

Logging

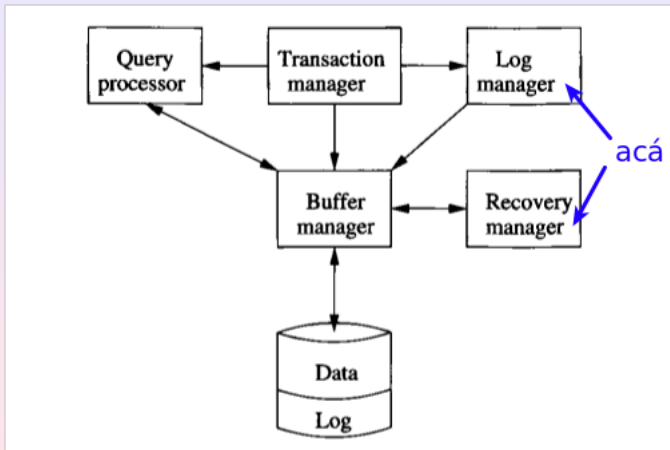
Logging

- ▶ ¿Para qué?
- ▶ ¿Por qué?
- ▶ ¿Cómo?
- ▶ ¿Quién?

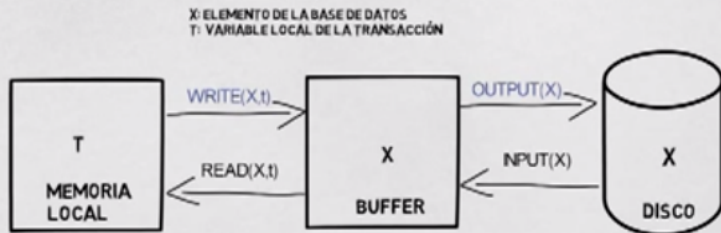
Logging

- ▶ ¿Para qué?: para llevar la base de datos a un estado consistente
- ▶ ¿Por qué?: Porque si se produce una falla, las transacciones en ejecución pueden dejar la base en un estado inconsistente. Ej.: transferencia bancaria (si la falla se produce entre el débito y el crédito)
- ▶ ¿Cómo?: el recovery se hace rehaciendo o deshaciendo cambios (según la política de logging)
- ▶ ¿Quién?: el módulo encargado del recovery es el *Recovery Manager*

Contexto



Operaciones de lectura y escritura



LAS OPERACIONES READ Y WRITE SON ENTRE LA MEMORIA LOCAL DE LA TRANSACCIÓN Y EL BUFFER.

LAS OPERACIONES INPUT Y OUTPUT SON ENTRE EL BUFFER Y EL DISCO.

INPUT(X) COPIA EL BLOQUE QUE CONTIENE EL ELEMENTO X DESDE EL DISCO AL BUFFER
 OUTPUT(X) COPIA EL BLOQUE QUE CONTIENE EL ELEMENTO X DESDE EL BUFFER AL DISCO
 FLUSH LOG: COMANDO EMITIDO POR EL LOG MANAGER PARA QUE EL BUFFER MANAGER
 FUERCE LOS LOG RECORDS A DISCO.

Archivo de log

- ▶ El *Recovery Manager* utiliza el archivo de log para recuperar un estado consistente de la base
- ▶ El log es un archivo *append-only*
- ▶ Registra todas las modificaciones de todas las transacciones
- ▶ Los registros son idempotentes, ¿por qué?

```
<START T1>  
<T1, A, 5>  
<START T2>  
<T2, B, 10>  
<T2, C, 15>  
<T1, D, 20>  
<COMMIT T1>  
<COMMIT T2>  
<CKPT>  
<START T3>  
<T3, E, 25>  
<T3, F, 30>
```

Figure 17.4: An undo log

Políticas de logging



Políticas de logging



UNDO LOGGING

$\langle T, X, V \rangle$. INDICA QUE LA TRANSACCION T HA CAMBIADO EL VALOR DE X CUYO VALOR ANTERIOR ERA V .

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

REGLAS

- 1 LOS REGISTROS DEL TIPO $\langle T, X, V \rangle$ SE ESCRIBEN A DISCO **ANTES** QUE EL NUEVO VALOR DE X SE ESCRIBA A DISCO EN LA BASE DE DATOS.
- 2 LOS REGISTROS $\langle \text{COMMIT } T \rangle$ SE ESCRIBEN A DISCO **DESPUES** QUE TODOS LOS ELEMENTOS MODIFICADOS POR T SE HAYAN ESCRITO EN DISCO.

Política: Undo (ejemplo libro de Ullman, cap. 17)

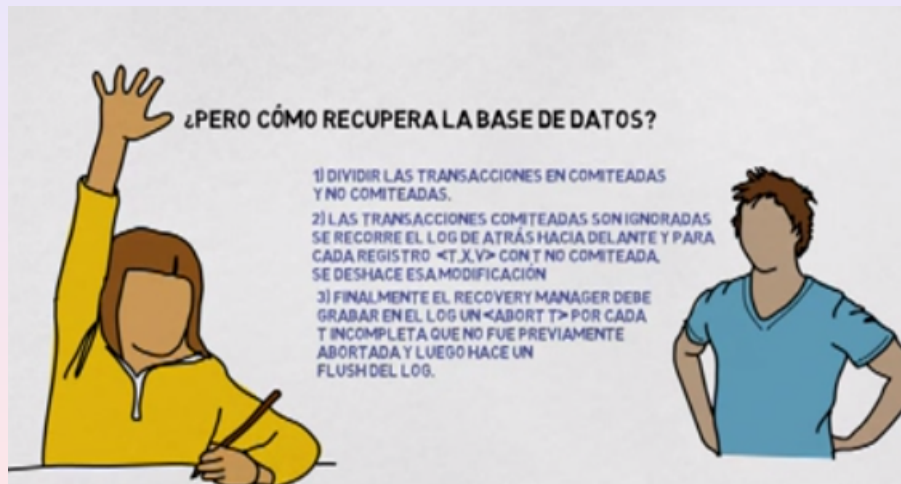
Supongamos que tenemos una transacción T:

A := A*2

B := B*2

Step	Action	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>
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>
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						

Política: Undo (Recovery)



¿PERO CÓMO RECUPERA LA BASE DE DATOS?

- 1) DIVIDIR LAS TRANSACCIONES EN COMITEADAS Y NO COMITEADAS.
- 2) LAS TRANSACCIONES COMITEADAS SON IGNORADAS SE RECORRE EL LOG DE ATRÁS HACIA DELANTE Y PARA CADA REGISTRO $\langle T, X, Y \rangle$ CON T NO COMITEADA, SE DESHACE ESA MODIFICACIÓN
- 3) FINALMENTE EL RECOVERY MANAGER DEBE GRABAR EN EL LOG UN $\langle \text{ABORT } T \rangle$ POR CADA T INCOMPLETA QUE NO FUE PREVIAMENTE ABORTADA Y LUEGO HACE UN FLUSH DEL LOG.

Política: Undo (Recovery)

$\langle \text{START } T_1 \rangle$

$\langle T_1, A, 5 \rangle$

$\langle \text{START } T_2 \rangle$

$\langle T_2, B, 10 \rangle$

$\langle T_2, C, 15 \rangle$

$\langle T_1, D, 20 \rangle$

$\langle \text{COMMIT } T_1 \rangle$

$\langle \text{COMMIT } T_2 \rangle$

$\langle \text{START } T_3 \rangle$

$\langle T_3, E, 25 \rangle$

$\langle T_3, F, 30 \rangle$



Figure 17.4: An undo log

Política: Undo (Desventajas)

DESVENTAJAS DE UNDO LOGGING:

NO PODEMOS COMITEAR UNA TRANSACCION SIN ANTES GRABAR TODOS SUS CAMBIOS EN DISCO.

REQUIERE QUE LOS ITEMS SEAN GRABADOS INMEDIATAMENTE DESPUES DE QUE LA TRANSACCION TERMINO, QUIZAS INCREMENTANDO EL NUMERO DE I/O.

REDO LOGGING

$\langle T, X, V \rangle$. INDICA QUE LA TRANSACCION T HA CAMBIADO EL VALOR DE X Y EL NUEVO VALOR ES V .

REGLAS

- 1 LOS REGISTROS DEL TIPO $\langle T, X, V \rangle$ SE ESCRIBEN A DISCO ANTES QUE EL NUEVO VALOR DE X SE ESCRIBA A DISCO EN LA BASE DE DATOS.
- 2 LOS REGISTROS $\langle \text{COMMIT } T \rangle$ SE ESCRIBEN A DISCO ANTES QUE TODOS LOS ELEMENTOS MODIFICADOS POR T SE HAYAN ESCRITO EN DISCO

Política: Redo (ejemplo libro de Ullman, cap. 17)

Supongamos que tenemos una transacción T:

A := A*2

B := B*2

Step	Action	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	

Política: Redo (Recovery)

PARA RECUPERAR LA BASE DE DATOS SE PROCEDE DE LA SIGUIENTE MANERA:

1) DIVIDIR LAS TRANSACCIONES EN COMPLETAS (CON ABORT O COMMIT) E INCOMPLETAS.

2) LAS TRANSACCIONES SIN ABORT NI COMMIT LAS PODEMOS IGNORAR, YA QUE ESTAMOS SEGUROS QUE SUS DATOS NUNCA LLEGARON A DISCO.

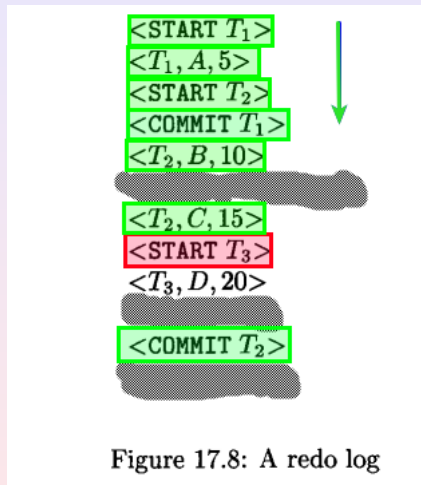
AQUELLAS TRANSACCIONES COMPLETAS DEBIDO A QUE SE ENCONTRÓ UN <ABORT T> PUEDEN SER IGNORADAS PORQUE SUS ACCIONES YA FUERON MANEJADAS PREVIAMENTE.

PARA AQUELLAS CON <START T> Y <COMMIT T>, NO ESTAMOS SEGUROS QUE SUS CAMBIOS SE HAYAN REGISTRADO EN LA BASE DE DATOS

POR LO TANTO, SE RECORRE EL LOG DESDE EL REGISTRO MÁS ANTIGUO HASTA EL FINAL Y PARA CADA REGISTRO <T,X,V> DONDE T ES UNA TRANSACCIÓN DE ESTE TIPO, SE REHACE ESA MODIFICACIÓN



Política: Redo (Recovery)



Política: Redo (Desventajas)

- ▶ Requiere que se mantengan en memoria todos los buffers modificados por la transacción hasta el commit (porque el $\langle \text{Commit } T \rangle$ se escribe en el log antes de que se graben los cambios)
- ▶ ¿Qué pasa si una transacción se extiende en el tiempo? ¿Y si modifica muchos bloques?

Política: Undo/Redo

UNDO/REDO LOGGING

LOS REGISTROS DEL TIPO $\langle T, X, V, W \rangle$ SE ESCRIBEN A DISCO ANTES QUE EL NUEVO VALOR DE X SE ESCRIBA A DISCO EN LA BASE DE DATOS

EL OBJETIVO ES DESHACER LAS TRANSACCIONES INCOMPLETAS Y REHACER AQUELLAS CON $\langle \text{START } T \rangle$ Y $\langle \text{COMMIT } T \rangle$ (AQUELLAS CON $\langle \text{START } T \rangle$ Y $\langle \text{ABORT } T \rangle$ SON IGNORADAS).

Política: Undo/Redo (ejemplo libro de Ullman, cap. 17)

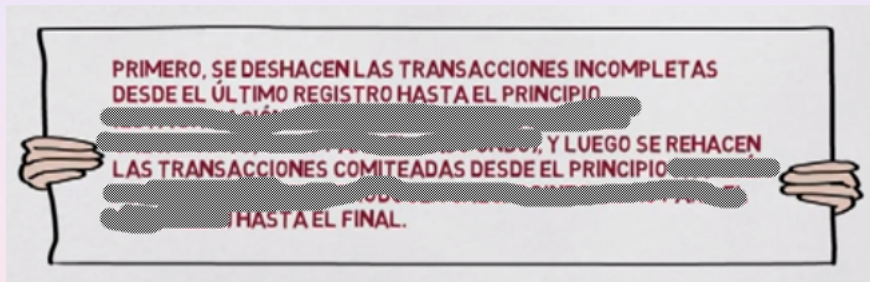
Supongamos que tenemos una transacción T:

A := A*2

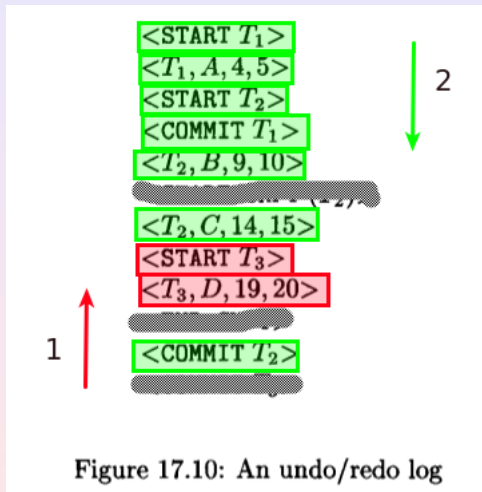
B := B*2

Step	Action	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	

Política: Undo/Redo (Recovery)



Política: Undo/Redo (Recovery)

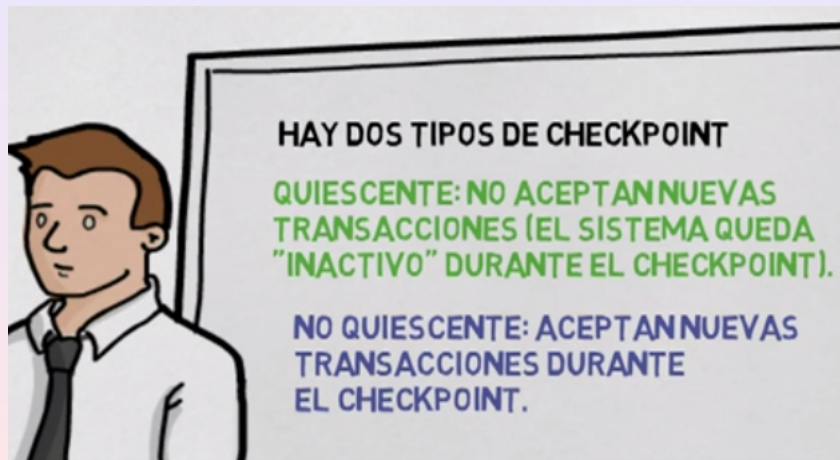


Checkpoint

- ▶ ¿Hace falta revisar todo el log para restaurar un estado consistente de la base?



Checkpoint



Política Undo con checkpoint quiescente

UNDO LOGGING CON CHECKPOINT QUIESCENTE.

PASOS A PARTIR DE QUE SE LANZA UN CHECKPOINT:

- 1) DEJAR DE ACEPTAR NUEVAS TRANSACCIONES.
- 2) ESPERAR A QUE TODAS LAS TRANSACCIONES ACTIVAS (AQUELLAS CON <START T> Y SIN COMMIT NI ABORT) COMIENEN O ABORTEN
- 3) ESCRIBIR UN <CKPT> EN EL LOG Y LUEGO EFECTUAR UN FLUSH.
- 4) ACEPTAR NUEVAS TRANSACCIONES

Política Undo con checkpoint quiescente (Recovery)

- ▶ Con Undo tenemos garantía de que los cambios se graban antes de que el <Commit T> aparezca en el log
- ▶ Por lo tanto es suficiente con revisar el log de atrás para adelante hasta encontrar un <CKPT>

Política Undo con checkpoint no quiescente

UNDO LOGGING CON CHECKPOINT NO QUIESCENTE.

PASOS A PARTIR DE QUE SE LANZA UN CHECKPOINT NO QUIESCENTE

1) ESCRIBIR **<START CKPT(T1,T2,...,TK)>** EN EL LOG, Y EFECTUAR UN FLUSH.
T1,T2,...,TK SON LAS TRANSACCIONES ACTIVAS AL MOMENTO DE
INTRODUCIR EL CHECKPOINT

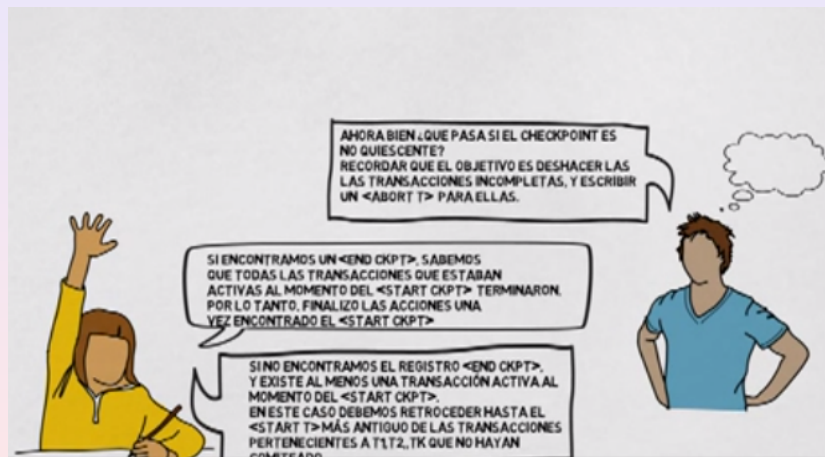
2) ESPERAR A QUE TODAS LAS TRANSACCIONES T1,T2,...,TK TERMINEN (YA SEA
ABORTANDO O COMITEANDO), PERO SIN PROHIBIR QUE EMPIECEN OTRAS
TRANSACCIONES

3) ESCRIBIR **<END CKPT>** EN EL LOG Y EFECTUAR UN FLUSH.

CHECKPOINT START LOG RECORD: <START CKPT (T1,...,TK)>

CHECKPOINT END LOG RECORD: <END CKPT>


Política Undo con checkpoint no quiescente (Recovery)





Política Redo con checkpoint no quiescente

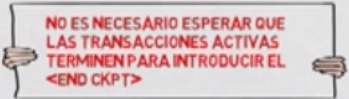
Redo Logging con Checkpoint no-Quiescente

La política para los <Start CKPT> y <End CKPT> son las siguientes:

Escribir un registro <Start CKPT(T_1, T_2, \dots, T_k)> en el log, y efectuar un flush. 
 T_1, T_2, \dots, T_k son las transacciones activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint.

Esperar a que todas las modificaciones realizadas por transacciones ya comiteadas al momento de introducir el <Start CKPT> sean escritas a disco. 

Escribir el registro <End CKPT> en el log y efectuar un flush. 



NO ES NECESARIO ESPERAR QUE
LAS TRANSACCIONES ACTIVAS
TERMINEN PARA INTRODUCIR EL
<END CKPT>

Política Redo con checkpoint no quiescente (Recovery)

PARA RECUPERAR LA BASE DE DATOS CON REDO LOGGING CON CHECKPOINT NO-QUIESCENTE



COMO SIEMPRE, EL OBJETIVO ES REHACER LAS ACCIONES DE LAS TRANSACCIONES COMITEADAS (AQUELLAS CON <START T> Y <COMMIT T>), Y ESCRIBIR UN <ABORT T> PARA CADA UNA DE LAS INCOMPLETAS. AQUELLAS CON <START T> Y <ABORT T> SON IGNORADAS.

Si encontramos un <End CKPT>, podemos ignorar todas aquellas transacciones que comitearon previamente al último registro <START CKPT>, ya que sus datos fueron escritos a disco y no es necesario rehacerlos.

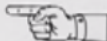


SOLO NOS CONCENTRAMOS EN LAS TRANSACCIONES ACTIVAS AL MOMENTO DEL CHECKPOINT Y LAS QUE COMENZARON DESPUÉS.

DEBEMOS COMENZAR DESDE EL <START T> MÁS ANTIGUO DE LAS TRANSACCIONES A REHACER (SI SE TRATA DE UNA DE LAS TRANSACCIONES PERTENECIENTES A T1 T2... TK PUEDE SER NECESARIO RETROCEDER MÁS ALLÁ DEL <START CKPT>)



Si no encontramos un registro <End CKPT> NO SABEMOS CON CERTEZA SI LAS MODIFICACIONES DE LAS TRANSACCIONES YA COMITEADAS AL MOMENTO DEL <START CKPT> FUERON ESCRITAS A DISCO. DEBEMOS RETROCEDER HASTA SU COMIENZO, PARA REHACER DE AHÍ EN MÁS.



PARA ESTO, SE UBICA EL REGISTRO <END CKPT> ANTERIOR Y SU CORRESPONDIENTE <START CKPT IS 152 SK>

Política Undo/Redo con checkpoint no quiescente (Recovery)

UNDO/REDO CON CHECKPOINTS NO QUIESCENTES

LA POLÍTICA PARA LOS <START CKPT> Y <END CKPT> SON LAS SIGUIENTES:

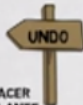
ESCRIBIR UN REGISTRO <START CKPT (T1 T2.. TK)> EN EL LOG, Y EFECTUAR UN FLUSH.
T1 T2.. TK SON LAS TRANSACCIONES ACTIVAS (AQUELLAS CON <START T> Y SIN
<COMMIT T>) AL MOMENTO DE INTRODUCIR EL CHECKPOINT

ESPERAR A QUE TODAS LAS MODIFICACIONES REALIZADAS POR TRANSACCIONES (COMITEADAS
O NO) AL MOMENTO DE INTRODUCIR EL <START CKPT> SEAN ESCRITAS A DISCO.

ESCRIBIR EL REGISTRO <END CKPT> EN EL LOG Y EFECTUAR UN FLUSH.

OPERATORIA:

PARA DESHACER LAS TRANSACCIONES INCOMPLETAS
DEBEREMOS RETROCEDER HASTA EL START MÁS ANTIGUO DE ELLAS



SI ENCONTRAMOS UN REGISTRO <END CKPT> SÓLO SERÁ NECESARIO REHACER
LAS ACCIONES EFECTUADAS DESDE EL REGISTRO <START CKPT> EN ADELANTE

SI NO ENCONTRAMOS EL <END CKPT>, PARA SABER DESDE DÓNDE COMENZAR
UTILIZAMOS EL MECANISMO USADO EN LA ESTRATEGIA REDO