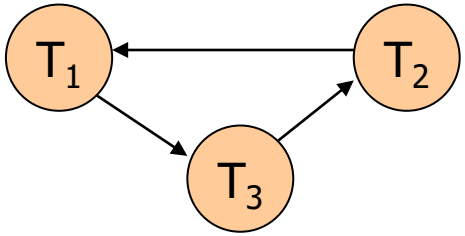
H_8 is conflict serializable and is conflict equivalent to T_1,T_2

Precedence Graph

- E.g. Consider the schedule H_{11}:

T_1:	R(X),			R(Y)
T_2:		W(Y),	R(X),	
T_3:			W(X),	

The Precedence graph is:



X

T_1:	R(X),			R(Y)
T_2:		W(Y),	R(X),	
T_3:			W(X),	

Conflict Operation? YES

Y

T_1:	R(X),			R(Y)
T_2:		W(Y),	R(X),	
T_3:			W(X),	

Conflict Operation? YES

H_{11} is NOT conflict serializable as the Precedence graph contains a cycle

Outline

- Transaction
- ACID Property
- Serial Schedule
- Conflict Serializability
- View Serializability

View Serializability

- There are weaker conditions than conflict serializability that also guarantee serializability
- E.g. Consider the schedule H_1 and H_2:

H_1:

T_1:			R(A)		W(B)		
T_2:	R(B)	W(A)				W(B)	
T_3:				R(A)			W(B)

H_2:

T_1:				R(A)	W(B)		
T_2:	R(B)	W(A)	W(B)				
T_3:						R(A)	W(B)

View Serializability

- *Equivalent*: same effects
- The effects of a history are the values produced by the *write* operations of *unaborted* transactions.
- A schedule is view serializable if it is view equivalent to a serial schedule.

View-serializability – Both schedules read and write the same data values ("view" the same data values).

View Equivalent

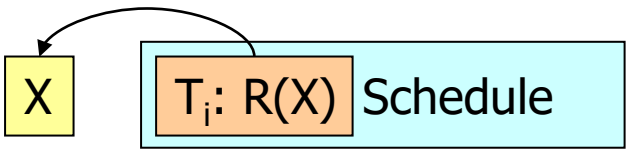
- Two schedules S_1 and S_2, where the same set of transactions participates in both schedules. They are said to be **view equivalent**
  1. If T_i <u>reads</u> the *initial* value of a data item in S_1, T_i also <u>reads</u> the *initial* value of the item in S_2.
  2. If T_i reads an item produced by T_j in S_1, T_i also reads the item produced by T_j in S_2.
  3. If T_i <u>writes</u> the *final* value of a data item in S_1, T_i also <u>writes</u> the *final* value of the item in S_2.

View Equivalent

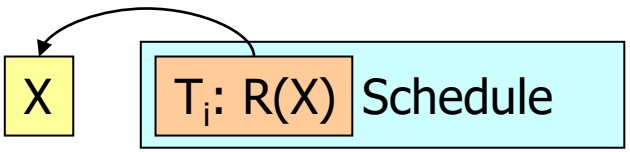
S_1 and S_2 are view equivalent

1

S_1: Initial Read

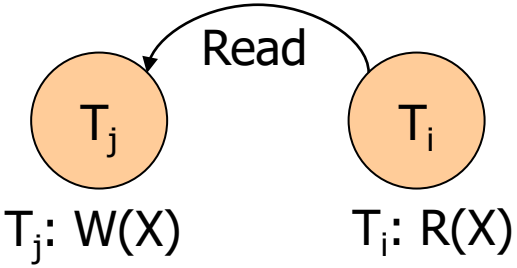


S_2: Initial Read

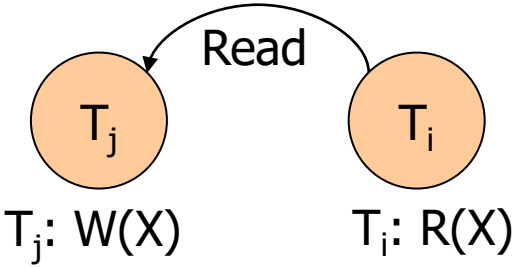


2

S_1:

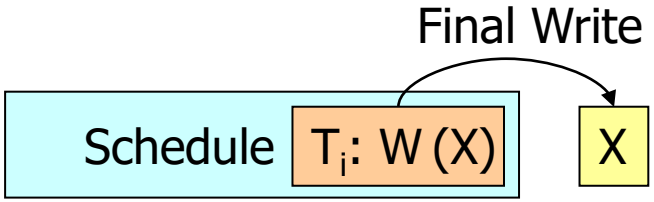


S_2:

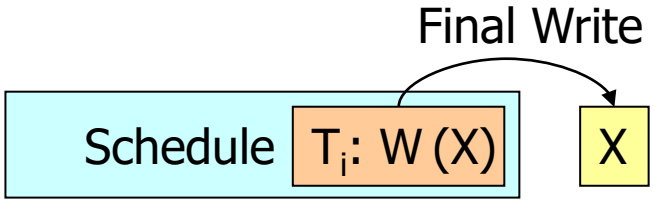


3

S_1:



S_2:



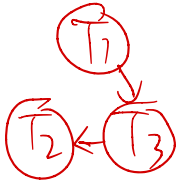
View Serializability

- S is **view serializable** if it is view equivalent to **a serial schedule**
- Suppose S_1 are view equivalent to S_2.
- S_2 is a serial schedule.
- In other words,

  S_1 is view equivalent to a serial schedule

  S_1 is said to be **view serialable**.

View Serializability



- E.g. Consider the schedule H_5:

H_5:

T_1:	R(X),				R(Y)
T_2:		R(Y),		R(X),	
T_3:			W(X),		

- Is it view serializable?

Consider a serial schedule H_6 :$T_1T_3T_2$

H_6:

T_1:	R(X),	R(Y),			
T_2:				R(Y),	R(X)
T_3:			W(X),		

View Serializability

Read (Point 1 and Point 2)

- E.g.

X

H_5:

T_1:	R(X),			R(Y)
T_2:		R(Y),	R(X),	
T_3:			W(X),	

H_6:

T_1:	R(X),	R(Y),		
T_2:			R(Y),	R(X)
T_3:			W(X),	

Y

H_5:

T_1:	R(X),			R(Y)
T_2:		R(Y),	R(X),	
T_3:			W(X),	

H_6:

T_1:	R(X),	R(Y),		
T_2:			R(Y),	R(X),
T_3:			W(X),	

View Serializability

Write (Point 3)

- E.g.

X

H_5:

T_1:	R(X),				R(Y)
T_2:		R(Y),		R(X),	
T_3:			W(X),		

H_6:

T_1:	R(X),	R(Y),			
T_2:				R(Y),	R(X)
T_3:			W(X),		

Y

H_5:

T_1:	R(X),				R(Y)
T_2:		R(Y),		R(X),	
T_3:			W(X),		

H_6:

T_1:	R(X),	R(Y),			
T_2:				R(Y),	R(X)
T_3:			W(X),		

View Serializability

- Answer
  - Yes, it is view serializable because it is view equivalent to

    $T_1T_3T_2 = R_1(X),R_1(Y),W_3(X),R_2(Y),R_2(X)$

View Serializability

- E.g.Consider the schedule H_7:

H_7:

T_1:	W (X),				R (Y)
T_2:		R(Y),	R (X),	W (Y),	

  - Is it view serializable?

Serializability in Practice

- In practice, a DBMS does not test for serializability of a given schedule.
- The approach take by the DBMS is to use specific protocols that are known to produce serializable schedules.
- These protocols could reduce the concurrency but eliminate conflicting cases.

Summary

- What is a transaction?
- The ACID Property
- What is a schedule, a serial schedule?
- What is conflict serializability?
- What is view serializability?
- Strict Two-Phase Locking