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

ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable processing of database transactions.

**Atomicity**: This ensures that all operations within a transaction are completed successfully. If any operation fails, the entire transaction fails and the database state is left unchanged.

Example: If a bank transfer transaction involves debiting one account and crediting another, both operations must succeed. If one fails, neither account is updated.

**Consistency**: This ensures that a transaction brings the database from one valid state to another, maintaining database integrity constraints.

Example: After a transfer, the total amount of money in the accounts should remain the same.

**Isolation**: This ensures that the intermediate state of a transaction is invisible to other transactions until it is complete, preventing transactions from interfering with each other.

Example: While one transaction is updating a record, other transactions cannot access this record in an inconsistent state.

**Durability**: This ensures that once a transaction is committed, it will remain so, even in the event of a system failure.

Example: After a successful transfer, the changes remain in the database even if the system crashes immediately afterward.

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

This is a database with a table accounts for explaining a bank transfer transaction.

CREATE TABLE accounts (

account\_id INT PRIMARY KEY,

balance DECIMAL(10, 2));

INSERT INTO accounts (account\_id, balance) VALUES (1, 1000.00), (2, 2000.00);

**Transaction with Locking:**

To simulate a transaction, we use BEGIN TRANSACTION and COMMIT or ROLLBACK.

-- Start transaction

BEGIN TRANSACTION;

-- Lock the rows to ensure atomicity and isolation

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 2;

-- Commit the transaction

COMMIT;

**Note:** If an error occurs, we would ROLLBACK instead of COMMIT.

**ISOLATION LEVELS**

SQL databases support different isolation levels to control the visibility of changes made by concurrent transactions.

The four standard isolation levels are:

1. **Read Uncommitted**: Allows dirty reads.
2. **Read Committed**: Prevents dirty reads.
3. **Repeatable Read**: Prevents dirty reads and non-repeatable reads.
4. **Serializable**: Prevents dirty reads, non-repeatable reads, and phantom reads.

Explanation of all isolation levels.

1. **Read Uncommitted**

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE account\_id = 1; -- Possible dirty read

ROLLBACK;

1. **Read Committed**

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE account\_id = 1; -- Prevents dirty reads

ROLLBACK;

1. **Repeatable Read**

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE account\_id = 1; -- Prevents dirty and non-repeatable reads

ROLLBACK;

1. **Serializable**

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE account\_id = 1; -- Prevents dirty, non-repeatable reads, and phantom reads

ROLLBACK;

### Demonstrating Concurrency Control

Consider two concurrent transactions trying to update the accounts table.

-- Transaction 1

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

-- Lock the row for reading and writing

SELECT balance FROM accounts WHERE account\_id = 1 FOR UPDATE;

-- Simulate some processing delay

WAITFOR DELAY '00:00:10';

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

-- Commit Transaction 1

COMMIT;

-- Transaction 2 (run concurrently with Transaction 1)

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

-- This will wait for Transaction 1 to complete due to the lock

SELECT balance FROM accounts WHERE account\_id = 1 FOR UPDATE;

UPDATE accounts SET balance = balance - 50 WHERE account\_id = 1;

-- Commit Transaction 2

COMMIT;

In this example, Transaction 2 will wait until Transaction 1 completes due to the serializable isolation level, ensuring the highest level of concurrency control.