**《数据库系统原理》实验报告**

### **实验题目：**实验17 并发控制技术实验

**姓名：**

**实验日期：**

**实验内容及完成情况：**（可续页）

### 实验17 并发控制技术实验

**实验学时：2学时**

**实验类型：验证**

**实验要求：选做**

**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。

**二、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。

**三、思考题**

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”。

数据库表：

-- 账户表

CREATE TABLE accounts (

account\_id INT PRIMARY KEY,

balance DECIMAL(10, 2),

version INT DEFAULT 1

);

-- 订单表

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

amount DECIMAL(10, 2)

);

python程序：

import pymysql

# 数据库连接配置

DB\_CONFIG = {

"host": "localhost",

"user": "root",

"password": "password",

"database": "test\_db",

"autocommit": False # 关闭自动提交，手动管理事务

}

def avoid\_non\_repeatable\_read(account\_id):

"""

避免不可重复读：读取账户余额，并通过版本号检查是否被修改

"""

connection = pymysql.connect(\*\*DB\_CONFIG)

try:

with connection.cursor() as cursor:

# 第一次读取账户余额和版本号

cursor.execute("SELECT balance, version FROM accounts WHERE account\_id = %s", (account\_id,))

result = cursor.fetchone()

if result is None:

print("Account not found.")

return

balance, version = result

print(f"Initial Balance: {balance}, Version: {version}")

# 模拟其他事务可能的更新操作

input("Simulate update in another transaction and press Enter to continue...")

# 第二次读取并检查版本号

cursor.execute("SELECT balance, version FROM accounts WHERE account\_id = %s", (account\_id,))

new\_balance, new\_version = cursor.fetchone()

if new\_version != version:

print("Non-repeatable read detected!")

else:

print(f"No changes detected. Balance: {new\_balance}, Version: {new\_version}")

finally:

connection.rollback() # 确保事务回滚，保持测试环境稳定

connection.close()

def avoid\_phantom\_read(min\_amount):

"""

避免幻读：统计高金额订单数量，使用加锁机制

"""

connection = pymysql.connect(\*\*DB\_CONFIG)

try:

with connection.cursor() as cursor:

# 第一次统计高金额订单

cursor.execute("SELECT COUNT(\*) FROM orders WHERE amount > %s FOR UPDATE", (min\_amount,))

initial\_count = cursor.fetchone()[0]

print(f"Initial Count of Orders > {min\_amount}: {initial\_count}")

# 模拟其他事务插入新的订单

input("Simulate insert in another transaction and press Enter to continue...")

# 第二次统计高金额订单

cursor.execute("SELECT COUNT(\*) FROM orders WHERE amount > %s FOR UPDATE", (min\_amount,))

new\_count = cursor.fetchone()[0]

if new\_count != initial\_count:

print("Phantom read detected!")

else:

print(f"No phantom read detected. Order Count: {new\_count}")

finally:

connection.rollback() # 确保事务回滚

connection.close()

if \_\_name\_\_ == "\_\_main\_\_":

print("Choose an option:")

print("1. Avoid Non-Repeatable Read")

print("2. Avoid Phantom Read")

choice = input("Enter your choice: ")

if choice == "1":

account\_id = int(input("Enter Account ID: "))

avoid\_non\_repeatable\_read(account\_id)

elif choice == "2":

min\_amount = float(input("Enter Minimum Order Amount: "))

avoid\_phantom\_read(min\_amount)

else:

print("Invalid choice.")

### ****程序说明****

#### ****避免不可重复读****

1. 在查询账户余额时，读取版本号。
2. 模拟其他事务更新同一账户。
3. 再次读取余额并比较版本号，发现数据是否被修改。

#### ****避免幻读****

1. 对高金额订单的统计结果加锁（FOR UPDATE）。
2. 模拟其他事务插入新订单。
3. 再次统计订单数量并检测是否发生幻读。

### ****测试方法****

1. 打开两个数据库会话，一个运行程序，另一个手动执行更新或插入语句。
2. 检查程序是否能正确检测到不可重复读或幻读的情况。

此设计通过显式锁机制和事务管理，解决了在“读已提交”隔离级别下的问题。

**实验总结：**

通过本次实验，我深入理解了 **数据库并发控制** 的基本原理，尤其是 **事务的隔离级别**（如读未提交、读已提交、可重复读和可序列化）对数据一致性的影响。在实际操作中，我熟练运用了显式加锁（如 SELECT ... FOR UPDATE）和事务管理策略，解决了 **脏读**、**不可重复读** 和 **幻读** 等常见问题。

此外，通过设计和测试多事务并发场景，我更深刻地体会到 **并发控制机制**（如悲观锁、乐观锁）的实际应用方法，以及如何在性能和一致性之间做出权衡。本次实验不仅巩固了数据库理论知识，也提升了实际开发中的问题排查和解决能力，为未来处理复杂并发场景奠定了坚实基础。

**教师评语及成绩**：