Module 11 – **Transaction Management**

- Transactions and ACID
- Atomicity and Durability
- Transactions and schedules
- Concurrent transaction execution
- Schedules – aborted transactions
- Classes of transaction schedules
- Lock based concurrency

- Database ‘objects’ are the units in which programs read or write information (units could be pages, records)

Transaction

- A series of reads and writes of database objects

Read

- Object is brought into main memory (frame in the buffer pool) from disk, and its value is copied into a program variable

Write

- An in-memory copy of the database object is first modified and then written to disk

4 Important Transaction Properties (ACID):

1. **Atomicity**:

2. **Consistency**:

3. **Isolation**:

4. **Durability**:

4 Important Transaction Properties (ACID):

1. **Atomicity**: Each transaction execution must carry out all or none of the actions. There can not be any incomplete transactions

2. **Consistency**:

3. **Isolation**:

4. **Durability**:

4 Important Transaction Properties (ACID):

1. **Atomicity**: Each transaction execution must carry out all or none of the actions. There can not be any incomplete transactions

2. **Consistency**: Each transaction runs by itself maintaining database consistency (responsibility of the user)

3. **Isolation**:

4. **Durability**:

4 Important Transaction Properties (ACID):

1. **Atomicity**: Each transaction execution must carry out all or none of the actions. There can not be any incomplete transactions

2. **Consistency**: Each transaction runs by itself maintaining database consistency (responsibility of the user)

3. **Isolation**: Transactions must be isolated, or protected, from the effects of concurrently scheduling of other transactions

4. **Durability**:

4 Important Transaction Properties (ACID):

1. **Atomicity**: Each transaction execution must carry out all or none of the actions. There can not be any incomplete transactions

2. **Consistency**: Each transaction runs by itself maintaining database consistency (responsibility of the user)

3. **Isolation**: Transactions must be isolated, or protected, from the effects of concurrently scheduling of other transactions

4. **Durability**: After the transaction completes successfully, the effects must persist despite a system crash and before all changes are written on disk

Atomicity and Durability

Transactions can be incomplete due to:

- Unsuccessful termination/aborted
  - some anomaly arises during execution (can be restarted)
- System crash (power supply)
- Unexpected situation (read an unexpected data value or be unable to access some disk)

DBMS ensures transaction atomicity by undoing the actions of incomplete transactions

A log writes completed transactions to disk when system restarts

Transaction can also be defined as a set of actions that are **partially** ordered

R(O) - transaction reading an object O

W(O) - transaction writing an object O

Abort - terminate and undo all the actions carried out thus far

Commit - complete successfully

Schedule

- A list of actions (reading, writing, aborting, or committing)

- Order in which two actions of a transaction T appear in a schedule must be the same as the order in which they appear in T

A **schedule** represents an actual or potential execution sequence

- As seen by the DBMS

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Complete schedule

- Has an abort or a commit

for each transaction whose actions are listed

Serial schedule

- Transactions are executed from start to finish, one by one
- Different transactions are not interleaved

Concurrent transaction execution

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Why have concurrency?

System throughput

- Overlapping I/O and CPU activity reduces the amount of time disks and processors are idle

Response time

- Interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly

Concurrent transaction execution

- Each transaction must preserve database consistency
- **Consistent database state** is defined during design phase
  (e.g.: employee salaries must be <80% of department budget)
- When a complete serial schedule is executed against a consistent database, result is a consistent database

Anomalies Associated with Interleaved Execution:

- Two committed transactions could run against a consistent database and leave it in an inconsistent state

- Two actions on the same data object **conflict** if at least one of them is a write

Three anomalous situations

```
write-read (WR) conflicts
read-write (RW) conflicts
write-write (WW) conflicts
```

Dirty read: transaction T2 could read a database object A that has been modified by another transaction T1, which has not yet committed

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

A was written by T1 and read by T2 before T1 has completed all its changes and committed.

Unrepeatable Reads (RW Conflicts)

T2 could change the value of an object A that has been read by a transaction T1, while T1 is still in progress

T1	T2
R(A) 5	
	R(A) 5
A++ 6	
	A-- 4

Both T1 and T2 read the same value of A (5), then T1 increments A by 1 (6) and T2 decrements A by 1 (4)

Although T2's change is not directly read by T1, it invalidates T1's assumption about the value of A, which is the basis for some of T1's subsequent actions

Overwriting Uncommitted Data (WW Conflicts)
T2 could overwrite the value of an object
which has already been modified by a transaction T1
while T1 is still in progress

Example: Two salaries of employee
E1 and E2 must be kept equal.

T1 sets E1's salary to $1,000
T2 sets E2's salary to $2,000
T1 sets E2's salary to $1,000
T2 sets E1's salary to $2,000

T1	T2
R(E1)	
R(E2)	
	R(E1)
	R(E2)
E1 = 1000	
	E2 = 2000
E2 = 1000	
	E1 = 2000

Result is not identical to the result of either of the two possible serial executions, and the interleaved schedule is therefore not serializable

Schedules and aborted transactions

- All actions of aborted transactions are to be undone

serializable schedule

- over a set S of transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over the set of committed transactions in S

serializability relies on the actions of aborted transactions being undone completely, which may be impossible in some situations

Schedules and aborted transactions

Account transfer program T1 deducts $100 from account A
Interest deposit program T2 reads the current values of accounts A and B and adds 6 percent interest to each, then commits
T1 is aborted

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Abort	

T2 has read a value for A that should never have been there! schedule is unrecoverable

If T2 had not yet committed we could cascade the abort of T1 and abort T2
(recursively abort any transaction that read data written by T2)

Recoverable schedule

- Transactions commit only after all transactions whose changes they read commit.

Avoid cascading aborts

- Read only the changes of committed transaction and aborting a transaction can be accomplished without cascading the abort

A has value 5
T1 changes A to 6
T2 changes A to 7
T1 now aborts, A becomes 5 again
Even if T2 commits, its change to A is lost!

A concurrency control technique called Strict 2PL can prevent this problem

conflict equivalent - schedules that involve the same actions of the same transactions. Relative order of any two conflicting operations is the same in both schedules

Classes of transaction schedules:

Conflict Serializable - a schedule that is conflict equivalent to some serial schedule

Recoverable - Transactions commit only after all transactions whose changes they read commit. Recovery is possible.

Avoid cascading aborts - Read only the changes of committed transaction and aborting a transaction can be accomplished without cascading the abort

Strict - Transactions can neither read nor write an item X until the last transaction that wrote X has committed or aborted.

Serial -Transactions are executed from start to finish, one by one. Different transactions are not interleaved.

Schedules and aborted transactions

Serial

T1	T2	T3
R(A)		
W(A)		
Commit		
	R(B)	
	W(B)	
	Commit	
		R(C)
		W(C)
		Commit

Schedules and aborted transactions

Serial

T1	T2	T3
R(A)		
W(A)		
Commit		
	R(B)	
	W(B)	
	Commit	
		R(C)
		W(C)
		Commit

Serializable

T1	T2	T3
R(A)		
	R(B)	
		R(C)
W(A)		
	W(B)	
		W(C)
Commit		
	Commit	
		Commit

Schedules and aborted transactions

Serial

T1	T2	T3
R(A)		
W(A)		
Commit		
	R(B)	
	W(B)	
	Commit	
		R(C)
		W(C)
		Commit

Serializable

T1	T2	T3
R(A)		
	R(B)	
		R(C)
W(A)		
	W(B)	
		W(C)
Commit		
	Commit	
		Commit

Conflict Serializable

T1	T2
R(A)	
	R(A)
W(B)	
	W(A)
Commit	
	Commit

Recoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Commit	
	Commit

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Abort	
	Abort

Recoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Commit	
	Commit

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Abort	
	Abort

Unrecoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	Commit
Abort	

Recoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Commit	
	Commit

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Abort	
	Abort

Unrecoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	Commit
Abort	

Avoid Cascading Aborts

Recoverable only

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
Abort	
	Abort

Avoid Cascading Results AND Recoverable

T1	T2
	R(A)
R(A)	
W(A)	
	W(A)
Abort	
	Commit

Lock based concurrency control

A Conflict in a DBMS can be defined as two or more different transactions accessing the same variable and at least one of them is a write operation

Conflict equivalent: schedules that involve the actions of the same transactions and order every pair of conflicting actions of two committed transactions in the same way (operate on the same data object and at least one of them is a write)

conflict serializable: a schedule that is conflict equivalent to some serial schedule.