Bases de Datos I

CLASE 7

Transacciones

Entornos Concurrentes

Río Negro
Universidad Nacional

Entornos concurrentes

- Entorno centralizado
  - Varias transacciones ejecutándose simultáneamente compartiendo recursos.
  - Deben evitarse los mismos problemas de consistencia de datos
  - Transacciones correctas, en ambientes concurrente pueden llevar a fallos
- Seriabilidad
  - Garantiza la consistencia de la BD

Entornos concurrentes

```
T0   READ( a )          T1  READ( a )
     a := a – 50             temp := a * 0.1
     WRITE( a )              a := a – temp
     READ( b )               WRITE( a )
     b := b + 50             READ( b )
     WRITE( b )              b := b + temp
                             WRITE( b )
```

- ✓ Resolver T0, T1 o T1, T0 se respeta A+B
- ✓ Ahora bien T0 T1 <> T1 T0

- **Planificación:** secuencia de ejecución de transacciones

Entornos concurrentes

- Involucra todas las instrucciones de las transacciones
- Conservan el orden de ejecución de las mismas
- Un conjunto de m transacciones generan m! planificaciones en serie
- La ejecución concurrente no necesita una planificación en serie.

Entornos concurrentes

```
READ(A)
A := A – 50
WRITE(A)

                READ(A)
                TEMP := A *  0.1
                A := A – TEMP
                WRITE(A)

READ(B)
B := B + 50
WRITE(B)

                READ(B)
                B := B + TEMP
                WRITE(B)
```

A + B se conserva

```
READ(A)
A := A – 50

                READ(A)
                TEMP := A *  0.1
                A := A – TEMP
                WRITE(A)

                READ(B)

WRITE(A)
READ(B)
B := B + 50
WRITE(B)

                B := B + TEMP
                WRITE(B)
```

A + B no se conserva

5

Entornos concurrentes

- Conclusiones
  - El programa debe conservar la consistencia
  - La inconsistencia temporal puede ser causa de inconsistencia en planificaciones en paralelo
  - Una planificación concurrente debe equivaler a una planificación en serie
  - Solo las instrucciones READ y WRITE son importantes y deben considerarse.

Entornos concurrentes

- Conflicto en planificaciones serializables
  - I1, I2 instrucciones de T1 y T2
    - Si operan sobre datos distintos. NO hay conflicto.
    - Si operan sobre el mismo dato
      - ✓ I1 = READ(Q) = I2, no importa el orden de ejecución
      - ✓ I1 = READ(Q), I2 = WRITE(Q) depende del orden de ejecución (I1 leerá valores distintos)
      - ✓ I1 = WRITE(Q), I2 = READ(Q) depende del orden de ejecución (I2 leerá valores distintos)
      - ✓ I1 = WRITE(Q) = I2, depende el estado final de la BD
  - I1, I2 está en conflicto si actúan sobre el mismo dato y al menos una es un write.

Entornos concurrentes

- Una Planificación S se transforma en una S´ mediante intercambios de instrucciones no conflictivas, entonces S y S´ son *equivalentes en cuanto a conflictos.*
- S´ es serializable en conflictos si existe S/ son equivalentes en cuanto a conflictos y S es planificable serie.

- Pruebas de seriabilidad
  - Algoritmo para determinar seriabilidad de conflictos: grafo dirigido (grafo de precedencia)

Entornos concurrentes

- Conjunto de vértices (transacciones de la planificación)
- Cto de aristas (Ti → Tj /
  - Ti ejecuta un write(q) antes que Tj un read(q)
  - Ti ejecuta un write(q) antes que Tj un write(q)
  - Ti ejecuta un read(q) antes que Tj un write(q)
- Si el grafo tiene ciclos la planificación no es serializable en conflictos.

Control de Concurrencia

- Métodos de control de concurrencia
  - Bloqueo
  - Basado en hora de entrada
- Bloqueo
  - Compartido *Lock_c*(dato)(solo lectura)
  - Exclusivo *Lock_e*(dato) (lectura/escritura)
  - Las transacciones piden lo que necesitan.
  - Los bloqueos pueden ser compatibles y existir simultáneamente (compartidos)

Control de Concurrencia

- Una transacción debe:
  - ✓ Obtener el dato (si está libre, o compartido y solicita compartido)
  - ✓ Si no lo puede obtener: Esperar (o abortar)
  - ✓ Usar el dato
  - ✓ Liberarlo (commit o rollback /abort)

Control de Concurrencia

- Puede ocurrir DEADLOCK

Lock_e(b)	
Read(b)	
b := b + 50	
	lock_c(a)
	read(a)
	lock_c(b)
lock_e(a)	
…..	….
commit	commit

- Ninguna libera, cada una espera por un dato que tiene bloqueado la otra
- Una de las dos debe retroceder, liberando sus datos.

Control de Concurrencia

- Conclusiones:
  - Si lo datos se liberan pronto → se evitan posibles deadlock
  - Si los datos se mantienen bloqueados → se evita inconsistencia.

- Protocolos de bloqueos
  - Indica si cada transacción puede bloquear y liberar c/u de sus datos.
  - Sea {T0, …, Tn} transacciones de la planificación S. Ti precede a Tj si existe Q dato / Ti bloqueo (A) de Q y Tj posteriormente otro bloqueo (B) en Q y Comp(A,B) = Falso (no compatibles).

Control de Concurrencia

- Protocolos de bloqueo
  - Dos fases
    - Requiere que las transacciones hagan bloqueos en dos fases:
      - Crecimiento: se obtienen datos
      - Decrecimiento: se liberan los datos
    - Garantiza seriabilidad en conflictos, pero no evita situaciones de deadlock.
    - Como se consideran operaciones
      - Fase crecimiento: se piden bloqueos en orden: compartido, exclusivo
      - Fase decrecimiento: se liberan datos o se pasa de exclusivo a compartido.

Control de Concurrencia

- Protocolo basado en hora de entrada
  - El orden de ejecución se determina por adelantado, no depende de quien llega primero
  - C/transacción recibe una HDE
    - Hora del servidor
    - Un contador
  - Si HDE(Ti) < HDE(Tj), Ti es anterior
  - C/Dato
    - Hora en que se ejecutó el último WRITE
    - Hora en que se ejecutó el último READ

Control de Concurrencia

- Las operaciones READ y WRITE que pueden entrar en conflicto se ejecutan y eventualmente fallan por HDE.
- Algoritmo de ejecución:
  - Ti Solicita READ(Q)
    - HDE(Ti) < HW(Q): rechazo (solicita un dato que fue escrito por una transacción posterior)
    - HDE(Ti) ≥ HW(Q): ejecuta y se establece HR(Q)=Max{HDE(Ti), HR(Q)}

Control de Concurrencia

- Ti solicita WRITE(Q)
  - HDE(Ti) < HR(Q): rechazo (Q fue utilizado por otra transacción anteriormente y supuso que no cambiaba)
  - HDE(Ti) < HW(Q): rechazo (se intenta escribir un valor viejo, obsoleto)
  - HDE(Ti) > [HW(Q) y HR(Q)]: ejecuta y HW(Q) se establece con HDE(Ti).
- Si Ti falla, y se rechaza entonces puede recomenzar con una nueva hora de entrada.

Control de Concurrencia

- Casos de Concurrencia. Granularidad
  - A registros caso más normal
  - Otros casos
    - BD completa
    - Áreas
    - Tablas
- Otras operaciones conflictivas
  - Delete(Q) requiere un uso completo del registro
  - Insert(Q) el dato permanece bloqueado hasta la operación finalice.

Recuperación en caso de fallos

- Consideraciones del protocolo basado en bitácora
  - Existe un único buffer de datos compartidos y uno para la bitácora
  - C/transacción tiene un área donde lleva sus datos
  - Una transacción que falla por bloqueos y aborta, debe dejar la db consistente: undo
  - El retroceso de una transacción puede llevar al retroceso de otras transacciones

- Retroceso en cascada
  - Falla una transacción → pueden llevar a abortar otras
  - Puede llevar a deshacer gran cantidad de trabajo.

Recuperación en caso de fallos

- Retroceso en cascada. Ejemplo
  - T1 opera sobre datos 1, 2, 3, 4.
  - T1 obtiene 1, lo actualiza y libera.
  - T2 obtiene 1, lo utiliza y finaliza.
  - T1 falla, aborta y deshace todo lo hecho
  - T2 usó un dato modificado por T1, pero al deshacer T1 ese valor no es correcto. T2 debe fallar. Pero T2 esta finalizada, y por durabilidad no puede abortarse.

- Solución:
  - Condición: una transacción Tj no puede finalizar si una transacción Ti anterior utiliza datos que Tj necesita y Ti no está finalizada.

Recuperación en caso de fallos

- Puede ocurrir que falle Ti, y que Tj deba retrocederse, pero que Tj ya terminó. Como actuar?
  - Protocolo de bloqueo de dos fases: los bloqueos exclusivos deben conservarse hasta que Ti termine.
  - HDE, agrega un bit, para escribir el dato, además de lo analizado, revisar el bit si está en 0 proceder, si está en 1 la transacción anterior no termino, esperar....

Recuperación en caso de fallos

- Bitácora
  - Idem sistemas monousuarios
  - Como proceder con checkpoint
    - Colocarlo cuando ninguna transacción esté activa. Puede que no exista el momento.
    - Checkpoint<L> L lista de transacciones activa al momento del checkpoint.
  - Ante un fallo
    - UNDO y REDO según el caso.
    - Debemos buscar antes del Checkpoint solo aquellas transacciones que estén en la lista.

Interbloqueos

- Como evitar los deadlock (interbloqueo)
  - Prevenirlos (evitarlos)
  - Detectarlos (recuperarlos)
- Prevención
  - Tomar todos los datos que se necesitan
    - ✓ Si hay éxito la transacción prosigue
    - ✓ No hay éxito la transacción espera
      - Posible inanición (se puede mejorar con prioridades)
    - ✓ Ordenar los datos parcialmente, se obtienen en orden o nada
    - ✓ HDE puede manejar prioridades para evitar inanición.

Interbloqueos

- Detección: Algoritmo
  - Detecta el bloqueo
    - ✓ Genera un grafo de pedidos de datos, si encuentra ciclo → deadlock
  - Corrige: Selección de la víctima
    - ✓ Elección: costo mínimo
    - ✓ Retroceder hasta donde?
    - ✓ Evitar inanición de la transacción retrocedida.

Bloqueo indefinido (inanición)

T0	T1	T2	T3	T4…….
lock_c(a)				
Read(a)				
	lock_e(a) (W)			
		lock_c(a)		
		read(a)		
Commit;				
			lock_c(a)	
			read(a)	
		commit;		
				lock_c(a)
				read(a)
			commit;	…….

Seguridad e Integridad

- Integridad: protección ante pérdidas accidentales de consistencia
  - ✓ Problemas durante el procesamiento de transacciones.
  - ✓ Control de concurrencia
  - ✓ Anomalías causadas por la distribución de datos sobre varias computadoras
  - ✓ Error lógico en una transacción que viola protecciones de inconsistencia

- Seguridad: protección contra intentos mal intencionados para modificar datos
  - ✓ Nivel físico
  - ✓ Nivel humano
  - ✓ Nivel SO
  - ✓ Red
  - ✓ DBMS

Seguridad e Integridad

- Nivel de seguridad físico
  - Protección del equipo ante problemas naturales, fallo de energía, etc.
  - Protección del DR contra robos, borrados, daños físicos
  - Protección de la red contra daños físicos
  - Soluciones
    - Replicar el hardware (discos espejos, múltiples accesos a la red (varios cables))
    - Seguridad física
    - Técnicas de software que aseguren posibles brechas de seguridad

Seguridad e Integridad

- Nivel de seguridad Humano
  - Protegerse ante robo de password. Distintas políticas
- Nivel de seguridad de SO
  - Protección contra logins inválidos
  - Protección de acceso a nivel de archivos
- Nivel de seguridad de Red
  - Cada sitio debe asegurar que se comunica con sitios autorizados
  - Los links deben protegerse contra robos y modificación de mensajes
  - Mecanismos de identificación y cifrado de mensajes.

Seguridad e Integridad

- Nivel de BD
  - Asumir la seguridad en todos los niveles anteriores
  - Usos específicos de la BD
    - Las autorizaciones de usuarios pueden ser sobre archivos, relaciones, o parte de estos.
    - Cada usuario debe tener su autorización para leer y/o escribir solo parte de los datos
  - Se debe asegurar autonomía local en BDD.
  - Control global sugiere control centralizado.

Seguridad e Integridad

- Seguridad tema del DBA a nivel BD
  - Autorizaciones a usuarios
  - Vistas
- Cifrado de datos
  - “ocultar” datos para que no sean visibles
  - Protocolos
    - Clave pública
    - Clave privada

Fin Clase 7

Río Negro
Universidad Nacional