Assignment 3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

ACID Properties of a Transaction

The ACID properties are crucial for ensuring reliable transaction processing in databases. Here's a brief explanation of each property:

1. **Atomicity** : Ensures that all operations within a transaction are completed; if any part of the transaction fails, the entire transaction is rolled back, leaving the database in its previous state.

2. **Consistency** : Ensures that a transaction brings the database from one valid state to another, maintaining database invariants (e.g., integrity constraints).

3. **Isolation** : Ensures that concurrently executing transactions do not affect each other's execution. This means the intermediate states of a transaction are invisible to other transactions until the transaction is committed.

4. **Durability** : Ensures that once a transaction has been committed, it remains so, even in the event of a system failure. Changes made by the transaction are permanently stored in the database.

SQL Statements to Simulate a Transaction with Locking and Different Isolation Levels

Below is an example of SQL statements simulating a transaction that includes locking and demonstrates different isolation levels.

Setup

First, ensure you have a sample table to work with:

```
CREATE TABLE IF NOT EXISTS Accounts (
  AccountID INT PRIMARY KEY AUTO_INCREMENT,
  AccountHolder VARCHAR(100) NOT NULL,
  Balance DECIMAL(10, 2) NOT NULL
);

INSERT INTO Accounts (AccountHolder, Balance) VALUES
('Alice', 1000.00),
('Bob', 1500.00);
```

Simulate a Transaction with Locking

In this example, we simulate transferring money from Alice's account to Bob's account, using explicit locking:

```
START TRANSACTION;

-- Lock Alice's and Bob's rows for update
SELECT Balance FROM Accounts WHERE AccountHolder = 'Alice' FOR UPDATE;
SELECT Balance FROM Accounts WHERE AccountHolder = 'Bob' FOR UPDATE;

-- Perform the transfer
UPDATE Accounts SET Balance = Balance - 100.00 WHERE AccountHolder = 'Alice';
UPDATE Accounts SET Balance = Balance + 100.00 WHERE AccountHolder = 'Bob';

-- Commit the transaction
COMMIT;
```

The `FOR UPDATE` clause locks the selected rows, ensuring no other transactions can modify them until the current transaction is completed.

Demonstrate Different Isolation Levels

Isolation levels control the visibility of data changes in a transaction to other transactions. The four standard isolation levels are `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.

1. READ UNCOMMITTED

This level allows a transaction to read uncommitted changes made by other transactions, potentially leading to dirty reads.

l

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

START TRANSACTION;

-- Simulate a read operation

SELECT * FROM Accounts;

COMMIT;

```

**2. READ COMMITTED**

This level ensures that any data read is committed at the moment it is read, preventing dirty reads but allowing non-repeatable reads.

sql

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;

-- Simulate a read operation

SELECT * FROM Accounts;

COMMIT;

**3. REPEATABLE READ**

This level ensures that if a transaction reads the same row twice, it will see the same
```

data both times, preventing non-repeatable reads but allowing phantom reads.

```
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;
-- Simulate a read operation
SELECT * FROM Accounts;
-- Another read operation within the same transaction
SELECT * FROM Accounts;
COMMIT;
```

4. SERIALIZABLE

This is the highest isolation level, ensuring complete isolation from other transactions. It prevents dirty reads, non-repeatable reads, and phantom reads by acquiring locks on all rows it reads.

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;
-- Simulate a read operation
SELECT * FROM Accounts;
COMMIT;
```

Concurrency Control Demonstration

To demonstrate concurrency control, consider two transactions:

Transaction 1:

```sql
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;
-- Read Alice's balance
SELECT Balance FROM Accounts WHERE AccountHolder = 'Alice';

-- Simulate a delay (e.g., due to some business logic processing)
DO SLEEP(5);

-- Update Alice's balance
UPDATE Accounts SET Balance = Balance - 50 WHERE AccountHolder = 'Alice';
COMMIT;
```

Transaction 2:

```sql
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;
```

```
-- Read Alice's balance
SELECT Balance FROM Accounts WHERE AccountHolder = 'Alice';

-- Update Alice's balance
UPDATE Accounts SET Balance = Balance - 50 WHERE AccountHolder = 'Alice';
COMMIT;
```