Capa de Servicios

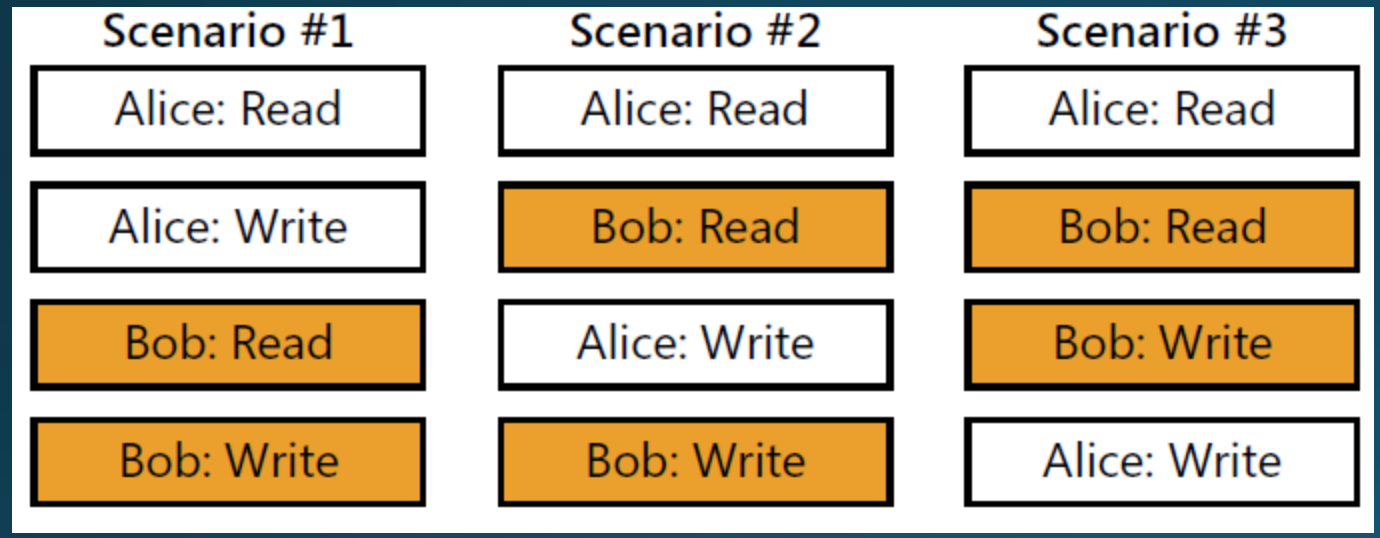
Universidad Centroamericana "José Simeón Cañas"

Ciclo 01-2020

Transacciones

- Conjunto de sentencias SQL que se ejecutan como un “todo”, a manera de asegurar la calidad de los datos en una base de datos
- Se dice que una operación es atómica cuando se ejecuta todo o nada de ella.
- Por ejemplo en una transacción bancaria de transferencia de fondos, se decrementa de la cuenta origen y se incrementa la cuenta destino, esta operación debe ser atómica, asegurando que ambas operaciones se llevarán a cabo.

Caso de ejemplo



Alice obtiene el asiento 100 y Bob el asiento 101

Alice y Bob reciben el asiento 100 como disponible. Alice reserva primero pero Bob sobrescribe el registro

Alice y Bob reciben el asiento 100 como disponible, Bob reserva primero y luego Alice sobrescribe a Bob

Considerar el siguiente caso: Alice y Bob reservarán un asiento en un teatro. Los asientos están numerados.

El proceso de reservación es el siguiente:

1. Lee la tabla de reservaciones para encontrar el siguiente asiento vacío
2. Actualiza la fila para asignar al cliente que se sentará en el asiento vacío

Reglas de la norma ACID

- **Atomicidad:** La transacción es todo o nada. Si alguna parte de la transacción no se completa toda la operación es cancelada y deshecha.
- **Consistencia:** Cuando la transacción es completada esta deja la base de datos en un "estado válido". Por ejemplo al borrar un registro de una tabla padre, las respectivas filas de las tablas hijos que hacían referencia a ese registro deben ser eliminados (ON CASCADE DELETE).
- **Aislamiento:** Cuando una transacción está activa si otros procesos quieren acceder a algún registro involucrado debe proveerles del contenido previo a la transacción de esos registros o bloquear el acceso al proceso hasta que la transacción este completa
- **Durabilidad:** Se refiere a la capacidad de la base de datos de protegerse ante fallos en esta, así si los datos de la base de datos llegan a corromperse se pueden revertir a un estado válido. Las bases de datos logran esto usando *logs de transacciones* o *transaction logs.*

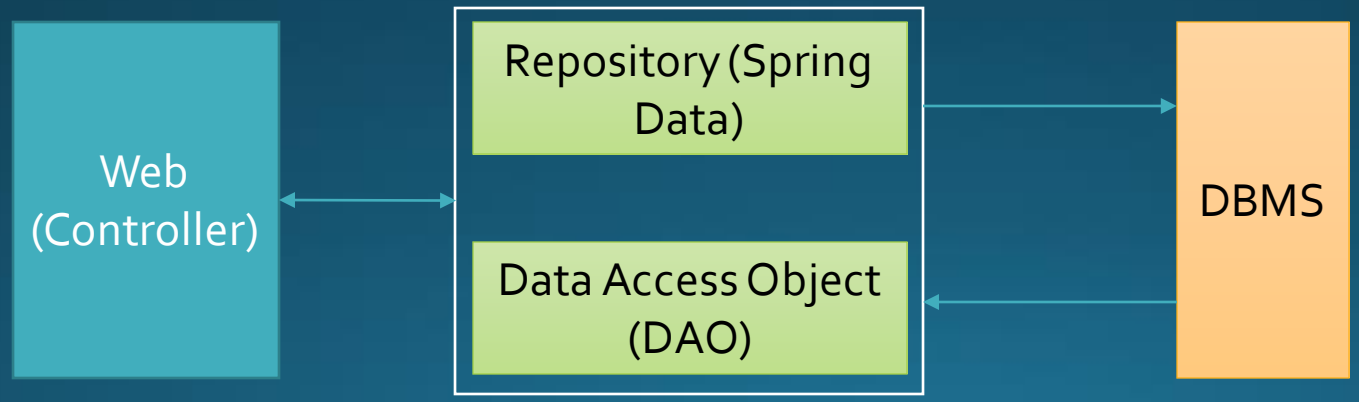
Transacciones (Cont.)

- Tres operaciones esenciales en una transacción:
  - BEGIN: Inicia una transacción
  - COMMIT: Hace a la transacción parte irreversible y permanente de la base de datos
  - ROLLBACK: Deshace las operaciones hechas en la transacción

Patrón de 4 capas

Muchas veces necesitamos ejecutar cierta lógica de negocio antes de persistir alguna información.

Los DAO y repositories, sin embargo, solamente proveen de acceso a la base de datos. Dicha lógica difiere de los objetivos de estos patrones.



Capa de Servicio

La capa de servicio nos permite introducer dicha lógica.

Spring nos permite crear interfaces y clases (como los DAOs) que representen un servicio determinado.

Normalmente, crearemos un servicio para cada entidad, como los DAOs, por ejemplo ClienteService, ProductoService, etc.

Spring nos provee de la anotación @Service, la cual nos permitirá inyectar automáticamente el objeto de servicio para su utilización (similar a la inyección de un DAO o repositorio)

La persistencia de los datos la haremos entonces desde el servicio y no desde el controlador. (Esto significa que ya no tendremos los DAOs en nuestro controlador)

Objetivo de la capa de servicio

- A parte de adicionar lógica de negocio, su objetivo es proveernos de métodos que estén manejados como transacciones
- Definimos los métodos como transaccionales con la anotación @Transactional
- Dicha anotación tiene la propiedad **rollbackFor,** al cual se le define la excepción por la cual queremos que la transacción se deshaga.
- Otra propiedad es **propagation**, que nos permite establacer el nivel de propagación de la transacción a otros métodos llamados desde el método del servicio

Propiedad **propagation**

- **Propagation.REQUIRED (Default):** Utiliza la transacción padre (método que lo mandó a llamar), crea una nueva si no existe ninguna.
- **Propagation.SUPPORTS:** Utiliza la transacción padre, pero no crea una transacción en caso no exista.
- **Propagation.MANDATORY:** Utiliza la transacción padre, lanza una excepción en caso no exista.
- **Propagation.REQUIRES_NEW:** Crea siempre una nueva transacción, totalmente aislada y sin relación a la transacción padre.
- **Propagation.NOT_SUPPORTED:** Ejecuta el método anotado de manera no transaccional, suspendiendo la transacción padre en caso existiera, y resumiendola posterior a la ejecución del método.

Visto de otra manera...

Método Padre	Método Hijo

Table 33-1 Transaction Attributes and Scope

Transaction Attribute	Client's Transaction	Business Method's Transaction
`Required`	None	T2
	T1	T1
`RequiresNew`	None	T2
	T1	T2
`Mandatory`	None	error
	T1	T1
`NotSupported`	None	None
	T1	None
`Supports`	None	None
	T1	T1

Arquitectura con capa de servicio