**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Theory Assignment#11**

**Submitted by:**

**Shreesha Pokharel**

013BSCCSIT038

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav** |  |

Lecturer

Department of Computer Science

Date of submission: 30th October, 2015

**Database concurrency control**

1. **Purpose of concurrency control**

Concurrency control in Database management systems ensures that *database transactions* are performed concurrently without violating the data integrity of the respective databases. Thus concurrency control is an essential element for correctness in any system where two database transactions or more, executed with time overlap, can access the same data, e.g., virtually in any general-purpose database system.

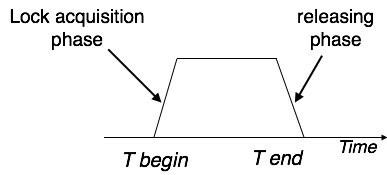
The general purpose of concurrency control:

* + 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.

Most high-performance transactional systems need to run transactions concurrently to meet their performance requirements. Thus, without concurrency control such systems can neither provide correct results nor maintain their databases consistent.

1. **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 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**

In distributed data base and transaction systems a distributed *commit protocol* is required to ensure that the effects of a distributed transaction are atomic, that is, either all the effects of the transaction persist or none persist, whether or not failures occur. Several commit protocols have been proposed in the literature. These are variations of what has become a standard and known as the two-phase commit (2PC) protocol.

1. **Index locking**

In databases an *index* is a data structure, part of the database, used by a database system to effectively navigate access to *user data*. Index data are system data distinct from user data, and consist primarily of pointers. Changes in a database (by insert, delete, or modify operations), may require indexes to be updated to maintain accurate user data accesses.

**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. Additionally, special database system transactions (not user-invoked transactions) may be invoked to maintain and modify an index, as part of a system's self-maintenance activities. When a portion of an index is locked by a transaction, other transactions may be blocked from accessing this index portion (blocked from modifying, and even from reading it, depending on lock type and needed operation).

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**

The 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.

1. **Deadlock handling 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.

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.