6: Transaction Management and Concurrency & Recovery

8.1 Explain ACID properties of transaction

⇒ • To understand the properties of transaction, consider a transaction of transferring 100 rupees from account A to account B.
• Let Ti be the transaction that transfers 100 rupees from account A to account B.
• This transaction can be defined as –
1) Read balance of account A
2) Withdraw 100 rupees from account A and write back result of balance update.
3) Read balance of account B
4) Deposit 100 rupees to account B and write back result of balance update.

1) Atomicity

⇒ Transaction must be treated as a single unit of operation.
– That means when a sequence of operation is performed in single transaction, they are treated as a single large operation.
– For e.g., Money transfer in above example
⇒ Suppose some type of failure occurs after write(A) and before write(B) then system may loge 100 rupees in calculation which may cause error as sum of original balance (A+B) in accounts A and B is not preserved. In such cases database should automatically restore original value of data items.

2) Consistency

⇒ Consistency is a state in database in which all valid data will be written to database.

- If transaction violates some consistency rule, the whole transaction will be rolled back and the database will be restored to its previous consistent state with those rules.
- On the other hand, if transaction is executed successfully then it will take database from one consistent state to another consistent state.
- For E.g., Money transfer in above example

⇒ Initially total balance of A is 1000 and B is 5000 so sum of balance is both accounts is 6000 while carrying out above transaction some type of failure occurs after write(A) and before write(B) then system may loge 100 rupees in calculation. As now sum of balance in both accounts is 5900 (which should be 6000) which is not consistent result which introduce inconsistency in database. This means that during a transaction the database may not be consistent.

3) Isolation

⇒ Isolation property ensures that each transaction must remain unware of other concurrently executing transactions

- Isolation property keeps multiple transaction separated from each other until they are completed.
- Operations occuring in a transaction are invisible to other transactions until the transaction commits or rolls back.
- For example, when a transaction changes a bank account balance other transaction cannot see the new balance until the transaction commits.

4) Durability

⇒ The result of the transaction that has been successfully committed to the database will remain unchanged even after database fails.

- Changes made during a transaction are permanent once the transaction commits.
- Even if the database server fails in between transaction, it will return to a consistent state when it is restarted.
- The database handles durability by transaction log.

Q.2 Transaction state diagram.

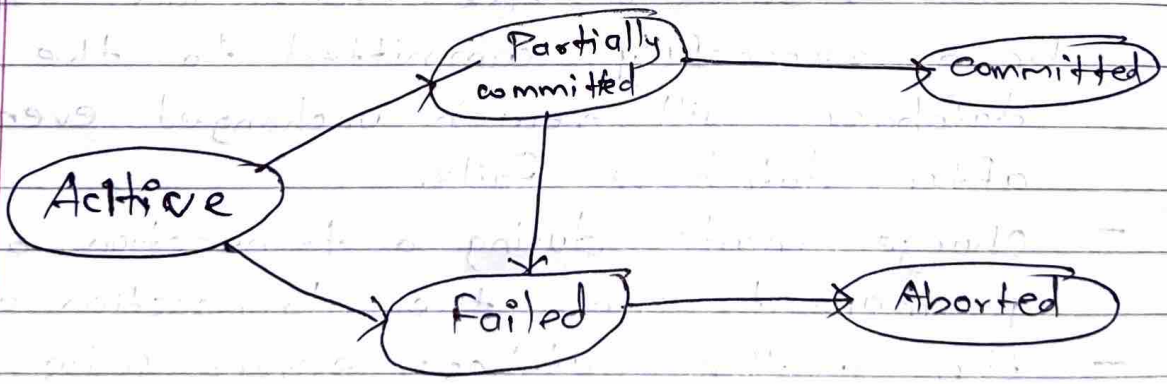
⇒ i) If transaction complete successfully it may be saved on database server called as committed transaction.

ii) Transaction may not always complete its transaction execution successfully. Such transaction must be aborted.

iii) A ~~dbo~~ aborted transaction must not have any changes that aborted transaction made to the database. Such changes must be rolled back

iv) Once a transaction committed, we cannot undo its effects by aborting it.

v) A transaction must be in one of the following:



a) Active

⇒ This is initial state of transaction execution.

- As soon as transaction starts it is in active state.
- Transaction remain in this state till transaction finishes.

b) Partially Committed

⇒ As soon as operation in transaction is executed transaction goes to the partially committed state.

- At this condition the ~~comp~~ transaction has completed its execution and ready to commit on database server. But, it is still possible that it may be aborted, and thus a hardware failure prohibit its successful completion.

c) Failed

⇒ A transaction enters the failed state after the system determines that transaction can no longer proceed with its normal execution.

d) Aborted

⇒ Failed transaction must be rolled back, then it enters the aborted state.

- In this stage system have two options:

1) Restart the transaction

⇒ A restarted transaction is considered to be a new transaction which may recover from possible failure.

2) Kill the transaction

⇒ Because the bad input or because the desired data were not presend in database an error can occurs. In this

we can kill transaction to recover from failure.

e) Committed

⇒ When last data item is written out, the transaction enters into the committed state.

- This state occurs after successful completion of transaction.
- A transaction is said to have terminated if has either committed or aborted.

Q.3 Explain concept of serializability with its type.

⇒ i) When multiple transactions are running concurrently then there is a possibility that the database may be left in the inconsistent state.

ii) A serializable schedule is the one that always leaves the database in consistent state.

iii) Serializability is the concept that helps us to check which schedules are serializable.

iv) A serial schedule is serializable schedule because in serial schedule, a transaction only starts when the other transactions finished execution.

v) A non-serializable schedule of n number of transaction is said to be serializable, if it is equivalent to the serial schedule of those n transactions

vi) Various forms of serial schedules are -

a) Conflict Serializability
b) View Serializability

a) Conflict Serializability

⇒ Two schedules are said to be in conflict serializability if one schedule can be converted into other schedule after swapping non-conflicting operations

- Consider schedule S has two consecutive instructions I_i and I_j from transactions T_i and T_j, respectively.
- IF I_i and I_j access to different data items then they will not conflict and can be swapped, without any problem.
- IF I_i and I_j access to same data item D then -

i) I_i = READ(D), I_j = READ(D), then no conflict

ii) I_i = READ(D), I_j = WRITE(D) then they conflict and cannot be swapped.

iii) I_i = WRITE(D), I_j = READ(D), then they conflict and cannot be swapped.

iv) I_i = WRITE(D), I_j = WRITE(D), then they conflict and cannot be swapped.

- So we can say that instruction conflict if both consecutive instructions operate on same data item and one of them is WRITE operation.

- For e.g.,

1) W(X) of T_1 and R(X) of T_2
⇒ Both have same data item and one of them is a write operation. Hence, conflict.

2) W(X) of T_1 and W(Y) of T_2
⇒ Both are write operations but they have different data items. Hence, non-conflict.

b) View Serializability
⇒ A schedule is said to view serializable if it is view equivalent to serial schedule.
- Two schedules said to view serializable if they satisfy following conditions -
Let D = data item,
S_1, S_2 = Transaction schedule,
T_i, T_j = database transaction.
i) If T_i reads initial value of D in S_1, then T_i also reads value of D in S_2.
ii) If T_i reads value of D written by T_j in S_1, then T_i also reads value of D written by T_j in S_2.
iii) If T_i writes final value of D in S_1, then T_i also writes final value of D in S_2.

Q.4 Explain TCL command.

⇒ i) TCL commands are used to manage the changes made to the data in a table by DML ~~com~~ statements.

ii) TCL commands are -

a) COMMIT Command

⇒ COMMIT command is used to permenanetly save any transaction into the database.

- When we use DML commands like INSERT, DELETE OR UPDATE the changes made by these commands are not permanent until the current session is closed, the changes made by these commands can be rolled back.
- To avoid that, we use the COMMIT command to mark the changes as permanent.
- It's syntax is -

```
COMMIT
```

b) SAVEPOINT Command

⇒ SAVEPOINT command is used to temporarily save a transaction so that you can roll back to the point whenever required.

- It's syntax is -

```
SAVEPOINT savepoint_name;
```

c) ROLLBACK Command

⇒ This command restores the database to the last committed state.

- It is also used with SAVEPOINT command to ~~make some changes~~ jump to a savepoint in an ongoing transaction.
- It's syntax is -

```
ROLLBACK
   OR
ROLLBACK TO savepoint_name ;
```

- For e.g.;

We have table class

⇒

id	name
1	A
2	B
4	D

```
INSERT INTO class VALUES (5, 'X');
COMMIT;

UPDATE class SET name = 'C' WHERE id=5;
SAVEPOINT A;

INSERT INTO class VALUES (6, 'Y');
SAVEPOINT B;

INSERT INTO class VALUES (7, 'Z');
SAVEPOINT C;
```

```
SELECT * FROM class;
```

⇒

id	name
1	A
2	B
4	D
5	C
6	Y
7	Z

```
ROLLBACK TO B;
SELECT * FROM class;
```

⇒

id	name
1	A
2	B
4	D
5	C
6	Y

```
ROLLBACK TO A;
SELECT * FROM class;
```

id	name
1	A
2	B
4	D
5	C

8.5 Explain deadlock and explain deadlock handling in DBMS with example.

⇒ i) A system is said to be in deadlock if there exists a set of transaction such that every transaction in the set is waiting for another transaction to complete its execution

- Example:

Consider two transactions as follows –

T_1 : This transaction first reads and then write data on data item X, and then read & write on data item Y.

T_1
Read(X)
Write(X)
Read (Y)
Write(Y)

T_2 : This transaction first reads and writes data on Y and then on X data item

T_2
Read (Y)
Write(Y)
Read (X)
Write (X)

- Consider above two transaction are executing using locking protocol as follows :

T_1	T_2	
LOCK-X(X)		
READ(X)		
WRITE(X)		
	LOCK-X(Y)	
	READ(Y)	
	WRITE(Y)	
LOCKX-(Y)		Wait for T_2 to unlock Y
READ(Y)		
WRITE(Y)		
	LOCKX-(X)	Wait for T_1 to unlock X
	READ(X)	
	WRITE(X)	

- So in above schedule, T_1 is waiting for T_2 to unlock data-item Y and T_2 is waiting for T_1 to unlock X.
- So system is in a deadlock state as there are ~~to~~ set of transactions T_1 and T_2 such that, both are waiting for each other transaction to complete.
- This state is called as Deadlock state.

- There are two principles methods to handle deadlock in system.

1) Deadlock Prevention

⇒ Deadlock prevention protocols ensures the system will never enter into a deadlock state.

- Various deadlock prevention techniques using timestamps as follows -

a) Wait - die

⇒ This is non-preemptive technique of deadlock prevention.

- When Ti wants to hold data item, currently hold by Tj, then Ti is allowed to wait if and only if it is older than Tj othewise Ti is rolled back.

b) Wound - wait

⇒ This is preemptive technique of deadlock prevention.

- When Ti wants to hold data item, currently hold by Tj, then Ti is allowed to wait if and only if Ti is younger than Tj othewise Tj is rolled back.

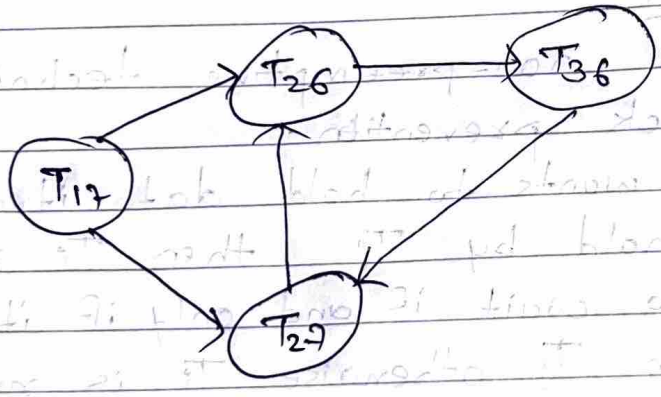
2) Deadlock Detection and Recovery

→ Deadlocks can be described as a wait-for-graph, which consists of a pair $G = (V, E)$.

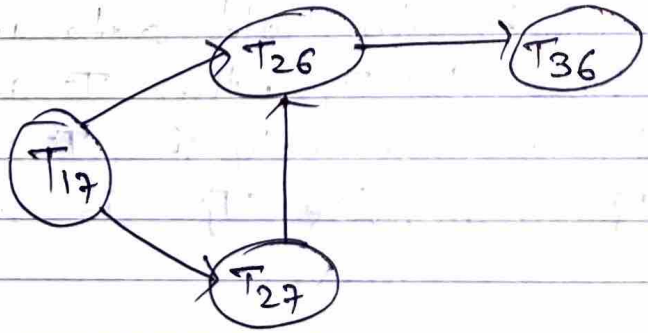
where V = set of vertices

E = set of edges

- If cycle is present in wait-for-graph then deadlock is present and transaction in cycle is deadlock.



- If no cycle is present it means no deadlock in system.



- Recovery From Deadlock

⇒ When deadlock is detected in the system, then system should be recovered using recovery schemes.

- Deadlock Recovery Methods:

1) Selection of victim
2) Rollback
3) Starvation.