**What Does @Transactional Actually Do?**

When you put @Transactional on a method, you’re basically telling Spring:

"Hey Spring, wrap this method in a database transaction. Make sure everything inside it happens completely—or not at all."

It's like a safety net for your database operations.

**Why Is It Important?**

You're transferring money between two accounts:

You deduct ₹500 from Account A.

You add ₹500 to Account B.

If something crashes after step 1 but before step 2, someone just lost ₹500.

With @Transactional, both steps are wrapped in a single transaction. If anything goes wrong:

Spring rolls everything back (like hitting “Undo”)

The database stays safe and consistent

You don’t have to manually manage the transaction

What Happens Behind the Scenes?When Spring sees @Transactional, it:

Opens a Hibernate session (if using JPA)

Starts a database transaction

Runs your method

Commits if everything goes well

Rolls back if there’s an error or exception

All of this happens automatically. No need to write .beginTransaction() or .commit() manually.

**Where Should You Use It?**

Service layer methods (where business logic lives)

Methods that:

Write to the database

Update multiple tables

Should not leave the DB in half-done state

Example:

@Transactional

public void updateCountryInfo(String code, String newName) {

Country country = countryRepository.findById(code).orElseThrow();

country.setName(newName);

// changes are saved together when the method completes

}