ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



DATABASE MANAGEMENT SYSYTEM

Lab Assignment #12

**Submitted by:**

Lokendra Puri  
013BSCCSIT023

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer |  |

Department of Computer Science

Date of Submission: October 29, 2015

Database concurrency control

Concurrency control is a database management systems (DBMS) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system. Concurrency control, when applied to a DBMS, is meant to coordinate simultaneous transactions while preserving data integrity. The Concurrency is about to control the multi-user access of Database

### Purpose of concurrency control

### To address conflict with simultaneous addressing

### To handle concurrent altering of data in a multi-user environment

### Coordinate simultaneous transactions while maintaining data integrity

### To further illustrate the concept of concurrency control, consider an example of two people using an electronic kiosk at the same time.

### Considering there is only one seat left in coach, without concurrency control, both the travelers may end up buying the same ticket.

### However, with the implementation of concurrency control, the database won’t allow this to happen.

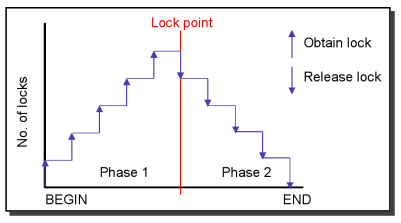
### Both travelers would still be able to access the train seating database, but concurrency control would allow only one traveler to purchase the seat

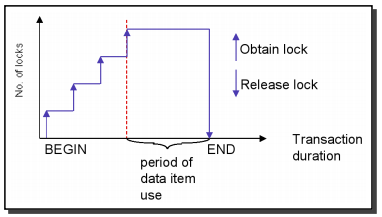
### Two phase locking

In databases and transaction processing, two-phase locking (2PL) is a concurrency control method that guarantees serializability. It is also the name of the resulting set of database transaction schedules (histories). The protocol utilizes locks, 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: Expanding phase: locks are acquired and no locks are released. Shrinking phase: locks are released and no locks are acquired. Two types of locks are utilized by the basic protocol: Shared and Exclusive locks. Refinements of the basic protocol may utilize more lock types. Using locks that block processes, 2PL may be subject to deadlocks that result from the mutual blocking of two or more transactions.

**Two-phase locking protocol**  
– Each transaction is executed in two phases  
∗ Growing phase: the transaction obtains locks  
∗ Shrinking phase: the transaction releases locks  
– The lock point is the moment when transitioning from the growing phase to the  
shrinking phase

**Properties** of the 2PL protocol  
**–** Generates **conflict-serializable**schedules  
**–** But schedules may cause **cascading aborts**  
∗If a transaction aborts after it releases a lock, it may cause other transactions that  
have accessed the unlocked data item to abort as well

**Strict 2PL locking** protocol  
**–** Holds the locks till the end of the transaction  
**–** Cascading aborts are avoided



### 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.

### 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.

### Commit protocols

### 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). Index Locking Protocol guarantees that Phantom Phenomenon won't occur. Index locking protocol states:

* 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.

### Lock granularity

• Fine-grained locks allow a lot of concurrency, but may cause problems with deadlocks.  
• Coarse-grained locks require fewer resources by the locking scheduler.  
• We want to allow fine-grained locks,but use (or switch to) coarser lockswhen needed.  
• Some DBMSs switch automatically - this is called lock escalation. The downside is that this easily leads to deadlocks.

### Time stamp ordering multi-version concurrency control

• In principle, all previous versions of DB elements could be saved.  
• This would allow any read operation to be consistent with the timestamp of the transaction.  
• Used in many systems for scheduling read only transactions. (In practice, only recent versions are saved.)

### Deadlock handling detection and resolution

• The DBMS sometimes must make a transaction wait for another transaction to release a lock.  
• This can lead to deadlock if e.g. A waits for B, and B waits for A.  
• In general, we have a deadlock exactly when there is a cycle in the waits-for graph.  
• Deadlocks are resolved by aborting some transaction involved in the cycle.

• Possibilities:  
– Examine the waits-for graph periodically to find any deadlock.  
– If a transaction lived for too long it may be involved in a deadlock - roll back.  
– Use timestamps, unique numbers associated with every transaction to prevent deadlocks.  
• Deadlocks are less likely if we lock entire relations - but this decreases throughput.

**Avoiding Deadlock by using timestamp:**

Two possible policies when T waits for U:

-Wait and Die: If T is youngest it is rolled back

- Wound wait: If U is youngest it is rolled back

In both cases there can be no waits-for cycles, because transactions only wait for younger (resp. older) transactions.