CONCURRENCY CONTROL

With the continuous expansion of a system the reliability of a system becomes more and more important. There is one thing that must be taken into consideration --- concurrency control. Using our project as an example, there may be multiple administrators trying to write to the database. That is to say, we have to run transactions concurrently to meet their performance requirements. Thus, without concurrency control our plugin can neither provide correct results nor maintain their databases consistent.

Concurrency control mechanisms firstly need to operate correctly, that is to say to maintain each transaction's integrity rules while transactions are running concurrently, and thus the integrity of the entire transactional system. Correctness needs to be achieved with as good performance as possible. In addition, increasingly a need exists to operate effectively while transactions are distributed over processes, computers, and computer networks. Other subjects that may affect concurrency control are recovery and replication.

In order to prevent data from being corrupted or invalidated, a lock is one of the important methods for concurrency control. Any single user can only modify those database records (that is, items in the database) to which they have applied a lock that gives them exclusive access to the record until the lock is released. Locking

not only provides exclusivity to writes but also prevents (or controls) reading of unfinished modifications.

From the programmer's point of view, there are two mechanisms for locking data in a database:

Pessimistic locking is where a record or page is locked immediately when the lock is requested.

Optimistic locking is where a record or page is only locked when the changes made to that record are updated.

The Optimistic locking situation is only appropriate when there is less chance of someone needing to access the record while it is locked; otherwise it cannot be certain that the update will succeed because the attempt to update the record will fail if another user updates the record first. With pessimistic locking it is guaranteed that the record will be updated.

In the sight of database, there are three mechanisms for locking data in a database:

Exclusive Locking: The resources have been locked only allow accessing the locking operation, other operations will not be accepted. Implementing the command of update data, that is, INSERT, UPDATE or DELETE command, SQL Server will use an exclusive lock automatically. However, when the object has other locks, we cannot use exclusive lock. Exclusive lock can be released until the end of transaction.

Shared locking: The resources which have been locked only allow other users reading, but not modifying. In SELECT command, SQL Server usually locked the object with shared lock. Usually when the data with the shared lock has been read, the shared lock would be released immediately.

Update locking: Update lock is created to avoid deadlock. When SQL Server updates the data, it will lock the data with update lock firstly, and the data can be read, but cannot be modified. When SQL Server sure to update operation of data, it will change to exclusive lock automatically. However, there are other locks in object, it cannot be locked with updated lock.

Different categories provide different performance, i.e., different average transaction completion rates, depending on transaction types mix, computing level of parallelism, and other factors. If selection and knowledge about trade-offs are available, then category and method should be chosen to provide the highest performance.

Source: http://toyhouse.cc/profiles/blogs/concurrency-control