Transactions in Spring Boot

give me the deep and detailed explaination of Transactions in spring boot so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explaination

Below is a **very deep, step-by-step, zero-confusion explanation of Transactions in Spring Boot**, written in **simple language**, with **real flow**, **examples**, **internal working**, and **common mistakes**. Read it once slowly—you will not have doubts after this.

🔥 Transactions in Spring Boot – Complete Deep Explanation

1️⃣ What is a Transaction? (Real-Life First)

A **transaction** is a **group of operations** that must be treated as **one single unit of work**.

🧠 Real-life example (Bank Transfer)

You transfer ₹1000 from **Account A → Account B**

Steps:

1. Deduct ₹1000 from Account A
2. Add ₹1000 to Account B

✔️ If **both succeed** → transaction successful
❌ If **any one fails** → everything must be **rolled back**

This **all-or-nothing rule** is a **transaction**.

2️⃣ Why Transactions are REQUIRED in Applications

Without transactions:

- Partial data may be saved ❌
- Database becomes inconsistent ❌
- Business logic breaks ❌

With transactions:

- Data remains **safe**
- Operations are **atomic**
- System becomes **reliable**

3️⃣ ACID Properties (VERY IMPORTANT)

Every transaction follows **ACID**:

🔹 A – Atomicity

> Either **all operations succeed** or **none happen**

Example:

- Debit ✔️
- Credit ❌
  → Rollback debit

🔹 C – Consistency

> Database moves from **one valid state to another**

Example:

- Balance never becomes negative
- Foreign keys remain valid

🔹 I – Isolation

> Multiple transactions should **not interfere** with each other

Example:

- Two users updating same record

- Each sees **correct data**

🔹 D – Durability

> Once committed, data **will not be lost**

Even if:

- Server crashes
- Application restarts

4️⃣ Transactions in Spring Boot – Big Picture

Spring Boot uses **Spring's Transaction Management**, which is part of the **Spring Framework**.

Internally it works with:

- **JPA**
- **Hibernate**
- **JDBC**
- Database (MySQL, PostgreSQL, etc.)

5️⃣ @Transactional – The HEART of Transactions

Spring provides `@Transactional` to manage transactions **automatically**.

📌 What `@Transactional` does internally

When a method starts:

1. Open database connection
2. Begin transaction
3. Execute method logic
4. If success → **COMMIT**
5. If exception → **ROLLBACK**
6. Close connection

You **do not write commit/rollback manually**.

6️⃣ Basic Example (VERY IMPORTANT)

❌ Without Transaction

```java
public void transferMoney() {
    debitAccount();
    creditAccount(); // fails here
}
```

Result: ❌ Money lost

✅ With Transaction

```java
@Transactional
public void transferMoney() {
    debitAccount();
    creditAccount();
}
```

Result:

- Success → Commit
- Failure → Rollback

7️⃣ Where to Use @Transactional (Best Practice)

✅ Use on Service Layer

```java
@Service
public class PaymentService {

    @Transactional
    public void makePayment() {
```

```
        // business logic
    }
}
```

❌ Do NOT use on:

- Controller
- Repository (unless required)
- Private methods

8️⃣ How Spring Actually Implements Transactions (INTERNAL FLOW)

Spring uses **AOP (Aspect Oriented Programming)**.

Internal Flow:

```
pgsql

Client
 ↓
Spring Proxy
 ↓
Transaction Manager
 ↓
Business Method
 ↓
Commit / Rollback
```

👉 Spring creates a **proxy class** around your service.

9️⃣ Transaction Manager (CRITICAL)

Spring uses different **Transaction Managers**:

Technology	Transaction Manager
JDBC	DataSourceTransaction Manager
JPA / Hibernate	JpaTransactionManager

Spring Boot **auto-configures** it for you.

🔟 Rollback Rules (VERY IMPORTANT)

Default Behavior

Exception Type	Rollback
RuntimeException	✅ YES
Error	✅ YES
Checked Exception	❌ NO

Force rollback for checked exception

```java
@Transactional(rollbackFor = Exception.class)
```

1️⃣1️⃣ Propagation (MOST CONFUSING – Explained Simply)

Propagation defines **how transactions behave when methods call each other**.

🔹 REQUIRED (DEFAULT)

- Join existing transaction
- Create new if none exists

```java
@Transactional
```

🔹 REQUIRES_NEW

- Suspend existing
- Create new transaction

```java
@Transactional(propagation = Propagation.REQUIRES_NEW)
```

Used for:

- Logging
- Audit tables

🔹 SUPPORTS

- Run in transaction if exists
- Else run without

🔹 MANDATORY

- Must have transaction
- Else exception

🔹 NOT_SUPPORTED

- Always run without transaction

🔹 NEVER

- Fail if transaction exists

1️⃣ 2️⃣ Isolation Levels (Database Concurrency Control)

Isolation controls **data visibility**.

Level	Problems Prevented
READ_UNCOMMITTED	Nothing
READ_COMMITTED	Dirty Reads

Level	Problems Prevented
REPEATABLE_READ	Non-repeatable Reads
SERIALIZABLE	All problems

Example:

```java
@Transactional(isolation = Isolation.READ_COMMITTED)
```

1️⃣3️⃣ Common Transaction Problems (INTERVIEW GOLD)

❌ Self Invocation Problem

```java
@Transactional
public void methodA() {
    methodB(); // ❌ Transaction not applied
}

@Transactional
public void methodB() {}
```

Why?

- Proxy not used
- Call happens inside same class

✅ Fix: Move method to another service

❌ Private Methods

Transactions **do NOT work** on private methods.

❌ Catching Exception

```java
try {
   // code
} catch(Exception e) {
   // swallowed → no rollback
}
```

Solution:

```java
throw e;
```

1️⃣4️⃣ Read-Only Transactions (Performance Boost)

```java
@Transactional(readOnly = true)
public List<User> getUsers() {}
```

Benefits:

- Faster
- Prevents accidental writes

1️⃣5️⃣ Real-World Example (COMPLETE)

```java
@Service
public class OrderService {

    @Transactional
    public void placeOrder() {
        saveOrder();
        updateInventory();
        makePayment();
    }
}
```

If **any step fails** → everything rolled back.

1️⃣6️⃣ Transaction + Database

Spring transactions rely on **database transactions**.

For example:

- MySQL uses InnoDB
- PostgreSQL supports ACID fully

👉 If DB does not support transactions → Spring cannot help.

1️⃣7️⃣ Summary (FINAL REVISION)

✔ Transaction = Unit of work
✔ ACID guarantees safety
✔ `@Transactional` manages commit/rollback
✔ Default rollback = RuntimeException
✔ Service layer is best place
✔ Propagation & Isolation control behavior
✔ Spring uses proxy + AOP
✔ Auto-configured in Spring Boot

🔥 Final Advice for You (Piyush)

Since you are **seriously learning Spring Boot backend**, mastering transactions is **non-negotiable**.
This topic:

- Appears in **interviews**
- Causes **real production bugs**
- Separates **beginners from professionals**

If you want next:
👉 **Transactions + Spring Data JPA**
👉 **Distributed Transactions**

👉 **Real interview questions**

👉 **Debugging transaction issues**

Just tell me 👍

Do you like this personality?