Chapter 6
Lock-Based Concurrency Control

Locking is the most commonly used method for enforcing transactional isolation. Most database management systems apply some kind of locking, possibly coupled with some other mechanism (such as transient versioning). With locking-based concurrency control, transactions are required to protect their actions by acquiring appropriate locks on the parts of the database they operate on. A read action on a data item is usually protected by a shared lock on the data item, which prevents other transactions from updating the data item, and an update action is protected by an exclusive lock, which prevents other transactions from reading or updating the data item.

If a transaction requests a lock on a data item in a situation in which some other transaction holds an exclusive lock on the data item, the requesting transaction has to wait for the conflicting lock to be released. Most often this means waiting for the other transaction to commit; all the locks still held by a transaction at commit time are released when the transaction has committed.

The isolation anomalies defined in the previous chapter for our key-range transaction model can be avoided with a locking protocol called key-range locking. This protocol prevents phantoms by effectively locking a range between two keys with a shared lock on one key (for a read action) or with exclusive locks on two successive keys (for an insert or a delete action). We give a proof of the correctness of key-range locking in the general case in which transactions can contain partial rollbacks with the feature that commit-duration locks acquired after setting a savepoint are released after completing a partial rollback to that savepoint.

A problem with locking is that transactions waiting for locks may get into a deadlock, where none of the transactions can proceed. Deadlocks must be resolved by aborting and rolling back one or more of the transactions involved. With ordinary lock requests, which automatically lead to a wait if the lock cannot be immediately granted, deadlocks involving holds of and waits for latches may also occur. In this case relying on deadlock resolution is not feasible; the occurrences of deadlocks between locks and latches must be prevented. The solution is to use a conditional
lock request, which results in granting the lock only if the lock can be granted without wait. On this line, we discuss the implementation of key-range locking for the actions in our key-range transaction model; a complete implementation is obtained by coupling the basic algorithms given in this chapter with the B-tree algorithms given in the two following chapters.

6.1 Locks and the Lock Table

A lock is a main-memory data item belonging to an active transaction that grants the transaction access to a specific part of the database. A transaction can execute an action (read, write, insert, delete) on a part of the database only if it has properly locked the relevant part of the database.

A lock includes information about its name, mode, duration, and the owning transaction. The lock name identifies the data item or the set of data items in the database that is the target of the lock. The name can be used to categorize locks into logical and physical locks.

The name of a logical lock identifies a part of the logical database, such as a single tuple in a relation, the set of tuples within a key range, or a whole relation. The name of a logical lock for tuple $(x, v)$ in relation $r(X, V)$ is the unique key $x$ (when $r$ is the only relation in the logical database) or the pair $(r, x)$ with the relation-id $r$ and the key $x$ (if several relations exist in the database). A lock named $x$ locks the tuple with key $x$ regardless of whether or not such a tuple exists when the lock is granted. Such a lock may be used to protect actions on that tuple or, say, actions on a key range delimited by that tuple.

The name of a physical lock identifies a part of the physical database, such as the location of a tuple in a specific data page, or a whole page or file, or a node or a bucket in an index structure. The lock name of a physical lock on page $p$ is the page-id $p$. Such a lock can be used to lock all the record positions in page $p$ no matter if there are actual records in those positions at the time of granting the lock. The lock name of a physical lock on record position $i$ in page $p$ is the record-id $(p, i)$. Such a lock locks the record position even if it does not contain a record at the time of granting the lock.

Operating on locks is more efficient if the lock names are fixed-length short values, such as four-byte integers. Instead of variable-length or structured lock names, actual implementations usually use a hash value $h(x)$ calculated from the name $x$ using some hash function $h$. Then a lock on key $x$ locks all tuples with keys $y$ that satisfy $h(y) = h(x)$.

The lock mode defines what sort of actions the lock permits. For reading a data item $x$, a transaction must acquire at least a read lock or an S lock on $x$. A read lock is shared: multiple transactions can have a read lock on $x$ simultaneously.

For writing a data item $x$, a transaction must acquire a write lock or an X lock on $x$. Write locks are exclusive: when a transaction has a write lock on $x$, other