SPRING BOOT MYSQL TRANSACTIONS

Encontré algunos tutoriales en español, pero ninguno era de calidad aceptable.

# Spring Boot Transaction Management + JPA Example

<https://www.codeusingjava.com/boot/trans/1>

Arrancamos con este documento, que me parece muy claro. Hay algunas diferencias, pero la explicación se puede seguir muy bien.

El proyecto demo es una aplicación que maneja una base de datos, con soporte de transacciones. El método insert() se usa para provocar una excepción, y esa excepción provoca que la transacción en cuyo transcurso se produce sea vuelta atrás, en lugar de ser confirmada. A consecuencia del rollback() nada cambia en la base de datos.

## Introducción

Una transacción es un conjunto de operaciones lógicas que deben realizarse de manera definida y ordenada. Transaction Management es una metodología efectiva que ayuda a lograr transacciones eficientes y sin ambigüedades. Transaction Management tiene como objetivo ofrecer consistencia y validez de los datos facilitando las características ACID.

## ¿Qué es ACID?

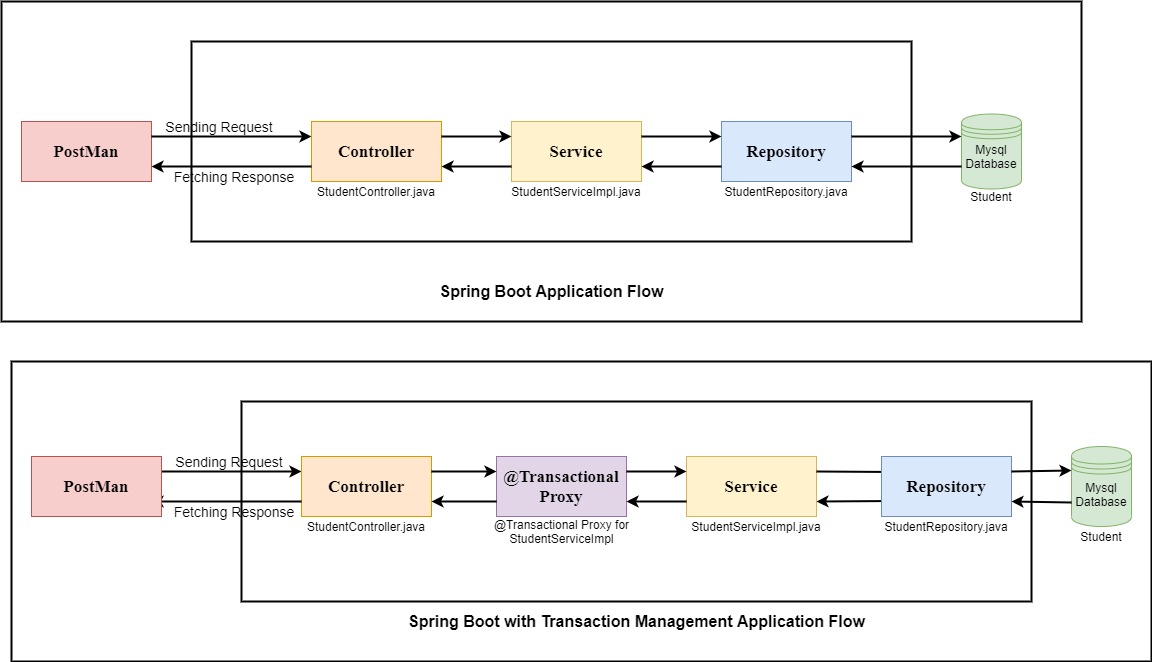
ACID significa Atomicidad, Consistencia, Aislamiento y Durabilidad que ayudan a manejar las excepciones y a mantener un seguimiento adecuado de las transacciones procesadas.

* **Atomicidad**: garantiza que se deban realizar todas o ninguna de las operaciones.
* **Coherencia**: garantiza que todas las operaciones se reviertan y se establezcan en el estado en el que se inició la transacción o que los cambios se hayan realizado y reflejado correctamente.
* **Aislamiento**: garantiza que las transacciones realizadas solo se reflejen después de una confirmación.
* **Durabilidad** - Asegura que las transacciones comprometidas se mantengan.

## Operaciones para controlar transacciones ACID usando JDBC

* Desactive auto-commit, esto le proporciona el control para ejecutar múltiples declaraciones SQL dentro de una sola transacción; de lo contrario, terminará ejecutando cada instrucción SQL como una transacción separada.
* Cada transacción debe confirmarse llamando al método commit(), esto permite que esos cambios persistan dentro de la base de datos de forma permanente, de lo contrario, los cambios no se reflejarán en la base de datos.
* En caso de que se arroje un error o una excepción dentro de la lógica de negocio, o que el SQL no se pueda ejecutar, revierta todas las operaciones llamando al método rollback(), de lo contrario, habrá una inconsistencia en los datos grabados en la base de datos.

## Diagrama del mecanismo de las transacciones

PostMan es una herramienta que se usa para enviar requests de prueba al backend. Las clases Controller, Service, Repository son conocidas y no las vamos a explicar. En la figura de abajo vemos el proxy transaccional, que es el mecanismo que da soporte automáticamente a las transacciones. El patrón se llama inyección de dependencias. Asegurarse de haber entendido bien esta figura, porque describe exactamente lo que va a suceder en la aplicación que estamos programando.

## Códigos fuente

Los códigos fuente están en el proyecto demo dentro de este mismo directorio. Los códigos fuente son los que están en el proyecto, no los del tutorial.

Los archivos que hay que revisar y asegurarse de entender son:

* src\main\resources\application.properties
* pom.xml
* src\main\java\tup\demo\DemoApplication.java
* src\main\java\tup\demo\controllers\StudentController.java
* src\main\java\tup\demo\repositories\StudentRepository.java
* src\main\java\tup\demo\services\StudentService.java
* src\main\java\tup\demo\services\StudentServiceImpl.java
* src\main\java\tup\demo\models\Student.java

Los comentarios en los archivos ayudan a entender.

Student.java es una clase POJO (Plain Old Java Object[[1]](#footnote-1)).

* La anotación @Entity realiza el mapeo de la clase Student con la tabla student\_info.
* La anotación @Table permite personalizar el nombre de la tabla de student a student\_info.
* La anotación @Id marca un atributo como clave principal.

Cree el RestController para realizar operaciones de StudentRegistration. Conectaremos automáticamente StudentServiceImpl y lo usaremos para realizar operaciones de servicio.

StudentController actúa como una puerta de enlace (gateway) para acceder a los servicios de la aplicación y realizar las operaciones según el tipo de solicitud. Dicho de otro modo, se ocupa de recibir los requests, y según la URL decidir a qué método de la clase Service va a llamar, y qué parámetros le va a pasar.

La clase StudentServiceImpl implementa la interfaz StudentService siguiendo el modelo MVC. Notar que la interfaz no tiene anotaciones. Las anotaciones no se heredan. La práctica es poner las anotaciones solo en la clase concreta que las necesita inmediatamente. La clase StudentServiceImpl tiene la anotación @Service.

StudentServiceImpl.java implementa la metodología de gestión de transacciones al poner la anotación @Transactional que crea un contenedor alrededor de las funciones de StudentServiceImpl, como retrieve, insert, delete, etc.

La anotación @Transactional incluye automáticamente el código necesario para deshabilitar auto\_commit() y para soportar Commit() y Rollback(). Esto ayuda a revertir la operación de la base de datos en caso de que surja alguna excepción o error, lo que garantiza la coherencia y la eficacia en el procesamiento de datos mediante la implementación de la propiedad ACID.

**Notar:**

* La anotación @Transaction funcionará solo para los métodos públicos de la clase anotada por la anotación @Service.
* Es necesario que la clase de servicio StudentServiceImple sea Autowired, de lo contrario, la anotación @Transactional no funcionará.

## Listo

Compilar y arrancar la aplicación.

## Uso de PostMan

En Postman tenemos que agregar un request.

Tipo POST.

La URL es <http://localhost:8080/insert>

Dentro del request ir a la solapa Body, seleccionar raw y pegar lo siguiente:

{

    "studentId":9,

    "studentName":"Hulk",

    "studentAge":145

}

Por supuesto que hay que ir poniendo los datos apropiados. En particular, el studentId es el que se usa dentro del método insert() para provocar la excepción.

En el dropdown que está a la derecha, asegurarse de seleccionar JSON.

Listo. Ahora solo queda enviarlo con Send. Recordar que la aplicación debe estar corriendo.

# Spring Boot Transaction Management

<https://www.youtube.com/watch?v=XL0EROsn5Yc&ab_channel=JavaCodeHouse>

Este otro tutorial esmuy pero muy bueno. Conviene mirarlo después de haber hecho el proyecto anterior. Dura 11 minutos. Inglés muy claro. La autora da una explicación breve y completa de las transacciones.

# Spring & Spring Data JPA: Managing Transactions

<https://www.youtube.com/watch?v=SUQxXg229Xg&ab_channel=ThorbenJanssen>

Este otro tutorial tiene un montón de reproducciones. El autor parece ser bastante popular. Dura 10 minutos. El inglés es muy claro, aunque se nota el acento del autor. Creo que conviene hacerlo después del anterior.

# Spring Boot Tutorial | How To Handle Exceptions

<https://www.youtube.com/watch?v=PzK4ZXa2Tbc&ab_channel=Amigoscode>

Estos son viejos conocidos. Hacen excelentes tutoriales. Este dura 14 minutos. El inglés es muy claro. En realidad este tutorial es un extracto de un curso más largo. Da por conocido un contexto que no explica. Para entenderlo antes hay que haber hecho el proyecto demo y haber mirado los dos tutoriales anteriores. Sobre esa base, lo que este explica se entiende. Claro que hay que aplicar lo que él dice a este sistema.

# Anotaciones

## CUIDADO

Hay que verificar qué biblioteca estamos importando, porque por ejemplo Java también soporta la anotación @Transacional, pero tiene un par de opciones menos que Spring.

## Transacción (informática)

<https://es.wikipedia.org/wiki/Transacci%C3%B3n_(inform%C3%A1tica)>

## @EnableTransactionManagement

Spring Boot usa transacciones de manera muy simple, primero use la anotación @EnableTransactionManagement para abrir el soporte de transacciones, y luego agregue la anotación @Transactional en cada método de servicio que necesite acceder a la base de datos y que necesite ser transaccional.

La anotación @EnableTransactionManagement se puede agregar en la clase principal, inmediatamente después de la anotación @SpringBootApplication.

La anotación @Transactional inyecta en el método anotado todas las dependencias y las funcionalidades para wrappearlo en una transacción, que comienza automáticamente antes de iniciarse el método que uno programó, se comete automáticamente si todo sale bien, o se vuelve atrás automáticamente si algo sale mal. Todo lo que ocurre entre el inicio automático y la finalización automática de la transacción queda incluido en la transacción.

El comportamiento de la transacción se puede customizar mediante los atributos que la anotación soporta.

* propagation: controla el comportamiento del método cuando ingresa en el contexto de una transacción.
* readonly: para acceder a datos que no se necesita editar. Mejora la performance y es más seguro.
* rollbackfor y norollbackfor controlan el comportamiento del método ante determinadas situaciones.

1. <https://es.wikipedia.org/wiki/Plain_Old_Java_Object> [↑](#footnote-ref-1)