WEEK 10

TRANSACTION – TIME STAMPING

STUDENT OBJECTIVES

- Upon completion of this video, you should be able to:
  - Given several transactions and read and write operations, determine the timestamps that would be given out.
  - Determine when a timestamp would cause an transaction to be rolled back.
  - Describe the 3 steps in Optimist checking for handling databases
  - Identify the situations when Optimistic checking would be suitable.

TIME STAMPING

- Each transaction gets a UNIQUE GLOBAL TIMESTAMP.
- The timestamp inflicts an order on the transactions.
- This method **DOES NOT** use locks, therefore deadlocks cannot occur, however cyclic restart (abort, restart) may occur
- Timestamps are always UNIQUE and MONOTONICLY increasing, each transaction has a time stamp start time → TS(T)

- All reads and writes in a transaction have the same timestamp. For example you might have READTS(X) or WRITETS(X).
  - READTS(X) is the largest timestamp among all the timestamps of transactions that have read X thus READTS(X) = TS(T) where T is the youngest transaction.
  - WRITETS(X): same rule as above
- Example:
  - T1 starts at 11:00, gets a TS(T1) → 1100
  - T2 starts at 11:12, gets a TS(T2) → 1112
  - T3 starts at 11:14, gets a TS(T3) → 1114
  - T1 reads X, so READTS(x) → 1100
  - Then T2 reads X, so READTS(X) changes to 1112
  - T3 wants to write to X, so WRITETS(X) → 1114

BASIC TIME STAMP ALGORITHM:

- Transaction T tries to **write** to X
  - If READTS(X) > TS(T) or if WRITETS(X) > TS(T) then abort and roll back T and reject the operation. (because a transaction younger than T has already read or written X before T has a chance to) an operation cannot happen in past.
  - If the above condition is false, then execute the write of X and set WRITETS(X) = TS(T) update TS.
- Transaction T tries to **read** X
  - if WRITETS(X) > TS(T) then abort and roll back T (because some younger transaction with timestamp after T has already written to X before T could read it.)
  - If the above condition is false, then execute the read of X and set READTS(X) to the larger of TS(T) and the current READTS(X)

- The DBMS executes transactions in order, if 2 transactions have conflicting operations, one is stopped, rescheduled and given a new timestamp. If T1 is aborted and rolled back, then any transactions that may have used a value written by T1 must also be rolled back
- Problems:
  - May require *cascading rollback*
  - Each value in the database requires 2 additional timestamp fields → WRITETS and READTS thus increases memory used and processing overhead
  - May be lots of overhead because of stopping and restarting transactions

OPTIMISTIC

- Previous strategies checking was done BEFORE the transaction, lots of overhead! What if NO checking was done before we run the transactions?
- Assumption that most transactions do not conflict. No need for locking or time stamping. Transaction can do whatever it wants until commit time.
- 3 Phases: Read, Validation, Write:
  - ***Read***: Transaction reads the database, makes updates to local copies of the data items. All updates are applied ONLY to the local copies.
  - ***Validation:*** (When ready to commit) Transactions are validated to make sure changes will not screw up the consistency, i.e. checks to see if serialization is violated. If validation is positive, the transaction goes to the write phase, if negative then transaction restarted and changes are discarded
  - ***Write:*** The actual database is written to with the changes
- Works best in a database that has few writes and mainly queries