实验报告：实现基于锁的事务支持

1. 概述

本实验旨在为 SimpleDB 实现一个基于锁的事务系统。实现内容包括：

- 锁管理器，管理事务锁的请求和释放。
- 死锁检测和恢复，确保系统不会因为死锁而进入僵局。
- 事务恢复，确保数据的一致性。

2. 实现细节

2.1 锁管理器

锁管理器负责管理所有事务的锁请求和释放，确保数据库操作的并发控制。

主要功能

- **锁类型**：支持共享锁（S 锁）和排他锁（X 锁）。
- **锁表**：使用 ConcurrentHashMap 记录每个数据项的锁状态和持有锁的事务。
- **请求锁**：当事务请求锁时，检查锁表，判断锁的兼容性并决定是否授予锁。
- **释放锁**：当事务结束时，释放其持有的所有锁。

2.2 死锁检测和恢复

死锁检测是通过构建依赖图（Dependency Graph）实现的，依赖图记录了事务之间的等待关系。

主要功能

- **依赖图**：使用 ConcurrentHashMap 记录每个事务的依赖关系。
- **死锁检测**：通过深度优先搜索（DFS）检测依赖图中的循环，从而判断是否存在死锁。
- **死锁恢复**：在检测到死锁时，通过抛出 TransactionAbortedException 来中止事务并进行恢复。

2.3 事务恢复

事务恢复包括在事务提交或回滚时，确保数据库的状态一致性。

主要功能

- **提交事务**：在事务完成其操作并提交时，释放所有持有的锁。
- **回滚事务**：在事务需要回滚时，恢复事务开始前的数据状态，并释放所有持有的锁。

3. 测试与验证

3.1 测试锁管理器

测试目标

- 验证锁管理器能够正确处理锁的请求和释放。
- 验证共享锁和排他锁的兼容性。

测试方法

- 编写单元测试，测试多个事务对同一数据项请求共享锁和排他锁的情况。

- 确保在事务释放锁后，其他事务能够正确获取锁。

3.2 测试死锁检测

测试目标

- 验证死锁检测器能够正确检测并处理死锁情况。
- 确保系统能够在检测到死锁后进行恢复。

测试方法

- 构造多个事务之间的循环依赖，模拟死锁情况。
- 编写单元测试，确保在检测到死锁时抛出 TransactionAbortedException。

3.3 测试事务恢复

测试目标

- 验证事务在提交和回滚时能够正确恢复数据一致性。
- 确保事务在提交或回滚后，所有持有的锁都被正确释放。

测试方法

- 编写单元测试，测试事务在提交和回滚后的数据状态。
- 确保在事务提交或回滚后，其他事务能够正确获取锁。

4. 遇到的问题与解决方案

4.1 问题一：锁兼容性判断

在实现锁管理器时，判断共享锁和排他锁的兼容性是一个挑战。通过详细设计锁表结构，确保在请求锁时能够正确判断当前锁状态。

4.2 问题二：死锁检测的复杂性

死锁检测需要构建依赖图并进行深度优先搜索。在实现过程中，通过优化依赖图的构建和搜索算法，提高了检测效率。

4.3 问题三：事务回滚的正确性

确保事务在回滚时能够恢复到初始状态。通过严格的单元测试和调试，确保每个步骤都能正确执行。

5. 总结

通过本实验的实现和测试，掌握了事务支持的基本原理和实现方法。实现了锁管理、死锁检测和事务恢复功能，确保了数据的一致性和系统的稳定性。在实现过程中，通过不断的调试和优化，解决了多个复杂问题，提高了系统的可靠性和性能。