**oST.XAVIER’S COLLEGE**

MAITIGHAR, KATHMANDU



Database Management System

Assignment #

Submitted By:

Aabhash Dhakal

013BSCCSIT001

2nd year/ 4th semester

Submitted to:

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer  Department of Computer Science |  |

# Database Concurrency Control

* 1. **Purpose Of Concurrency Control**

The purpose of Concurrency Control is :

* To enforce Isolation (through mutual exclusion) among conflicting transactions.
* To preserve database consistency through consistency preserving execution of transactions.
* To resolve read-write and write-write conflicts.
* In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.
  1. **Two Phase locking**

Locking is an operation which secures

(a) permission to Read

(b) permission to Write a data item for a transaction.

Example:

Lock (X). Data item X is locked in behalf of the requesting transaction.

Unlocking is an operation which removes these permissions from the data item.

Example:

Unlock (X): Data item X is made available to all other transactions.

Lock and Unlock are Atomic operations.

Two locks modes:

(a) shared (read)

(b) exclusive (write).

Shared mode: shared lock (X)

More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.

Exclusive mode: Write lock (X)

Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.

Conflict matrix



Two-Phase Locking Techniques: Essential components

Lock Manager:

Managing locks on data items.

Lock table:

Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.



* 1. **Time Stamp 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**
  2. **Index Locking**

Index locking:

* Every relation must have at least one index.
* A transaction can access tuples only after finding them through one or more indices on the relation
* A transaction *Ti* that performs a lookup must lock all the index leaf nodes that it accesses, in S-mode
* Even if the leaf node does not contain any tuple satisfying the index lookup (e.g. for a range query, no tuple in a leaf is in the range)
* A transaction *Ti* that inserts, updates or deletes a tuple *ti* in a relation *r*
  + - * must update all indices to *r*
      * must obtain exclusive locks on all index leaf nodes affected by the insert/update/delete
  + The rules of the two-phase locking protocol must be observed
  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 for each lock and time refers to handling of lock request and release.

The cost of implementing locks depends on the size of data items. There are two types of lock granularity:

* Fine granularity
* Coarse granularity
  1. **Time Stamp ordering multi-version concurrency control**

he timestamp-ordering protocol ensures serializability among transactions in their conflicting read and writes operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

* The timestamp of transaction Ti is denoted as TS(Ti).
* Read time-stamp of data-item X is denoted by R-timestamp(X).
* Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows −

* **If a transaction Ti issues a read(X) operation −**
* If TS(Ti) < W-timestamp(X)
  + - * Operation rejected.
* If TS(Ti) >= W-timestamp(X)
  + - * Operation executed.
* All data-item timestamps updated.
* **If a transaction Ti issues a write(X) operation −**
* If TS(Ti) < R-timestamp(X)
  + - Operation rejected.
* If TS(Ti) < W-timestamp(X)
  + - Operation rejected and Ti rolled back.
* Otherwise, operation executed.