Gestión de transacciones

M. Elena Rodríguez González

PID_00171671

Universitat Oberta de Catalunya

Índice

Introducción

Uno de los objetivos más importantes de los sistemas de gestión de bases de datos (SGBD) es garantizar la integridad de los datos almacenados en las bases de datos (BD) que gestionan. La integridad tiene que ver con la consistencia y la calidad de los datos. Hay diversas causas que pueden comprometer esa integridad: el acceso simultáneo de usuarios diferentes a una misma BD, una situación de avería, el hecho de que se haya decidido tener datos replicados para mejorar el rendimiento en el acceso a la BD o que una operación pueda comprometer una regla de integridad definida sobre la BD.

En este módulo estamos interesados en las posibles anomalías que se deriven del acceso simultáneo de diversos usuarios a la misma BD y en el hecho de asegurar la disponibilidad de la BD ante fallos o desastres, como sería el caso de una avería en los dispositivos de almacenamiento externo, un apagón o un incendio. Hay que tener presente que los datos de una organización casi siempre son uno de sus activos principales, una herramienta indispensable para el desarrollo normal de las actividades que lleva a cabo.

El SGBD tiene que afrontar todas estas posibles anomalías y, para hacerlo, se fundamenta en el concepto de transacción y en una serie de mecanismos para gestionar esas transacciones.

Objetivos

En el material didáctico de este módulo, el estudiante encontrará las herramientas básicas para alcanzar los siguientes objetivos:

1. Comprender los problemas que se derivan del acceso concurrente de diversos usuarios a una misma BD.

2. Saber qué es una transacción, qué propiedades tiene que cumplir y cómo se utiliza.

3. Comprender las funciones que tiene que cumplir un SGBD en la gestión de transacciones, tanto con respecto al control del acceso concurrente por parte de los usuarios como en el caso de fallos o desastres que pongan en peligro la disponibilidad de los datos.

4. Conocer el funcionamiento de las reservas, la técnica más sencilla para el control de la concurrencia.

5. Tener conocimientos básicos de cómo puede evitar un SGBD que se pierdan o se estropeen datos mediante copias de seguridad y dietarios.

6. Ser capaz de desarrollar aplicaciones que utilicen de forma correcta y eficiente los servicios de gestión de transacciones que ofrecen los SGBD.

1. Problemática asociada a la gestión de transacciones

En los SGBD, el concepto de transacción representa la unidad de trabajo por lo que se refiere a control de concurrencia y recuperación. La gestión de transacciones que realiza el SGBD protege las aplicaciones de las anomalías importantes que se pueden producir si no se lleva a cabo.

A continuación veremos, con ejemplos, los problemas que pueden surgir cuando se ejecutan de forma concurrente, y sin ningún control por parte del SGBD, diferentes transacciones.

Supongamos que una aplicación de una entidad bancaria ofrece a los usuarios una función que permite transferir cierta cantidad de dinero de una cuenta de origen a una cuenta de destino. Esta función podría ejecutar los pasos que mostramos en la siguiente tabla (en pseudocódigo):

Transferencia de cantidad Q de cuenta_origen a cuenta_destino	
N.° operación	**Operaciones que hay que ejecutar**
1	saldo_origen:= leer_saldo(cuenta_origen) Comprobar que saldo_origen es mayor o igual a Q (suponemos que hay saldo suficiente para hacer la transferencia)
2	saldo_destino:= leer_saldo(cuenta_destino)
3	escribir_saldo(cuenta_origen, saldo_origen – Q)
4	escribir_saldo(cuenta_destino, saldo_destino + Q)
5	registrar_movimiento("transferencia", cuenta_origen, cuenta_destino, Q) Crear un registro para anotar la transferencia en una tabla de movimientos, y poner también la fecha y la hora, por ejemplo

Hay que considerar las anomalías que se producirán si no se toma ninguna precaución:

1) Supongamos que un usuario empieza a ejecutar una de estas transferencias y, justo después del tercer paso, un apagón hace que el proceso no acabe. En este caso, se habrá restado la cantidad transferida al saldo de la cuenta de origen, pero no se habrá sumado al de la cuenta de destino. Esta posibilidad representa un peligro grave.

Desde el punto de vista de la aplicación, las operaciones que se ejecutan cuando se hace la transferencia se tienen que llevar a cabo completamente o no se tienen que efectuar en absoluto; es decir, la transferencia no puede quedar a medias.

2) Supongamos que dos usuarios diferentes (A y B) intentan hacer dos transferencias al mismo tiempo desde cuentas de origen diferentes y hacia la misma cuenta de destino. Analicemos qué puede pasar si, por cualquier motivo y sin ningún control por parte del SGBD, los pasos de las transacciones se ejecutan de forma concurrente en el siguiente orden:

Ejecución concurrente de dos transferencias bancarias			
N.° operación	**Transferencia usuario A (Q = 20)**	**N.° operación**	**Transferencia usuario B (Q = 40)**
1	saldo_origen:= leer_saldo(cuenta_origen1) Comprobar que saldo_origen es mayor o igual a 20 (suponemos que hay saldo suficiente para hacer la transferencia)		
2	saldo_destino:= leer_saldo(cuenta_destino)		
		1	saldo_origen:= leer_saldo(cuenta_origen2) Comprobar que saldo_origen es mayor o igual a 40 (suponemos que hay saldo suficiente para hacer la transferencia)
		2	saldo_destino:= leer_saldo(cuenta_destino)
3	escribir_saldo(cuenta_origen1, saldo_origen – 20)		
4	escribir_saldo(cuenta_destino, saldo_destino + 20)		
5	registrar_movimiento("transferencia", cuenta_origen1, cuenta_destino, 20)		
		3	escribir_saldo(cuenta_origen2, saldo_origen – 40)
		4	escribir_saldo(cuenta_destino, saldo_destino + 40)
		5	registrar_movimiento("transferencia", cuenta_origen2, cuenta_destino, 40)

El resultado final es que la cuenta de destino tiene como saldo la inicial más 40, en vez de más 60. Esto es incorrecto, ya que se ha perdido la cantidad que ha transferido el usuario A.

Hay que impedir de alguna manera que el acceso concurrente de diversos usuarios produzca resultados anómalos.

Cada usuario, individualmente, tiene que tener la percepción de que sólo él trabaja con la BD. En el ejemplo que hemos planteado, la ejecución de la transferencia que efectúa el usuario B ha interferido en la ejecución de la transferencia que lleva a cabo el usuario A. Si las dos transferencias se hubieran ejecutado correctamente aisladas la una de la otra, el saldo total de la cuenta de destino habría sido el saldo inicial más 60.

3) Imaginemos que un error de programación de la función de transferencia hace que el saldo de la cuenta de destino reciba como nuevo valor la cantidad que se ha transferido, en vez de sumarla al saldo anterior. Naturalmente, este

comportamiento será incorrecto, ya que no se corresponde con el deseo de los usuarios, y dejará la BD en un estado inconsistente: los saldos que tendrían que tener las cuentas de acuerdo con los movimientos registrados (en el quinto paso) no coincidirían con los que se han realmente almacenado.

En conclusión, es misión de los diseñadores y los programadores que las transacciones verifiquen los requisitos de los usuarios.

4) Planteémonos qué pasaría si, después de utilizar la aplicación durante unos cuantos días y en un momento de plena actividad, se produce un error fatal del dispositivo de almacenamiento externo en el que se guarda la BD, de manera que ésta deja de estar disponible.

En definitiva, tiene que haber mecanismos para evitar la pérdida tanto de los datos más antiguos como de las actualizaciones más recientes.

2. Definición y propiedades de las transacciones

El acceso a los datos que hay en una BD se hace mediante la ejecución de operaciones en el SGBD correspondiente. Puesto que estamos interesados en SGBD relacionales, estas operaciones, a alto nivel, serán sentencias SQL. Además, con vistas a resolver el tipo de problemas que hemos planteado en el apartado anterior, estas operaciones se agrupan en transacciones.

Una **transacción** es un conjunto de operaciones (de lectura y actualización) sobre la BD que se ejecutan como una unidad indivisible de trabajo. La transacción acaba su ejecución confirmando o cancelando los cambios que se han llevado a cabo sobre la BD.

Un programa empieza a trabajar con una BD conectándose a ella de una manera adecuada y estableciendo una sesión de trabajo que permite efectuar operaciones de lectura y actualización (inserciones, borrados, modificaciones) de la BD. Para hacer una operación tiene que haber una transacción activa (o en ejecución), que siempre es única. La transacción activa se puede iniciar mediante una instrucción especial o automáticamente cuando se hace la primera operación en el SGBD.

Toda transacción debería de cumplir cuatro propiedades, conocidas como **propiedades ACID**:

ACID

ACID es una sigla que se forma a partir de las iniciales de las palabras atomicidad, consistencia, aislamiento y definitividad (*atomicity*, *consistency*, *isolation* y *definitivity*).

1) Atomicidad. El conjunto de operaciones que constituyen la transacción es la unidad atómica, indivisible, de ejecución. Esto quiere decir que, o bien se ejecutan todas las operaciones de la transacción (y, en este caso, la transacción confirma los resultados) o bien no se ejecuta ninguna en absoluto (y, en este caso, la transacción cancela los resultados). En definitiva, el SGBD tiene que garantizar el todo o nada para cada transacción:

a) Para confirmar los resultados producidos por la ejecución de una transacción, disponemos de la sentencia SQL `COMMIT`.

b) En caso contrario, ya sea porque alguna cosa impide que se acabe de ejecutar la transacción (por ejemplo, un corte de luz), ya sea porque la transacción acaba con una petición explícita de cancelación por parte del programa de aplicación, el SGBD tiene que deshacer todos los cambios que la transacción haya hecho sobre la BD hasta ese momento, como si dicha transacción nunca hubiera existido. En los dos casos se dice que la transacción ha abortado (en

inglés, *abort*) la ejecución. Para cancelar de manera explícita los resultados producidos por la ejecución de una transacción, disponemos de la sentencia SQL `ROLLBACK`.

2) Consistencia. La ejecución de una transacción tiene que preservar la consistencia de la BD. En otras palabras, si antes de ejecutarse una transacción la BD se encuentra en un estado consistente (es decir, en un estado en el que se verifican todas las reglas de integridad definidas sobre la BD), al acabar la ejecución de la transacción la BD también tiene que quedar en un estado consistente, si bien, mientras la transacción esté activa, la BD podría caer momentáneamente en un estado inconsistente.

3) Aislamiento. Una transacción no puede ver interferida su ejecución por ninguna otra transacción que se esté ejecutando de forma concurrente con ésta. En definitiva, el SGBD tiene que garantizar el correcto aislamiento de las transacciones.

4) Definitividad. Los resultados producidos por una transacción que confirma (es decir, que ejecuta la operación de `COMMIT`) tienen que ser definitivos en la BD; nunca se pueden perder, independientemente de que se produzcan fallos o desastres, hasta que otra transacción cambie esos resultados y los confirme. Al contrario, los resultados producidos por una transacción que aborta su ejecución se han de descartar de la BD.

Es importante destacar que las propiedades que acabamos de presentar no son independientes entre ellas; por ejemplo, las propiedades de atomicidad y de definitividad están estrechamente interrelacionadas. Además, el hecho de garantizar las propiedades ACID de las transacciones no es solamente una misión del SGBD, sino también de las aplicaciones que lo utilizan y, por consiguiente, de su desarrollador.

3. Interferencias entre transacciones

En este apartado presentaremos los tipos de interferencias que se pueden producir si las transacciones que se ejecutan de manera concurrente no verifican la propiedad de aislamiento.

Antes de entrar en estas interferencias, es importante destacar que, si hay dos transacciones que se ejecutan de forma concurrente, una de éstas sólo puede interferir en la ejecución de la otra si se dan las siguientes circunstancias:

a) Las dos transacciones acceden a una misma porción de la BD.

b) Como mínimo una de las dos transacciones, sobre esta porción común de la BD a la que acceden, efectúa operaciones de actualización.

En otras palabras, cuando las transacciones que se ejecutan de forma concurrente sólo hacen lecturas, no se producirán nunca interferencias. De manera similar, en caso de que las transacciones hagan actualizaciones, si éstas se realizan sobre porciones diferentes, no relacionadas en la BD, tampoco se pueden producir interferencias.

A continuación presentamos, mediante ejemplos, los tipos de interferencias que pueden haber entre dos transacciones T1 y T2 que se procesan de forma concurrente si no están aisladas de una manera adecuada entre ellas:

1) Actualización perdida. Esta interferencia se produce cuando se pierde un cambio efectuado por una transacción sobre un dato a causa de la presencia de otra transacción que también cambia el mismo dato. Esto podría suceder en una situación como la que se muestra a continuación:

Transacción T1 (reintegro de 20)	Transacción T2 (reintegro de 40)
saldo:= leer_saldo(cuenta)	
	saldo:= leer_saldo(cuenta)
escribir_saldo(cuenta, saldo – 20)	
	escribir_saldo(cuenta, saldo – 40)
COMMIT	
	COMMIT

T1 y T2 ejecutan un mismo tipo de transacción; en este caso, un reintegro de una misma cuenta bancaria. Las dos transacciones leen el mismo valor del saldo de la cuenta, lo actualizan de forma independiente (asumimos que hay saldo suficiente en la cuenta para hacer los reintegros) y restan a este saldo la cantidad que se ha sustraído.

Suponiendo que el SGBD ejecuta las operaciones que constituyen cada transacción sin ningún control y en el orden que se propone en el ejemplo, el cambio correspondiente a la sustracción de T1 se pierde. En consecuencia, el saldo disminuye sólo en 40, en vez de en 60. En definitiva, T1 ha visto interferida su ejecución a causa de la presencia de T2. Si el orden de ejecución de las operaciones de cada transacción hubiera sido el siguiente:

Transacción T1 (reintegro de 20)	Transacción T2 (reintegro de 40)
saldo:= leer_saldo(cuenta)	
	saldo:= leer_saldo(cuenta)
	escribir_saldo(cuenta, saldo – 40)
escribir_saldo(cuenta, saldo – 20)	
COMMIT	
	COMMIT

se habría producido igualmente la interferencia. En este caso, se habría perdido el cambio efectuado por T2. En consecuencia, el saldo disminuiría sólo en 20, en vez de en 60. En este caso, T2 habría visto interferida su ejecución a causa de la presencia de T1.

En definitiva, la interferencia ocurre porque se producen dos lecturas consecutivas de un mismo dato (el saldo de una misma cuenta) seguidas de dos cambios consecutivos del mismo dato (de nuevo, el saldo de una misma cuenta). Simplemente, si la secuencia de operaciones hubiera sido, por ejemplo, la que se muestra a continuación:

Transacción T1 (reintegro de 20)	Transacción T2 (reintegro de 40)
saldo:= leer_saldo(cuenta)	
escribir_saldo(cuenta, saldo – 20)	
	saldo:= leer_saldo(cuenta)
	escribir_saldo(cuenta, saldo – 40)
COMMIT	
	COMMIT

la interferencia no se habría producido. En este caso, T2 recuperaría el valor del saldo de cuenta dejado por T1 y, teniendo en cuenta este nuevo valor de saldo para la cuenta, efectuaría su propio reintegro. En consecuencia, el saldo de la cuenta disminuiría en 60.

2) Lectura no confirmada. Esta interferencia se puede producir cuando una transacción recupera un dato pendiente de confirmación que ha sido modificado por otra transacción que se ejecuta de forma concurrente con la transacción que recupera el dato. Eso podría suceder en diversas situaciones, como las que se muestran a continuación:

Transacción T1 (consulta saldo)	Transacción T2 (reintegro de 20)
	saldo:= leer_saldo(cuenta)
	escribir_saldo(cuenta, saldo – 20)
saldo:= leer_saldo(cuenta)	
COMMIT	
	ROLLBACK

Primeramente, la transacción T2 lee el saldo de la cuenta y lo disminuye en la cantidad que se quiere reintegrar. A continuación, la transacción T1 efectúa una consulta de saldo de la misma cuenta sobre la que T2 hace el reintegro. El valor de saldo que obtiene T1 está pendiente de confirmar, es un dato provisional, ya que T2 todavía no ha confirmado sus resultados. Acto seguido, la transacción T1 finaliza su ejecución y confirma los resultados. Finalmente, T2 cancela su ejecución. Esta cancelación causa que los resultados producidos por T2 sean descartados de la BD, de manera que el saldo de la cuenta sea el que había antes de empezar la ejecución de T2. En consecuencia, T1 ha recuperado un valor que oficialmente nunca ha existido y ha visto interferida su ejecución por la transacción T2. Si las transacciones T1 y T2 hubieran estado aisladas correctamente, T1 nunca habría recuperado el valor provisional dejado por T2 y que finalmente ha sido descartado.

En el ejemplo previo, la interferencia de lectura no confirmada se produce a causa de la cancelación de la transacción que modifica los datos. Sin embargo, la interferencia se puede producir igualmente en caso de que la transacción que modifica datos confirme los resultados.

Imaginemos ahora que tenemos dos transacciones, T1 y T2. La transacción T1 hace la consulta de un saldo de una cuenta corriente, mientras que T2 efectúa un par de reintegros de la misma cuenta corriente. Supongamos que el orden de ejecución de las operaciones es el que se muestra a continuación y que el SGBD no efectúa ningún control sobre el orden de ejecución de estas operaciones:

Transacción T1 (consulta saldo)	Transacción T2 (dos reintegros de 50)
	saldo:= leer_saldo(cuenta)
	escribir_saldo(cuenta, saldo – 50)
saldo:= leer_saldo(cuenta)	
COMMIT	
	saldo:= leer_saldo(cuenta)
	escribir_saldo(cuenta, saldo – 50)
	COMMIT

En este caso, y aunque la transacción T2 confirma los resultados, T1 ve interferida su ejecución por T2 y recupera un dato provisional pendiente de confirmación. Este dato corresponde a un saldo provisional para la cuenta corriente que corresponde al saldo que queda después del primer reintegro.

3) Lectura no repetible. Esta interferencia se produce cuando una transacción, por los motivos que sea, necesita leer dos veces un mismo dato y en cada lectura recupera un valor diferente por el hecho de que hay otra transacción que se ejecuta simultáneamente y que efectúa una modificación del dato leído. Esto podría pasar en diversas situaciones, como la que se muestra a continuación:

Transacción T1 (reintegro de 20)	Transacción T2 (consulta saldo)
	saldo:= leer_saldo(cuenta)
saldo:= leer_saldo(cuenta)	
escribir_saldo(cuenta, saldo – 20)	
	saldo:= leer_saldo(cuenta)
COMMIT	
	COMMIT

La transacción T2, que consulta dos veces el saldo de una misma cuenta corriente, recupera en cada lectura un valor diferente por el hecho de que la transacción T1, entre las dos lecturas, efectúa un reintegro e interfiere en la ejecución de la transacción T2. Si las transacciones se hubieran aislado correctamente entre ellas, T2 habría recuperado el mismo valor para el saldo de cuenta corriente en las dos lecturas: o bien habría recuperado el valor que correspondiera al saldo de la cuenta antes de que se efectuara el reintegro de T1, o bien, al saldo que quedara después de que se efectuara el reintegro de T1.

4) Análisis inconsistente (y el caso particular de fantasmas). Los tres tipos de interferencias anteriores se producen con respecto a un único dato de la BD, es decir, ocurren cuando dos transacciones intentan acceder a un mismo ítem

de datos y, como mínimo, una de las dos transacciones modifica este ítem de datos. Pero también puede haber interferencias con respecto a la visión que dos transacciones tienen de un conjunto de datos que están interrelacionados.

Esto puede pasar, por ejemplo, cuando una transacción T1 lee unos datos mientras que otra transacción T2 actualiza una parte de ellos. T1 puede obtener un estado de los datos incorrecto, como sucede con las siguientes transacciones:

Transacción T1 (consulta de saldos)	Transacción T2 (transferencia)
saldo2:= leer_saldo(cuenta2)	
	saldo1:= leer_saldo(cuenta1)
	escribir_saldo(cuenta1, saldo1 – 100)
saldo1:= leer_saldo(cuenta1)	
COMMIT	
	saldo2:= leer_saldo(cuenta2)
	escribir_saldo(cuenta2, saldo2 + 100)
	COMMIT

Los saldos que lee T1 no son correctos. No se corresponden ni con los de antes de la transferencia entre las dos cuentas ni con los de después, sino a un estado intermedio de T2 que no se tendría que haber visto nunca fuera del ámbito de T2. En consecuencia, T1 ha visto interferida su ejecución por T2.

Un caso particular bastante frecuente de esta interferencia son los **fantasmas**. Esta interferencia se puede producir cuando una transacción lee un conjunto de datos relacionado y hay otra transacción que dinámicamente cambia este conjunto de datos de interés añadiendo nuevos datos. Básicamente, la interferencia ocurre cuando se produce la siguiente secuencia de eventos:

- Una transacción T1 lee una serie de datos que cumplen una condición C determinada.

- Una transacción T2 inserta nuevos datos que cumplen la condición C o bien actualiza datos que no satisfacían la condición C y que ahora sí que la satisfacen.

- La transacción T1 vuelve a leer los datos que satisfacen la condición C o bien alguna información que depende de estos datos.

La consecuencia de esto es que T1 ve interferida su ejecución por T2 y encuentra un fantasma, es decir, unos datos que antes no cumplían la condición y que ahora sí que la cumplen. O también podría pasar que T1 no viera el fantasma directamente, sino el efecto que tiene en otros datos, tal como muestran los siguientes ejemplos:

Transacción T1 (lista de cuentas)	Transacción T2 (creación de cuentas)
leer todas las cuentas del banco. Imaginemos que sólo tenemos tres cuentas (cuenta1, cuenta2 y cuenta3) mostrar datos cuenta1 mostrar datos cuenta2 mostrar datos cuenta3	
	crear_cuenta(cuenta4)
	saldo_inicial(cuenta4, 100)
	COMMIT
sumar el saldo de todas las cuentas obtenemos saldo cuenta1 + saldo cuenta2 + saldo cuenta3 + 100 (la cuenta4 con saldo 100 es el fantasma)	
COMMIT	

En este primer ejemplo, la cuenta 4 es un fantasma desde el punto de vista de T1. Y además T1 ve el efecto que tiene en la suma de saldos que se produce y que, desde su punto de vista, da un resultado incoherente. Si T1 hubiera estado aislada correctamente de la transacción T2, una vez ejecutada la primera consulta, nunca tendría que haber encontrado los datos correspondientes a la cuenta 4.

Finalmente, el siguiente ejemplo muestra un fantasma que se produce a consecuencia de una actualización de datos por parte de la transacción T2. Imaginemos que los propietarios de las cuentas 1 y 2 viven en Barcelona; los propietarios de la cuenta 3 residen en Madrid, y los titulares de la cuenta 4, que vivían en Tarragona, notifican que ahora residirán en Barcelona.

Transacción T1 (lista de cuentas clientes de Barcelona)	Transacción T2 (cambio residencia)
leer cuentas clientes Barcelona mostrar datos cuenta1 mostrar datos cuenta2	
	cambiar_residencia(cuenta4, Barcelona)
sumar el saldo de todas las cuentas de clientes de Barcelona obtenemos saldo cuenta1 + saldo cuenta2 + saldo cuenta4 (la cuenta4 es el fantasma)	
COMMIT	
	COMMIT

En el ejemplo anterior, la cuenta 4 es nuevamente un fantasma desde el punto de vista de la transacción T1.

4. Nivel de concurrencia

Un SGBD puede resolver los problemas de interferencias entre transacciones que hemos visto anteriormente de dos maneras:

a) Cancelar automáticamente (en inglés, *abort*) las transacciones problemáticas y deshacer los cambios que han podido producir sobre la BD.

b) Suspender la ejecución de una las transacciones problemáticas temporalmente y retomarla cuando haya desaparecido el peligro de interferencia. En algunos casos, esta situación también puede comportar la cancelación de transacciones.

Las dos soluciones implican un coste en términos de disminución del rendimiento de la BD. Precisamente, en lo relativo a la gestión de transacciones, uno de los objetivos de los SGBD es minimizar estos efectos negativos.

Se denomina **nivel de concurrencia** al grado de aprovechamiento de los recursos de proceso disponibles, según el solapamiento de la ejecución de las transacciones que acceden de forma concurrente a la BD y se confirman.

El objetivo del SGBD es aumentar el trabajo efectivo (es decir, el trabajo realmente útil para los usuarios) efectuado por unidad de tiempo. Sin duda, las transacciones que suspenden su ejecución no hacen trabajo efectivo, y todavía menos lo hacen las transacciones que finalmente cancelan su ejecución.

Uno de los grandes retos de la gestión de transacciones es alcanzar el nivel de concurrencia adecuado. Esto se consigue intentando que no se produzcan cancelaciones o suspensiones de ejecución de las transacciones cuando no es realmente necesario para impedir una interferencia. Desgraciadamente, este objetivo nunca se satisface del todo, ya que implicaría un esfuerzo excesivo y sería perjudicial para el rendimiento global por otros motivos. Los SGBD intentan obtener un compromiso óptimo entre el nivel de concurrencia que permiten y el coste que esto comporta en términos de tareas de control.

5. Fundamentos teóricos: seriabilidad y recuperabilidad

Antes de describir las técnicas que los SGBD pueden implementar para evitar las interferencias que hemos presentado anteriormente, hay que definir de una manera precisa los criterios que, desde un punto de vista teórico, se tienen que cumplir para considerar que las transacciones están correctamente aisladas entre ellas. Estos criterios quedan recogidos en la teoría de la seriabilidad y de la recuperabilidad.

La seriabilidad asume que todas las transacciones confirman los resultados. Por lo tanto, la seriabilidad nos da los criterios que se tienen que cumplir para garantizar que no se producen interferencias entre las transacciones cuando éstas confirman los resultados. En otras palabras, la seriabilidad ignora la posibilidad de que se puedan producir cancelaciones de las transacciones.

Por su parte, la teoría de la recuperabilidad nos da los criterios adicionales que se tienen que cumplir para evitar interferencias en caso de que se puedan producir cancelaciones de las transacciones.

5.1. Seriabilidad

La seriabilidad considera que las transacciones están formadas por dos tipos muy sencillos de **acciones** (u operaciones) sobre datos elementales. Estas acciones son operaciones de lectura (representadas por el símbolo R(G), en el que G designa el dato elemental que se está leyendo) y de escritura (representadas por el símbolo W(G)). Estos datos elementales sobre los que actúan las operaciones de lectura y de escritura se llaman **gránulos**.

Ejecución de operaciones de lectura y de escritura

Las operaciones de `SELECT` desencadenan la ejecución de acciones de lectura (R(G)).

Por su parte, las operaciones de `INSERT`, `DELETE` y `UPDATE` desencadenan la ejecución de acciones de lectura (R(G)) seguidas de acciones de escritura (W(G)).

Un **gránulo** es la unidad de datos controlada individualmente por el SGBD en lo que respecta al control de concurrencia.

Finalización de las transacciones

También consideraremos acciones las operaciones de finalización de las transacciones, es decir, las operaciones de `COMMIT` y `ROLLBACK`.

El tamaño del gránulo puede variar según el SGBD. Los tamaños de gránulo más frecuentes son la página (o bloque) de disco y el registro (o fila) de una tabla. Hay otras posibilidades, como por ejemplo, que el gránulo sea una tabla

entera de la BD. De hecho, en general, un mismo SGBD es capaz de trabajar con diferentes niveles de granularidad (fila, página, tabla, etc.), según las demandas de los usuarios de la BD.

Nota

En este módulo, y mientras no se diga explícitamente lo contrario, trabajaremos con gránulos de un tamaño equivalente a la página.

Potencialmente, cuanto más fina es la granularidad, más alto es el nivel de concurrencia que se puede alcanzar, ya que la probabilidad de que dos transacciones quieran acceder a un mismo gránulo disminuye. La desventaja de esto es que la sobrecarga (en inglés, *overhead*) del SGBD será mayor, puesto que tendrá que ejecutar más acciones y se tendrán que mantener muchos más datos de control para garantizar el correcto aislamiento de las transacciones.

Al contrario, cuanto más basta es la granularidad, más alto es el riesgo de que se puedan producir problemas de interferencias, ya que la probabilidad de que dos transacciones quieran acceder a un mismo gránulo aumenta. La ventaja de esto es que la sobrecarga del SGBD será menor, puesto que tendrá que ejecutar menos acciones y se tendrán que mantener menos datos de control para garantizar el correcto aislamiento de las transacciones.

Partiendo de esta versión simplificada de las transacciones, la ejecución concurrente de un conjunto de transacciones (en las que se preserva el orden de acciones dentro de cada transacción) recibe el nombre de **horario** o **historia**.

Ejemplo de horario

Dadas dos transacciones T1 y T2 que quieren ejecutar las siguientes acciones:

T1	R(A)	R(B)	R(C)	COMMIT
T2	R(A)	W(A)	COMMIT	

un horario posible para T1 y T2 podría ser:

N.° acción	T1	T2
1	R(A)	
2		R(A)
3	R(B)	
4		W(A)
5	R(C)	
6	COMMIT	
7		COMMIT

Un **horario en serie** es un horario en el que no hay solapamiento entre las acciones de las transacciones implicadas. Los horarios en serie nunca tienen interferencias. Dado un conjunto de *n* transacciones, tendremos *n!* horarios en serie posibles.

Ejemplo de horarios en serie

Las transacciones T1 y T2 del ejemplo anterior tendrán asociados dos horarios en serie (n = 2, 2! = 2), que, de una manera simplificada, designamos como T1; T2 y T2; T1. En forma de tabla, cada uno de estos horarios quedaría tal como sigue:

Horario en serie número 1: T1; T2		
N.° acción	**T1**	**T2**
1	R(A)	
2	R(B)	
3	R(C)	
4	COMMIT	
5		R(A)
6		W(A)
7		COMMIT

Horario en serie número 2: T2; T1		
N.° acción	**T1**	**T2**
1		R(A)
2		W(A)
3		COMMIT
4	R(A)	
5	R(B)	
6	R(C)	
7	COMMIT	

Hay que destacar que los horarios en serie previos producen resultados diferentes. En concreto, el valor del gránulo A que recupera T1 al hacer la acción R(A) es diferente en cada horario. En el caso del primer horario, T1 recupera (acción número 1) el valor de A que hay en la BD antes de que T2 haga el cambio (acción número 6). En el caso del segundo horario en serie, T1 recupera (acción número 4) como valor de A el que deja T2 (acción número 2).

En un horario, dos acciones se consideran **conflictivas** (o **no conmutables**) si pertenecen a transacciones diferentes, y el orden en el que se ejecutan estas acciones puede afectar al valor del gránulo que haya leído una de las transacciones o al valor final del gránulo.

Las situaciones de conflicto entre transacciones sólo pueden aparecer cuando las transacciones actúan (como mínimo) sobre un mismo gránulo y al menos una de las dos acciones es una acción de escritura.

Ejemplos de acciones conflictivas y no conflictivas

Dadas las dos transacciones T1 y T2 del ejemplo y el siguiente horario:

N.° acción	T1	T2
1	R(A)	
2		R(A)
3	R(B)	
4		W(A)
5	R(C)	
6	COMMIT	
7		COMMIT

las acciones 1 y 4 son acciones conflictivas: el valor recuperado por T1 para el gránulo A será diferente según si la lectura se ejecuta antes o después que la escritura realizada por T2.

En cambio, las acciones 1 y 2, por ejemplo, no son conflictivas, aunque trabajen sobre el mismo gránulo, ya que son dos acciones de lectura. De una manera similar, las acciones 4 y 5, por ejemplo, tampoco son conflictivas, aunque una de éstas (la acción 4) sea de escritura, puesto que las acciones operan sobre gránulos diferentes.

Un horario se considera **correcto** –es decir, **sin interferencias**– cuando el orden relativo de todos los pares de acciones conflictivas es el mismo que en algún horario en serie.

Los horarios correctos se denominan **horarios seriables** y siempre producen el mismo resultado que algún horario en serie. Este horario en serie que da resultados equivalentes se conoce como **horario en serie equivalente**.

El criterio que acabamos de presentar, denominado **seriabilidad de conflictos**, nos indica las condiciones precisas que tiene que tener un horario para que podamos considerarlo correcto, suponiendo que todas las transacciones confirmen sus resultados.

Seriabilidad de un horario

A veces, un horario seriable puede tener asociados varios horarios en serie equivalentes. Naturalmente, los horarios en serie son también seriables.

La idea última de este criterio es bastante simple. Los horarios en serie (sin solapamientos de las operaciones efectuadas por las transacciones), por definición, no presentan interferencias, puesto que no trabajan con valores provisionales dejados por otras transacciones. Si tenemos un horario H con solapamientos que da los mismos resultados que un horario sin solapamientos HS (es decir, que un horario en serie), podremos afirmar que el horario H es correcto y que, en consecuencia, no tiene interferencias. Incluso podremos afirmar que el horario HS es un horario en serie equivalente (HSE) al horario H.

Ejemplo de horario seriable

Dadas las transacciones T1 y T2 del ejemplo, el siguiente horario H:

N.° acción	T1	T2
1	R(A)	
2		R(A)
3	R(B)	
4		W(A)
5	R(C)	
6	COMMIT	
7		COMMIT

es seriable, es decir, correcto, sin interferencias. Adicionalmente, el horario en serie equivalente (HSE) al horario H propuesto sería el que vendría dado por T1; T2. Este HSE quedaría representado en forma de tabla tal como sigue:

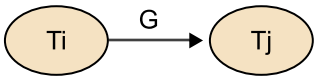
Horario en serie número 1: T1; T2		
N.° acción	**T1**	**T2**
1	R(A)	
2	R(B)	
3	R(C)	
4	COMMIT	
5		R(A)
6		W(A)
7		COMMIT

Para saber si un horario es seriable o no lo es, utilizaremos el grafo de precedencias.

El **grafo de precedencias** es una estructura de datos conceptual (no implementada en los SGBD) con forma de grafo dirigido etiquetado, de manera que:

a) Los nodos representan transacciones.
b) Las etiquetas de los arcos designan gránulos.
c) Los arcos indican las relaciones de precedencia establecidas en el horario, según los pares de acciones conflictivas que éste presenta.

Más concretamente, si el grafo de precedencias tiene un arco etiquetado con un gránulo G con origen Ti y destino Tj como el que se muestra a continuación:



ello significa que, en el horario, las transacciones Ti y Tj han ejecutado un par de acciones conflictivas sobre el gránulo G y que, además, en el tiempo, en primer lugar se ejecuta la acción que realiza la transacción Ti.

Los grafos de precedencias asociados a horarios seriables (es decir, los horarios correctos, sin interferencias) son acíclicos. Y, al contrario, los horarios con interferencias tienen ciclos en el grafo de precedencias.

Suponiendo que el grafo de precedencias asociado a un horario es acíclico (horario seriable), el grafo de precedencias nos ayuda a encontrar los horarios en serie equivalentes. Para encontrar estos horarios, simplemente hay que hacer un recorrido del grafo (basado en ordenación topológica) tal como sigue:

1) Elegir un nodo del grafo al que no llegue ningún arco y apuntar la transacción que se representa.

2) Eliminar el nodo del grafo y todos los arcos que tienen como origen este nodo.

Los pasos 1) y 2) se tienen que repetir hasta que el grafo de precedencias esté vacío.

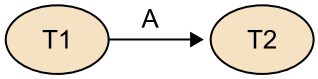
Ejemplos de horarios seriables y grafos de precedencias correspondientes

Dadas las transacciones T1 y T2 del ejemplo y el siguiente horario H:

N.° acción	T1	T2
1	R(A)	

N.° acción	T1	T2
2		R(A)
3	R(B)	
4		W(A)
5	R(C)	
6	COMMIT	
7		COMMIT

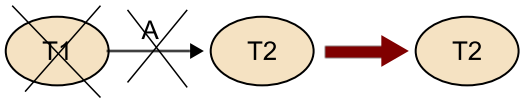
el grafo de precedencias asociado al horario H es el siguiente:



Puesto que la única pareja de acciones conflictivas que ejecutan las transacciones T1 y T2 son las acciones 1 y 4 y, siguiendo el orden, primero se ejecuta la acción efectuada por la transacción T1, eso da lugar a un arco (etiquetado con el gránulo A) en el grafo de precedencias con origen T1 y destino T2.

Como el grafo de precedencias es acíclico, podemos afirmar que el horario H es seriable (sin interferencias) y que, como mínimo, hay un HSE que da los mismos resultados que el horario H.

Si hacemos un recorrido basado en ordenación topológica del grafo de precedencias, el nodo que representa la transacción T1 es el único al que no llega ningún arco. Esta transacción será la primera en el horario en serie equivalente HSE asociado al horario H. Eliminemos del grafo de precedencias el nodo que representa T1 y los arcos que salen de este nodo. Nos quedamos así con un grafo de precedencias que sólo tiene un nodo que representa la transacción T2:



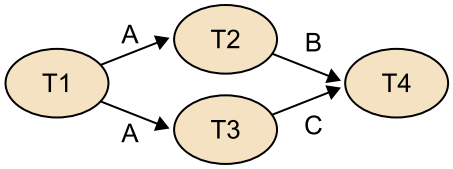
Si continuamos con el procedimiento descrito, ahora sólo podemos elegir el nodo que corresponde a T2. Eliminemos el nodo y el grafo queda vacío. Por lo tanto, el horario seriable H tiene como horario en serie equivalente HSE T1;T2.

Supongamos ahora que tenemos cuatro transacciones (T1, T2, T3 y T4) y que se produce el siguiente horario H:

N.° acción	T1	T2	T3	T4
1	R(A)			
2	W(A)			
3		R(A)		
4			R(A)	
5			R(C)	
6			W(C)	
7		R(B)		
8		W(B)		

N.° acción	T1	T2	T3	T4
9				R(B)
10				R(C)
11				COMMIT
12			R(A)	
13		COMMIT		
14	COMMIT			
15			COMMIT	

El grafo de precedencias asociado al horario H es el que se muestra a continuación:

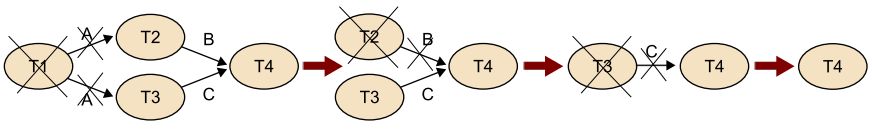


Los arcos corresponden a las siguientes parejas de acciones conflictivas:

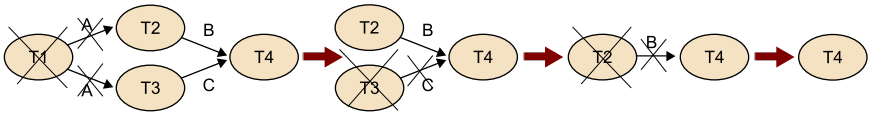
- Arco entre T1 y T2 sobre A. Corresponde a las acciones 2 y 3.
- Arco entre T1 y T3 sobre A. Corresponde a las acciones 2 y 4. También podría corresponder a la pareja de acciones 2 y 12, que daría lugar a otro arco idéntico. Un mismo arco (es decir, con nodo de origen y de destino idéntico e igual etiqueta) sólo se introducirá una vez en el grafo de precedencias, ya que no aporta información nueva.
- Arco entre T2 y T4 sobre B. Corresponde a las acciones 8 y 9.
- Arco entre T3 y T4 sobre C. Corresponde a las acciones 6 y 10.

Como el grafo de precedencias es acíclico, el horario H es seriable y ha de tener asociado algún HSE. Haciendo el recorrido basado en ordenación topológica del grafo de precedencias, encontramos dos HSE.

HSE 1: T1; T2; T3; T4



HSE 2: T1; T3; T2; T4



El motivo por el que hay dos HSE está relacionado con el hecho de que las transacciones T2 y T3 acceden a gránulos diferentes de la BD y, por lo tanto, no hay ninguna pareja de acciones conflictivas entre ellas.

Para finalizar este subapartado, a continuación se muestran posibles horarios (y los grafos de precedencias asociados) que corresponden a las interferencias que hemos estudiado anteriormente.

Ved también

Hemos estudiado las interferencias entre transacciones en el apartado 3 de este módulo didáctico.

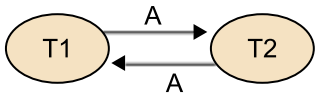
1) Actualización perdida. Un posible horario para esta interferencia es:

N.° acción	T1	T2
1	R(A)	
2		R(A)
3	W(A)	
4		W(A)
5	COMMIT	
6		COMMIT

Las transacciones T1 y T2 leen el mismo gránulo y recuperan el mismo valor. A continuación, las dos transacciones modifican el gránulo. En nuestro ejemplo, el cambio efectuado por T1 en la acción 3 se pierde, aunque T1 confirma resultados (la transacción T1 no cumple la propiedad de definitividad; de hecho, es como si T1 nunca hubiera existido). El valor que finalmente queda como definitivo en la BD es el que deja la transacción T2. Esto no habría pasado nunca en un horario seriable. El grafo de precedencias (con un ciclo sobre el gránulo A) asociado al horario es el que se muestra a continuación:

Recordad

Los horarios seriables siempre dan el mismo resultado que algún horario en serie.



2) Lectura no confirmada. Un posible horario para esta interferencia es:

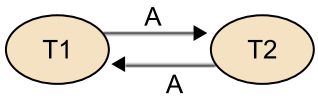
Cancelación de las transacciones

En este subapartado no se trata la interferencia de lectura no confirmada cuando se produce la cancelación de una de las transacciones implicadas.

Recordad que la teoría de la seriabilidad ignora la posibilidad de cancelaciones de las transacciones.

N.° acción	T1	T2
1		R(A)
2		W(A)
3	R(A)	
4	COMMIT	
5		R(A)
6		W(A)
7		COMMIT

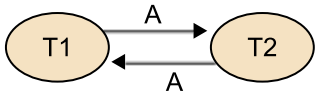
La transacción T1 recupera un valor provisional, pendiente de confirmación y propuesto por T2, que nunca será definitivo sobre la BD, puesto que T2 sobrescribe el valor. Esto no habría pasado nunca en un horario seriable. El grafo de precedencias (con un ciclo sobre el gránulo A) asociado al horario es el que se muestra a continuación:



3) Lectura no repetible. Un posible horario para esta interferencia es:

N.° acción	T1	T2
1		R(A)
2	R(A)	
3	W(A)	
4		R(A)
5	COMMIT	
6		COMMIT

La transacción T2 encuentra dos valores diferentes para el gránulo A: la primera lectura encuentra el valor original de A en la BD, mientras que la segunda lectura encuentra el valor propuesto por la transacción T1. Esto no habría pasado nunca en un horario seriable. El grafo de precedencias (con un ciclo sobre el gránulo A) asociado al horario es el que se muestra a continuación:

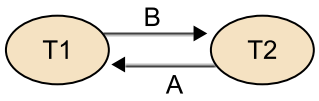


4) Análisis inconsistente. Un posible horario para esta interferencia es:

N.° acción	T1	T2
1	R(B)	
2		R(A)
3		W(A)
4	R(A)	
5	COMMIT	
6		R(B)
7		W(B)

N.° acción	T1	T2
8		COMMIT

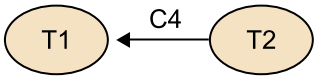
La transacción T1 no encuentra valores correctos para los gránulos A y B. En el caso del gránulo B, encuentra los valores que hay en la BD antes de que empiece la ejecución de la transacción T2. En cambio, en el caso del gránulo A, la transacción T1 encuentra los valores que propone la transacción T2 para el gránulo A. Esto no habría pasado nunca en un horario seriable. El grafo de precedencias (con un ciclo entre los gránulos A y B) asociado al horario es el que se muestra a continuación:



Como recordaréis, entre las interferencias de análisis inconsistente teníamos el caso particular de fantasmas. Es especialmente interesante la relación entre las interferencias de fantasmas y la seriabilidad. Si intentamos representar de una manera simplificada el primer ejemplo que vimos al ver las interferencias provocadas por un fantasma, obtendremos:

N.° acción	T1	T2
1	R(C1)	
2	R(C2)	
3	R(C3)	
4		R(C4)
5		W(C4)
6	R(C1)	
7	R(C2)	
8	R(C3)	
9	R(C4)	
10	COMMIT	
11		COMMIT

C1, C2, C3 y C4 son los gránulos que almacenan los datos de las cuentas 1, 2, 3 y 4 respectivamente. El horario es seriable, ya que es equivalente a un horario en serie en el que se ejecuta primero la transacción T2 y después la transacción T1. Si hacemos el grafo de precedencias veremos que es acíclico:



Nota

En el ejemplo suponemos que los datos de cada cuenta corriente están en una página diferente (recordad que, por defecto, consideramos que el tamaño del gránulo es la página).

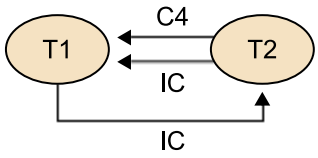
Para crear una nueva cuenta corriente (la cuenta 4), hay que leer una página con suficiente espacio libre (representada por el gránulo C4) y añadir la nueva cuenta a la página escogida (esto queda representado por la acción W(C4) en el ejemplo).

A primera vista, este resultado es confuso, ya que el horario parece correcto y, no obstante, corresponde a una interferencia que queremos impedir. El origen de esta confusión es bastante sutil. Hay que tener en cuenta que el SGBD, para leer todos los registros de la tabla de cuentas, tiene que consultar algún tipo de información de control interna, oculta al usuario, que le permita recorrer estos registros y saber cuándo ya no queda ninguno más.

Además, para crear un nuevo registro hay que actualizar esta misma información de control. Si suponemos, simplificándolo, que toda esta información de control está almacenada en un gránulo IC, obtendremos el siguiente horario, más ajustado a la realidad que el anterior:

N.° acción	T1	T2
1	R(IC)	
2	R(C1)	
3	R(C2)	
4	R(C3)	
5		R(C4)
6		W(C4)
7		R(IC)
8		W(IC)
9	R(IC)	
10	R(C1)	
11	R(C2)	
12	R(C3)	
13	R(C4)	
14	COMMIT	
15		COMMIT

Este horario no es seriable, ya que las acciones sobre IC y C4 fuerzan relaciones de precedencia incompatibles entre las transacciones, tal como muestra el siguiente grafo de precedencias asociado al horario:



También hay que tener presente que, siempre que una transacción pide al SGBD que acceda a los registros que cumplen una condición de búsqueda C, se tienen que considerar todos los registros, incluyendo los que finalmente se vea que no satisfacen C. Esto es necesario para prever la posibilidad de que otras transacciones concurrentes los actualicen de manera que cumplan C, cosa que daría lugar también a una interferencia de tipo fantasma.

Así pues, para que el criterio de seriabilidad, tal como se ha descrito, sea correcto, en los horarios se tienen que considerar las acciones de lectura y escritura de toda la información que utiliza el SGBD para llevar a cabo búsquedas de datos, tanto si las aplicaciones pueden acceder a esta información como si ésta sólo es de uso interno del SGBD.

5.2. Recuperabilidad

Ya hemos visto que algunas interferencias se producen al cancelar las transacciones. Cancelar una transacción representa deshacer todos sus cambios y recuperar el valor anterior que había en la BD de los gránulos que ha modificado la transacción que cancela su ejecución. Esto puede provocar interferencias si estos mismos gránulos han sido leídos o escritos por otras transacciones.

La seriabilidad es un criterio de aislamiento que ignora la posibilidad de que se produzcan cancelaciones. Por lo tanto, para evitar las interferencias que éstas provocan, tenemos que exigir nuevas condiciones a la ejecución de las transacciones.

Un horario cumple el **criterio de recuperabilidad** si ninguna transacción Ti que lee o escribe un gránulo escrito por otra transacción Tj confirma sin que antes lo haya hecho Tj.

El horario del ejemplo que seguidamente proponemos no verifica el criterio de recuperabilidad:

N.° acción	T1	T2
1		R(A)
2		W(A)
3	R(A)	
4	COMMIT	
5		ROLLBACK

Nota

En el horario del ejemplo, si T2, en vez de cancelar los resultados, los hubiera confirmado (ejecutando como acción 5 una operación de `COMMIT`), la interferencia de lectura no confirmada no se habría producido (en este supuesto, el horario en serie equivalente sería T2;T1).

A pesar de esto, el horario tampoco verificaría el criterio de recuperabilidad, ya que T1 recupera datos no confirmados y confirma los resultados antes de que lo haga T2.

De hecho, el horario presenta una interferencia de lectura no confirmada. T1 lee un dato modificado por T2 y que está pendiente de confirmación. Además, T1 finaliza su ejecución (confirmando resultados) antes de que se acabe la ejecución de la transacción T2. Finalmente, dado que la transacción T2 no confirma los resultados, la interferencia de lectura no confirmada al final se produce, ya que la transacción T1 ha recuperado un valor que nunca debería haber recuperado.

La situación todavía podría ser peor si la transacción que recupera datos pendientes de confirmación los intenta modificar, tal como muestra el siguiente horario:

N.° acción	T1	T2
1		R(A)
2		W(A)
3	R(A)	
4	W(A)	
5	COMMIT	
6		ROLLBACK

En este caso, la transacción T1 recupera un dato pendiente de confirmación y, basándose en la lectura hecha, realiza una modificación (acción 4). Cuando la transacción T2 cancela su ejecución, el SGBD restaura el valor que había del gránulo A en la BD antes de que se inicie la ejecución de la transacción T2. Por lo tanto, a consecuencia de esta cancelación, se pierde el cambio que ha efectuado la transacción T1 sobre A. Esto hace que no se verifique la propiedad de definitividad para la transacción T1.

El SGBD puede tratar el problema que muestran los ejemplos previos de dos maneras posibles:

1) Impedir que las transacciones trabajen con datos pendientes de confirmación. Esto se puede conseguir, por ejemplo, bloqueando la ejecución de la transacción que quiere trabajar con estos datos hasta que la transacción que haya hecho las modificaciones finalice su ejecución. En nuestro primer ejemplo, el horario quedaría tal como sigue:

N.° acción	T1	T2	Comentarios
1		R(A)	T2 lee el gránulo A.
2		W(A)	T2 cambia el gránulo A.
3	[R(A)]		T1 quiere leer el gránulo A, que ha sido modificado por T2. T2 está en ejecución y, por lo tanto, éste es un dato provisional. El SGBD bloquea la ejecución de T1, que no efectúa la lectura. No se procesan nuevas peticiones de T1 hasta la finalización de T2.
4		ROLLBACK	T2 cancela su ejecución. El SGBD descarta los cambios producidos por la transacción T2. En concreto, el SGBD restaura el valor que había del gránulo A antes de que se iniciara la ejecución de T2.
5	R(A)		T1 desbloquea su ejecución, efectúa la lectura del gránulo A y encuentra un valor correcto.
6	COMMIT		T1 acaba su ejecución y confirma los resultados.

2) Permitir las transacciones que trabajen con datos no confirmados, siempre que estas transacciones no intenten confirmar los resultados antes de la finalización de la transacción que ha modificado los datos. En caso de que una transacción que haya trabajado con datos no confirmados quiera confirmar resultados, el SGBD bloqueará su ejecución. Es más, la finalización de la transacción que ha trabajado con datos no confirmados queda supeditada a la finalización de la transacción que ha modificado los datos y ambas acabarán su ejecución de la misma manera. De nuevo, en nuestro primer ejemplo, el horario quedaría tal como sigue:

N.° acción	T1	T2	Comentarios
1		R(A)	T2 lee el gránulo A.
2		W(A)	T2 cambia el gránulo A.
3	R(A)		El SGBD permite que T1 lea datos aunque estén pendientes de confirmación.
4	[COMMIT]		T1 quiere confirmar los resultados. El SGBD no se lo permite y bloquea la ejecución hasta la finalización de la transacción T2. T1 y T2 finalizarán su ejecución de la misma manera.
5		ROLLBACK	T2 cancela su ejecución. El SGBD descarta los cambios producidos por la transacción T2. En concreto, el SGBD restaura el valor que había del gránulo A antes de que se iniciara la ejecución de T2.

N.° acción	T1	T2	Comentarios
6	[COMMIT] ABORT		T1 desbloquea su ejecución y no puede confirmar los resultados, dado que ha recuperado valores que finalmente no se confirman. El SGBD aborta la ejecución lanzando una acción de ABORT.

Cancelación involuntaria

Usaremos la acción de `ABORT` para representar la cancelación involuntaria (es decir, inducida por el SGBD) de una transacción.

Las consecuencias de su ejecución son idénticas a la cancelación voluntaria de una transacción (acción de `ROLLBACK`).

La consecuencia de esta manera de trabajar es que, tal como se puede observar en el ejemplo anterior, la cancelación de una transacción T implica la cancelación de todas las transacciones que hayan trabajado con algún gránulo que haya escrito la transacción T. Esta situación puede provocar una **cascada de cancelaciones**, tal como muestra el siguiente ejemplo:

N.° acción	T1	T2	T3	T4	Comentarios
1	R(A)				
2	W(A)				
3		R(A)			T2 lee un dato pendiente de confirmación.
4		R(B)			
5		W(B)			
6			R(B)		T3 lee un dato pendiente de confirmación.
7			R(C)		
8			W(C)		
9				R(C)	T4 lee un dato pendiente de confirmación.
10				[COMMIT]	T4 quiere confirmar los resultados. El SGBD no se lo permite y bloquea la ejecución hasta la finalización de la transacción T3.
11			[COMMIT]		T3 quiere confirmar los resultados. El SGBD no se lo permite y bloquea la ejecución hasta la finalización de la transacción T2.
12		[COMMIT]			T2 quiere confirmar los resultados. El SGBD no se lo permite y bloquea la ejecución hasta la finalización de la transacción T1.
13	ROLLBACK	ABORT	ABORT	ABORT	T1 cancela su ejecución. La cancelación de T1 provoca la cancelación de T2, que, a su vez, origina la de T3. Finalmente, la cancelación de T3 desencadena la cancelación de T4.

Cancelar transacciones es un proceso costoso (y más todavía si estas transacciones estaban dispuestas a confirmar sus resultados). Por esto, en general, los SGBD no permiten este tipo de comportamiento y tienden a impedir que las transacciones recuperen datos pendientes de confirmación.

6. Visión externa de las transacciones

El SQL estándar fuerza a que, una vez que se haya establecido una conexión con la BD, la primera sentencia SQL que queramos ejecutar mediante el SQL interactivo **implícitamente** inicie la ejecución de una transacción. Una vez iniciada la transacción, ésta permanecerá activa hasta que **explícitamente** y de una manera obligatoria indiquemos su finalización.

Última versión del SQL estándar

Cuando hablemos de las sentencias SQL, siempre nos referiremos a la última versión del SQL estándar, ya que tiene como subconjunto todas las anteriores y, por lo tanto, todo lo que era válido en la anterior lo continuará siendo en la siguiente. Sólo especificaremos el año de una versión del SQL cuando queramos enfatizar que se hizo una aportación determinada concretamente en esa versión.

Por defecto, el SQL estándar fuerza que esta transacción nunca vea interferida su ejecución y que tampoco pueda interferir en la ejecución de otras transacciones. En definitiva, por defecto, el SGBD deberá garantizar el correcto aislamiento de todas las transacciones que accedan de forma concurrente a la BD. En otras palabras, el SGBD tendrá que garantizar la seriabilidad y la recuperabilidad del horario que se produzca.

Para informar sobre las características asociadas a una transacción desde el SQL:1992, disponemos de la siguiente sentencia:

```
SET TRANSACTION <modo_acceso>;
```

Notación

La notación para representar la sintaxis de las sentencias SQL será la siguiente:

- Las palabras en negrita son palabras reservadas del lenguaje.
- La notación `[...]` quiere decir que lo que hay entre los corchetes es opcional.
- La notación `{A|...|B}` quiere decir que hemos de escoger entre todas las opciones que hay entre las llaves, pero que debemos poner una obligatoriamente.

en la que `<modo_acceso>` puede ser `READ ONLY`, en caso de que la transacción sólo consulte la BD, o `READ WRITE`, en caso de que la transacción modifique la BD.

La sentencia previa sólo se puede ejecutar en caso de que no haya ninguna transacción en ejecución en la sesión de trabajo establecida con la BD; si hay alguna, el SQL estándar especifica que el SGBD debería reportar una situación de error. Adicionalmente, las características especificadas serán aplicables al resto de transacciones que se ejecuten posteriormente durante la sesión de trabajo.

Para indicar la finalización de una transacción, el SQL estándar nos ofrece la sentencia siguiente:

```
{COMMIT|ROLLBACK}[WORK];
```

Mientras que `COMMIT` confirma todos los cambios producidos contra la BD durante la ejecución de la transacción, `ROLLBACK` los deshace y deja la BD como estaba antes de que se iniciara la transacción. La palabra reservada `WORK` sólo sirve para explicar qué hace la sentencia y es opcional.

Ejemplos de uso de la sentencia `SET TRANSACTION`

Supongamos que tenemos una BD de un banco que guarda datos de las cuentas de los clientes. En concreto, consideremos que tenemos la siguiente tabla (clave primaria subrayada):

```
cuentas(num_cuenta, tipo_cuenta, saldo, comision)
```

Considerando que hemos establecido la conexión con la BD, podemos ejecutar las siguientes sentencias durante nuestra sesión de trabajo con los efectos que se comentan:

Sentencia	Comentarios
`SET TRANSACTION READ WRITE;`	Informamos de que las transacciones que se ejecutarán en la sesión establecida pueden hacer lecturas y cambios en la BD.

`UPDATE cuentas`
`SET saldo=saldo*1.10`
`WHERE num_cuenta="234509876";`
	Se inicia la ejecución de una transacción que puede hacer lecturas y cambios en la BD.
	Incrementamos en un 10% el saldo de la cuenta número 234509876.

`SELECT *`
`FROM cuentas`
`WHERE num_cuenta="234509876";`
	Recuperamos los datos de la cuenta número 234509876.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.

`UPDATE cuentas`
`SET saldo=saldo-500`
`WHERE num_cuenta="234509876";`
	Esta sentencia inicia implícitamente la ejecución de una nueva transacción que puede hacer lecturas y cambios en la BD.
	Transferimos 500 € de la cuenta 234509876 a la cuenta 987656574. Suponemos que hay suficiente saldo.
	Disminuimos el saldo de la cuenta de origen.

`UPDATE cuentas`
`SET saldo=saldo+500`
`WHERE num_cuenta="987656574";`
	Incrementamos el saldo de la cuenta de destino.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.

`SELECT saldo`
`FROM cuentas`
`WHERE tipo_cuenta="ahorro a plazo";`
	Esta sentencia inicia implícitamente la ejecución de una nueva transacción que puede hacer lecturas y cambios en la BD.
	Consultamos el saldo de las cuentas de un tipo determinado.

`SET TRANSACTION READ ONLY;`	Esta sentencia genera un error que nos será reportado por el SGBD. No podemos cambiar las características de las transacciones porque ya tenemos una transacción en ejecución.
`ROLLBACK;`	Como se ha producido un error, cancelamos la transacción.

Sentencia	Comentarios
`SET TRANSACTION READ ONLY;`	Informamos de que las transacciones que se ejecuten en la sesión de trabajo a partir de este momento sólo leerán la BD. Además, la sentencia también inicia la ejecución de una transacción.

`SELECT saldo`
`FROM cuentas`
`WHERE tipo_cuenta="ahorro a plazo";`
	Consultamos el saldo de las cuentas de un tipo determinado.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.

El comienzo implícito de transacciones, en un entorno de aplicación real, puede crear confusiones sobre el alcance de cada transacción, si este alcance no se documenta correctamente. Por esto, el SQL:1999 propone utilizar la siguiente sentencia:

```
START TRANSACTION [<modo_acceso>];
```

Inicio de las transacciones

Muchos SGBD incorporan sentencias propias para marcar de forma explícita el inicio de las transacciones. En la mayoría de los casos, esta sentencia es `BEGIN WORK` o, simplemente, `BEGIN`, porque la palabra clave `WORK` es opcional.

en la que `<modo_acceso>` puede ser `READ ONLY` o `READ WRITE`. Si no se especifica el modo de acceso, la sentencia simplemente inicia la ejecución de una nueva transacción, de acuerdo con las características que se hayan especificado previamente. Si antes no se ha especificado ninguna característica, el SQL estándar enuncia que la transacción se tiene que considerar de tipo `READ WRITE`.

Ejemplos de uso de la sentencia `START TRANSACTION`

En la BD del ejemplo anterior y asumiendo que hemos establecido la conexión con la BD, podemos ejecutar las siguientes sentencias con los efectos que se comentan:

Sentencia	Comentarios
`START TRANSACTION READ ONLY;`	Informamos de que empieza la ejecución de una transacción de sólo lectura.

`SELECT saldo`
`FROM cuentas`
`WHERE tipo_cuenta="ahorro a plazo";`
	Consultamos el saldo de las cuentas de un tipo determinado.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.
`START TRANSACTION READ WRITE;`	Se inicia la ejecución de una transacción que puede consultar y modificar la BD.

`UPDATE cuentas`
`SET saldo=saldo*1.10`
`WHERE num_cuenta="234509876";`
	Incrementamos en un 10% el saldo de la cuenta número 234509876.

`SELECT *`
`FROM cuentas`
`WHERE num_cuenta="234509876";`
	Recuperamos los datos de la cuenta número 234509876.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.

Sentencia	Comentarios
`START TRANSACTION;`	Inicio de una nueva transacción. No se indican sus características. Por lo tanto, se aplican las especificadas anteriormente. En consecuencia, la transacción puede consultar la BD y modificarla.

`UPDATE cuentas`
`SET saldo=saldo-500`
`WHERE num_cuenta="234509876";`
	Transferencia bancaria.
	Disminuimos el saldo de la cuenta de origen.

`UPDATE cuentas`
`SET saldo=saldo+500`
`WHERE num_cuenta="987656574";`
	Incrementamos el saldo de la cuenta de destino.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.

`SELECT saldo`
`FROM cuentas`
`WHERE tipo_cuenta="ahorro a plazo";`
	Esta sentencia inicia implícitamente la ejecución de una nueva transacción que puede hacer lecturas y cambios en la BD. Por lo tanto, no es obligatorio marcar explícitamente el inicio de las transacciones, aunque sea conveniente. De esta manera se asegura la compatibilidad con las versiones previas del estándar.
	Consultamos el saldo de las cuentas de un tipo determinado.

`COMMIT;`	Confirmamos los resultados producidos por la transacción.

6.1. Relajación del nivel de aislamiento

Ved también

Hemos presentado el nivel de concurrencia en el apartado 4 de este módulo didáctico.

Hasta ahora habíamos considerado que siempre era necesario garantizar la seriabilidad y la recuperabilidad de las transacciones para garantizar una protección total ante cualquier tipo de interferencias. No obstante, esta protección total exige una sobrecarga del SGBD en términos de gestión de información de control y una disminución del nivel de concurrencia.

En determinadas circunstancias, es conveniente relajar el nivel de aislamiento y posibilitar que se produzcan interferencias. Esto es correcto si se sabe que estas interferencias no ocurrirán realmente o si en el entorno de aplicación en el que nos encontramos no es importante que se produzcan.

Si nos centramos en el SQL estándar, las instrucciones `SET TRANSACTION` y `START TRANSACTION` permiten relajar el nivel de aislamiento. Tienen la siguiente sintaxis:

```
SET TRANSACTION {READ ONLY|READ WRITE},
ISOLATION LEVEL <nivel_aislamiento>;

START TRANSACTION [{READ ONLY|READ WRITE}],
ISOLATION LEVEL <nivel_aislamiento>;
```

en la que `<nivel_aislamiento>` puede ser `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` o `SERIALIZABLE`.

El **nivel de aislamiento** determina las interferencias que pueden desencadenar otras transacciones en la transacción que empieza. De acuerdo con los tipos de interferencia que hemos descrito, la siguiente tabla indica las que se evitan con cada nivel de aislamiento:

Ved también

Hemos estudiado los tipos de interferencias entre transacciones en el apartado 3 de este módulo didáctico.

	Actualización perdida	Lectura no confirmada	Lectura no repetible y análisis inconsistente (excepto fantasmas)	Fantasmas
READ UNCOMMITTED	Sí	No	No	No
READ COMMITTED	Sí	Sí	No	No
REPEATABLE READ	Sí	Sí	Sí	No
SERIALIZABLE	Sí	Sí	Sí	Sí

En ella los niveles aparecen de menos a más estrictos y, por lo tanto, de menos a más eficientes:

1) El nivel **READ UNCOMMITTED** protege los datos actualizados y evita que ninguna otra transacción los actualice hasta que no se acabe la transacción. No ofrece ninguna garantía con respecto a los datos que lea la transacción. Pueden ser datos actualizados por una transacción que todavía no ha confirmado y, además, otra transacción los puede actualizar inmediatamente.

2) El nivel **READ COMMITTED** protege parcialmente las lecturas e impide que la transacción lea los datos actualizados por otra transacción que todavía no se hayan confirmado.

3) El nivel **REPEATABLE READ** impide que otra transacción actualice un dato que haya leído la transacción hasta que ésta no se acabe. De esta manera, la transacción puede volver a leer este dato sin riesgo de que lo hayan cambiado.

4) El nivel **SERIALIZABLE** ofrece un aislamiento total y evita cualquier tipo de interferencias, incluyendo los fantasmas. Esto significa que no solamente protege los datos que haya visto la transacción, sino también cualquier información de control que se haya utilizado para hacer búsquedas.

La definición del SQL estándar establece que un SGBD concreto tiene la obligación de garantizar como mínimo el nivel de aislamiento que la transacción haya solicitado, aunque puede optar por ofrecer un aislamiento más elevado. Por lo tanto, el único nivel que un SGBD tiene la obligación de implementar es el más alto, el SERIALIZABLE.

6.2. Responsabilidades del SGBD y del desarrollador

Hemos visto qué es una transacción y qué propiedades tiene que cumplir. Examinemos la contribución del SGBD para conseguir garantizar estas propiedades y los aspectos que dependen del desarrollador de las aplicaciones.

Ved también

Hemos presentado el concepto de *transacción* y las propiedades que tiene que cumplir en el apartado 2 de este módulo didáctico.

1) Responsabilidades del SGBD

a) Conseguir que el horario que se produzca a medida que el SGBD recibe peticiones de lectura o escritura, y de `COMMIT` o `ROLLBACK` de las transacciones que se ejecuten de forma concurrente sobre la BD, sea seriable y recuperable. Naturalmente, en caso de que se haya relajado el nivel de aislamiento para algunas transacciones, será necesario que el SGBD considere correctos más horarios.

El SGBD consigue la seriabilidad y la recuperabilidad de los horarios, sobre todo, de dos maneras (no necesariamente excluyentes entre ellas): cancelando automáticamente las transacciones problemáticas o suspendiendo la ejecución de la transacción hasta que la pueda retomar sin problemas. El conjunto de mecanismos que se responsabiliza de estas tareas se llama **control de concurrencia**.

Es necesario que estos mecanismos sean tan transparentes a la programación como sea posible, de manera que no se añadan dificultades innecesarias al desarrollo. No obstante, a veces es necesario ofrecer servicios[1] que modifiquen el comportamiento por defecto del SGBD para aumentar el nivel de concurrencia.

[(1)]Por ejemplo, la posibilidad de relajar el nivel de aislamiento de las transacciones.

b) Comprobar que los cambios que ha hecho una transacción verifican todas las reglas de integridad que se han definido en la BD. Esto se puede hacer justo antes de aceptar el `COMMIT` de la transacción, rechazándolo si se viola alguna regla, o inmediatamente después de que se ejecute cada petición dentro de la transacción.

c) Impedir que en la BD permanezcan cambios de transacciones que no se llegan a confirmar y que se pierdan los cambios que han llevado a cabo transacciones confirmadas en caso de que se produzcan cancelaciones de transacciones, caídas del SGBD o de las aplicaciones, desastres (como incendios) o fallos de los dispositivos externos de almacenaje. En general, hablamos de **recuperación** para referirnos al conjunto de mecanismos que se encargan de estas tareas.

2) Tareas del desarrollador de aplicaciones

a) Identificar con precisión las transacciones de una aplicación, es decir, el conjunto de operaciones que necesariamente se tiene que ejecutar de una manera atómica sobre la BD de acuerdo con los requerimientos de los usuarios.

En este sentido, las transacciones tendrían que durar el mínimo imprescindible. En concreto, puede ser muy peligroso que una aplicación tenga una transacción en ejecución mientras se espera la entrada de información por parte del usuario. A veces, los usuarios pueden tardar bastante rato en proporcionar ciertos datos o, simplemente, en apretar el botón de aceptación de un mensaje. Incluso es posible que cualquier circunstancia los haga dejar a medias lo que hacían y que la aplicación se quede bastante tiempo a la espera de que el usuario vuelva a ella. Hasta que el usuario no permite que la transacción se acabe, ésta puede impedir la actualización o incluso la lectura de los datos a los que ya haya accedido. Esto significa más gasto de recursos y un freno importante en el nivel de concurrencia posible. Por lo tanto, y siempre que sea posible, se suele recomendar que durante una transacción no se pare nunca la ejecución de la aplicación a la espera de que se produzca una actuación determinada por parte del usuario.

b) Garantizar que las transacciones mantienen la consistencia de la BD de acuerdo con los requerimientos de los usuarios y teniendo en cuenta las restricciones de integridad y los disparadores definidos en la BD.

c) Considerar aspectos de rendimiento. En particular, el desarrollador tiene que ser capaz de estudiar y mejorar el nivel de concurrencia de acuerdo con los conocimientos que tenga de los mecanismos de control de concurrencia del SGBD y las posibilidades de modificar su funcionamiento.

7. Control de concurrencia mediante reservas

Hay diversas técnicas de control de concurrencia, cada una de las cuales presenta múltiples variantes. No obstante, las técnicas basadas en reservas suelen ser las más utilizadas en los SGBD. Dentro de las técnicas basadas en reservas, nosotros veremos la más básica, las reservas S y X[2].

(2)La técnica de reservas S y X que veremos en este apartado se aplica, con pequeñas variantes, en SGBD como Informix y DB2.

7.1. Petición y liberación de reservas

La idea básica del uso de reservas es que una transacción tiene que obtener una reserva de un gránulo antes de poder operar con él. Inicialmente, hay dos tipos o modalidades de reservas: las **reservas compartidas** (en inglés, *shared*), o reservas S, que permiten llevar a cabo lecturas del gránulo, y las **reservas exclusivas** (en inglés, *exclusive*), o reservas X, que permiten hacer lecturas y escrituras.

Para pedir una reserva de un gránulo G, con modalidad m (S o X), una transacción tiene que ejecutar una acción (u operación) de adquisición de reserva, que denominaremos lock(G,m). Para liberar una reserva del gránulo G, será necesario que la transacción ejecute una acción para liberar la reserva, que denominaremos unlock(G).

Cuando una transacción pide una reserva sobre un gránulo, el SGBD decide si se la puede conceder, cosa que hará si el tipo de reserva que se le pide no es incompatible con ninguna de las reservas que el SGBD ya haya concedido para el mismo gránulo. La siguiente tabla, conocida como **matriz de compatibilidad**, indica las modalidades de reservas que son compatibles entre ellas y las modalidades que no lo son, en el caso de las reservas S y X:

	Compartida (S)	**Exclusiva (X)**
Compartida (S)	Sí	No
Exclusiva (X)	No	No

Esta tabla nos muestra que, en un momento dado, un gránulo puede estar reservado por N transacciones con modalidad S o bien por una única transacción con modalidad X.

Más concretamente, el significado de las acciones lock(G,m) y unlock(G) ejecutadas por una transacción T queda descrito tal como sigue:

1) Significado de la operación de lock(G,m)

a) Si el gránulo G no está reservado por ninguna otra transacción, la reserva sobre el gránulo G con la modalidad deseada se otorga a T. La transacción T puede continuar con su ejecución.

El SGBD necesita tener constancia de que se ha otorgado una reserva con modalidad m a la transacción T. Por esto, para cada posible gránulo G de la BD, el SGBD mantiene una lista que guarda qué transacciones tienen reserva concedida sobre el gránulo G y con qué modalidad. En definitiva, cada vez que se otorga una reserva con modalidad m sobre un gránulo G en beneficio de una transacción T, el SGBD añade una nueva entrada (un par <T, m>) a la **lista de transacciones** que tienen reserva concedida sobre el gránulo G.

b) Si el gránulo G está reservado por otras transacciones que tienen G reservado con modalidades compatibles con la modalidad que pide T, la reserva sobre el gránulo G con modalidad m se otorga a T y el SGBD añade una nueva entrada a la lista de transacciones que tienen reserva concedida sobre el gránulo G. La transacción T puede continuar con su ejecución.

Trabajando con reservas S y X, este caso representa la situación en la que la transacción T pide reserva S sobre un gránulo G cuando hay otras transacciones que también tienen reserva S sobre el mismo gránulo G. Puesto que las reservas S son compatibles entre ellas, la reserva se puede otorgar a la transacción T.

c) Si el gránulo G está reservado para otras transacciones con modalidades incompatibles con la modalidad que pide T, la reserva no se puede otorgar. En este caso, T bloquea su ejecución. La ejecución de T estará bloqueada hasta que T no pueda adquirir la reserva sobre el gránulo G con la modalidad deseada.

Trabajando con reservas S y X, este caso representa o bien una situación en la que T pide reserva S sobre el gránulo G y existe otra transacción que tiene otorgada una reserva X sobre el gránulo G, o bien una situación en la que T pide una reserva X sobre el gránulo G y existe otra transacción que tiene una reserva (S o X) otorgada sobre el gránulo G.

Adicionalmente, en este caso hay una situación particular que debemos considerar. Una transacción T que ha reservado un gránulo G con una reserva S puede intentar convertirla en reserva X ejecutando una acción de lock(G,X), pero puede ser bloqueada si hay otra transacción Ti que tenía una reserva S del mismo gránulo. En caso de que no haya ninguna transacción Ti trabajando sobre el gránulo G, la reserva será concedida a la transacción T y se tendrá que modificar la entrada[3] correspondiente a T en la lista de transacciones que tie-

[3] Más concretamente, será necesario modificar la modalidad de la reserva otorgada a T sobre el gránulo G.

nen reserva concedida sobre G. Este proceso de transformación de una reserva S en una reserva X en beneficio de una transacción se conoce como **fortalecimiento de una reserva**.

El SGBD necesita saber qué transacciones han bloqueado su ejecución porque no han podido adquirir una reserva con modalidad m sobre un gránulo G. Para hacerlo, el SGBD mantiene, para cada gránulo de la BD, una cola de transacciones bloqueadas. En definitiva, cuando una transacción bloquea su ejecución, el SGBD añade una entrada nueva (un par <T, m>) a la **cola de transacciones** que están bloqueadas en espera de adquirir una reserva sobre G.

2) Significado de la operación de unlock(G)

a) Liberar la reserva que T tenía concedida sobre el gránulo G. Esto implica que el SGBD elimina la entrada correspondiente a T de la lista de transacciones que tienen reserva otorgada sobre el gránulo G.

b) Si hay transacciones bloqueadas, el SGBD, siguiendo el orden de la cola de transacciones bloqueadas a la espera de adquirir reserva sobre el gránulo G, hará las siguientes acciones:

Gestión de la cola de transacciones

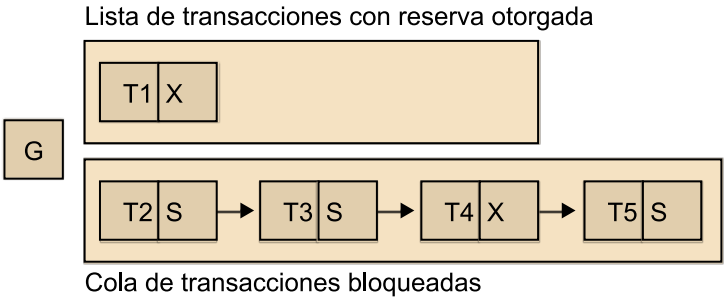
En este módulo seguiremos una política FIFO para gestionar la cola de transacciones bloqueadas a la espera de adquirir reserva, aunque puede haber otras políticas.

- Si la modalidad es compatible con las otras reservas concedidas sobre el gránulo G, la reserva se otorga. La entrada correspondiente a la transacción que estaba bloqueada en la cola de transacciones a la espera de adquirir reserva sobre G pasa a la lista de transacciones que tienen reserva otorgada sobre G. La transacción bloqueada puede retomar su ejecución. Este paso se repite hasta que la cola de transacciones bloqueadas a la espera de adquirir reservas sobre el gránulo G quede vacía, o hasta que encontremos la primera reserva que no se puede conceder.

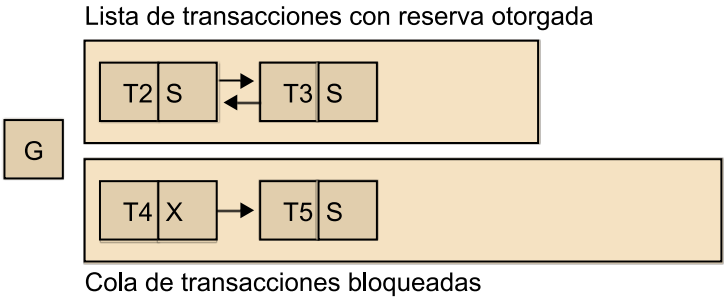
- Si la modalidad es incompatible con las reservas concedidas sobre el gránulo G, la transacción continúa bloqueada en la cola. En este caso, el proceso de otorgar nuevas reservas se detiene con el objetivo de evitar la inanición de las transacciones bloqueadas.

Ejemplo de transacciones bloqueadas

Supongamos que, en un momento dado, sobre un gránulo G tenemos la transacción T1 con reserva otorgada sobre G con modalidad X. Adicionalmente, tenemos que las transacciones T2, T3 y T5 están bloqueadas en la cola de transacciones, a la espera de adquirir una reserva con modalidad S sobre el gránulo G. La transacción T4 también está bloqueada, pero en este caso, a la espera de adquirir una reserva sobre G con modalidad X. Temporalmente, las reservas que no se han podido adquirir vienen en el siguiente orden: primero la que pide T2, después la de T3, a continuación la que pide T4 y, finalmente, la de la transacción T5. Esto daría lugar a la situación que seguidamente se muestra de una manera gráfica:



Supongamos ahora que la transacción T1 quiere liberar su reserva mediante la ejecución de una acción de unlock(G). Si aplicamos el procedimiento descrito, T2 y T3 adquirirían la reserva sobre el gránulo G, ya que las modalidades que las dos piden (modalidad S) son compatibles entre ellas. T4 continuaría bloqueada, puesto que la modalidad que pide (X) es incompatible con las que se otorgan. El proceso se detiene, aunque se podría otorgar la reserva que ha pedido T5. Por lo tanto, T5 continuaría bloqueada. La idea es conseguir que en un momento u otro T4 consiga la reserva con modalidad X que necesita sobre G. En consecuencia, después de la ejecución de unlock(G) por parte de T1, pasaríamos a una situación como la que seguidamente se muestra de una manera gráfica:



7.2. Transacciones bien formadas

La adquisición y la liberación de reservas por parte de las transacciones tienen que seguir ciertas reglas que hacen que **las transacciones estén bien formadas**. Estas reglas son las siguientes:

1) Una transacción T no intentará ejecutar una acción de lectura o escritura sobre un gránulo G si previamente no ha adquirido una reserva sobre el gránulo G que le permita hacer la acción que desea. Por simplicidad, supondremos que la petición de reserva sobre el gránulo G se ejecuta justo antes que la ejecución de la acción de lectura o escritura.

2) Una transacción T no ejecutará una acción de unlock(G) si previamente no había efectuado una operación de lock(G,m).

3) Una transacción T no vuelve a ejecutar una acción de lock(G,m) si antes ya había llevado a cabo la acción de lock(G,m), salvo en el caso de que entre las dos acciones se haya ejecutado una acción de unlock(G). Sin embargo, puede suceder que una transacción fortalezca una reserva adquirida previamente. En el caso de reservas S y X, esto quiere decir que una transacción T puede ejecutar una acción de lock(G,S) y después intentar fortalecer la reserva llevando a cabo una acción de lock(G,X) sin que en medio la transacción T haya ejecutado una operación de unlock(G).

4) Llegará un punto en el tiempo en que la transacción T liberará todas las reservas que tenía concedidas.

Finalmente, asumiremos que cada transacción puede ejecutar las acciones que seguidamente se especifican. En las acciones para las que haya que pedir reservas, también se indican las modalidades de reserva que cada transacción tiene que adquirir en el supuesto de que el SGBD trabaje con reservas S y X.

a) R(G)[4]: lectura del gránulo G. La transacción sólo quiere leer el gránulo. La transacción tiene que adquirir una reserva sobre G con modalidad S antes de intentar ejecutar el R(G).

[4]Las operaciones de lectura (R(G)) se corresponden con la ejecución de sentencias SQL de `SELECT`.

b) RU(G)[5]: lectura con intención de modificación posterior del gránulo G. Más adelante en el tiempo, la transacción querrá ejecutar una operación de escritura (W(G)) sobre el gránulo. La transacción tiene que adquirir una reserva X sobre el gránulo antes de intentar ejecutar el RU(G). Esta reserva, una vez adquirida, le permite hacer tanto la operación de lectura como la acción posterior de escritura.

[5]Las operaciones de lectura con intención de actualización (RU(G)) se corresponden con la ejecución de sentencias SQL de `INSERT`, `DELETE` y `UPDATE`.

En este caso, la transacción también podría pedir una reserva S antes de hacer la acción de RU(G) y fortalecer la reserva en X antes de hacer la operación de W(G). La ventaja de esta opción es que se permite que otras transacciones puedan leer el gránulo mientras la transacción que ha efectuado el RU(G) no quiera efectuar el W(G), lo que hace aumentar el nivel de concurrencia. Esta opción tiene la desventaja de que favorece la aparición de **abrazos mortales** (es decir, esperas indefinidas) entre las transacciones, que es lo que constituye el principal problema inherente a las técnicas basadas en reservas. Cuando se produce un abrazo mortal, el SGBD sólo lo puede solucionar cancelando alguna de las transacciones implicadas en dicho abrazo mortal.

Ved también

En el apartado 7.4 de este módulo didáctico se trata la problemática asociada a los abrazos mortales.

Puesto que la cancelación de transacciones es un proceso muy costoso, el SGBD tiende a evitar las situaciones que favorecen abrazos mortales. Por esto, la transacción pide una reserva con modalidad X cuando se sabe que la lectura tendrá asociada posteriormente una acción de escritura. Una vez adquirida, esta modalidad de reserva permite que la transacción se asegure no sólo la lectura del gránulo G, sino también su escritura posterior.

c) W(G): escritura del gránulo G. Antes, de una manera obligatoria, la transacción tiene que haber hecho una operación de RU(G) y tiene que haber adquirido la reserva correspondiente que le permita hacer las dos acciones. Por lo tanto, no necesita adquirir ninguna reserva adicional antes de hacer la acción de escritura.

d) COMMIT. La transacción quiere acabar su ejecución con una operación de confirmación de resultados. Si no lo ha hecho antes, en este punto liberará todas las reservas que había adquirido durante la ejecución.

e) ROLLBACK. La transacción quiere acabar, de una manera voluntaria, su ejecución con una operación de cancelación de resultados. El SGBD deberá descartar los resultados. Si no lo ha hecho antes, en este punto la transacción liberará todas las reservas que había adquirido durante la ejecución.

f) ABORT. El SGBD ha decidido cancelar la transacción y descarta sus resultados. El SGBD hará los mismos pasos que en la acción de ROLLBACK.

Ejemplos de transacciones que siguen la técnica de reservas S y X

Dadas las siguientes transacciones:

T1	R(B)	RU(C)	W(C)	R(E)	COMMIT
T2	R(F)	R(A)	RU(F)	W(F)	COMMIT

un ejemplo de transacción (por ejemplo, T1) que no está bien formada podría ser el siguiente:

T1	R(B)	LOCK(C,X)	RU(C)	W(C)	LOCK(E,S)	R(E)	UNLOCK(C)	COMMIT

Los motivos por los que la transacción T1 no está bien formada son los siguientes: T1 hace un R(B) sin tener la reserva pertinente y no libera la reserva adquirida sobre E.

Un ejemplo de transacción (de nuevo, por ejemplo, T1) que está bien formada sería el siguiente:

T1	LOCK(B,S)	R(B)	LOCK(C,X)	RU(C)	W(C)	LOCK(E,S)	R(E)	UNLOCK(B)	UNLOCK(C)	UNLOCK(E)	COMMIT

En el caso de reservas S y X, es irrelevante el orden en el que una transacción libera las reservas. Por convención, asumiremos que se liberan en el mismo orden en el que se han adquirido.

Otro ejemplo de transacción bien formada (ahora la transacción T2) sería el siguiente:

T2	LOCK(F,S)	R(F)	LOCK(A,S)	R(A)	LOCK(F,X)	RU(F)	W(F)	UNLOCK(F)	UNLOCK(A)	COMMIT

En este ejemplo es importante destacar el hecho de que la transacción T2 ha necesitado fortalecer la reserva que inicialmente tenía otorgada sobre el gránulo F.

7.3. Protocolo de reservas en dos fases

Aunque los servicios de petición y liberación de reservas que acabamos de describir y el hecho de tener transacciones bien formadas son la base sobre la que se construye el control de la concurrencia, no garantizan nada por sí mismos. Por ejemplo, si las transacciones reservan los gránulos justo antes

de operar con ellos y los liberan inmediatamente después, es evidente que se pueden producir exactamente las mismas interferencias que si no se hace nada, tal como muestra el siguiente ejemplo:

N.° acción	T1	T2
1		LOCK(A,S)
2		R(A)
3		UNLOCK(A)
4	LOCK(A,X)	
5	RU(A)	
6	W(A)	
7	UNLOCK(A)	
8		LOCK(A,S)
9		R(A)
10		UNLOCK(A)
11	COMMIT	
12		COMMIT

En el horario anterior, aunque disponemos de transacciones bien formadas y se utilizan reservas S y X, tenemos una interferencia de lectura no repetible entre las transacciones T1 y T2, puesto que la transacción T2 halla un valor diferente para el gránulo A en cada una de las lecturas que efectúa. Si las transacciones hubieran estado correctamente aisladas entre ellas, esto no habría sucedido nunca.

En consecuencia, es necesario añadir nuevas restricciones en la adquisición y la liberación de reservas por parte de las transacciones.

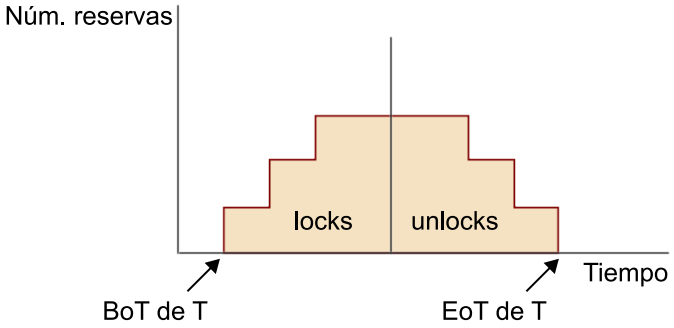
Una transacción cumple lo que se denomina **protocolo de reservas en dos fases** (PR2F) si reserva cualquier gránulo en la modalidad adecuada antes de operar con él y nunca adquiere o fortalece una reserva después de haber liberado cualquier otra antes. Si todas las transacciones utilizan el PR2F y confirman sus resultados, obtendremos horarios seriables.

Recordad

De acuerdo con la teoría de la seriabilidad, los horarios seriables son correctos, es decir, no presentan interferencias.

Hemos estudiado la teoría de la seriabilidad en el subapartado 5.1 de este módulo didáctico.

Alternativamente, de una manera gráfica, podemos ver el PR2F tal como se muestra a continuación:



El eje de ordenadas representa el número de acciones de adquisición de reserva de gránulos que una transacción T lleva a cabo durante su ejecución. Por su parte, el eje de abscisas representa el paso del tiempo:

- la transacción T empieza su ejecución en un instante de tiempo determinado (representado BoT[6])
- y la acaba en otro instante de tiempo (representado EoT[7]).

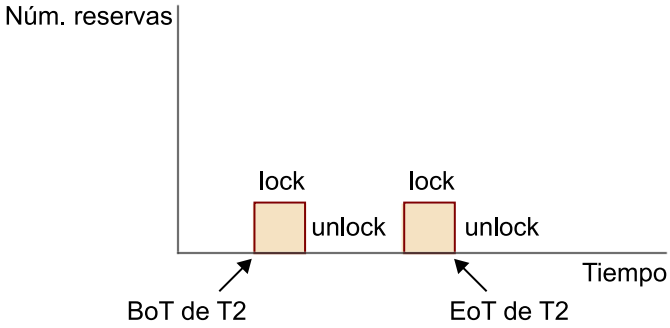
[6]BoT quiere decir *begin of transaction*.

[7]EoT quiere decir *end of transaction*.

Durante la ejecución de T se distinguen dos fases: en la fase conocida como *fase creciente*, T pide las reservas que necesita, y en la fase conocida como *fase menguante*, T libera los gránulos que había reservado previamente. Entre las dos fases, y durante un tiempo, las reservas se mantienen para que la transacción pueda hacer las operaciones que quiere en la BD. Una vez que una transacción T libera su primera reserva, ya no puede pedir más reservas. Por lo tanto, antes de liberar la reserva, una transacción T tiene que estar segura de que no necesita adquirir otras nuevas.

Forzando que las transacciones sigan este protocolo, y bloqueándolas y retomándolas de acuerdo con las peticiones y las liberaciones de reservas, el SGBD consigue alterar horarios no seriables para convertirlos en seriables. Es importante remarcar que el mismo SGBD genera las operaciones de petición y liberación de reservas. Lo hace de una manera transparente, sin que el programador se tenga que preocupar, de acuerdo con las acciones de lectura y escritura recibidas.

En el horario del ejemplo anterior, las transacciones –concretamente la transacción T2– no sigue el PR2F, puesto que libera una reserva sobre un gránulo (acción 3) que posteriormente volverá a necesitar reservar (acción 8). En definitiva, T2 se precipita liberando la reserva que tenía concedida sobre el gránulo y acaba mezclando peticiones de adquisición y liberación de reservas. La gráfica de adquisición y liberación de reservas quedaría tal como sigue:



Forzando que las transacciones sigan el PR2F, el horario del ejemplo habría quedado tal como se muestra a continuación:

N.° acción	T1	T2	Comentarios
1		LOCK(A,S)	T2 consigue una reserva S sobre el gránulo A. T2 no libera A hasta que no lo vuelva a necesitar.
2		R(A)	
3	LOCK(A,X)		T1 intenta conseguir una reserva X sobre el gránulo A. Esta reserva es incompatible con la que está en posesión de T2. T1 bloquea su ejecución hasta que T2 no libere la reserva.
4		R(A)	
5		UNLOCK(A)	T2 libera su reserva sobre el gránulo A. T1 puede adquirir la reserva sobre el gránulo A y desbloquea su ejecución.
6	RU(A)		
7	W(A)		
8		COMMIT	
9	COMMIT		

Las celdas sombreadas representan el rato que la transacción T1 está bloqueada.

Ahora, el horario es correcto (sin interferencias) y, por lo tanto, se puede serializar. El horario en serie equivalente es T2; T1.

Ved también

Hemos estudiado la recuperabilidad en el subapartado 5.2 de este módulo didáctico.

Como ya sabemos, hay interferencias que se producen cuando las transacciones recuperan valores de otras transacciones que cancelan su ejecución. Para que estas situaciones no se produzcan, hay que garantizar la recuperabilidad de los horarios. Por ejemplo, el siguiente horario (que no verifica el criterio de recuperabilidad) presenta una interferencia de lectura no confirmada:

N.° acción	T1	T2
1		RU(A)
2		W(A)
3	R(A)	
4	COMMIT	

N.° acción	T1	T2
5		ROLLBACK

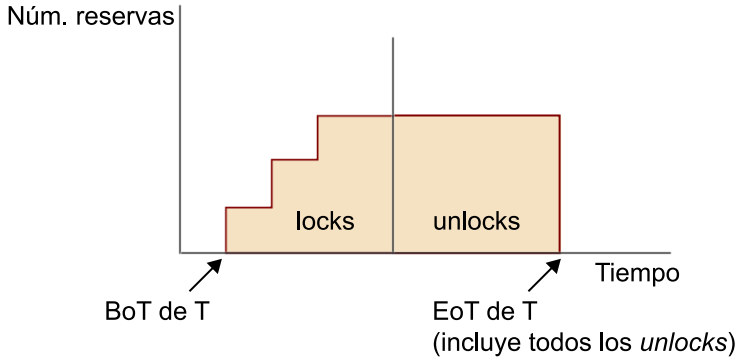
Si las transacciones utilizan el PR2F, el horario podría quedar tal como sigue:

N.° acción	T1	T2
1		LOCK(A,X)
2		RU(A)
3		W(A)
4		UNLOCK(A)
5	LOCK(A,S)	
6	R(A)	
7	UNLOCK(A)	
8	COMMIT	
9		ROLLBACK

El ejemplo anterior muestra que el PR2F que acabamos de ver (también conocido como **PR2F básico**) es insuficiente para tratar estas situaciones. Por esto se tiene que utilizar una variante del protocolo: el PR2F estricto.

Una transacción cumple el PR2F estricto si satisface el PR2F (el básico) y, además, no libera ninguna reserva hasta que no acaba su ejecución. Si todas las transacciones cumplen el PR2F estricto, se garantiza no sólo la seriabilidad, sino también la recuperabilidad de los horarios. Además, también se evita la posibilidad de que se produzcan cancelaciones en cascada de las transacciones.

El PR2F estricto impide que las transacciones puedan trabajar con valores no confirmados. De una manera gráfica, lo podemos representar tal como se muestra a continuación:



Veamos ahora cómo quedaría transformado el horario del ejemplo anterior si se siguiera el PR2F estricto:

N.° acción	T1	T2	Comentarios
		LOCK(A,X)	
1		RU(A)	
2		W(A)	
	LOCK(A,S)		T1 bloquea su ejecución hasta que T2 no libere la reserva.
3		ROLLBACK(U(A))	T2 libera la reserva al acabar (de forma simplificada se ha representado U(A)).
4	R(A)		T1 adquiere la reserva, desbloquea su ejecución y completa las acciones que tiene que hacer.
5	COMMIT (U(A))		

Este horario es correcto, sin ninguna interferencia. Tiene asociado como horario en serie equivalente T1. No se hace ninguna mención de T2 en el horario en serie equivalente, ya que T2 ha cancelado su ejecución y, en consecuencia, oficialmente no ha existido nunca.

El PR2F estricto se usa en los SGBD para garantizar el nivel de aislamiento máximo de las transacciones (nivel `SERIALIZABLE` en el SQL). Entre las ventajas que tiene, destacamos las siguientes:

Ved también

Hemos estudiado los niveles de aislamiento propuestos por el SQL estándar en el subapartado 6.1 de este módulo didáctico.

1) Simplifica la implementación y el uso del sistema: evita el problema de tener que establecer el momento en el que una transacción puede entrar en la segunda fase, libera reservas y renuncia a pedirlas para acceder a nuevos datos.

2) Elimina la posibilidad de cancelaciones en cascada.

Como contrapartida, el nivel de concurrencia que permite alcanzar es más bajo, puesto que los datos están bloqueados hasta la finalización de las transacciones, e impide que otras transacciones tengan acceso a ellos.

El caso particular de los fantasmas se puede tratar mediante el uso de reservas para hacer las lecturas y actualizar cualquier información que se haya utilizado en la búsqueda de los datos, incluyendo información interna o de control.

7.4. Abrazos mortales

La utilización de reservas y la suspensión de la ejecución de transacciones introducen la posibilidad de que algunas transacciones queden en situación de espera indefinida.

Se dice que se ha producido un **abrazo mortal** cuando la ejecución de dos transacciones o más queda bloqueada indefinidamente por el hecho de que, para reanudar la ejecución, todas requieren que otra de las transacciones implicadas libere alguna reserva ya obtenida.

Ejemplo de horario con abrazo mortal

Supongamos que tenemos tres transacciones (T1, T2 y T3) que quieren ejecutar las siguientes acciones:

T1	R(A)	R(B)	COMMIT		
T2	RU(B)	W(B)	RU(C)	W(C)	COMMIT
T3	R(C)	RU(A)	W(A)	COMMIT	

El siguiente horario presenta una situación de abrazo mortal:

N.° acción	T1	T2	T3	Comentarios
1	LOCK(A,S)			
2	R(A)			
3			LOCK(C,S)	
4			R(C)	
5		LOCK(B,X)		
6		RU(B)		
7		W(B)		
8			LOCK(A,X)	T3 ve bloqueada su ejecución hasta que T1 no libera la reserva sobre A.
9		LOCK(C,X)		T2 ve bloqueada su ejecución hasta que T3 no libera la reserva sobre C.
10	LOCK(B,S)			T1 ve bloqueada su ejecución hasta que T2 no libera la reserva sobre B.
11				Situación de abrazo mortal: T1, T2 y T3 están bloqueadas a la espera de adquirir reservas que no se liberarán hasta que el SGBD no cancele una de las tres transacciones.

Ante la posibilidad de que se produzcan abrazos mortales, los SGBD basados en reservas tienen que optar por una de las tres siguientes posibilidades:

- Prevenirlos antes de que se produzcan.
- Detectarlos y resolverlos una vez que se hayan producido.
- Definir un tiempo de espera máximo que, si se supera, haga que se cancele automáticamente la transacción.

Casi todos los SGBD optan por la segunda opción, que es la única que trataremos en este módulo didáctico.

Un SGBD detecta los abrazos mortales buscando ciclos de espera. Puede hacer esta búsqueda en diferentes momentos:

- Siempre que una transacción pida una reserva y no la obtenga inmediatamente. Esto permitiría detectar los abrazos mortales rápidamente, pero no es frecuente, ya que añade un coste excesivo a las peticiones de reserva.

- A intervalos de tiempos regulares, que no deberían ser muy largos.

- Cuando haya transacciones sospechosas (como mínimo dos) que estén más de un tiempo determinado a la espera.

Una vez detectado un abrazo mortal, lo único que puede hacer el SGBD es romper el ciclo cancelando una o diversas de las transacciones implicadas. Para escoger una se puede mirar si alguna transacción participa en diversos abrazos mortales con el fin de romperlos con una única cancelación, o qué transacción hace menos tiempo que se ejecuta (presumiblemente, para deshacer menos trabajo), qué transacción ha hecho menos cambios sobre la BD[8], etc.

Finalmente, también hay SGBD que, una vez escogida la transacción que se tiene que cancelar, en vez de cancelar la transacción completa para resolver un abrazo mortal, cancelan únicamente la última sentencia SQL ejecutada con un aviso de error y dan a la aplicación (o al usuario) la posibilidad de cancelar la transacción o continuar su ejecución lanzando nuevas sentencias SQL. Esto, que podría ser aceptable en ciertos entornos de aplicación, vulnera una de las propiedades de las transacciones, en concreto, la propiedad de atomicidad. Por lo tanto, desde un punto de vista teórico sería un procedimiento incorrecto.

[8]En este sentido, las transacciones que sólo efectúan lecturas sobre la BD son las transacciones ideales para ser canceladas, ya que no cambian la BD y, en consecuencia, no hace falta que el SGBD deshaga nada.

Recordad

A un alto nivel, las aplicaciones y los usuarios trabajan con SQL.

Ved también

Las propiedades de las transacciones se explican en el apartado 2 de este módulo didáctico.

7.5. Reservas y nivel de aislamiento

Hemos visto que en el uso de reservas S y X, el hecho de trabajar con transacciones bien formadas y de utilizar el PR2F estricto garantiza el aislamiento correcto del conjunto de transacciones que se ejecutan de forma concurrente. En otras palabras, estos elementos garantizan la seriabilidad y la recuperabilidad de los horarios.

Sabemos también que el SQL nos permite (por ejemplo con la sentencia `SET TRANSACTION`) especificar el nivel de aislamiento con el que trabajarán las transacciones. Cada nivel de aislamiento evita ciertas interferencias y posibilita la aparición de otras.

El SGBD siempre gestiona las reservas de escritura con el máximo rigor. Esto implica que, con independencia del nivel de aislamiento especificado, todas las reservas con modalidad X (que se corresponden con la ejecución de acciones RU(G) seguidas de acciones W(G)) se mantienen hasta la finalización de la transacción. No obstante, la manera de utilizar las reservas para lectura (que se corresponden con acciones R(G)) puede variar, tal como indicamos a continuación:

1) En el **nivel `READ UNCOMMITTED`** no se efectúa ninguna reserva para lectura (ni para la lectura de datos de la BD ni de datos de control). Los datos que se leen pueden haber sido actualizados por una transacción que todavía no ha confirmado. En este nivel de aislamiento, las transacciones ni están bien formadas ni siguen el PR2F.

2) En el **nivel `READ COMMITTED`** las reservas S de lectura de cada gránulo G se mantienen sólo hasta después de cada lectura del gránulo G. No se efectúa ninguna reserva de datos internos o de control. El uso de reservas garantiza que lo que se lee se ha confirmado, ya que las reservas para actualización siempre se mantienen hasta la finalización de la transacción. En este nivel de aislamiento, las transacciones, aunque están bien formadas, no siguen el PR2F.

3) En el **nivel `REPEATABLE READ`** las reservas S de lectura de cada gránulo G se mantienen hasta que la transacción no necesita volverlas a leer. Como el SGBD sólo puede estar seguro de que una transacción no necesitará volver a leer unos datos hasta la finalización de las transacciones, las reservas S se mantienen hasta la finalización de la transacción. No se efectúa ninguna reserva de datos internos o de control. En este nivel de aislamiento, las transacciones están bien formadas y siguen el PR2F estricto.

4) En el **nivel `SERIALIZABLE`** se piden reservas S para todos los datos que se han leído de la BD, incluyendo también los datos de control. Las reservas se mantienen siempre hasta que las transacciones finalizan. En este nivel de aislamiento, las transacciones están bien formadas y siguen el PR2F estricto, no sólo para los datos que quieren recuperar de la BD, sino también para los datos de control.

8. Recuperación

Los **mecanismos de recuperación** del SGBD tienen que garantizar la atomicidad y la definitividad de las transacciones. El objetivo es que nunca se pierdan los cambios de las transacciones que han confirmado y que nunca se mantengan cambios efectuados por transacciones que cancelan.

El SGBD tiene que tratar adecuadamente las siguientes situaciones:

1) La cancelación voluntaria (`ROLLBACK`) de una transacción a petición de la aplicación.

2) La cancelación involuntaria (`ABORT`) de una transacción a causa de fallos de la aplicación (por ejemplo, la realización de una división por cero), de la violación de restricciones de integridad o de decisiones del mecanismo de control de concurrencia del SGBD (por ejemplo, cuando la transacción está implicada en un abrazo mortal), etc.

3) La caída del sistema (por ejemplo, a causa de una avería de software o de un corte de luz), que provocaría la cancelación de todas las transacciones activas, es decir, de todas las transacciones en ejecución en el momento de la caída.

4) La caída del sistema en caso de que pueda haber cambios efectuados por transacciones confirmadas que todavía no hayan sido transferidos al dispositivo de almacenamiento permanente (por ejemplo, disco) en el que está guardada la BD.

5) La destrucción total o parcial de la BD a causa de desastres (por ejemplo, un incendio o una inundación) o de fallos de los dispositivos (por ejemplo, un error de escritura en el disco).

Las situaciones descritas en 1), 2) y 3) comprometen la propiedad de atomicidad de las transacciones, mientras que las situaciones descritas en 4) y 5) comprometen su propiedad de definitividad.

Podemos distinguir dos partes en la recuperación:

a) La restauración, que garantiza la atomicidad y la definitividad ante cancelaciones (voluntarias o involuntarias) de las transacciones y caídas del sistema.

b) La reconstrucción, que recupera el estado de la BD ante una pérdida total o parcial de los datos que hay almacenados en los dispositivos de almacenamiento externo a causa de fallos o desastres.

A veces también se habla de recuperación para referirse a la restauración.

8.1. Restauración

La restauración tiene que ser capaz de efectuar dos tipos de operaciones: la **restauración hacia atrás**, que implica deshacer los cambios de una transacción que ha cancelado su ejecución, y la **restauración hacia delante**, que comporta rehacer los cambios de una transacción confirmada.

Un factor que hay que tener en cuenta es la utilización de memorias intermedias (en inglés, *buffers*) por parte del SGBD. Las memorias intermedias son zonas de memoria volátil que el SGBD utiliza para guardar los gránulos de la BD a los que acceden las transacciones con el objetivo de mejorar el rendimiento del sistema.

Nota

Las políticas para transferir cambios de las memorias intermedias a la memoria externa pueden variar según el SGBD.

No todas las acciones de lectura y escritura de gránulos solicitadas por las transacciones se traducen en operaciones aplicadas a los dispositivos físicos, ya que algunas se pueden llevar a cabo directamente en las memorias intermedias. Así pues, en un momento dado, hay acciones de escritura que no han llegado a la BD en memoria externa y otras que sí que lo han hecho. En general, y en el peor de los casos, esto puede ser independiente del hecho de que las transacciones hayan confirmado o no: puede haber escrituras confirmadas que no hayan llegado a memoria externa y escrituras no confirmadas que sí que hayan llegado.

Para deshacer cambios y rehacerlos, el SGBD utiliza una estructura de datos con información de cambios, el **dietario** (en inglés, *log*) que guarda información de las modificaciones que han efectuado las transacciones y de las confirmaciones y las cancelaciones de éstas.

Los registros de cambios suelen llevar un identificador de transacción, el estado anterior del gránulo modificado y el posterior. Los registros de confirmación y cancelación tienen que contener el identificador de transacción. Todos estos registros se almacenan en orden cronológico.

La siguiente figura muestra un posible fragmento de dietario:

Primero, T1 cambia el gránulo A, que pasa a tener un nuevo estado, A'. Después, T2 cambia B, que ahora tendrá un nuevo estado, B'. Por su parte, T1 cambia C, que pasa a tener un nuevo estado, C'. Finalmente, T2 confirma, T3 cambia D a un nuevo estado, D', y T1 cancela su ejecución.

Para que el dietario permita deshacer y rehacer transacciones, es indispensable que se escriban los registros de cambio, a partir de los datos en las memorias intermedias, antes de que las modificaciones lleguen a la BD en memoria externa y que las transacciones confirmen.

Veamos cómo se utiliza el dietario:

1) Cuando una transacción cancela su ejecución, tanto si es de una manera voluntaria como involuntaria, se tienen que deshacer los cambios que había hecho hasta aquel momento. Para ello, los registros de cambio en el dietario se recorren hacia atrás. Para que esto sea eficiente, se tienen que enlazar todos los registros de cambio de una misma transacción y se tiene que poder identificar el primer registro de cambio para cada transacción dentro del dietario para no tener que continuar buscando hacia atrás.

2) Cuando se produce una caída, se tienen que deshacer los cambios de todas las transacciones que haya activas, que se cancelan, y rehacer los de todas las transacciones confirmadas que puedan tener modificaciones que no hayan llegado a la BD en memoria externa. Para hacerlo, primero se tiene que recorrer el dietario hacia atrás, y recuperar valores anteriores de las transacciones no confirmadas, y después hacia adelante, y recuperar valores posteriores de transacciones confirmadas.

La dificultad que plantea el comportamiento en caso de caídas es saber hasta qué punto hay que recorrer el dietario hacia atrás y a partir de qué punto se tiene que recorrer hacia adelante. Es decir, hay que saber a partir de qué posición del dietario no habrá más registros de cambio ni de transacciones activas, ni de transacciones confirmadas con cambios que no hayan llegado a la memoria externa.

Esto se soluciona mediante un nuevo tipo de registro del dietario: los registros de punto de control.

Un **registro de punto de control** identifica un momento en el que el SGBD lleva a la memoria externa todos los gránulos modificados que hay en las memorias intermedias. El registro contiene, además, una lista de todas las transacciones activas en ese instante.

Para deshacer transacciones, el SGBD sólo tendrá que recorrer el dietario hasta que encuentre todos los registros de cambio de las transacciones activas en el último punto de control. Para rehacerlas, sólo tendrá que buscar registros de transacciones confirmadas a partir de este último punto de control. Los SGBD tienen que seguir una política determinada de generación de puntos de control, cosa que harán cada cierto tiempo o según cualquier otro criterio.

Es importante destacar que los SGBD, en función de las políticas que definan para transferir los gránulos modificados desde las memorias intermedias hasta la memoria externa, pueden simplificar considerablemente los procesos de restauración que hemos descrito, tal como se muestra a continuación:

1) Si antes de confirmar una transacción el SGBD siempre lleva todos los cambios a la memoria externa, ante una caída nunca será necesario rehacer transacciones confirmadas.

2) Si el SGBD no lleva cambios de transacciones no confirmadas a la memoria externa, nunca será necesario deshacer los cambios de transacciones canceladas.

El inconveniente de estas políticas es que pueden empeorar el rendimiento o gastar demasiados recursos a causa de las restricciones impuestas en la gestión de memorias intermedias. Por ejemplo, en el caso de un SGBD que utilizara la política 2), una transacción de actualización masiva de datos tendría que guardar todos los gránulos que contienen estos datos cambiados en las memorias intermedias, como mínimo, hasta la confirmación de la transacción.

8.2. Reconstrucción

Para poder reconstruir el estado de una BD después de una pérdida parcial o total de datos, es necesario utilizar dos fuentes de información:

a) Una copia de seguridad que contenga un estado correcto de la BD.

b) El contenido del dietario a partir del momento en el que se hizo la copia de seguridad.

Con respecto a la copia de seguridad, para cada BD se tiene que establecer una política de realización de copias que garantice que el trabajo de un período demasiado largo de tiempo no se perderá nunca. Esta política tiene que consi-

derar aspectos como, por ejemplo, la localización física de la BD y de las copias de seguridad (si están en el mismo sitio aumenta el riesgo de pérdida). Naturalmente, el esfuerzo que se haga tiene que ser proporcional a la importancia de la pérdida de información.

Podemos hacer una clasificación de las copias de seguridad según las siguientes características:

1) Las copias de seguridad pueden ser **estáticas**, también llamadas *copias en frío*, o **dinámicas**, también llamadas *copias en caliente*. Las primeras exigen que se detenga la actividad de los usuarios y las aplicaciones (cuando menos, las actualizaciones), mientras que las segundas no.

2) Las copias de seguridad pueden ser **completas** o **incrementales**. Las completas copian todo el estado de la BD, y las incrementales, sólo los cambios efectuados desde que se ha hecho la copia anterior.

La información que guardan los registros del dietario es necesaria para hacer las mismas tareas ante una caída del sistema después de recuperar el estado de una copia de seguridad. De esta manera, se puede evitar la pérdida de todos los cambios que se hayan hecho desde la última copia de seguridad. Para hacerlo posible es conveniente almacenar el dietario en dispositivos físicos diferentes de los de la BD, para que no se pierdan junto con ésta.

9. Transacciones en PostgreSQL

Por defecto, y si no se indica expresamente lo contrario, PostgreSQL trabaja con transacciones implícitas (este modo de trabajo también se conoce con el nombre de *autocommit activado*). Esto quiere decir que cualquier grupo de sentencias SQL que seleccionemos (por ejemplo, desde el PgAdmin) y enviemos a ejecutar será tratado como una transacción. Si el grupo de sentencias enviado no genera ningún error, los resultados pasarán a ser definitivos en la BD. De lo contrario, los resultados serán descartados por el SGBD.

Ya sabemos que trabajar con transacciones implícitas en un entorno de aplicación real puede crear confusiones sobre el alcance de cada transacción. Por esto, PostgreSQL nos ofrece la sentencia del SQL estándar `START TRANSACTION`, y también una sentencia propia, la sentencia `BEGIN`, para indicar de forma explícita el comienzo de una transacción. Cuando se indica explícitamente el comienzo de una transacción, PostgreSQL desactiva la modalidad *autocommit* y la transacción permanecerá activa hasta que confirmemos o cancelemos sus resultados de forma explícita. Para indicar la finalización de la transacción, disponemos de las sentencias del SQL estándar `COMMIT` y `ROLLBACK`.

Ved también

Hemos presentado las sentencias del SQL para trabajar con transacciones en el apartado 6 de este módulo didáctico.

Adicionalmente, también tenemos disponible la sentencia `SET TRANSACTION` del SQL estándar para indicar las características de la transacción (si es `READ ONLY` o `READ WRITE` y el nivel de aislamiento) en caso de que no se haya hecho anteriormente; por ejemplo, con las sentencias `START TRANSACTION` o `BEGIN`. Si el usuario no ha especificado ninguna característica para las transacciones que quiere ejecutar, por defecto, PostgreSQL considerará que son transacciones `READ WRITE` que trabajan con un nivel de aislamiento `READ COMMITTED`.

Aunque PostgreSQL permite especificar cualquiera de los niveles de aislamiento propuestos por el SQL estándar (`READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` y `SERIALIZABLE`), de hecho, internamente sólo trabaja con dos niveles de aislamiento. Estos niveles son los de `READ COMMITTED` y `SERIALIZABLE`:

Nota

Si indicamos un nivel de aislamiento `READ UNCOMMITTED`, PostgreSQL, internamente, lo transformará en `READ COMMITTED`.

Si indicamos un nivel de aislamiento `REPEATABLE READ`, PostgreSQL, internamente, lo transformará en `SERIALIZABLE`.

1) El nivel de aislamiento `READ COMMITTED` evita que la transacción se vea involucrada en interferencias de actualización perdida y de lectura no confirmada. La transacción se podría ver implicada en interferencias de lectura no repetible y de análisis inconsistente (incluyendo fantasmas).

2) Por su parte, el nivel de aislamiento `SERIALIZABLE` evita cualquier tipo de interferencia.

El hecho de que PostgreSQL sólo tenga que considerar internamente dos niveles de aislamiento está relacionado con el mecanismo para el control de concurrencia que implementa. Este mecanismo se basa en lo que se conoce como *modelo de control de concurrencia multiversión* (en inglés, *multiversion concurrency control*, abreviado MVCC), que explicamos seguidamente.

MVCC

El modelo de control de concurrencia MVCC no sólo se halla disponible en PostgreSQL, sino que también lo implementan otros SGBD, como por ejemplo, Oracle y SQL Server.

Para cada gránulo[9] de la BD, el MVCC mantiene diversas versiones. Las diferentes versiones de un mismo gránulo reflejan las distintas acciones de escritura (W(G)) que las transacciones `READ WRITE` realizan. Cuando una transacción quiere acceder a un gránulo, el SGBD escoge la versión más adecuada a las necesidades de la transacción. Adicionalmente, cuando una transacción escribe un gránulo, si el SGBD autoriza la escritura, se crea una nueva versión de gránulo y se conservan las versiones anteriores. Esta nueva versión de gránulo sólo pasa a ser visible para el resto de transacciones cuando la transacción que ha creado la versión confirma los resultados.

(9)El tamaño habitual de gránulo en el MVCC es el registro (o la fila).

Los usuarios y las aplicaciones no son conscientes de que hay múltiples versiones de un mismo gránulo; el SGBD, internamente, decide qué versión de gránulo hay que escoger.

En el MVCC, las acciones de sólo lectura (R(G)) nunca generan reservas, ni tampoco ven nunca bloqueada la ejecución. Todavía más, estas acciones de lectura siempre leen datos confirmados. El SGBD escoge la versión más adecuada para sus intereses. El hecho de que una versión sea más adecuada depende del nivel de aislamiento con el que trabaje la transacción que intenta ejecutar las acciones de lectura:

Recordad

Las acciones de sólo lectura (R(G)), en términos del SQL, se corresponden con sentencias `SELECT`. Las acciones de sólo lectura pueden ser ejecutadas tanto por transacciones `READ ONLY` como por transacciones `READ WRITE`.

1) En caso de que la transacción trabaje con un nivel de aislamiento `READ COMMITTED`, la versión de gránulo que se escoge en beneficio de la transacción es la versión más recientemente confirmada.

2) Si la transacción trabaja con un nivel de aislamiento `SERIALIZABLE`, se escoge la versión más recientemente confirmada en la BD y que, al mismo tiempo, sea previa al instante de inicio de la transacción.

Por su parte, en el MVCC, las acciones de actualización (es decir, las acciones de RU(G) seguidas de acciones de W(G)) requieren la adquisición de una reserva sobre el gránulo en modalidad exclusiva (modalidad de reserva X) por parte de la transacción que quiere efectuar la operación de actualización. Esta reserva se mantiene (de acuerdo con el PR2F estricto) hasta la finalización de la transacción. La adquisición de la reserva autoriza la creación de una nueva versión del gránulo que sólo será visible en el resto de transacciones cuando la transacción autorizada a crear la nueva versión del gránulo confirme los re-

Recordad

Las acciones de actualización (RU(G) seguidas de W(G)), en términos del SQL, se corresponden con sentencias `INSERT`, `DELETE` y `UPDATE`. Las acciones de actualización sólo pueden ser realizadas por transacciones `READ WRITE`.

sultados. La creación de la nueva versión del gránulo se hace siempre a partir de la versión del gránulo más recientemente confirmada, para evitar que se puedan perder cambios confirmados.

Ved también

Hemos explicado la modalidad de reserva X y el PR2F estricto en el apartado 7 de este módulo didáctico.

Evidentemente, como se utilizan reservas con modalidad X para gestionar las acciones de actualización que efectúan las transacciones, se puede producir una situación de abrazo mortal. Para solucionarla, se necesitará que el SGBD cancele alguna de las transacciones implicadas. En el caso de PostgreSQL, no se puede conocer *a priori* qué transacción es la que se cancelará. Una vez escogida la transacción que se quiere cancelar, PostgreSQL descartará todos los cambios que la transacción haya producido y reportará un error al usuario o la aplicación que había iniciado la ejecución de la transacción.

A continuación, mostramos un par de ejemplos de aplicación del mecanismo MVCC para el caso de ejecución concurrente de transacciones `READ ONLY` y `READ WRITE`.

Ejemplo de aplicación del mecanismo MVCC

Supongamos que tenemos una tabla de cuentas (clave primaria subrayada) con las columnas y las filas que se indican a continuación:

Cuentas		
num_cuenta	saldo	tipo_cuenta
234509876	100	a la vista
897654323	200	a la vista
435789000	18000	ahorro a plazo

Imaginemos que tenemos dos transacciones, T1 (`READ ONLY`) y T2 (`READ WRITE`), que quieren ejecutar las siguientes sentencias SQL:

T1	`SELECT SALDO`
	`FROM CUENTAS`
	`WHERE NUM_CUENTA="234509876";`
		`SELECT SALDO`
		`FROM CUENTAS`
		`WHERE NUM_CUENTA="234509876";`
			`COMMIT;`

T2	`UPDATE CUENTAS`
	`SET SALDO=SALDO+100`
	`WHERE NUM_CUENTA="234509876";`
			`COMMIT;`

Suponiendo que el tamaño del gránulo es la fila, veamos qué resultados produce un horario como el que seguidamente se indica en caso de que T1 trabaje con un nivel de aislamiento `READ COMMITTED` y T2 trabaje con un nivel de aislamiento `SERIALIZABLE`[10]:

T1	T2	Comentarios
`START TRANSACTION READ ONLY, ISOLATION LEVEL READ COMMITTED;`		

(10)De hecho, en el ejemplo, es irrelevante el nivel de aislamiento con el que trabaje T2. Independientemente del nivel de aislamiento, T2 siempre mostrará el mismo comportamiento.

T1	T2	Comentarios
	`START TRANSACTION READ WRITE, ISOLATION LEVEL SERIALIZABLE;`	
`SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";`		Se recupera un saldo de 100 para T1 (versión más recientemente confirmada).
	`UPDATE CUENTAS SET SALDO=SALDO+100 WHERE NUM_CUENTA="234509876";`	La ejecución de esta sentencia requiere que internamente se genere una petición de adquisición de reserva X sobre la fila a la que se quiere acceder. Ninguna transacción tiene reserva X sobre la fila que corresponde a la cuenta número 234509876. La reserva se otorga y se autoriza la creación de una nueva versión de gránulo (nueva fila). El cambio sólo es visible para la transacción T2.
	`COMMIT;`	T2 libera su reserva. Tenemos una nueva versión para la fila que corresponde a la cuenta número 234509876. Esta nueva versión pasa a ser pública y tiene un saldo para la cuenta de 200 €.
`SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";`		Se recupera un saldo de 200 € para T1 (versión más recientemente confirmada). T1 lee dos veces el mismo dato y en cada lectura, aunque recupera valores confirmados, obtiene valores diferentes. Se ha producido una interferencia de lectura no repetible. Esto es porque la transacción T1 trabaja con un nivel `READ COMMITTED`.
`COMMIT;`		

Veamos ahora qué pasaría si las 2 transacciones trabajaran con un nivel `SERIALIZABLE`:

T1	T2	Comentarios
`START TRANSACTION READ ONLY, ISOLATION LEVEL SERIALIZABLE;`		
	`START TRANSACTION READ WRITE, ISOLATION LEVEL SERIALIZABLE;`	
`SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";`		Se recupera un saldo de 100 para T1 (versión más recientemente confirmada que existe antes de que se inicie la ejecución de T1).
	`UPDATE CUENTAS SET SALDO=SALDO+100 WHERE NUM_CUENTA="234509876";`	La ejecución de esta sentencia requiere que internamente se genere una petición de adquisición de reserva X sobre la fila a la que se quiere acceder. Ninguna transacción tiene reserva X sobre la fila que corresponde a la cuenta número 234509876. La reserva se otorga y se autoriza la creación de una nueva versión de gránulo (nueva fila). El cambio sólo es visible para la transacción T2.
	`COMMIT;`	T2 libera su reserva. Tenemos una nueva versión para la fila que corresponde a la cuenta número 234509876. Esta nueva versión pasa a ser pública y tiene un saldo para la cuenta de 200 €.

T1	T2	Comentarios
SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";		Se recupera un saldo de 100 para T1 (versión más recientemente confirmada que existe antes de que se inicie la ejecución de T1). Se evita la interferencia de lectura no repetible. En definitiva, el nivel SERIALIZABLE evita que se puedan leer datos que han sido confirmados con posterioridad al inicio de la transacción que intenta leer estos datos. El horario es correcto y tiene como horario en serie equivalente el que vendría dado por T1;T2.
COMMIT;		

Tratamiento del mecanismo MVCC de las transacciones READ WRITE

Un caso especialmente problemático es el del tratamiento que el mecanismo MVCC hace de las transacciones READ WRITE cuando éstas también ejecutan operaciones de sólo lectura (ejecución de sentencias SELECT del SQL). El problema se presenta porque el MVCC no efectúa reservas con modalidad S para gestionar estas operaciones.

Tomemos de nuevo nuestra tabla de cuentas con los datos indicados inicialmente. Supongamos, además, que tenemos dos transacciones, T1 y T2, de tipo READ WRITE que quieren ejecutar las siguientes sentencias SQL:

T1	SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";	UPDATE CUENTAS SET SALDO=SALDO+100 WHERE NUM_CUENTA="234509876";	SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";	COMMIT;
T2	UPDATE CUENTAS SET SALDO=SALDO+50 WHERE NUM_CUENTA="234509876";	COMMIT;		

Considerando que el tamaño del gránulo es la fila, veamos qué resultados produce un horario como el que se indica a continuación en caso de que T1 trabaje con un nivel de aislamiento READ COMMITTED y T2, con un nivel de aislamiento SERIALIZABLE[11]:

T1	T2	Comentarios
START TRANSACTION READ WRITE, ISOLATION LEVEL READ COMMITTED;		
	START TRANSACTION READ WRITE, ISOLATION LEVEL SERIALIZABLE;	
SELECT SALDO FROM CUENTAS WHERE NUM_CUENTA="234509876";		Se recupera un saldo de 100 para T1 (versión más recientemente confirmada).
	UPDATE CUENTAS SET SALDO=SALDO+50 WHERE NUM_CUENTA="234509876";	La ejecución de esta sentencia requiere que internamente se genere una petición de adquisición de reserva X sobre la fila a la que se quiere acceder. Ninguna transacción tiene reserva X sobre la fila que corresponde a la cuenta número 234509876. La reserva se otorga y se autoriza la creación de una nueva versión de gránulo (nueva fila). El cambio sólo es visible para la transacción T2.

[11] De nuevo, en el ejemplo, es irrelevante el nivel de aislamiento con el que trabaje T2.

T1	T2	Comentarios
	`COMMIT;`	T2 libera su reserva. Tenemos una nueva versión para la fila que corresponde a la cuenta número 234509876. Esta nueva versión pasa a ser pública y tiene un saldo para la cuenta de 150 €.

`UPDATE CUENTAS`
`SET SALDO=SALDO+100`
`WHERE NUM_CUENTA="234509876";`
		La ejecución de esta sentencia requiere que internamente se genere una petición de adquisición de reserva X sobre la fila a la que se quiere acceder.
		Ninguna transacción tiene reserva X sobre la fila que corresponde a la cuenta número 234509876. La reserva se otorga y se autoriza la creación de una nueva versión de gránulo (nueva fila). Esta nueva versión se crea a partir de la versión de gránulo más recientemente confirmada (esta versión de gránulo es la que ha sido creada para la transacción T2). El cambio sólo es visible para la transacción T1.
		El saldo de la cuenta número 234509876 pasa a ser de 250 €.

`SELECT SALDO`
`FROM CUENTAS`
`WHERE NUM_CUENTA="234509876";`
		Se recupera un valor de saldo para la cuenta número 234509876 de 250 €. Se ha producido una interferencia.

`COMMIT;`		

La ejecución anterior muestra que la transacción T1 ha visto interferida su ejecución por la transacción T2. Si T1 se hubiera ejecutado de una manera aislada, sin solapamientos con la ejecución de T2, la segunda operación de `SELECT` que efectúa habría recuperado un valor para el saldo de la cuenta corriente número 234509876 de 200 €, en vez de 250. Todavía más, el valor de 200 € sería el valor esperado por la transacción T1, puesto que la primera sentencia `SELECT` que ejecuta T1 recupera un valor de 100 € y la transacción T1 incrementa el saldo únicamente en 100 unidades. La fuente del problema es que la operación de `UPDATE` que ejecuta T1 obligatoriamente se ha de basar en la nueva fila que crea y confirma la transacción T2. La ejecución de este `UPDATE` se traduce internamente en una nueva operación de lectura con la intención posterior de actualización (es decir, en una secuencia de RU(G) seguida de W(G)). En consecuencia, se produce una interferencia de lectura no repetible. Como la transacción T1 trabaja con un nivel de aislamiento `READ COMMITTED`, se acepta el hecho de que la interferencia de lectura no repetible se pueda (como efectivamente pasa) producir.

Puesto que la creación de una nueva versión de gránulo siempre se efectúa a partir de la versión más recientemente confirmada (de lo contrario, como ya se ha explicado, se podrían perder cambios confirmados), el mecanismo MVCC resuelve la problemática antes planteada en el nivel de aislamiento `SERIALIZABLE` cancelando la ejecución de la transacción que intenta actualizar datos que ha leído previamente, si éstos han sido cambiados y confirmados por otra transacción que se ejecuta de forma concurrente.

Recordad

El nivel de aislamiento `SERIALIZABLE` garantiza que no se produce ningún tipo de interferencia, es decir, garantiza que las transacciones verifican la propiedad de aislamiento.

Sobre nuestras transacciones del ejemplo, T1 y T2, y considerando ahora que las dos trabajan con un nivel de aislamiento `SERIALIZABLE`, la ejecución concurrente de las dos produciría los resultados que se presentan a continuación:

T1	T2	Comentarios
`START TRANSACTION READ WRITE,` `ISOLATION LEVEL SERIALIZABLE;`		
	`START TRANSACTION READ WRITE,` `ISOLATION LEVEL SERIALIZABLE;`	

`SELECT SALDO`
`FROM CUENTAS`
`WHERE NUM_CUENTA="234509876";`
		Se recupera un saldo de 100 para T1 (versión más recientemente confirmada antes de que se inicie la ejecución de T1).

T1	T2	Comentarios

	`UPDATE CUENTAS`
	`SET SALDO=SALDO+50`
	`WHERE NUM_CUENTA="234509876";`
		La ejecución de esta sentencia requiere que internamente se genere una petición de adquisición de reserva X sobre la fila a la que se quiere acceder.
		Ninguna transacción tiene reserva X sobre la fila que corresponde a la cuenta número 234509876. La reserva se otorga y se autoriza la creación de una nueva versión de gránulo (nueva fila). El cambio sólo es visible para la transacción T2.

	`COMMIT;`	T2 libera su reserva. Tenemos una nueva versión para la fila que corresponde a la cuenta número 234509876. Esta nueva versión pasa a ser pública y tiene un saldo para la cuenta de 150 €.

`UPDATE CUENTAS`
`SET SALDO=SALDO+100`
`WHERE NUM_CUENTA="234509876";`
		En este punto, el SGBD cancela la ejecución de la transacción T1 porque intenta modificar datos que ha leído previamente y que han cambiado a causa de la presencia de una transacción que ha confirmado los cambios. En consecuencia, se evita la interferencia.
		El SGBD reporta la cancelación de la transacción por medio de un mensaje de error.
		Los posibles cambios que haya hecho la transacción (en nuestro caso concreto del ejemplo, ninguno) sobre la BD serán descartados.

`-- Aviso de error y ROLLBACK de la transacción`		

Para acabar este apartado, hemos de decir que las ventajas principales del MVCC se pueden resumir tal como sigue:

1) Se incrementa significativamente el nivel de concurrencia en relación con las reservas S y X. Las transacciones `READ ONLY` nunca bloquean su ejecución a causa de las transacciones `READ WRITE` ni a la inversa. Las transacciones `READ WRITE` sólo pueden suspender su ejecución si entran en conflicto con otras transacciones `READ WRITE`.

2) Siempre se garantiza la recuperabilidad de los horarios por el hecho de que las transacciones siempre leen datos confirmados.

3) El MVCC facilita los mecanismos de restauración hacia atrás. Para deshacer los resultados de una transacción que cancela su ejecución, simplemente hay que destruir (liberando el espacio) las versiones que haya podido crear.

4) El MVCC facilita la realización de copias de seguridad dinámicas de la BD.

Por otra parte, entre las desventajas principales del MVCC destacaremos las siguientes:

1) Es necesario disponer de cierta capacidad de almacenaje (en relación con las reservas S y X) para absorber la creación de nuevas versiones de gránulos, y más si la BD se actualiza frecuentemente. Si hiciera falta, el SGBD puede

purgar periódicamente versiones antiguas de gránulos y liberar el espacio que ocupaban estas versiones. Si se quiere forzar la purga de versiones antiguas, en el caso de PostgreSQL disponemos de la sentencia `VACUUM`.

2) En ciertos casos se puede producir la cancelación de transacciones `READ WRITE` por el hecho de que estas transacciones no generan reservas con modalidad S para poder ejecutar operaciones de sólo lectura (sentencias `SELECT` del SQL).

Resumen

El objetivo de la gestión de transacciones es garantizar la integridad de la BD ante el acceso simultáneo de múltiples usuarios y evitar así cualquier pérdida de información.

El acceso a una BD se hace por medio de transacciones. Una transacción tiene que verificar las propiedades ACID, es decir, la atomicidad, que garantiza que se ejecutan todas las operaciones o no se hace ninguna de ellas; la consistencia, para garantizar que la BD quede en un estado correcto, en el que se verifiquen las reglas de integridad que se hayan definido en la misma; el aislamiento, para que las actualizaciones de una transacción no interfieran en el trabajo de otras, y la definitividad, que evita la pérdida de los cambios de las transacciones que confirman su ejecución.

Para asegurar el aislamiento correcto de las transacciones, el SGBD tiene que garantizar la seriabilidad y la recuperabilidad de los horarios. Para hacerlo, los SGBD pueden utilizar técnicas como las reservas o el modelo de control de concurrencia multiversión (MVCC).

Por otro lado, los mecanismos de recuperación se encargan de garantizar la atomicidad y la definitividad de las transacciones. Para poder llevar a cabo la recuperación (restauración o reconstrucción) de la BD, es necesario disponer de dietarios y de copias de seguridad de la misma.

Además, el desarrollador de una aplicación tiene que identificar correctamente las transacciones (de acuerdo con los requerimientos de los usuarios) y garantizar la integridad de los datos en el ámbito de la aplicación. También tiene que garantizar que el nivel de concurrencia permita un rendimiento correcto, considerando factores como la relajación de los niveles de aislamiento o las esperas a causa de la interacción con el usuario.

Actividades

1. Como actividad, os proponemos que probéis los ejemplos que hemos presentado en el apartado 9 en relación con el mecanismo de control de concurrencia MVCC sobre PostgreSQL.

Para hacerlo, además de crear la tabla de cuentas e insertar en ella las filas indicadas, tendréis que abrir dos sesiones de trabajo diferentes con el SGBD; por ejemplo, con el PgAdmin.

En cada sesión, será necesario ejecutar una de las transacciones; por ejemplo, en la primera sesión, la transacción T1 y en la segunda sesión, la transacción T2. Debéis tener especial cuidado de ejecutar las operaciones asociadas a cada transacción en el orden que se indica en el módulo. Para hacerlo, tendréis que cambiar de una sesión a la otra y ejecutar las operaciones de una en una.

También es conveniente que marquéis explícitamente el inicio de las transacciones (y las características que tienen) con el fin de desactivar la ejecución implícita de transacciones.

Finalmente, pensad que puede haber cambios en la sintaxis de alguna de las sentencias SQL en PostgreSQL. En el material didáctico, de una manera deliberada, se ha utilizado la sintaxis de SQL estándar. Por lo tanto, tendréis que revisar los manuales de sintaxis de PostgreSQL.

Ejercicios de autoevaluación

1. Sea un SGBD **sin ningún mecanismo de control de concurrencia**, supongamos que se produce el horario que mostramos a continuación (en el que R = lectura, RU = lectura con intención de escritura, y W = escritura; las acciones se han numerado para facilitar su referencia):

N.° acción	T1	T2	T3	T4
1	R(C)			
2			RU(E)	
3			W(E)	
4		RU(B)		
5		W(B)		
6				R(A)
7			R(F)	
8		R(C)		
9	RU(F)			
10	W(F)			
11		RU(A)		
12		W(A)		
13				R(B)
14		RU(E)		
15		W(E)		
16			R(F)	
17	COMMIT			
18			COMMIT	

N.° acción	T1	T2	T3	T4
19		COMMIT		
20				COMMIT

a) Decid si hay interferencias, cuáles, entre qué transacciones y para qué gránulos. Dad el grafo de precedencias asociado.

b) ¿Es seriable el horario anterior? ¿Y recuperable? Justificad la respuesta brevemente.

2. Tenemos una tabla R(X,Y), en la que la clave primaria está subrayada y las filas tienen valores (1,1), (2,2), (3,3), etc. También tenemos las siguientes transacciones:

T1	`SELECT SUM(Y) FROM R`	`COMMIT`		

T2	`UPDATE R`
	`SET Y=Y*2`
	`WHERE X=20`
		`UPDATE R`
		`SET Y=Y*2`
		`WHERE X=30`
			`UPDATE R`
			`SET Y=Y*2`
			`WHERE X=40`
				`COMMIT`

T3	`DELETE FROM R` `WHERE X=20`	`DELETE FROM R` `WHERE X=30`	`COMMIT`	

Antes de que ninguna de estas transacciones se ejecute, la suma de los valores de Y de la tabla R es 2015 (esta suma incluye los valores de Y de 20, 30 y 40).

También sabemos que el SGBD usa reservas S y X como mecanismo de control de concurrencia.

Si las transacciones anteriores se ejecutan de forma concurrente utilizando el nivel de aislamiento de SQL `SERIALIZABLE`, explicad **brevemente** qué sumas pueden ser producidas por la transacción T1. Supongamos que las lecturas que lleva a cabo la transacción T1 se ejecutan siempre seguidas. Consideremos también que el gránulo que se utiliza es la fila.

3. Sea un SGBD **sin ningún mecanismo de control de concurrencia**, supongamos que se produce el horario que mostramos a continuación (en el que R = lectura, RU = lectura con intención de escritura, y W = escritura; las acciones se han numerado para facilitar su referencia):

N.° acción	T1	T2	T3	T4
1	RU(B)			
2	W(B)			
3				R(D)
4		R(A)		
5		R(B)		
6			RU(A)	
7			W(A)	
8				RU(C)
9	RU(C)			
10				W(C)
11	W(C)			
12		R(A)		

N.° acción	T1	T2	T3	T4
13			RU(D)	
14			W(D)	
15				COMMIT
16			COMMIT	
17	COMMIT			
18		COMMIT		

a) Decid si hay interferencias, cuáles, entre qué transacciones y para qué gránulos. Dad el grafo de precedencias asociado.

b) ¿Es seriable el horario anterior? ¿Y recuperable? Justificad la respuesta **brevemente**.

c) Supongamos ahora que las cuatro transacciones trabajan con el mismo nivel de aislamiento y que el control de concurrencia se efectúa mediante reservas S y X. Indicad cómo quedaría el horario anterior para los niveles de aislamiento `READ UNCOMMITTED` y `SERIALIZABLE`. Para cada uno de estos niveles, indicad también los horarios en serie equivalentes que se producen a consecuencia de aplicar reservas.

4. Supongamos que tenemos el siguiente fragmento del dietario:

N.° registro	Registros
1	T1, inicio_transacción
2	Punto de control: <T1>
3	T1, escribir A, 10, 20
4	T1, escribir B, 20, 30
5	T1, confirmar
6	T2, inicio_transacción
7	T3, inicio_transacción
8	T2, escribir C, 30, 40
9	Punto de control <T2, T3>
10	T3, escribir A, 20, 60
11	T3, cancelar
12	T2, escribir D, 10, 20
13	T4, inicio_transacción
14	T4, escribir F, 30, 40
15	T4, confirmar
16	← Caída del sistema

El significado de los diversos registros del dietario se describe a continuación:

- Ti, inicio_transacción: Ti inicia la ejecución.

- Tj, escribir, P, v1, v2: Tj modifica P, el valor de P antes de la modificación es v1 y después de la modificación es v2.
- Ti, confirmar: Ti confirma los resultados.
- Ti, cancelar: Ti cancela los resultados.
- Punto de control: registro de punto de control: <Lista de transacciones activas>

a) ¿Qué acciones llevará a cabo nuestro SGBD para poder restaurar el estado de la BD después de la caída del sistema si la política que sigue el SGBD es la descrita en el material docente, es decir, en cada punto de control se llevan a la memoria externa todos los cambios de las memorias intermedias?

b) ¿Qué cambia en la respuesta del apartado a) si ahora, en el momento de confirmar una transacción, el SGBD también lleva los cambios hechos por esta transacción a la memoria externa?

5. Consideremos un SGBD que, por lo que se refiere a recuperación, trabaja con la política que seguidamente se describe:

- Nunca lleva cambios de transacciones en ejecución (es decir, de transacciones que todavía no han confirmado los resultados) desde las memorias intermedias hasta la memoria externa.
- Cuando una transacción confirma, el SGBD transfiere los cambios que la transacción haya hecho desde las memorias intermedias hasta la memoria externa.

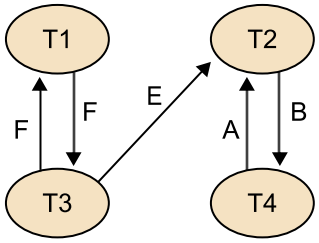
En el entorno descrito, ¿podemos afirmar que no hace falta que el SGBD mantenga un dietario?

Solucionario

Ejercicios de autoevaluación

1.

a) El grafo de precedencias asociado al horario es:



Se dan las siguientes interferencias:

- Lectura no repetible entre las transacciones T1 y T3 sobre el gránulo F, acciones 7-10-16.
- Análisis inconsistente entre T2 y T4 sobre los gránulos A y B, acciones 5-6-12-13.

b) Cómo hemos visto en el punto a), se producen interferencias y, por lo tanto, no se trata de un horario seriable.

Con respecto a la recuperabilidad, un horario es recuperable cuando ninguna transacción Tx que lee o escribe un gránulo escrito por otra transacción Ty confirma antes de que confirme Ty.

En el caso del horario propuesto, se trata de un horario recuperable. Por un lado, la transacción T1 escribe el gránulo F, que después leerá T3, y T1 confirma antes que T3; por otro lado, la transacción T2 escribe los gránulos A y B, que serán leídos por T4, y T2 confirma antes que T4, y finalmente, T2 lee el gránulo E, que ha sido previamente escrito por T3, pero T2 confirma después que T3.

2. Como las transacciones se ejecutan utilizando el nivel de aislamiento `SERIALIZABLE`, no se producirá ninguna interferencia entre ellas. Por lo tanto, la ejecución de nuestras tres transacciones tiene que dar el mismo resultado que el de alguna de las ejecuciones en serie. Las ejecuciones en serie que podemos tener son las siguientes:

- **T1;** T2; T3
- **T1**, T3; T2
- T2, **T1;** T3
- T3; **T1;** T2
- T2; T3; **T1**
- T3; T2; **T1**

En los dos primeros casos, la transacción T1 se ejecuta en primer lugar. Como las acciones de lectura de T1 se llevan a cabo todas seguidas, T1 lee los valores iniciales de Y antes de que se ejecuten los `UPDATE` de T2 y los `DELETE` de T3. Por lo tanto, la suma de los valores de Y leída por T1 será 2015.

En el tercer caso, la transacción T1 se ejecuta después de la transacción T2, es decir, después de que se lleven a cabo los `UPDATE`. Por lo tanto, la suma de los valores de Y leída por T1 será 2105.

En el cuarto caso, la transacción T1 se ejecuta después de que se lleve a cabo la transacción T3, es decir, después de que se ejecuten los `DELETE`. Por lo tanto, la suma de los valores de Y leída por T1 será 1965.

En el quinto caso, la transacción T1 se ejecuta después de que se lleven a cabo T2 y T3, es decir, después de que se ejecuten los `UPDATE` y los `DELETE`. Por lo tanto, la suma de los valores de Y leída por T1 será 2005.

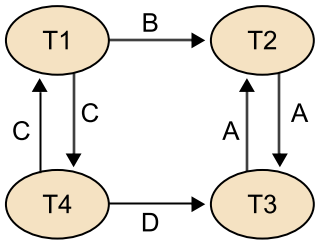
Finalmente, en el sexto caso, la transacción T1 se ejecuta después de que se ejecuten T3 y T2, es decir, después de la ejecución de los `DELETE` y los `UPDATE`. Por lo tanto, la suma de los valores de Y leída por T1 será 2005.

Resumiendo, los valores leídos por la transacción T1, teniendo en cuenta que las transacciones se ejecutan utilizando el nivel de aislamiento `SERIALIZABLE`, son los siguientes:

Orden de ejecución de las transacciones	Suma valores Y que recupera T1
T1; T2; T3	2015
T1, T3; T2	2015
T2, **T1;** T3	2105
T3; **T1;** T2	1965
T2; T3; **T1**	2005
T3; T2; **T1**	2005

3.

a) El grafo de precedencias asociado al horario es:



Se dan las siguientes interferencias:

- Lectura no repetible entre las transacciones T2 y T3 sobre el gránulo A, acciones 4-7-12.
- Actualización perdida entre las transacciones T1 y T4 sobre el gránulo C, acciones 8-9-10-11.

b) Como hemos visto en el punto a), se producen interferencias y, por lo tanto, no se trata de un horario seriable.

Con respecto a la recuperabilidad, un horario es recuperable cuando ninguna transacción Tx que lee o escribe un gránulo escrito por otra transacción Ty confirma antes de que confirme Ty.

En el caso del horario propuesto, se trata de un horario recuperable. Por un lado, la transacción T1 escribe el gránulo C, previamente escrito por la transacción T4, pero T1 no confirma antes que T4; por otro lado, la transacción T2 lee los gránulos A y B, escritos previamente por las transacciones T3 y T1, y T2 no confirma antes de que lo hagan éstas.

c) Con el modo de aislamiento `READ UNCOMMITTED` no se hacen reservas de lectura, y las de escritura se mantienen hasta el final de la transacción.

N.° acción	T1	T2	T3	T4
1	L(B,X)			
2	RU(B)			
3	W(B)			
4				R(D)
5		R(A)		
6		R(B)		

N.° acción	T1	T2	T3	T4
7			L(A,X)	
8			RU(A)	
9			W(A)	
10				L(C,X)
11				RU(C)
12	L(C,X)			
13				W(C)
14		R(A)		
15			L(D,X)	
16			RU(D)	
17			W(D)	
18				COMMIT(U(C))
19	RU(C)			
20	W(C)			
21			COMMIT (U(A), U(D))	
22	COMMIT (U(B), U(C))			
23		COMMIT		

En sombreado se muestran los bloqueos de las transacciones que no pueden conseguir reserva.

Como podemos ver, la interferencia de lectura no repetible se continúa produciendo; por lo tanto, el horario no es seriable y no hay ningún horario en serie equivalente.

En el caso del modo de aislamiento `SERIALIZABLE`, tanto las reservas de lectura como las de escritura se mantienen hasta el final de la transacción.

N.° acción	T1	T2	T3	T4
1	L(B,X)			
2	RU(B)			
3	W(B)			
4				L(D,S)
5				R(D)
6		L(A,S)		
7		R(A)		
8		L(B,S)		
9			L(A,X)	

N.° acción	T1	T2	T3	T4
10				L(C,X)
11				RU(C)
12	L(C,X)			
13				W(C)
14				COMMIT (U(D), U(C))
15	RU(C)			
16	W(C)			
17	COMMIT (U(B), U(C))			
18		R(B)		
19		R(A)		
20		COMMIT (U(A), U(B))		
21			RU(A)	
22			W(A)	
23			L(D,X)	
24			RU(D)	
25			W(D)	
26			COMMIT (U(A), U(D))	

En sombreado se muestran los bloqueos de las transacciones que no pueden conseguir reserva.

En este caso, el nivel de aislamiento garantiza que no se produce ningún tipo de interferencia (el horario es seriable y recuperable). El horario en serie equivalente es T4; T1; T2; T3.

4.

a) Será necesario recorrer hacia atrás el dietario para deshacer los cambios de las transacciones no confirmadas. Es decir, se tendrá que recorrer hacia atrás el dietario para encontrar todos los registros de modificación de las transacciones activas desde el último punto de control. En nuestro caso, será necesario deshacer la acción que T2 ha hecho en el instante 8. De T3 no se tendrá que deshacer nada, porque sus cambios no están en la memoria externa (el primer cambio de T3 se produce después del último punto de control). Además, será necesario rehacer los cambios de las transacciones confirmadas. Por lo tanto, se tendrá que recorrer hacia adelante el dietario y buscar los registros de cambio de las transacciones confirmadas desde el último punto de control. En nuestro caso, se tendrá que rehacer la acción que T4 ha hecho en el instante 14.

b) Con esta política no hace falta rehacer los cambios de las transacciones confirmadas. Por lo tanto, el cambio con respecto al apartado a) es que no será necesario rehacer la acción de T4 en el instante 14.

5. No, aunque pueda parecer que el dietario no es necesario en el SGBD descrito, hemos de recordar que el dietario no sólo se utiliza para restaurar la BD, sino que también se usa para poderla reconstruir; por ejemplo, en el caso de que el dispositivo de almacenamiento

externo que guarda la BD pierda los datos a causa de una avería o una catástrofe (incendio, inundación, etc.).

Glosario

abrazo mortal *m* Situación que se produce, usando reservas, cuando la ejecución de dos o más transacciones queda bloqueada, sin posibilidad de ser retomada, porque cada una de ellas espera a obtener una reserva que está en posesión de otra de las transacciones que hay implicadas.

ACID *f pl* Acrónimo formado por las palabras *atomicity*, *consistency*, *isolation* y *definitivity* (atomicidad, consistencia, aislamiento y definitividad) que indica las propiedades que toda transacción debe cumplir.

BD *f* Sigla correspondiente a *base de datos*.

cancelación de una transacción *f* Finalización de una transacción sin que se confirmen las actualizaciones hechas en la BD.

confirmación de una transacción *f* Finalización de una transacción que causa que los cambios realizados pasen a ser definitivos en la BD.

control de concurrencia *m* Conjunto de técnicas que utiliza un SGBD para evitar que se produzcan interferencias entre transacciones que se ejecutan de forma concurrente.

copia de seguridad *f* Copia de un estado correcto de una BD que se utiliza para reconstruir ésta en caso de pérdida total o parcial de los datos.

dietario *m* Estructura de datos utilizada por el SGBD para almacenar información de los cambios que han hecho las transacciones y cómo (las transacciones) han acabado su ejecución. Se utiliza en las tareas de recuperación.

gránulo *m* Unidad de datos controlada individualmente por el SGBD en relación con el control de concurrencia y la recuperación.

horario *m* Ejecución concurrente, en un cierto orden, de las acciones que incorporan un conjunto de transacciones.

interferencia *f* Comportamiento anómalo que puede producir el acceso concurrente de diversos usuarios a la BD si no se toman las precauciones adecuadas y que pone en peligro la integridad de la misma o hace que llegue información errónea a los usuarios.

nivel de aislamiento *m* Grado de protección que ofrece el SGBD a una transacción según los tipos de interferencias de los que la protege.

nivel de concurrencia *m* Grado de aprovechamiento de los recursos de proceso disponibles según el solapamiento de ejecución de las transacciones que acceden de forma concurrente a la BD y que consiguen confirmar.

protocolo de reservas en dos fases *m* Manera de utilizar reservas por parte de las transacciones que garantiza la seriabilidad de los horarios, pero no su recuperabilidad. En la primera fase permite a las transacciones adquirir reservas, y en la segunda, sólo liberarlas.

protocolo de reservas en dos fases estricto *m* Variante estricta del protocolo de reservas en dos fases en la que las reservas no se liberan hasta que no acaba la transacción. Garantiza, además de la seriabilidad, la recuperabilidad de los horarios.

PR2F *m* Protocolo de reservas en dos fases.

punto de control *m* Registro del dietario que identifica un instante en el que el SGBD escribe en memoria externa todos los cambios que se han hecho en las memorias intermedias y que contiene los identificadores de todas las transacciones que hay activas en ese instante.

reconstrucción *f* Conjunto de tareas necesarias para recuperar el último estado de una BD cuando, a causa de fallos o desastres, se produce una pérdida total o parcial de los datos guardados en la misma.

recuperabilidad *f* Criterio formal que define qué condiciones debe cumplir un horario para que las cancelaciones no provoquen interferencias.

recuperación *f* Conjunto de tareas necesarias para garantizar la atomicidad y la definitividad de las transacciones. Tiene la misión de conseguir que no se pierdan los cambios que hayan hecho las transacciones confirmadas y que se deshagan los que hayan hecho las transacciones que no confirman su ejecución.

reserva *f* Derecho que puede adquirir una transacción para acceder a un gránulo de la BD, que se pide antes de acceder al gránulo y se libera posteriormente. La reserva se pide en una cierta modalidad que permite hacer la acción que la transacción desea sobre el gránulo.

restauración *f* Conjunto de tareas necesarias para garantizar la atomicidad y la definitividad de las transacciones ante cancelaciones voluntarias o involuntarias y frente a caídas del SGBD.

seriabilidad *f* Criterio formal que define las condiciones que tiene que tener un horario para garantizar que las transacciones están correctamente aisladas entre ellas en el supuesto de que todas las transacciones confirmen sus resultados.

transacción *f* Secuencia de operaciones de lectura y actualización de la BD que cumple las propiedades ACID.

Bibliografía

Este módulo didáctico se basa parcialmente (con el consentimiento del autor) en el siguiente material:

Costa Vallés, P. (2004). "Transaccions en les bases de dades"; "Control de concurrència i recuperació". En: *Bases de dades II* (2.ª ed.). Barcelona: UOC.

Este material está publicado como capítulo de libro en la obra siguiente:

Sistac Planas, J. (coord). (2000). *Tècniques avançades de bases de dades*. Barcelona: EDIUOC (colección "Manuals", 38).

Además, también se ha utilizado la siguiente bibliografía:

Bernstein, P. A.; Hadzilacos, A.; Goodman, N. (1987). *Concurrency control and recovery in database systems*. Reading (Massachusetts): Addison Wesley.

Este libro actualmente está descatalogado, aunque los autores están preparando una nueva edición de la obra. Para muchos, es la mejor obra dedicada al tema que se trata en este módulo didáctico. Actualmente, el libro original se halla disponible en la siguiente dirección (último acceso: septiembre 2010):

http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx.

Gray, J.; Reuter, A. (1993). *Transaction processing. Concepts and techniques*. San Francisco: Morgan Kaufmann Publishers.

J. Gray ha sido unos de los investigadores e implementadores de referencia en el área de procesamiento de transacciones en SGBD. Por su trabajo, recibió el premio A. M. Turing en el año 1998. Éste es el galardón más prestigioso que se concede dentro de la ciencia informática.

Melton, J.; Simon, A. R. (2002). *SQL:1999. Understanding relational language components*. San Francisco: Morgan Kaufmann Publishers.

Este libro presenta la sintaxis del SQL estándar, concretamente la versión correspondiente al SQL:1999. En las versiones posteriores del SQL estándar no hay cambios en relación con las sentencias que permiten la gestión de transacciones.

Ramakrishnan, R.; Gehrke, J. (2003). *Database management systems* (3.ª ed.). Boston: McGraw-Hill.

Éste es uno de los otros libros de referencia en asignaturas sobre BD. Dedica tres capítulos a la gestión de transacciones.

The PostgreSQL Global Development Group (2009). *PostgreSQL 8.4.2 documentation*

Esta obra está disponible en la siguiente dirección (último acceso: septiembre 2010):

http://www.postgresql.org/