**Lab Report - Lab 5: SimpleDB Transactions**

**Introduction**

This lab focused on implementing a simple locking-based transaction system in SimpleDB. The primary objective was to integrate lock management and ensure that transactions are handled correctly, adhering to the ACID properties. This report discusses the implementation details, design decisions, and challenges faced during the lab.

**Design Decisions**

**Lock Management**

- **Lock Granularity**: Implemented page-level locking as recommended. This approach balances the need for concurrency and complexity management.
- **Lock Types**: Implemented both shared and exclusive locks. Shared locks are used for reading operations, while exclusive locks are necessary for write operations.
- **Lock Acquisition and Release**: Integrated lock acquisition in the **BufferPool.getPage()** method to ensure that locks are obtained before any data page is accessed. Locks are released when transactions complete, ensuring that no locks are held beyond the life of a transaction.

**Transaction Handling**

- **NO STEAL/FORCE Policy**: Modified the **evictPage** method to ensure that dirty pages are not evicted before being committed. This approach helps maintain the atomicity and durability of transactions.
- **Transaction Completion**: Enhanced the **transactionComplete()** method to handle both commit and abort operations. Commit operations flush dirty pages to disk, while abort operations restore the page content from the disk.

**Deadlock Detection and Resolution**

- **Detection Strategy**: Implemented a simple dependency graph to detect deadlocks. The system checks for cycles in the graph whenever a new lock is requested.
- **Resolution Strategy**: Decided to abort transactions involved in a deadlock to ensure system responsiveness and prevent indefinite blocking.

**API Modifications**

No significant changes were made to the existing SimpleDB API. However, additional methods were added to the **BufferPool** class to handle specific lock management tasks like checking if a transaction holds a lock.

**Challenges and Reflections**

The most challenging aspect of this lab was ensuring correct implementation of the locking mechanism without introducing deadlocks or performance bottlenecks. Debugging concurrency issues required careful analysis and testing, especially when integrating new test cases provided for this lab.

Another challenge was the implementation of the NO STEAL policy, particularly when handling scenarios where the buffer pool is full of dirty pages.

**Time Spent and Difficulties Encountered**

- **Time Spent**: Approximately 25 hours spread over two weeks.
- **Difficulties**: Debugging deadlock scenarios was particularly tricky, requiring multiple iterations to get right. Implementing and testing the NO STEAL/FORCE policies also posed significant challenges.

**Conclusion**

This lab provided a practical experience in managing transactions in a database system with a focus on concurrency control, recovery, and deadlock management. The implementation of a locking protocol in SimpleDB helped in understanding the complexities involved in transaction systems and the importance of rigorous testing in concurrent environments.