

# Bases de Datos

Clase 17: Recuperación de Fallas

# Recuperación de Fallas

**Log y Recovery Manager** se encargan de asegurar Atomicity y Durability

# ¿Pero qué puede salir mal?

Fallas en la ejecución:

- Datos erróneos
  - Solución: restricciones de integridad, data cleaning
- Fallas en el disco duro
  - Solución: RAID, copias redundantes

# ¿Pero qué puede salir mal?

Fallas en la ejecución:

- Catástrofes
  - Solución: copias distribuidas
- Fallas del sistema
  - Solución: **Log y Recovery Manager**

# Log Manager

Una página se va llenando secuencialmente con *logs*

Cuando la página se llena, se almacena en disco

Todas las transacciones escriben el *log* de manera concurrente

# Log Manager

Registra todas las acciones de las transacciones

# Log Records

Los *logs* comunes son:

- **<START T>**
- **<COMMIT T>**
- **<ABORT T>**
- **<T UPDATE>**

¿Cómo los usamos?

# Undo Logging

Forma de escribir los *logs* para poder hacer *recovery* del sistema

Idea: cuando todo falle, echamos pie atrás



# Undo Logging

Los *logs* son:

- $\langle \text{START } T \rangle$
- $\langle \text{COMMIT } T \rangle$
- $\langle \text{ABORT } T \rangle$
- $\langle T, X, t \rangle$  donde  $t$  es el valor **antiguo** de  $X$

# Undo Logging

Regla 1: si **T** modifica  $X$ , el *log*  $\langle \mathbf{T}, X, t \rangle$  debe ser escrito antes que el valor  $X$  sea escrito en disco

Regla 2: si **T** hace *commit*, el log  $\langle \mathbf{COMMIT T} \rangle$  debe ser escrito justo después de que todos los datos modificados por **T** estén almacenados en disco

# Undo Logging

En resumen:

- Escribir el log  $\langle \mathbf{T}, X, t \rangle$
- Escribir los datos a disco
- Escribir  $\langle \mathbf{COMMIT T} \rangle$
- Hacer *flush* a disco del *log*

# Recuperación con Undo Logging

Detectando fallas en el *log*:

... <START T> ... <COMMIT T> ...



# Recuperación con Undo Logging

Detectando fallas en el *log*:

... <START T> ... <ABORT T> ...



# Recuperación con Undo Logging

Detectando fallas en el *log*:

... <START T> ...



# Recuperación con Undo Logging

Supongamos que mientras usamos nuestro sistema, se apagó de forma imprevista

Leyendo el *log* podemos hacer que la base de datos quede en un estado consistente

# Recovery

Algoritmo para un *Undo Logging*

Procesamos el log desde el final hasta el principio:

- Si leo **<COMMIT T>**, marco **T** como realizada
- Si leo **<ABORT T>**, marco **T** como realizada
- Si leo **<T, X, t>**, debo restituir  $X := t$  en disco, si no fue realizada.
- Si leo **<START T>**, lo ignoro



# Recovery

Algoritmo para un *Undo Logging*

- ¿Hasta dónde tenemos que leer el *log*?
- ¿Qué pasa si el sistema falla en plena recuperación?
- ¿Cómo trucamos el *log*?

# Recovery

Uso de *Checkpoints*

Utilizamos *checkpoints* para no tener que leer el *log* entero y para manejar las fallas mientras se hace *recovery*

# Recovery

Uso de *Checkpoints*

- Dejamos de escribir transacciones
- Esperamos a que las transacciones actuales terminen
- Se guarda el *log* en disco
- Escribimos <**CKPT**> y se guarda en disco
- Se reanudan las transacciones

# Recovery

Uso de *Checkpoints*

Ahora hacemos *recovery* hasta leer un <CKPT>

**Problema:** es prácticamente necesario apagar el sistema para guardar un *checkpoint*

# Recovery

Uso de *Nonquiescent Checkpoints*

**Nonquiescent Checkpoints** son un tipo de *checkpoint* que no requiere "apagar" el sistema

# Recovery

Uso de *Nonquiescent Checkpoints*

- Escribimos un *log* **<START CKPT (T<sub>1</sub>, ..., T<sub>n</sub>)>**, donde T<sub>1</sub>, ..., T<sub>n</sub> son transacciones activas
- Esperamos hasta que T<sub>1</sub>, ..., T<sub>n</sub> terminen, sin restringir nuevas transacciones
- Cuando T<sub>1</sub>, ..., T<sub>n</sub> hayan terminado, escribimos **<END CKPT>**

# Undo Recovery

Uso de *Nonquiescent Checkpoints*

- Avanzamos desde el final al inicio
- Si encontramos un **<END CKPT>**, hacemos *undo* de todo lo que haya después del inicio del *checkpoint*
- Si encontramos un **<START CKPT (T<sub>1</sub>, ..., T<sub>n</sub>)>** sin su **<END CKPT>**, debemos analizar el log desde el inicio de la transacción más antigua entre T<sub>1</sub>, ..., T<sub>n</sub>

# Ejemplo

Uso de *Checkpoints* en *Undo Logging*

Considere este *log* después de una falla:

| Log                   |
|-----------------------|
| <START T1>            |
| <T1, a, 5>            |
| <START T2>            |
| <T2, b, 10>           |
| <START CKPT (T1, T2)> |
| <T2, c, 15>           |
| <START T3>            |
| <T1, d, 20>           |
| <COMMIT T1>           |
| <T3, e, 25>           |
| <COMMIT T2>           |
| <END CKPT>            |



# Ejemplo

Uso de *Checkpoints* en *Undo Logging*

Ahora considere este *log* después de una falla:

| Log                   |
|-----------------------|
| <START T1>            |
| <T1, a, 5>            |
| <START T2>            |
| <T2, b, 10>           |
| <START CKPT (T1, T2)> |
| <T2, c, 15>           |
| <START T3>            |
| <T1, d, 20>           |
| <COMMIT T1>           |
| <T3, e, 25>           |

# Undo Logging

**Problema:** no es posible hacer **COMMIT** antes de almacenar los datos en disco

Por lo tanto las transacciones se toman más tiempo en terminar!

# Redo Logging

Los *logs* son:

- $\langle \text{START } \mathbf{T} \rangle$
- $\langle \text{COMMIT } \mathbf{T} \rangle$
- $\langle \text{ABORT } \mathbf{T} \rangle$
- $\langle \mathbf{T}, X, v \rangle$  donde  $v$  es el valor **nuevo** de  $X$

# Redo Logging

Regla 1: Antes de modificar cualquier elemento *X* en disco, es necesario que todos los *logs* estén almacenados en disco, incluido el **COMMIT**

Esto es al revés respecto a *Undo Logging*

# Redo Logging

En resumen:

- Escribir el log  $\langle \mathbf{T}, X, v \rangle$
- Escribir  $\langle \mathbf{COMMIT T} \rangle$
- Hacer *flush* a disco del *log*
- Escribir los datos en disco

# Recovery

Algoritmo para un *Redo Logging*

Procesamos el *log* desde el principio hasta el final:

- Identificamos las transacciones que hicieron **COMMIT**
- Hacemos un *scan* desde el principio
- Si leo  $\langle \mathbf{T}, X, v \rangle$ :
  - Si **T** no hizo **COMMIT**, no hacer nada
  - Si **T** hizo **COMMIT**, reescribir con el valor **v**
- Para cada transacción incompleta, escribir **<ABORT T>**

# Recovery

Uso de *Checkpoints* en *Redo Logging*

¿Cómo utilizamos los checkpoints en el Redo Logging?

# Recovery

Uso de *Checkpoints* en *Redo Logging*

- Escribimos un *log* **<START CKPT (T<sub>1</sub>, ..., T<sub>n</sub>)>**, donde T<sub>1</sub>, ..., T<sub>n</sub> son transacciones activas y sin **COMMIT**
- Guardar en disco todo lo que haya hecho **COMMIT** hasta ese punto
- Una vez hecho, escribir **<END CKPT>**



# Redo Recovery

Uso de *Checkpoints* en *Redo Logging*

- Revisar el *log* desde el final al inicio
- Si encontramos un **<END CKPT>**, debemos retroceder hasta su respectivo **<START CKPT (T<sub>1</sub>, ..., T<sub>n</sub>)>**, y comenzar a hacer *redo* desde la transacción más antigua entre T<sub>1</sub>, ..., T<sub>n</sub>
- No se hace *redo* de las transacciones con **COMMIT** antes del **<START CKPT (T<sub>1</sub>, ..., T<sub>n</sub>)>**

# Redo Recovery

Uso de *Checkpoints* en *Redo Logging*

Si encontramos un **<START CKPT (T<sub>1</sub>, ..., T<sub>n</sub>)>** sin su **<END CKPT>**, debemos retroceder hasta encontrar un **<END CKPT>**

# Ejemplo

Uso de *Checkpoints* en *Redo Logging*

Considere este *log* después de una falla:

| Log               |
|-------------------|
| <START T1>        |
| <T1, a, 5>        |
| <START T2>        |
| <COMMIT T1>       |
| <T2, b, 10>       |
| <START CKPT (T2)> |
| <T2, c, 15>       |
| <START T3>        |
| <T3, e, 25>       |
| <END CKPT>        |
| <COMMIT T2>       |
| <COMMIT T3>       |

# Redo Logging

**Problema:** no es posible ir grabando los valores de  $X$  en disco antes que termine la transacción

Por lo tanto se congestiona la escritura en disco!

# Undo/Redo Logging

Es la solución para obtener mayor performance que mezcla las estrategias anteriormente planteadas

# Técnicas de Logging

## Resumen

Undo

Redo

Trans. Incompletas

Cancelarlas

Ignorarlas

Trans. Comiteadas

Ignorarlas

Repetirlas

Escribir **COMMIT**

Después de almacenar en disco

Antes de almacenar en disco

**UPDATE** *Log Record*

Valores antiguos

Valores nuevos