ST. XAVIER’S COLLEGE

**Maitighar,Kathmandu**

**(Affiliated to Tribhuvan University)**



**Database Management System**

**Lab Assignment #**

**Submitted By**

Ajita Khatiwada

B.Sc. CSIT

Year II/IV Semester

013BSCIT004

**Submitted To:**

Er. Sanjay Kumar Yadav

Lecturer,

Department of Computer Science

St. Xavier’s College

Maitighar, Kathmandu

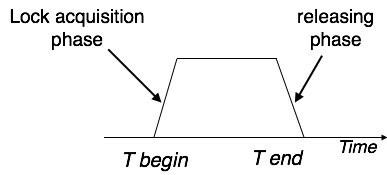
**Submitted On:**

30 october , 2015

**Database concurrency control**

1. **Purpose of concurrency control**
2. **Two phase locking**

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

1. **Limitations of CCMs**
2. **Time-setup based protocols**

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

1. **Commit protocols**

In computer networking and databases, the three-phase commit protocol (3PC) is a distributed algorithm which lets all nodes in a distributed system agree to commit a transaction. Unlike the two-phase commit protocol (2PC) however, 3PC is non-blocking.

1. **Index Locking**

**Index locking** is a technique used to maintain **index** integrity. A portion of an **index** is locked during a database transaction when this portion is being accessed by the transaction as a result of attempt to access related user data.

1. **Lock granularity**

It deals with the cost of implementing locks depending upon the space and time. Here, space refers to data structure in [DBMS](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) for each lock and time refers to handling of lock request and release.

1. **Time setup ordering multi version concurrency control**

Each transaction is issued a timestamp when it enters the system. If an old transaction Ti has time-stamp TS(Ti ), a new transaction Tj is assigned timestamp TS(Tj ) such that TS(Ti )

The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Suppose a transaction Ti issues a read(Q)ν 1. If TS(Ti ) ≤ W-timestamp(Q), then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back.ν 2. If TS(Ti )≥ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to max(R-timestamp(Q), TS(Ti )).

Multiversion schemes keep old versions of data item to increase concurrency. Multiversion Timestamp Orderingλ Multiversion Two-Phase Lockingλ Each successful write results in the creation of a new version of theν data item written. Use timestamps to label versions.ν When a read(Q) operation is issued, select an appropriate version ofν Q based on the timestamp of the transaction, and return the value of the selected version. reads never have to wait as an appropriate version is returnedν immediately.

1. **Deadlock handeling detection and resolution**

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur −

* If TS(Ti) < TS(Tj) − that is Ti, which is requesting a conflicting lock, is older than Tj − then Ti is allowed to wait until the data-item is available.
* If TS(Ti) > TS(tj) − that is Ti is younger than Tj − then Tidies. Ti is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −

* If TS(Ti) < TS(Tj), then Ti forces Tj to be rolled back − that is Ti wounds Tj. Tj is restarted later with a random delay but with the same timestamp.
* If TS(Ti) > TS(Tj), then Ti is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.