Base de Datos. 1C2009
Clase práctica de Recuperación
Docente: Alejandro Eidelsztein

RECUPERACION ANTE FALLAS

CONCEPTOS:

SYSTEM FAILURES O CRASHES

RESILIENCIA

LOG: REGISTRA LA HISTORIA DE LOS CAMBIOS EFECTUADOS A LA BD

PROCESO DE RECUPERACION (RECOVERY)
(LA OPERACION DE RECOVERY DEBE SER IDEMPOTENTE)

ESTILOS DE LOGGING:
- UNDO LOGGING
- REDO LOGGING
- UNDO/REDO LOGGING

LOG MANAGER

RECOVERY MANAGER

COMPONENTES:

QUERY PROCESSOR

TRANSACTION MANAGER

LOG MANAGER

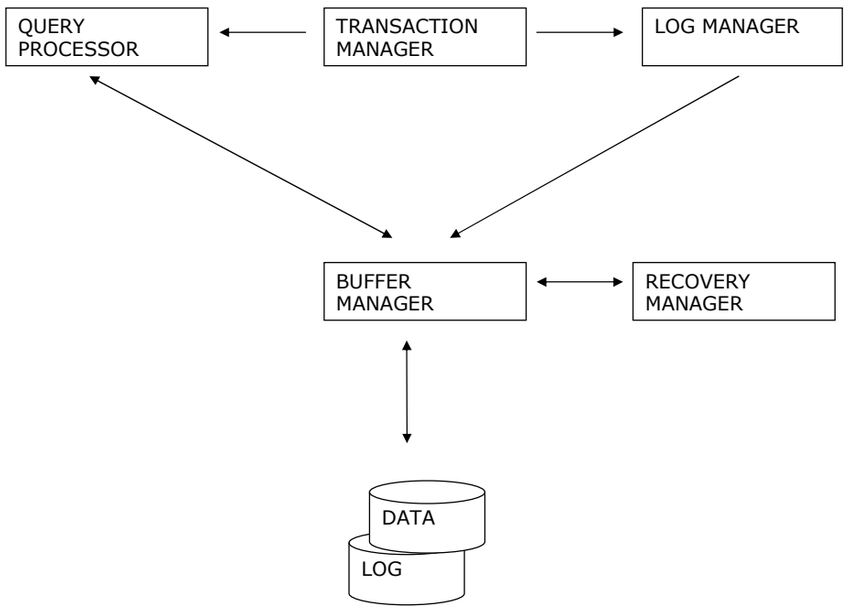
BUFFER MANAGER

RECOVERY MANAGER

DATA

LOG

INTERACCION DE COMPONENTES



ESPACIOS DE MEMORIA:

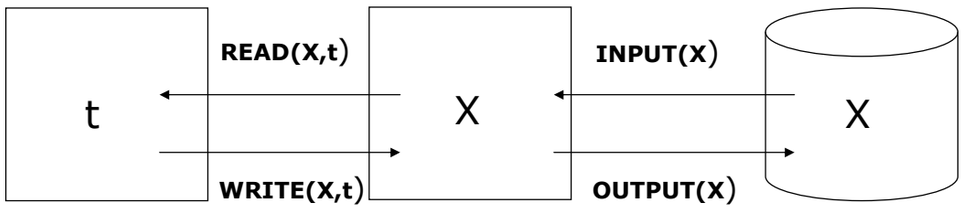
LOCAL DE LA TRANSACCION

POOL DE BUFFERS

DISK BLOCKS (CONTIENEN LOS ITEMS)

OPERACIONES PRIMITIVAS Y ESPACIOS DE MEMORIA

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



MEM LOCAL DE LA TRANS **POOL DE BUFFERS** **DISK BLOCKS DE LA BD**

OPERACIONES (PRIMITIVAS) DE UNA TRANSACCION:

INPUT(X): COPIA EL BLOQUE QUE CONTIENE A X AL POOL DE BUFFERS

READ(X, t): COPIA X A LA VARIABLE LOCAL t DE LA TRANSACCION. SI EL BLOQUE QUE CONTIENE X NO ESTA EN EL POOL DE BUFFERS, PRIMERO EJECUTA INPUT(X). Y LUEGO ASIGNA EL VALOR DE X A LA 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 ESTA EN EL 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 EL BLOQUE CONTENIENDO X DESDE EL POOL DE BUFFERS 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.

FLUSH LOG: COMANDO EMITIDO POR EL LOG MANAGER PARA QUE EL BUFFER MANAGER FUERCE LOS LOG RECORDS A DISCO.

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 ALOCADOS POR EL BUFFER MANAGER. EL COMANDO FLUSH-LOG ORDENA COPIAR LOS BLOQUES QUE CONTIENEN LOS LOG RECORDS A DISCO.

EL LOG ES UN ARCHIVO DEL TIPO *APPEND-ONLY*.

LAS TRANSACCIONES 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 TRANSACCION T HA EMPEZADO

Commit Log Record:

<COMMIT T>: INDICA QUE LA TRANSACCION T A COMPLETADO EXITOSAMENTE. LOS CAMBIOS HECHOS POR T DEBERIAN ESTAR EN DISCO.

Abort Log Record:

<ABORT T>: INDICA QUE LA TRANSACCION T PODRIA NO HABER COMPLETADO EXITOSAMENTE. LOS CAMBIOS REALIZADOS POR T NO DEBEN APARECER EN DISCO.

Update Log Record (distinto para cada estilo de logging)**:**

Undo Logging:	**<T, X, v>**
Redo Logging:	**<T, X, w>**
Undo/Redo Logging:	**<T, X, v, w>**

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>. INDICA QUE LA TRANSACCION T HA CAMBIADO EL VALOR DE X CUYO VALOR ANTERIOR ERA v.

REGLAS DEL UNDO-LOGGING:

U1: SI T MODIFICA X, LUEGO EL LOG RECORD <T, X, v> DEBE GRABARSE EN DISCO ANTES QUE EL NUEVO VALOR DE X SEA GRABADO EN DISCO.

U2: SI T "COMITEA", ENTONCES SU COMMIT LOG RECORD <COMMIT> DEBE SER GRABADO EN DISCO SOLO DESUPUES DE QUE TODOS LOS ITEMS CAMBIADOS POR T HAN SIDO GRABADOS EN DISCO (Y TAN PRONTO COMO SEA POSIBLE)

DE U1 Y U2 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 <u>UNDO LOGGING</u>:

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) <u>DIVIDIR LAS TRANSACCIONES ENTRE COMPLETAS E INCOMPLETAS</u>

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

2) <u>DESHACER LOS CAMBIOS DE LAS T INCOMPLETAS</u>

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.

<u>EJEMPLO DE RECOVERY (UNDO LOGGING):</u>

PARA EL EJEMPLO ANTERIOR, SUPONGAMOS QUE EL CRASH OCURRE:

1. DESPUES DE (12)

2. ENTRE (11) Y (12)

3. ENTRE (10) Y (11)

4. ENTRE (8) Y (10)

5. ANTES DE (8)

SOLUCION:

1. DESPUES DE (12): TODOS LOS LOG RECORDS DE T SON IGNORADOS.

2. ENTRE (11) Y (12): 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, GRABAR <ABORT T> Y FLUSHEAR LOG.

3. ENTRE (10) Y (11): IDEM (2) (B)

4. ENTRE (8) Y (10): IDEM (3) (AUNQUE QUIZAS LOS CAMBIOS EN A Y/O B NO HAYAN LLEGADO A DISCO

5. ANTES DE (8):NO SABEMOS SI ALGUN LOG RECORD LLEGO A DISCO, PERO ESTO NO ES UN PROBLEMA POR LA REGLA U1.

EL UNDO LOGGING TIENE LAS SIGUIENTES DESVENTAJAS:

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:

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

UPDATE LOG RECORD PARA REDO-LOGGING:

EN REDO LOGGING EL SIGNIFICADO DE <T, X, w> ES: T ESCRIBIO EL NUEVO VALOR w PARA EL ITEM X.

REGLA DE REDO-LOGGING:

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

R1: 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

UNDO LOGGING VS REDO LOGGING:

LOS PRINCIPALES DIFERENCIAS ENTRE REDO Y UNDO LOGGING SON:

1) 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".

2) MIENTRAS QUE UNDO LOGGING REQUIERE GRABAR LOS CAMBIOS (DE LOS ITEMS) EN DISCO ANTES DE QUE EL <COMMIT T> SE GRABE EN DISCO, REDO LOGGING REQUIERE QUE EL <COMMIT T> SE GRABE EN DISCO ANTES QUE CUALQUIER CAMBIO (DE LOS ITEMS) SE GRABE EN DISCO.

3) MIENTRAS QUE LOS VALORES ANTERIORES ES LO QUE NECESITAMOS CUANDO USAMOS LAS REGLAS U1 Y U2, PARA RECUPERAR USANDO REDO LOGGING NECESITAMOS EN CAMBIO LOS VALORES NUEVOS DE LOS ITEMS.

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 NO-COMITEADAS PUEDEN SER TRATADAS COMO SI NUNCA HUBIERAN EXISTIDO Y LAS 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> ENCONTRANDO:

A) SI T NO ES UNA TRANSACCION "COMITEADA", NO HACER NADA
B) SI T ES "COMITEADA", GRABAR EL VALOR w PARA X

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. DESPUES DE (9)

2. ENTRE (8) Y (9)

3. ANTES DE (8)

SOLUCION:

1. DESPUES DE (9): RECORRER EL LOG DE ARRIBA HACIA ABAJO RESTAURANDO A Y B EN 16. SI OCURRIERA ENTRE (10) Y (11) O DESPUES DE (11) LAS RESTAURACIONES SERIAN REDUNDANTES PERO "INOFENSIVAS".

2. ENTRE (8) Y (9): SI EL LOG RECORD <COMMIT T> LLEGO A DISCO, ENTONCES IDEM (1), SINO IDEM (3)

3. ANTES DE (8): EL <COMMIT T> SEGURO NO LLEGO A DISCO, ENTONCES NINGUN CAMBIO ES HECHO POR T Y EVENTUALMENTE UN <ABORT T> ES GRABADO EN DISCO.

LOS DOS ESTILOS DE LOGGING VISTOS HASTA AHORA TIENEN LAS SIGUIENTES DESVENTAJAS:

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

POR OTRO LADO, EL REDO LOGGING REQUIERE MANTENER TODOS LOS BLOQUES MODIFICADOS EN EL BUFFER POOL HASTA QUE LA TRANSACCION "COMITEA" Y LOS LOG RECORDS FUERON FLUSHEADOS, QUIZAS INCREMENTANDO EL PROMEDIO DE BUFFERS REQUERIDOS POR LAS TRANSACCIONES.

AHORA VEREMOS OTRO TIPO DE LOGGING LLAMADO UNDO/REDO LOGGING, EL CUAL PROVEE MAYOR FLEXIBILIDAD PARA ORDENAR LAS ACCIONES A EXPENSAS DE MANTENER MAS INFORMACON 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:

UR1: 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.

UR2: *UN <COMMIT T> RECORD DEBE SER FLUSHEADO A DISCO INMEDIATAMENTE DESPUES DE QUE APAREZCA EN EL LOG* ***(REGLA OPCIONAL)***

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. LOS PASOS A SEGUIR SON:

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

EJEMPLO (RECOVERY):

PARA EL EJEMPLO ANTERIOR, SUPONGAMOS QUE EL CRASH OCURRE:

1. DESPUES DE QUE <COMMIT T> ES FLUSHEADO A DISCO

2. ANTES DE QUE <COMMIT T> LLEGUE A DISCO

SOLUCION:

1. DESPUES DE QUE <COMMIT T> ES FLUSHEADO A DISCO: ENTONCES T ES IDENTIFICADA COMO "COMITEADA". ESCRIBIMOS 16 PARA A Y B EN DISCO (OPERACIÓN REDO)

2. ANTES DE QUE <COMMIT T> LLEGUE A DISCO: ENTONCES T ES TRATADA COMO INCOMPLETA Y RESTAURAMOS B Y A EN 8 (OPERACIÓN UNDO)

CHECKPOINTING

SE HACE PERIODICAMENTE

ES PARA NO TENER QUE RECORRER SIEMPRE TODO EL LOG DURANTE EL RECOVERY

CHECKPOINTING QUIESCENTE EN UNDO-LOGGING:

CHECKPOINT LOG RECORD: **<CKPT>**

CHECKPOINTING:

1. PARAR DE RECIBIR NUEVAS TRANSACCIONES

2. ESPERAR HASTA QUE LAS TRANSACCIONES ACTIVAS "COMITEEN" O ABORTEN Y HAYAN ESCRITO UN COMMIT O UN ABORT EN EL LOG

3. FLUSHEAR EL LOG A DISCO

4. ESCRIBIR UN CHECKPOINT RECORD LOG <CKPT> Y FLUSHEAR EL LOG NUEVAMENTE

5. VOLVER A ACEPTAR TRANSACCIONES

RECOVERY (USANDO UN UNDO-LOG CON CHECKPOINT QUIESCENTE):

RECORRER EL LOG DESDE EL FINAL HACIA ATRAS IDENTIFICANDO LAS TRANSACCIONES INCOMPLETAS Y RESTAURANDO LOS VALORES ANTERIORES DE LOS ITEMS HASTA QUE ENCONTRAMOS UN <CKPT> RECORD.

CHECKPOINTING NO-QUIESCENTE EN UNDO-LOGGING:

CHECKPOINT START LOG RECORD :**<START CKPT (T1,…,Tk)>**
CHECKPOINT END LOG RECORD: **<END CKPT>**

CHECKPOINTING:

1. ESCRIBIR UN LOG RECORD <START CKPT (T1,…,Tk)> Y FLUSHEAR EL LOG.
LAS T1,…,Tk SON TODAS LAS TRANSACCIONES ACTIVAS (TODAVIA NO HAN "COMITEADO" NI ESCRITO SUS CAMBIOS EN DISCO)

2. ESPERAR HASTA QUE TODAS LAS T1,…,Tk "COMITEEN" O ABORTEN (SIN PROHIBIR QUE OTRAS TRANSACCIONES COMIENCEN)

3. CUANDO TODAS LAS T1,…,Tk HAYAN COMPLETADO, ESCRIBIR UN LOG RECORD <END CKPT> Y FLUSHEAR EL LOG.

RECOVERY (USANDO UN UNDO-LOG CON CHECKPOINT NO-QUIESCENTE):

RECORRER EL LOG DESDE EL FINAL HACIA ATRAS IDENTIFICANDO LAS TRANSACCIONES INCOMPLETAS RESTAURANDO LOS VALORES ANTERIORES DE LOS ITEMS HASTA ENCONTRAR:

1. SI PRIMERO ECONTRAMOS UN <END CKPT>, LUEGO RECORRER HACIA ATRAS HASTA EL PROXIMO <START CKPT ()> Y LUEGO PARAR.

2. SI PRIMERO ENCONTRAMOS UN <START CKPT (T1,…,Tk)> (UN CRASH OCURRIO DURANTE EL CHECKPOINT) DEBEMOS RECORRER HACIA ATRAS HASTA EL START DE LA MAS TEMPRANA DE LAS Ti INCOMPLETAS (LAS T1,…,Tk + LAS QUE EMPEZARON DESPUES)

NOTA DEL DOCENTE: 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.