Database Management Systems

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Lecture Session-16

Concurrent Transactions and Schedules

innovate achieve lead

Content

- *Concurrent Transactions*
- *Transaction Schedule*
- *Serial and Concurrent Schedules*
- *Need for Concurrency Control*
- *Conflicting Operations*
- *Conflict Equivalent Schedule*
- *Test for Conflict Serializability*
- *View Equivalent Schedule*
- *View Serializability*

Introduction

❖ Multiprogramming in modern systems increases the throughput drastically, as the resources are shared by more than one process.

❖ Similarly in a DBMS multiple transactions are executed concurrently.

A transaction is a collection of operations that perform a single logical operation or function in a database application. Each transaction is a unit of *atomicity*.

❖ Here, for transactions we consider data items as resources because transactions process data by accessing them.

❖ When multiple transactions access data elements in a concurrent way, this may destroy the consistency of the database.

Transaction Schedule

The descriptions that specify the execution sequence of instructions in a set of transactions are called as *schedules*.

Hence schedule can describe the execution sequence of more than one transaction.

Here, in the above schedule the transactions T_1 & T_2 are executed in a serial manner i.e., first all the instructions of the transaction T_1 are executed, and then the instructions of T_2 are executed. Hence the above schedule is known as *serial schedule*.

T_1	T_2
Read (A) A = A + 50 Read (B) B = B + A Write (B)	Read (B) B = B + 75 Write (B)

T_1 and T_2 are transaction

Read (A) – Reads data item A

Write (B) – Writes the data item B

In a serial schedule, instructions belonging to one single transaction appear together.

A serial schedule does not exploit the concurrency. Hence, it is less efficient.

If the transactions are executed concurrently then the resources can be utilized more efficiently hence more throughput is achieved.

A *serial schedule* always results in correct database state that reflect the real world situations.

When the instructions of different transactions of a schedule are executed in an interleaved manner use call such schedules are called *concurrent schedules*.
This kind of concurrent schedules may result in incorrect database state.

Time ↓

T_1	T_2
Read (A)	
A = A + 50	
	Read (A)
	A = A + 100
Write (B)	
	Write (A)

Why Concurrency control is needed?

- **The Lost Update Problem**
  This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

- **The Temporary Update (or Dirty Read) Problem**
  This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 17.1.4).
  The updated item is accessed by another transaction before it is changed back to its original value.

- **The Incorrect Summary Problem**
  If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

It is desirable that a schedule, after execution must leave the database in a consistent state.

The result of a concurrent execution must be same as the result of executing the transactions in serial way.

A concurrent schedule whose result is same as that of a serial schedule is called as *concurrent serializable schedule*.

Conflicting Operations

For transactions T_1 & T_2 the order of read operation on any data element does not matter.

{$T_1R(Q)$, $T_2R(Q)$} or {$T_2R(Q)$, $T_1R(Q)$} does not matter.

The result is same and does not lead to any conflict.

Here, Q is the data element.

But {$T_1R(Q)$, $T_2W(Q)$} is not same as {$T_2W(Q)$, $T_1R(Q)$}

If I_i and I_j are the operations (instructions) two different transactions on the same data item, and at least one of these instructions is a WIRTE operation then we say that I_i and I_j are *conflict operations*.

Here, *I* stands for instruction and *i* and *j* are transactions.

Hence it is evident that if we swap non-conflicting operations of a concurrent schedule, it will not affect the final result.

Look at the following example.

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

(S_1)

Concurrent schedule with T_1 & T_2 accessing A, B (data item)

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

(S_2)

Swap W(A) in T_2 with R(B) in T_1 (because they are non conflicting)

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

(S_3)

Swap R(A) of T_2 with R(B) of T_1 and W(A) in T_2 with W(B) in T_1 (Since non conflicting)

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

(S_4)

Swap R(A) of T_2 with W(B) of T_1

Now, the final schedule is a serial schedule

Conflict Equivalent Schedules

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*.

Further, we say that a schedule S is *conflict serializable* if it is conflict equivalent to a serial schedule.

In our example S_4 in the above example is a serial schedule and is conflict equivalent to S_1. Hence S_1 is a conflict serializable schedule.

T_1	T_2
R(θ)	
	W(θ)
W(θ)	

In this schedule we cannot perform any swap between instructions of T_1 and T_2. Hence it is not conflict serializable

Test for Conflict Serializability

Let S be a schedule.

We construct a precedence graph.

Each transaction participating in the schedule will become a vertex.

The set of edges consist of all edges $T_i \rightarrow T_j$ for which one of the following three conditions hold-

T_i executes W(Q) before T_j executes R(Q)
T_i executes R(Q) before T_j executes W(Q)
T_i executes W(Q) before T_j executes W(Q)

T_0	T_1
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
W(B)	

Infact their schedule is non conflict serializable

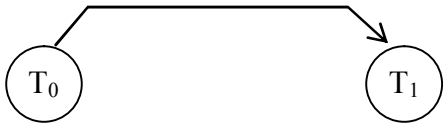
T_1 writes(A) after T_0 writes(A) hence we draw on edge from $T_0 \rightarrow T_1$
T_1 reads(B) before T_0 write (B) hence we can draw an edge from $T_1 \rightarrow T_0$

At any moment of time, while developing the graph in the above manner, if we see a cycle then the schedule is *not conflict serializable*. If no cycles at the end, then it is *conflict serializable.* Hence the above schedule is not serializable.

Now let us consider the following transaction which is *conflict serializable* and discussed earlier.

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

Now, let us draw a precedence graph for the above schedule –
To write A before T_1 reads A. hence we have $T_0 \rightarrow T_1$.



We have only one edge in this graph, and no cycles. Hence it is *conflict serializable*.

Serial schedule:

- A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule.
- Otherwise, the schedule is called nonserial schedule.

Serializable schedule:

- A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.

Result equivalent:

- Two schedules are called result equivalent if they produce the same final state of the database.

Conflict equivalent:

- Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

Conflict serializable:

- A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.

- Being serializable is not the same as being serial
- Being serializable implies that the schedule is a correct schedule.
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.

View Equivalent Schedules

Two schedules *S* and *S′* (where same set of transactions participate in both schedules), are said to be *view equivalent* if the following three conditions are met.

For each data item Q, if the transaction T_i reads the initial value of Q in S, then transaction T_i must in schedule S′, also read the initial value of Q.

For each data item Q, if transaction T_i executes *read (Q)* in S, and the value produced by transaction T_j (if any) then transaction T_i must in schedule S also read the value of Q that was produced by transaction T_j.

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′.

Now, we say that a schedule S is ***view serializable*** if it is *view equivalent* to a serial schedule.

Note: Every conflict serializable schedule is view serializable. But not all view serializable schedules are conflict serializable.

Summary

- ✓ *What are concurrent Transactions*
- ✓ *What are Serial and Concurrent Schedules*
- ✓ *Why Concurrency Control needed in DBMS*
- ✓ *Conflict Equivalent Schedule and its importance*
- ✓ *Test for Conflict Serializability*