**《数据库系统原理》实验报告**

**实验题目：并发控制技术实验**

**姓名： 郑德凯** **实验日期： 2024年 12月 24日**

**实验内容及完成情况：**

# 一、实验目的

本实验的目的是使学生掌握数据库并发控制的基本原理及其应用方法。

# 实验内容

1、验证并发操作带来的数据不一致性问题，包括丢失修改、不可重复读和读“脏”数据等情况。要求通过取消查询分析器的自动提交功能，创建两个不同的用户，分别登录查询分析器，同时打开两个客户端；通过SQL语言设计具体例子展示不同的封锁级别的应用场景，验证各种封锁级别的并发控制效果，以进一步理解封锁技术是如何解决事务并发导致的问题。

DBMS通常提供四级隔离级别：读未提交（read uncommitted）、读已提交（read committed）、可重复读（repeatable read）、可串行化（serializable）。

四级隔离级别与三级封锁协议大致对应关系如下：

①读未提交隔离级别提供最大的事务并发度，但仅能避免丢失修改，对应一级封锁协议。

②读已提交隔离级别提供的事务并发度减弱，能够避免丢失修改和脏读，对应二级封锁协议。

③可重复读隔离级别提供的事务并发度进一步减弱，能够避免丢失修改和脏读和不可重复读，对应增强的二级封锁协议。

④可串行化隔离级别提供最小的事务并发度，能够避免所有的事务并发控制问题，对应三级封锁协议。

准备一个包含银行账户信息的表，例如：

CREATE TABLE Accounts (

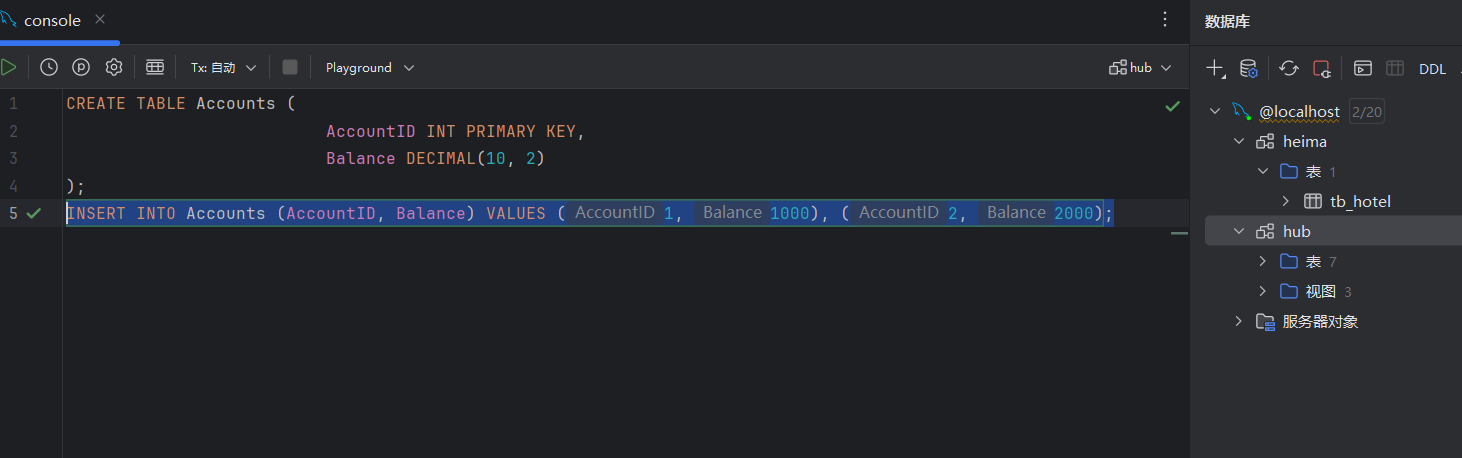
AccountID INT PRIMARY KEY,

Balance DECIMAL(10, 2)

);

插入一些初始数据：

INSERT INTO Accounts (AccountID, Balance) VALUES (1, 1000), (2, 2000);



示例场景

场景1: 读未提交（Read Uncommitted）

User A 开始一个事务，从 Account 1 转账 500 到 Account 2，但尚未提交。

User B 在同一时间开始另一个事务，读取 Account 1 的余额，可能会看到还未提交的更改。

-- User A:

BEGIN TRANSACTION;

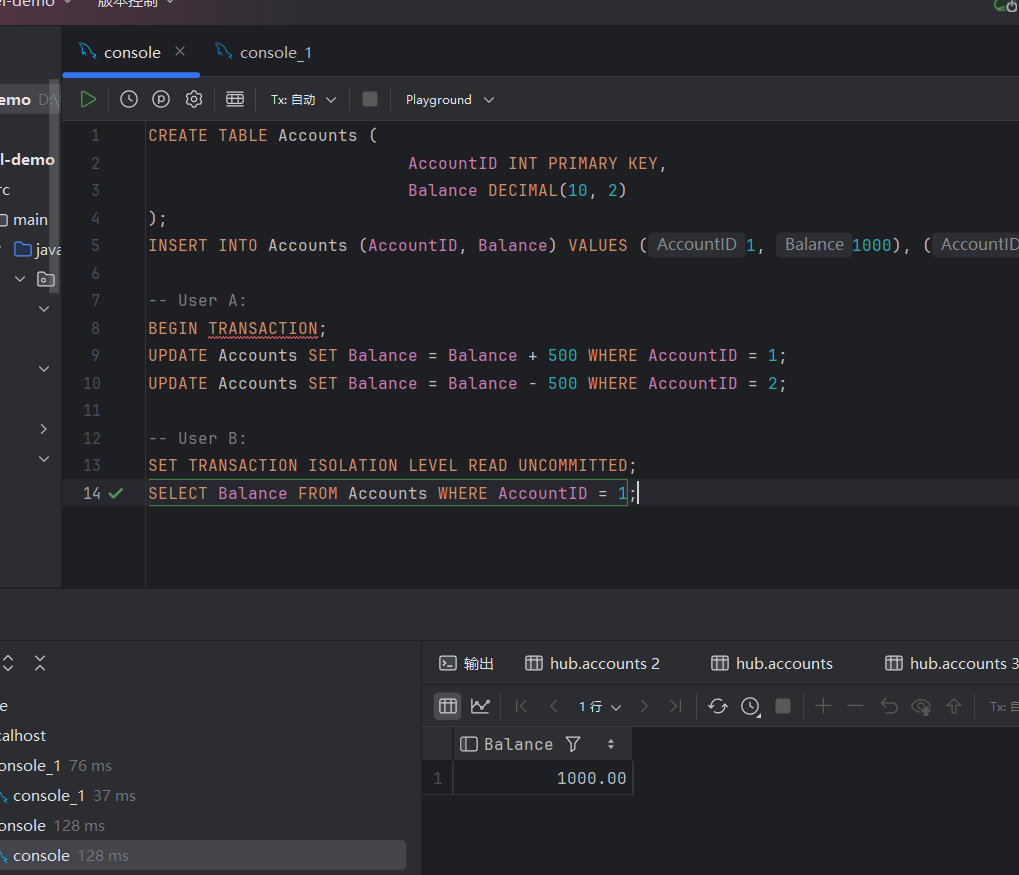
UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;

UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;

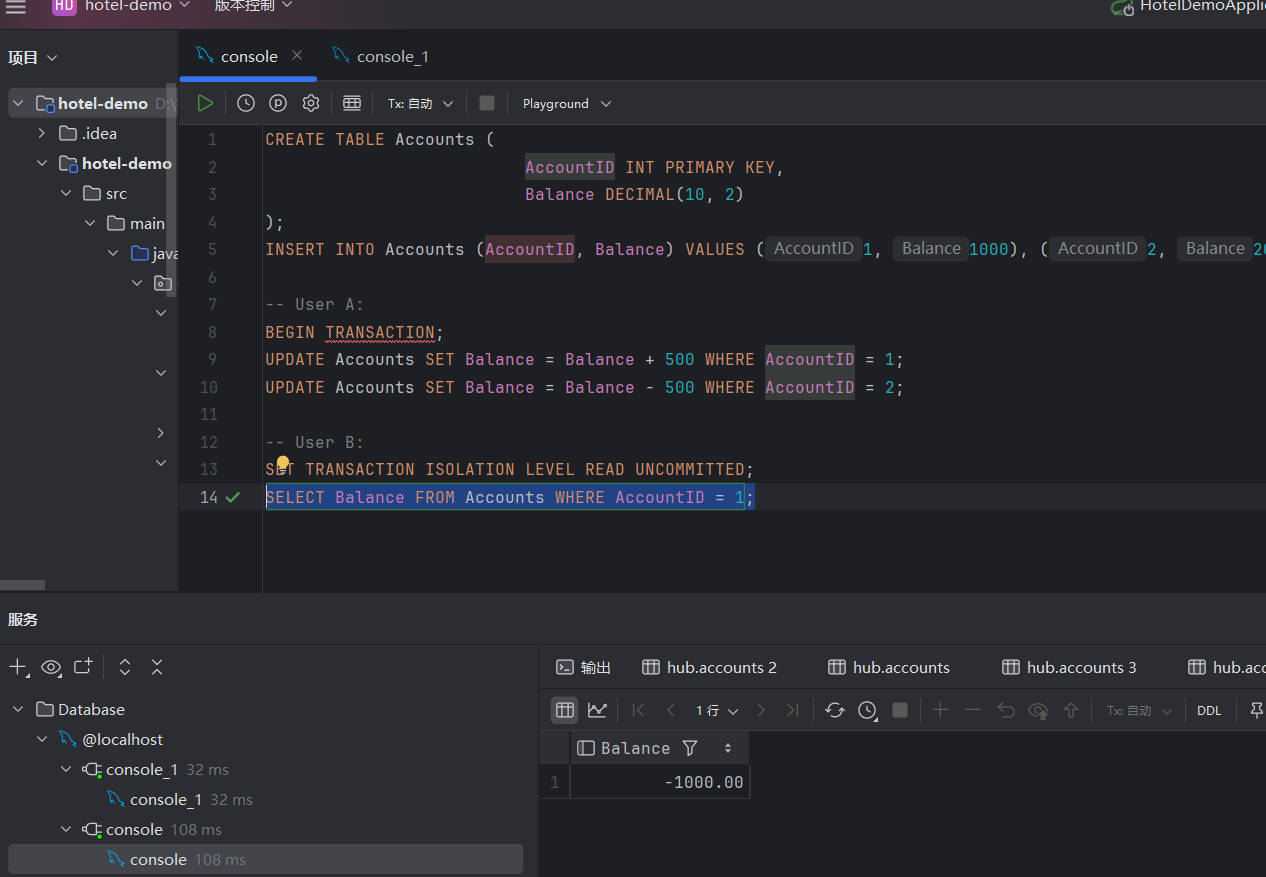
-- User B:

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT Balance FROM Accounts WHERE AccountID = 1;



在commit后，发现以及发生了修改



场景2: 读已提交（Read Committed）

使用此隔离级别，User B 将不会看到 User A 的未提交修改。

-- User A:

BEGIN TRANSACTION;

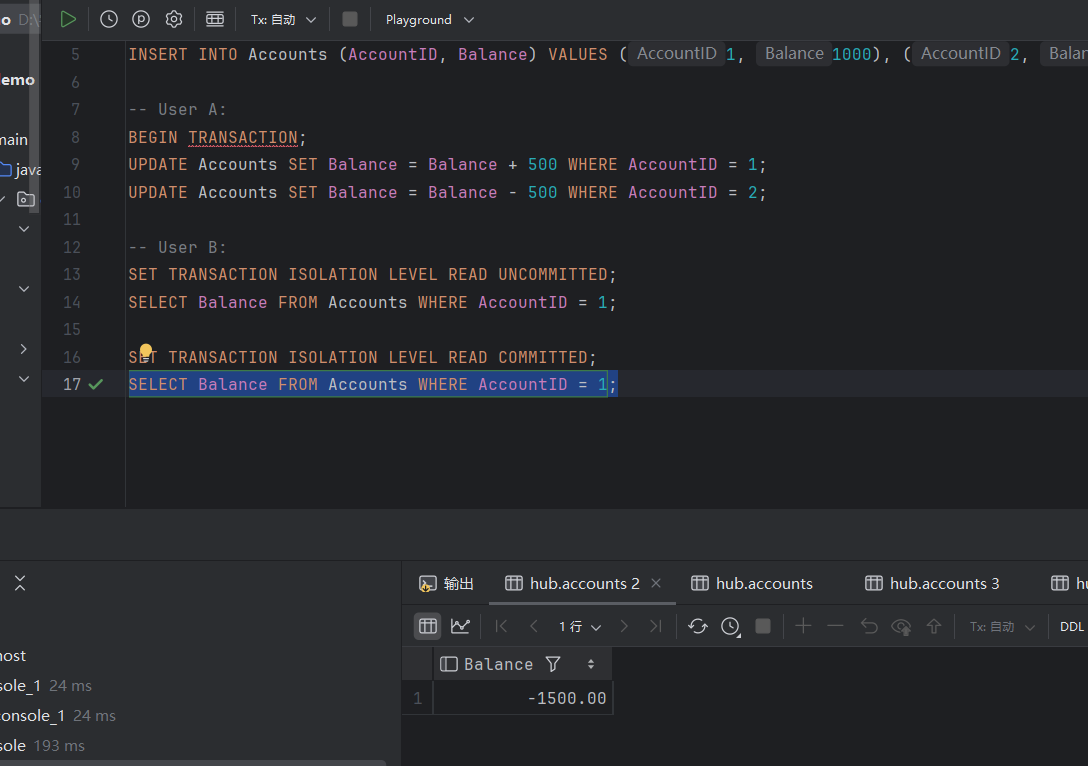
UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;

UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;

-- User B:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT Balance FROM Accounts WHERE AccountID = 1; -- 这里 User B 不会看到未提交的数据



场景3: 可重复读（Repeatable Read）

如果 User B 开始了一个事务，并在 User A 提交之前读取了 Account 1 的余额，那么即使 User A 的事务提交了，User B 再次读取时也会看到第一次读取的结果，因为可重复读防止了不可重复读的问题。

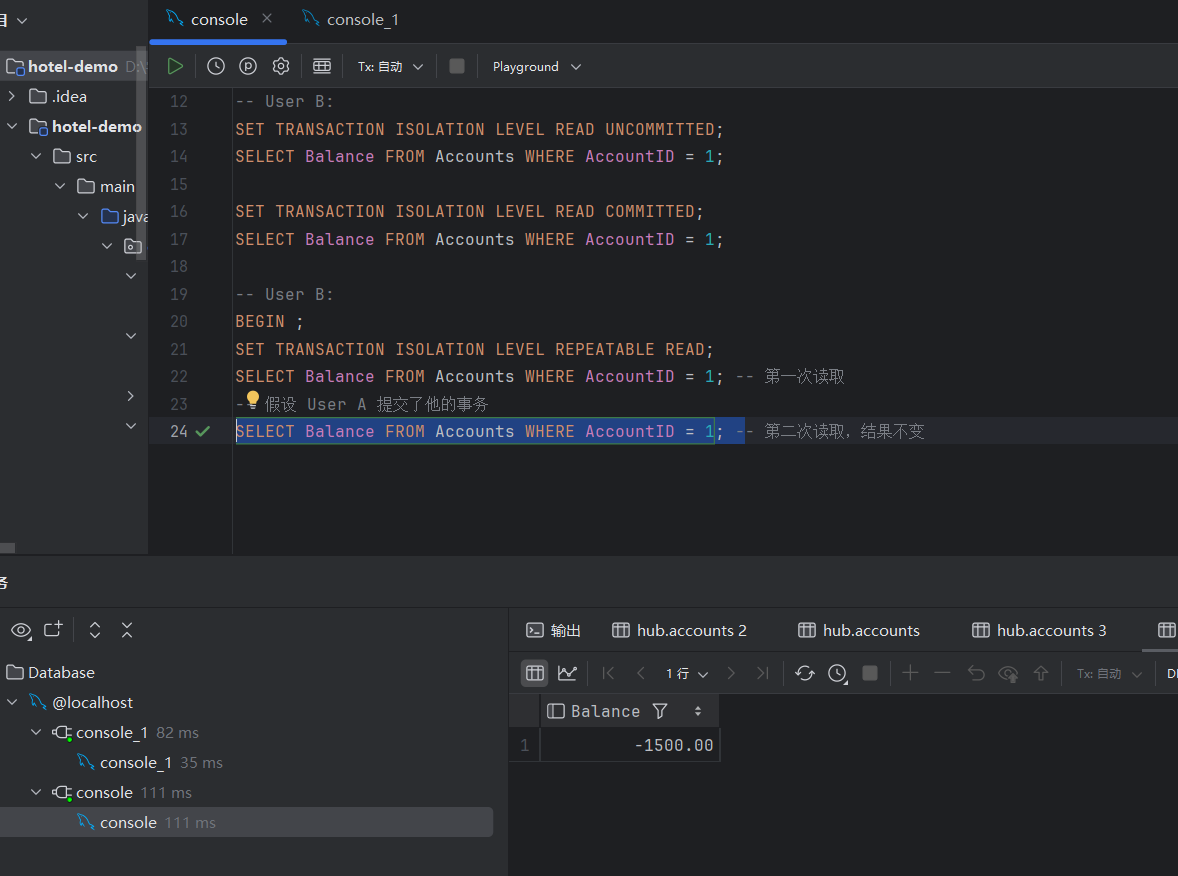
-- User B:

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SELECT Balance FROM Accounts WHERE AccountID = 1; -- 第一次读取

SELECT Balance FROM Accounts WHERE AccountID = 1; -- 第二次读取，结果不变



场景4: 可串行化（Serializable）

此隔离级别确保所有事务都是完全串行化的，即好像它们是按顺序执行的一样。这意味着在 User B 的事务完成之前，User A 的事务不能对 Account 1 或 Account 2 进行任何修改。

sql

深色版本

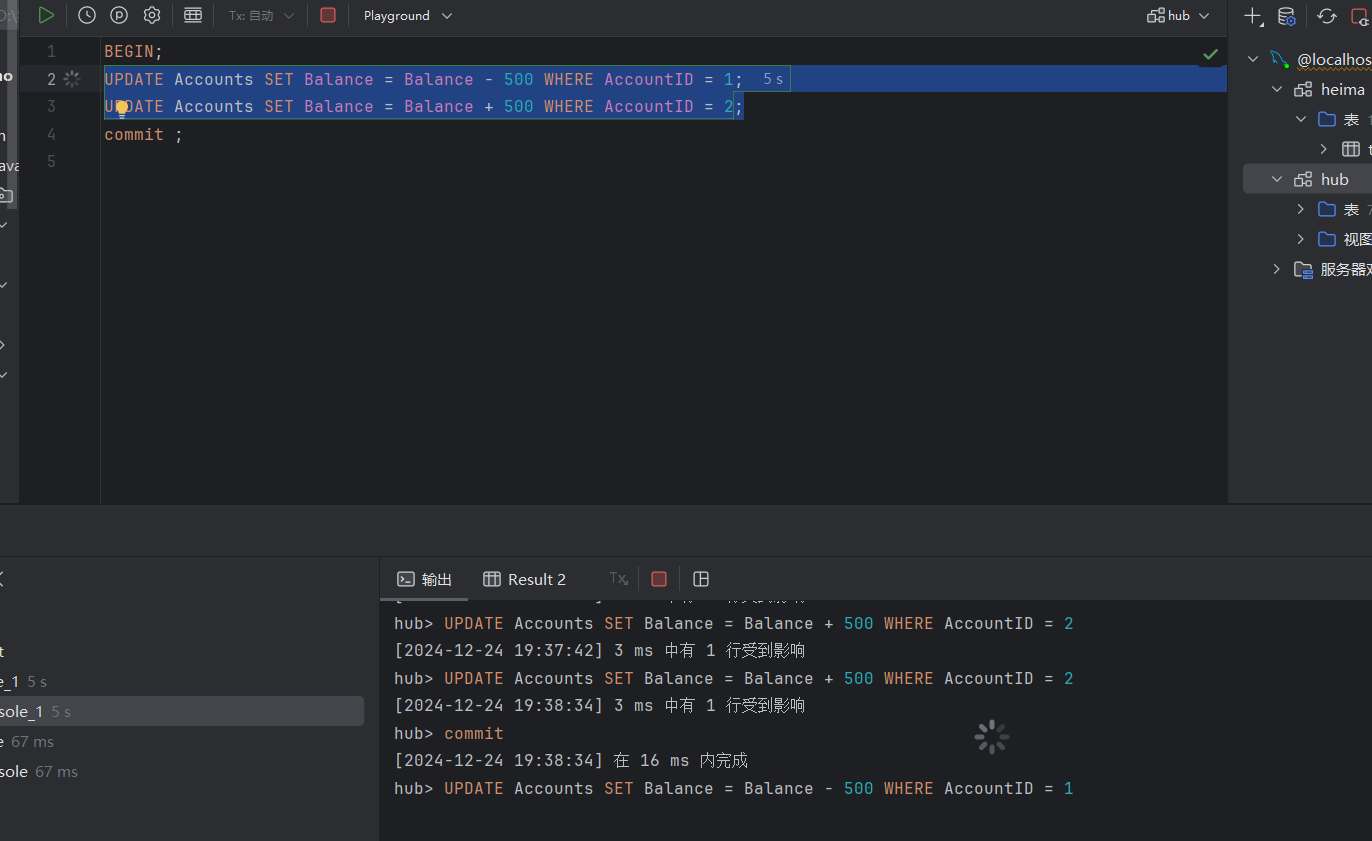
-- User B:

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT Balance FROM Accounts WHERE AccountID = 1;

-- User A 尝试执行他的转账操作，将会被阻塞直到 User B 的事务结束



# 课后习题

1.试分析在“读已提交”隔离级别下，设计一个应用程序以避免“不可重复读”和“幻影”。

假设我们有一个 Accounts 表，其中包含账户信息，并且我们想要实现一个功能，该功能允许用户查询特定账户的余额两次，并确保两次查询的结果一致（即避免不可重复读），同时确保在查询期间没有新的账户被插入到结果集中（即避免幻影读）。

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

Balance DECIMAL(10, 2)

) ENGINE=InnoDB;

我们将创建一个存储过程来模拟这个应用程序的行为。我们将设置隔离级别为 READ COMMITTED 并演示如何使用锁来防止不可重复读和幻影读。

DELIMITER $$

CREATE PROCEDURE QueryAccountBalance(IN account\_id INT)

BEGIN

-- 设置隔离级别为 READ COMMITTED

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

-- 开始一个新的事务

START TRANSACTION;

-- 使用 SELECT ... FOR UPDATE 来锁定行，防止其他事务修改或删除这些行

SELECT 'First Read' AS 'Read Type', Balance

FROM Accounts

WHERE AccountID = account\_id

FOR UPDATE;

-- 模拟处理时间，以增加并发操作的可能性

DO SLEEP(2);

-- 再次查询同一账户的余额，确保数据一致性

SELECT 'Second Read' AS 'Read Type', Balance

FROM Accounts

WHERE AccountID = account\_id

FOR UPDATE;

-- 提交事务

COMMIT;

END$$

DELIMITER ;

测试场景

场景 1: 避免不可重复读

User A 调用 QueryAccountBalance 存储过程查询 AccountID = 1 的余额。

在 User A 的第一次查询和第二次查询之间，User B 尝试更新 AccountID = 1 的余额。

因为 FOR UPDATE 锁定了 AccountID = 1 的行，所以 User B 的更新操作会被阻塞，直到 User A 的事务完成。这保证了 User A 的两次查询结果一致。

场景 2: 避免幻影读

如果我们希望进一步避免幻影读，可以考虑对整个表或者特定范围的数据加锁。例如，如果我们想确保在查询期间没有新的账户被插入到某个范围内：

-- 假设我们有一个查询条件是 Balance > 1000

START TRANSACTION;

SELECT \* FROM Accounts WHERE Balance > 1000 FOR UPDATE;

COMMIT;

在这个例子中，FOR UPDATE 不仅锁定了满足条件的行，还阻止了其他事务在这个范围内插入新的记录，从而避免了幻影读现象。

**实验总结：**

**在这次实验之后，我掌握了数据库并发控制的基本原理及其应用方法**。

**教师评语及成绩**：