**Acid property**

In the context of databases, the term "ACID" is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These are the four key properties that ensure reliability and consistency in database transactions. Let's define each property with examples and their uses:

1. Atomicity:
   * Definition: Atomicity ensures that either all operations of a transaction complete successfully and the database is updated, or none of them are performed at all (i.e., the transaction is rolled back).
   * Example: Consider a bank transfer where money is debited from one account and credited to another. Atomicity ensures that if the credit operation succeeds, the debit operation also succeeds, and vice versa.
   * Use: It guarantees that the database remains in a consistent state even if a transaction fails midway, preventing situations where only part of a transaction is applied, which could lead to inconsistencies.
2. Consistency:
   * Definition: Consistency ensures that the database transitions from one valid state to another valid state after a transaction, maintaining all integrity constraints.
   * Example: In an online shopping scenario, when a customer places an order, inventory levels should be updated immediately to reflect the correct stock after the transaction is committed.
   * Use: It ensures that any transaction will bring the database from one valid state to another, adhering to all defined rules and constraints.
3. Isolation:
   * Definition: Isolation ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, one after another.
   * Example: If two users are simultaneously updating the same bank account, isolation ensures that each transaction is executed independently and without interference from other transactions, preserving data integrity.
   * Use: It prevents "dirty reads," "non-repeatable reads," and "phantom reads," thereby maintaining transaction integrity in a multi-user environment.
4. Durability:
   * Definition: Durability guarantees that once a transaction is committed, it will remain so even in the event of a system failure (such as power loss or crash).
   * Example: After a successful funds transfer in banking, the updated balance should persist and not be lost even if the system crashes immediately after committing the transaction.
   * Use: It ensures that committed transactions are permanently saved and can be recovered in the face of failures, thus preserving data consistency and integrity over time.

 **Atomicity:**

* **Example:** Consider a banking application where a user transfers money from one account to another. The transaction involves deducting an amount from one account and crediting it to another.
* **SQL Usage:** To ensure atomicity, SQL provides transaction control statements such as BEGIN TRANSACTION, COMMIT, and ROLLBACK. For example:

sql

Copy code

BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountNumber = '123456';

UPDATE Accounts SET Balance = Balance + 100 WHERE AccountNumber = '654321';

COMMIT;

If any part of the transaction fails (e.g., due to an error or constraint violation), the entire transaction can be rolled back to maintain atomicity.

 **Consistency:**

* **Example:** In an e-commerce database, when a customer places an order, the database must ensure that the order details are saved correctly and consistently.
* **SQL Usage:** SQL enforces consistency through constraints (e.g., primary keys, foreign keys, check constraints) and transactions. For example:

sql

Copy code

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

Constraints ensure that each order is associated with a valid customer, maintaining database integrity and consistency.

 **Isolation:**

* **Example:** Multiple users concurrently accessing and updating the same database records. Isolation ensures that each transaction sees a consistent database state and does not interfere with others.
* **SQL Usage:** SQL uses locking mechanisms and isolation levels (e.g., Read Committed, Repeatable Read, Serializable) to manage concurrent transactions. For example:

sql

Copy code

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

-- Perform SELECT or UPDATE statements here

COMMIT;

Different isolation levels control how concurrent transactions interact with each other, preventing phenomena like dirty reads or non-repeatable reads.

 **Durability:**

* **Example:** After a transaction commits successfully, the changes made to the database should persist even in the event of a system crash or failure.
* **SQL Usage:** SQL ensures durability by logging transactions and using mechanisms like write-ahead logging (WAL) to ensure that committed changes are written to stable storage (disk) before acknowledging the transaction commit. For example:

sql

Copy code

BEGIN TRANSACTION;

-- Perform INSERT, UPDATE, DELETE operations

COMMIT;

Once COMMIT is executed, the changes are durable, meaning they will survive system crashes or failures.