50.043 Database Systems and Big Data

Lab 3 - Spring 2024

Group Members

Name	Student ID
Atul Parida	1006184
Chandrasekar Akash	1006228
Jignesh Motwani	1006178

Summary of Code & Design Decisions

Addition of ReadWriteLock

The *ReadWriteLock* class provides functionality for managing read and write locks in a transactional setting. It allows multiple transactions to acquire shared read locks simultaneously, while only one transaction can acquire an exclusive write lock. Key methods include *readLock*, *writeLock*, *readUnlock*, *writeUnlock*, and *unlock*. The class uses a set to track transactions holding locks and a map to track lock acquisitions. It ensures thread safety through synchronized blocks and provides methods to query lock status and holders. Though a Java implementation for *ReadWriteLock* already exists, we developed a custom implementation better suited for SimpleDB's needs.

Addition of LockManager

The *LockManager* class manages locks on pages for transactions in a database system, majorly simplifying the process of lock management in BufferPool and other locations. It tracks locks using a map associating pages with their corresponding *ReadWriteLock* objects and maintains a dependency graph to detect deadlocks. Methods are provided to acquire read and write locks, release locks, and check lock status. The class ensures proper concurrency control by coordinating lock acquisition and release among transactions while also detecting and resolving deadlock scenarios. A *transactionComplete()* method was implemented in *BufferPool* to ensure the *LockManager* was notified of transaction completion, releasing locks accordingly.

We implemented a two-phase locking protocol using Shared and Exclusive locks along with a deadlock detection mechanism to maximize concurrency. We ensured the correct permissions for file access were used within *getPage()*, which handles page retrieval, along with locking protocols implemented for reading, insertion and deletion mechanisms. A *NO STEAL* policy was implemented in *evictPage()* to ensure that dirty pages were not evicted by the system.