**ST.XAVIER’S COLLEGE**

(Affiliated to Tribhuwan University)

Maitighar, Kathmandu

****

**Database Management System**

Assignment #8

Submitted By:

Dikita Tuladhar

013BSCCSIT018

2nd Year/ 4th Semester

Submitted to:

|  |  |
| --- | --- |
| Er. Sanjay Kr. Yadhav  Lecturer  Department of Computer Science |  |

Date of Submission 30th October, 2015

Database Concurrency Control

1. Purpose of Concurrency Control

In [information technology](https://en.wikipedia.org/wiki/Information_technology) and [computer science](https://en.wikipedia.org/wiki/Computer_science), especially in the fields of [computer programming](https://en.wikipedia.org/wiki/Computer_programming), [operating systems](https://en.wikipedia.org/wiki/Operating_systems), [multiprocessors](https://en.wikipedia.org/wiki/Multiprocessor), and [databases](https://en.wikipedia.org/wiki/Database), concurrency control ensures that correct results for [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing) operations are generated, while getting those results as quickly as possible.

Computer systems, both [software](https://en.wikipedia.org/wiki/Software) and [hardware](https://en.wikipedia.org/wiki/Computer_hardware), consist of modules, or components. Each component is designed to operate correctly, i.e., to obey or to meet certain consistency rules. When components that operate concurrently interact by messaging or by sharing accessed data (in [memory](https://en.wikipedia.org/wiki/Computer_memory) or [storage](https://en.wikipedia.org/wiki/Computer_data_storage)), a certain component's consistency may be violated by another component. The general area of concurrency control provides rules, methods, design methodologies, and [theories](https://en.wikipedia.org/wiki/Scientific_theory) to maintain the consistency of components operating concurrently while interacting, and thus the consistency and correctness of the whole system. Introducing concurrency control into a system means applying operation constraints which typically result in some performance reduction. Operation consistency and correctness should be achieved with as good as possible efficiency, without reducing performance below reasonable levels. Concurrency control can require significant additional complexity and overhead in a [concurrent algorithm](https://en.wikipedia.org/wiki/Concurrent_algorithm) compared to the simpler [sequential algorithm](https://en.wikipedia.org/wiki/Sequential_algorithm).

PURPOSE:

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

1) The Lost Update Problem

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.

Example:

• T1 withdraws £10 from an account with balx, initially £100.

• T2 deposits £100 into same account.

• Serially, final balance would be £190.

2) The Temporary Update (or Dirty Read) Problem

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

Occurs when one transaction can see intermediate results of another transaction before it has committed.

Example:

• T4 updates balx to £200 but it aborts, so balx should be back at original value of £100.

• T3 has read new value of balx (£200) and uses value as basis of £10 reduction, giving a new balance of £190, instead of £90.

3)The Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Occurs when transaction reads several values but second transaction updates some of them during execution of first.

Example:

• T6 is totaling balances of account x (£100), account y (£50), and account z (£25).

• Meantime, T5 has transferred £10 from balx to balz, so T6 now has wrong result (£10 too high).

1. Two Phase Locking

The most widely used locking protocol, called Strict Two-Phase Locking, or Strict 2PL, has two rules. The first rule is

1. If a transaction T wants to read (respectively, modify) an object, it first requests a shared (respectively, exclusive) lock on the object.

Of course, a transaction that has an exclusive lock can also read the object; an additional shared lock is not required. A transaction that requests a lock is suspended until the DBMS is able to grant it the requested lock. The DBMS keeps track of the locks it has granted and ensures that if a transaction holds an exclusive lock on an object, no other transaction holds a shared or exclusive lock on the same object. The second rule in Strict 2PL is

1. All locks held by a transaction are released when the transaction is completed.

Requests to acquire and release locks can be automatically inserted into transactions by the DBMS; users need not worry about these details.

In effect, the locking protocol allows only 'safe' interleaving of transactions. If two transactions access completely independent parts of the database, they concurrently obtain the locks they need and proceed merrily on their ways. On the other band, if two transactions access the same object, and one wants to modify it, their actions are effectively ordered serially, all actions of one of these transactions (the one that gets the lock on the common object first) are completed before (this lock is released and) the other transaction can proceed.

We denote the action of a transaction T requesting a shared (respectively, exclusive) lock on object 0 as 5T(0) (respectively, XT(O)) and omit the subscript denoting the tn1l1saction when it is clear from the context. As an example, consider the schedule shown in Figure 16.4. This interleaving could result in a state that cannot result from any serial execution of the three transactions. For instance, T1 could change A from 10 to 20, then T2 (which reads the value 20 for A) could change B from 100 to 200, and then T1 would read the value 200 for B. If run serially, either Tl or T2 would execute first, and read the values 10 for A and 100 for B: Clearly, the interleaved execution is not equivalent to either serial execution.

If the Strict 2PL protocol is used, such interleaving is disallowed. Let us see why. Assuming that the transactions proceed as before, T1 would obtain an exclusive lock on A first and then read and write A (Figure I). Then, 1'2 would request a lock on A.

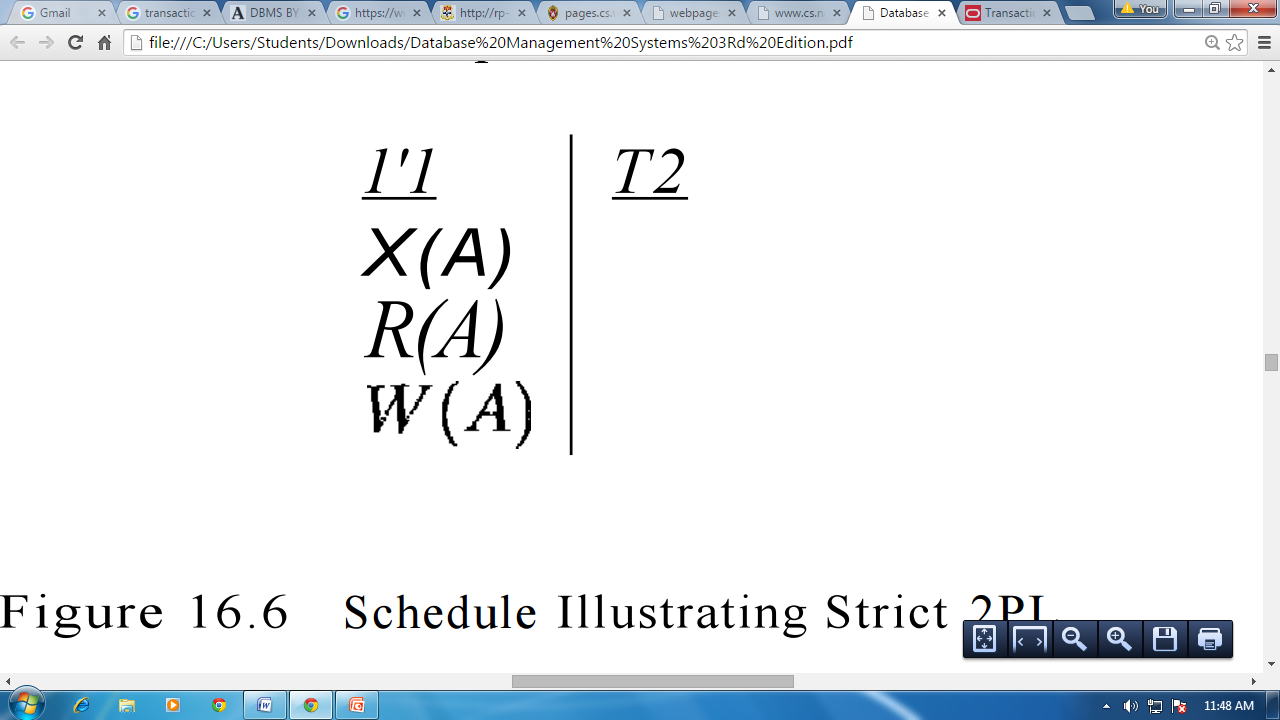


Figure I: Schedule Illustrating Strict 2PL

However, this request cannot be granted until 1'1 releases its exclusive lock on A, and the DBMS therefore suspends 1'2. 1'1 now proceeds to obtain an exclusive lock on B, reads and writes B, then finally commits, at which time its locks are released. T2's lock request is now granted, and it proceeds. In this example the locking protocol results in a serial execution of the two transactions, shown in Figure II.

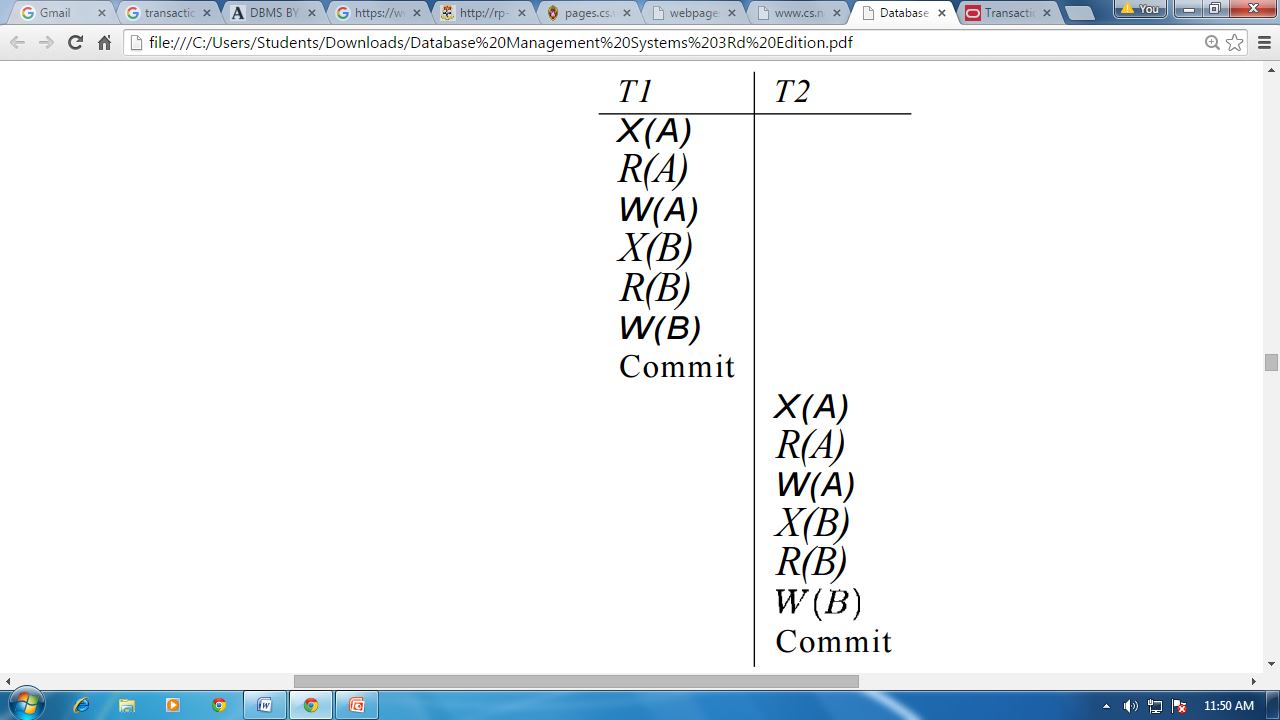


Figure II: Schedule Illustrating Strict 2PL with Serial Execution

In general, however, the actions of different transactions could be interleaved. As an example, consider the interleaving of two transactions shown in Figure III, which is permitted by the Strict 2PL protocol. It can be shown that the Strict 2PL algorithm allows only serializable schedules. None of the anomalies discussed in Section 16.3.:3 can arise if the DBMS implements strict 2PL.

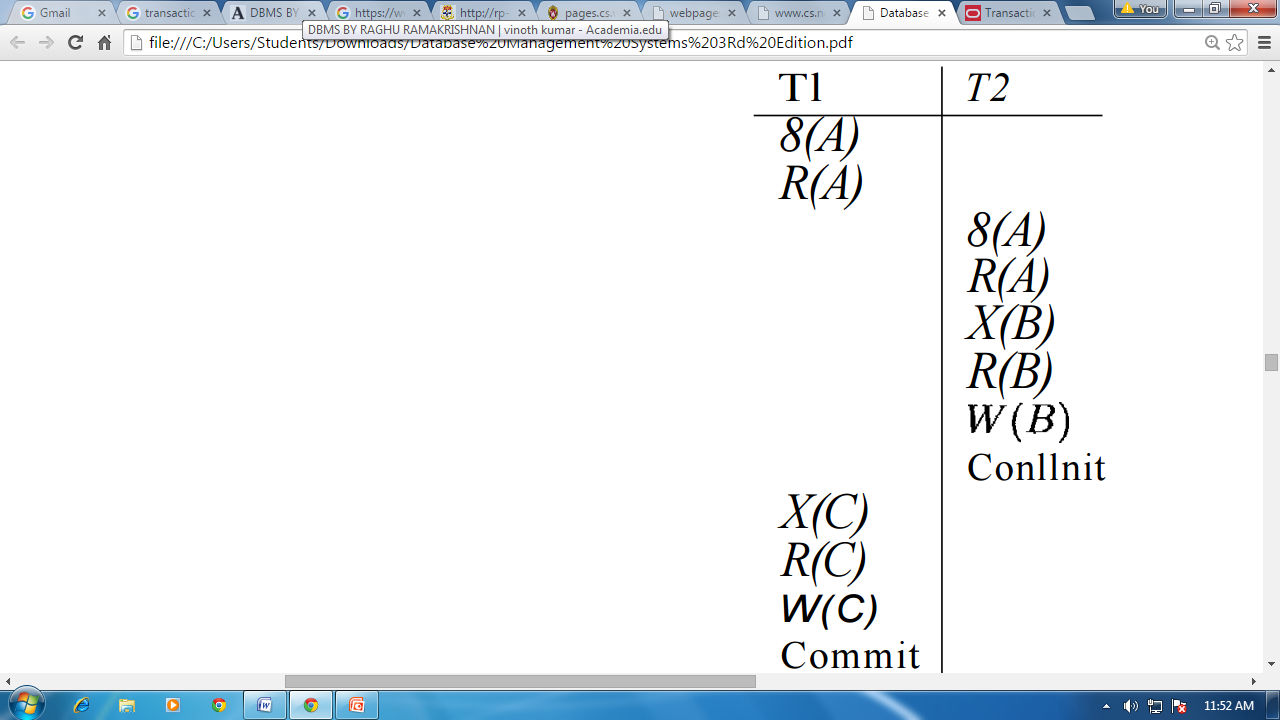


Figure III: Schedule Following Strict 2PL with Interleaved Actions

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

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.

Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write 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 −

o If TS(Ti) < W-timestamp(X)

 Operation rejected.

o If TS(Ti) >= W-timestamp(X)

 Operation executed.

o All data-item timestamps updated.

• If a transaction Ti issues a write(X) operation −

o If TS(Ti) < R-timestamp(X)

 Operation rejected.

o If TS(Ti) < W-timestamp(X)

 Operation rejected and Ti rolled back.

o Otherwise, operation executed.

Thomas' Write Rule

This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making Ti rolled back, the 'write' operation itself is ignored.

1. Commit Protocols
2. Index Locking
3. Lock Granularity
4. Time Stamp ordering Multi-Version Concurrency Control
5. Deadlock Handling Detection and Resolution