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

**ACID Properties of a Transaction:**

A transaction is a single, all-or-nothing unit of work that ensures data consistency and integrity in a database. The ACID properties of a transaction guarantee that database transactions are processed reliably and securely. The four ACID properties are:

1. **Atomicity:**

* Ensures that a transaction is treated as a single, indivisible unit of work. If any part of the transaction fails, the entire transaction is rolled back, and the database is returned to its previous state.

1. **Consistency:**

* Ensures that a transaction maintains the consistency of the database by enforcing constraints, triggers, and rules.

1. **Isolation:**

* Ensures that multiple transactions can execute concurrently without interfering with each other's execution. This property ensures that the effects of one transaction are not visible to other transactions until it is committed.

1. **Durability:**

* Ensures that once a transaction is committed, its effects are permanent and cannot be rolled back.

To demonstrate different isolation levels in SQL, we will use two transactions that concurrently update the balance of a bank account. Here are the SQL statements to create the table and insert initial data:

**CREATE TABLE accounts (**

**acctnum INT PRIMARY KEY,**

**balance DECIMAL(10, 2)**

**);**

**INSERT INTO accounts (acctnum, balance) VALUES (12345, 100.00);**

**INSERT INTO accounts (acctnum, balance) VALUES (7534, 200.00);**

**Read Uncommitted Isolation Level:**

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 7534;

SELECT balance FROM accounts WHERE acctnum = 12345;

COMMIT;

**Read Committed Isolation Level:**

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 7534;

COMMIT;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE acctnum = 12345;

COMMIT;

**Repeatable Read Isolation Level:**

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE acctnum = 12345;

**-- Run the following transaction in another session:**

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 7534;

COMMIT;

SELECT balance FROM accounts WHERE acctnum = 12345;

COMMIT;

**Serializable Isolation Level:**

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT balance FROM accounts WHERE acctnum = 12345;

**-- Run the following transaction in another session:**

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 7534;

COMMIT;

SELECT balance FROM accounts WHERE acctnum = 12345;

COMMIT;

In the above examples, the first transaction transfers $100 from account 12345 to account 7534, while the second transaction reads the balance of account 12345. The different isolation levels affect the behaviour of the second transaction, as follows:

* In the Read Uncommitted isolation level, the second transaction reads the uncommitted changes made by the first transaction, resulting in a dirty read.
* In the Read Committed isolation level, the second transaction reads the committed changes made by the first transaction, resulting in a committed read.
* In the Repeatable Read isolation level, the second transaction reads the same data throughout its duration, even if the first transaction commits changes.
* In the Serializable isolation level, the second transaction cannot read data that has been modified but not yet committed by the first transaction, and no other transactions can modify data that has been read by the second transaction until it is committed.

These examples demonstrate how different isolation levels can affect the behaviour of concurrent transactions in SQL.