Timestamps:-

→ With each tramaction T_i in the system associated a unique fixed timestamp denoted by $TS(T_i)$.

→ This timestamp is assigned by the database system before the tramaction T_i starts execution.

→ The time stamps of the tramactions determine the serializability order. If $TS(T_i) < TS(T_j)$ then the system must ensure that the produced schedule is equivalent to a serial schedule in which tramaction T_i appears before tramaction T_j.

To implement this scheme, we associate with each data item Q with two timestamp values.

(1) W-timestamp(Q):- denotes the largest timestamp of any tramaction that executed write(Q) succesfully.

(2) R-timestamp(Q):- denotes the largest timestamp of any tramaction that executed read(Q). succesfully.

These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

Time stamp-ordering protocol

→ The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

→ This protocol operates as follows.

Step I (I) suppose that transaction T_i issues read(Q)

→ (i) If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i needs to read a value of Q, that was already overwritten.

→ Hence the read operation is rejected and T_i is rolled back.

(ii) If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the read operation is executed and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and $TS(T_i)$.

Step II:- suppose that transaction T_i issues write(Q).

(i) If $TS(T_i) < R\text{-Timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that the value would never be produced.

Hence the system rejects the write operation and rolls T_i back.

(ii) If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q. (3)

Hence the system rejects the write operation and rolls T_i back.

(iii) Otherwise, the system executes the write operation and sets W-timestamp(Q) to $TS(T_i)$.

→ If a transaction T_i is rolled back by the concurrency control scheme as a result of issuance of either read or write operation, the system assigns it a new timestamp and restarts it.

→ Timestamp ordering protocol ensures conflict serializability.

→ This is because conflicting operations are processed in timestamp order.

→ This timestamp ordering protocol ensures freedom from deadlock. since no transaction never waits.

Thomas Write Rule:-

→ This modification to the timestamp-ordering protocol, called Thomas' write rule,

→ suppose that transaction T_i issues write(Q).

(1) If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was previously needed, and it had been assumed that the value would never be produced.

→ Hence, the system rejects the write operation

(2) If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q.

→ Hence this write operation can be ignored.

(3) Otherwise, the system executes the write operation and sets W-timestamp(Q) to $TS(T_i)$.

~~→ The difference between these rules and~~

→ The timestamp ordering protocol requires that T_i be rolled back if T_i issues write(Q) and

$TS(T_i) < W\text{-timestamp}(Q)$.

→ $TS(T_i) \geq R\text{-timestamp}(Q)$, we ignore the obsolete write.