**ACID** (Atomicity, Consistency, Isolation, Durability) are properties that guarantee the reliability of transactions in database systems. Let's break down each property with an example in SQL:

1. Atomicity: Atomicity ensures that a transaction is treated as a single unit, which either completes fully or not at all. If any part of the transaction fails, the entire transaction fails, and the database state remains unchanged.

Example: Suppose you have a banking application where you want to transfer money from one account to another. The atomicity property ensures that if the debit operation (removing money from one account) succeeds, the corresponding credit operation (adding money to another account) also succeeds. If either operation fails (due to system crash, power outage, or any other reason), both operations are rolled back to maintain consistency.

**sql**

BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = '123'; -- Debit $100 from AccountID '123'

UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = '456'; -- Credit $100 to AccountID '456'

COMMIT TRANSACTION; -- If successful, commit the changes

```

2. **Consistency**: Consistency ensures that only valid data is written to the database. The database should be in a consistent state before and after the transaction.

**Example**: In an online shopping application, when a customer places an order, the database should ensure that the products ordered are deducted from the inventory only if all items are available. If any item is out of stock, the entire transaction (order) should be rolled back to maintain consistency.

**sql**

BEGIN TRANSACTION;

-- Check if all items are available

SELECT COUNT(\*) INTO @AvailableItems FROM Inventory WHERE ProductID IN ('1', '2', '3') AND Stock > 0;

IF @AvailableItems = 3 THEN

-- Deduct items from inventory and insert order details

UPDATE Inventory SET Stock = Stock - 1 WHERE ProductID IN ('1', '2', '3');

INSERT INTO Orders (OrderID, CustomerID, ProductID, Quantity) VALUES ('123', '456', '1', 1);

INSERT INTO Orders (OrderID, CustomerID, ProductID, Quantity) VALUES ('123', '456', '2', 1);

INSERT INTO Orders (OrderID, CustomerID, ProductID, Quantity) VALUES ('123', '456', '3', 1);

ELSE

-- Rollback the transaction if items are not available

ROLLBACK TRANSACTION;

END IF;

COMMIT TRANSACTION; -- Commit changes if everything is successful

```

3. **Isolation**: Isolation ensures that multiple transactions can occur concurrently without affecting each other. Each transaction should be independent of and unaware of other transactions occurring at the same time.

Example: If two users attempt to update the same bank account simultaneously, the isolation property ensures that their transactions are processed as if they were executed sequentially, maintaining data consistency.

**sql**

-- User 1's transaction

BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance - 50 WHERE AccountID = '123';

COMMIT TRANSACTION;

-- User 2's transaction (concurrently executed)

BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance + 50 WHERE AccountID = '123';

COMMIT TRANSACTION;

```

4. **Durability:** Durability guarantees that once a transaction is committed, it will remain committed even in the case of a system failure (e.g., power outage or crash). The changes made by the committed transaction are permanent and persist in the system.

**Example**: After a successful transfer of funds between accounts, even if the database server crashes immediately afterward, the changes (debit from one account and credit to another) are still reflected when the system restarts.

**sql**

BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = '123'; -- Debit $100 from AccountID '123'

UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = '456'; -- Credit $100 to AccountID '456'

COMMIT TRANSACTION; -- Once committed, changes are durable

```

In summary, ACID properties ensure that database transactions are reliable, predictable, and maintain data integrity in various scenarios, providing a robust foundation for transaction management in SQL databases.