Gestión de transacciones

Estudiaremos lo referente al procesamiento de transacciones, que es un tema muy importante de cara al aseguramiento de la integridad de los datos. Veremos lo que es una transacción, sus características y estados, analizaremos como se implementa cada una de las propiedades de una transacción, explicaremos como el SGBD gestiona la ejecución concurrente de transacciones de manera que se preserve la integridad de la base de datos, y finalmente analizaremos lo que pasa cuando se presentan fallos durante la ejecución de transacciones y como el SGBD se recupera de ellos.

Al terminar el estudio de esta unidad usted estará en capacidad de identificar y justificar la importancia de las transacciones como instrumento para garantizar la integridad de los datos, y podrá diseñar planificaciones de ejecución concurrente de transacciones que garanticen la integridad y recuperabilidad de los datos.

Iniciaremos dando un vistazo general al procesamiento de transacciones.

1. Panorámica del procesamiento de transacciones

El término "transacción" ya le debe sonar familiar, es muy usado en el ámbito de las bases de datos. De hecho, cuando empezamos a construir una base de datos, durante el estudio preliminar identificamos tanto requerimientos de datos como **transacciones** de datos. Desde ese punto de vista (el del usuario) una transacción es una acción específica que se realiza a través de una aplicación, por ejemplo: registrar una venta, registrar un préstamo, comprar un boleto de avión, reservar una habitación de hotel, etc. Acciones que a nivel de base de datos implicarían varias operaciones de consulta y actualización de datos.

Recuerde que cuando estudiamos el componente de integridad dentro de la seguridad de los datos, ya habíamos identificado a la gestión de transacciones como un elemento clave para asegurar la consistencia de los datos.

Para que comprenda mejor, considere el caso de estudio PEDIDOS que se expone en el Anexo de este documento, y pregúntese ¿qué sucede en la base de datos cuando se registra un pedido? Pues, sucede que al procesar la transacción "registrar pedido" en la base de datos ocurren una serie de actualizaciones:

1. Agregar un nuevo pedido a la tabla PEDIDOS.
2. Agregar los ítems solicitados a la tabla ITEMS
3. Actualizar las existencias de los productos ordenados en la tabla PRODUCTOS (control de stock).
4. Actualizar el monto de ventas para el vendedor que receptó el pedido.
5. Actualizar el monto de ventas para la oficina del vendedor.

Para asegurar el éxito de la transacción, esas 5 operaciones deben ejecutarse TODAS de manera sucesiva. De otra forma se producirían graves errores de inconsistencia en los datos. Pues, ¿qué sucedería si por algún evento no previsto (corte de suministro de energía, o falla en el hardware o error del software, etc.) el proceso se interrumpiera entre la 4ta y 5ta actualización? En ese caso, el pedido quedaría registrado en la base de datos e incluso las existencias de los productos y el monto de ventas del vendedor se actualizarían correctamente, pero no sucedería lo mismo con las ventas de la oficina conservaría el mismo valor que tenía antes del inicio de la transacción, aun cuando el pedido si fue registrado. Esto obviamente provocaría una grave inconsistencia en los datos, ya que no se cumpliría la premisa de que “las ventas de una oficina deben ser iguales a la suma de las ventas de los empleados de esa oficina”.

Ese control -el de asegurar que al procesar una transacción no se produzcan inconsistencias en los datos- es responsabilidad del SGBD a través del componente *gestor de transacciones*.

2. Soporte de transacciones

Céntrese ahora en comprender lo que es una transacción, sus propiedades y los estados que la definen, desde una perspectiva más orientada al SGBD.

Una transacción es una secuencia de una o más sentencias que juntas forman una unidad de trabajo o de ejecución. Y aquí es importante remarcar el término **unidad**, pues una transacción constituye una tarea que debe ejecutarse en su totalidad o en su defecto no ejecutarse, pero jamás ejecutarse de manera parcial.

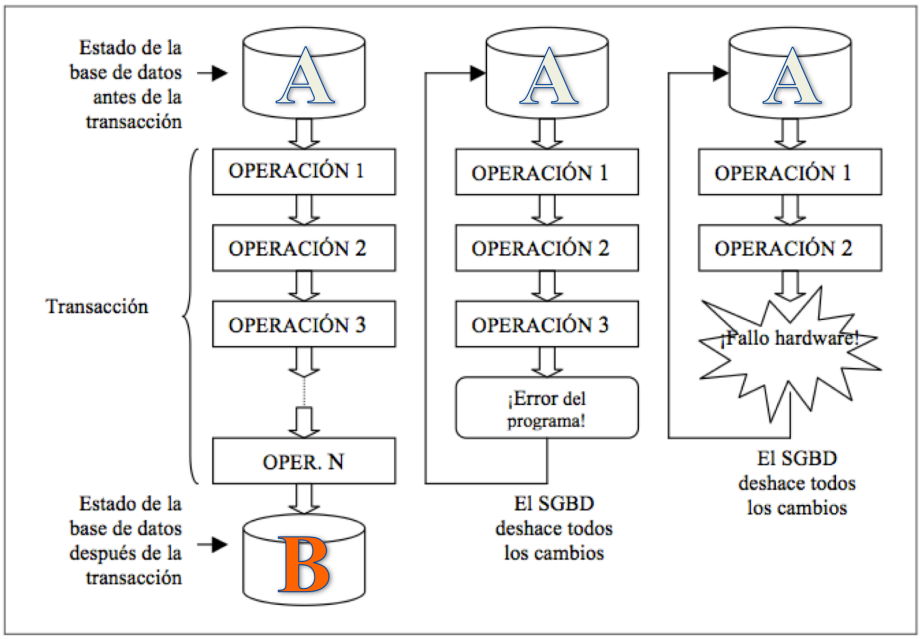


Figura 1. Ilustración del concepto de transacción

Cada operación de una transacción realiza una parte de la tarea, y todas son necesarias para completarla exitosamente. Por ello, el agrupar un conjunto de operaciones dentro de una transacción le indica al SGBD que se trata de una tarea que debe ejecutarse atómicamente para que la base de datos permanezca en un estado consistente (o todas se efectúan o ninguna). El SGBD es responsable de mantener este compromiso incluso si el programa de aplicación aborta o si se produce un fallo en el hardware a mitad de la transacción. Analice la Figura 1 donde se ilustra lo antes expuesto.

Tenga en cuenta que cuando hablamos de operaciones de una transacción nos referimos a operaciones del lenguaje del SGBD (SQL en bases de datos relacionales), y específicamente operaciones de consulta y actualización. A estas operaciones las llamaremos simplemente **leer** y **escribir** (read y write) a efectos de simplificar las explicaciones sobre el procesamiento de transacciones. Leer para hacer referencia a la consulta de un elemento de dato, y escribir para referirnos a su actualización.

Una vez claro el concepto, hablemos de las propiedades. No olvide que toda transacción debe cumplir 4 propiedades, llamadas propiedades ACID:

- **A**tomicity (Atomicidad): Una transacción es una unidad de trabajo indivisible, que exige se ejecuten o todas sus operaciones o ninguna, pero jamás solo una parte de ellas.
- **C**onsistency (Consistencia): Una transacción es una unidad de integridad, por lo que su ejecución debe preservar la consistencia de los datos.
- **I**solation (Aislamiento): Una transacción es una unidad de aislamiento. Implica que la ejecución de una transacción no debe afectar la ejecución de otras.

- **D**urability (Durabilidad): Una transacción es una unidad de recuperación. Si la transacción se finalizó con éxito, los cambios en la base de datos deben ser permanentes.

El SGBD siempre debe garantizar el cumplimiento de las propiedades ACID

Al hablar de atomicidad, en cambio aparecen los posibles estados de una transacción. La Figura 2 complementa la explicación de dichos estados y el flujo que entre ellos puede ocurrir; como se ve, una transacción termina cuando llega a un estado confirmada[1] o abortada.

Cuando ocurre un fallo y la transacción pasa a estado *fallida*, el SGBD debe deshacer (hacer rollback) de todos los cambios efectuados hasta ese punto. ¿Cómo lo hace? ... normalmente guarda un registro (un log) de los cambios hechos el cuál le sirve para regresar atrás. Solo pasará a estado *abortada* cuando se haya retrocedido en todos los cambios, con ello se habrá garantizado la **atomicidad** de la transacción, "o todo o nada". Y si una transacción ha terminado abortándose, entonces se puede optar por reiniciarla o cancelarla. Se optaría por lo primero siempre que el problema no haya correspondido a errores de la aplicación o por incompatibilidad de datos.

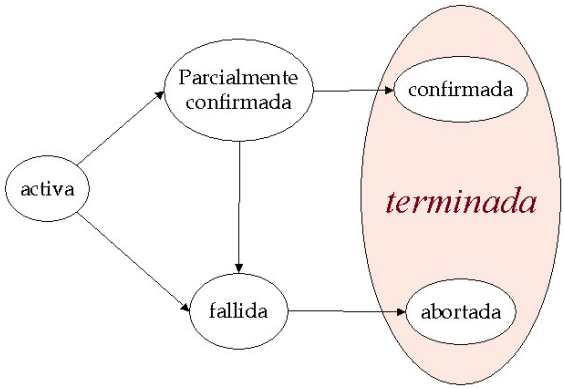


Figura 2. Estados de una transacción

La otra propiedad muy importante de las transacciones es el **aislamiento**, la cual tiene incidencia cuando el SGBD recibe varías transacciones a la vez y las organiza para que se ejecuten concurrentemente[2]. Pues en ese caso surge la pregunta ¿cómo evitar que la ejecución de una transacción afecte a otra?, ¿cómo garantizar el aislamiento? ... lo veremos al discutir el control de concurrencia en bases de datos.

3. Control de concurrencia

La ejecución concurrente de transacciones es un tema crítico que merece mucha atención. En sistemas multiusuario, varios clientes acceden simultáneamente a la base de datos y por lo tanto muchas transacciones pueden requerir ser ejecutadas por el SGBD de forma simultánea.

Lo más sencillo sería que el SGBD no admita la concurrencia, y en el evento de que varias transacciones lleguen al mismo tiempo, las ponga en cola, y ejecute solo una transacción a la vez. Esto sin duda garantizaría la consistencia de los datos, pero no sería eficiente, porque si la ejecución es secuencial, la dispersión en los tiempos de respuesta a los usuarios podría ser muy alta.

[1] En bases de datos los términos "confirmada" y "comprometida" son similares.
[2] No confundir concurrencia con paralelismo. El paralelismo implica ejecutar literalmente varios procesos al mismo tiempo, requiere capacidades de multiprocesamiento. La concurrencia en cambio implica realizar simultáneamente dos o más tareas, pero alternando en el tiempo la ejecución de operaciones de una y otra.

Lo más adecuado en este caso, entonces, es que varias transacciones se ejecuten concurrentemente (alternando operaciones de distintas transacciones).

El problema de la concurrencia radica en que si no se la administra adecuadamente puede provocar inconsistencias en la base de datos. La Figura 3 demuestra el riesgo que se corre cuando el SGBD ejecuta concurrentemente dos transacciones que acceden y modifican la tabla PRODUCTOS del caso de estudio PEDIDOS (Anexo de este documento). Para este caso en un inicio ambas transacciones asumen que hay en existencias 200 unidades del producto 3-8; la transacción A recepta un pedido por 100 unidades de ese producto, mientras que la transacción B acepta en cambio uno por 150 unidades, pues en las dos transacciones se supone que hay suficiente stock, aun cuando sabemos que esto no es cierto ya que ambos pedidos suman 250 unidades. Esto en primera instancia ya es un problema, en vista que se está violando una regla del negocio que dice que "no se puede aceptar pedidos más allá del stock existente". Pero el problema mayor radica en el estado de inconsistencia en que queda la base de datos al final de la ejecución de ambas transacciones. Si se aceptan pedidos por 250 unidades y en stock habían 200 el saldo al final debería ser –50 y no 50 como se ha dado en este caso. Aun cuando se haya violado la regla del negocio ya mencionada, no se puede aceptar tal estado de inconsistencia en la base de datos.

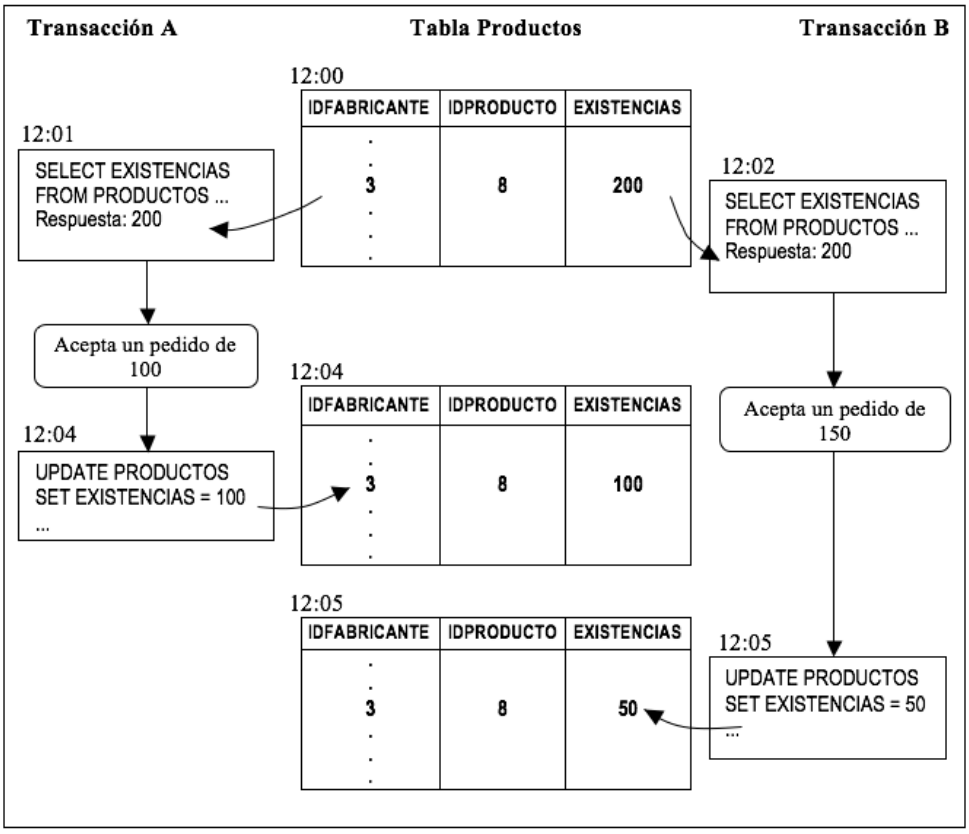


Figura 3. El riesgo en la ejecución concurrente de transacciones

Sin embargo, como ya dijimos las ejecuciones concurrentes son necesarias y es tarea del SGBD controlar dicha concurrencia para evitar que viole el aislamiento y se destruya la consistencia de los datos. Para ello debe evaluar y elegir un esquema de ejecución que asegure el aislamiento.

La secuencia en la que se programa la ejecución de las operaciones de distintas transacciones se denomina *planificación*. Por ejemplo: para el caso de la Figura 3, la planificación sería la que se muestra en la Figura 4.

Orden	Transacción A	Transacción B
1	leer(EXISTENCIAS)	
2		leer(EXISTENCIAS)
3	EXISTENCIAS = EXISTENCIAS - 100	
4		EXISTENCIAS = EXISTENCIAS - 150
5	escribir(EXISTENCIAS)	
6		escribir(EXISTENCIAS)

Figura 4. Ejemplo planificación

Un tipo de planificación es la llamada *planificación* ***secuencial*** que es similar a ejecutar una transacción a la vez (sin concurrencia). Para el caso que estamos analizando una posibilidad de planificación secuencial es <Transaccion A -> Transaccion B>, Figura 5.

Orden	Transacción A	Transacción B
1	leer(EXISTENCIAS)	
2	EXISTENCIAS = EXISTENCIAS - 100	
3	escribir(EXISTENCIAS)	
4		leer(EXISTENCIAS)
5		EXISTENCIAS = EXISTENCIAS - 150
6		escribir(EXISTENCIAS)

Figura 5. Ejemplo planificación secuencial

La planificación secuencial garantiza la consistencia de los datos, pero no constituye una ejecución concurrente. Lo que se necesita es una planificación que permita la ejecución concurrente y que obtenga los mismos resultados que una planificación secuencial.

Aquí es importante aclarar, que, para efectos de evaluar una planificación concurrente, las operaciones que nos van a interesar realmente son ***leer*** (consulta) y ***escribir*** (agregación, modificación o eliminación). Esto porque durante el acceso a la base de datos (para consultar o actualizar) es cuando pueden presentarse conflictos entre transacciones que estén operando sobre un mismo elemento. En SQL estándar la operación *leer* está asociada a la sentencia de consulta SELECT, y la operación de *escribir* está asociada en cambio a las operaciones INSERT, DELETE y UPDATE.

Secuencialidad

La *secuencialidad* es la propiedad que una planificación concurrente debe cumplir con el fin de asegurar la consistencia de los datos, debiendo para ello ser equivalente[3] a una planificación secuencial. Lo que se conoce como ***planificación secuenciable***[4].

Para elaborar una planificación concurrente secuenciable se debe partir de una planificación secuencial, a partir de la cual se va intercalando el orden de ejecución de las instrucciones de una y otra transacción conservando siempre la propiedad de secuencialidad de la planificación resultante.

Pero, ¿cómo saber si se conserva la secuencialidad? La respuesta está en el estudio de las formas de secuencialidad existente, las que nos dan normas que debemos considerar al momento de intercalar las operaciones.

Son dos las formas de secuencialidad o serializabiliadad existentes:

1. Secuencialidad en cuanto a conflictos
2. Secuencialidad en cuanto a vistas

Secuencialidad en cuanto a conflictos

[3] Dos planificaciones se consideran como equivalentes si ambas derivan en el mismo resultado

[4] Algunos autores usan los términos "serie" en lugar de " secuencial", y " serializable" en lugar de "secuenciable".

En la secuencialidad en términos de conflictos dos instrucciones consecutivas de diferentes transacciones se pueden intercalar si no existe conflicto entre ellas. Y existe conflicto solo cuando ambas operan sobre el mismo elemento de dato y al menos una de ellas lo escribe. Para complementar lo explicado en el texto se veamos dos ejemplos más.

EJEMPLO 1: Consideremos dos transacciones A y B dentro de nuestro caso de estudio "PEDIDOS"; las cuales actualizan los montos de ventas (en OFICINAS y en EMPLEADOS), una vez receptados dos pedidos del mismo vendedor por un importe de 2000 y 5000 dólares respectivamente. Se asume que son transacciones que se ejecutan de forma simultánea. La Figura 6 muestra una planificación secuencial para la ejecución concurrente de estas dos transacciones asumiendo que la transacción A se ejecuta primero.

Orden	Transacción A	Transacción B
1	leer(OFICINAS.VENTAS)	
2	Incrementar OFICINAS.VENTAS en $ 2,000	
3	escribir(OFICINAS.VENTAS)	
4	leer(EMPLEADOS.VENTAS)	
5	Incrementar EMPLEADOS.VENTAS en $ 2,000	
6	escribir(EMPLEADOS.VENTAS)	
7		leer(OFICINAS.VENTAS)
8		Incrementar OFICINAS.VENTAS en $ 5,000
9		escribir(OFICINAS.VENTAS)
10		leer(EMPLEADOS.VENTAS)
11		Incrementar EMPLEADOS.VENTAS en $ 5,000
12		escribir(EMPLEADOS.VENTAS)

Figura 6. Planificación secuencial para ejemplo 1

Como ya se aclaró lo que interesa para efectos de planificación son solo las operaciones de leer y escribir, el resto de operaciones se deben ejecutar indistintamente dentro de la ejecución concurrente, pero eso sí conservando la misma posición dentro de la transacción a la que pertenecen. La planificación P1 de la Figura 7 muestra la misma planificación, pero excluyendo las operaciones de cálculo.

P1

Orden	Transacción A	Transacción B
1	leer(oficinas.ventas)	
2	escribir(oficinas.ventas)	
3	leer(empleados.ventas)	
4	escribir(empleados.ventas)	
5		leer(oficinas.ventas)
6		escribir(oficinas.ventas)
7		leer(empleados.ventas)
8		escribir(empleados.ventas)

P2

Orden	Transacción A	Transacción B
1	leer(oficinas.ventas)	
2	escribir(oficinas.ventas)	
3	leer(empleados.ventas)	
4		leer(oficinas.ventas)
5	escribir(empleados.ventas)	
6		escribir(oficinas.ventas)
7		leer(empleados.ventas)
8		escribir(empleados.ventas)

P3

Orden	Transacción A	Transacción B
1	leer(oficinas.ventas)	
2	escribir(oficinas.ventas)	
3		leer(oficinas.ventas)
4	leer(empleados.ventas)	
5		escribir(oficinas.ventas)
6	escribir(empleados.ventas)	
7		leer(empleados.ventas)
8		escribir(empleados.ventas)

P4

Orden	Transacción A	Transacción B
1	leer(oficinas.ventas)	
2	escribir(oficinas.ventas)	
3		leer(oficinas.ventas)
4		escribir(oficinas.ventas)
5	leer(empleados.ventas)	
6	escribir(empleados.ventas)	
7		leer(empleados.ventas)
8		escribir(empleados.ventas)

Figura 7. Planificaciones para ejemplo 1

Ahora analicemos las instrucciones de esta planificación. Para lograr una planificación concurrente necesitamos intercalar instrucciones consecutivas de la transacción A y B. En este caso las instrucciones 4 y 5 son consecutivas y están en distintas transacciones. En vista que acceden a distinto elemento de dato aun cuando una de ellas es una operación escribir no presentan conflicto, por lo tanto, podemos intercambiar su orden de ejecución y la planificación quedaría como lo refleja la planificación P2 de la Figura 7. Igual intercambio podemos hacer entre las instrucciones 3 y 4 y entre las instrucciones 5 y 6 de P2; y posteriormente entre las instrucciones 4 y 5 de P3. La planificación concurrente al final quedaría como la mostrada en P4. Y no se puede realizar ningún otro intercambio puesto que para todas las posibilidades existe conflicto. Por ejemplo: las instrucciones 2 y 3 acceden al mismo elemento de datos (OFICINAS.VENTAS) y la primera es una operación de escritura por lo tanto existe conflicto entre ellas. La planificación resultante (P4) se dice que es secuenciable en cuanto a conflictos.

Nótese que solo se cambia es el orden de ejecución de las operaciones, pero cada operación se mantiene en la transacción a la que pertenece conservando su posición.

EJEMPLO 2: Veamos ahora la comprobación de la secuencialidad en términos de conflictos. Considere las siguientes transacciones:

T1	T2	T3
escribir(X)	leer(X)	leer(X)
leer(Y)	escribir(X)	escribir(Y)
escribir(Y)	escribir(Y)	

La Figura 8 muestra 3 posibles ejecuciones concurrentes de esas 3 transacciones. Y queremos saber si esas planificaciones son secuenciables en cuanto a conflictos, para ello por cada planificación se elabora un grafo de precedencia de acuerdo al procedimiento explicado en el texto básico (Figura 9), si el grafo contiene algún ciclo no es secuenciable. En este ejemplo la planificación (a) no es secuenciable, (b) y (c) si lo son; (b) es secuenciable en términos de conflictos respecto a T1 -> T2 -> T3, mientras que (c) es secuenciable respecto a T2 -> T1 -> T3.

T1	T2	T3
escribir(X)		
	leer(X)	
		leer(X)
	escribir(X)	
leer(Y)		
escribir(Y)		
	escribir(Y)	
		escribir(Y)

(a)

T1	T2	T3
escribir(X)		
	leer(X)	
	escribir(X)	
		leer(X)
leer(Y)		
escribir(Y)		
	escribir(Y)	
		escribir(Y)

(b)

T1	T2	T3
	leer(X)	
	escribir(X)	
escribir(X)		
		leer(X)
	escribir(Y)	
leer(Y)		
escribir(Y)		
		escribir(Y)

(c)

Figura 8. Planificaciones concurrentes ejemplo 2

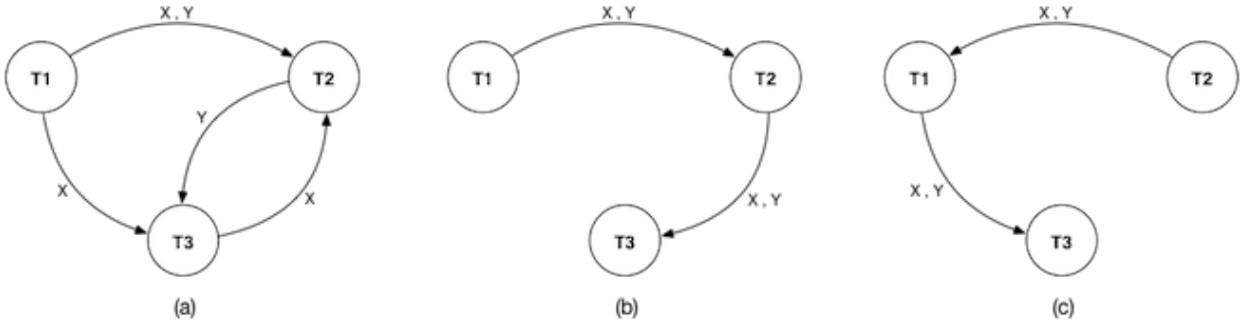


Figura 9. Grafos de precedencia planificaciones ejemplo 2

Actividad propuesta:

Dadas las siguientes transacciones:

T1	T2	T3	T4
escribir(A)	escribir(C)	leer(B)	escribir(B)
escribir(D)	leer(A)	leer(C)	escribir(D)
leer(F)	leer(D)	escribir(D)	
	escribir(F)	escribir(C)	
	escribir(D)	escribir(F)	

1. Partiendo de la planificación secuencial T1 -> T2 -> T3 -> T4. Construya una planificación secuenciable en términos de conflictos.
2. Usando grafos de precedencia valide la planificación resultante del punto anterior para verificar que en efecto es secuenciable en cuanto a conflictos.

Secuencialidad en cuanto a vistas

Otra forma de obtener planificaciones secuenciables es a través de la secuencialidad en cuanto a vistas. Las condiciones para que una planificación concurrente sea equivalente la planificación secuencial en cuanto a vistas son bastante sencillas y se resumen aquí.

1. En ambas planificaciones (en la concurrente y en la secuencial) debe cumplirse que el valor original de un mismo elemento de dato debe ser accedido o leído por la misma transacción (en caso de que exista una operación de lectura).
2. Si en la planificación secuencial existen dos instrucciones de diferentes transacciones donde la una escribe un elemento de datos y la otra lee el valor del mismo elemento que fue escrito por la anterior, el orden de estas dos instrucciones debe mantenerse en la planificación concurrente.
3. En ambas planificaciones la transacción que escribe el valor final de un elemento de datos debe ser la misma.

Siempre que estas condiciones se cumplan podemos intercalar instrucciones y obtendremos una planificación secuenciable en cuanto a vistas.

En este caso no es necesario que las operaciones objeto de intercambio, sean operaciones consecutivas. Es posible intercalar operaciones no consecutivas siempre que se cumplan las condiciones antes mencionadas, y obviamente que además cada operación conserve la misma posición dentro de la transacción a la que pertenece.

Veamos un ejemplo. En la Figura 10 se muestran dos planificaciones equivalentes en cuanto a vistas siendo la segunda una planificación secuenciable en cuanto a vistas ya que se deriva de una planificación secuencial.

Trans A	Trans B	Trans C
Escribir(B)		
Leer(A)		
Escribir(A)		
	Leer(B)	
	Escribir(B)	
	Escribir(A)	
		Leer(B)
		Escribir(A)

Planificación secuencial

Trans A	Trans B	Trans C
Escribir(B)		
	Leer(B)	
	Escribir(B)	
		Leer(B)
Leer(A)		
	Escribir(A)	
Escribir(A)		
		Escribir(A)

Planificación secuenciable en cuanto a vistas

Figura 10. Planificaciones equivalentes en términos de vistas

Para nuestro ejemplo desde el punto de vista de la secuencialidad en cuanto a conflictos la operación escribir(A) de la transacción B esta en conflicto con la operación escribir(A) de la transacción A por lo que no sería posible realizar un cambio entre ellas; pero desde el punto de vista de la secuencialidad en cuanto a vistas esto es posible ya que la escritura final del elemento de datos A lo realiza la transacción C siendo esta una escritura a ciegas, es decir que no importa el valor de A generado por las otras transacciones.

Se cumple entonces para el ejemplo todas las condiciones de la secuencialidad en cuanto a vistas:

1. En ambas planificaciones es la transacción A la que lee el valor inicial de elemento de datos A.
2. En ambas planificaciones la transacción B lee el valor del elemento B producido por la transacción A, al igual que la transacción C lee el elemento B producido por la transacción B.
3. En ambas planificaciones es la transacción C la que escribe el valor final del elemento de datos A y es la transacción B la que escribe el valor final del elemento de datos B.

Actividad propuesta:

Complemente la práctica de la secuencialidad en términos de vista, consultado otros recursos y ejemplos. Luego, analice la siguiente planificación y determine si es secuenciable en términos de vistas respecto a la planificación secuencial T1 -> T2 -> T3 -> T4

T1	T2	T3	T4
	escribir(T)		
		leer(T)	
		escribir(T)	
			leer(T)
leer(P)			
escribir(P)			
	leer(P)		
	escribir(Q)		
		leer(Q)	
escribir(Q)			
			escribir(Q)
		leer(U)	
		escribir(P)	
	leer(U)		
			escribir(U)
leer(S)			
escribir(S)			

Al momento de buscar como intercalar instrucciones en una planificación secuencial para obtener otra equivalente y concurrente se puede aplicar conceptos de todas las formas de secuencialidad, lo importante es que al final la planificación resultante sea secuenciable. Toda planificación secuenciable en términos de conflictos lo es también en términos de vistas, pero no al contrario.

Tanto las planificaciones concurrentes que son secuenciables en cuanto a conflictos como las que son secuenciables en cuanto a vistas, garantizan la consistencia en los datos siempre y cuando la ejecución concurrente concluya con éxito y no exista ningún fallo. Pero sabemos que esto no siempre es así, por lo tanto, la planificación concurrente debe también contar con otra propiedad que es la **recuperabilidad** en caso de fallos.

Uso de bloqueos para implementación del aislamiento

La secuencialidad, es una de las herramientas que los SGBD usan para asegurar el aislamiento de las transacciones, tal que en una ejecución concurrente se organicen las operaciones de manera que el resultado individual de cada transacción no se vea afectado al alternar la ejecución con otras transacciones. Complementariamente los SGBD usan una figura denominada *bloqueo* a través del cual

cuando una transacción accede a un elemento de datos, este adquirirá un estatus especial de reserva denominado bloqueo, por el cual solo una transacción puede operar un elemento al mismo tiempo.

4. Recuperación de la base de datos

La **recuperabilidad** es la propiedad que debe cumplir cualquier ejecución concurrente de manera que asegure la atomicidad de las transacciones en caso de fallo.

La Figura 11 muestra un caso que permite evidenciar el problema de planificaciones no recuperables. Analice la planificación (a), en este caso T2 lee el valor de A que escribió T1, por lo tanto, T2 depende de T1, y por razón similar T3 depende de T2. Si la transacción T1 falla justo antes de la última operación, por la propiedad de atomicidad T1 debe deshacerse totalmente, al hacerlo quedaría ya sin efecto la escritura de A que realizó T1, por lo que la lectura de A que hizo T2 ya no sería consistente, por lo que también se debe abortar T2 (no hay problema porque T2 aún no ha finalizado). El problema es T3, ya que al abortar T2 la lectura de A que hizo T3 tampoco es consistente, se debería abortar T3 también, pero no es posible ya que, en el momento del fallo ya había finalizado. Esto sin duda provoca una grave inconsistencia.

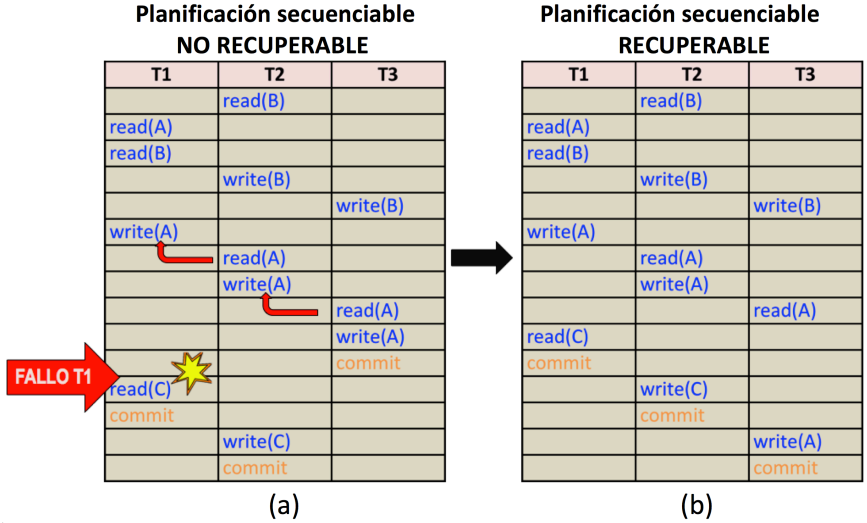


Figura 11. Ejemplo planificación no recuperable y recuperable

Toda planificación debe ser recuperable, y para que ello se cumpla el compromiso de las transacciones dependientes debe ocurrir después del comprometimiento de las transacciones de las que dependen. Tal como se muestra en la planificación (b) de la Figura 11.

Para asegurar la **consistencia** y **atomicidad** en una ejecución concurrente no solo es necesaria una planificación secuenciable sino también recuperable.

Actividad propuesta:

Considere las transacciones A, B y C que se muestran a continuación:

Transacción A	Transacción B	Transacción C
leer(A)	leer(C)	leer(B)
C = A	A = B	A = B
escribir(C)	escribir(A)	escribir(A)
leer(B)	B = B + C	leer(C)
A = A + B	escribir(B)	C = A - B
escribir(A)		escribir(C)

	Suponga que estas tres transacciones se ejecutan concurrentemente y necesitamos establecer una planificación para su ejecución. Realice entonces lo siguiente: a) Elija a su criterio una planificación secuencial para la ejecución de estas tres transacciones. b) Tomando como punto de partida la planificación secuencial del punto anterior, aplique los conceptos de secuencialidad y recuperabilidad y construya una planificación concurrente que garantice la secuencialidad y recuperabilidad. Nota. - No interesa un tipo de secuencialidad específico puede ser cualquiera de los estudiados e incluso ambos. No olvide hacer constar cual fue la planificación secuencial que le sirvió como punto de partida.

5. Definición de transacciones en SQL

En SQL estándar se especifica que una transacción SQL comienza automáticamente con la primera sentencia SQL ejecutada por un usuario o un programa. La transacción continua con las sentencias subsiguientes hasta que finaliza a consecuencia de cualquiera de los siguientes escenarios:

- Una sentencia COMMIT finaliza la transacción con éxito, haciendo que los cambios a la base de datos sean permanentes. Una nueva transacción comienza inmediatamente después de la sentencia COMMIT.
- Una sentencia ROLLBACK aborta la transacción deshaciendo las modificaciones que haya efectuado a la base de datos. Una nueva transacción comienza inmediatamente después de la sentencia ROLLBACK.
- La terminación de un programa con éxito (para SQL programado) también finaliza la transacción correctamente, igual que si hubiera ejecutado una sentencia COMMIT.
- La terminación anormal del programa (para SQL programado), también aborta la transacción, del mismo modo que si hubiera ejecutado una sentencia ROLLBACK.

Algunos SGBD adaptan el modelo de transacciones de SQL estándar para proporcionar capacidades adicionales. La Figura 12 nos muestra un ejemplo de transacciones de los SGBD Sybase y Oracle.

En el caso de Sybase se incluyen cuatro sentencias de procesamiento de transacciones:

- La sentencia BEGIN TRANSACTION señala el comienzo explícito de la transacción.
- La sentencia COMMIT TRANSACTION señala el final con éxito de una transacción
- La sentencia SAVE TRANSACTION establece un punto de guarda a mitad de una transacción. Sybase guarda el estado de la base de datos en un punto actual de la transacción y le asigna al estado guardado un nombre de punto de guarda, especificado en la sentencia.
- La sentencia ROLLBACK TRANSACTION tiene dos papeles. Si se designa un punto de guarda en la sentencia ROLLBACK Sybase deshace los cambios de la base de datos efectuados desde el punto de guarda. Si no hay ningún punto de guarda designado la sentencia ROLLBACK deshace los cambios desde la sentencia BEGIN TRANSACTION.

El mecanismo de punto de guarda de Sybase es usado por muchos SGBD y es especialmente útil en transacciones complejas que contienen muchas sentencias. Oracle de hecho maneja conceptos parecidos, la principal diferencia es que Oracle se adapta más al estándar establecido (la transacción inicia automáticamente con la primera sentencia SQL).

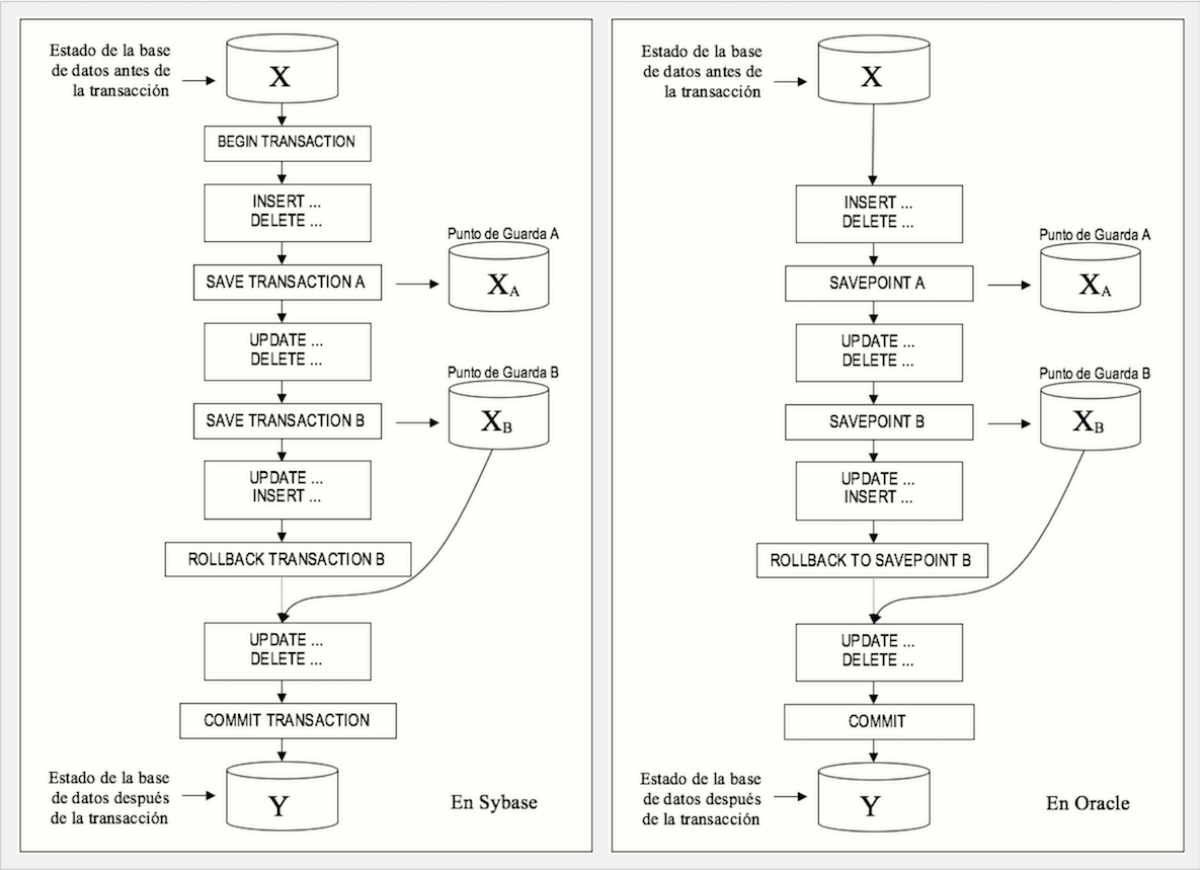


Figura 12. El modelo de transacción de Sybase vs Oracle

.-

ANEXO

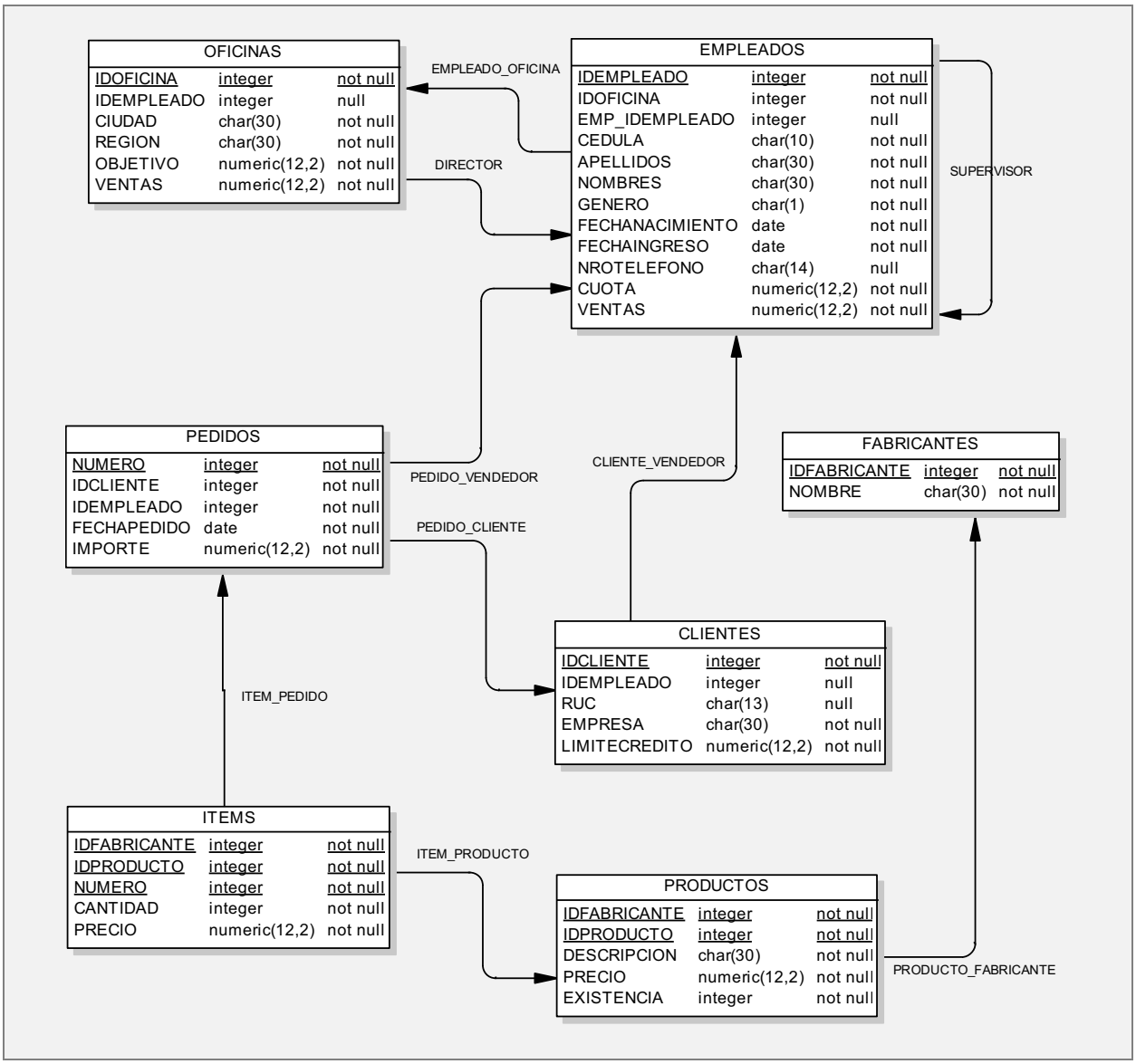
CASO DE ESTUDIO PEDIDOS

Descripción del problema

Imaginémonos en una empresa de distribución que desea informatizar el procesamiento de pedidos. Las consideraciones básicas a tomar en cuenta en un principio son las siguientes:

- La empresa tiene varias oficinas o localidades donde recepta los pedidos.
- Cada oficina cuenta con un equipo de vendedores y empleados quienes se encargan de procesar los pedidos de cada cliente.
- A su vez cada vendedor tiene un supervisor, el cual a su vez tiene bajo su control a varios vendedores.
- Un cliente puede solicitar uno o más productos en un solo pedido.
- Cada producto se codifica basándose en el fabricante y a un número interno.

Esquema relacional



Restricciones de integridad para campos derivados

- La suma del importe de los pedidos receptados por un vendedor debe corresponder a valor de ventas para ese vendedor (empleados.ventas).
- La suma de las ventas de los vendedores de una misma oficina debe corresponder al total de ventas de esa oficina (oficinas.ventas).
- El importe total de cada pedido (pedidos.importe) es igual a la suma del importe de cada ítem correspondiente a ese pedido.