**CHAPTER SEVEN**

**CONCURRENCY CONTROL TECHNIQUES**

**Introduction**

Concurrency control techniques are meant to ensure the isolation property of concurrently executing transactions by ensuring serializability of schedules. In this, serializability is guaranteed by use of protocols (set of rules and procedures or methods). These protocols include: -

(i) Locking protocol

(ii) Time Stamp protocol

(iii) Optimistic protocol

**Terminology**

**Lock:** Reserving a record for use.

**Protocol:** Set of rules (procedure) for doing a certain task.

**Locking protocol/ techniques**

A lock is a variable associated with a data item that describes the status of the item with respect to operations that can be applied to it.

A lock guarantees exclusive use of data item to a current transaction. Transaction T1 does not have access to a data item that is currently used by transaction T2. A transaction acquires a lock prior to data access. The lock is released (Unlock) when the transaction is completed so that another transaction can lock the data item for its exclusive use.

**Types of locks**

**i) Binary lock**

These locks have two types of states or values i.e. locked or unlocked (1 or 0) if the value of the lock on an item is 1 then that item cannot be accessed by a database operation that requests the item if the value of the lock on the item is 0 the item can be accessed when requested. Binary locking uses two operations *lock\_item* and *unlock\_item*.when a transaction accesses an unlocked item it issues a lock\_item operation and when it’s through with the item it issues the unlock\_item operation, which sets the lock to 0.so that the item can be accessed by other transactions. Thus binary locks enforce mutual exclusion on the data item.

**ii) Shared Locks and Exclusive Locks**

**(Write Locks)**

Shared locks are used during read operations since read operations cannot conflict. More than one transaction is permitted to hold read locks simultaneously of the same data item.

Exclusive locks give a transaction exclusive access to a data item. As long as a transaction holds an exclusive lock no other transaction can read or update that data item.

**Deadlocks and starvation**

A deadlock occurs when each transaction T1 in a set of two or more transactions is waiting for some item that is locked by another transaction T2 in the set. Consequently T1 and T2 wait indefinitely each waiting for the other to unlock the required data item. Such a deadlock is known as **deadly embrace**.

**Deadlock avoidance protocols**

This can be done using two-phase locking techniques.

**2-Phase locking**

To ensure serializability the 2- phase locking protocol defines how transaction acquire and relinquish locks. 2-phase locking guarantees serializability but it does not prevent deadlocks. The 2- phases are:

(a) Growing phase in which a transaction acquires all the required locks without unlocking any data. Once all the locks have been acquired the transaction is in its locked point.

(b) Shrinking phase in which a transaction releases all locks and cannot obtain any new lock.

Rules governing the 2-Phase protocol are:

(i) 2 transactions cannot have conflicting locks

(ii) No unlock operation can proceed an unlock operation in the same transaction.

(iii) No data is affected until all locks are obtained i.e. until the transaction is in the locked point.

**Deadlock Prevention**

A transaction requesting a new lock is aborted if there is a possibility that a dead lock can occur. If the transaction is aborted, all the changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlock.

**Deadlock Detection (wait and die)**

The DBMS periodically tests the database for deadlocks. If the deadlock is found one of the transactions (the "victim”) is aborted (rolled back and restarted) and the other transaction continues.

**Starvation**

This problem occurs when a transaction is allowed to process for an indefinite period of time while other transactions are kept waiting for long for an item locked by the processing transaction to be free. This may occur if the locking scheme for locked item is unfair, giving priority to some transactions over others. A solution to this will be the use of fair waiting schemes e.g. first come first served queue.

**Conclusion**

The best deadlock control method depends on the database environment, if the probability is low, deadlock detection is recommended, if probability is high, deadlock prevention is recommended and if response time is not high on the system priority list deadlock avoidance might be employed.

**Time Stamping Method**

The time stamping approach, to schedule concurrent transactions assign a global unique time stamp to each transaction. The time stamp value uses an explicit order in which transactions are submitted to the DBMS. The stamps must have 2 properties;

1. Uniqueness - which assures that no equal time stamp values can exist.
2. Monotonic - which assures that time stamp values always increase.

All database operations read and write within the same transaction must have the same time stamp. The DBMS executes conflicting operations in the time stamp order thereby ensuring serialisability of the transactions.

If 2 transactions conflict, one is often stopped, re-scheduled and assigned a new time stamp value. The main draw back of time stamping approach is that each value stored in the database requires 2 additional time- stamp fields, one for the last time the field was read and one for the last update. Time stamping thus increases the memory needs and the databases.

**Optimistic Methods**

The optimistic approach is based on the assumption that the majority of database operations do not conflict. The optimistic approach does not require locking or time stamping techniques; instead a transaction is executed without restrictions until it is committed. In this approach, each transaction moves through 2 or



3 phases; read, validation and write phase.

**Read Phase**

The transaction reads the database, executes the needed computations and makes the updates to private copy of the database values. All update operations of the transaction are recorded in a temporary update file, which is not accessed by the remaining transactions.

**Validation Phase**

The transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If a validation phase is negative, the transaction is restarted and the changes are discarded.

**Write Phase**

The changes are permanently applied (written) to the database.

**Conclusion**

The optimistic approach is acceptable for mostly read or query database system that require very few update transactions.