**ST. XAVIER’S COLLEGE**

**MAITIGHAR, KATHMANDU**

****

**Database Management System**

**Theory Assignment #12**

**Submitted by:**

Anu Kadel

013BSCIT007

**Submitted to:**

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

Lecturer

Department of Computer Science

St. Xavier’s College, Maitighar

**THEORY ASSIGNMENT#12**

**DATABASE CONCURRENCY CONTROL**

Process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of transactions, is known as concurrency control.

Concurrency control in [Database management systems](https://en.wikipedia.org/wiki/Database_management_system) , ensures that [*database transactions*](https://en.wikipedia.org/wiki/Database_transaction) are performed [concurrently](https://en.wikipedia.org/wiki/Concurrency_%28computer_science%29) without violating the [data integrity](https://en.wikipedia.org/wiki/Data_integrity) of the respective [databases](https://en.wikipedia.org/wiki/Database). 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.

1. **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.

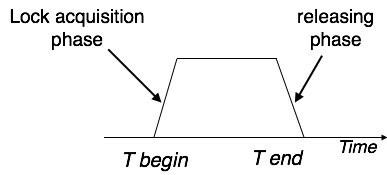
1. **TWO PHASE LOCKING**

In [databases](https://en.wikipedia.org/wiki/Database) and [transaction processing](https://en.wikipedia.org/wiki/Transaction_processing), **two-phase locking** (**2PL**) is a [concurrency control](https://en.wikipedia.org/wiki/Concurrency_control) method that guarantees [serializability](https://en.wikipedia.org/wiki/Serializability). It is also the name of the resulting set of [database transaction](https://en.wikipedia.org/wiki/Database_transaction) [schedules](https://en.wikipedia.org/wiki/Schedule_%28computer_science%29) (histories). The protocol utilizes [locks](https://en.wikipedia.org/wiki/Lock_%28computer_science%29), applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life.

By the 2PL protocol locks are applied and removed in two phases:

1. Expanding phase: locks are acquired and no locks are released.
2. Shrinking phase: locks are released and no locks are acquired.

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. **LIMITATION OF CCMS**

* Threads can be expensive. Overhead of scheduling, context-switching and synchronization.
* Concurrent programs can run slower than their sequential counterparts even with multiple CPUs

1. **TIME-STAMP-BASED PROTOCOL**

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 PROTOCOL**

* Commit protocols are used to ensure atomicity across sites
  + a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  + not acceptable to have a transaction committed at one site and aborted at another
* The *two-phase commit* (2 *PC*) protocol is widely used
* The *three-phase commit* (3 *PC*) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol.

1. **INDEX LOCKING**

Index locking protocol states:

* Every relation must have at least one index. Access to a relation must be made only through one of the indices on the relation.
* A transaction *Ti* that performs a lookup must lock all the index buckets that it accesses, in S-mode.
* A transaction *Ti* may not insert a tuple *ti* into a relation *r*  without updating all indices to *r*.
* *Ti* must perform a lookup on every index to find all index buckets that could have possibly contained a pointer to tuple *ti*, had it existed already, and obtain locks in X-mode on all these index buckets. *Ti* must also obtain locks in X-mode on all index buckets that it modifies.
* The rules of the two-phase locking protocol must be observed.

1. **LOCK GRANULARITY**

Granularity of locking (level in tree where locking is done):

* *fine granularity* (lower in tree): high concurrency, high locking overhead
* *coarse granularity*  (higher in tree): low locking overhead, low concurrency
* A lockable unit of data defines its granularity. Granularity can be coarse (entire database) or it can be fine (a tuple or an attribute of a relation).
* Data item granularity significantly affects concurrency control performance. Thus, the degree of concurrency is low for coarse granularity and high for fine granularity.
* Example of data item granularity:
  + A field of a database record (an attribute of a tuple)
  + A database record (a tuple or a relation)
  + A disk block
  + An entire file
  + The entire database

1. **TIME STAMP ORDERING MULTI VERSION CONCURRENCY CONTROL**

* **Timestamp**
  + A monotonically increasing variable (integer) indicating the age of an operation or a transaction. A larger timestamp value indicates a more recent event or operation.
  + Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.
* Multiversion concurrency control techniques
  + This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. Thus unlike other mechanisms a read operation in this mechanism is never rejected.
  + Side effect:
    - Significantly more storage (RAM and disk) is required to maintain multiple versions. To check unlimited growth of versions, a garbage collection is run when some criteria is satisfied.
* Multiversion technique based on timestamp ordering
  + This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction.
    - Thus unlike other mechanisms a read operation in this mechanism is never rejected.
  + Side effects: Significantly more storage (RAM and disk) is required to maintain multiple versions. To check unlimited growth of versions, a garbage collection is run when some criteria is satisfied.

1. **DEADLOCK HANDLING DETCTION AND RESOLUTON**

When dealing with locks two problems can arise, the first of which being deadlock. Deadlock refers to a particular situation where two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource. Some computers, usually those intended for the time-sharing and/or real-time markets, are often equipped with a hardware lock, or hard lock, which guarantees exclusive access to processes, forcing serialization. Deadlocks are particularly disconcerting because there is no general solution to avoid them.

A fitting analogy of the deadlock problem could be a situation like when you go to unlock your car door and your passenger pulls the handle at the exact same time, leaving the door still locked. If you have ever been in a situation where the passenger is impatient and keeps trying to open the door, it can be very frustrating. Basically you can get stuck in an endless cycle, and since both actions cannot be satisfied, deadlock occurs.

* **Deadlock prevention**
  + A transaction locks all data items it refers to before it begins execution.
  + This way of locking prevents deadlock since a transaction never waits for a data item.
  + The conservative two-phase locking uses this approach.
* **Deadlock detection and resolution**
  + In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.
  + A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle. One of the transaction o
* **Deadlock avoidance**
  + There are many variations of two-phase locking algorithm.
  + Some avoid deadlock by not letting the cycle to complete.
  + That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.
  + Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim.