Hao Heng

28th

Transactions, locks, and isolation levels

Transaction, by its definition, is a single work unit consisted with a sequence of SQL statement. Transaction is designed to deal with concurrency issue happened in database. So, let’s take a look at the problem, i.e. the concurrency issue.

Concurrency issue means there are more than one operation that want to operate on the same data in the database, the system must make some rules to manage these operations, otherwise, users are very likely get wrong results and the data in the database will be meaningless.

To solve this problem, the system introduces transaction which represents an operation of one user and give these transactions different permission to let these operations be done in a specific order to maintain the database integrity. Therefore, the system gives four isolation levels to specify the general target management the system wants to achieve.

Start from the lowest isolation level, read uncommitted. At this level, the system will give user whatever the system has at the moment that the user makes a request. The user doesn’t care if the data he is trying to access is being modified or too old, therefore, at this level, transactions won’t conflict with each other, they all will be committed. This isolation level usually leads to a problem called dirty read.

The next level is read committed level, which can prevent dirty read because the transaction that try to read the data must wait until the transaction that update the data finished. But it can’t stop non repeatable problem

The next level is repeatable read level, which can make sure that when a transaction read a data at different time will still get the same result. But this isolation can’t prevent phantom read problem. Phantom read means the transaction wants to read the data that has been deleted or been inserted new data in that address.

The highest level is serializable level. This level will solve all kind of read problem mentioned above.

According to those levels, the system applies different kind of locks to make sure the transaction will be executed in orders. The system gives three kind of locks, shared lock, update lock, exclusive clock. Those locks only be applied in pessimistic control.

The shared look allows multiple transactions to read the same data, cause when the transaction just want to read, the data integrity is well preserved.

The update lock will be turn on if there is a transaction is trying to update the data. If other read transactions keep running, it will likely cause dirty read. So, the update lock will stop the read transaction until the update is completed.

The exclusive lock will be turned on when the actual update begins. Once this lock is placed, the transaction can’t be roll back. After the actual updating is completed, the lock will be released.

In the database, not all the transactions are specified or controlled by the users. There also are three kind of transactions in database. It includes auto-commit transaction, implicit transaction, explicit transaction.

The auto-commit transaction is called and committed or rolled back by the system, users have no control on them at all.

The implicit transaction needs users to manually commit or roll back. But they still automatically be called by the system.

The explicit transaction is the kind that users need to write the complete query in the SQL statement.

29th

If the service response very slow and it even throw an error, it’s either the database can’t use the existing way to locate the target data or the target data has been modified by someone without notifying other users.

If it is the first reason, it is very likely the data base is doing table scan when it is trying to find the data. That means the index of that table no longer can be used. Then I would like to check the system log and observe extended event to see if the index is missing at the beginning. Tuning advisor can help me to locate the missing index. If the index is already there, then go to check if there is a massive data update happened recently which will cause a long time of updating the index. In this case, as an operator, I need to rebuild the index for this table. After fixing the possible index problem, using the execution plan to compare if the performance is improved or not.

If the index is still functional well, then probably the data itself has a problem. The strategy is almost the same, checking system log, extended events, using DBCC, DMV to see recent transactions. The purpose of this is to see if there are any change that haven’t been committed or any deadlocks remained to be solved. If the update has been made and yet not committed, the system probably needs to rollback the transaction since those modification is from last Friday. To compare with the original data from last week, temporal table is a very useful tool to restore the data at a specific time period.

Another important observation has to be made is the activity running during the user using the database. If there are some other transactions and the system’s isolation level is not set to the right level, then problems such as dirty read will likely occurs.