Recuperación ante fallas

Logging

Ignacio Chiapella[1]

[1]Departamento de Computación
Faculta de Ciencias Exactas y Naturales
Universidad de Buenos Aires

23 de Octubre de 2020

Motivación

- El software falla
- El hardware falla
- Cuando se produce una falla, pueden haber quedado **transacciones sin terminar**

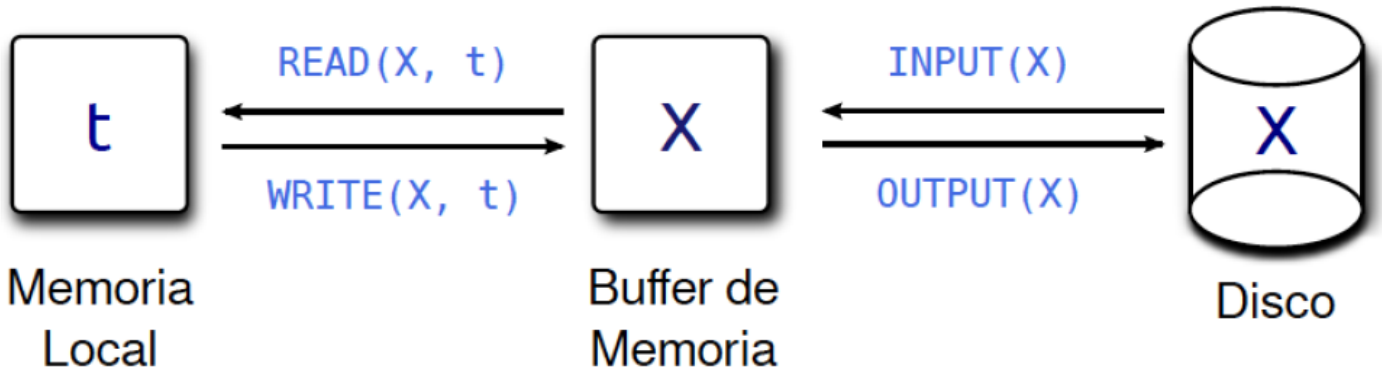
⚠ Toda transacción que no haya comiteado, se asume que dejó **datos inconsistentes** en la base

⚠ Incluso si una transacción comiteó, puede que hayan quedado cambios que **no se hayan bajado a disco**

- Necesitamos algún mecanismo para **restaurar la consistencia** en caso de fallas

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Problema: Memoria vs Disco



- **INPUT(X)**: Copia el bloque de disco que contiene el ítem X a un buffer de memoria.
- **READ(X, t)**: Copia el ítem X del buffer de memoria a la variable temporal t (memoria local de la transacción).
- **WRITE(X, t)**: Copia el valor de la variable local t al ítem X en el pool buffers de memoria.
- **OUTPUT(X)**: Copia el bloque que contiene el ítem X del buffer de memoria al disco.
- **FLUSH LOG:** Comando emitido por el Log Manager para que el Buffer Manager fuerce la escritura de los registros de Log al disco.

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Ejemplo

#	T	buffer manager	t	Mem A	Mem B	Disco A	Disco B
1		INPUT(A)		**8**		8	8
2	READ(A,t)		**8**	8		8	8
3	t := t*2		**16**	8		8	8
4	WRITE(A,t)		16	**16**		8	8
5		INPUT(B)	16	16	**8**	8	8
6	READ(B,t)		**8**	16	8	8	8
7	t := t*2		**16**	16	8	8	8
8	WRITE(B,t)		16	16	**16**	8	8
9		OUTPUT(A)	16	16	16	**16**	8
10	COMMIT			16	16	16	8
11		OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si se corta la luz entre las operaciones 9 y 10?
- ¿Y si se corta entre la 10 y la 11?
- Y este ejemplo es con una única transacción, ¿Qué pasa si tenemos varias en simultáneo?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

El Log

- Archivo *append-only*
- Lleva un registro de todos los cambios realizados sobre la base
- Lo vamos a pensar como una **secuencia de registros**
- Los registros indican **cambios** (escrituras) sobre la base de datos, más allá de si esos cambios se bajaron a disco o no
- Cada transacción puede generar varios registros
- Algunos registros son generados por el funcionamiento interno de la base

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Tipos de Registros

- Start record: < START T_i >
- Commit record: < COMMIT T_i >
- Abort record: < ABORT T_i >
  - El primer registro de cualquier transacción T_i debería ser un Start record y el último un Commit o un Abort record
  - Una transacción está **incompleta** si hay un Start record en el log pero no hay ni un Commit ni un Abort record posterior
  - Entre el Start record y su correspondiente Commit o Abort record, puede haber cualquier cantidad de Update record
- Update record: La estructura depende del método de Logging utilizado

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

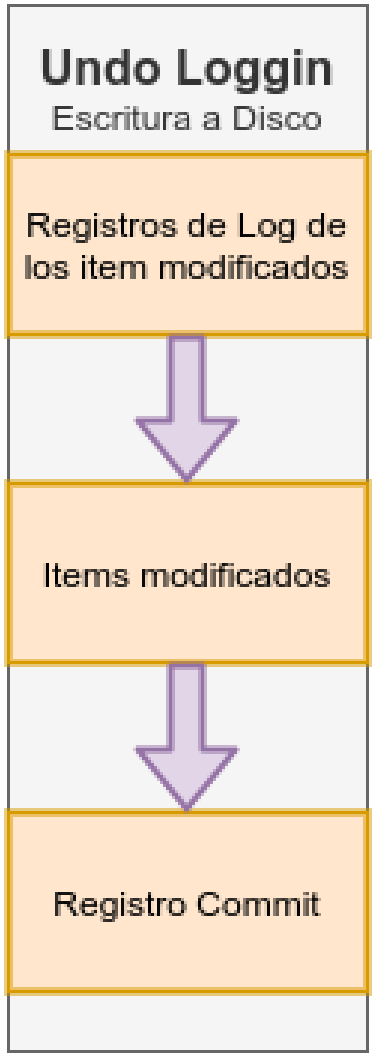
Métodos de Logging

- El método de Logging determina qué información se guarda en el Log tras cada modificación y qué procedimiento se lleva a cabo para recuperarse tras una falla
  - Undo Logging: **Deshacer** los cambios realizados por **transacciones que no llegaron a comitear** antes de la falla
  - Redo Logging: **Rehacer** los **cambios comiteados** que no se bajaron a disco antes de la falla
  - Undo/Redo Logging: Una mezcla de ambos.

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO Logging

- Forma de los Update Record: $< T_i, X, v >$
  - La transacción T_i actualizó el valor del registro X que **antes valía** v
- Reglas:
  - Cada Update Record $< T_i, X, v >$ se baja a disco **antes** de bajar a disco el nuevo valor de X
  - Cada Commit Record $<$ COMMIT $T_i >$ se baja a disco **después** de bajar a disco todos los cambios realizados por T_i
- Una vez que los cambios estén asegurados en el disco, registrar en el log que esos cambios son válidos (commit)
  - Puede haber cambios en el disco que aún no haya registrado como válidos (commit) en el log



UNDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)
1	<START T>						8
2			INPUT(A)		**8**		8
3		READ(A,t)		**8**	8		8
4		t := t*2		**16**	8		8
5	<T, A, 8>	WRITE(A,t)		16	**16**		8
6			INPUT(B)	16	16	**8**	8
7		READ(B,t)		**8**	16	8	8
8		t := t*2		**16**	16	8	8
9	<T, B, 8>	WRITE(B,t)		16	16	**16**	8
10			OUTPUT(A)	16	16	16	**16**
11			OUTPUT(B)		16	16	16
12	<COMMIT T>	COMMIT			16	16	16

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)	16	16	16	**16**	8
11			OUTPUT(B)		16	16	16	**16**
12	<COMMIT T>	COMMIT			16	16	16	16

- ¿Qué pasa si hay una falla tras el paso ....?
  - 12:

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)	16	16	16	**16**	8
11			OUTPUT(B)		16	16	16	**16**
12	<COMMIT T>	COMMIT			16	16	16	16

- ¿Qué pasa si hay una falla tras el paso ....?
  - 12: No hay que hacer nada, la base quedó consistente
  - 11:

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)	16	16	16	**16**	8
11			OUTPUT(B)		16	16	16	**16**
12	<COMMIT T>	COMMIT			16	16	16	16

- ¿Qué pasa si hay una falla tras el paso ....?
  - 12: No hay que hacer nada, la base quedó consistente
  - 11: La transacción quedó incompleta, deshacer los cambios
    - Escribir 8 para A y B en el disco, agregar < ABORT T > al log y bajarlo al disco
  - 9:

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)	16	16	16	**16**	8
11			OUTPUT(B)		16	16	16	**16**
12	<COMMIT T>	COMMIT			16	16	16	16

- ¿Qué pasa si hay una falla tras el paso ....?
  - 12: No hay que hacer nada, la base quedó consistente
  - 11: La transacción quedó incompleta, deshacer los cambios
    - Escribir 8 para A y B en el disco, agregar < ABORT T > al log y bajarlo al disco
  - 9: Ídem anterior
    - En este caso, la base quedó consistente pero podría no ser así...

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

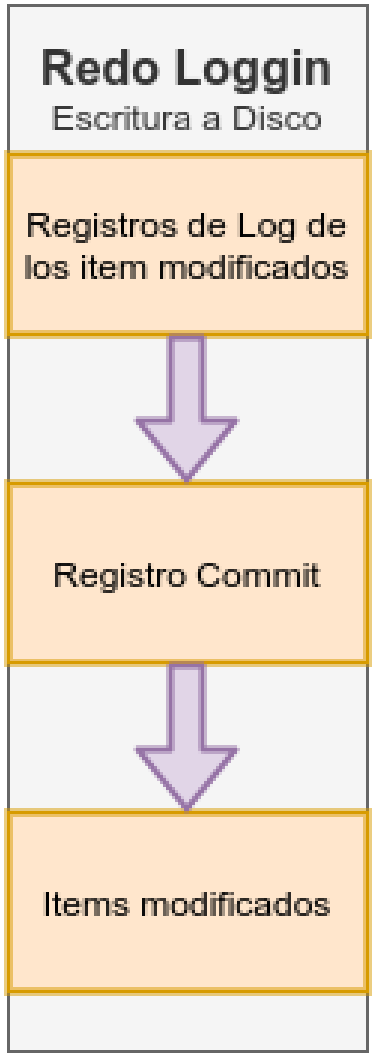
UNDO Logging - Recuperación

- **Idea**: Identificar las transacciones incompletas y deshacer los cambios que realizaron en el disco
  - Recorrer el log desde el final hacia atrás
  - Para cada transacción T_i, recordar si se encuentra un Commit Record o un Abort Record
  - Por cada Update Record $< T_i, X, v >$,
    - Si se encontró un Commit Record o un Abort Record para T_i, no hacer nada
    - Si no, T_i es una transacción incompleta y hay que deshacer el cambio: Asignar v a X
  - Por cada T_i incompleta agregar un Abort Record al log y bajarlo al disco

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

REDO Logging

- Forma de los Update Record: < T_i, X, v >
  - La transacción T_i actualizó el valor del registro X que **pasa a valer** v
- Reglas:
  - Cada Update Record < T_i, X, v > se baja a disco **antes** de bajar a disco el nuevo valor de X
  - Cada Commit Record < COMMIT T_i > se baja a disco **antes** de bajar a disco cualquiera de los cambios realizados por T_i
- Primero asegurar los cambios en el log (commit), luego escribirlos en el disco
  - Las actualizaciones en el log no necesariamente quedaron reflejadas en el disco



REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)
1	<START T>						8
2			INPUT(A)		**8**		8
3		READ(A,t)		**8**	8		8
4		t := t*2		**16**	8		8
5	<T, A, 16>	WRITE(A,t)		16	**16**		8
6			INPUT(B)	16	16	**8**	8
7		READ(B,t)		**8**	16	8	8
8		t := t*2		**16**	16	8	8
9	<T, B, 16>	WRITE(B,t)		16	16	**16**	8
10	<COMMIT T>	COMMIT			16	16	8
11			OUTPUT(A)		16	16	**16**
12			OUTPUT(B)		16	16	16

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 16>	WRITE(B,t)		16	16	**16**	8	8
10	<COMMIT T>	COMMIT			16	16	8	8
11			OUTPUT(A)		16	16	**16**	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso ....?
  - 11:

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 16>	WRITE(B,t)		16	16	**16**	8	8
10	<COMMIT T>	COMMIT			16	16	8	8
11			OUTPUT(A)		16	16	**16**	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso ....?
  - 11: Rehacer: Escribir 16 para A y B en el disco
    - Los cambios no se bajaron a disco pero la transacción comiteó
  - 12:

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 16>	WRITE(B,t)		16	16	**16**	8	8
10	<COMMIT T>	COMMIT			16	16	8	8
11			OUTPUT(A)		16	16	**16**	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso ....?
  - 11: Rehacer: Escribir 16 para A y B en el disco
    - Los cambios no se bajaron a disco pero la transacción comiteó
  - 12: Ídem anterior
    - En este caso, la base quedó consistente pero podría no ser así...
  - 9:

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 16>	WRITE(B,t)		16	16	**16**	8	8
10	<COMMIT T>	COMMIT			16	16	8	8
11			OUTPUT(A)		16	16	**16**	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso ....?
  - 11: Rehacer: Escribir 16 para A y B en el disco
    - Los cambios no se bajaron a disco pero la transacción comiteó
  - 12: Ídem anterior
    - En este caso, la base quedó consistente pero podría no ser así...
  - 9: No modificar el disco pero abortar la transacción
    - Agregar < ABORT T > al log y bajarlo a disco

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

REDO Logging - Recuperación

- **Idea**: Identificar las transacciones completas y rehacer los cambios que no llegaron a bajarse al disco
  - Recorrer el log una primera vez para identificar cuáles son las transacciones completas y cuáles las incompletas
  - Recorrer el log una segunda vez, desde el principio hacia adelante
  - Por cada Update Record $< T_i, X, v >$,
    - Si T_i es incompleta (en la primera pasada no se encontró un Commit Record ni un Abort Record para T_i), no hacer nada
    - Si no, T_i es una transacción completa y hay que actualizar el cambio en el disco: Asignar v a X
  - Por cada T_i incompleta agregar un Abort Record al log y bajarlo al disco

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging

- Forma de los Update Record: $< T_i, X, v_0, v_1 >$
  - La transacción T_i actualizó el valor del registro X que **antes valía** v_0 y **pasa a valer** v_1
- Reglas:
  - Cada Update Record $< T_i, X, v_0, v_1 >$ se baja a disco **antes** de bajar a disco el nuevo valor de X
- Cada vez que se va a cambiar algo en disco, primero grabar el registro de dicho cambio en el disco

UNDO/REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)
1	<START T>						8
2			INPUT(A)		**8**		8
3		READ(A,t)		**8**	8		8
4		t := t*2		**16**	8		8
5	<T, A, 8, 16>	WRITE(A,t)		16	**16**		8
6			INPUT(B)	16	16	**8**	8
7		READ(B,t)		**8**	16	8	8
8		t := t*2		**16**	16	8	8
9	<T, B, 8, 16>	WRITE(B,t)		16	16	**16**	8
10			OUTPUT(A)		16	16	**16**
11	<COMMIT T>	COMMIT			16	16	16
12			OUTPUT(B)		16	16	16

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8, 16>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)		16	16	**16**	8
11	<COMMIT T>	COMMIT			16	16	16	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso 11?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8, 16>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)		16	16	**16**	8
11	<COMMIT T>	COMMIT			16	16	16	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso 11?
  - Si el < COMMIT T > llegó a bajarse al disco,

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8, 16>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)		16	16	**16**	8
11	<COMMIT T>	COMMIT			16	16	16	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso 11?
  - Si el < COMMIT T > llegó a bajarse al disco, rehacer
    - Escribir 16 para A y B en el disco (REDO)
    - Si no,

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8, 16>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)		16	16	**16**	8
11	<COMMIT T>	COMMIT			16	16	16	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso 11?
  - Si el < COMMIT T > llegó a bajarse al disco, rehacer
    - Escribir 16 para A y B en el disco (REDO)
    - Si no, deshacer y abortar
- ¿Qué pasa si hay una falla tras el paso 8?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging - Ejemplo

#	Log	T	buffer manager	t	M(A)	M(B)	D(A)	D(B)
1	<START T>						8	8
2			INPUT(A)		**8**		8	8
3		READ(A,t)		**8**	8		8	8
4		t := t*2		**16**	8		8	8
5	<T, A, 8, 16>	WRITE(A,t)		16	**16**		8	8
6			INPUT(B)	16	16	**8**	8	8
7		READ(B,t)		**8**	16	8	8	8
8		t := t*2		**16**	16	8	8	8
9	<T, B, 8, 16>	WRITE(B,t)		16	16	**16**	8	8
10			OUTPUT(A)		16	16	**16**	8
11	<COMMIT T>	COMMIT			16	16	16	8
12			OUTPUT(B)		16	16	16	**16**

- ¿Qué pasa si hay una falla tras el paso 11?
  - Si el < COMMIT T > llegó a bajarse al disco, rehacer
    - Escribir 16 para A y B en el disco (REDO)
    - Si no, deshacer y abortar
- ¿Qué pasa si hay una falla tras el paso 8?
  - Escribir 8 para A y B en el disco (UNDO)
  - Agregar < ABORT T > al log y bajarlo a disco

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

UNDO/REDO Logging - Recuperación

- Aplicar **UNDO** a las transacciones incompletas en orden inverso
  - Recorrer el log desde el final hacia atrás
  - Para cada transacción T_i, recordar si se encuentra un Commit Record o un Abort Record
  - Por cada Update Record $< T_i, X, v_0, v_1 >$,
    - Si se encontró un Commit Record o un Abort Record para T_i, no hacer nada
    - Si no, T_i es una transacción incompleta y hay que deshacer el cambio: Asignar v_0 a X
- Aplicar **REDO** a las transacciones comiteadas en orden
  - Recorrer el log una segunda vez, desde el principio hacia adelante
  - Por cada Update Record $< T_i, X, v_0, v_1 >$,
    - Si T_i es incompleta (en la primera pasada no se encontró un Commit Record ni un Abort Record para T_i), no hacer nada
    - Si no, T_i es una transacción completa y hay que actualizar el cambio en el disco: Asignar v a X
- Por cada T_i incompleta agregar un Abort Record al log y bajarlo al disco

Un ejemplo de la vida real...

- Tenemos una base de datos que estuvo funcionando sin parar durante 1 año
- La base de datos ejecuta al rededor de 10.000 transacciones por día
- Utiliza REDO logging

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Un ejemplo de la vida real...

- Tenemos una base de datos que estuvo funcionando sin parar durante 1 año
- La base de datos ejecuta al rededor de 10.000 transacciones por día
- Utiliza REDO logging

- Se cayó el sistema, hay que recuperarlo.

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Un ejemplo de la vida real...

- Tenemos una base de datos que estuvo funcionando sin parar durante 1 año
- La base de datos ejecuta al rededor de 10.000 transacciones por día
- Utiliza REDO logging

- Se cayó el sistema, hay que recuperarlo.

- ¿Cuánto pesa el log? ¿Cuántos registros tiene?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Un ejemplo de la vida real...

- Tenemos una base de datos que estuvo funcionando sin parar durante 1 año
- La base de datos ejecuta al rededor de 10.000 transacciones por día
- Utiliza REDO logging

- Se cayó el sistema, hay que recuperarlo.

- ¿Cuánto pesa el log? ¿Cuántos registros tiene?

- ¿Tiene sentido empezar a rehacer los cambios de las transacciones que se efectuaron hace un año?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint

- Quiescente: Impide que ingresen nuevas transacciones hasta completar el checkpoint
- No quiescente: Permite que ingresen nuevas transacciones mientras se realiza el checkpoint

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint Quiescente con UNDO Logging

- Etapas:
  1. Dejar de aceptar nuevas transacciones
  2. Esperar a que todas las transacciones activas (aquellas con Start Record pero sin Commit ni Abort Record) comitéen o aborten
  3. Agregar el registro < CKPT > al log y bajarlo a disco
  4. Volver a aceptar nuevas transacciones
- Recuperación: Aplicar UNDO desde el final del log hasta el último checkpoint (el primero encontrado)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint Quiescente con UNDO Logging

< START T_1 >
< T_1, A, 5 >
< START T_2 >
< T_2, B, 10 >
< T_2, C, 15 >
< T_1, D, 20 >
< COMMIT T_1 >
< ABORT T_2 >
< **CKPT** >
< START T_3 >
< T_3, E, 25 >

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint Quiescente con UNDO Logging

< START T_1 >
< T_1, A, 5 >
< START T_2 >
< T_2, B, 10 >
< T_2, C, 15 >
< T_1, D, 20 >
< COMMIT T_1 >
< ABORT T_2 >
< **CKPT** >
< START T_3 >
< T_3, E, 25 >

- Nuevo tipo de registro < CKPT > que se agrega al log cuando se realiza un checkpoint

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint Quiescente con UNDO Logging

< START T_1 >
< T_1, A, 5 >
< START T_2 >
< T_2, B, 10 >
< T_2, C, 15 >
< T_1, D, 20 >
< COMMIT T_1 >
< ABORT T_2 >
< **CKPT** >
< START T_3 >
< T_3, E, 25 >

- Nuevo tipo de registro < CKPT > que se agrega al log cuando se realiza un checkpoint
- ¿Cómo aplicamos UNDO ahora?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint Quiescente con UNDO Logging

< START T_1 >
< T_1, A, 5 >
< START T_2 >
< T_2, B, 10 >
< T_2, C, 15 >
< T_1, D, 20 >
< COMMIT T_1 >
< ABORT T_2 >
< **CKPT** >
< START T_3 >
< T_3, E, 25 >

- Nuevo tipo de registro < CKPT > que se agrega al log cuando se realiza un checkpoint
- ¿Cómo aplicamos UNDO ahora?
  - Deshacer T_3 (escribir 25 en E) y abortar (agregar < ABORT T_3 > al log y bajarlo al disco)
- ¿El checkpoint puede aparecer en cualquier momento?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint Quiescente con UNDO Logging

< START T_1 >
< T_1, A, 5 >
< START T_2 >
< T_2, B, 10 >
< T_2, C, 15 >
< T_1, D, 20 >
< COMMIT T_1 >
< ABORT T_2 >
< **CKPT** >
< START T_3 >
< T_3, E, 25 >

- Nuevo tipo de registro < CKPT > que se agrega al log cuando se realiza un checkpoint
- ¿Cómo aplicamos UNDO ahora?
  - Deshacer T_3 (escribir 25 en E) y abortar (agregar < ABORT T_3 > al log y bajarlo al disco)
- ¿El checkpoint puede aparecer en cualquier momento?
  - No, sólo cuando no hay transacciones activas
  - Para Redo tiene el problema de cuanto tiempo esperar para agregar el registro <CKPT>

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

- Etapas:
  1. Agregar el registro < START CKPT(T_1, T_2, ..., T_k) > al log y bajarlo al disco (siendo T_1, T_2, ..., T_k las transacciones activas en el momento)
  2. Esperar a que todas las transacciones T_1, T_2, ..., T_k comitéen o aborten, pero sin restringir que empiecen nuevas transacciones
  3. Agregar el registro < END CKPT > al log y bajarlo a disco
- Recuperación:
  - Aplicar UNDO desde el final del log
  - Si se encuentra un < END CKPT >, seguir hasta el < START CKPT(...) >
  - Si se encuentra un < START CKPT(T_1, T_2, ..., T_k) >, seguir hasta el < START T_i > más antiguo, con $1 \leq i \leq k$

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>
< ABORT T_1>
< T_3, E, 25>
< COMMIT T_2>
< **END CKPT**>
< T_3, F, 30>

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>
< ABORT T_1>
< T_3, E, 25>
< COMMIT T_2>
< **END CKPT**>
< T_3, F, 30>

- Nuevo tipo de registro < START CKPT (...) > que se agrega al log cuando se empieza un checkpoint
  - (...) todas las transacciones activas al momento, Ti.

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>
< ABORT T_1>
< T_3, E, 25>
< COMMIT T_2>
< **END CKPT**>
< T_3, F, 30>

- Nuevo tipo de registro < START CKPT (...) > que se agrega al log cuando se empieza un checkpoint
  - (...) todas las transacciones activas al momento, Ti.
- Nuevo tipo de registro < END CKPT > que se agrega al log cuando se finaliza un checkpoint
  - Todas las Ti deben estar terminadas.

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>
< ABORT T_1>
< T_3, E, 25>
< COMMIT T_2>
< **END CKPT**>
< T_3, F, 30>

- Nuevo tipo de registro < START CKPT (...) > que se agrega al log cuando se empieza un checkpoint
  - (...) todas las transacciones activas al momento, Ti.
- Nuevo tipo de registro < END CKPT > que se agrega al log cuando se finaliza un checkpoint
  - Todas las Ti deben estar terminadas.
- ¿Cómo aplicamos UNDO ahora?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>
< ABORT T_1>
< T_3, E, 25>
< COMMIT T_2>
< **END CKPT**>
< T_3, F, 30>

- Nuevo tipo de registro < START CKPT (...) > que se agrega al log cuando se empieza un checkpoint
  - (...) todas las transacciones activas al momento, Ti.
- Nuevo tipo de registro < END CKPT > que se agrega al log cuando se finaliza un checkpoint
  - Todas las Ti deben estar terminadas.
- ¿Cómo aplicamos UNDO ahora?
  - Deshacer T_3 (escribir F=30, E=25) y abortar (agregar < ABORT T_3 > al log y bajarlo al disco)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>

- ¿Y ahora?

Checkpoint No-Quiescente con UNDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< T_2, B, 10>
< **START CKPT T_1,T_2**>
< T_2, C, 15>
< START T_3>
< T_1, D, 20>

- ¿Y ahora?
- Todas las transacciones están incompletas, hay que deshacerlas y abortarlas
  - Escribir D=20, C=15, B=10, A=5
  - Agregar < ABORT T_1 >, < ABORT T_2 >, < ABORT T_3 > al log y bajarlo al disco

Checkpoint No-Quiescente con REDO Logging

- Etapas:
  1. Agregar el registro < START CKPT(T_1, T_2, ..., T_k) > al log y bajarlo al disco (siendo T_1, T_2, ..., T_k las transacciones activas en el momento)
  2. Esperar a que se bajen a disco todos los cambios realizados por transacciones **ya comiteadas al momento de iniciar** el checkpoint
  3. Agregar el registro < END CKPT > al log y bajarlo a disco
- Recuperación:
  - Identificar el último checkpoint que finalizó correctamente (el último < Start CKPT(...) > que tiene un < END CKPT > posterior)
  - Aplicar REDO desde el < START CKPT(...) > sobre las transacciones comiteadas que **iniciaron después de iniciar** el checkpoint (su correspondiente < START T > está después del < START CKPT(...) >)
  - Aplicar REDO desde el el inicio de las transacciones comiteadas que **estaban activas al iniciar** el checkpoint (aparece en el START CKPT)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con REDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< COMMIT T_1>
< T_2, B, 10>
< START CKPT T_2>
< T_2, C, 15>
< START T_3>
< T_3, D, 20>
< END CKPT>
< COMMIT T_2>
< COMMIT T_3>

- ¿Cómo aplicamos REDO ahora?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con REDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< COMMIT T_1>
< T_2, B, 10>
< START CKPT T_2>
< T_2, C, 15>
< START T_3>
< T_3, D, 20>
< END CKPT>
< COMMIT T_2>
< COMMIT T_3>

- ¿Cómo aplicamos REDO ahora?
  - Rehacer T_2 y T_3 (escribir B=10, C=15, D=20)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con REDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< COMMIT T_1>
< T_2, B, 10>
< START CKPT T_2>
< T_2, C, 15>

- ¿Y ahora?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con REDO Logging

< START T_1>
< T_1, A, 5>
< START T_2>
< COMMIT T_1>
< T_2, B, 10>
< START CKPT T_2>
< T_2, C, 15>

- ¿Y ahora?
  - Rehacer T_1 (escribir A=5) y abortar T_2 (agregar < ABORT T_2 > al log y bajarlo al disco)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO/REDO Logging

- Etapas:
  1. Agregar el registro < START CKPT(T_1, T_2, ..., T_k) > al log y bajarlo al disco (siendo T_1, T_2, ..., T_k las transacciones activas en el momento)
  2. Esperar a que se bajen a disco todos los cambios realizados **antes de iniciar** el checkpoint
  3. Agregar el registro < END CKPT > al log y bajarlo a disco
- Recuperación:
  - Deshacer transacciones incompletas
    - Aplicar UNDO desde el final del log
    - Si se encuentra un < START CKPT(T_1, T_2, ..., T_k) >, seguir hasta el < START T_i > más antiguo, con $1 \leq i \leq k$
  - Rehacer transacciones comiteadas
    - Identificar el último checkpoint que finalizó correctamente (el último < START CKPT(...) > que tiene un < END CKPT > posterior)
    - Aplicar REDO desde el < START CKPT(...) > sobre las transacciones comiteadas que **iniciaron después de iniciar** el checkpoint (su correspondiente < START T > está después del < START CKPT(...) >)

Las transacciones activas al iniciar el checkpoint también se rehacen

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO/REDO Logging

< START T_1>
< T_1, A, 4, 5>
< START T_2>
< START T_9>
< T_9, X, 9, 90>
< ABORT T_9>
< COMMIT T_1>
< T_2, B, 9, 10>
< START CKPT T_2>
< T_2, C, 14, 15>
< START T_3>
< T_3, D, 19, 20>
< END CKPT>
< COMMIT T_2>

- ¿Cómo aplicamos UNDO/REDO ahora?

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO/REDO Logging

< START T_1>
< T_1, A, 4, 5>
< START T_2>
< START T_9>
< T_9, X, 9, 90>
< ABORT T_9>
< COMMIT T_1>
< T_2, B, 9, 10>
< START CKPT T_2>
< T_2, C, 14, 15>
< START T_3>
< T_3, D, 19, 20>
< END CKPT>
< COMMIT T_2>

- ¿Cómo aplicamos UNDO/REDO ahora?
  - Deshacer T_3 (escribir D=19)
  - Abortar T_3 (agregar < ABORT T_3 > al log y bajarlo al disco)
  - Rehacer T_2 (escribir C=15)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Checkpoint No-Quiescente con UNDO/REDO Logging

< START T_1>
< T_1, A, 4, 5>
< START T_2>
< START T_9>
< T_9, X, 9, 90>
< ABORT T_9>
< COMMIT T_1>
< T_2, B, 9, 10>
< START CKPT T_2>
< T_2, C, 14, 15>

- ¿Y ahora?

Checkpoint No-Quiescente con UNDO/REDO Logging

< START T_1>
< T_1, A, 4, 5>
< START T_2>
< START T_9>
< T_9, X, 9, 90>
< ABORT T_9>
< COMMIT T_1>
< T_2, B, 9, 10>
< START CKPT T_2>
< T_2, C, 14, 15>

- ¿Y ahora?
  - Deshacer T_2 (escribir C=14, B=9)
  - Abortar T_2 (agregar < ABORT T_2 > al log y bajarlo al disco)
  - Rehacer T_1 (escribir A=5)

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

El Fin

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA