Transaction

Concepts

Properties of Transaction(ACID)

Atomicity : All or nothing

- Transaction must done completly or not
    - So, if crash occur during transaction, then rollback all progress

Cosistency : Integrity constraint

- After transaction, DB should satisfy integrity constraint
    - That means, should correctly reflect property of real world's data

Isolation : Independent

- Should not be interfered by other transactions

Durability : Safe save

- Result of compeleted transactions should permanently affect to DB data

Transaction States

Active

- Initial & Intermediate State
- State during execute transaction

Partially committed

- Intermediate State
- The final statement has been executed
- State before final checking process(ACID ensuring process)

Failed

- Intermediate State
- Aborted during transaction
- Rollback or other process is still executing(can't ensure correcness of DB)

Aborted

- Completed falied state
- completely Rolled back and restored

Commited

- Completed successful state
- Completely executing transaction

Schedules

- Indicate sequence of instructions as chronological order
- Simple view of transaction
    - only considering **Read** and **Write** operation
    - ignore other operation

Serializability

- Basic Assumption : each transaction preserve data consistencty(no currption due to wrong transaction)
    - So, **serial execution** always preserve database consistency
- If, any schedule **equivalent** to serial schedule, we could say that schedule as serializable schedule
    - **equivalance** mean same output for any instance of same schema

Conflict Serialiability

- Schedule S is **confilict serializable** if it is **conflict equivalent** to a serial schedule
    - **conflict equivalent** mean that schdule A and S can be transformed by swapping series of **non-conflicting** instruction
    - **non-conflicting** instruction : instruction pair that doesn't effect to result even though switching theirs execution sequence
        - Only for **read-read** pair

Testing Serializabiliy

- Using **Precedence graph** to test serializability
    - a direct graph that visualize executiin sequence of transaction
    - draw a arc from T_i to T_j if, T_i and T_j have conflicting insturction and T_i insturction call more ealier
- If, precedence graph is acyclic(non cycle graph) we can transform transaction to serial execution by topological sorting

Recovery

- Recoverability
    - if affected transaction commit before original transaction committed
- Cascading Rollback : roll back all effected transaction

- Cascadeless Schedules : prevent cascading rollback
    - to avoid it, must **read after commit**!
    - by prevent cascading rollback, every cascadeless schedule is **recoverable**