ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

ACID Properties

The expansion of the term ACID defines for:

1. Atomicity.
2. Consistancy.
3. Isolation.
4. Durability

1) Atomicity

The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

2) Consistency

The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

3) Isolation

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets

complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

4) Durability

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

SQL Transactions with Locking and Isolation Levels

Let's create an example using two tables: Accounts and Transactions.

Setup

```
-- Create Accounts table

CREATE TABLE Accounts (

    AccountID INT AUTO_INCREMENT PRIMARY KEY,

    AccountHolder VARCHAR(100) NOT NULL,

    Balance DECIMAL(10, 2) NOT NULL

);
```

```
-- Create Transactions table
CREATE TABLE Transactions (
    TransactionID INT AUTO_INCREMENT PRIMARY KEY,
    AccountID INT,
    Amount DECIMAL(10, 2) NOT NULL,
    TransactionDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);

-- Insert sample data
INSERT INTO Accounts (AccountHolder, Balance) VALUES ('Alice', 1000.00), ('Bob', 500.00);
```

Example Transaction with Locking

Atomic Transfer from Alice to Bob

```
START TRANSACTION;

-- Lock Alice's account to prevent other transactions from modifying it
SELECT Balance FROM Accounts WHERE AccountID = 1 FOR UPDATE;

-- Lock Bob's account to prevent other transactions from modifying it
SELECT Balance FROM Accounts WHERE AccountID = 2 FOR UPDATE;

-- Update Alice's balance
```

```
UPDATE Accounts SET Balance = Balance - 200.00 WHERE AccountID = 1;

-- Update Bob's balance
UPDATE Accounts SET Balance = Balance + 200.00 WHERE AccountID = 2;

-- Insert into Transactions table
INSERT INTO Transactions (AccountID, Amount) VALUES (1, -200.00), (2, 200.00);

COMMIT;
```

Isolation Levels

Read Uncommitted

```
-- Set isolation level to Read Uncommitted
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

START TRANSACTION;

-- Alice reads Bob's balance, which might include uncommitted changes
SELECT Balance FROM Accounts WHERE AccountID = 2;

COMMIT;
```

Read Committed

```
-- Set isolation level to Read Committed
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

START TRANSACTION;

-- Alice reads Bob's balance, which only includes committed changes

SELECT Balance FROM Accounts WHERE AccountID = 2;

COMMIT;

Repeatable Read

-- Set isolation level to Repeatable Read

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

-- Alice reads Bob's balance, which remains consistent throughout the transaction

SELECT Balance FROM Accounts WHERE AccountID = 2;

-- Perform some operation (e.g., after some other transaction might have attempted an update)

-- This read will still see the initial balance

SELECT Balance FROM Accounts WHERE AccountID = 2;

COMMIT;

Serializable:

-- Set isolation level to Serializable

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

-- Alice reads Bob's balance
SELECT Balance FROM Accounts WHERE AccountID = 2;

-- Any other transaction trying to update Bob's balance will be blocked until this transaction completes

-- Perform an update
UPDATE Accounts SET Balance = Balance + 50.00 WHERE AccountID = 2;

COMMIT;
```

Explanation of Isolation Levels

Read Uncommitted: Transactions can see uncommitted changes from other transactions. This can lead to dirty reads.

Read Committed: Transactions only see committed changes from other transactions, preventing dirty reads.

Repeatable Read: Ensures that if a transaction reads a value at the start, it will see the same value if it reads again, preventing non-repeatable reads.

Serializable: Provides the highest level of isolation by ensuring that transactions are executed sequentially, effectively preventing phantom reads and ensuring full isolation.