Assignment –3

ACID Properties:

1. **Atomicity:** This property ensures that a transaction is either completed in its entirety or not at all. In other words, all operations within a transaction are treated as a single unit. If any part of the transaction fails, the entire transaction is rolled back, leaving the database in its original state.
2. **Consistency:** Consistency ensures that the database remains in a valid state before and after the transaction. This means that the integrity constraints, such as foreign key constraints, uniqueness constraints, etc., are always enforced. Transactions cannot leave the database in a state that violates these constraints.
3. **Isolation:** Isolation ensures that the operations within a transaction are independent of other concurrent transactions. Even if multiple transactions are executing simultaneously, each transaction should appear to be executing in isolation. This prevents interference between transactions and ensures data integrity.
4. **Durability:** Durability guarantees that once a transaction is committed, its effects persist even in the event of system failures, such as power outages or crashes. The changes made by committed transactions are permanently stored in the database and are not lost.

SQL Statements for Simulating a Transaction:

Let's consider an example scenario where we have a banking application, and we want to transfer funds from one account to another within a transaction.

```
BEGIN TRANSACTION;

-- Deduct amount from sender's account
UPDATE accounts
SET balance = balance - 100
WHERE account_number = 'sender_account_number';

-- Add amount to receiver's account
UPDATE accounts
SET balance = balance + 100
WHERE account_number = 'receiver_account_number';
```

```
COMMIT;
```

In this SQL transaction, we're deducting $100 from the sender's account and adding $100 to the receiver's account. If any of the **UPDATE** statements fail, the entire transaction will be rolled back.

Different Isolation Levels for Concurrency Control:

SQL databases offer different isolation levels to control the visibility of changes made by concurrent transactions. Let's demonstrate this using different isolation levels:

5. **Read Uncommitted:** This is the lowest isolation level where transactions can see uncommitted changes made by other transactions.

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

6. **Read Committed:** Transactions can only see changes committed by other transactions. This prevents dirty reads but may allow non-repeatable reads and phantom reads.

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

7. **Repeatable Read:** Transactions guarantee that if a query is executed twice within the same transaction, the result set will remain the same, even if other transactions commit changes in the meantime. This prevents non-repeatable reads but may allow phantom reads.

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

8. **Serializable:** This is the highest isolation level where transactions are completely isolated from each other. It ensures that even phantom reads are not allowed by locking the entire range of records that a query accesses until the transaction is completed.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

These isolation levels offer different trade-offs between data consistency and concurrency.