CC42A – BASES DE DATOS
Profesores: Claudio Gutiérrez, Gonzalo Navarro
Auxiliar: Mauricio Monsalve

Auxiliar 9

- **Repaso de materia**
- **Ejercicios para el control**

REPASO DE TRANSACCIONES

Transacciones

Las *transacciones* son operaciones en las bases de datos, ejecuciones mínimas, que cumplen con ser atómicas (se ejecutan por completo, no parcialmente, o bien no se ejecutan), consistentes con la base de datos, son aisladas entre sí (independientes en ejecución) y durables (sus efectos perduran en la base de datos). Estas operaciones se programan en SQL (begin tran –nombre- … exec –pedido- … rollback –nombre- … commit –nombre-). Para administrar las *transacciones* deben existir políticas de *bloqueo*. A una operación se le pueden hacer bloqueos de *lectura* y/o *escritura*. Los niveles de bloque llegan desde un atributo (campo), pasando por filas, columnas, hasta bloquear relaciones (tablas).

Tipos de bloqueo:

- Compartido: Sólo lectura. Se pueden leer por varios a la vez, pero no actualizar.
- Actualización: Permite escribir. Sólo una transacción tiene acceso a la escritura en este elemento. Se convierte en exclusivo si una transacción trata de escribir o en compartido si se trata de leer.
- Exclusivo: Una y sólo una transacción opera en el elemento, escribiendo datos.
- Intención: Establece jerarquía de bloqueo. Ejemplo: Varias transacciones piden bloqueo compartido y un pide exclusivo, el resultado es bloqueo compartido, por mayoría.
- Actualización masiva (el nombre lo dice).
- Modificación de datos: La relación completa se bloquea. El esquema va a cambiar.

Control de concurrencia: Problema de espera infinita por bloqueo mutuo entre transacciones (deadlock) y cada una espera a la otra; solución: Dar prioridad, permitir y revisar si hay deadlocks, concesión simultánea. Política FIFS (primero en llegar, primero en servir) para solucionar livelock (espera infinita porque otras transacciones obtienen primero el derecho).
Planificadores: Debe haber planificadores para conflictos de acceso. Puede permitir esperas indefinidas o aborto y reinicio de transacciones.
Asumir escritura: Operaciones LOCK y UNLOCK.
Serializabilidad: Secuenciabilidad. ¿Cuándo una planificación es secuenciable? Algoritmo: Dibujar grafo, dibujar interbloqueos como arcos. Si hay ciclos, la planificación no es secuenciable.

Algoritmo específico de revisión si un plan S es serializable

(1) Para cada transacción T_i en S, crear un nodo en el grafo G.
(2) Para cada Leer(X) antes de Escribir(X), *y viceversa*, en transacciones T_i a T_j, agregar un arco de T_i a T_j en G: $T_i \rightarrow T_j$.
(3) Para cada Escribir(X) antes de Escribir(X) en transacciones T_i a T_j, agregar arco $T_i \rightarrow T_j$ a G.
(4) El plan S será serializable (o secuenciable) si no tiene ciclos. Además, el camino indicado por los arcos **es** el plan a seguir.

Protocolo de dos fases: Implementación de transacciones que indica que todo bloqueo (LOCK) aparece antes de un dsebloqueo (UNLOCK). Propiedad: Secuenciable. Problema: Existencia de deadlocks.
Protocolos basados en grafos: Ej. basado en árboles, con prioridad de elementos de bloqueo (celda, fila, columna, tabla, etc.).
Control de fallos: Si se llega a COMMIT, no hay fallo. Si no, pueden haber varios motivos de fallos (deadlocks, fallo aritmético, aborto por usuario, error de software/hardware). La caída puede ser en memoria volátil o en memoria permanente. **Solución con registro (LOG).** Solución sencilla y eficiente. Checkpointing: Cuando actualizar operaciones, cuando los cambios serán permanentes. Actualización inmediata y diferida (cuando se llega a COMMIT, todo primero al log). Shadow paging, backups de páginas (número predefinido en la BD).

EJERCICIOS RESUELTOS

Ejercicios para el control

1) Cuando se hace un checkpoint sobre un log con updates diferidos no debe permitirse que ninguna transacción se esté ejecutando, mientras que si se usa log con updates inmediatos no existe este problema. Explique por qué (C3, año 2003-1).

El log con updates diferidos realiza los cambios una vez que estos son seguros sobre la base de datos. Por esto mismo, **no hay rollback, guarda sólo cuando se llega a COMMIT**. Entonces, la actualización del log se debe realizar cuando no hay transacciones en ejecución.
→ **Otra cosa a saber:** ***¿Qué se debe hacer en el caso de bases de datos distribuidas?*** (con distintos repositorios de datos) En este caso debe haber un log para cada sitio/repositorio y un log que indica el logging distribuido (log cabecera).

2) Indique lo que ocurre en el log con las siguientes transacciones T1 y T2: R1(X), W1(X), R2(Z), R2(Y), W2(Z), R1(Y), W1(Y), C1, C2. Hágalo para logging inmediato y diferido.

En el log de actualización inmediata se ven las operaciones en el orden anteriormente indicado mientras que en el log de actualización diferida las operaciones de T1 están primero porque T1 llega a COMMIT primero que T2.

Ejercicios de cálculo de costos: Ver guía.