<u>Assignment 3</u>: 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.

Solution:-

The ACID properties are fundamental principles that ensure the reliability and integrity of transactions within a database system.

Atomicity: This property ensures that a transaction is treated as a single unit of work. Either all the operations within the transaction are successfully completed, or none of them are. There's no partial execution. If any part of the transaction fails, the entire transaction is rolled back to its initial state.

Consistency: This property ensures that the database remains in a consistent state before and after the transaction. It means that any data modifications performed by a transaction must abide by all the constraints, rules, and relationships defined in the database schema. The integrity of the data is maintained throughout the transaction.

Isolation: This property ensures that the execution of multiple transactions concurrently does not result in interference or inconsistency. Each transaction should operate independently of other transactions, as if it were the only transaction executing on the database. Isolation prevents transactions from seeing each other's intermediate states, ensuring that the outcome of each transaction is consistent and predictable.

Durability: This property ensures that once a transaction is committed, its changes are permanently saved in the database and survive system failures such as crashes or power outages. Even if the system crashes immediately after a transaction commits, the changes made by that transaction should still be present when the system recovers.

Now, for demonstrating different isolation levels and concurrency control in SQL, let's simulate a scenario where multiple transactions are accessing and modifying the same data concurrently. We'll use SQL statements to showcase different isolation levels.

```
CREATE TABLE Account (
   account_id INT PRIMARY KEY,
   balance DECIMAL(10, 2));

INSERT INTO Account (account_id, balance) VALUES (1, 1000.00);
```

We'll run two transactions concurrently, each updating the balance of the account:

```
Transaction1:
START TRANSACTION;

UPDATE Account SET balance = balance - 100 WHERE account_id = 1;
-- Simulate some processing time
SELECT SLEEP(5);
```

```
UPDATE Account SET balance = balance + 100 WHERE account_id = 1;

COMMIT;
```

Transaction 2:

```
START TRANSACTION;

UPDATE Account SET balance = balance - 50 WHERE account_id = 1;
-- Simulate some processing time
SELECT SLEEP(5);
UPDATE Account SET balance = balance + 50 WHERE account_id = 1;

COMMIT;
```

We can set different isolation levels for each transaction using the SET TRANSACTION ISOLATION LEVEL statement before starting the transaction. For example:
READ COMMITTED: Transaction 1 can read and modify committed data, but it will not see uncommitted changes made by Transaction 2.

REPEATABLE READ: Transaction 1 will not see changes made by Transaction 2 until it commits.

SERIALIZABLE: Transactions are completely isolated from each other. Neither Transaction 1 nor Transaction 2 can see changes made by the other until both transactions commit.

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
```

By running these transactions with different isolation levels, you can observe how the database manages concurrency and isolation.