RECUPERACIÓN ANTE FALLAS EN BASES DE DATOS

Materia: Base de Datos

Cuatrimestre: 1C2015

Docente: Alejandro Eidelsztein

CONCEPTOS:

- Tolerancia a Fallas
- System Failures o Crashes (1)
- Resiliencia

(1) No veremos otros tipos de fallas como *transaction* y *media failures*

LOGGING/RECOVERY:

- El LOG registra la historia de los cambios efectuados a la BD

- Logging
  (Alimentación del LOG)

- Recovery
  (Proceso de Recuperación)
  (La operación de Recovery debe ser Idempotente)

- Estilos de Logging (1):
  UNDO Logging
  REDO Logging
  UNDO/REDO Logging

- Log Manager

- Recovery Manager

(1) Algunos autores se refieren a esta clasificación como: Steal/Force, No-Steal/No-Force y Steal/No-Force respectivamente.

COMPONENTES:

Query Processor

Transaction Manager

Log Manager

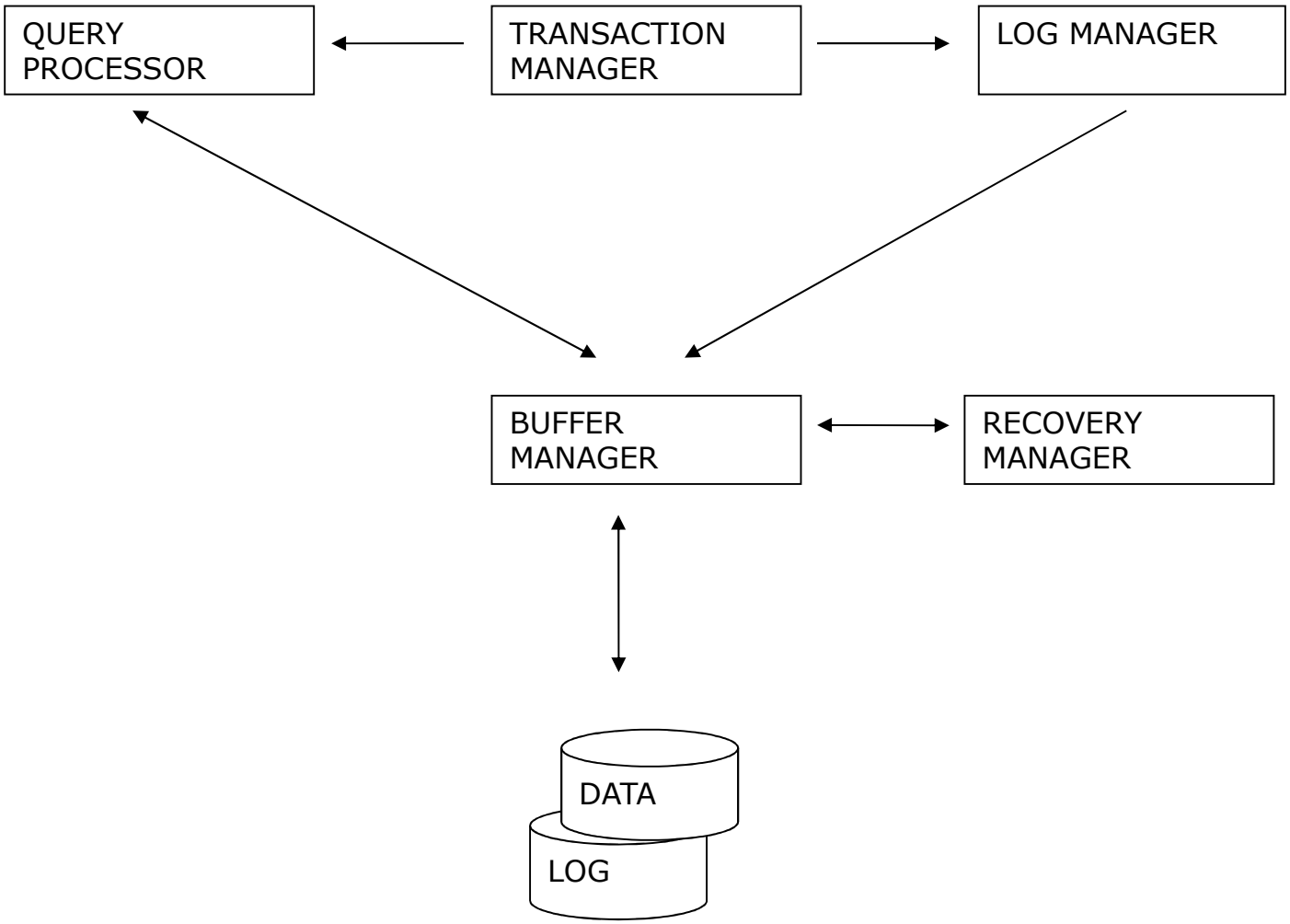
Buffer Manager

Recovery Manager

Data

Log

INTERACCION DE COMPONENTES



ESPACIOS DE MEMORIA:

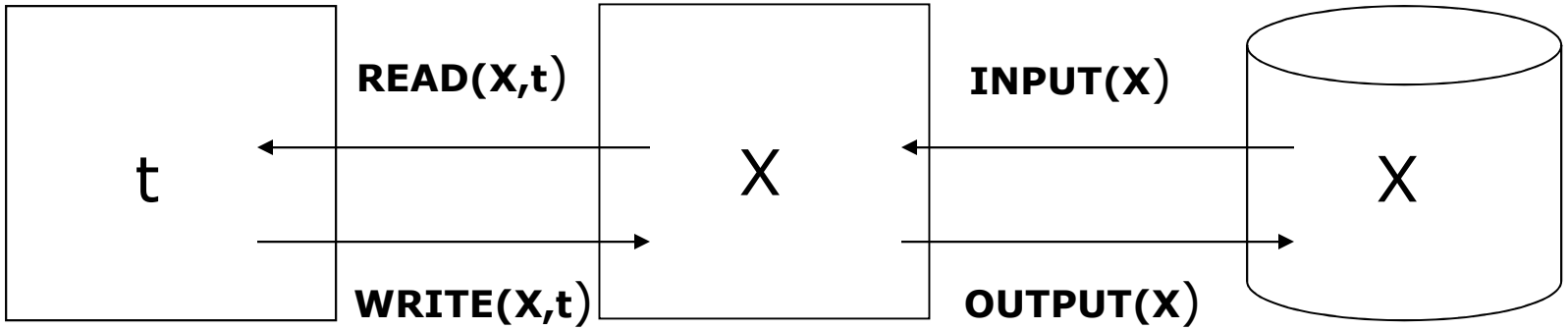
Local de la Transacción

Pool de Buffers

Bloques de Disco

OPERACIONES PRIMITIVAS Y ESPACIOS DE MEMORIA

- **X: ITEM**
- **t: VARIABLE LOCAL**



MEM. LOCAL DE LA TRANS. **POOL DE BUFFERS** **BLOQUES DE DISCO DE LA BD**

OPERACIONES PRIMITIVAS:

INPUT(X)
Copia el bloque que contiene a X del Disco al Pool de Buffers

READ(X, t)
Copia X a la variable local t de la transacción. Si el bloque que contiene X no está en el Pool de Buffers, primero ejecuta INPUT(X) y luego asigna el valor de X la a variable local t.

WRITE(X, t)
Copia el valor de la variable local t a X en el pool de buffers. Si el bloque que contiene X no está en le Pool de Buffers, primero ejecuta INPUT de X y luego copia el valor de t a X en el Pool de Buffers.

OUTPUT(X)
Copia del bloque conteniendo X desde el Pool de Buffers a Disco.

FLUSH LOG
Comando emitido por el Log Manager para que el Buffers Manager fuerce los Log Records a Disco.

Se asume que los ITEMS no exceden un bloque.

Los READS y los WRITES son emitidos por las transacciones.

Los INPUTS y los OUTPUTS son emitidos por el Buffer Manager.

LOG:

- Archivo compuesto por Log Records manejado por el Log Manager. En el mismo se registran ("apendean") las acciones *importantes* de las transacciones.

- Los Log Records, al igual que los Items, inicialmente son creados en el Pool de Buffers y son ubicados por el Buffer Manager.

- El comando Flush-Log ordena copiar los bloques que contienen los Log Records a Disco.

- El Log es una archivo del tipo Append-Only.

- Las transacciones se ejecutan concurrentemente y por lo tanto los Logs Records se graban en forma intercalada en el Log.

LOG RECORDS:

START LOG RECORD:

<START T> Indica que la transacción T ha empezado

COMMIT LOG RECORD:

<COMMIT T> Indica que la transacción T a completado exitosamente. Los cambios hechos por T deberían estar en Disco.

ABORT LOG RECORD:

<ABORT T> Indica que la transacción T podría no haber completado exitosamente. Los cambios realizados por T no deben aparecer en Disco.

UPDATE LOG RECORD (distinto para cada estilo de logging):

<T, X, v>	UNDO Logging
<T, X, w>	REDO Logging
<T, X, v, w>	UNDO/REDO Logging

ESTILOS DE LOGGING:

UNDO Logging

REDO Logging

UNDO/REDO Logging

UNDO LOGGING:

EN UNDO LOGGING EL RECOVERY MANAGER RESTAURA LOS VALORES ANTERIORES DE LOS ITEMS.

UPDATE LOG RECORD PARA UNDO-LOGGING:

ES DE LA FORMA: **<T, X, v>**

SIGNIFICA QUE: **LA TRANSACCIÓN T HA CAMBIADO EL VALOR DEL ITEM X CUYO VALOR ANTERIOR ERA v**

REGLAS DEL UNDO-LOGGING:

U_1: SI T MODIFICA X, LUEGO EL LOG RECORD <T, X, v> DEBE GRABARSE EN DISCO ANTES DE QUE EL NUEVO VALOR DE X SEA GRABADO EN DISCO.

U_2: SI T "COMITEA", ENTONCES SU COMMIT LOG RECORD <COMMIT T> DEBE SER GRABADO EN DISCO SOLO DESPUES DE QUE TODOS LOS ITEMS CAMBIADOS POR T HAN SIDO GRABADOS EN DISCO (Y TAN PRONTO COMO SEA POSIBLE)

DE U_1 Y U_2 SE DESPRENDE QUE LOS ELEMENTOS ASOCIADOS A T DEBEN SER GRABADOS EN DISCO EN EL SIGUIENTE **ORDEN**:

A. LOS LOG RECORDS INDICANDO LOS CAMBIOS A LOS ITEMS
B. LOS ITEMS CON SUS NUEVOS VALORES
C. EL COMMIT LOG RECORD

EJEMPLO (UNDO LOGGING):

T= READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

PASO	ACCION	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A, t) (1)	8	8		8	8	
3	t:= t*2	16	8		8	8	
4	WRITE(A, t)	16	16		8	8	<T, A, 8>
5	READ(B,t) (1)	8	16	8	8	8	
6	t:= t*2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 8>
8	FLUSH LOG						
9	OUTPUT(A)	16	16	16	16	8	
10	OUTPUT(B)	16	16	16	16	16	
11							<COMMIT T>
12	FLUSH LOG						

(1) Si no se encontrara el item A en el pool de buffers se ejecutará un INPUT(A) para traerlo de disco. Idem para el item B.

RECUPERACION USANDO UNDO LOGGING:

LAS T INCOMPLETAS DEBEN SER DESHECHAS

SI OCURRE UNA FALLA DEL SISTEMA EL RECOVERY MANAGER USA EL LOG PARA RESTAURAR LA BD A UN ESTADO CONSISTENTE. LOS PASOS A SEGUIR SON:

1) DIVIDIR LAS TRANSACCIONES ENTRE COMPLETAS E INCOMPLETAS

<START T>….<COMMIT T>…. = COMPLETA
<START T>….<ABORT T>……. = COMPLETA
<START T>…… = INCOMPLETA

2) DESHACER LOS CAMBIOS DE LAS T INCOMPLETAS

EL RECOVERY MANAGER RECORRE EL LOG **DESDE EL FINAL HACIA ARRIBA** RECORDANDO LAS T DE LAS CUALES HA VISTO UN RECORD <COMMIT> O UN RECORD <ABORT>. A MEDIDA QUE VA SUBIENDO SI ENCUENTRA UN RECORD <T, X, v>, ENTONCES:

A. SI T ES UNA TRANSACCION CUYO <COMMIT> (O <ABORT>) HA SIDO VISTO, ENTONCES NO HACER NADA, T HA "COMITEADO" (O ABORTADO) Y NO DEBE SER DESHECHA.

B. SINO, T ES UNA TRANSACCION INCOMPLETA. EL RECORY MANAGER DEBE CAMBIAR EL VALOR DE X A v (X HA SIDO ALTERADO JUSTO ANTES DEL CRASH)

C. FINALMENTE EL RECOVERY MANAGER DEBE GRABAR EN EL LOG UN <ABORT T> POR CADA T INCOMPLETA QUE NO FUE PREVIAMENTE ABORTADA Y LUEGO HACER UN FLUSH DEL LOG.

AHORA SE PUEDE REANUDAR LA OPERACION NORMAL DE LA BD.

EJEMPLO DE RECOVERY (UNDO LOGGING):

Para el ejemplo anterior, supongamos que el crash ocurre:

1. Después de (12)
2. Entre (11) y (12)
3. Entre (10) y (11)
4. Entre (8) y (10)
5. Antes de (8)

EJEMPLO (UNDO LOGGING):

T = READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

PASO	ACCION	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A, t) (1)	8	8		8	8	
3	t:= t*2	16	8		8	8	
4	WRITE(A, t)	16	16		8	8	<T, A, 8>
5	READ(B,t) (1)	8	16	8	8	8	
6	t:= t*2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 8>
8	FLUSH LOG						
9	OUTPUT(A)	16	16	16	16	8	
10	OUTPUT(B)	16	16	16	16	16	
11							<COMMIT T>
12	FLUSH LOG						

(1) Si no se encontrara el item A en el pool de buffers se ejecutará un INPUT(A) para traerlo de disco. Idem para el item B.

Solución:

1. Todos los Log Records de T son ignorados.
2. A) Si <COMMIT T> fue "flusheado" entonces idem (1), B) Sino recorrer el Log de abajo hacia arriba, restaurar B y luego A en 8, escribir un <ABORT T> y "flushear" el Log.
3. Idem (2) (B)
4. Idem (3) (Aunque quizás los cambios en A y/o B no hayan llegado a Disco)
5. No sabemos si algún Log Record llegó a Disco. De todas formas proceder idem (4)

EL UNDO LOGGING TIENE LAS SIGUIENTES CARACTERÍSTICAS:

- No podemos "Comitear" una transacción sin antes grabar todos sus cambios en Disco.

- Requiere que los Items sean grabados inmediatamente después de que la transacción terminó, quizás incrementando el número de I/O.

REDO LOGGING:

EN REDO LOGGING EL RECOVERY MANAGER REHACE LAS TRANSACCIONES "COMITEADAS".

UPDATE LOG RECORD PARA REDO-LOGGING:

ES DE LA FORMA: **<T, X, w>**

SIGNIFICA QUE: **T ESCRIBIO EL NUEVO VALOR w PARA EL ITEM X**

REGLA DE REDO-LOGGING:

HAY UNA SOLA REGLA, LLAMADA **"WRITE AHEAD LOGGING RULE"**:

R_1: ANTES DE MODIFICAR CUALQUIER ITEM X EN DISCO, ES NECESARIO QUE TODOS LOS LOG RECORDS CORRESPONDIENTES A ESA MODIFICACION DE X, INCLUYENDO EL UPDATE RECORD <T, X, w> Y EL <COMMIT T> RECORD, DEBEN APARECER EN DISCO.

DE R1 SE DESPRENDE QUE LOS ELEMENTOS ASOCIADOS A T DEBEN SER GRABADOS EN DISCO EN EL SIGUIENTE **ORDEN**:

A. LOS LOG RECORDS INDICANDO LOS CAMBIOS A LOS ITEMS
B. EL COMMIT LOG RECORD
C. LOS ITEMS CON SUS NUEVOS VALORES

EJEMPLO (REDO LOGGING):

T= READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

PASO	ACCION	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A, t)	8	8		8	8	
3	t:= t*2	16	8		8	8	
4	WRITE(A, t)	16	16		8	8	<T, A, 16>
5	READ(B,t)	8	16	8	8	8	
6	t:= t*2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 16>
8							<COMMIT T>
9	FLUSH LOG						
10	OUTPUT(A)	16	16	16	16	8	
11	OUTPUT(B)	16	16	16	16	16	

RECUPERACION USANDO REDO LOGGING:

UNA CONSECUENCIA IMPORTANTE DEL REDO LOGGING ES QUE AL MENOS QUE EL LOG TENGA UN <COMMIT T> RECORD, SABEMOS QUE NINGUNO DE LOS CAMBIOS QUE HA HECHO T HAN SIDO GRABADOS EN DISCO.

LAS T "COMITEADAS" DEBEN SER REHECHAS

POR LO TANTO EL RECOVERY MANAGER USANDO EL REDO-LOG DEBE:

1) DIVIDIR LAS TRANSACCIONES ENTRE "COMITEADAS" Y "NO-COMITEADAS".

<START T>….<COMMIT T>…. = "COMITEADA"
<START T>….<ABORT T>……. = "NO-COMITEADA" (ABORTADA)
<START T>…… = "NO-COMITEADA" (INCOMPLETA)

2) RECORRER EL LOG **DESDE EL PRINCIPIO HACIA ABAJO.** POR CADA LOG RECORD <T, X, w> ENCONTRADO HACER:

A. SI T **NO** ES UNA TRANSACCION "COMITEADA", NO HACER NADA
B. SI T ES "COMITEADA", GRABAR EL VALOR w PARA X
C. POR CADA TRANSACCION T INCOMPLETA, GRABAR UN <ABORT T> EN EL LOG Y HACER FLUSH DEL LOG.

EJEMPLO DE RECOVERY (REDO LOGGING):

Para el ejemplo anterior, supongamos que el crash ocurre:

1. Después de (9)
2. Entre (8) y (9)
3. Antes de (8)

EJEMPLO (REDO LOGGING):

T = READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

PASO	ACCION	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A, t)	8	8		8	8	
3	t:= t*2	16	8		8	8	
4	WRITE(A, t)	16	16		8	8	<T, A, 16>
5	READ(B,t)	8	16	8	8	8	
6	t:= t*2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 16>
8							<COMMIT T>
9	FLUSH LOG						
10	OUTPUT(A)	16	16	16	16	8	
11	OUTPUT(B)	16	16	16	16	16	

Solución:

1. Recorrer el Log de arriba hacia abajo restaurando A y B en 16. Si ocurriera entre (10) y (11) o después de (11) las restauraciones serían redundantes pero "inofensivas".

2. Si el Log Record <COMMIT T> llegó a Disco, entonces idem (1), sino idem (3)

3. El <COMMIT T> seguro no llegó a Disco, entonces ningún cambio es hecho por T. Luego escribir un <ABORT T> y "flushear" el Log.

UNDO LOGGING VS. REDO LOGGING:

Las principales diferencias entre REDO y UNDO Logging son:

- Mientras que UNDO Logging cancela los efectos de las T Incompletas e ignora las T "Comiteadas" durante el recovery, REDO Logging ignora las Incompletas y repite los cambios hechos por las "Comiteadas".

- Mientras que UNDO Logging requiere grabar los cambios (de los Items) en Disco antes que el <COMMIT T> se grabe en Disco, REDO Logging requiere que el <Commit> se grabe en Disco antes que cualquier cambio (de los Items) se grabe en Disco.

- Mientras que los valores anteriores de los Items es lo que necesitamos para recuperar usando UNDO Logging, para recuperar usando REDO Logging necesitamos en cambio los valores nuevos.

¿PODEMOS MEJORAR LO VISTO HASTA AHORA?

Los dos estilos de Logging vistos hasta ahora tienen las siguientes características:

- Como ya dijimos, el UNDO Logging requiere que los Items sean grabados inmediatamente después de que la transacción terminó, quizás incrementando el número de I/O.

- Por otro lado, el REDO Logging requiere mantener todos los bloques modificados en el Pool de Buffers hasta que la transacción "Comitea" y los Log Records fueron "flusheados", quizás incrementando el promedio de buffers requeridos por las transacciones.

- Ahora veremos otro tipo de Logging llamado UNDO/REDO, el cual provee mayor flexibilidad para ordenar las acciones pero a expensas de mantener más información en el Log.

UNDO/REDO LOGGING:

EN UNDO/REDO LOGGING EL RECOVERY MANAGER TIENE LA INFORMACION EN EL LOG PARA DESHACER LOS CAMBIOS HECHOS POR T, O REHACER LOS CAMBIOS HECHOS POR T.

UPDATE LOG RECORD PARA UNDO/REDO-LOGGING:

EL UPDATE LOG RECORD AHORA TIENE LA FORMA: **<T, X, v, w>**

SIGNIFICA QUE: **LA TRANSACCION T CAMBIO EL VALOR DEL ITEM X, CUYO VALOR ANTERIOR ERA v Y SU NUEVO VALOR ES w**

REGLA DEL UNDO/REDO LOGGING:

UR_1: ANTES DE MODIFICAR CUALQUIER ITEM X EN DISCO DEBIDO A LOS CAMBIOS EFECTUADOS POR ALGUNA TRANSACCION T, ES NECESARIO QUE EL UPDATE RECORD <T, X, v, w> APAREZCA EN DISCO.

UR_2: UN <COMMIT T> RECORD DEBE SER FLUSHEADO A DISCO INMEDIATAMENTE DESPUES DE QUE APAREZCA EN EL LOG **(REGLA OPCIONAL)**

ORDEN: SE DESPRENDE QUE EL <COMMIT T> PUEDE PRECEDER O SEGUIR A CUALQUIERA DE LOS CAMBIOS DE LOS ITEMS EN DISCO.

EJEMPLO (UNDO/REDO LOGGING):

T= READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

PASO	ACCION	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A, t)	8	8		8	8	
3	t:= t*2	16	8		8	8	
4	WRITE(A, t)	16	16		8	8	<T, A, 8, 16>
5	READ(B,t)	8	16	8	8	8	
6	t:= t*2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 8, 16>
8	FLUSH LOG						
9	OUTPUT(A)	16	16	16	16	8	
10							<COMMIT T>
11	OUTPUT(B)	16	16	16	16	16	

PODEMOS OBSERVAR EL <COMMIT T> EN EL MEDIO DEL OUTPUT DE LOS ITEMS A Y B.
EL PASO (10) PODRIA ESTAR ANTES DE (8) O (9), O DESPUES DE (11).

RECUPERACION USANDO UNDO/REDO LOGGING:

COMO YA DIJIMOS TENEMOS LA INFORMACION EN EL LOG PARA DESHACER LOS CAMBIOS HECHOS POR T, O REHACER LOS CAMBIOS HECHOS POR T, O SEA QUE ES UNA COMBINACION DE LOS ALGORITMOS DE RECOVERY DE UNDO Y REDO.

SE DEBEN DESHACER LAS T INCOMPLETAS Y REHACER LAS "COMITEADAS"

POR LO TANTO EL RECOVERY MANAGER DEBE:

1) **DESHACER (UNDO) TODAS LAS TRANSACCIONES INCOMPLETAS** EN EL ORDEN LA MAS RECIENTE PRIMERO.

2) **REHACER (REDO) TODAS LAS TRANSACCIONES "COMITEADAS"** EN EL ORDEN LA MAS ANTIGUA PRIMERO

3) POR ULTIMO, POR CADA TRANSACCION T INCOMPLETA ENCONTRADA GRABAR UN <ABORT T> EN EL LOG Y HACER FLUSH DEL LOG.

EJEMPLO DE RECOVERY (UNDO/REDO LOGGING):

Para el ejemplo anterior, supongamos que el crash ocurre:

1. Después de que <COMMIT T> es flusheado a Disco

2. Antes de que <COMMIT T> llegue a Disco

EJEMPLO (UNDO/REDO LOGGING):

T = READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

PASO	ACCION	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A, t)	8	8		8	8	
3	t:= t*2	16	8		8	8	
4	WRITE(A, t)	16	16		8	8	<T, A, 8, 16>
5	READ(B,t)	8	16	8	8	8	
6	t:= t*2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 8, 16>
8	FLUSH LOG						
9	OUTPUT(A)	16	16	16	16	8	
10							<COMMIT T>
11	OUTPUT(B)	16	16	16	16	16	

PODEMOS OBSERVAR EL <COMMIT T> EN EL MEDIO DEL OUTPUT DE LOS ITEMS A Y B.
EL PASO (10) PODRIA ESTAR ANTES DE (8) O (9), O DESPUES DE (11).

Solución:

1. Entonces T es identificada como “COMITEADA”. Escribimos 16 para A y B en Disco (Operación REDO)

2. Entonces T es tratada como INCOMPLETA y restauramos B y A en 8 y luego escribimos un <ABORT> t y “flusheamos” el log (Operación UNDO)

RESUMEN:

Dada la transacción:

T= READ(A, t); t:= t*2; WRITE(A, t); READ(B, t); t:= t*2; WRITE(B, t)

Y los **ITEMS A** Y **B** inicialmente igual a 8.

El Log a generar para cada estilo de Logging será:

UNDO: <START T>; <T, A, 8>; <T, B, 8>; <COMMIT T>

REDO: <START T>; <T, A, 16>; <T, B, 16>; <COMMIT T>

UNDO/REDO: <START T>; <T, A, 8, 16>; <T, B, 8, 16>; <COMMIT T>

CHECKPOINTING:

Se hace periódicamente.

Es para no tener que recorrer siempre todo el Log durante el Recovery.

Hay dos estilos:

- Quiescente (quieto)
  Se detiene la recepción de nuevas transacciones durante el Checkpointing.

- No Quiescente
  Se siguen recibiendo nuevas transacciones durante el Checkpointing.

BIBLIOGRAFÍA:

Los ejemplos de esta clase fueron tomados del siguiente libro:

DATABASE SYSTEMS – THE COMPLETE BOOK, DE H. GARCIA MOLINA, J. D. ULLMAN Y J. WIDOM, 2002.

¿PREGUNTAS?