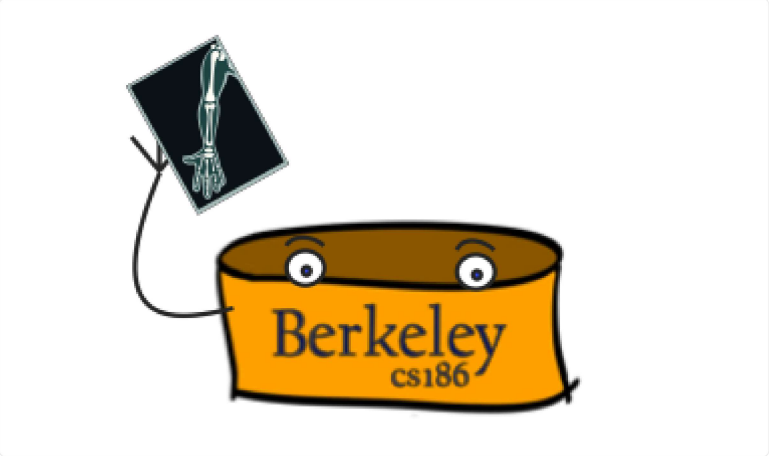
Part 0: Skeleton Code

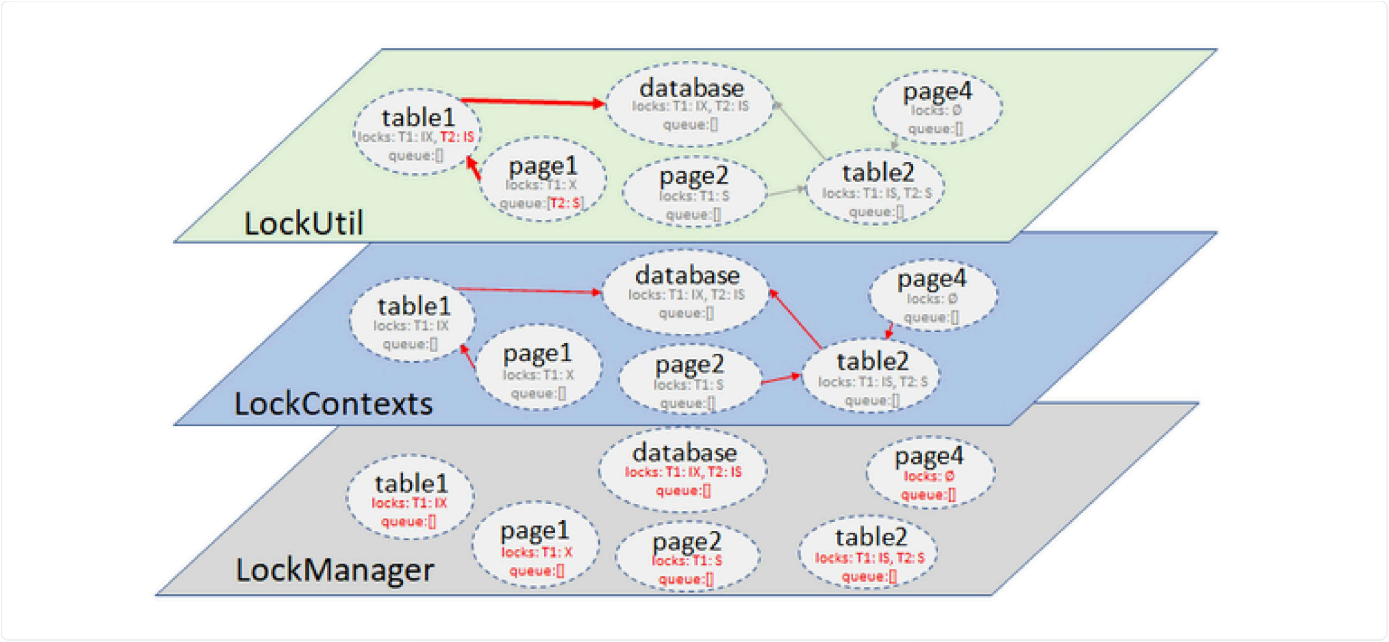


Data x-ray

Read through all of the code in the `concurrency` directory (including the classes that you are not touching - they may contain useful methods or information pertinent to the project). Many comments contain critical information on how you must implement certain functions.

Try to understand how each class fits in: what is each class responsible for, what are all the methods you have to implement, and how does each one manipulate the internal state. Trying to code one method at a time without understanding how all the parts of the lock manager work often results in having to rewrite significant amounts of code.

Layers



The skeleton code divides multigranularity locking into three layers.

- The `LockManager` object manages all the locks, treating each resource as independent (it doesn't consider the resource hierarchy at all). This level is responsible queuing logic, blocking/unblocking transactions as necessary, and is the single source of authority on whether a transaction has a certain lock. If the `LockManager` says T1 has X(database), then T1 has X(database).
- A collection of `LockContext` objects, which each represent a single lockable object (e.g. a page or a table) lies on top of the `LockManager`. The `LockContext` objects are connected according to the hierarchy (e.g. a `LockContext` for a table has the database context as its parent, and its pages' contexts as children). The `LockContext` objects all share a single `LockManager`, and each context enforces multigranularity constraints on its methods (e.g. an exception will be thrown if a transaction attempts to request X(table) without IX(database)).
- A declarative layer lies on top of the collection of `LockContext` objects, and is responsible for acquiring all the intent locks needed for each S or X request that the database uses (e.g. if S(page) is requested, this layer would be responsible for requesting IS(database), IS(table) if necessary).

In Part 1, you will be implementing the bottom layer (`LockManager`) and lock types. In Part 2, you will be implementing the middle and top layer (`LockContext` and `LockUtil`), and integrate your changes into the database.

Previous
Getting Started

Next
Part 1: Queuing

Last updated 4 months ago