Record locking

From Wikipedia, the free encyclopedia

Record locking is the technique of preventing simultaneous access to data in a database, to prevent inconsistent results.

The classic example is demonstrated by two bank clerks attempting to update the same bank account for two different transactions. Clerk 1 and clerk 2 both retrieve (ie, copy) the account's current balance. Clerk 1 applies one transaction and refiles the new balance. Clerk 2 applies a different transaction and refiles a new balance that obliterates the information saved by clerk 1. The resulting account balance no longer reflects the first transaction.

To prevent inconsistencies created by that kind of unlimited access, the account's record can instead be immediately *locked* upon being retrieved for any subsequent update. Anyone attempting to retrieve the same record for editing is denied access because of the lock (although, depending on the implementation, they may be able to view the record without editing it). Once the record is saved or edits are canceled, the lock is released, thereby always insuring consistent data within the record being edited.

Use of locks

Record locks (hereafter lock(s)) need to be managed between the entities requesting the records such that no entity is given too much service via successive **grants**, and no other entity is effectively locked out. Care should also be used to avoid a deadlock condition which can bring the application or system to a halt. The entities that request a lock can be either individual applications (programs) or an entire processor.

The application or system should be designed such that any lock is held for the shortest amount of time possible. Given that there may be considerable overhead to the process of requesting, and subsequent granting of a lock, it may make sense to investigate if an entity can forego a lock if the purpose is simply to read non-critical data.

There are two main types of locks that can be requested:

Exclusive locks

Exclusive locks are as the name implies, exclusively held by a single entity. If the locking schema was represented by a list, the **holder list** would contain only one entry. Since this type of lock effectively blocks any other entity that requires the lock from processing, care must be used to:

- Ensure the lock is held for the least amount of time possible
- Do not hold the lock across system/function calls where the entity is no longer running on the processor - this can lead to deadlock
- Ensure that if the entity is unexpectedly exited for any reason, the lock is freed otherwise deadlock will likely occur.

Non-holders of the lock (aka **waiters**) should perhaps be maintained in a list that is serviced in a round robin fashion. This would ensure that any possible waiter would get equal chance to obtain the lock and not be locked out. To further speed up the process, if an entity has gone to sleep waiting for a lock, performance is improved if the entity is notified of the grant, instead of discovering it on some sort of system timeout driven wakeup.

Shared locks