**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**DATABASE MANAGEMENT SYSTEM**

**THEORY ASSIGNMENT #11\_2**

**Submitted by:**

Rojesh Tamrakar

013BSCCSIT032

**Submitted to:**

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

Lecturer

Department of Computer Science

Date of submission: 30th August, 2015

**Database Concurrency Control**

In information technology and computer science, especially in the fields of computer programming, operating systems, multiprocessors, and databases, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible. Computer systems, both software and hardware, consist of modules, or components. Each component is designed to operate correctly, i.e., to obey or to meet certain consistency rules.

Process of managing simultaneous operations on the database without having them interfere with one another.

• Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.

• Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

1. **Purpose of Concurrency Control**

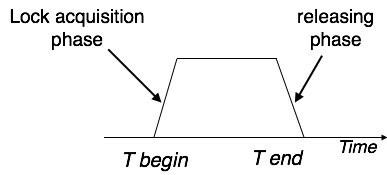
1. Lost updates: This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect. Successfully completed update is overridden by another user.

2. The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction. The reading transaction end with incorrect results.

3. The incorrect summary problem: While one transaction takes the summary over the values of all the instances of a repeated data item, a second transaction updates some instances of that data item. The resulting summary does not reflect a correct result for any precedence order between the two transactions.

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.

1. **Limitations of CCMs**

Concurrency Control is a type of management style where employers or supervisors constantly monitor how employees are working while the work is still in progress. This kind of management makes employees feel like slaves and lowers their morale to work, which lowers production. It also creates a sense of mistrust between the employers and the employees.

1. **Time-stamp-based protocols**

A timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the transaction start time. We will refer to the timestamp of transaction T as TS(T). Concurrency control techniques based on timestamp ordering do not use locks; hence, deadlocks cannot occur.

the algorithm associates with each database item X two timestamp (TS) values are:

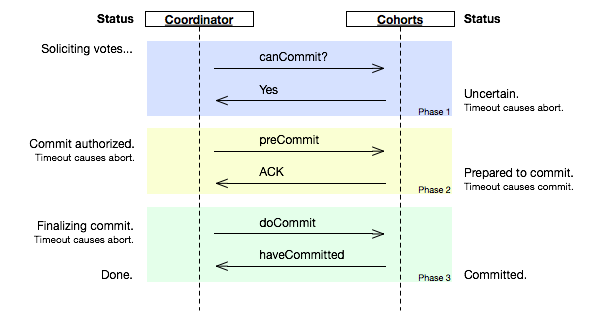
1. Read\_TS(X): The read timestamp of item X; this is the largest timestamp among all the timestamps of transactions that have successfully read item X—that is, read\_TS(X) = TS(T), where T is the youngest transaction that has read X successfully.

2. Write\_TS(X): The write timestamp of item X; this is the largest of all the timestamps of transactions that have successfully written item X—that is, write\_TS(X) = TS(T), where T is the youngest transaction that has written X successfully.

1. **Commit protocols**

In computer networking and databases, the three-phase commit protocol (3PC)[1] 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. Specifically, 3PC places an upper bound on the amount of time required before a transaction either commits or aborts. This property ensures that if a given transaction is attempting to commit via 3PC and holds some resource locks, it will release the locks after the timeout.

In describing the protocol, we use terminology similar to that used in the [two-phase commit protocol](https://en.wikipedia.org/wiki/Two-phase_commit_protocol). Thus we have a single coordinator site leading the transaction and a set of one or more cohorts being directed by the coordinator.

[](https://en.wikipedia.org/wiki/File:Three-phase_commit_diagram.png)

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.[1] 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.

* 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 and it 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**
2. **Time stamp ordering multi-version concurrency control**
3. **Deadlock handling detection & resolution**