Servicios de Back-End

Las **transacciones** son un conjunto de acciones que realizan cambios de estado en recursos, representan una unidad básica de trabajo, los cambios se realizan mediante acciones (inserción, borrado, consultas…) y la transacción es indivisible ya que o se ejecuta entera o no se ejecuta.

Las transacciones son **ACID (atomicity, consistency, isolation and durability)**, estas siglas representan las siguientes propiedades:

- Atomicidad. La transacción es indivisible.
- Consistencia. Al finalizar si ejecución, el sistema debe quedar en estado estable. Si no se puede realizar, debe devolver el sistema a su estado inicial (rollback).
- Aislamiento. Una transacción no se ve afectada por otras. Los cambios no deben ser visibles hasta que la transacción finalice y es necesario que la transacción bloquee los recursos que va a actualizar.
- Durabilidad. Sus efectos son permanentes una vez que ha finalizado correctamente.

La ejecución de las transacciones es correcta si se ejecuta **en serie** (se ejecutan secuencialmente) o **seriabilizable**. Las seriabilizabilidad se puede conseguir mediante el uso de mecanismos de sincronización como los **bloqueos (locks)**.

Los sistemas tienen **invariantes**. Son relaciones que se deben cumplir siempre entre los componentes de un sistema informático.

Existen **estados consistentes** en los cuales el sistema está únicamente si se satisfacen todas sus invariantes. Cualquier cambio puede hacer pasar al sistema por un estado transitorio inconsistente. Si la transacción falla, el sistema debe quedar en un estado consistente.

Hay varios **modelos de transacciones** entre los cuales están:

- Transacciones planas. Todo proceso realizado en su interior se encuentra en el mismo nivel de jerarquía. Si el proceso fue correcto finaliza con un *commit_transaction*, si fallo, finaliza con un *rollback_transaction*.
  Estas transacciones están limitadas ya que en caso de error tenemos 2 opciones, o hacer rollback para volver al estado inicial (puede que tengamos que hacer una **actualización masiva** dependiendo del número de cambios hechos) o tratar de arreglar el error a partir del punto donde se produjo el fallo.
- Transacciones anidadas. Llamada de una transacción desde otra. Un commit de una transacción hace visibles todos sus cambios visibles a todas las transacciones precedentes en la jerarquía de llamadas.
- Transacciones con *savepoints*. Para evitar el efecto todo o nada se utilizan savepoints (puntos de control), cuando es necesario un rollback se vuelve al savepoint, pero los cambios solo serán visibles una vez se haga el commit.

Entonces ¿Cómo resolvemos el problema de la actualización masiva? **En lugar de realizar una transacción, la dividimos en varias.**

Para poder asegurar el aislamiento de las transacciones debemos tener el **control de la concurrencia**. Para tener este control existen las **leyes de la concurrencia**:

1. La ejecución concurrente no debe causar que los programas de aplicación funcionen incorrectamente.
2. La ejecución concurrente no debe tener menor rendimiento o tiempos de respuesta mayores que la ejecución en serie.

Es posible que se **viole el aislamiento** de las siguientes 3 formas:

- Actualización perdida. La escritura de una transacción es ignorada por otra, que realiza una nueva escritura basada en la versión previa del objeto.
- Lectura sucia. Una transacción lee un objeto escrito por otra en un estado intermedio.

- Lectura no repetible. Una transacción lee una actualización intermedia realizada por otra transacción.

Si se pueden prevenir estos 3 tipos de violaciones, se resuelve el problema del aislamiento.

En las transacciones existe la acción del **bloqueo**. Existen 2 tipos:

- Bloqueo compartido. No se requiere el uso exclusivo del objeto.
- Bloqueo exclusivo. Se requiere el uso exclusivo del objeto.

Con los bloqueos se crean 2 tipos de transacciones:

- Transacciones bien formadas: Todas sus lecturas y escrituras están cubiertas por bloqueos (READ cubierto por un SLOCK y WRITE cubierto por un XLOCK).
- Transacciones en 2 fases: Todos los bloqueos preceden a todos los desbloqueos.
    - Fase de crecimiento. Adquiere los bloqueos.
    - Fase de contracción. Libera los bloqueos.

Para lograr el aislamiento completo de cualquiera de las combinaciones posibles de un conjunto de transacciones se deben programar atendiendo a los siguientes criterios: Escribir siempre transacciones bien formadas, establecer bloqueos exclusivos en los objetos que se vayan a actualizar, escribir transacciones en 2 fases (no liberar bloqueos hasta que no se necesite realizar más bloqueos) y mantener los bloqueos exclusivos hasta que se realice el Commit o el rollback.

El aislamiento completo no evita las **lecturas fantasmas** asociadas a inserciones y borrado de objetos.

Dado que el aislamiento completo es costoso existen varios **grados de aislamiento**:

- Grado 0 (caos) → Las transacciones no sobrescriben datos sucios en transacciones de nivel 1.
- Grado 1 (lecturas no comprometidas) → No hay pérdida de actualizaciones en el ámbito de la transacción (hasta el COMMIT).
- Grado 2 (lecturas comprometidas) → No hay pérdida de actualizaciones en el ámbito de la transacción ni la lectura de datos sucios.
- Grado 3 (lectura repetible) → No hay pérdida de actualizaciones ni lectura de datos sucios y hay lecturas repetitivas. Es el aislamiento completo garantizado.

Cuanto menor grado de aislamiento, mayor la facilidad en la concurrencia, pero se pueden producir violaciones de aislamiento.

Aspecto	Grado 0	Grado 1	Grado 2	Grado 3
Nombre	Caos	*Read Uncommited* Consulta *(Browse)*	*Read Commited*	*Repeatable Read*
Dependencias consideradas	Ninguna	WRITE → WRITE	1º y WRITE → READ	2º y READ → WRITE
Protección proporcionada	Permite los niveles superiores	0º y no se pierden actualizaciones	1º y no hay lecturas *sucias*	2º y hay lecturas repetibles
Datos validados *(committed)*	Escrituras visibles inmendiatamente	Escrituras visibles al fin de la transacción	Igual que 1º	Igual que 1º
Datos sucios	No se sobreescriben datos *sucios* ajenos	Nadie sobreescribe datos *sucios*	1º y no se leen datos *sucios*	2º y no se *ensucian* datos ya leidos
Protocolo de bloqueo	Bloqueos exclusivos cortos en escrituras	Bloqueos exclusivos largos en escrituras	1º y bloqueos cortos compartidos lectura	1º y bloqueos largos compartidos lectura
Estructura de la transacción	Bien formadas (Wrt)	Bien formadas (Wrt) Dos fases (Wrt)	Bien formadas. Dos fases (Wrt)	Bien formadas Dos fases.
Concurrencia	Muy alta. Casi no hay esperas	Alta: Sólo espera bloqueos escritura	Media: 1º y bloqueo en algunas lecturas	Baja: bloqueo se mantiene hasta EOT
Sobrecarga *(overhead)*	Mínima.	Pequeña	Media	Media
Rollback	No existe	Funciona	Igual que 1º	Igual que 1º
Recuperación del sistema	Peligroso. Pérdidas y violaciones 3º	Aplicar log en orden 1º	Igual que 1º	Aplicar log en orden <<<

Si se produce una situación de bloqueo recíproco de recursos que producen espera ilimitada, se ha producido un **DEADLOCK (Interbloqueo)**. La solución más simple es la de **nunca parar**, hacer rollback y volver a intentar de nuevo la transacción. Para detectar el DEADLOCK se puede hacer mediante **timeout** o mediante un **grafo de esperas** en el que se puede ver si que transacción depende de quién.