The ACID principles are a set of properties that ensure reliable processing of database transactions. ACID stands for Atomicity, Consistency, Isolation, and Durability. These principles are essential to maintain the integrity and reliability of a database system, especially in scenarios involving concurrent transactions and system failures. Here is an explanation of each principle with examples:

### Atomicity

Atomicity ensures that each transaction is treated as a single, indivisible unit. This means that either all operations within the transaction are completed successfully, or none of them are applied.

\*\*Example:\*\*

Consider a banking system where you want to transfer $100 from Account A to Account B. This operation involves two steps:

1. Deduct $100 from Account A.

2. Add $100 to Account B.

If the transaction is atomic, both steps must succeed or both must fail. If the system crashes after deducting $100 from Account A but before adding $100 to Account B, the entire transaction is rolled back, leaving both accounts unchanged.

```sql

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 'A';

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 'B';

COMMIT;

```

If an error occurs between the two `UPDATE` statements, the transaction will be rolled back, ensuring atomicity.

### Consistency

Consistency ensures that a transaction brings the database from one valid state to another valid state, maintaining the integrity constraints of the database.

\*\*Example:\*\*

Using the same banking example, consistency would ensure that the total amount of money remains the same before and after the transaction. If there are integrity constraints (e.g., balances cannot be negative), these constraints must be enforced at the end of the transaction.

```sql

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 'A' AND balance >= 100;

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 'B';

COMMIT;

```

Here, the `balance >= 100` constraint ensures that Account A does not go into a negative balance, maintaining consistency.

### Isolation

Isolation ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, one after the other.

\*\*Example:\*\*

Consider two transactions:

- Transaction 1: Transfer $100 from Account A to Account B.

- Transaction 2: Transfer $200 from Account B to Account A.

If these transactions run concurrently without isolation, the intermediate states might corrupt the final balances. Proper isolation levels (e.g., Serializable, Repeatable Read, Read Committed) prevent such anomalies.

```sql

-- Transaction 1

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 'A';

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 'B';

COMMIT;

-- Transaction 2

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

UPDATE accounts SET balance = balance - 200 WHERE account\_id = 'B';

UPDATE accounts SET balance = balance + 200 WHERE account\_id = 'A';

COMMIT;

```

Setting the isolation level to `SERIALIZABLE` ensures that the transactions are executed in isolation, preventing intermediate inconsistencies.

### Durability

Durability ensures that once a transaction has been committed, it remains committed even in the case of a system failure. The changes made by the transaction are permanently recorded in the database.

\*\*Example:\*\*

After transferring $100 from Account A to Account B and committing the transaction, the changes should be durable. Even if the database server crashes immediately after the commit, the changes should be preserved when the server is restarted.

```sql

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 'A';

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 'B';

COMMIT;

```

After the `COMMIT` statement, the database system ensures that the changes are written to non-volatile storage, making the transaction durable.

In summary, the ACID properties (Atomicity, Consistency, Isolation, Durability) are crucial for ensuring the correctness, reliability, and robustness of database transactions. They help manage the complexities of concurrent operations and system failures, maintaining the integrity and reliability of the data.