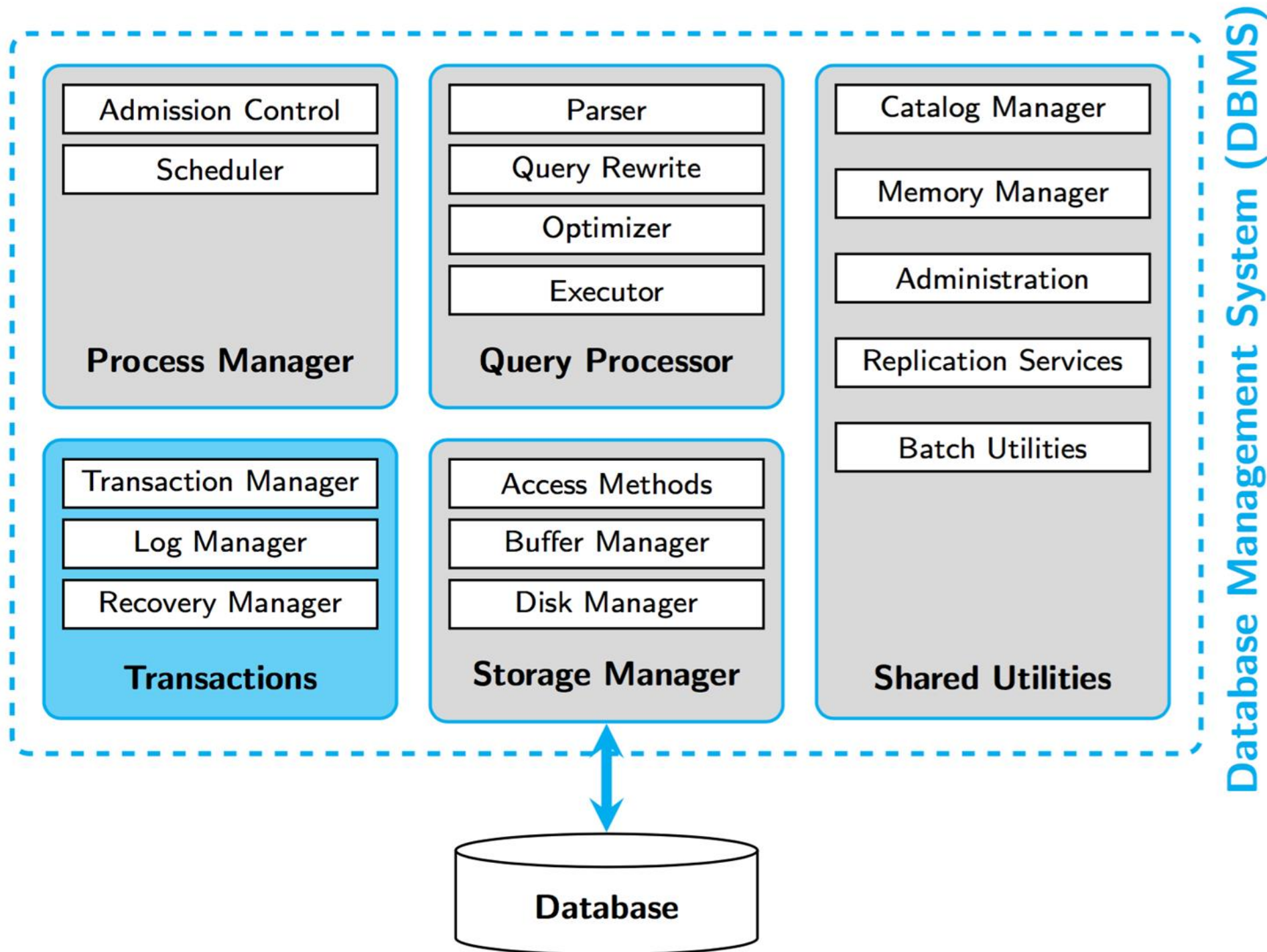


Bases de Datos

Clase 13: Recuperación de Fallas

Transactions



Transactions

Componente que asegura las propiedades **ACID**



Atomicity
Consistency
Isolation
Durability



Atomicity:

O se ejecutan todas las operaciones de la transacción, o no se ejecuta ninguna.



Consistency:

Cada transacción preserva la consistencia de la BD
(restricciones de integridad, etc.).



Isolation:

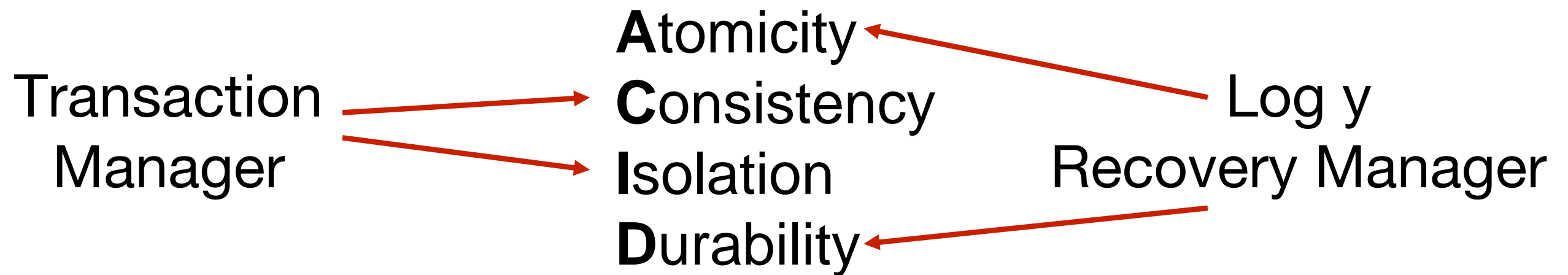
Cada transacción debe ejecutarse como si se estuviese ejecutando sola, de forma aislada.



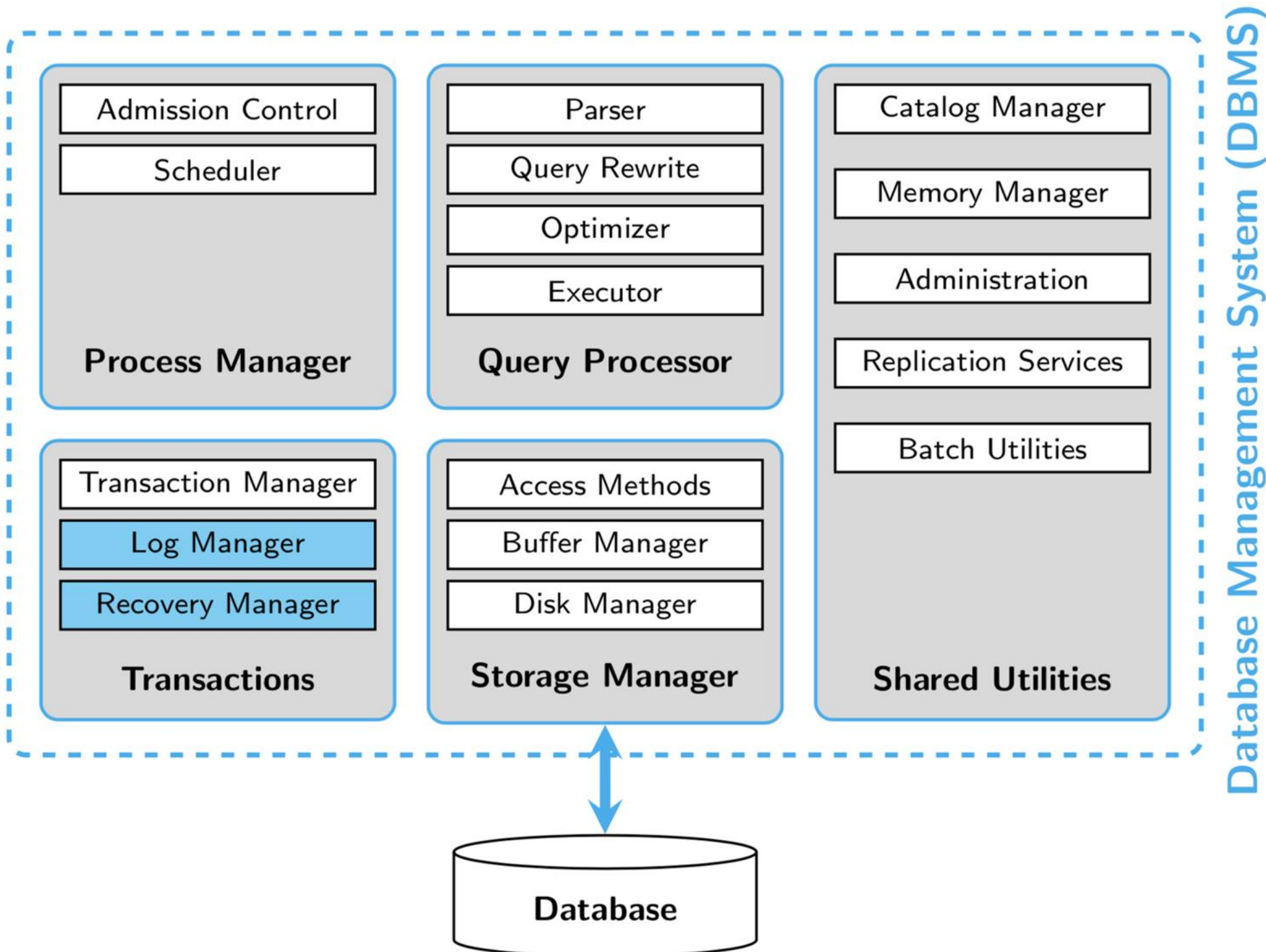
Durability:

Los cambios que hace cada transacción son permanentes en el tiempo, independiente de cualquier tipo de falla.

Asegurar las propiedades **ACID**



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

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 , todos *logs* $\langle \mathbf{T}, X, t \rangle$ deben ser escritos antes que el valor X sea escrito en disco

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

Undo Logging

En resumen:

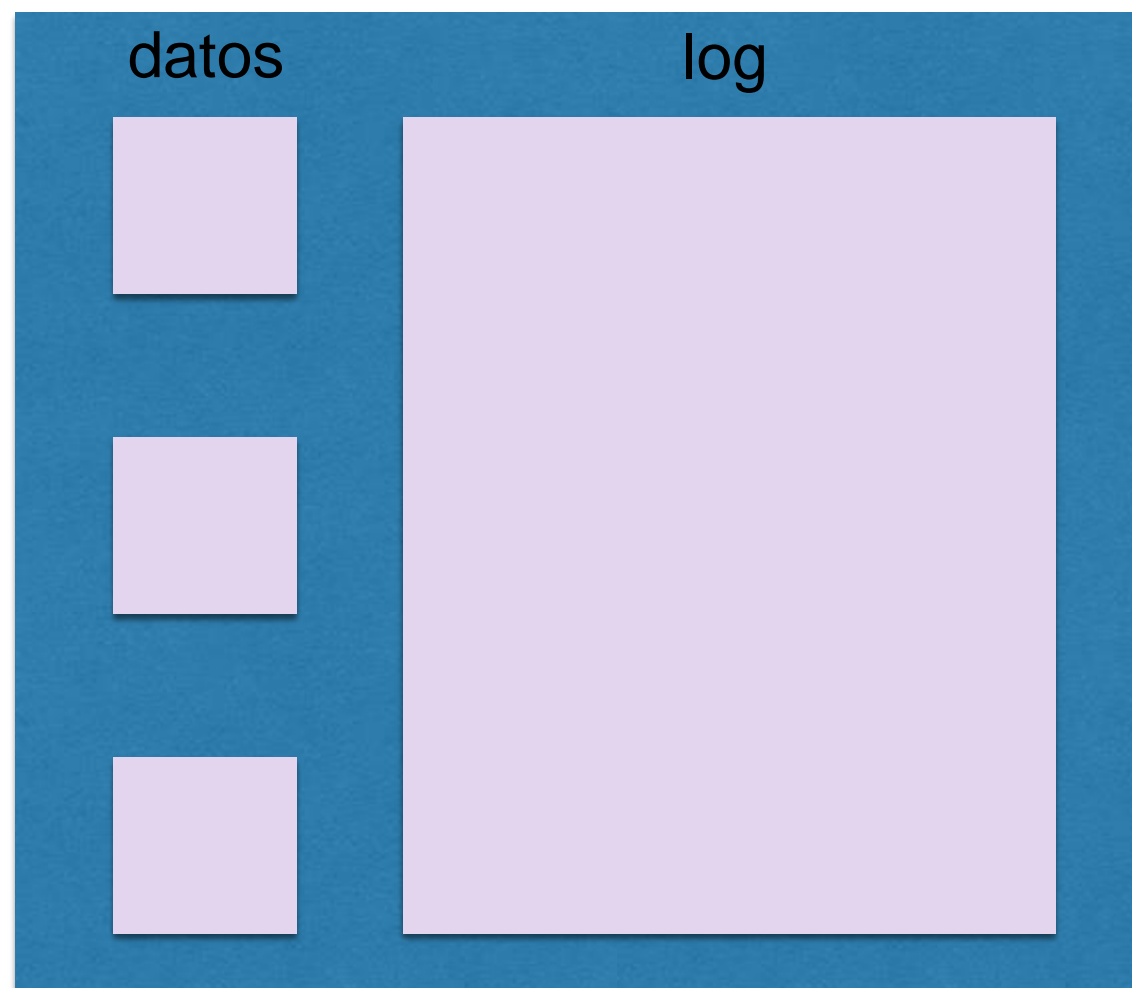
- **T** cambia el valor de **X** (t valor antiguo)
 - Generar el log $\langle \mathbf{T}, \mathbf{X}, t \rangle$
 - Escribir todos logs $\langle \mathbf{T}, \mathbf{X}, t \rangle$ al disco
 - Escribir valor nuevo de **X** a disco
 - Escribir $\langle \text{COMMIT } \mathbf{T} \rangle$
- } repeat
- } write-ahead logging

Undo Logging – en la BD

DB

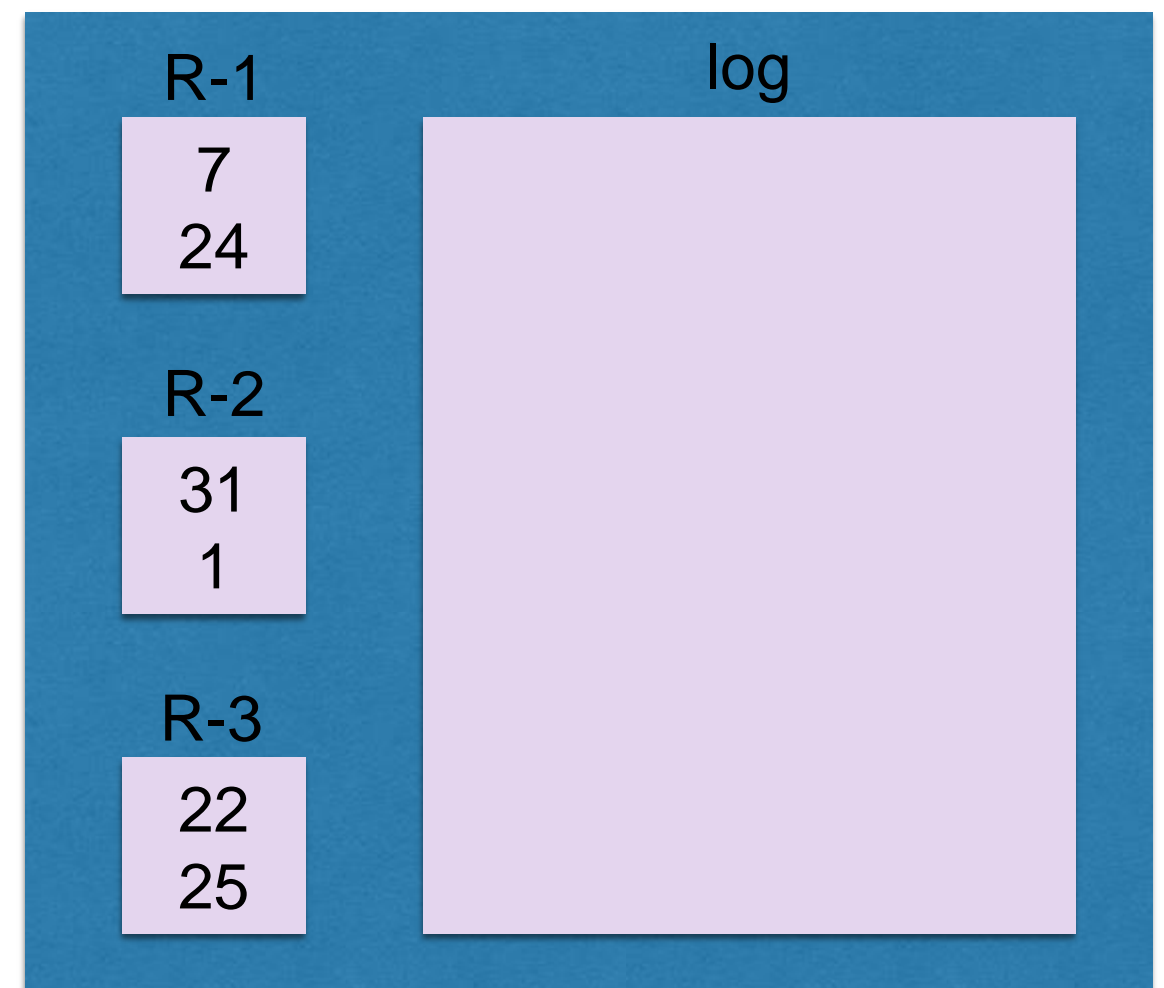
T1: voy a empezar

Buffer



R(A int)

Disco

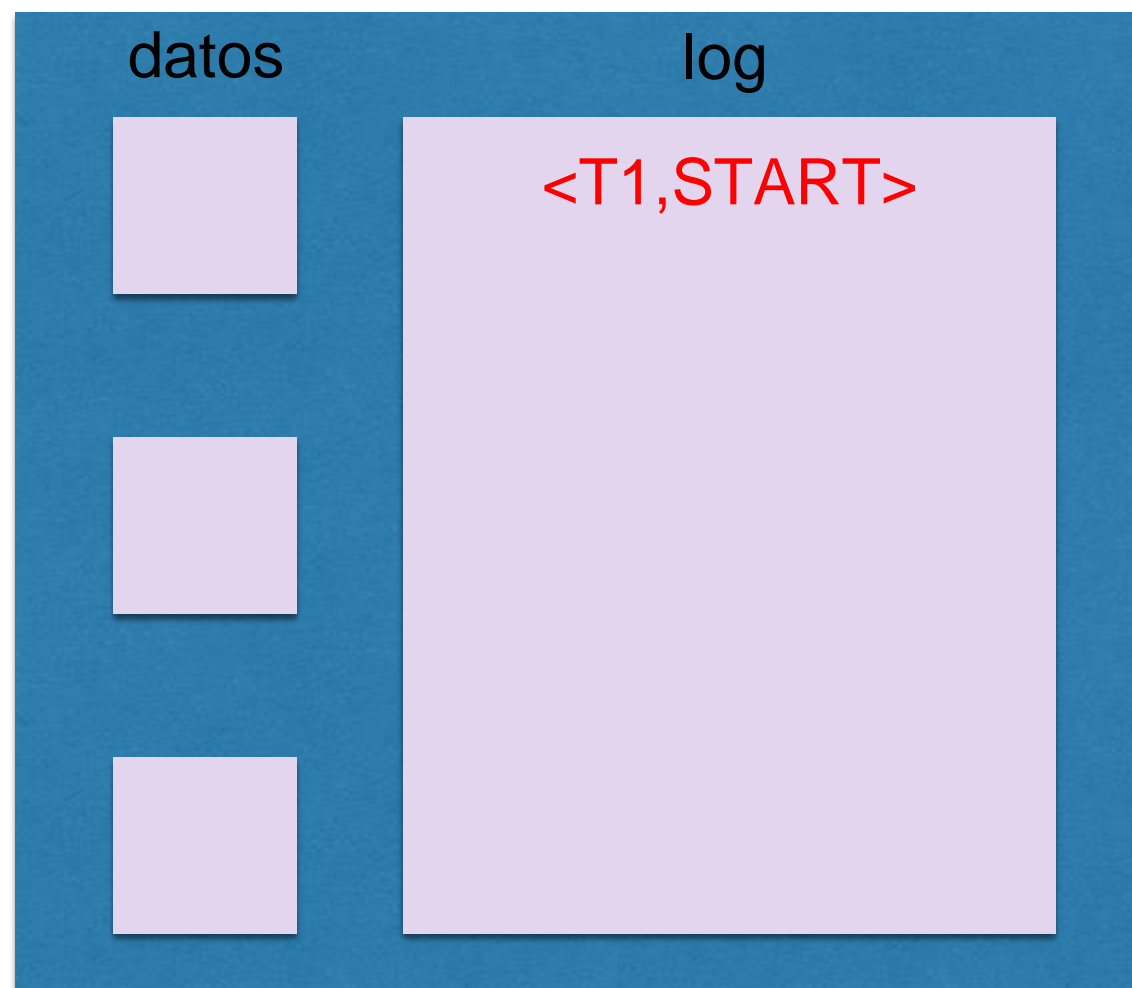


Undo Logging – en la BD

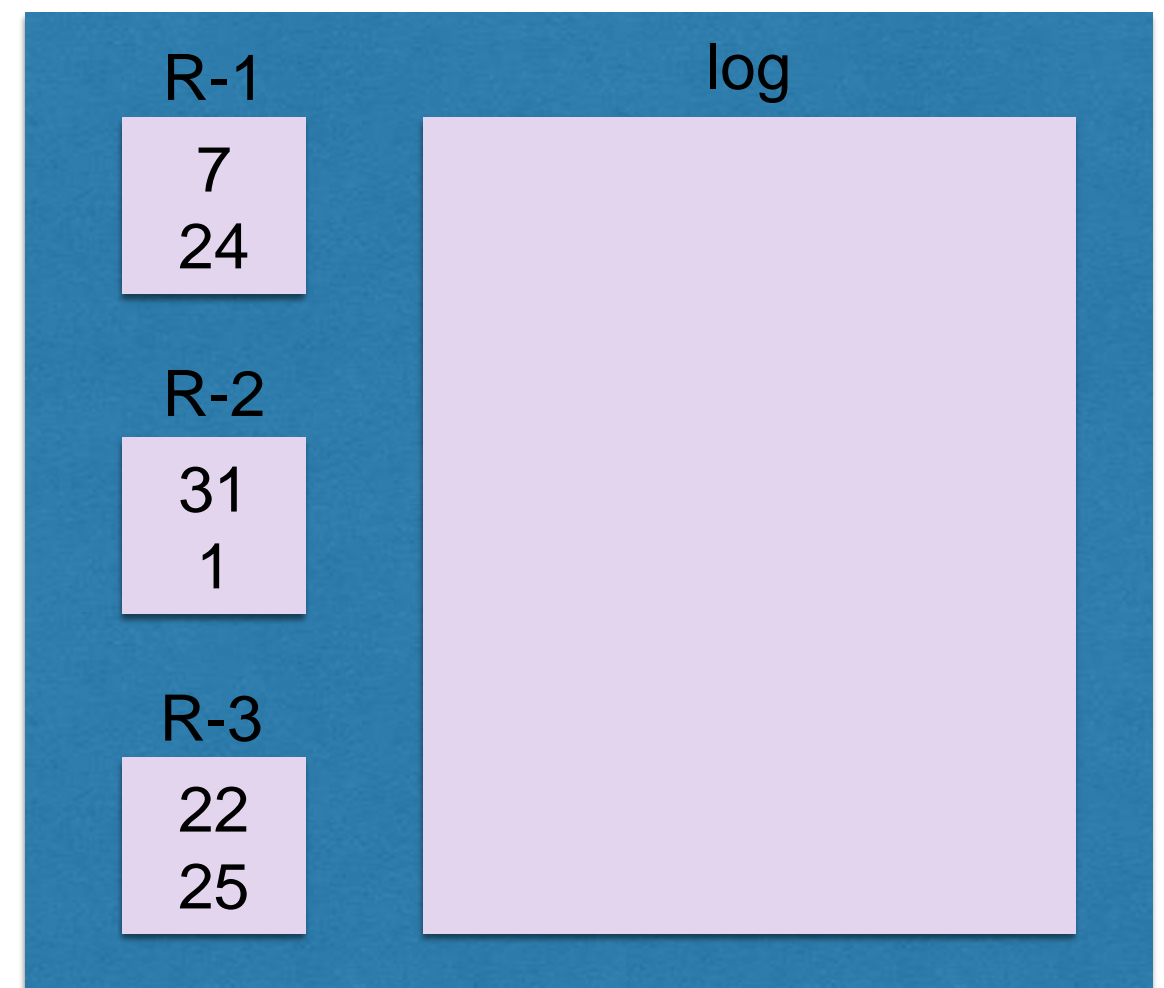
DB

T1: voy a empezar

Buffer



Disco

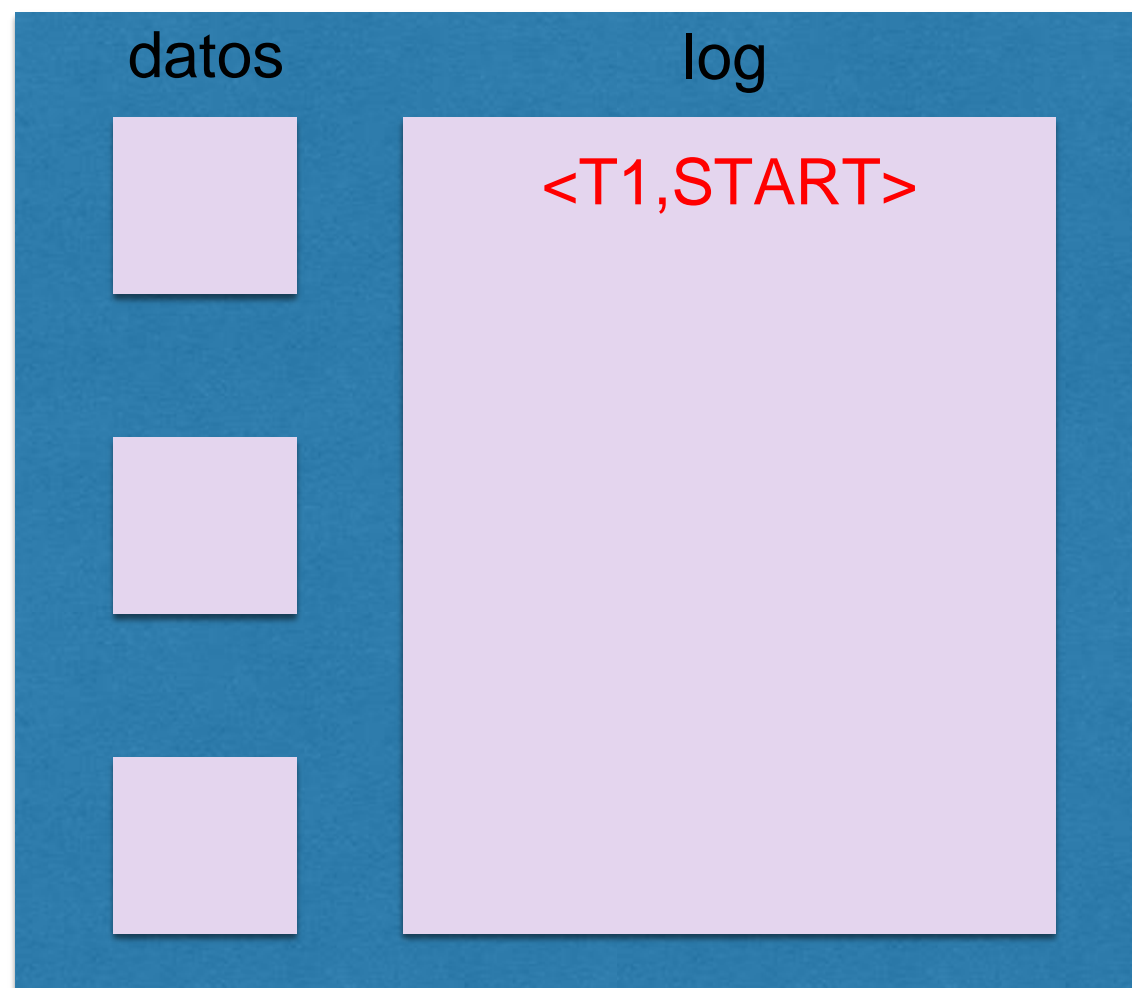


Undo Logging – en la BD

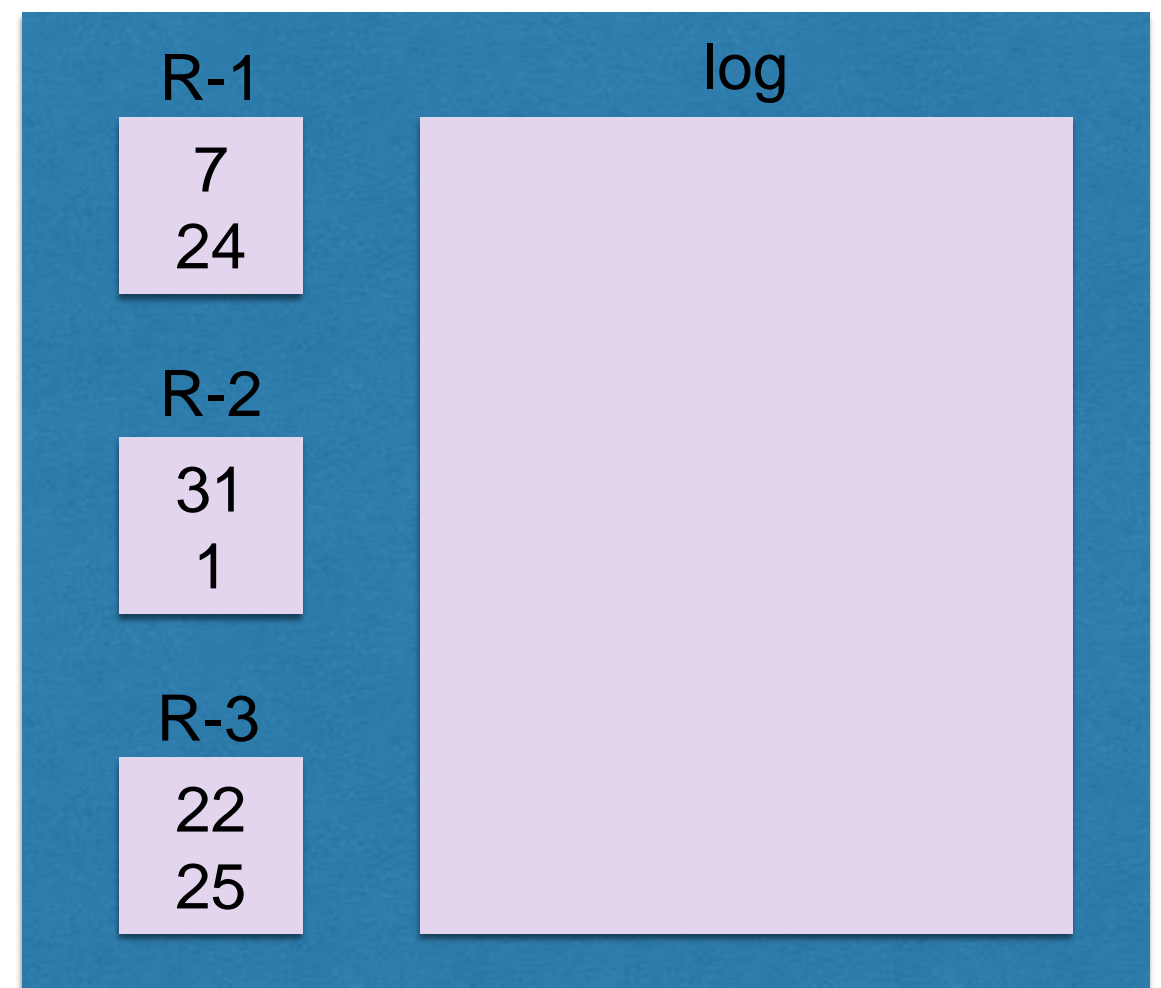
DB

T1: Necesito primera
tupla de página 2 de R!

Buffer



Disco

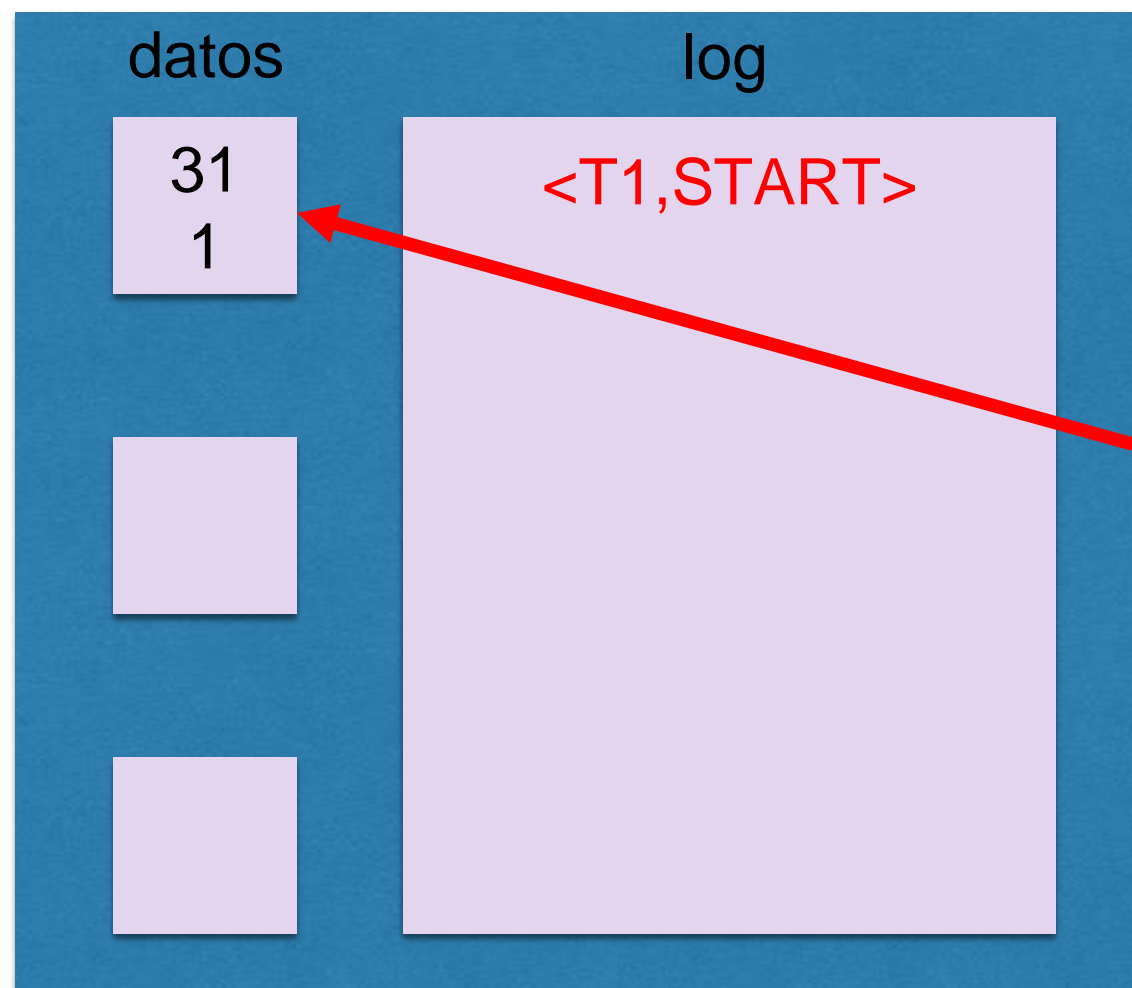


Undo Logging – en la BD

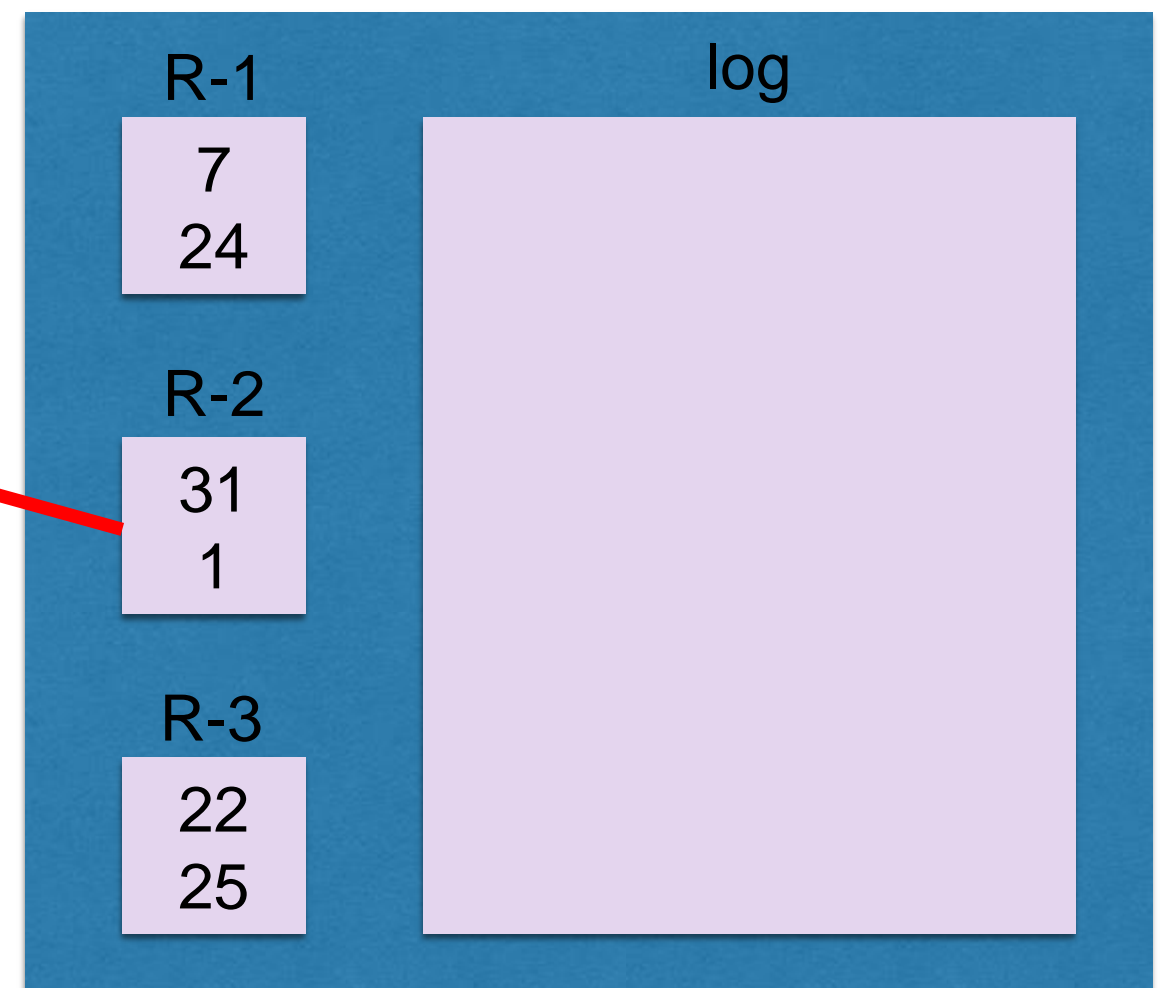
DB

T1: Necesito primera
tupla de página 2 de R!

Buffer



Disco

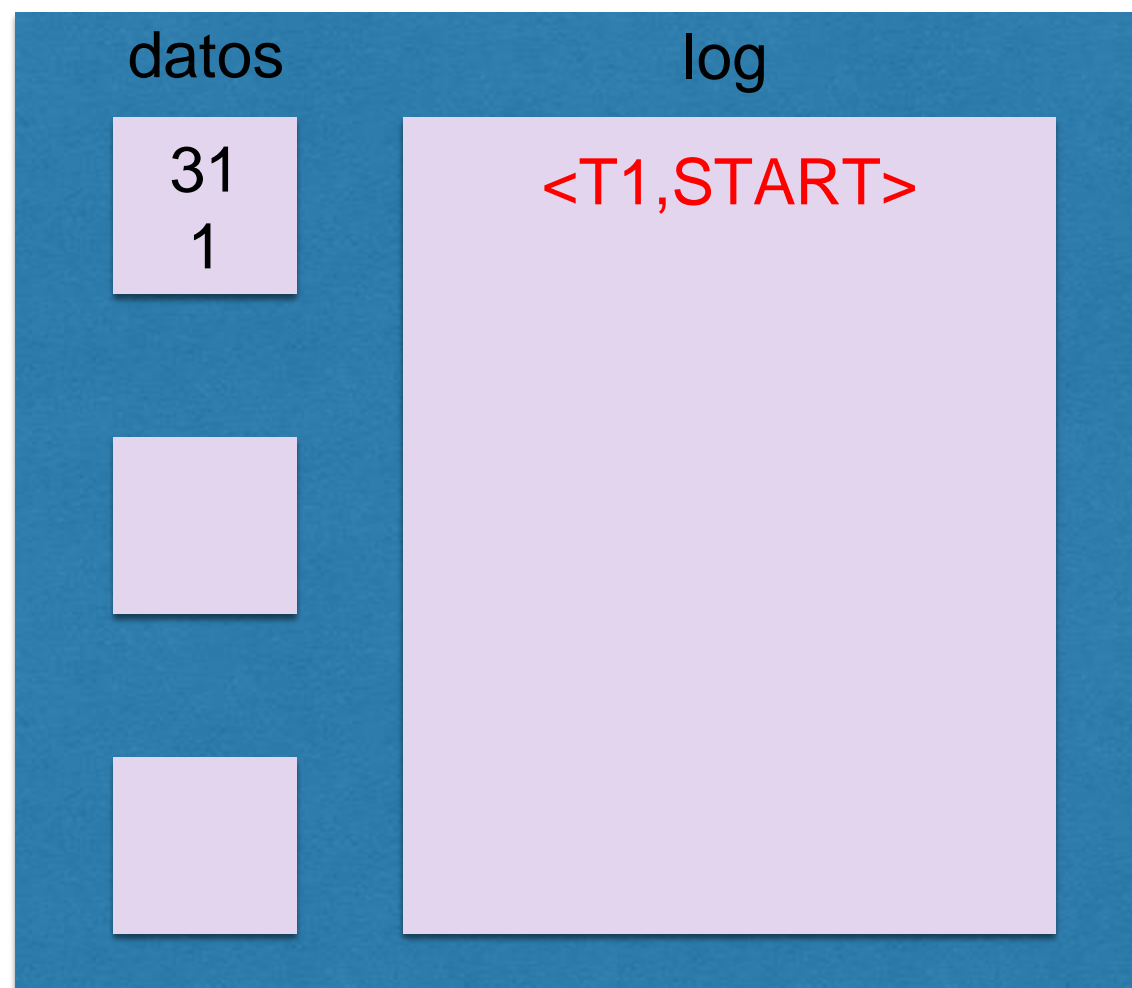


Undo Logging – en la BD

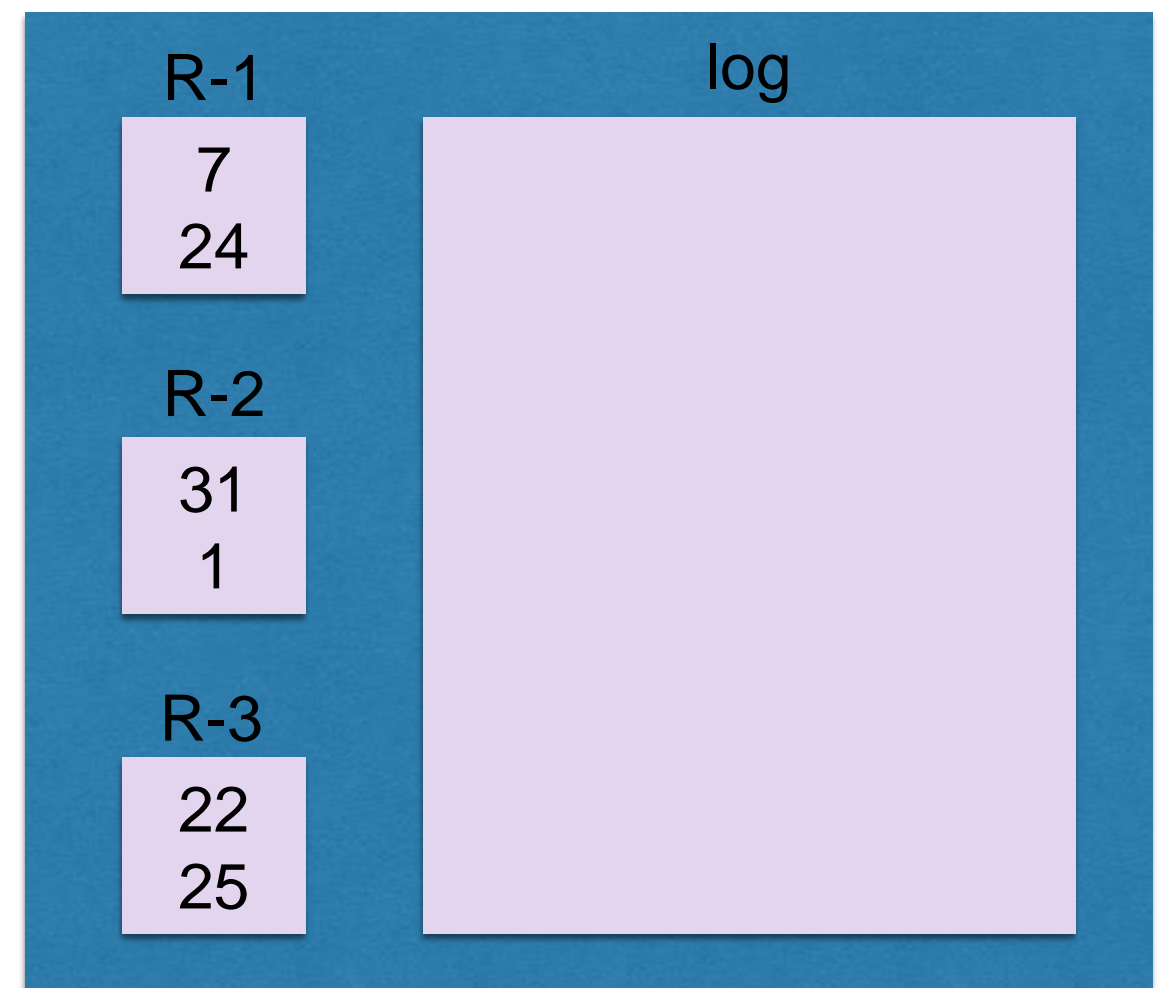
DB

T1: Cambio 31 a 99!

Buffer



Disco

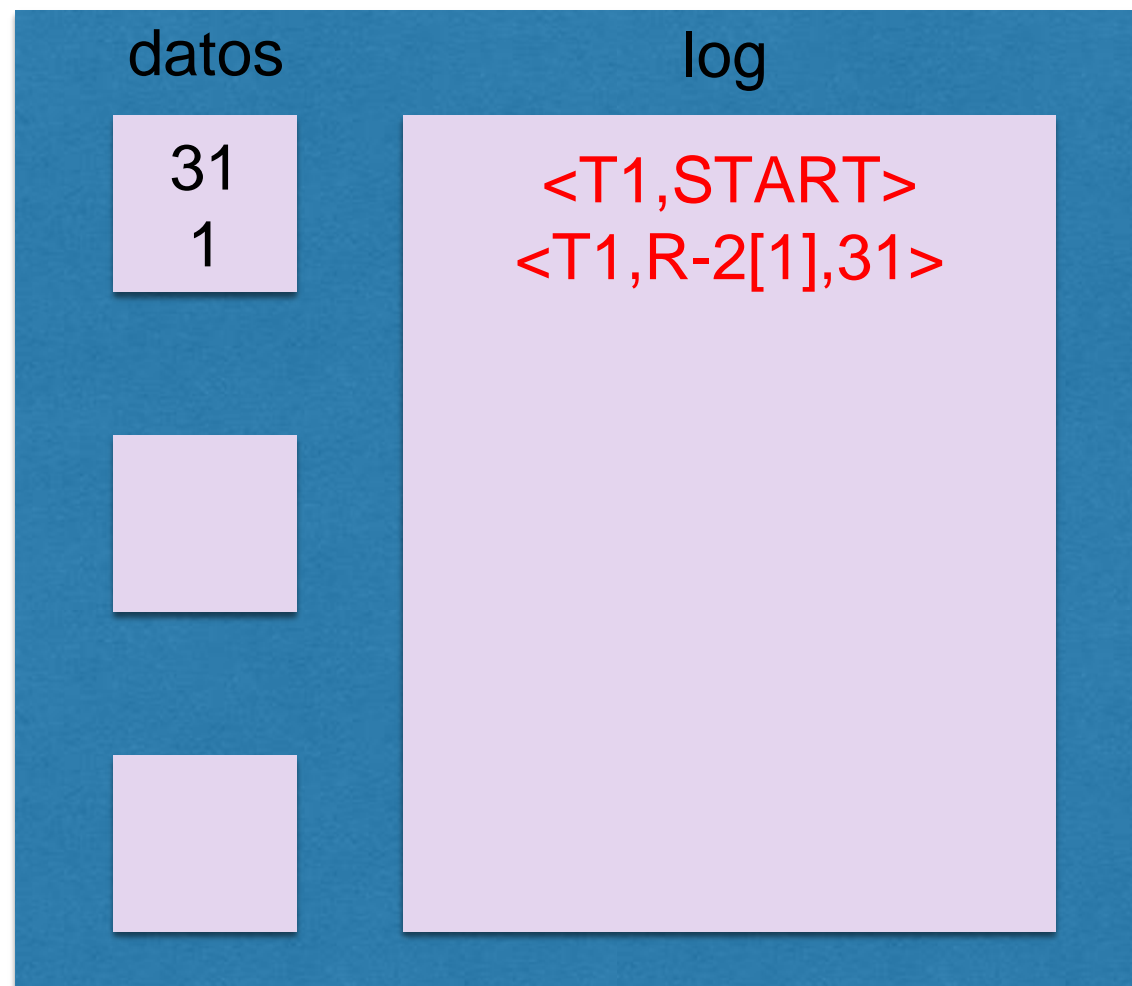


Undo Logging – en la BD

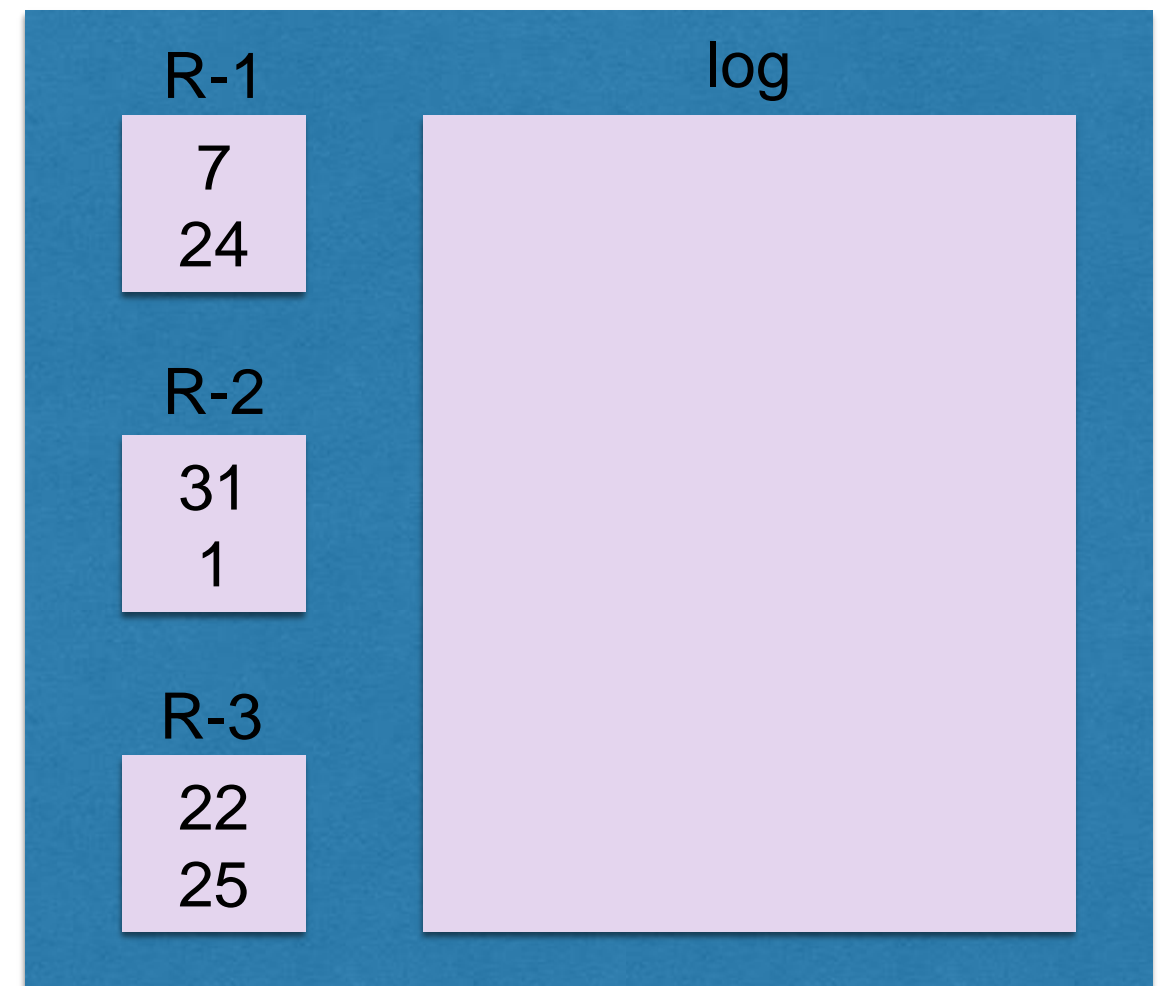
DB

T1: Cambio 31 a 99!

Buffer



Disco

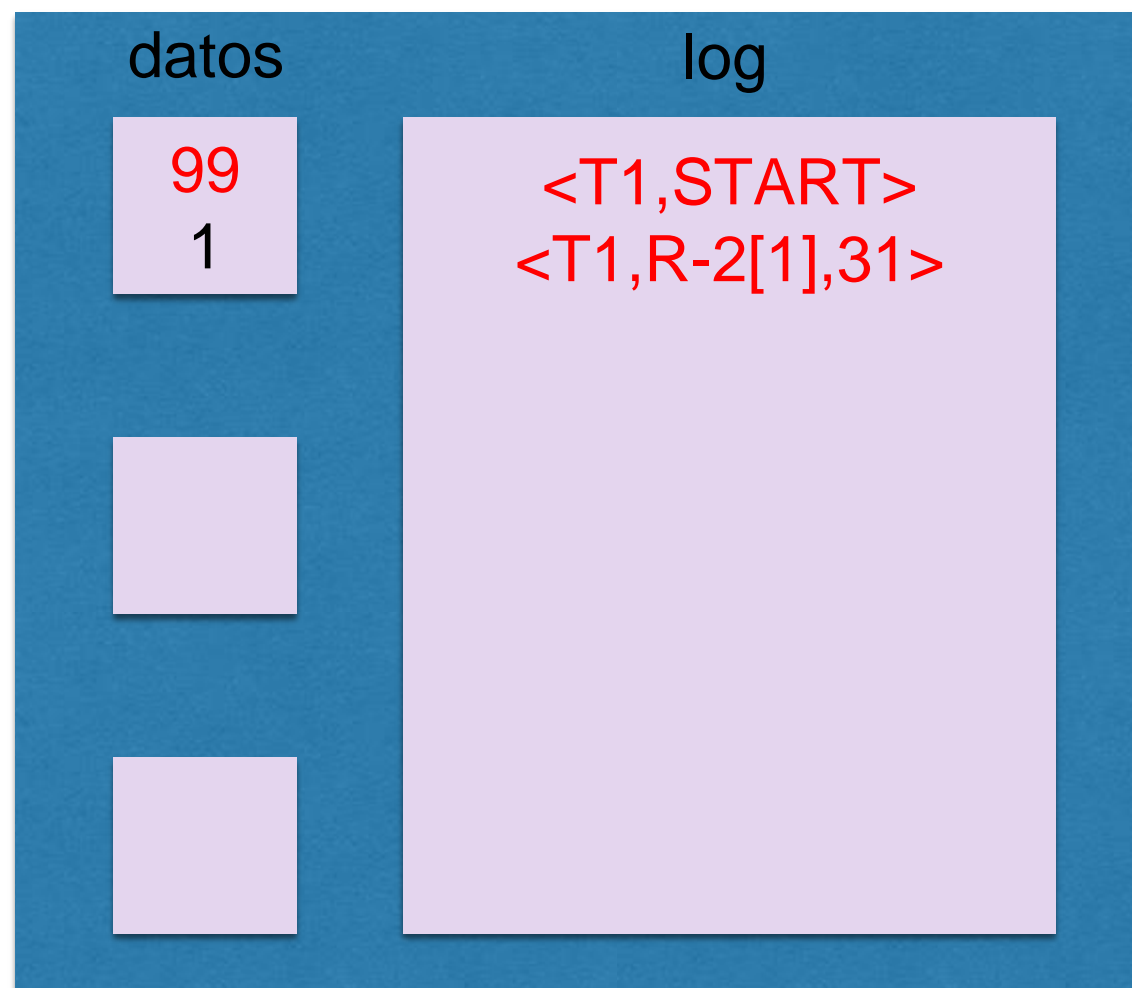


Undo Logging – en la BD

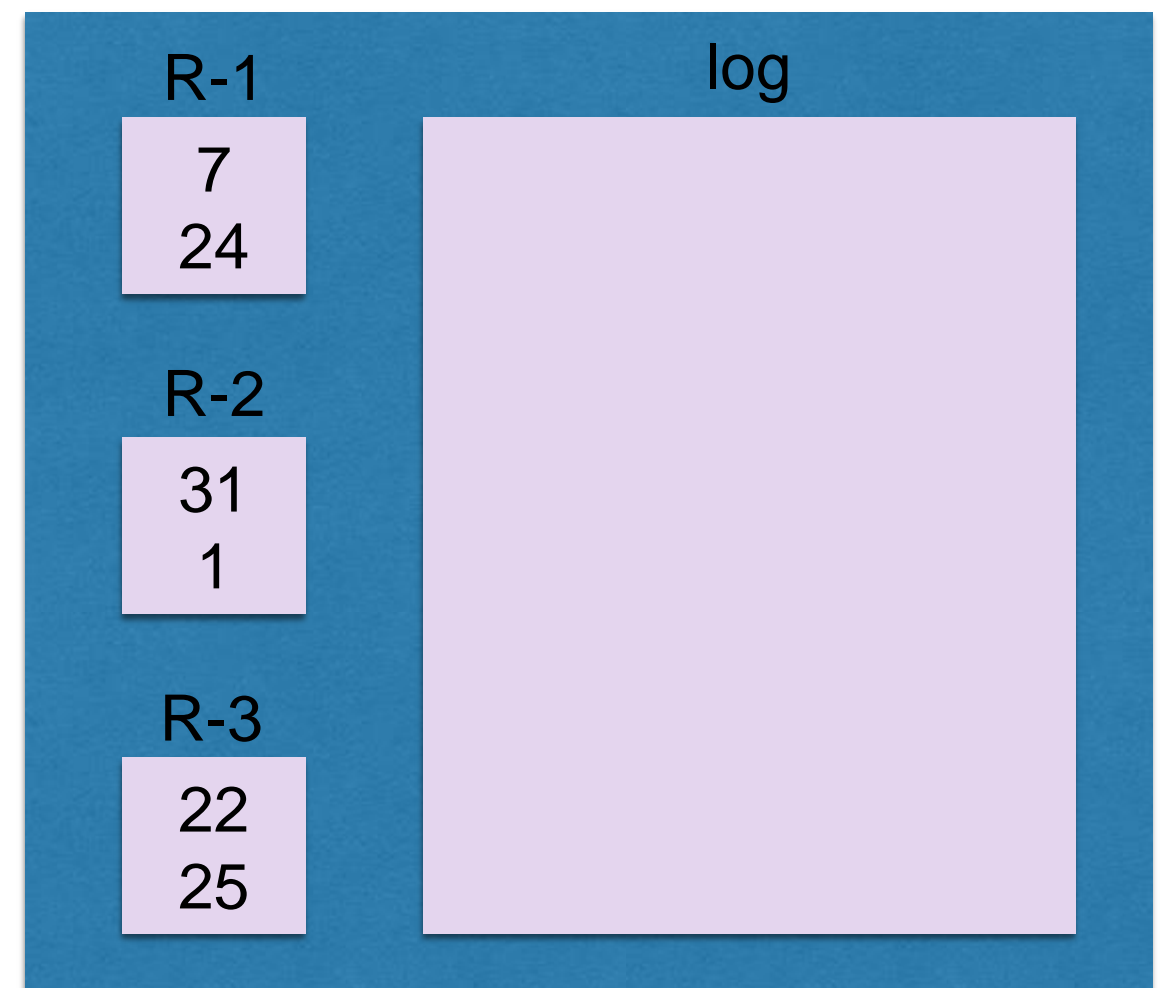
DB

T1: Cambio 31 a 99!

Buffer



Disco

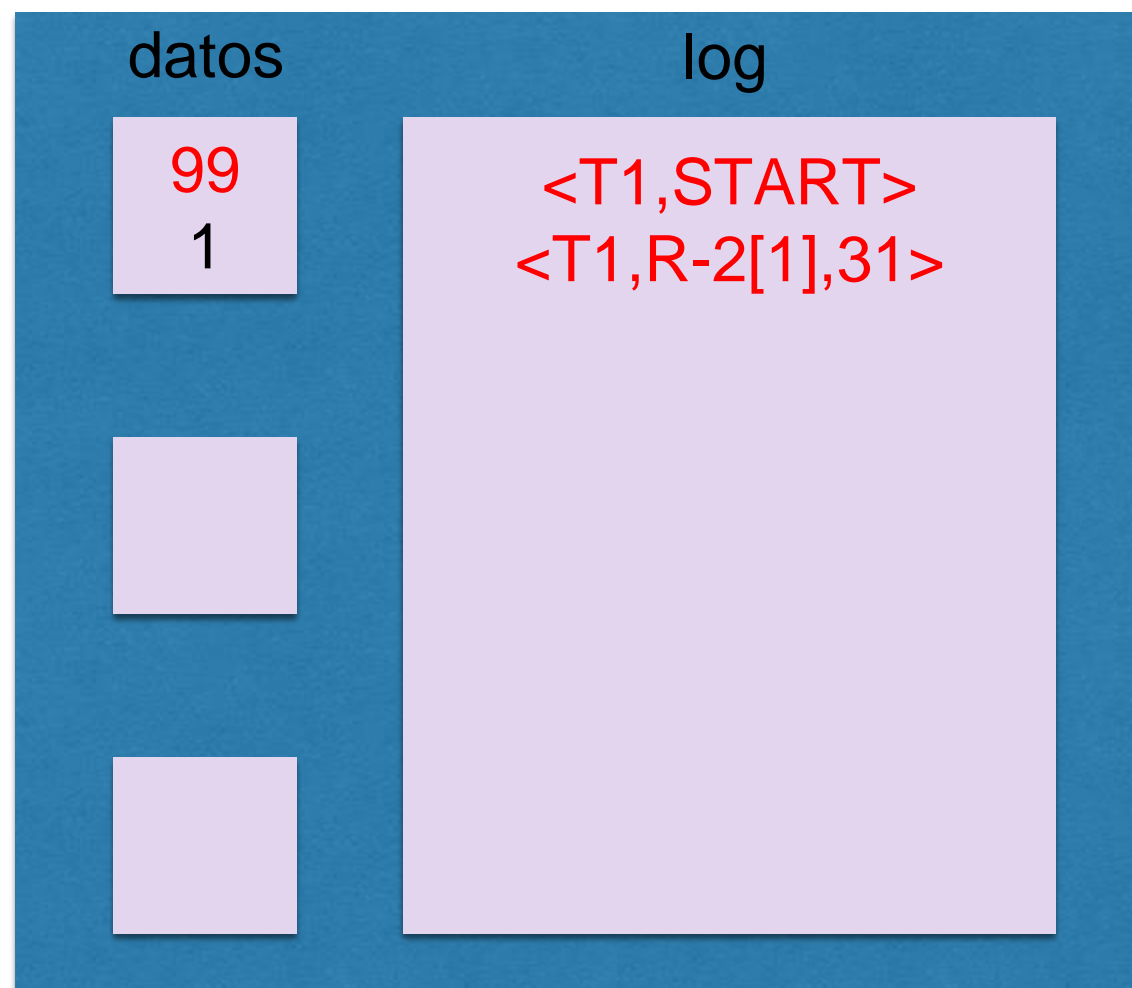


Undo Logging – en la BD

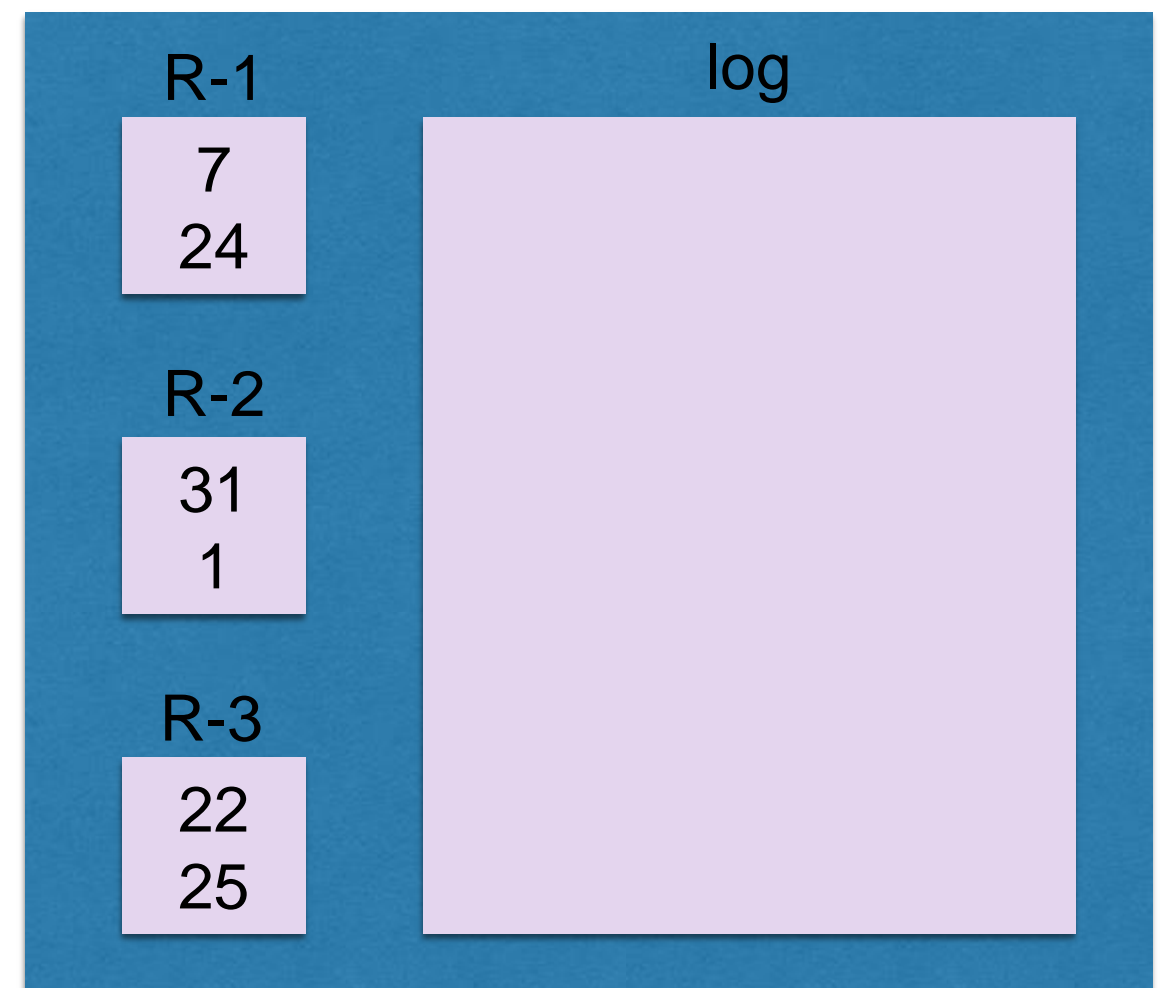
DB

T1: Cambio 99 a 23!

Buffer



Disco

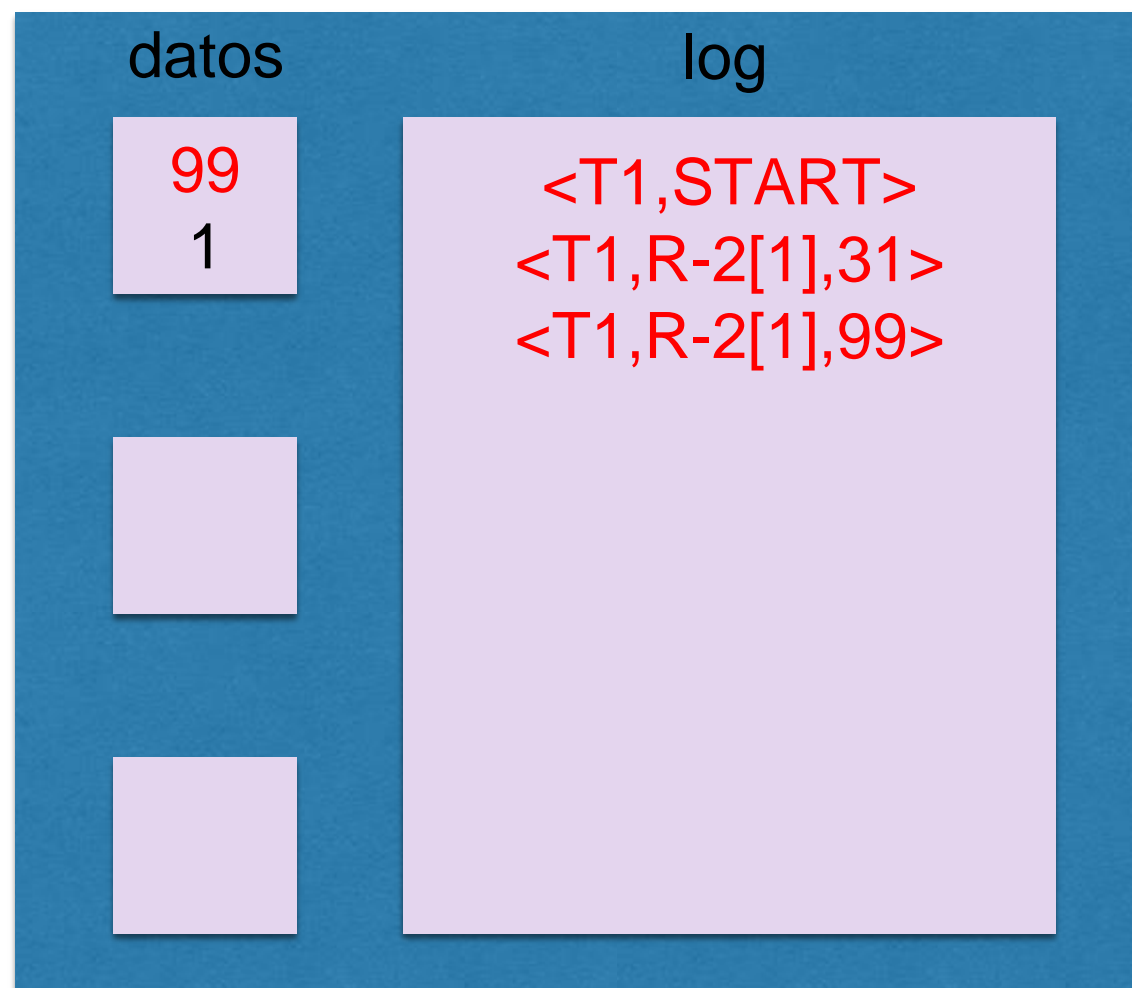


Undo Logging – en la BD

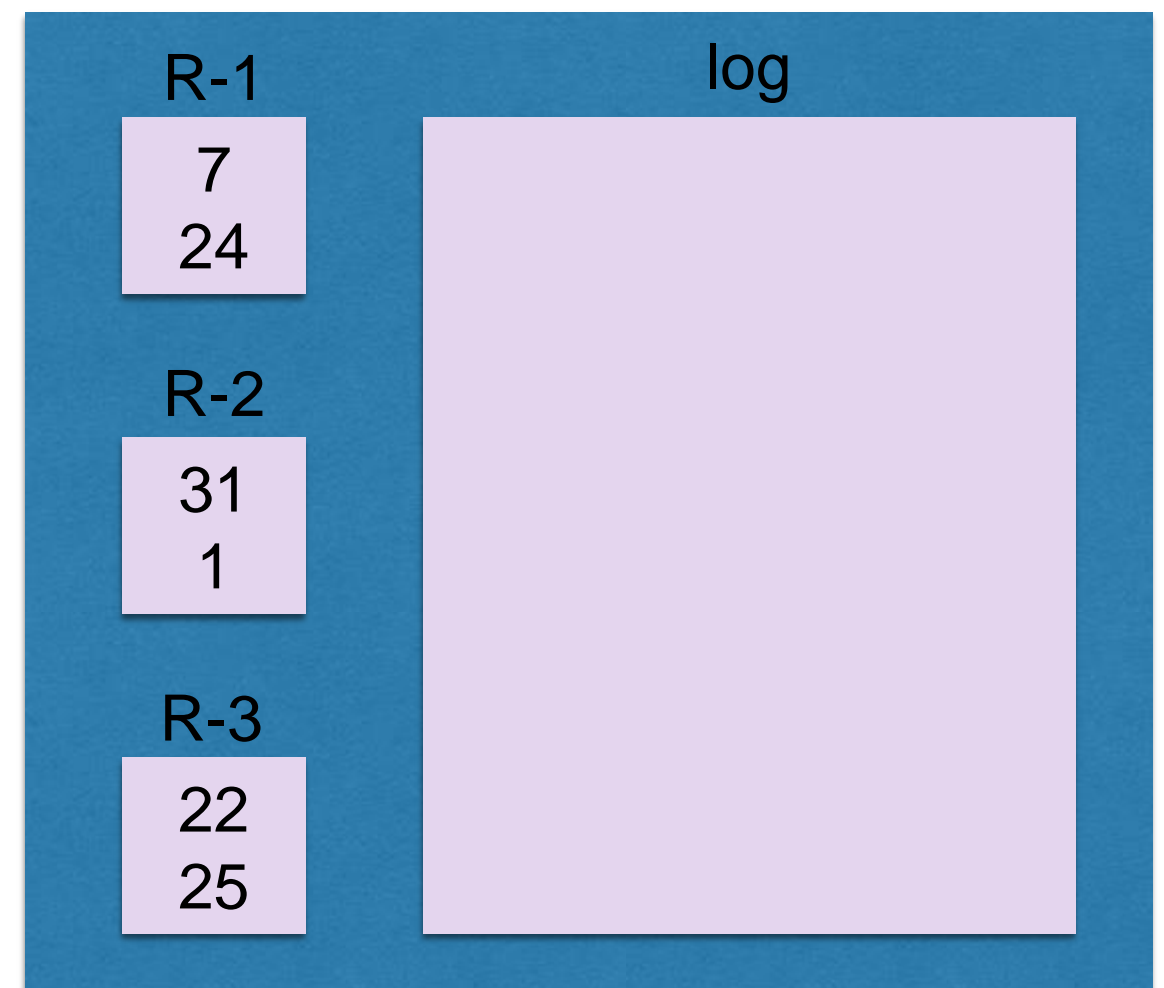
DB

T1: Cambio 99 a 23!

Buffer



Disco

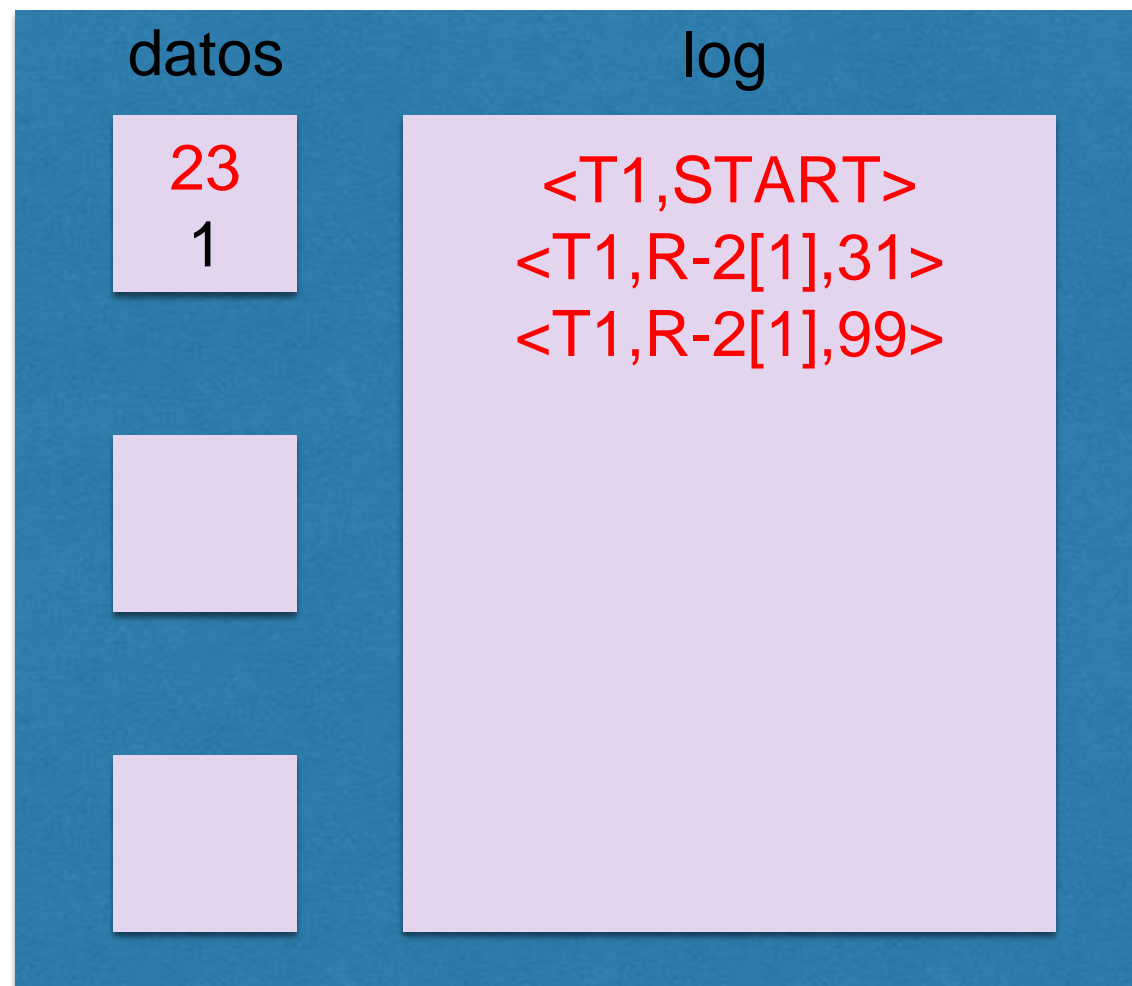


Undo Logging – en la BD

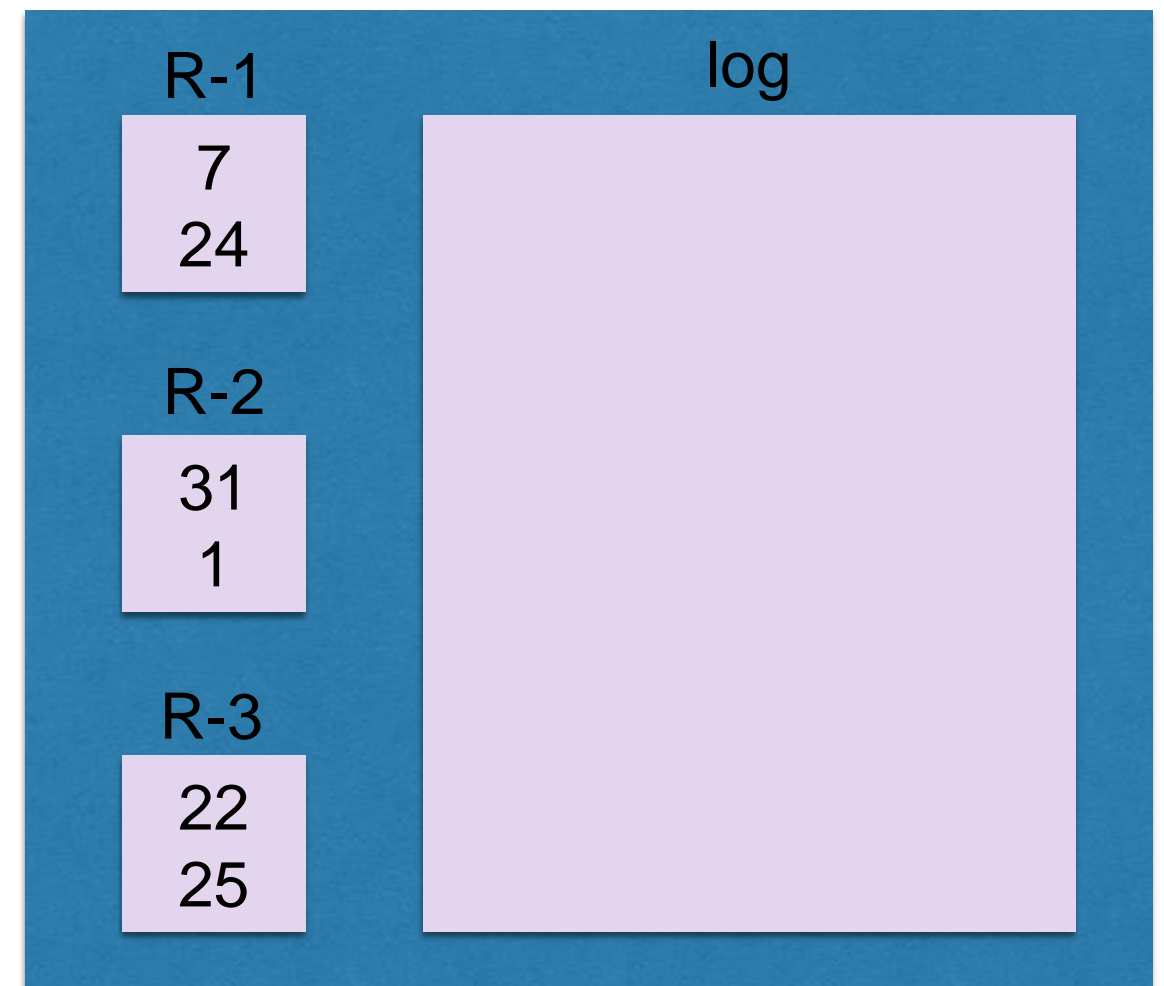
DB

T1: Cambio 99 a 23!

Buffer



Disco

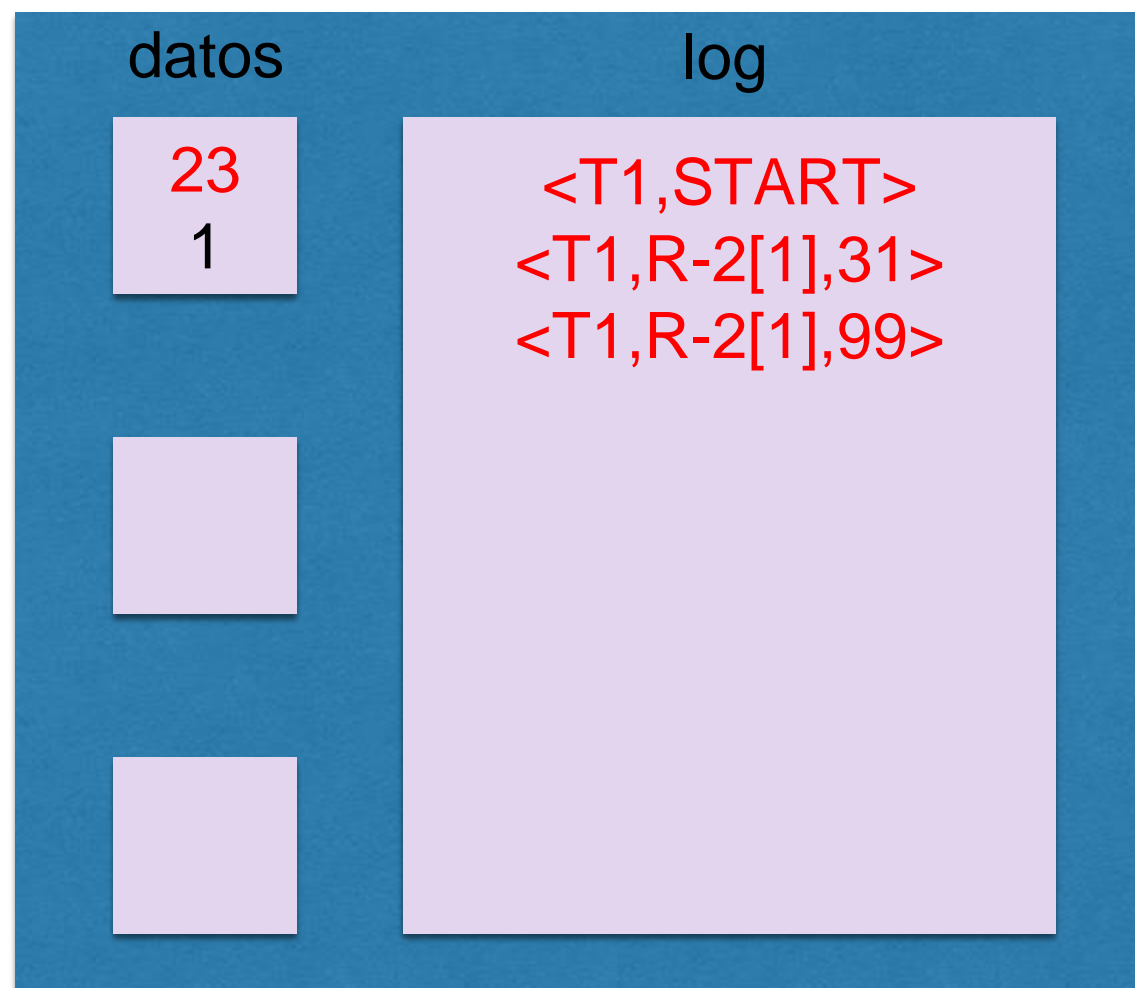


Undo Logging – en la BD

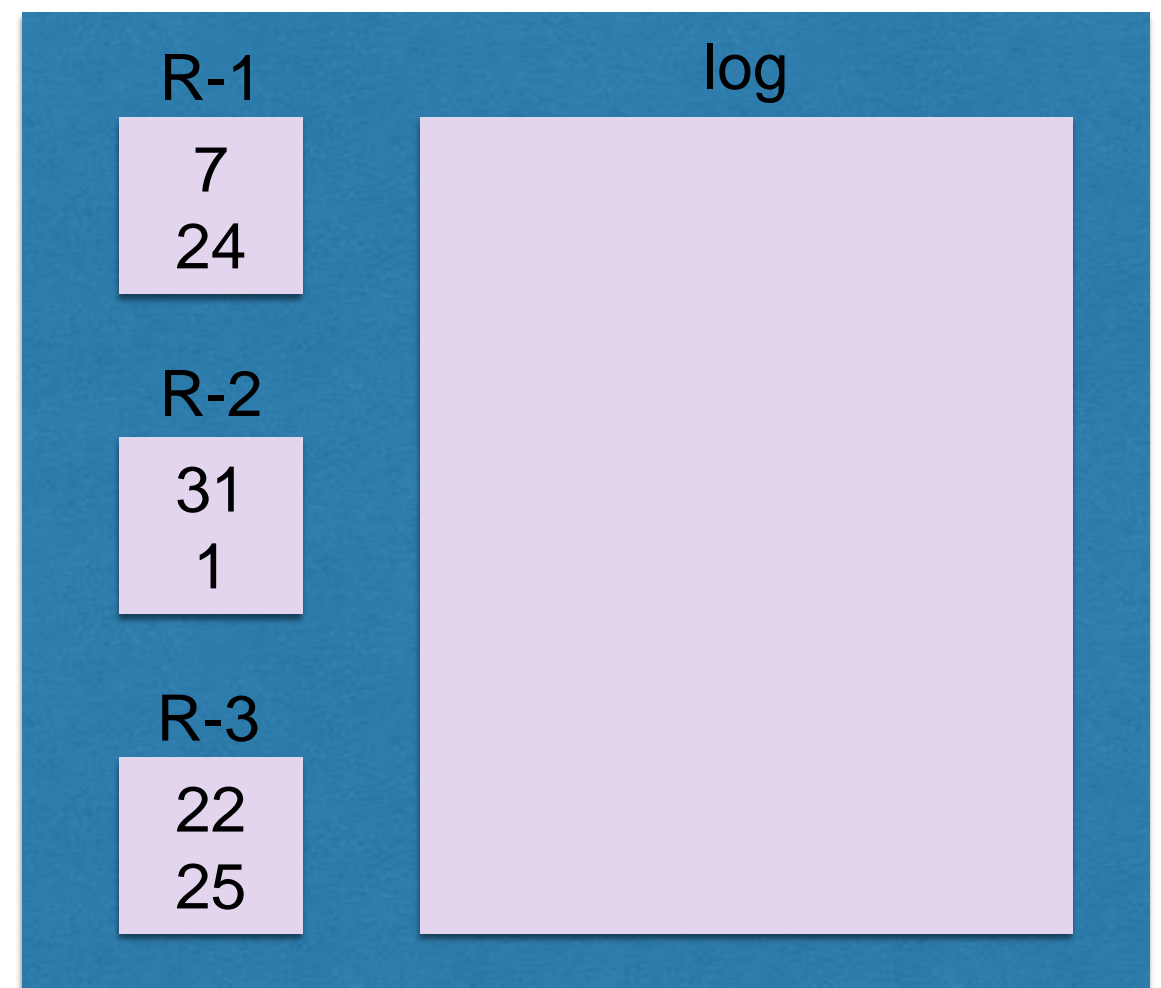
DB

T1: estoy listo!

Buffer



Disco

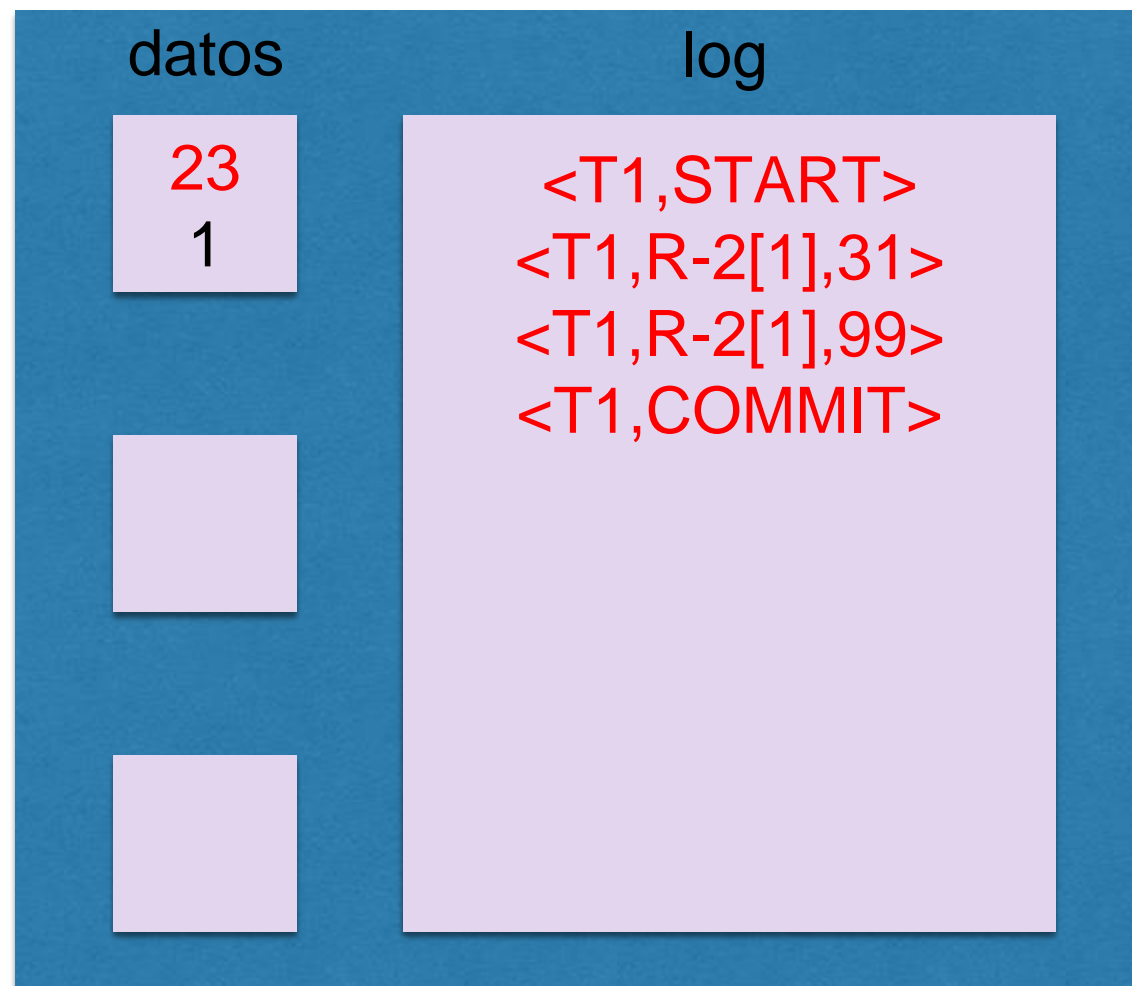


Undo Logging – en la BD

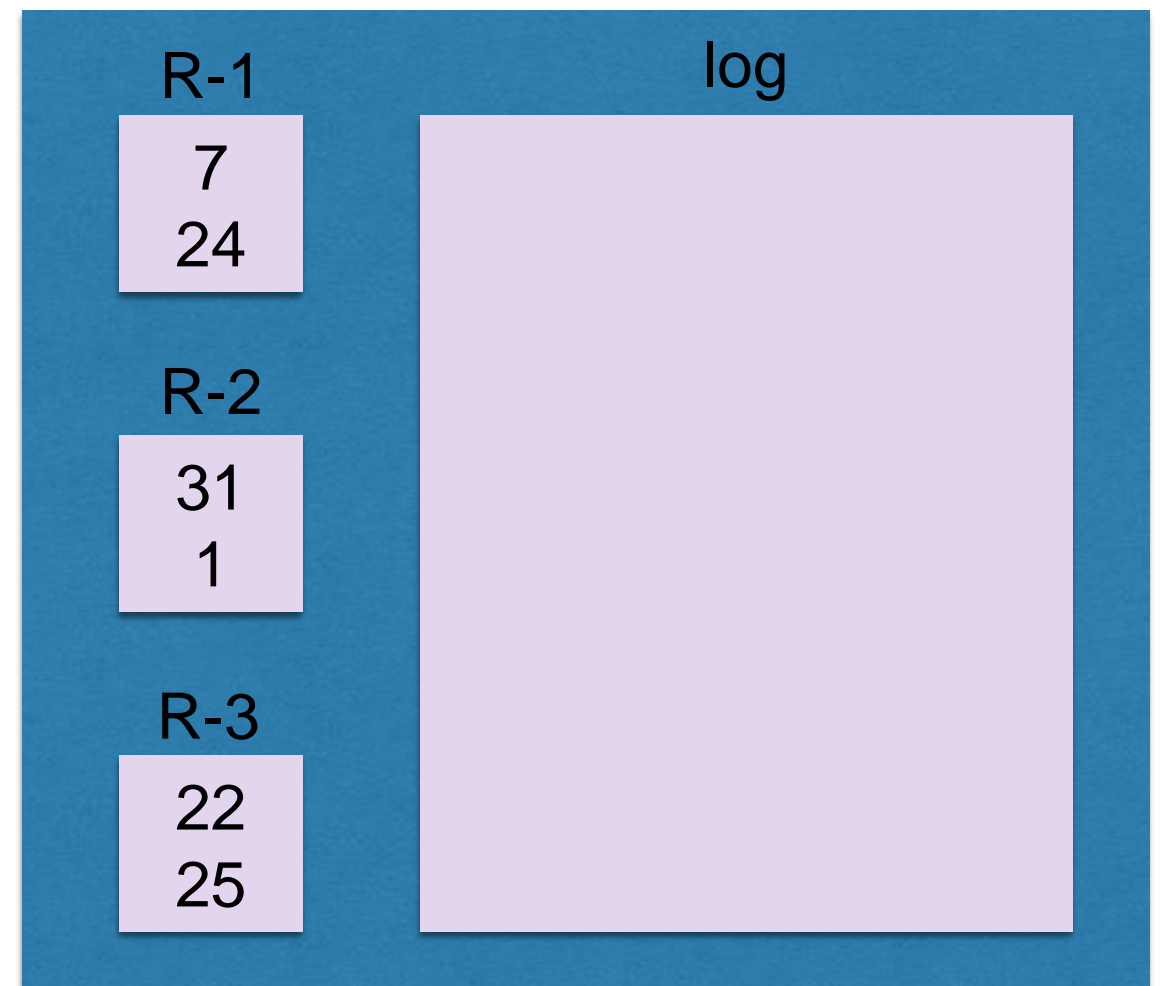
DB

T1: estoy listo!

Buffer



Disco

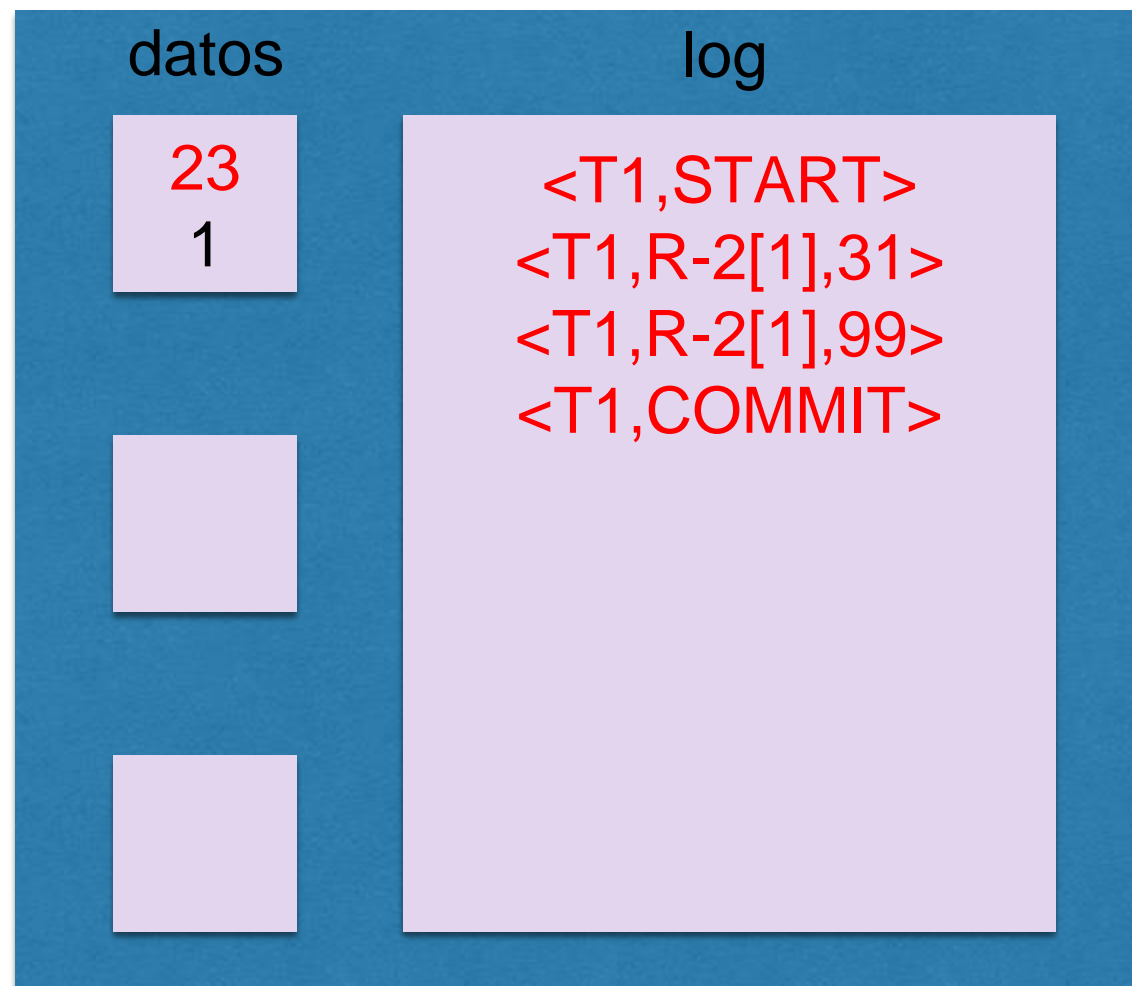


Undo Logging – en la BD

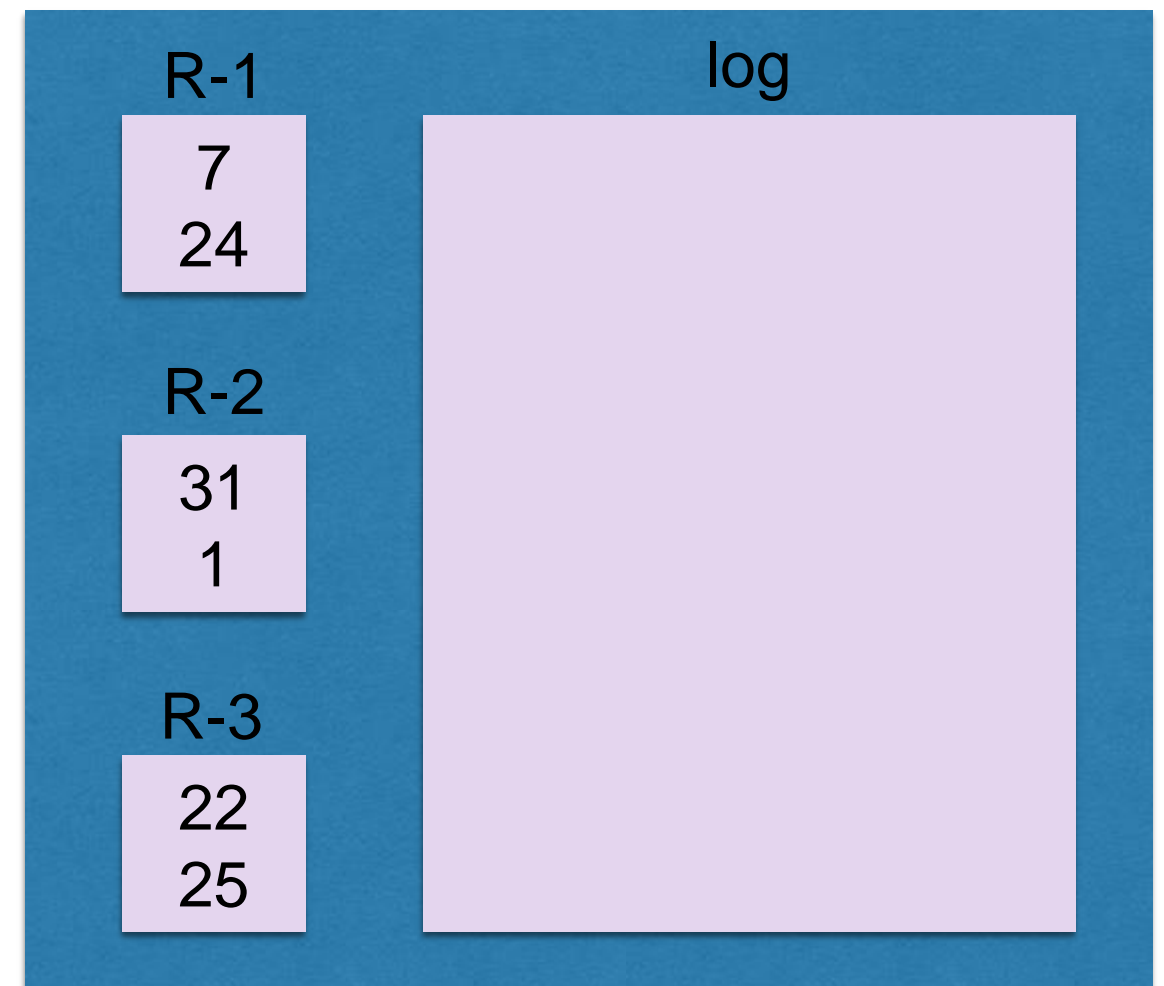
DB

Vamos al disco!

Buffer



Disco

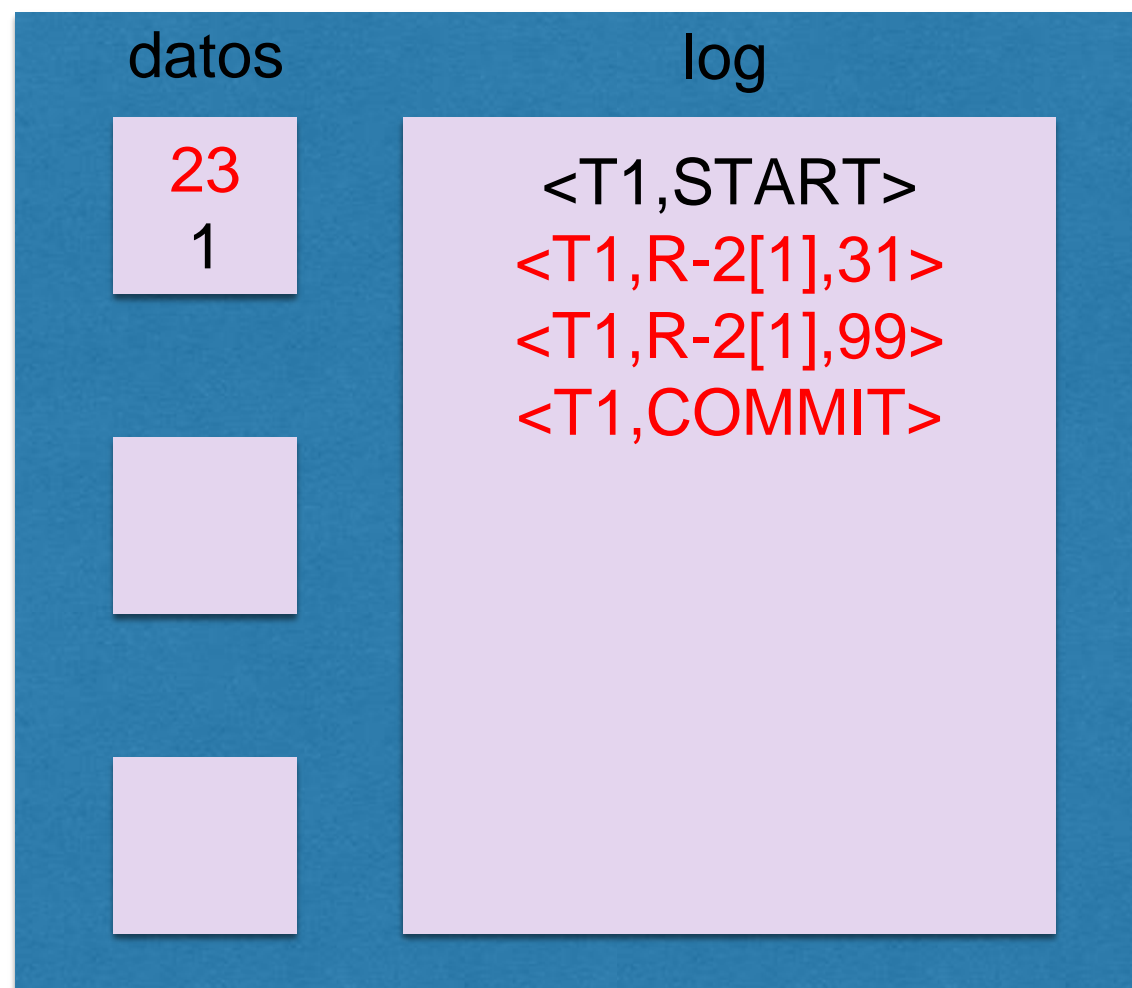


Undo Logging – en la BD

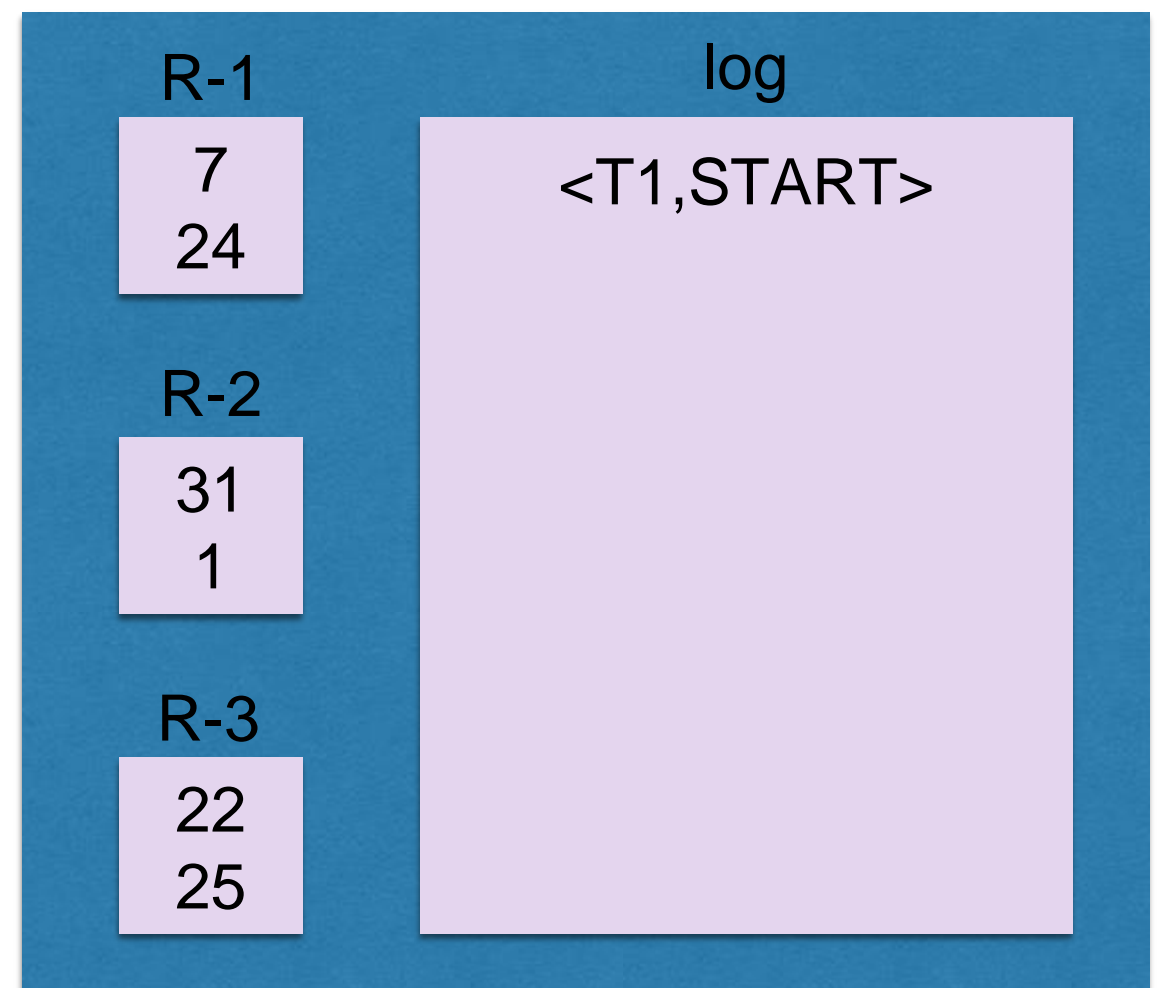
DB

Vamos al disco!

Buffer



Disco

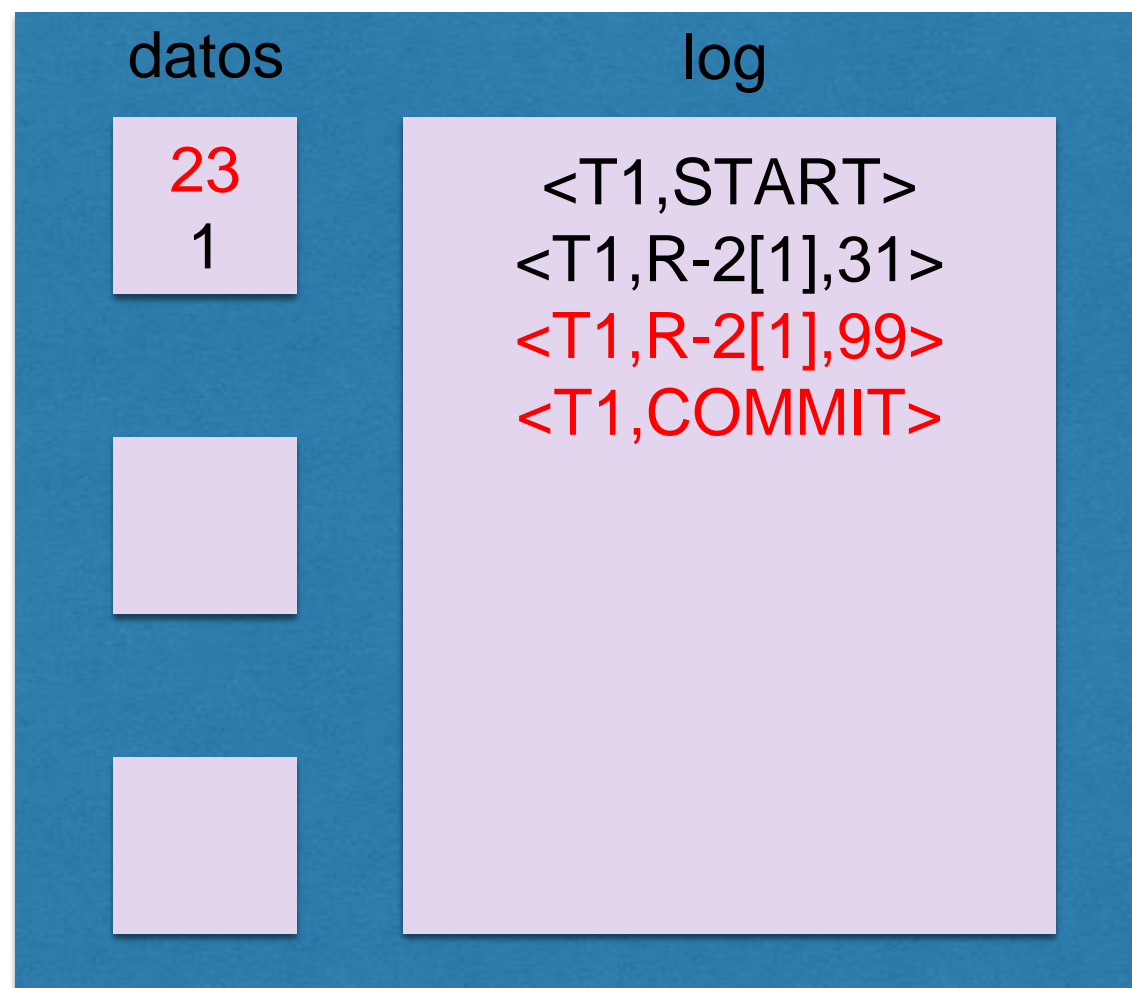


Undo Logging – en la BD

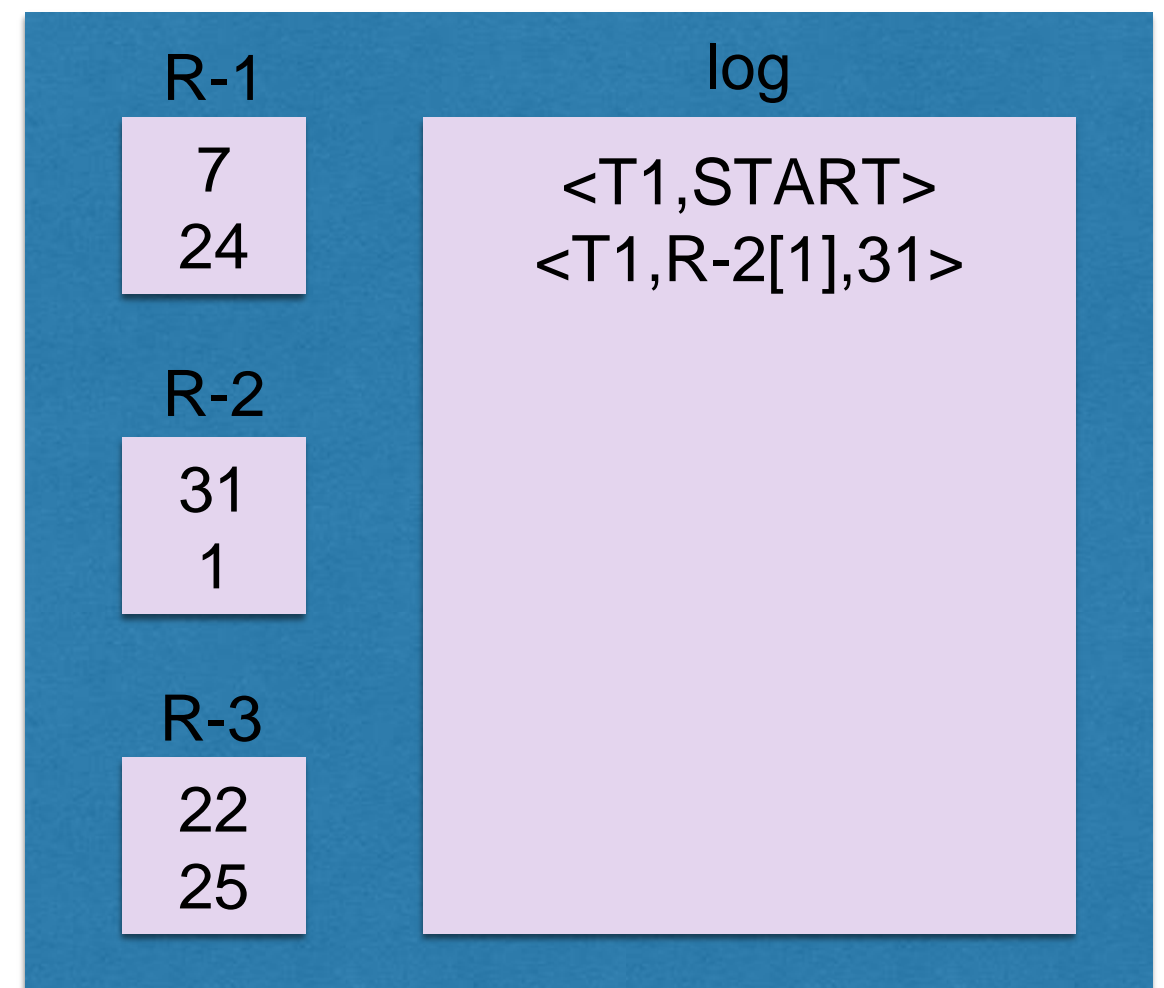
DB

Vamos al disco!

Buffer



Disco

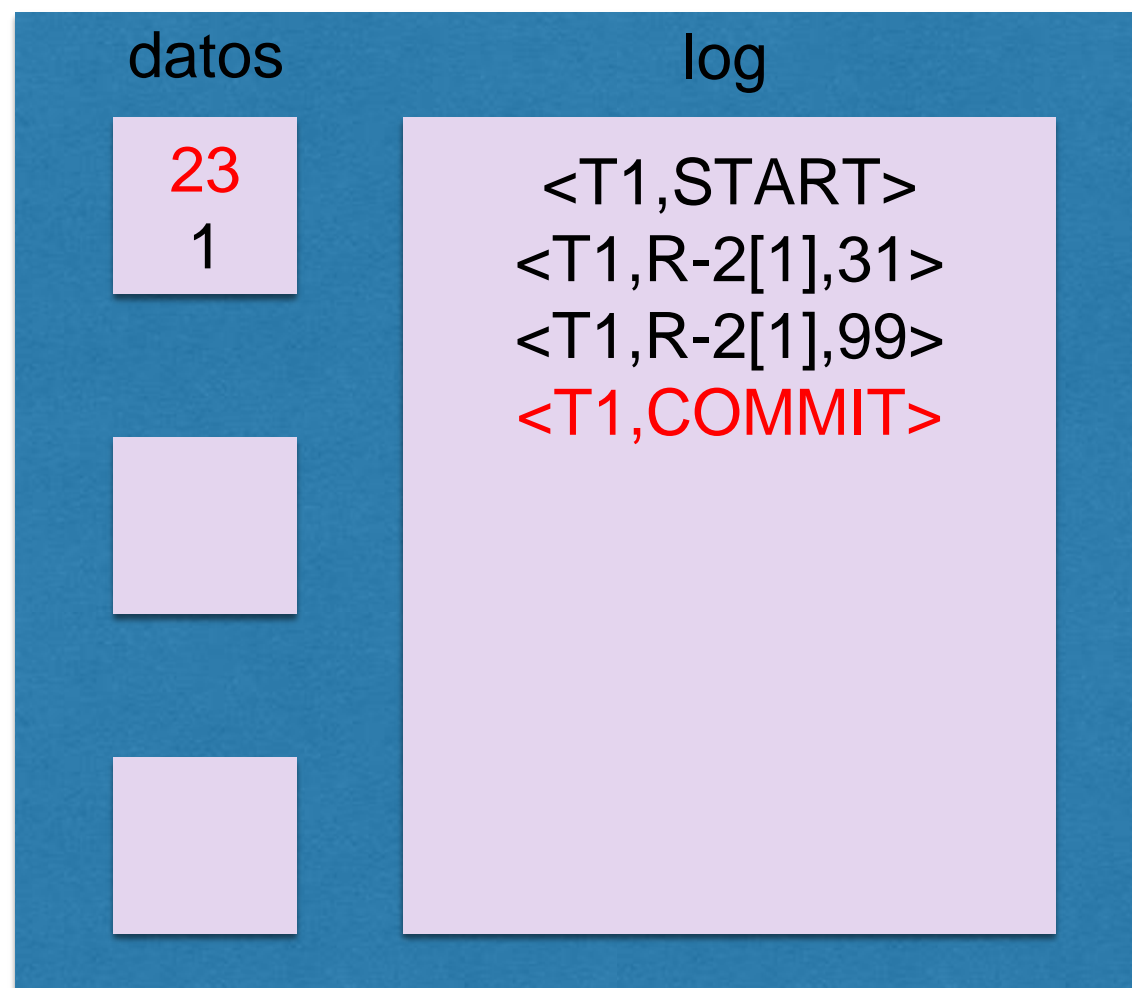


Undo Logging – en la BD

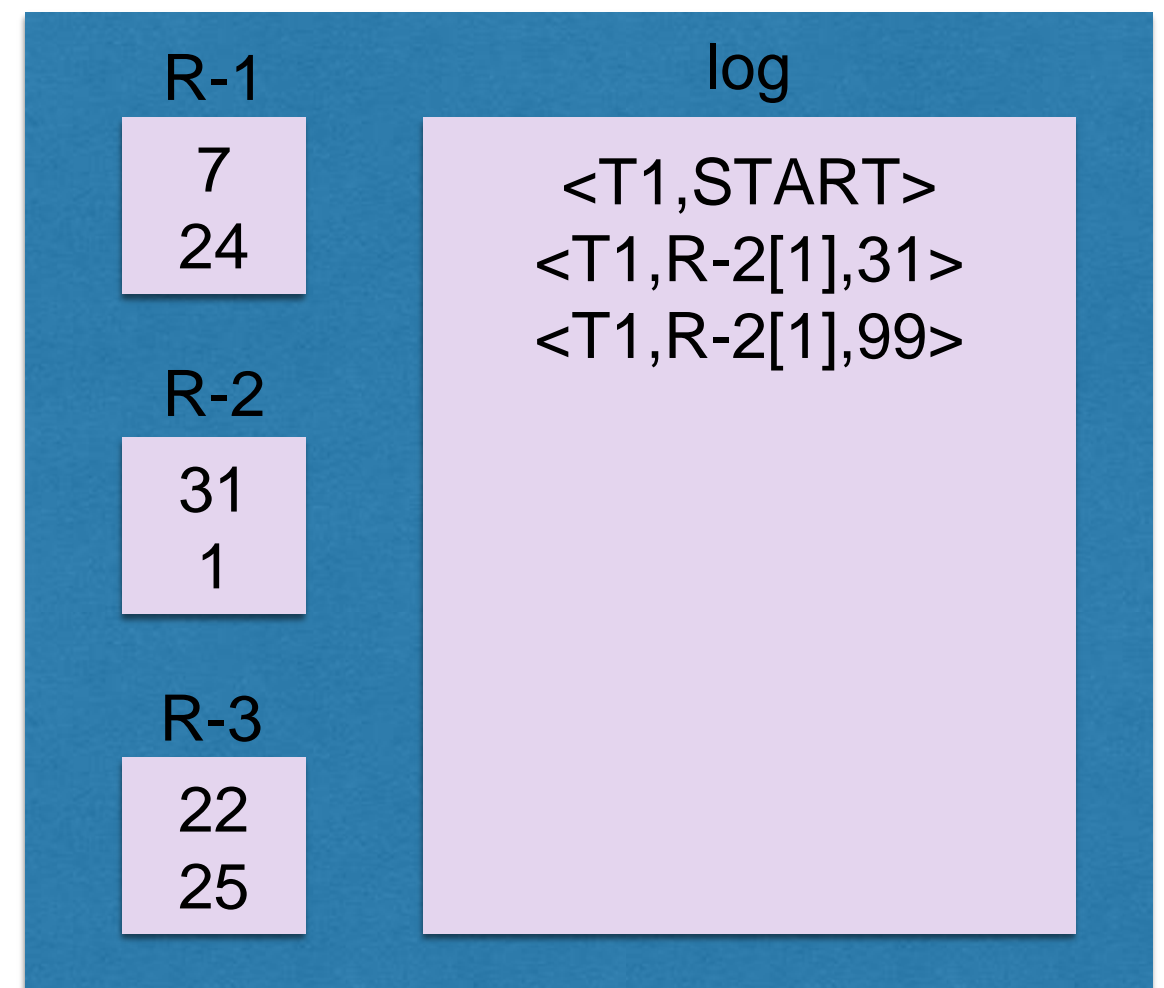
DB

Vamos al disco!

Buffer



Disco

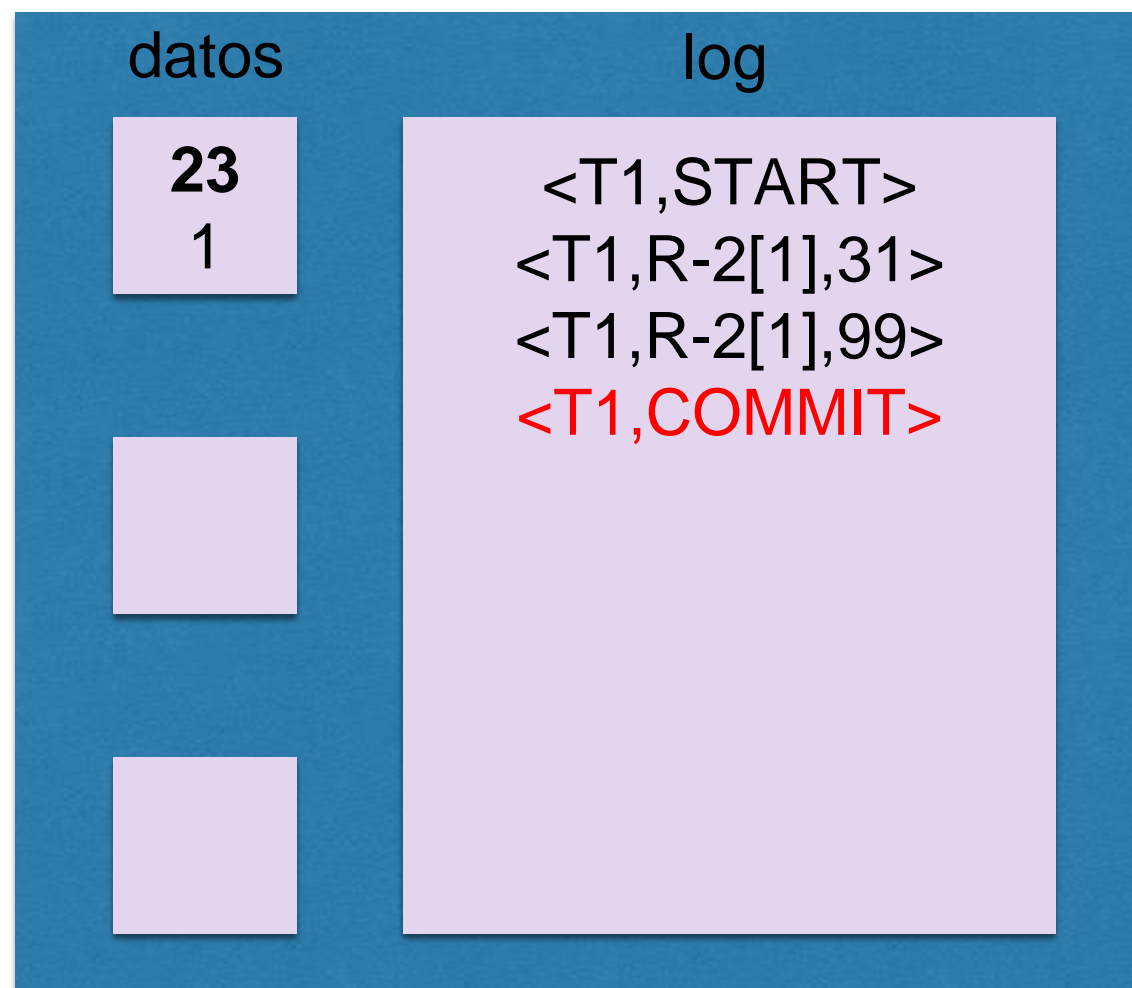


Undo Logging – en la BD

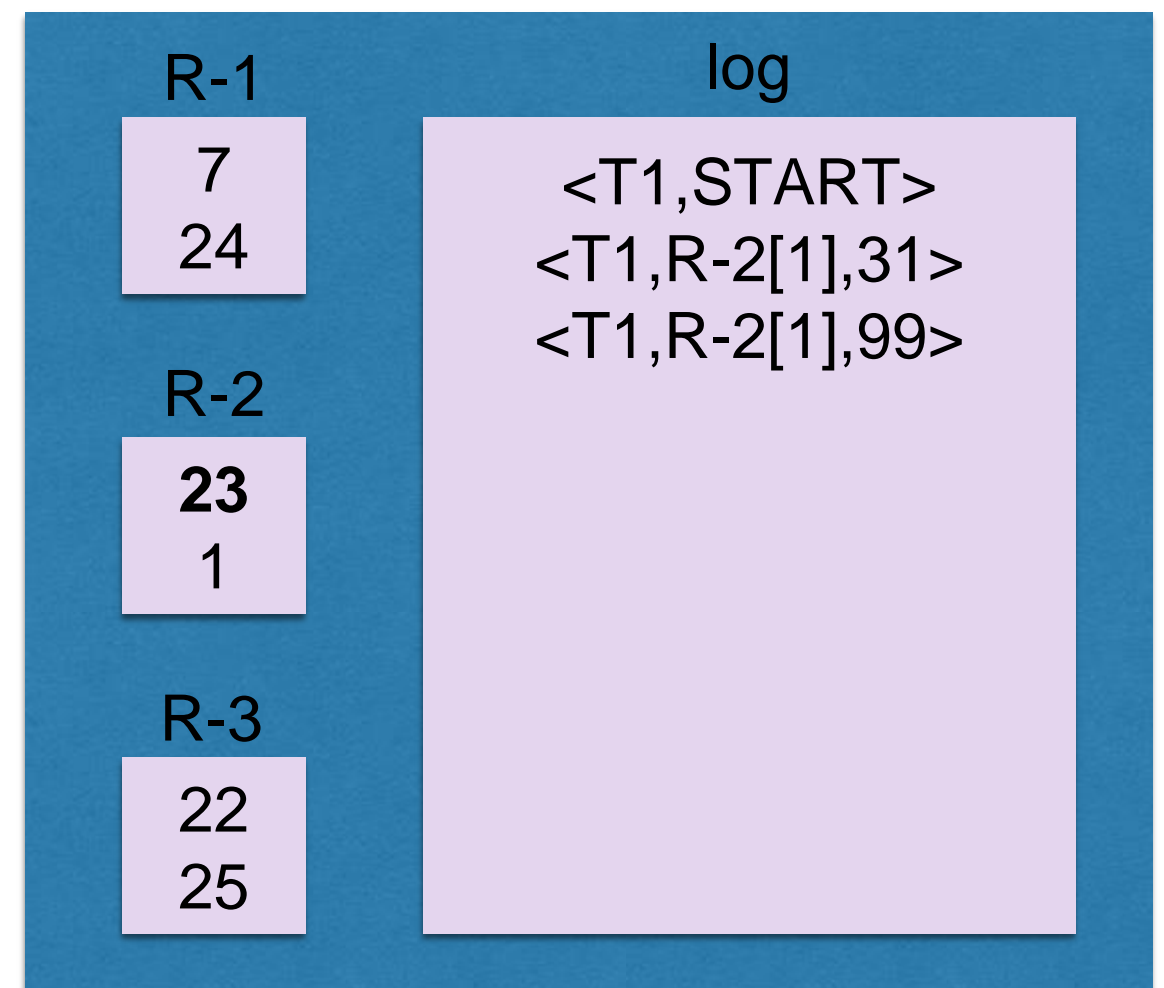
DB

Vamos al disco!

Buffer



Disco

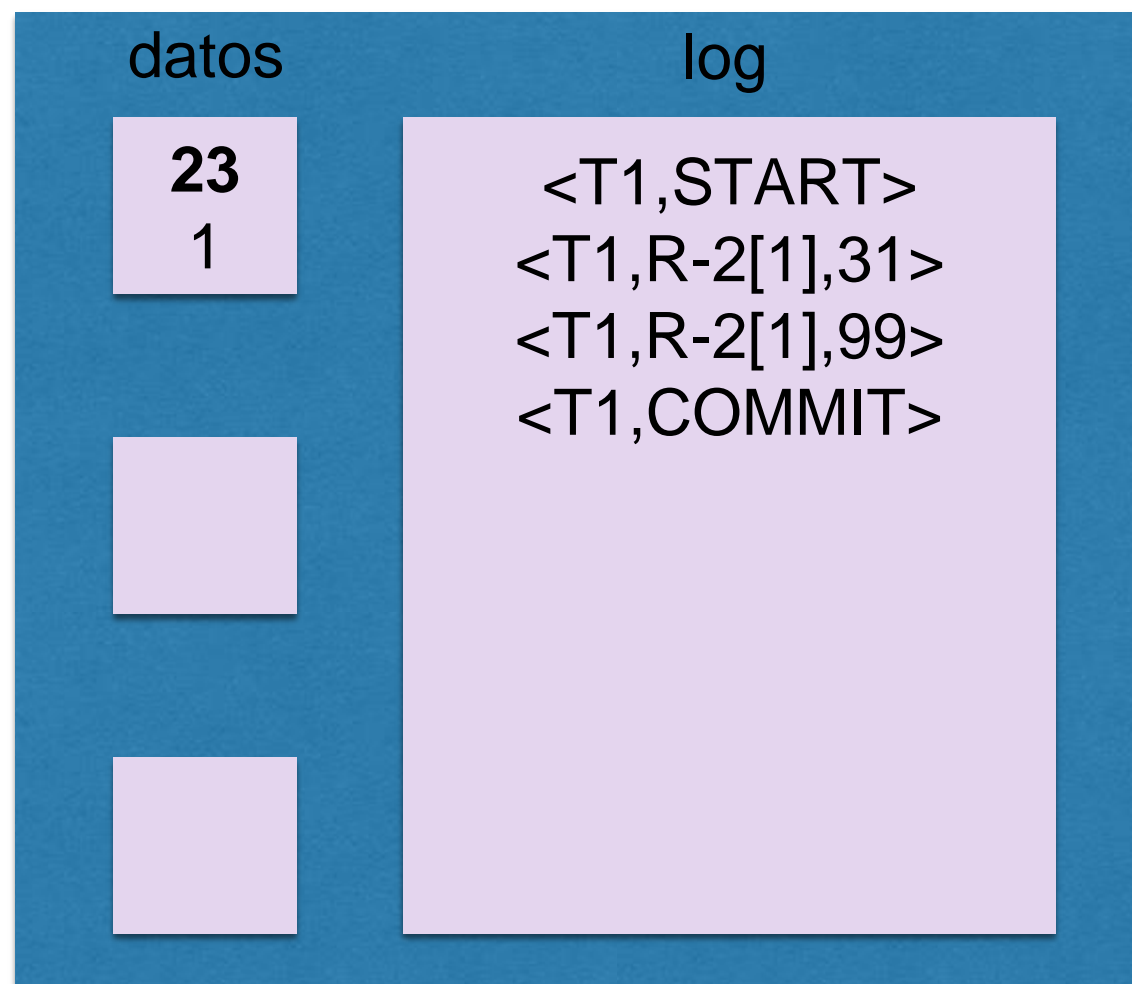


Undo Logging – en la BD

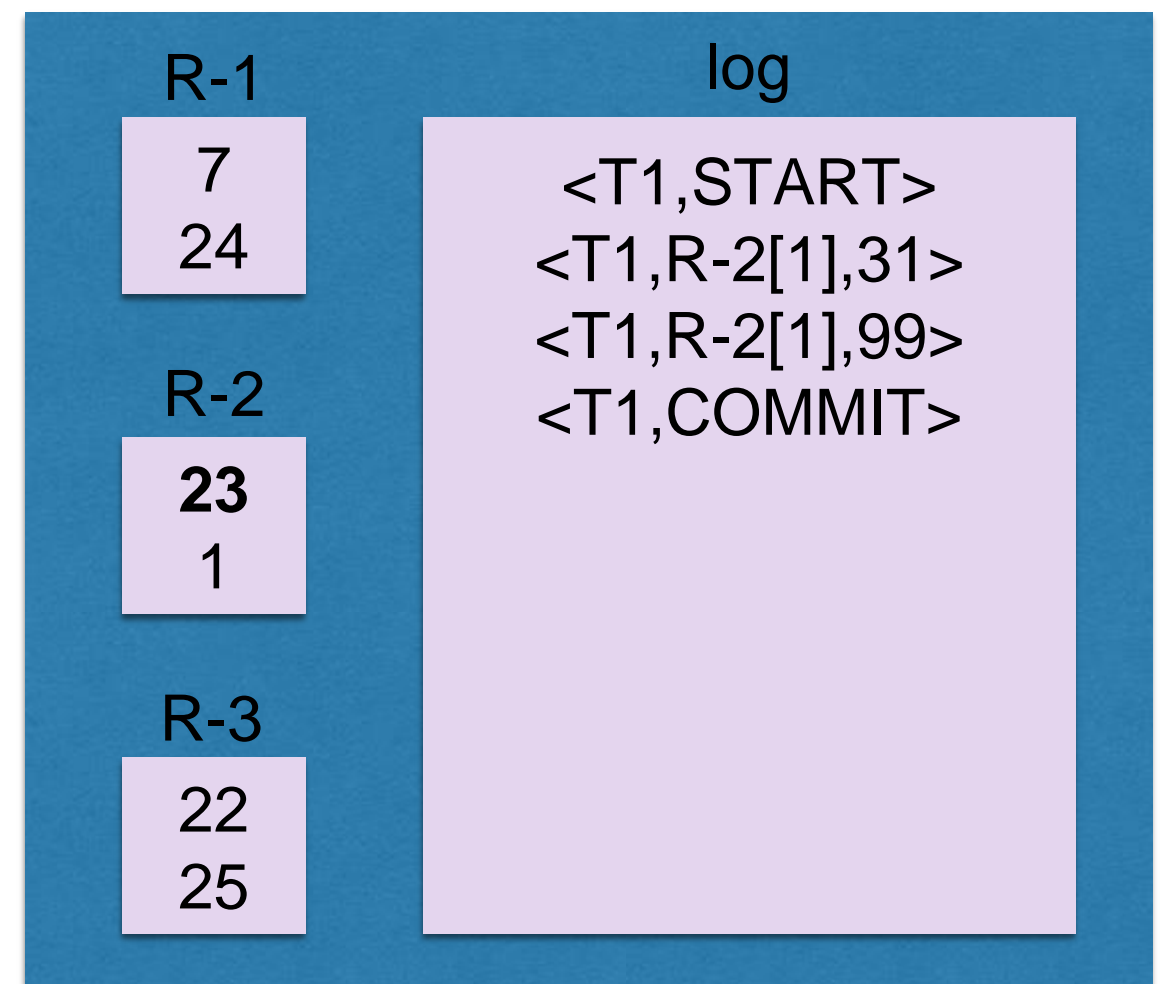
DB

Vamos al disco!

Buffer



Disco



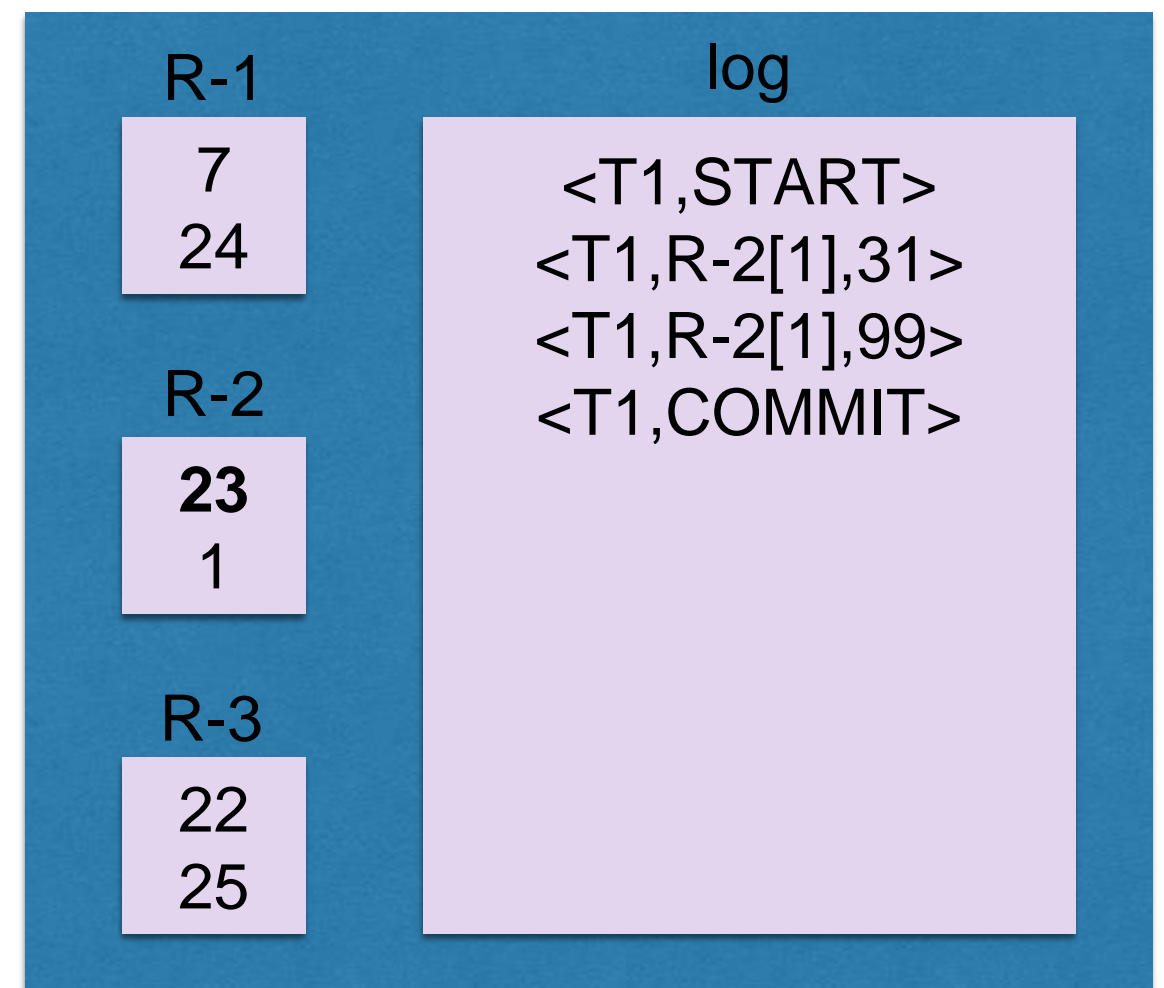
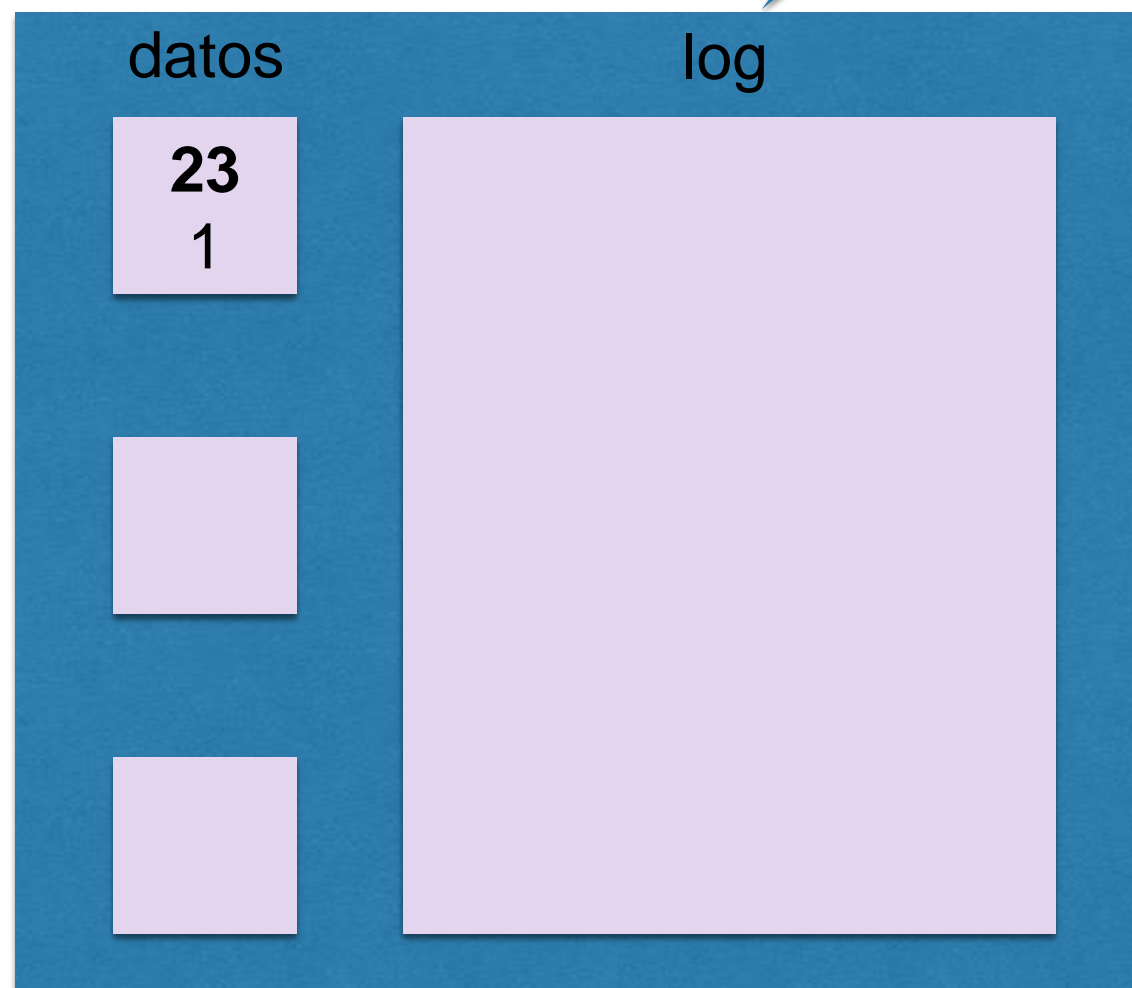
Undo Logging – en la BD

DB

Libera el buffer

Buffer

Disco

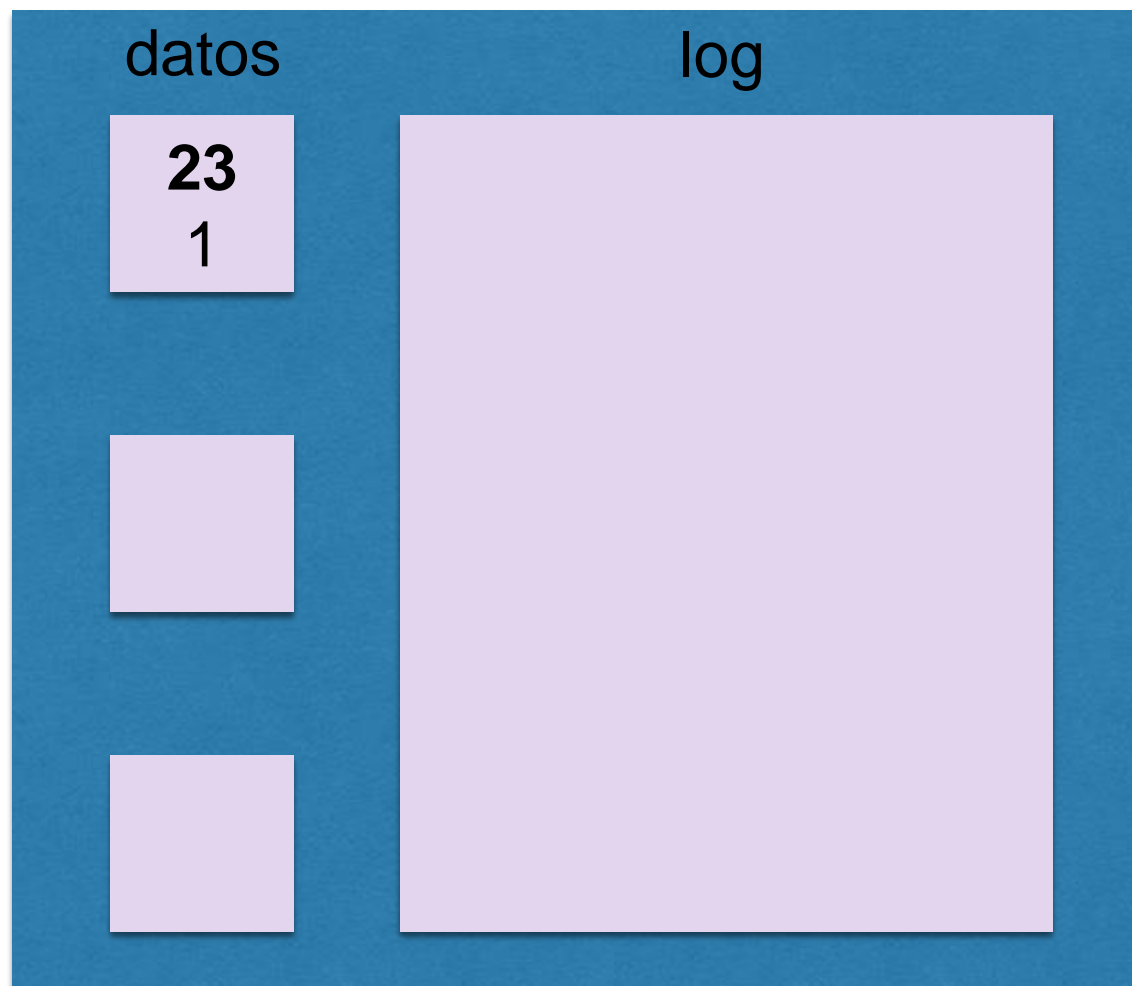


Undo Logging – en la BD

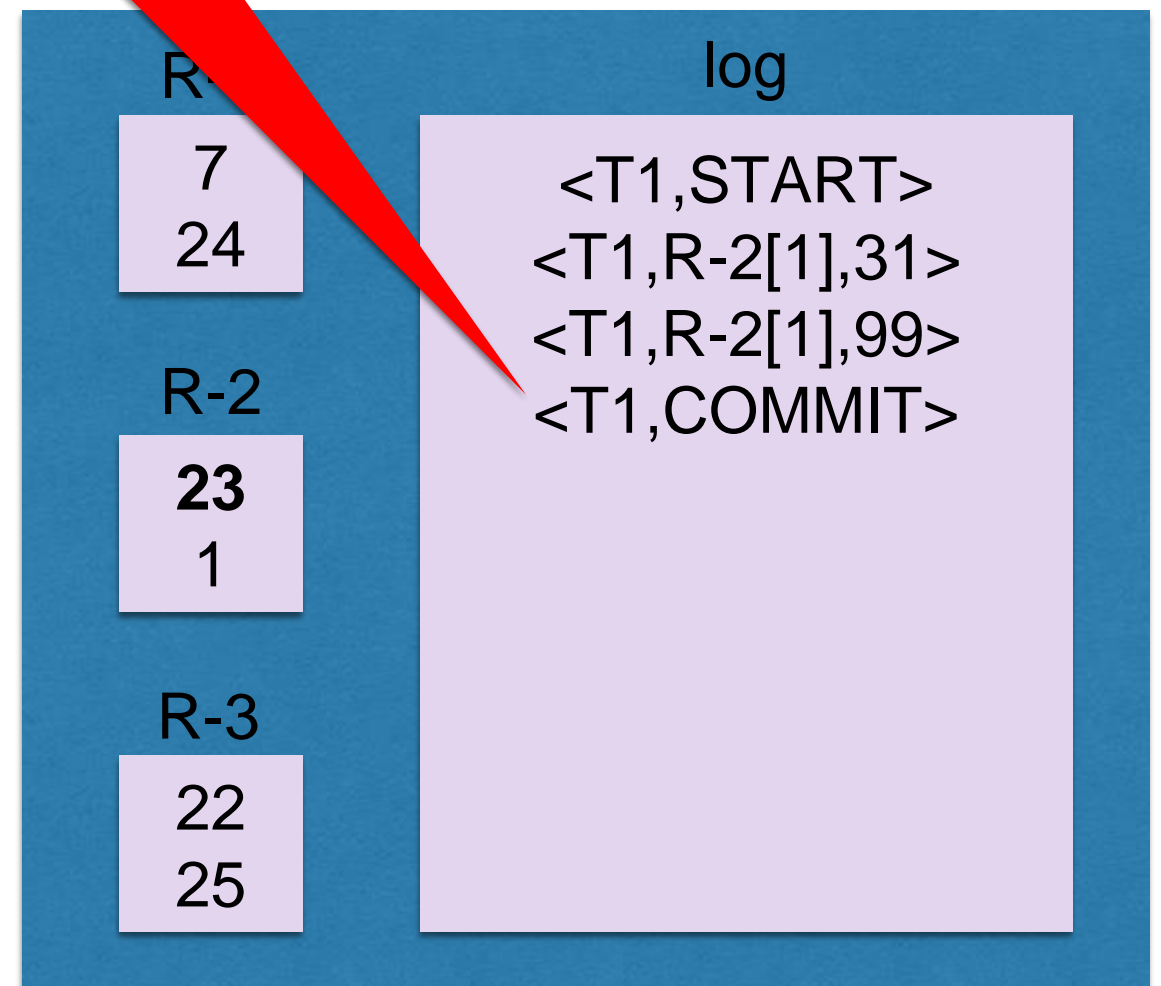
DB

COMMIT =
datos están en disco

Buffer



Disco



Recuperación con Undo Logging

Detectando fallas en el *log*:

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

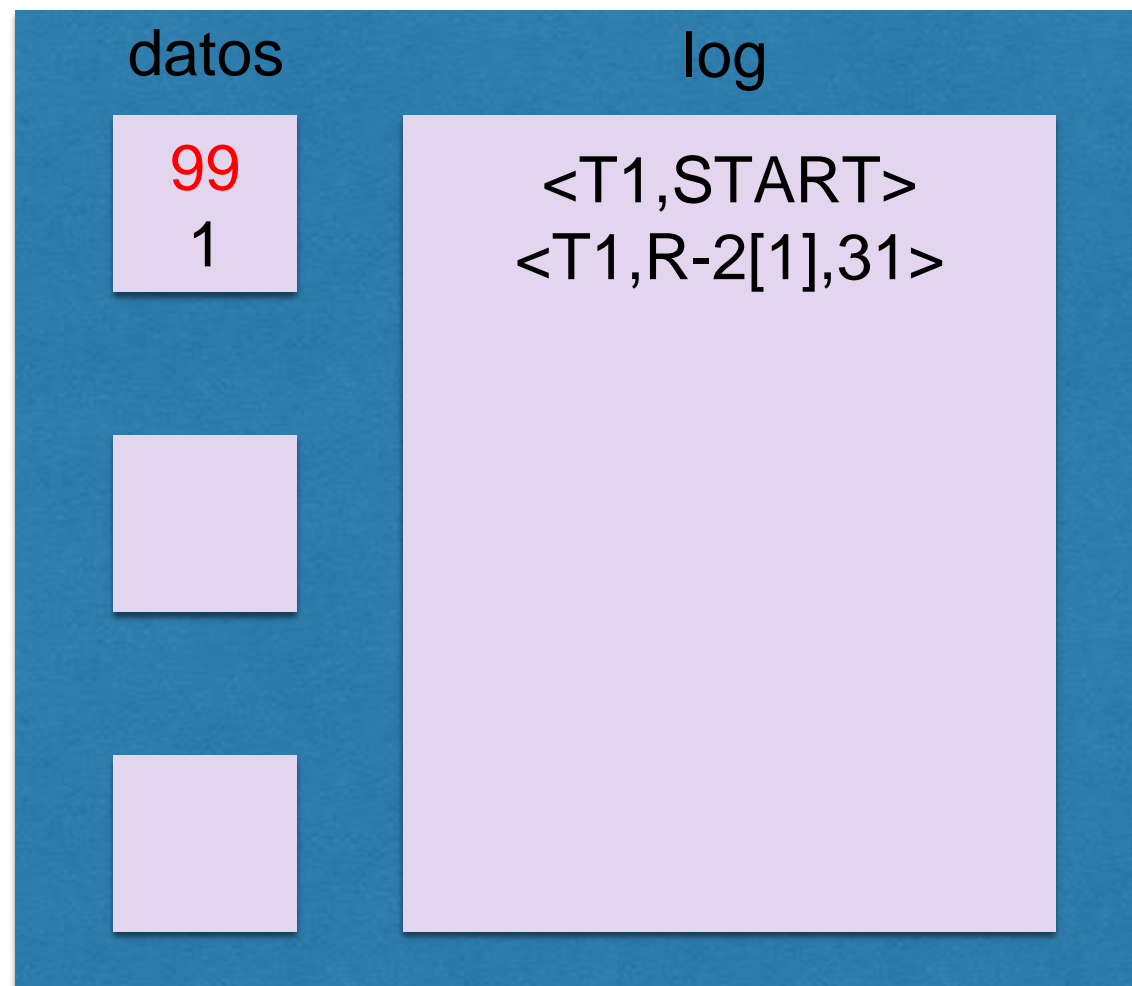


Undo Logging – en la BD

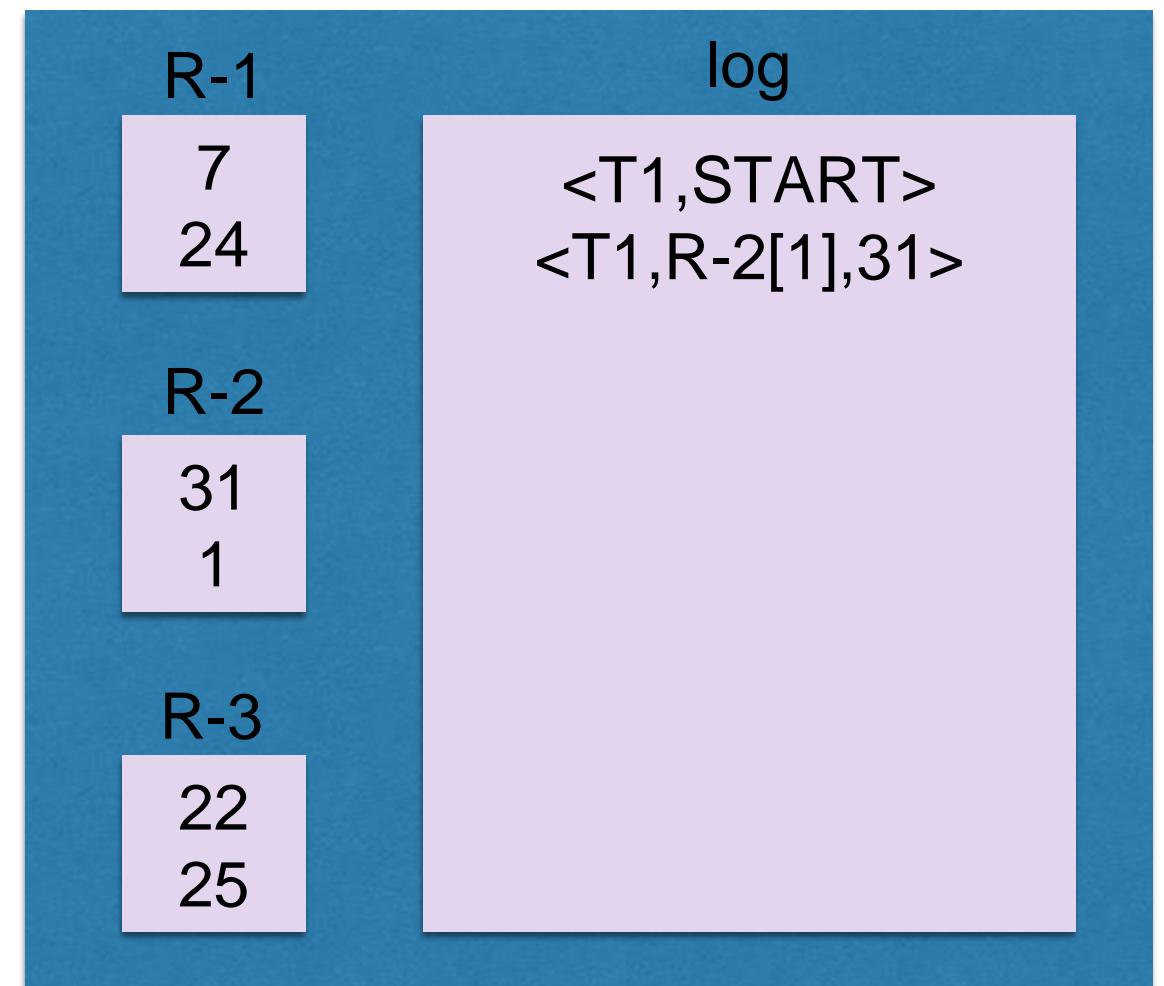
DB

T1: Cambio 31 a 99!

Buffer



Disco

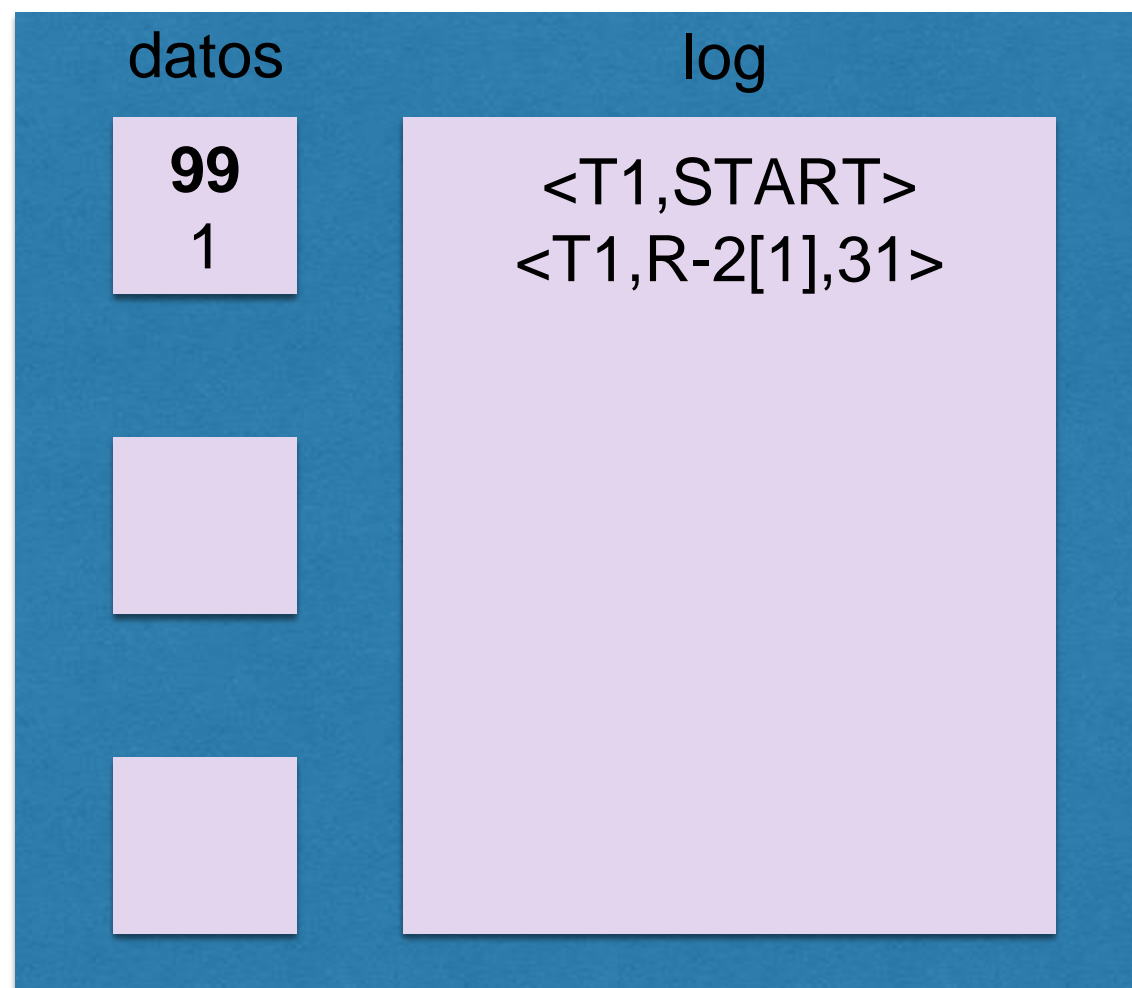


Undo Logging – en la BD

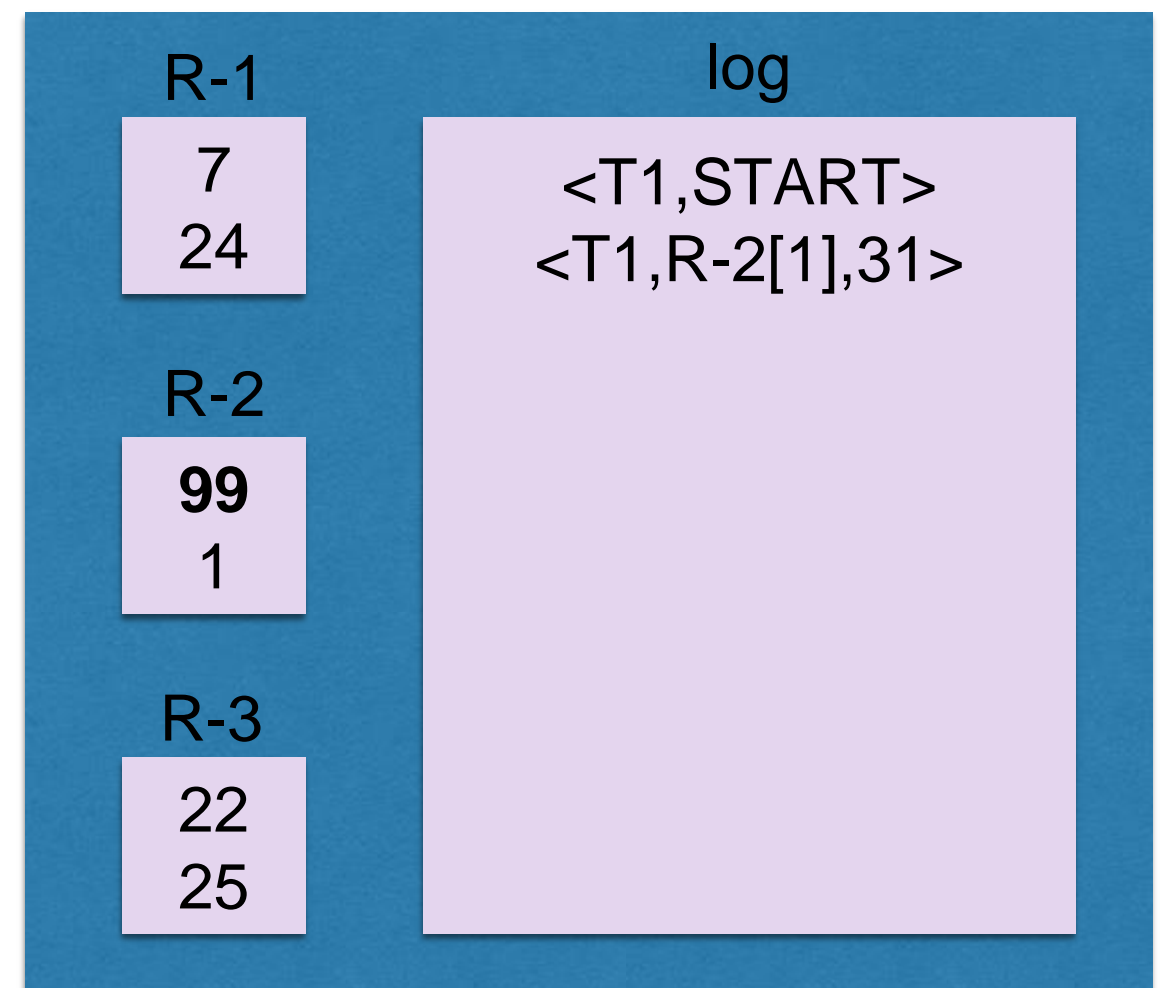
DB

T1: Cambio 31 a 99!

Buffer



Disco

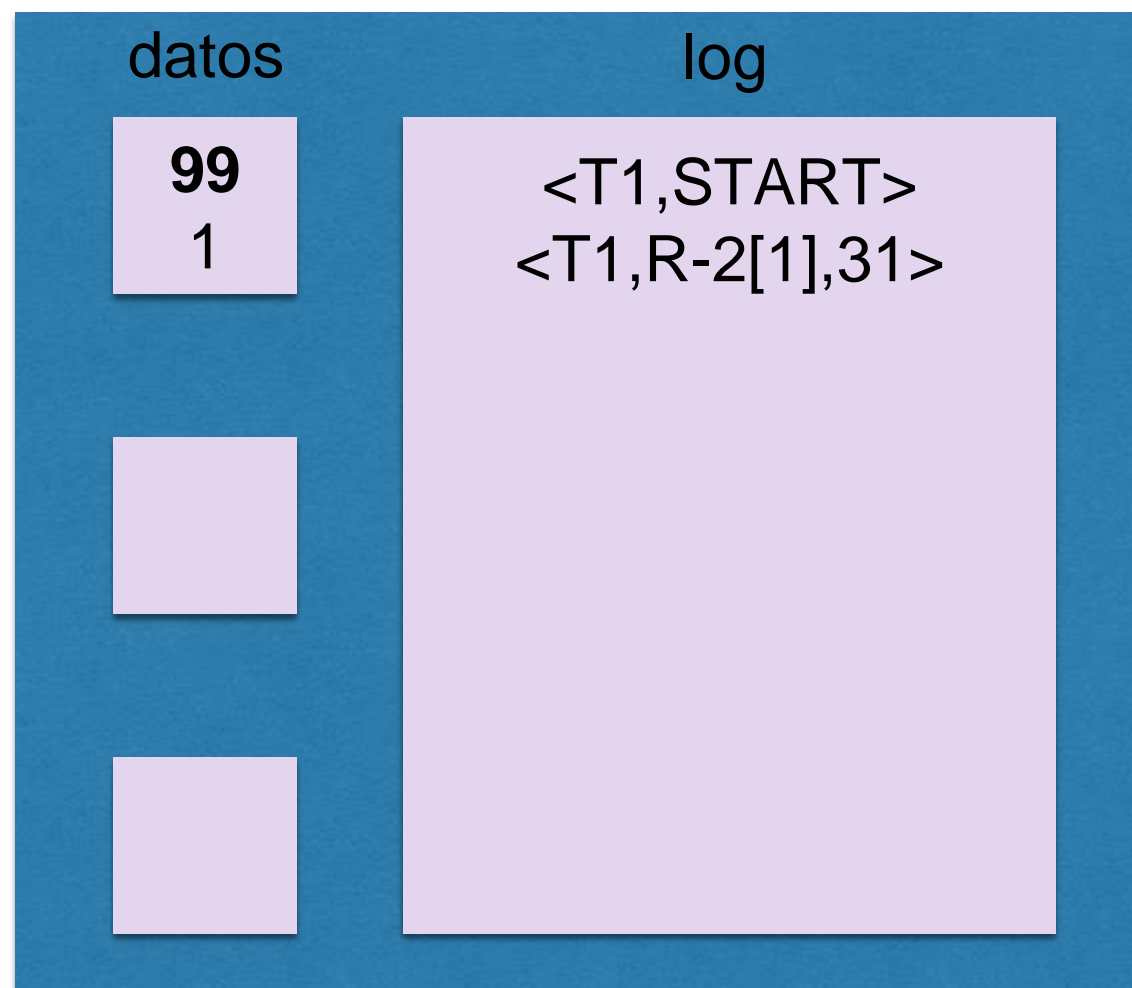


Undo Logging – en la BD

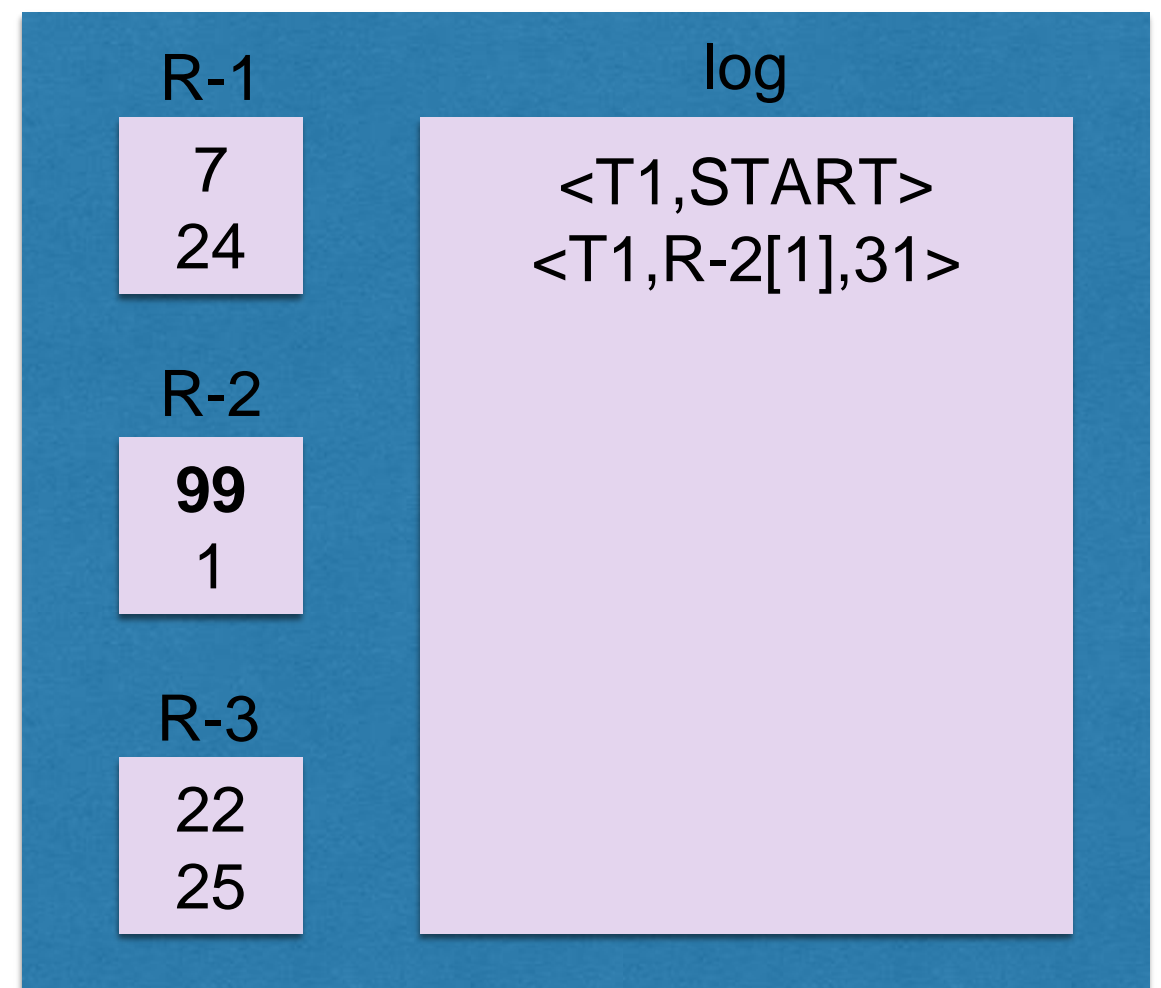
DB

T1: bota la ejecución

Buffer



Disco

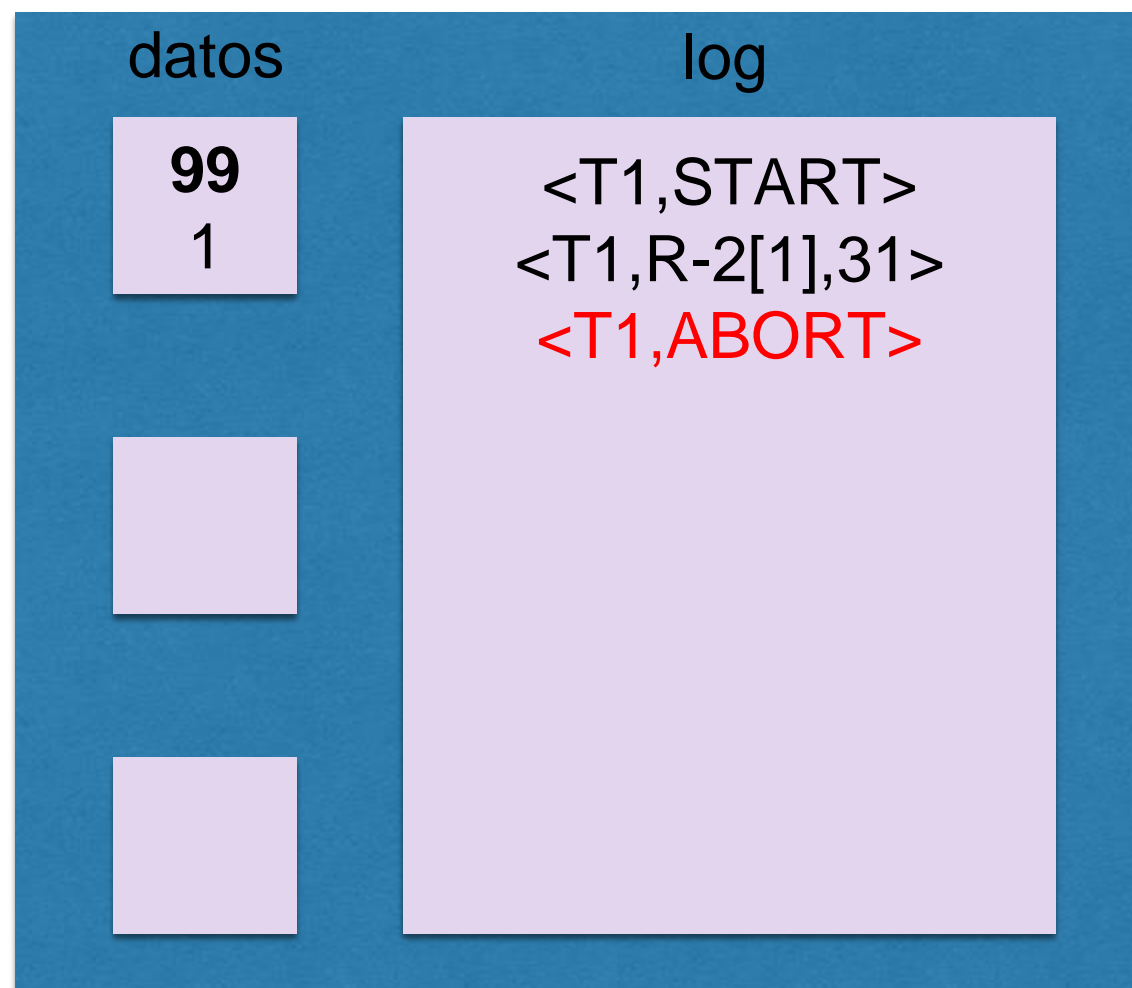


Undo Logging – en la BD

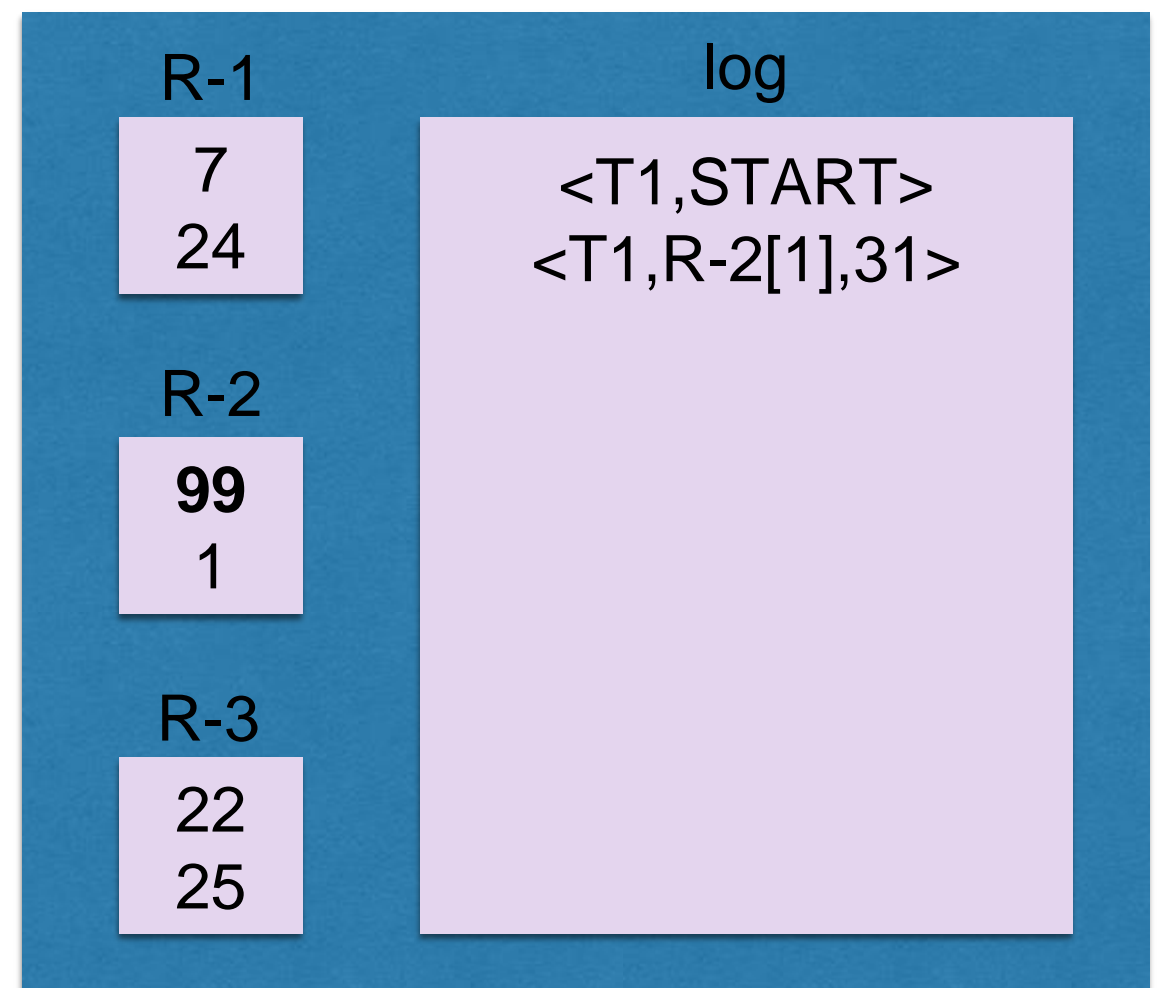
DB

T1: bota la ejecución

Buffer



Disco



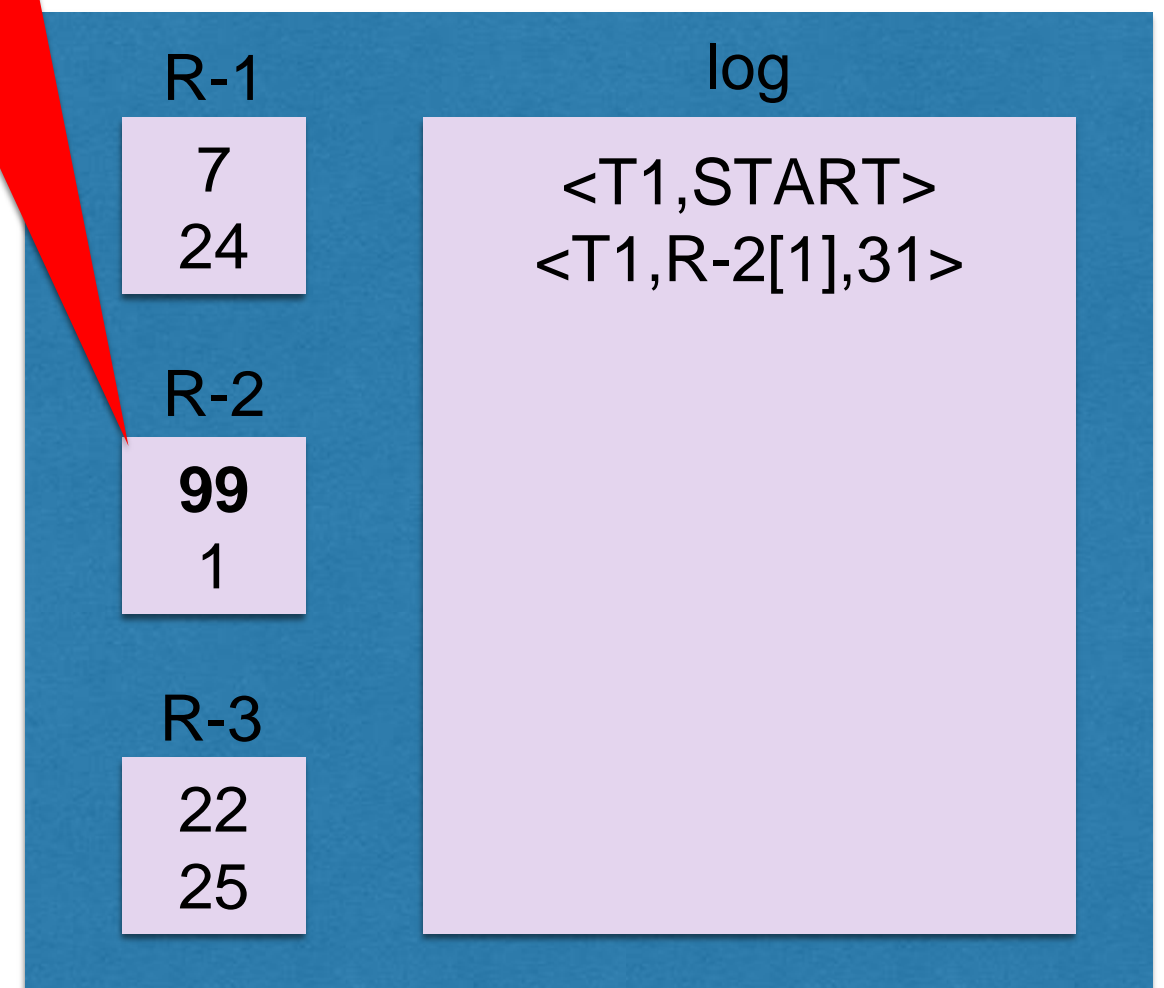
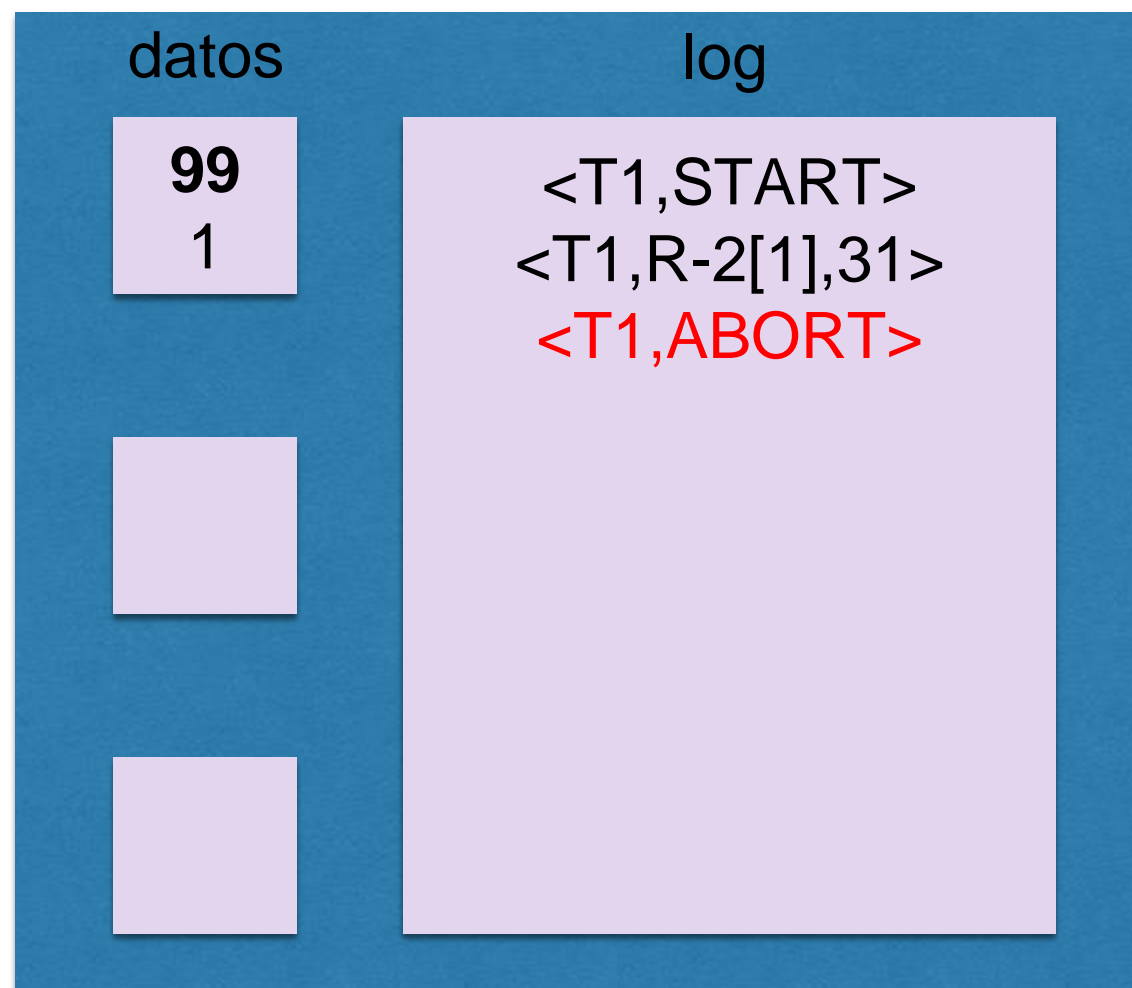
Undo Logging – en la BD

DB

Asegurate deshacer los
cambios!

Buffer

Disco



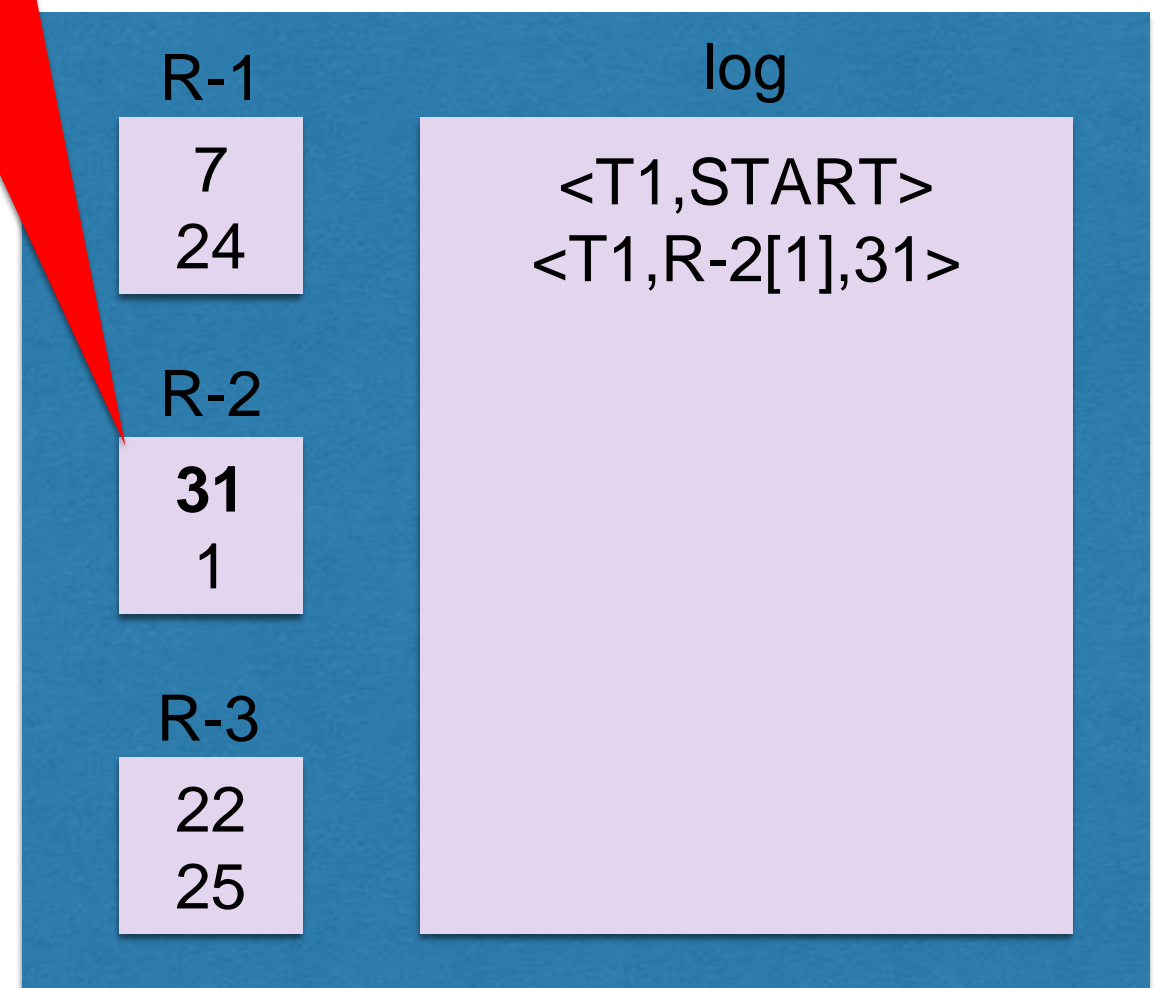
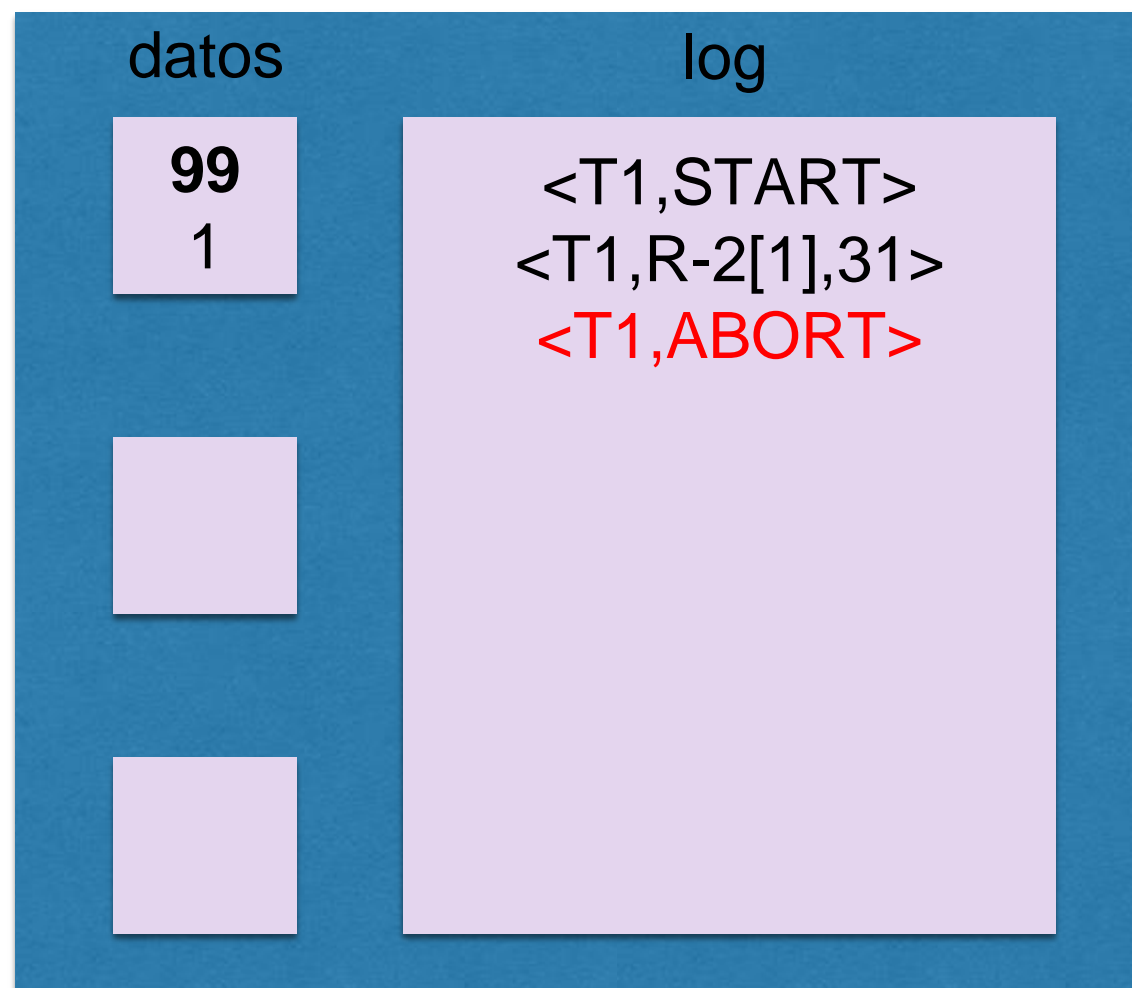
Undo Logging – en la BD

DB

Asegurate deshacer los
cambios!

Buffer

Disco

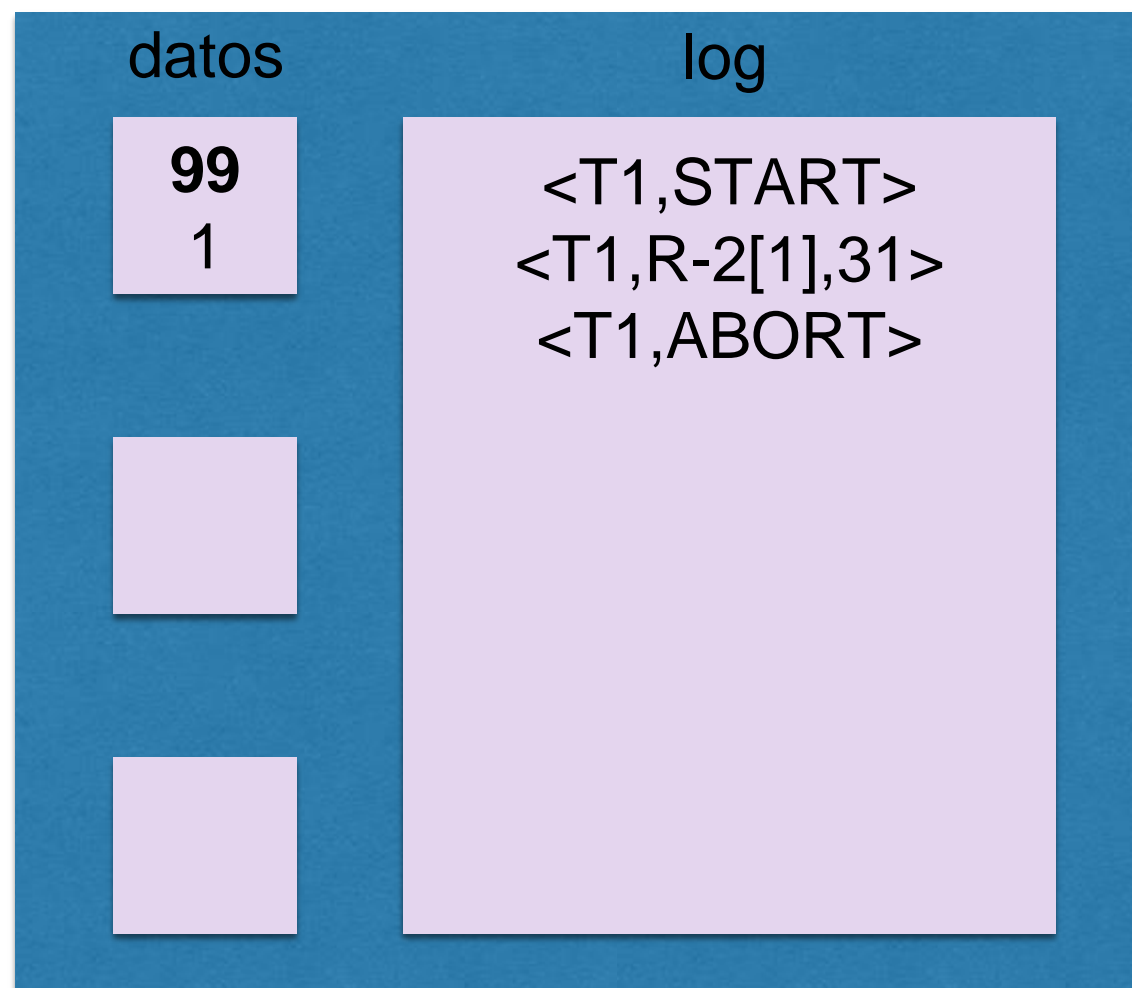


Undo Logging – en la BD

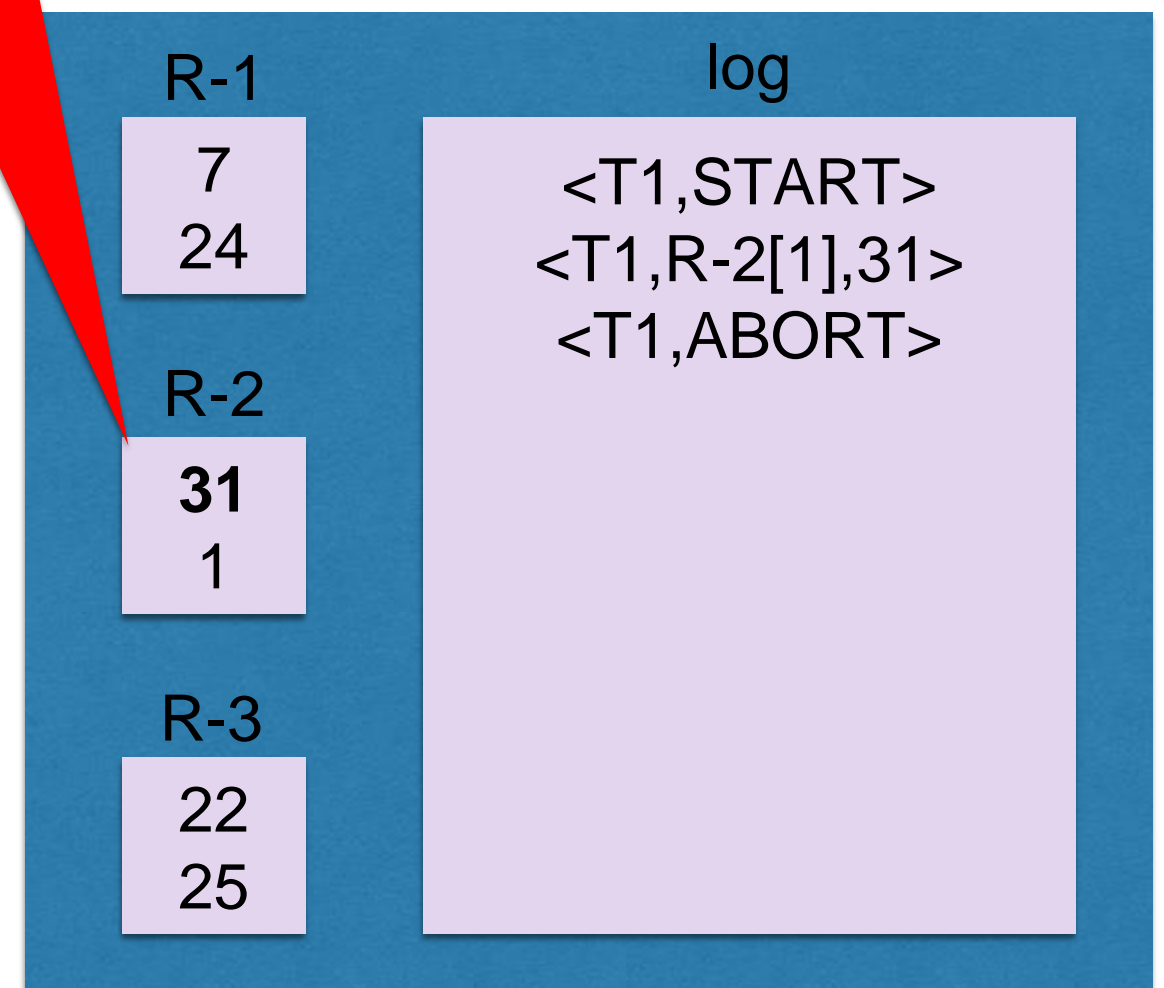
DB

Asegurate deshacer los cambios!

Buffer



Disco



Recuperación con Undo Logging

Detectando fallas en el *log*:

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



Recuperación con Undo Logging

Detectando fallas en el *log*:

... <START T> ...

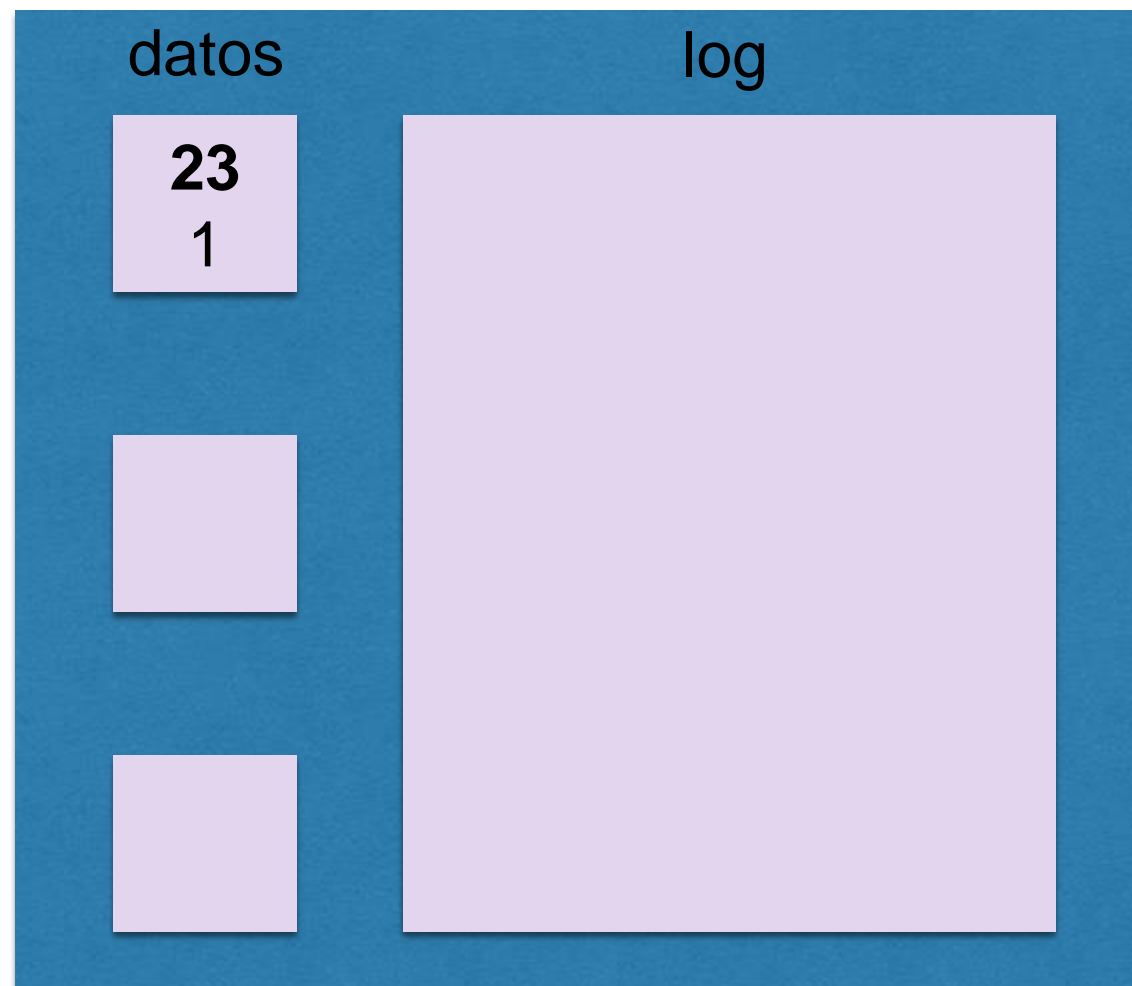


Undo Logging – en la BD

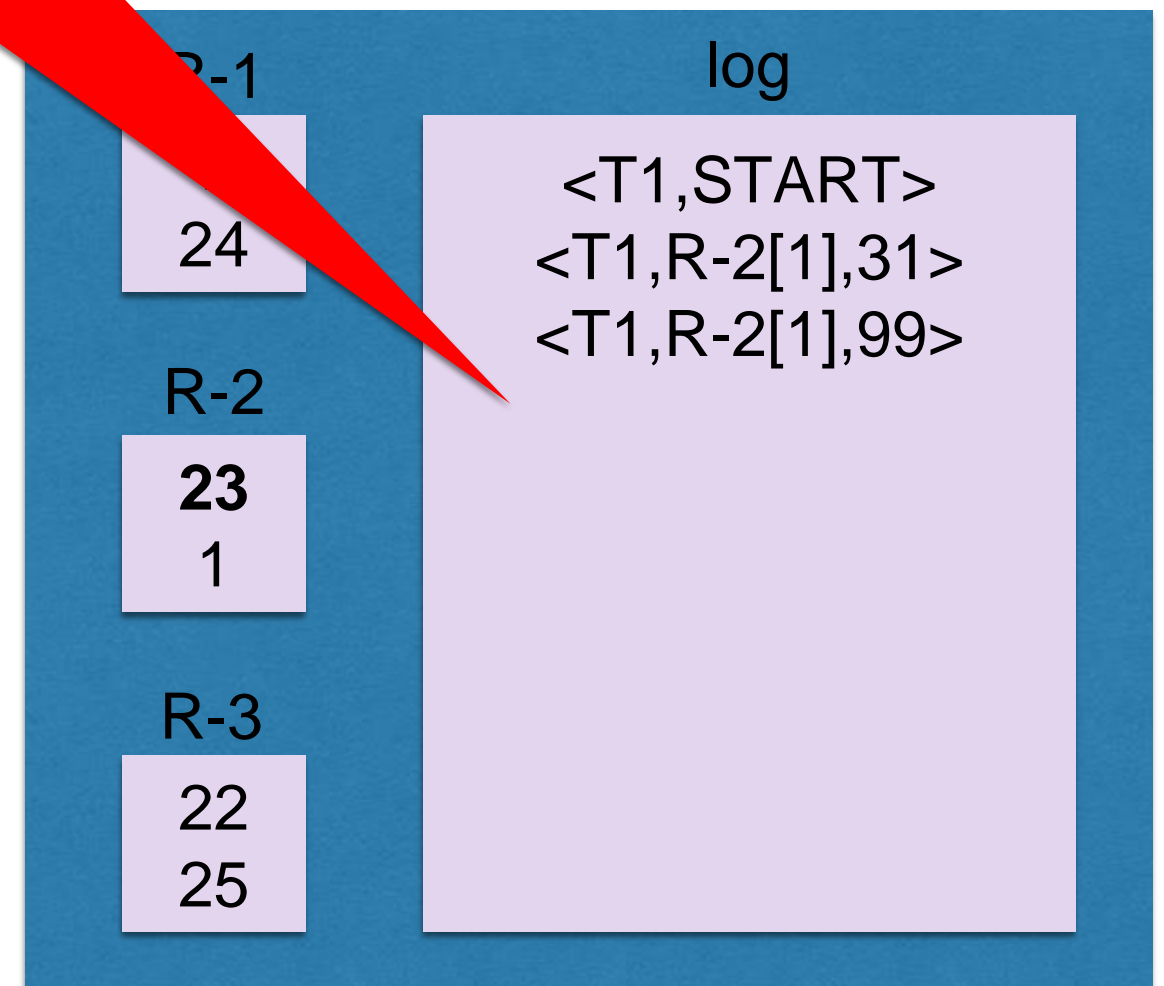
DB

Puedo hacer UNDO
(no sé si los datos están en disco)

Buffer



Disco



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 $\langle \text{COMMIT } T \rangle$, marco T como realizada
- Si leo $\langle \text{ABORT } T \rangle$, marco T como realizada
- Si leo $\langle T, X, t \rangle$, debo restituir $X := t$ en disco, si no fue realizada.
- Si leo $\langle \text{START } T \rangle$, 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* $\langle \text{START CKPT } (T_1, \dots, T_n) \rangle$, donde T_1, \dots, T_n son transacciones activas
- Esperamos hasta que T_1, \dots, T_n terminen, sin restringir nuevas transacciones
- Cuando T_1, \dots, T_n hayan terminado, escribimos $\langle \text{END CKPT} \rangle$

Undo Recovery

Uso de *Nonquiescent Checkpoints*

- Avanzamos desde el final al inicio
- Si encontramos un $\langle \text{END CKPT} \rangle$, hacemos *undo* de todo lo que haya *empezado* después del inicio del *checkpoint*
- Si encontramos un $\langle \text{START CKPT } (T_1, \dots, T_n) \rangle$ sin su $\langle \text{END CKPT} \rangle$, debemos analizar el log desde el inicio de la transacción más antigua entre T_1, \dots, T_n

Ejemplo

Uso de *Checkpoints* en *Undo Logging*

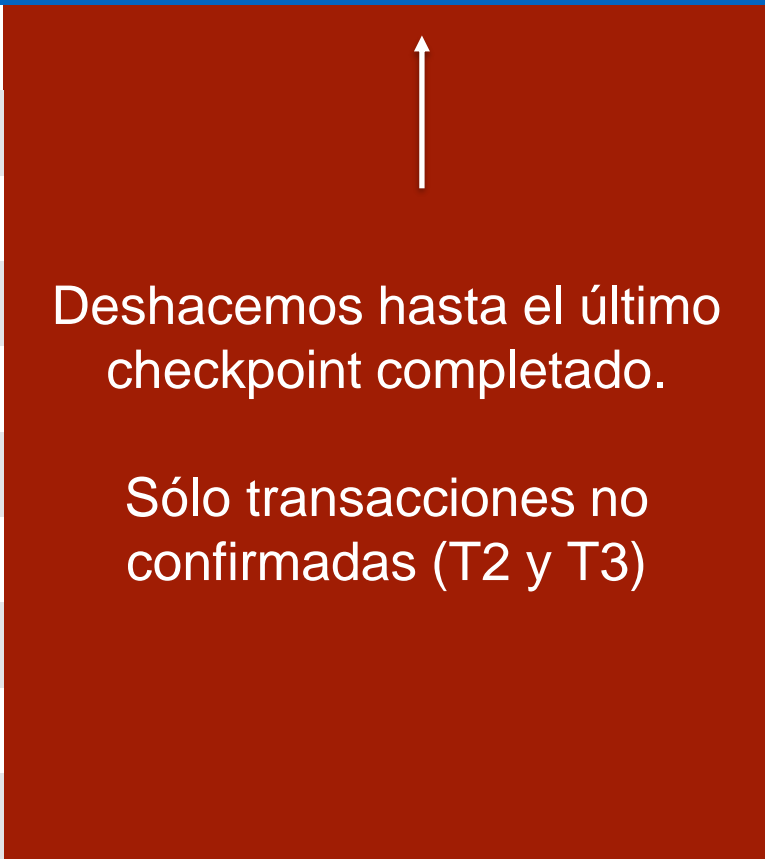
Considere este *log* después de una falla:

Log	
<START T1>	Podemos truncar esta parte
<T1, a, 5>	
<START T2>	
<T2, b, 10>	
<START CKPT (T1, T2)>	T1 y T2 activas
<T2, c, 15>	Deshacemos solo esta parte
<START T3>	
<T1, d, 20>	
<COMMIT T1>	Noten que T3 partió después del checkpoint
<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>	 <p>Deshacemos hasta el último checkpoint completado.</p> <p>Sólo transacciones no confirmadas (T2 y T3)</p>
<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:

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

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:

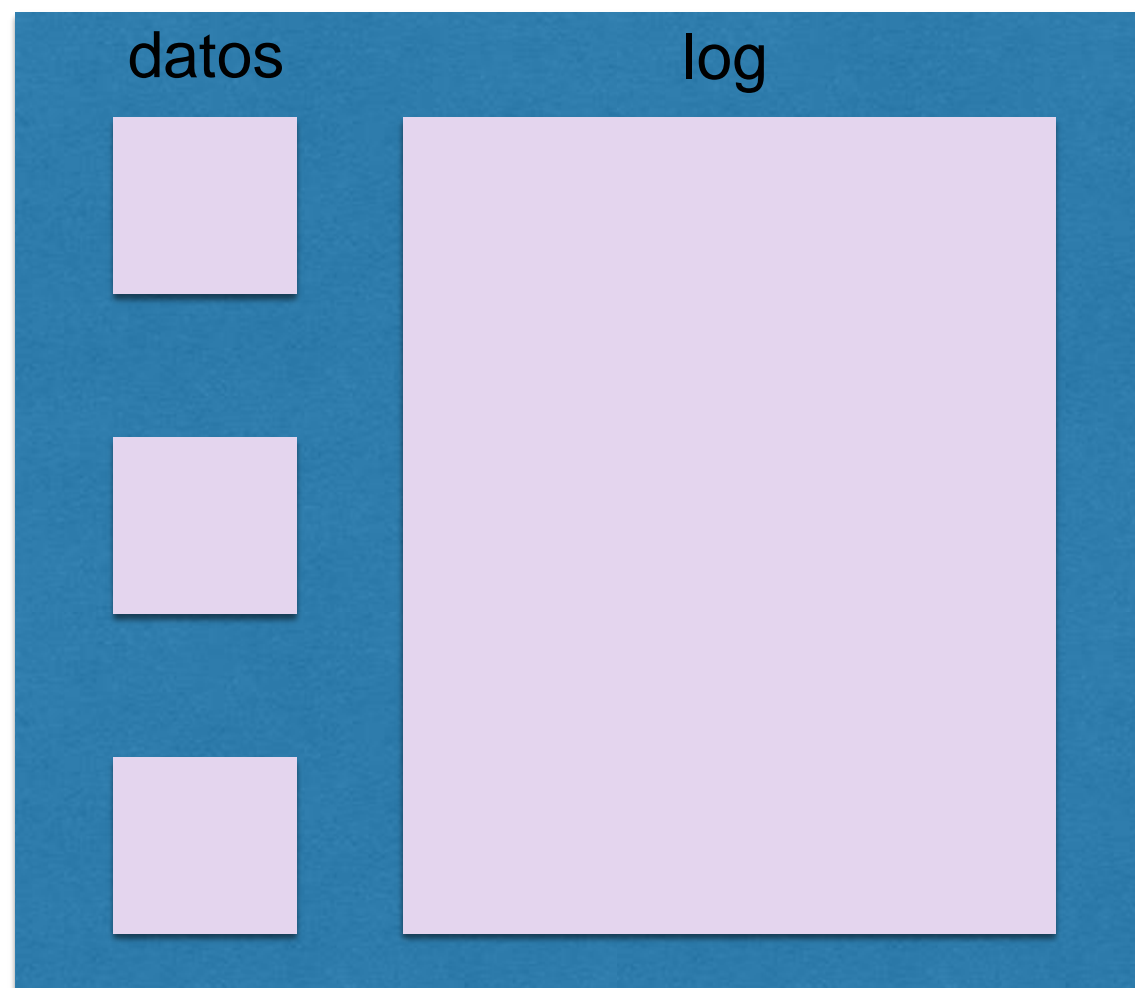
- Escribir el log $\langle \mathbf{T}, X, v \rangle$
- Escribir $\langle \text{COMMIT } \mathbf{T} \rangle$
- Escribir los datos en disco
- Escribir $\langle \mathbf{T}, \text{END} \rangle$ en log (en disco)

Redo Logging – en la BD

DB

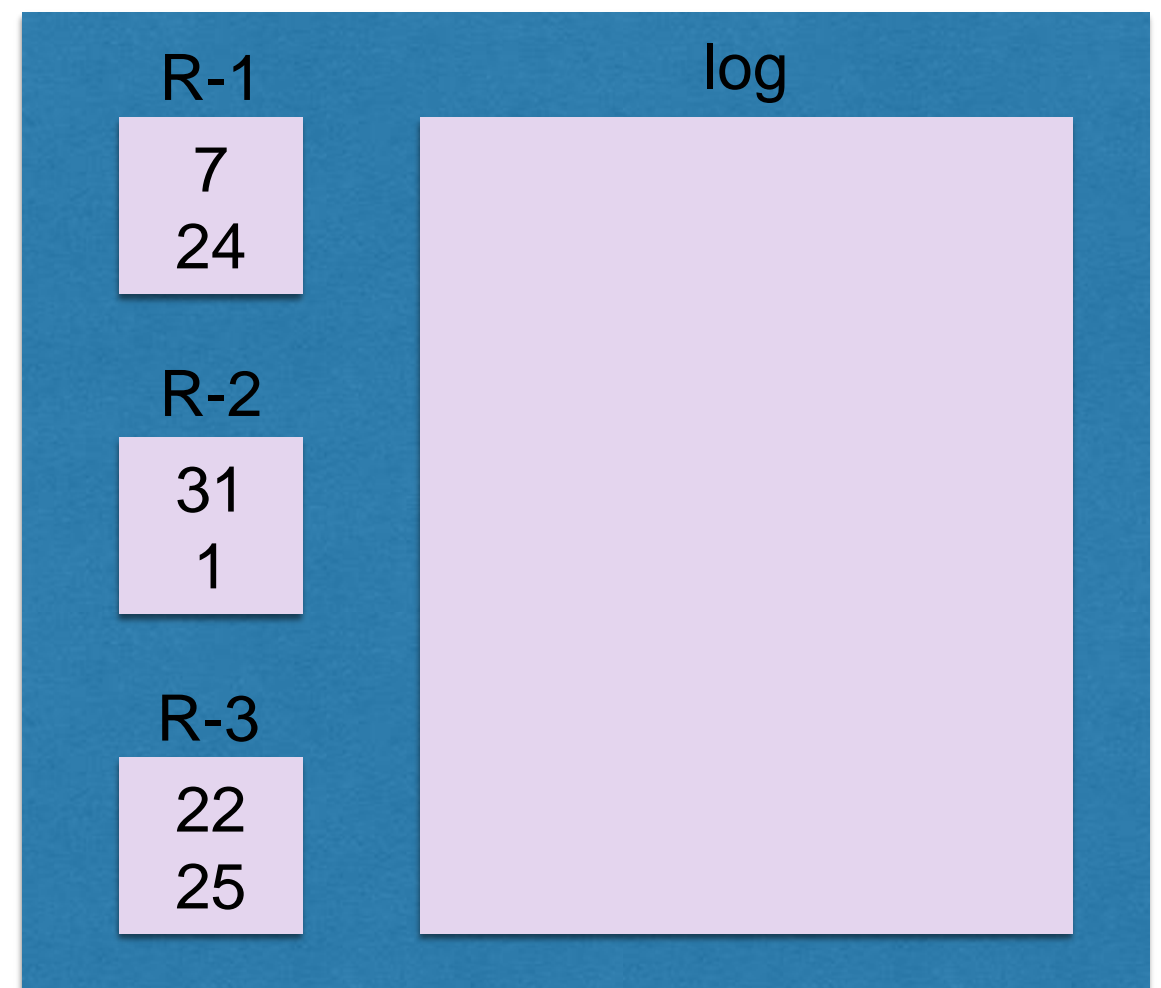
T1: voy a empezar

Buffer



R(A int)

Disco

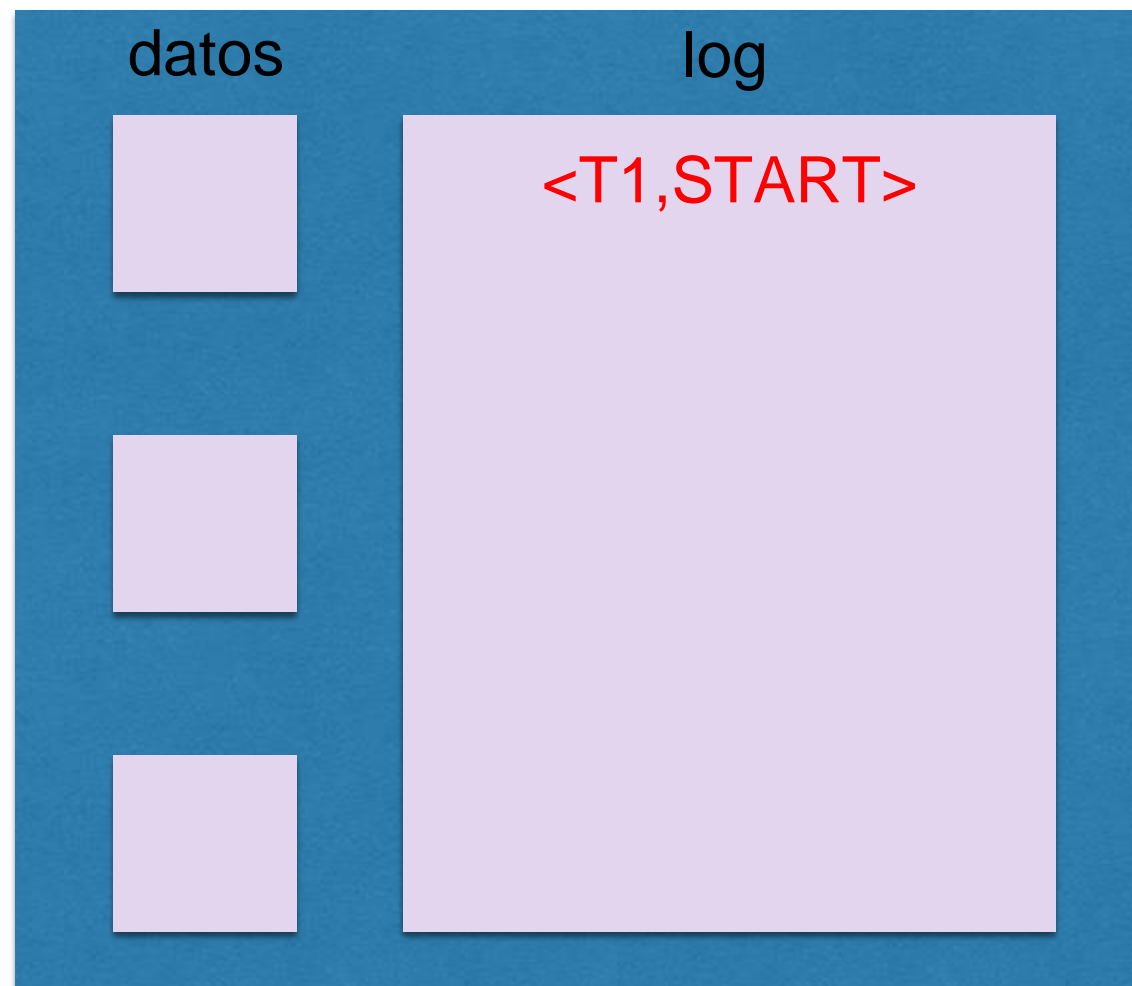


Redo Logging – en la BD

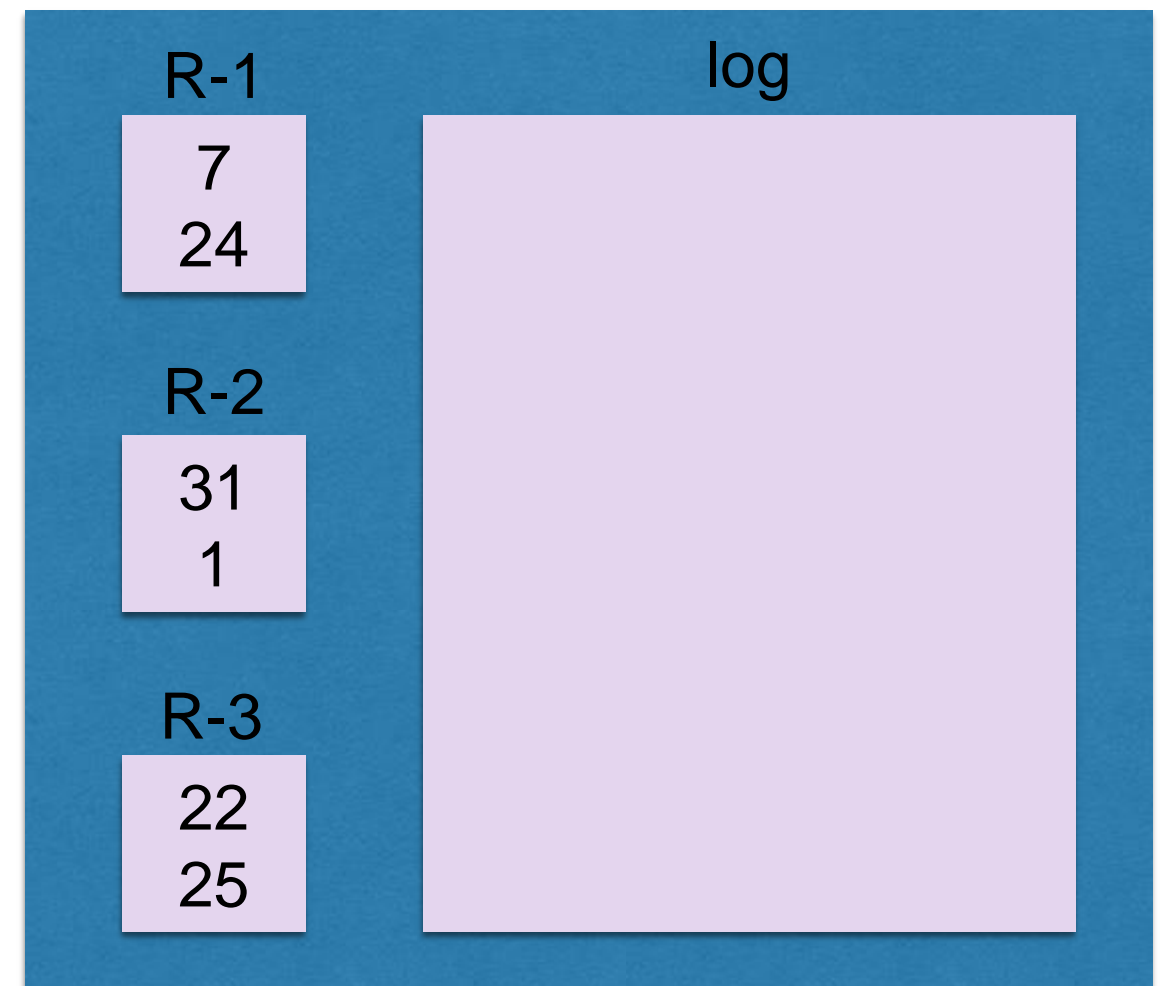
DB

T1: voy a empezar

Buffer



Disco



Redo Logging – en la BD

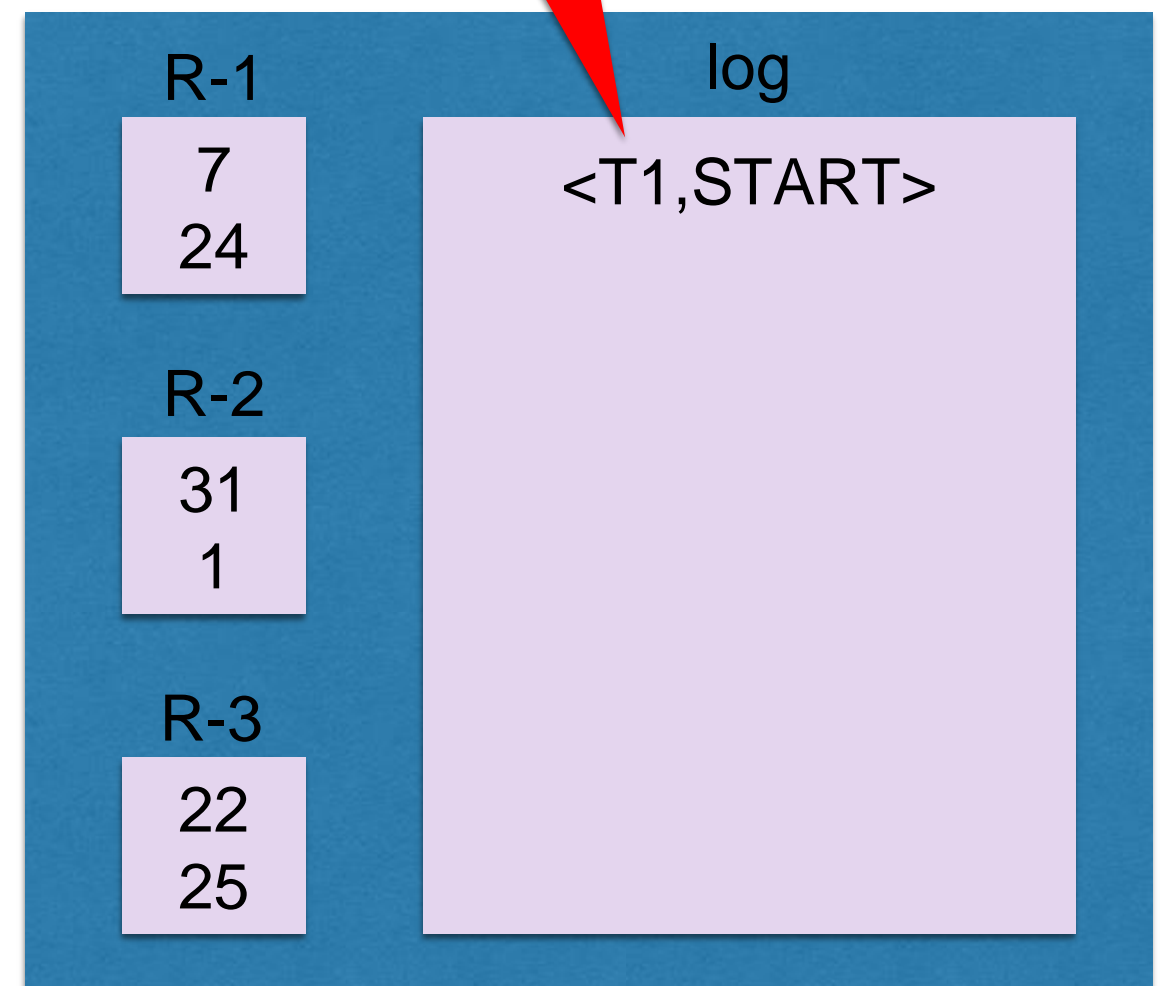
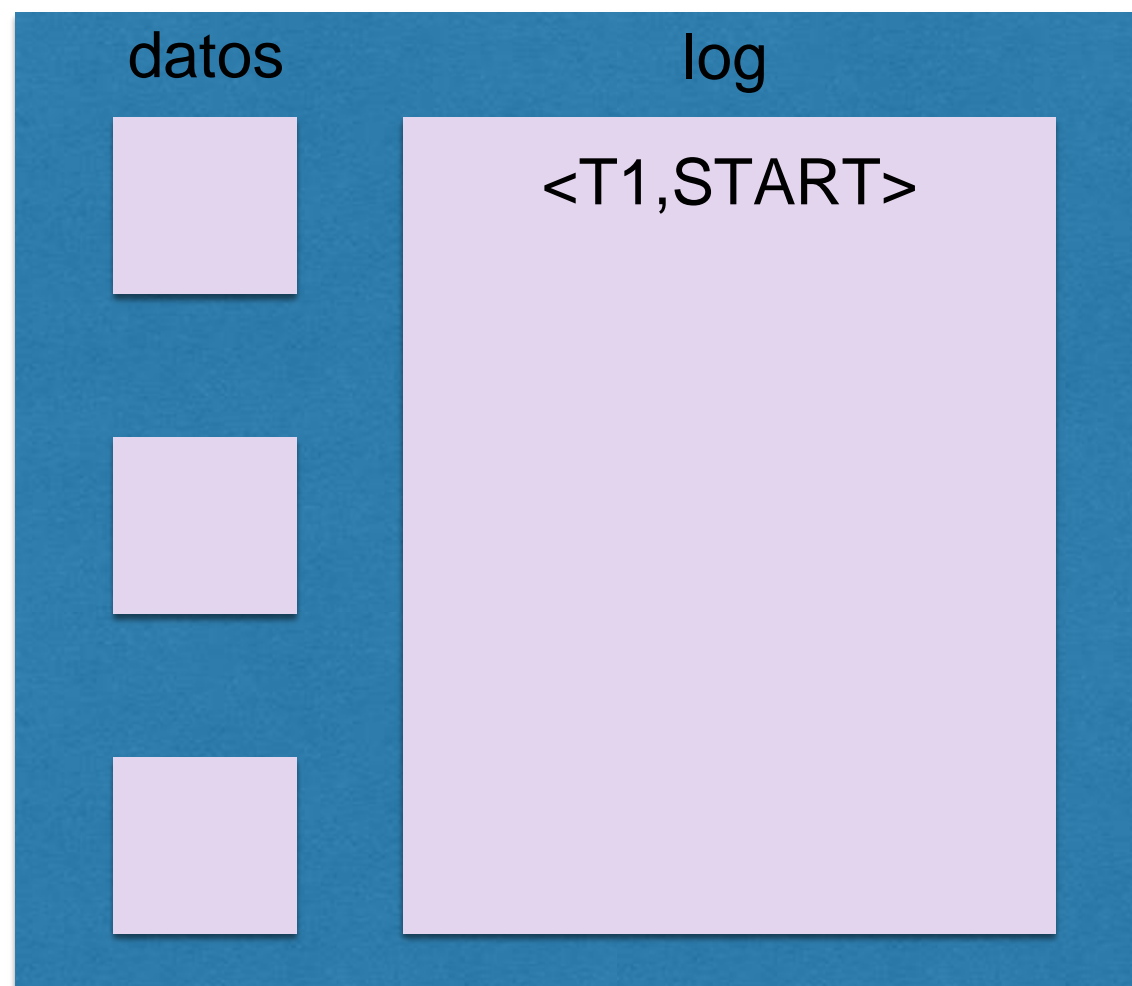
DB

T1: voy a empezar

Incluso puedo escribir
log al disco al tiro

Buffer

Disco

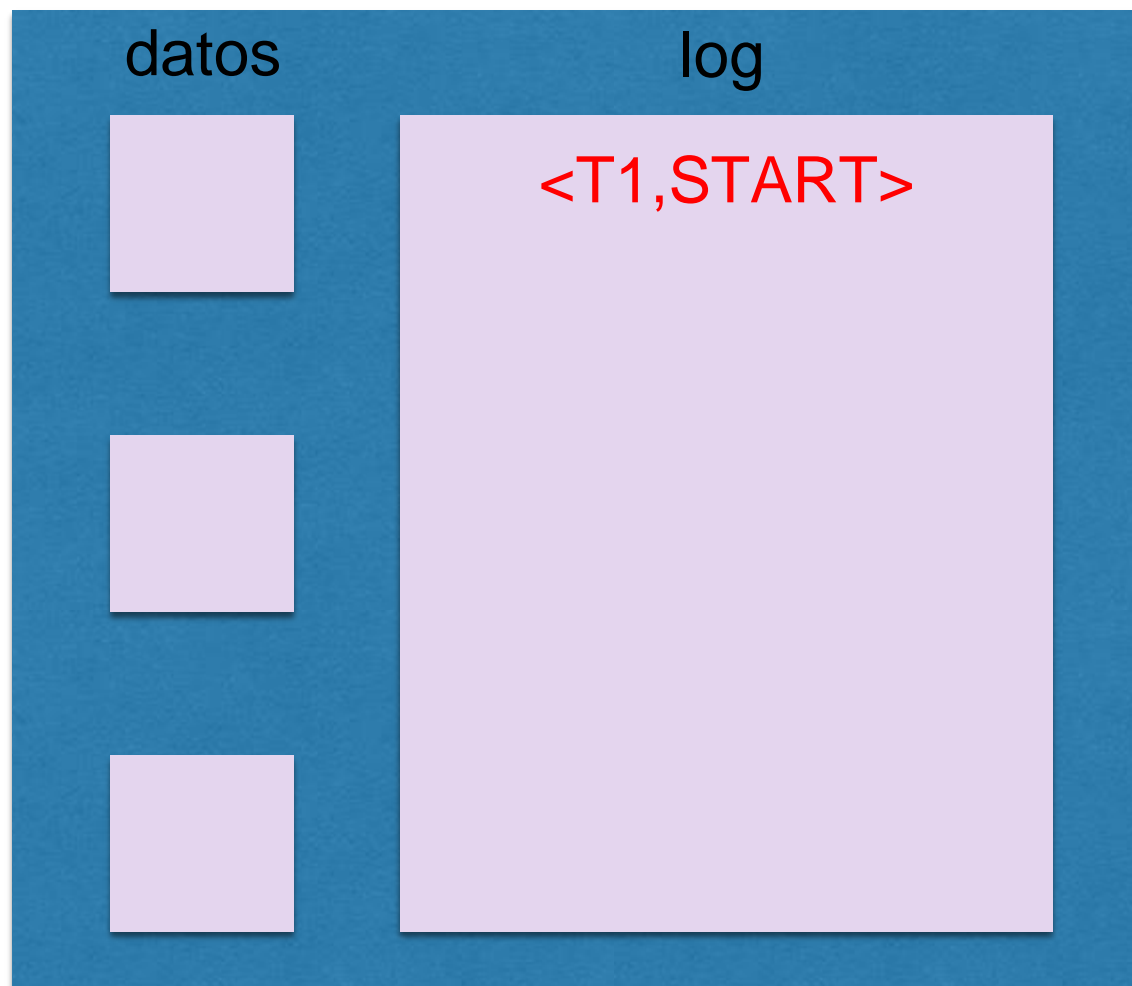


Redo Logging – en la BD

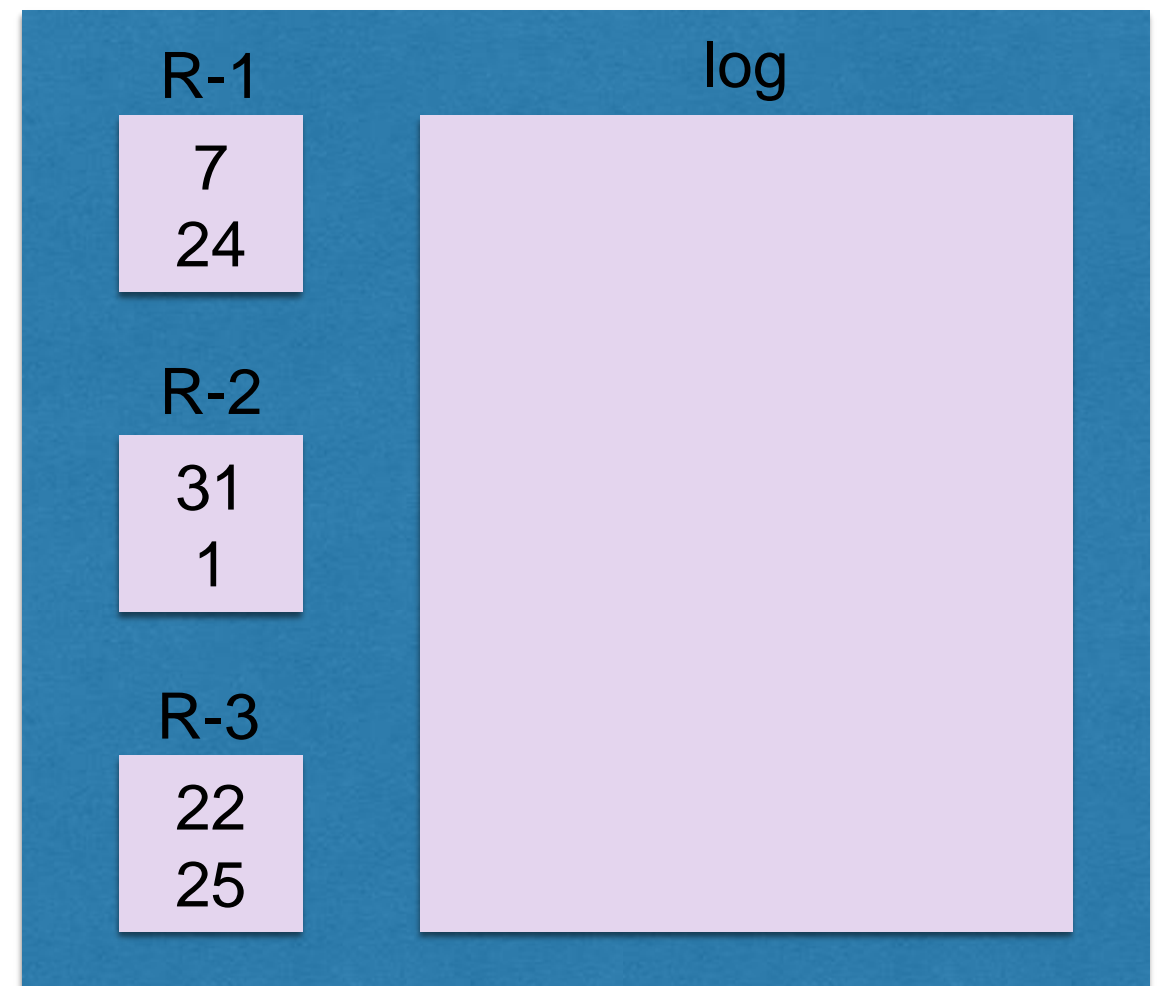
DB

T1: Necesito primera
tupla de página 2 de R!

Buffer



Disco

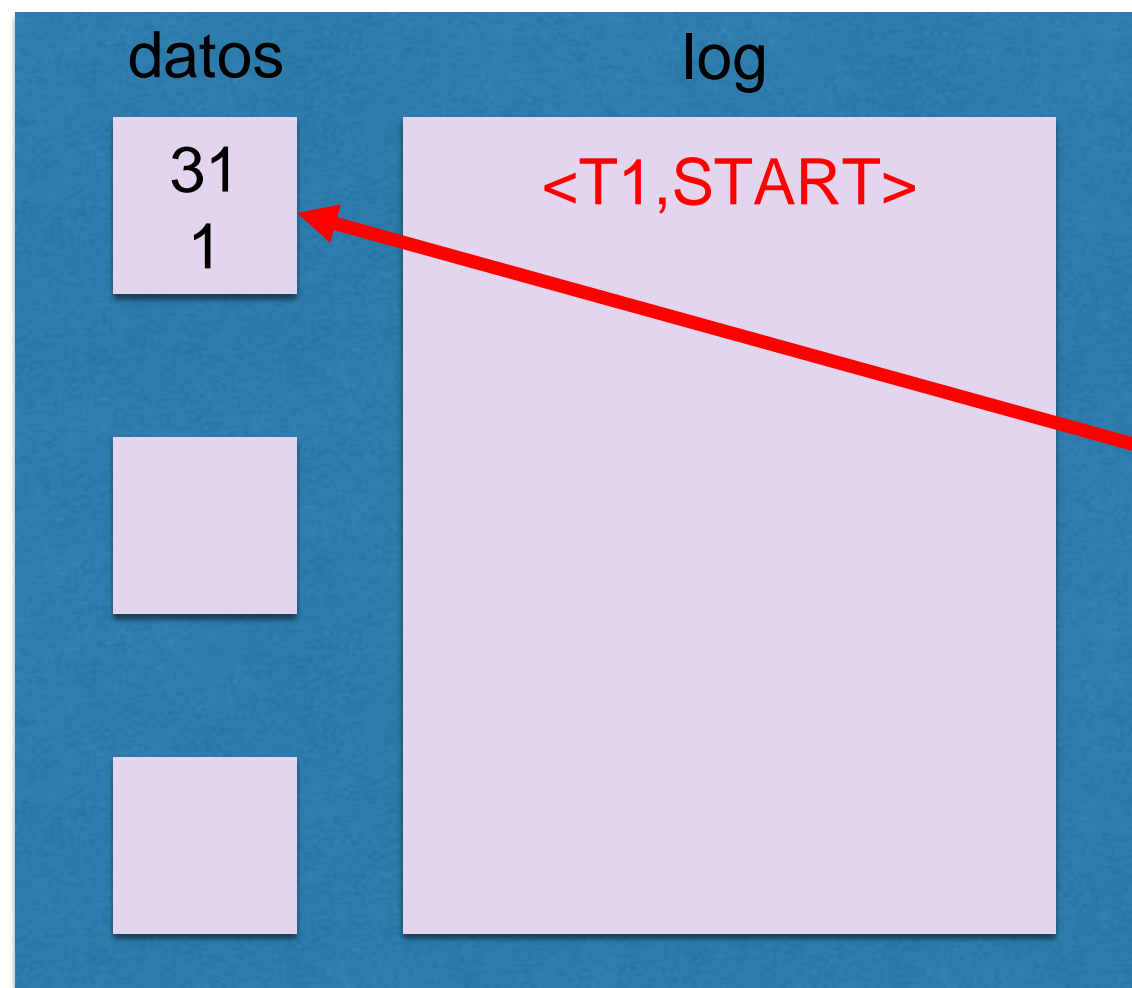


Redo Logging – en la BD

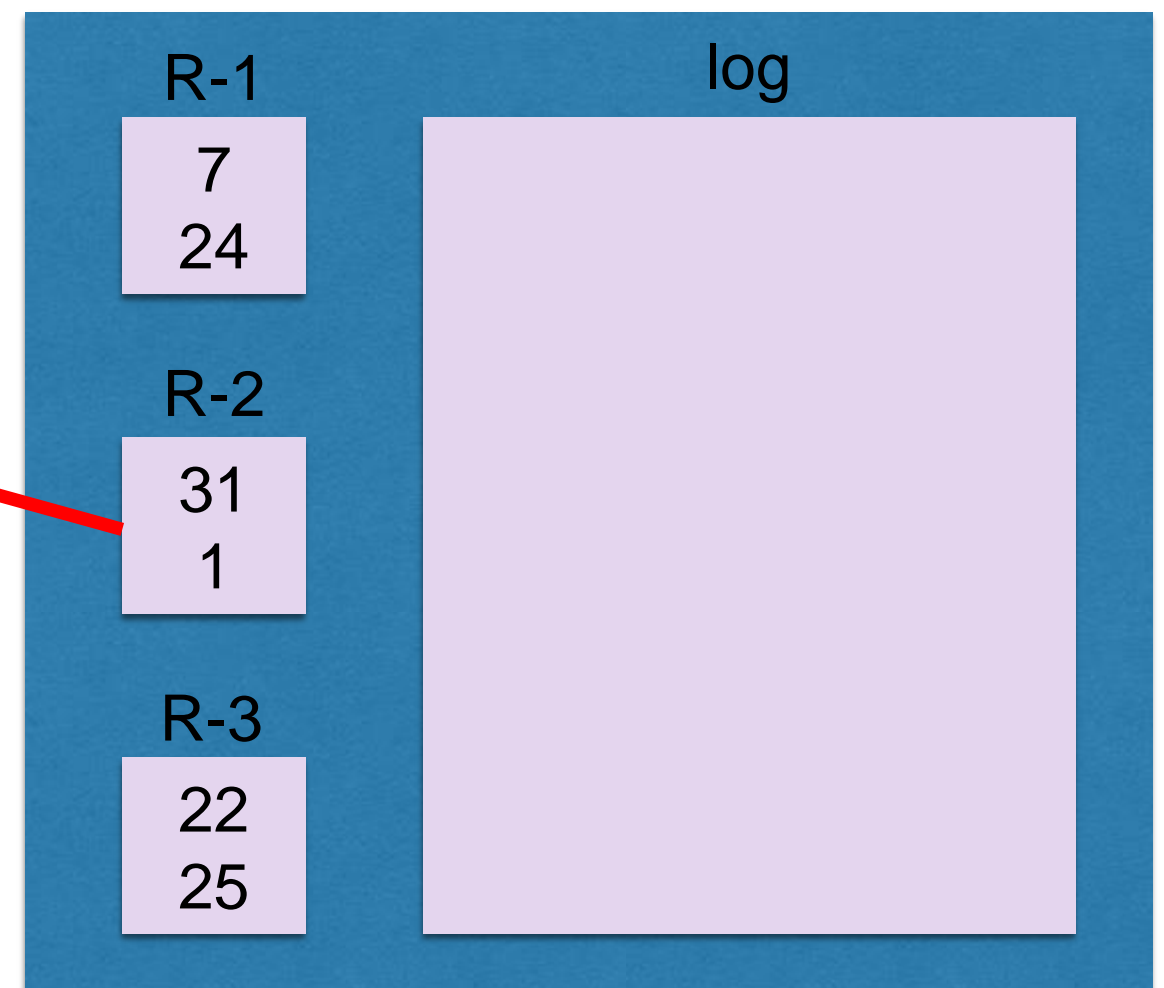
DB

T1: Necesito primera
tupla de página 2 de R!

Buffer



Disco

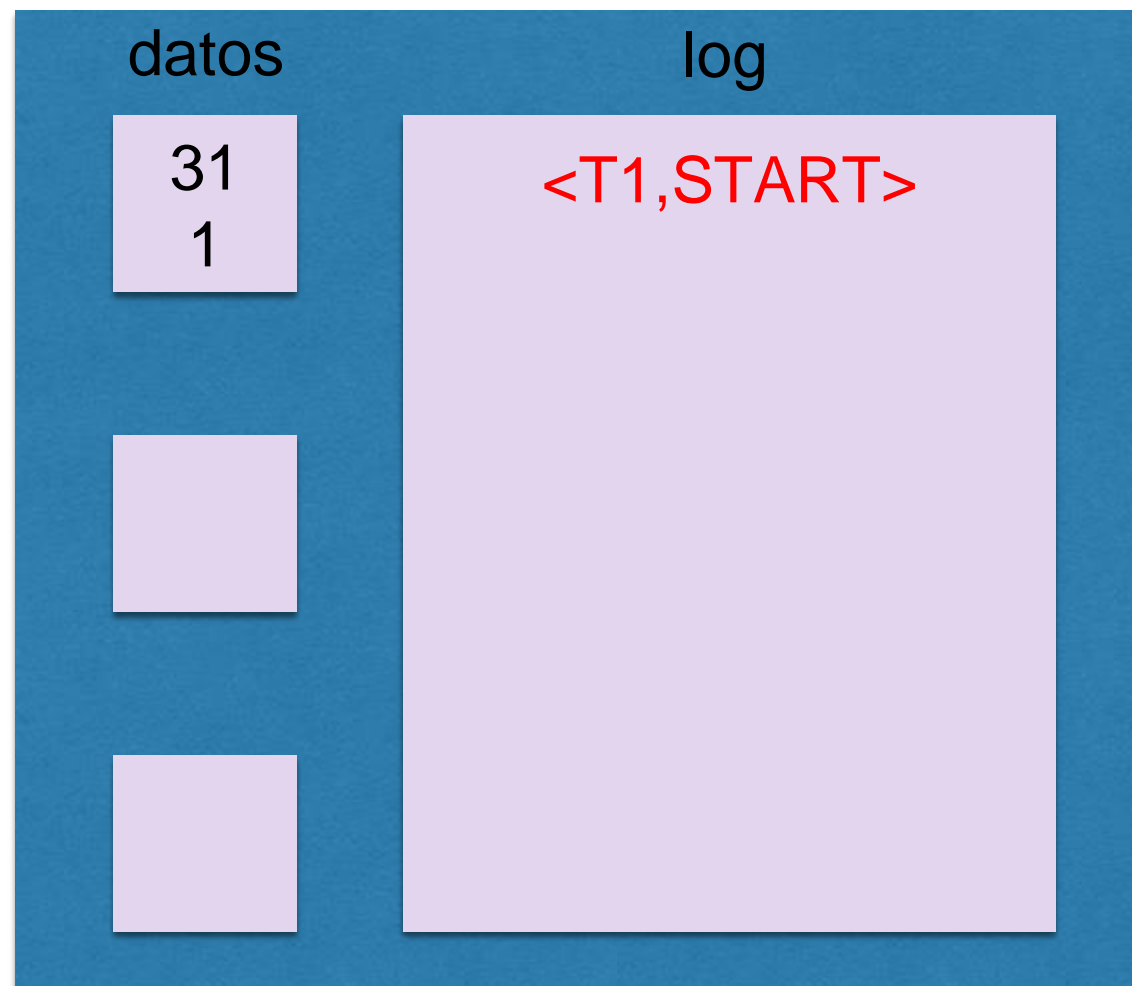


Redo Logging – en la BD

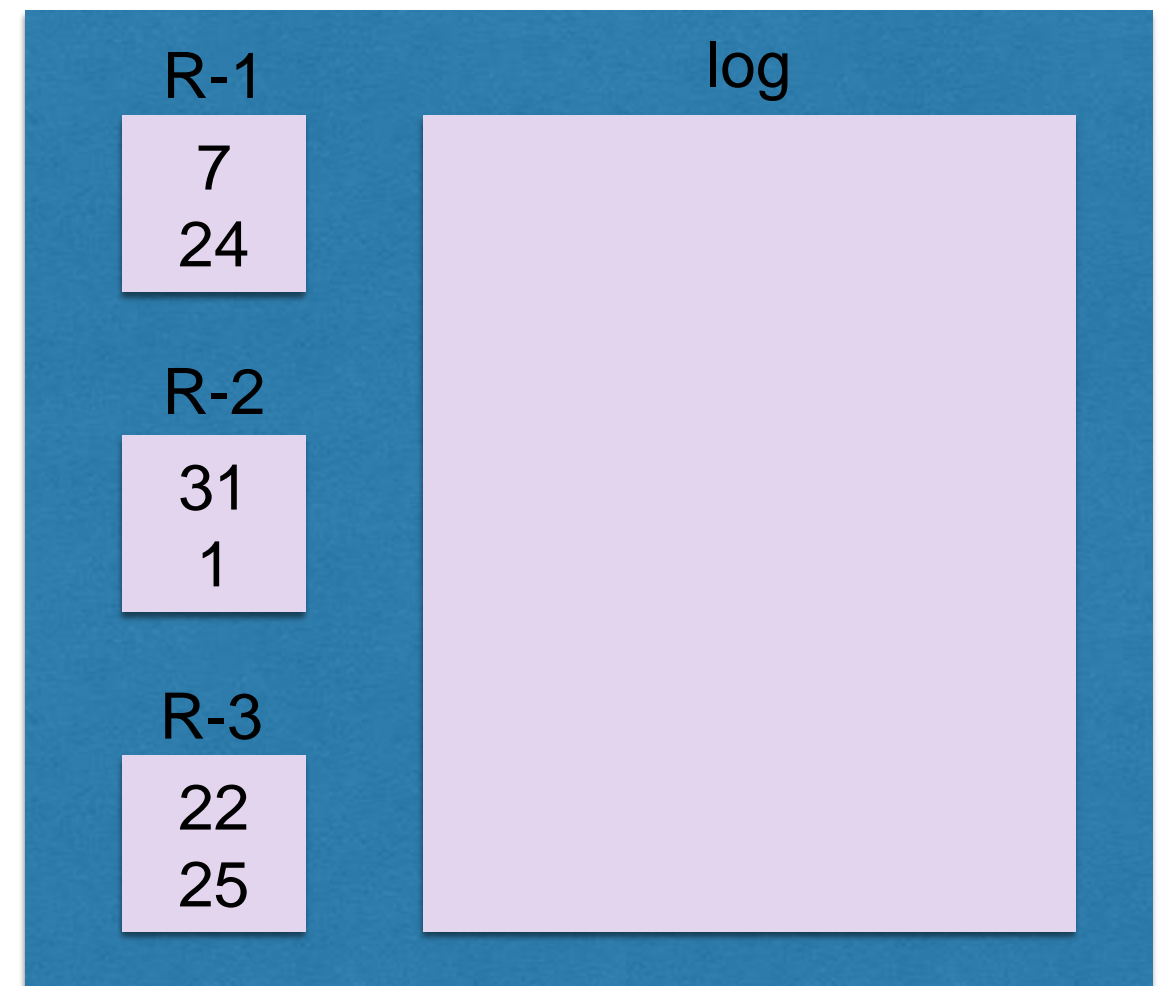
DB

T1: Cambio 31 a 99!

Buffer



Disco

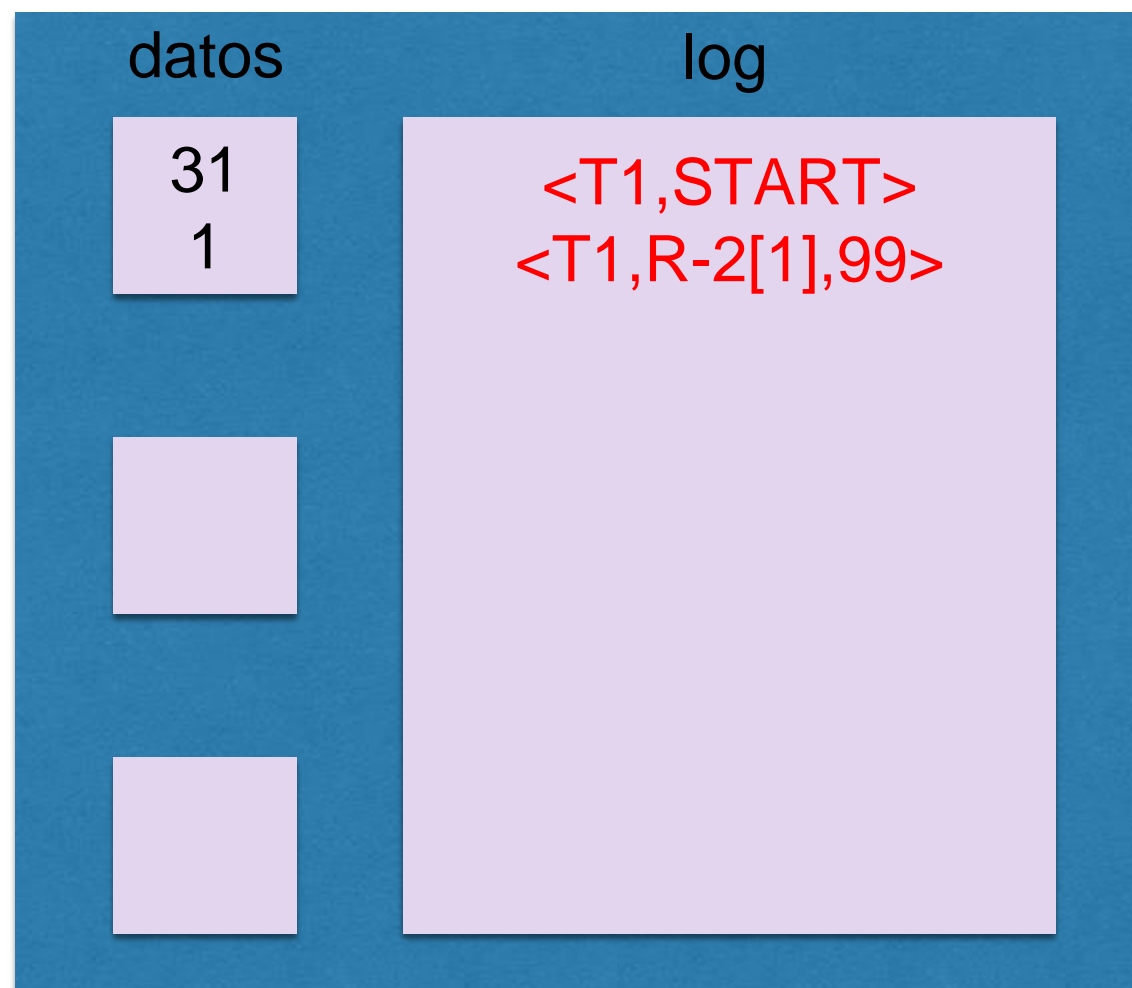


Redo Logging – en la BD

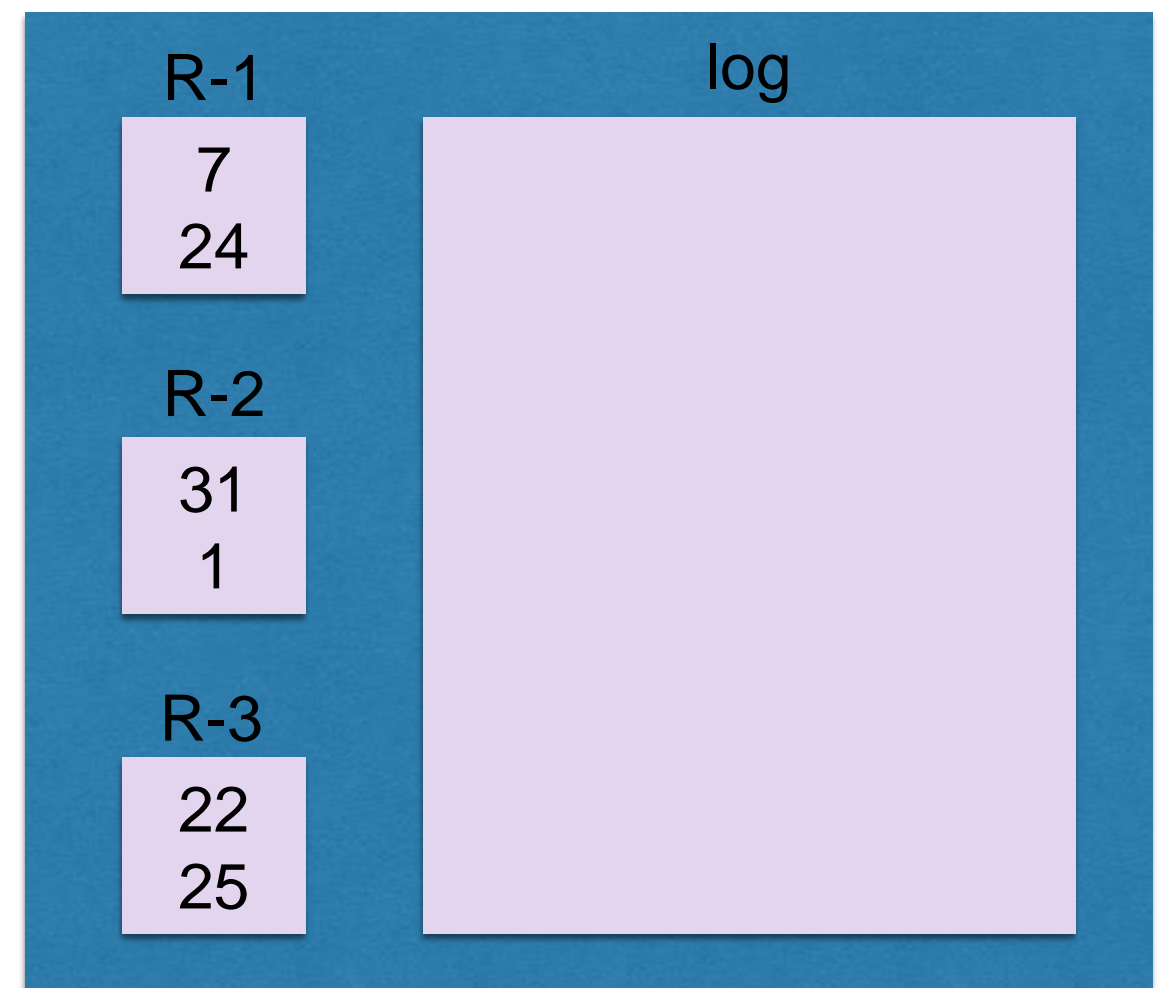
DB

T1: Cambio 31 a 99!

Buffer



Disco

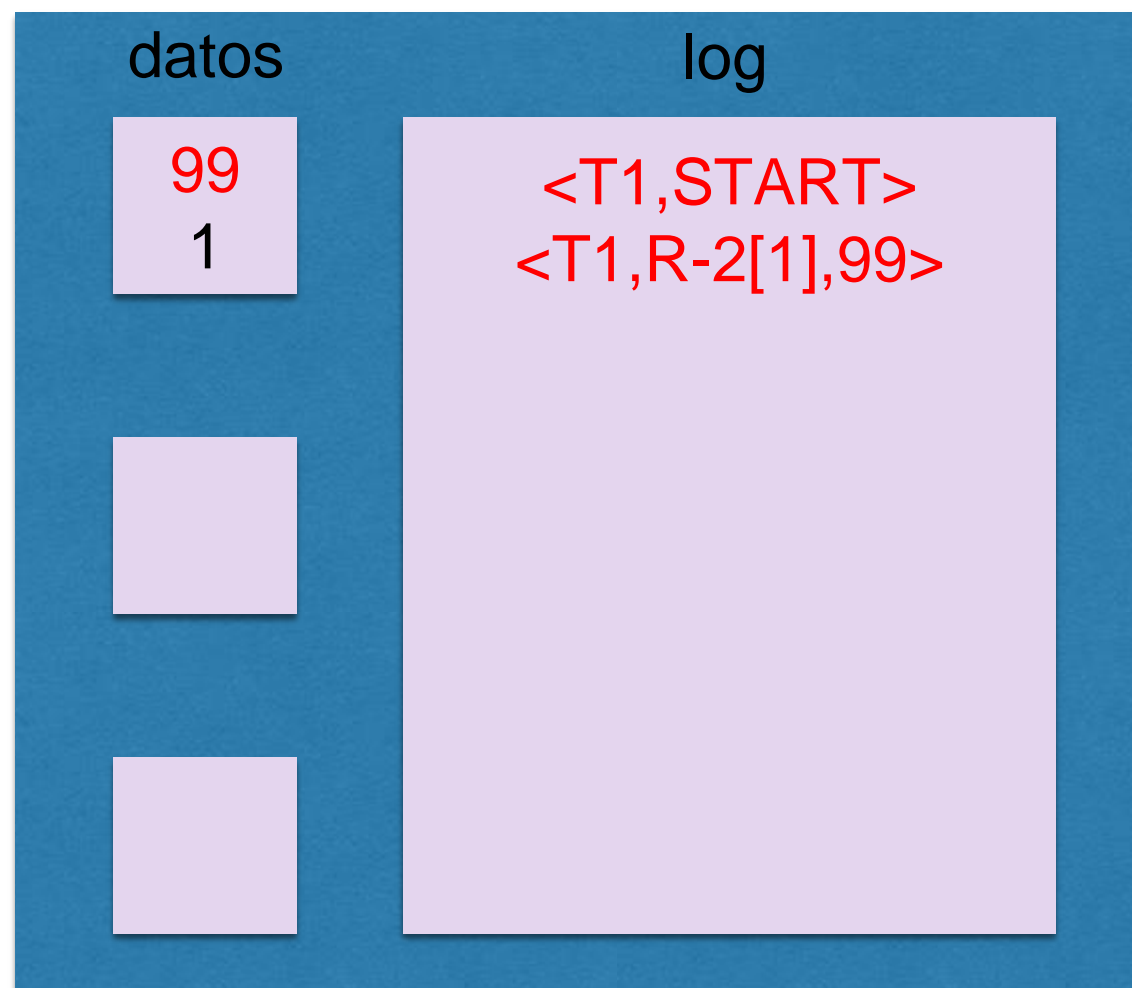


Redo Logging – en la BD

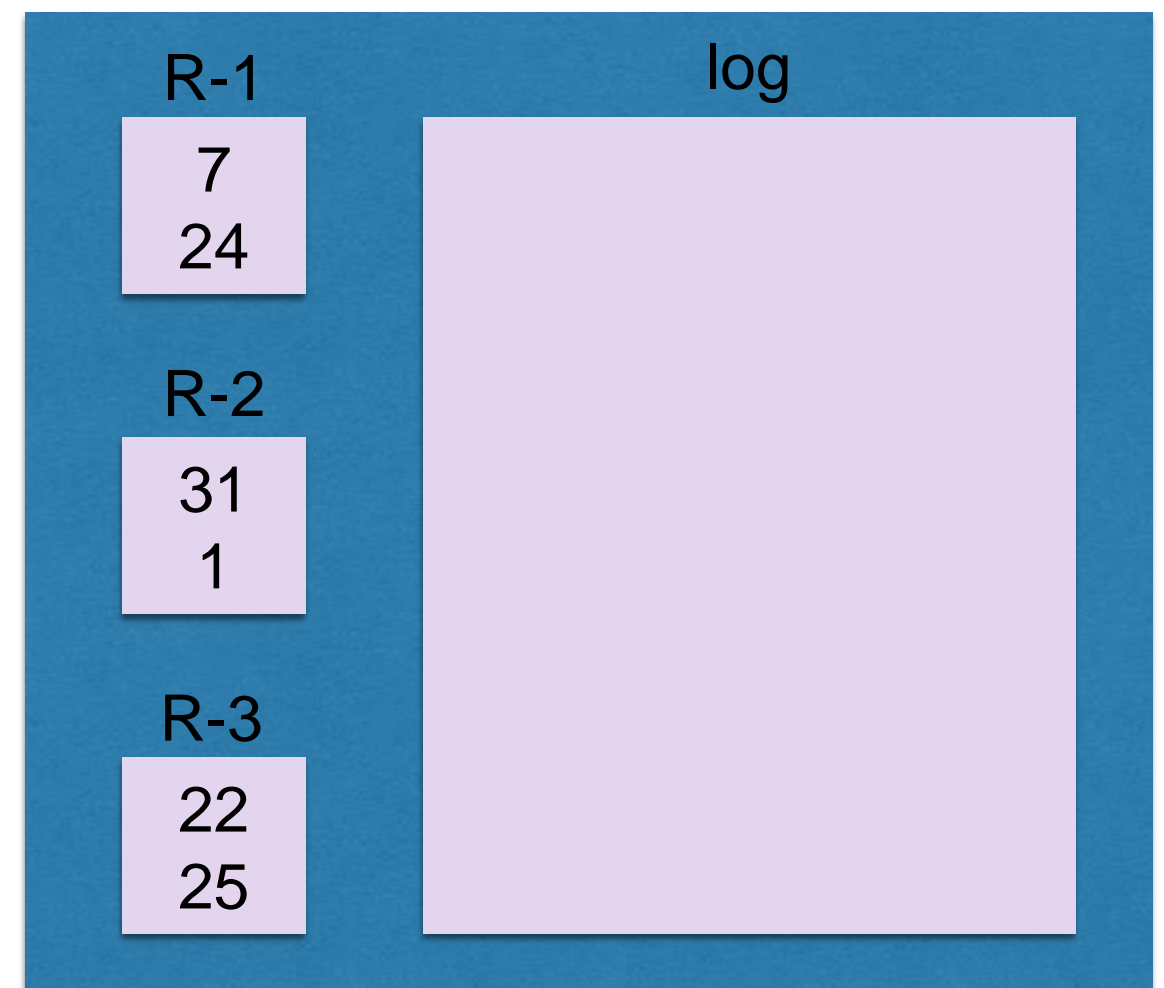
DB

T1: Cambio 31 a 99!

Buffer



Disco

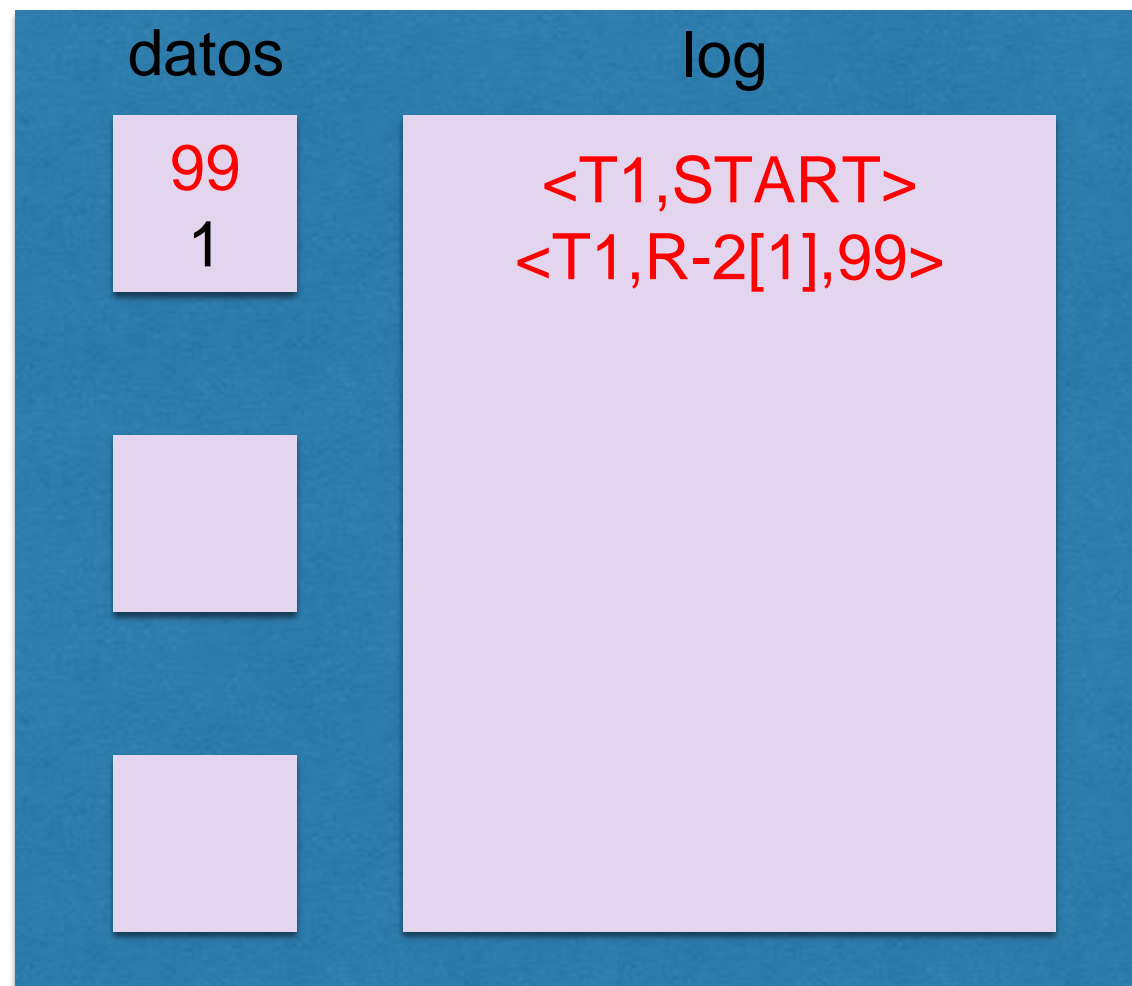


Redo Logging – en la BD

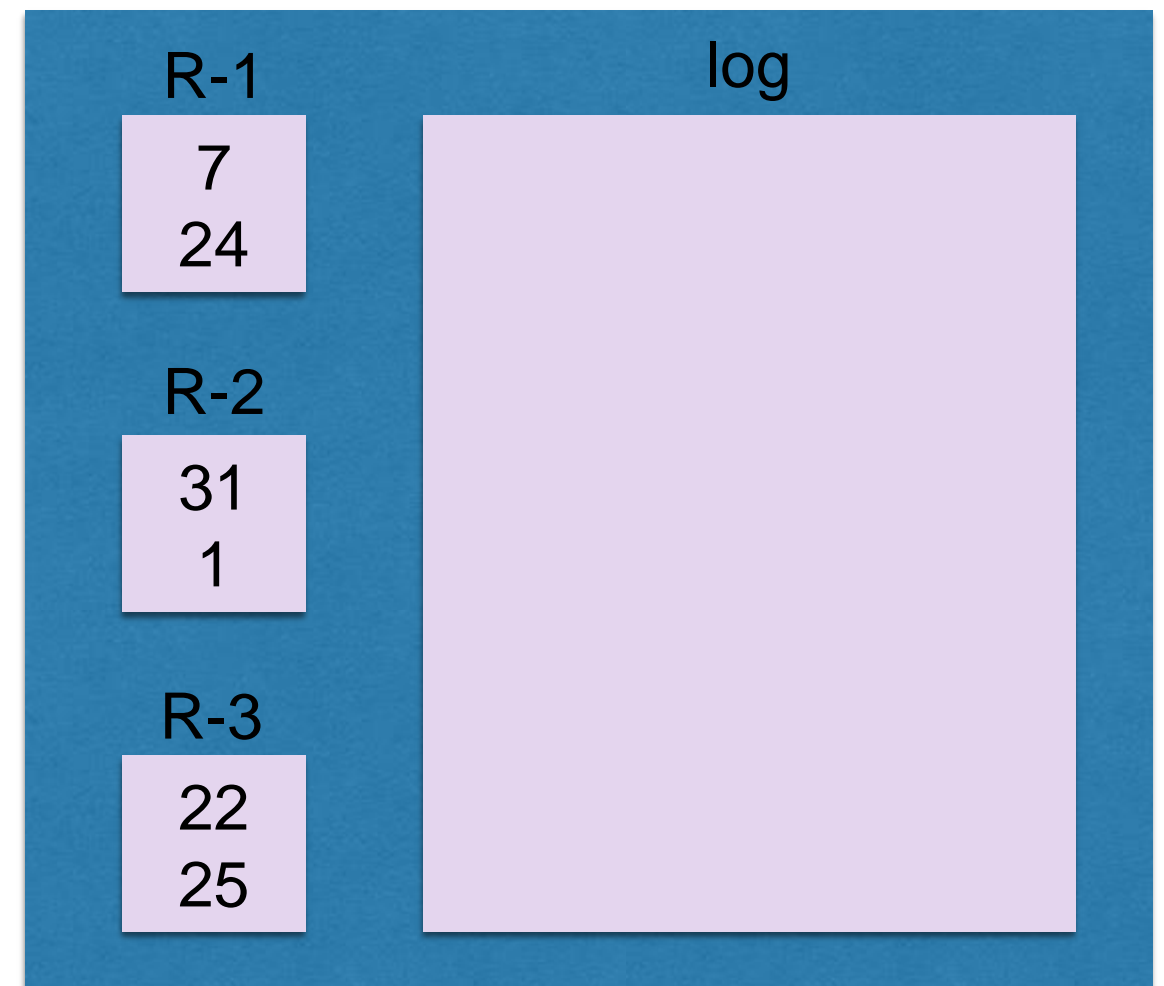
DB

T1: Cambio 99 a 23!

Buffer



Disco

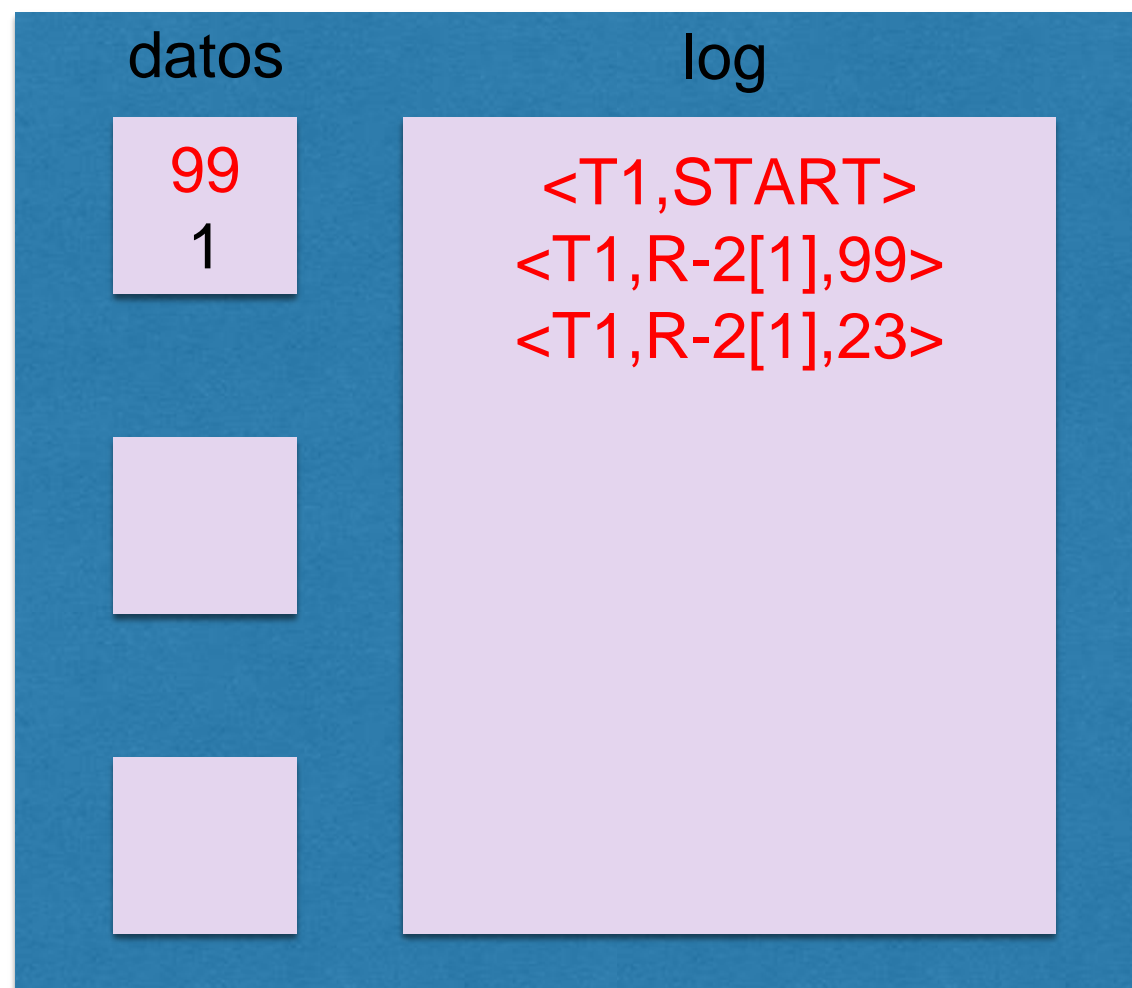


Redo Logging – en la BD

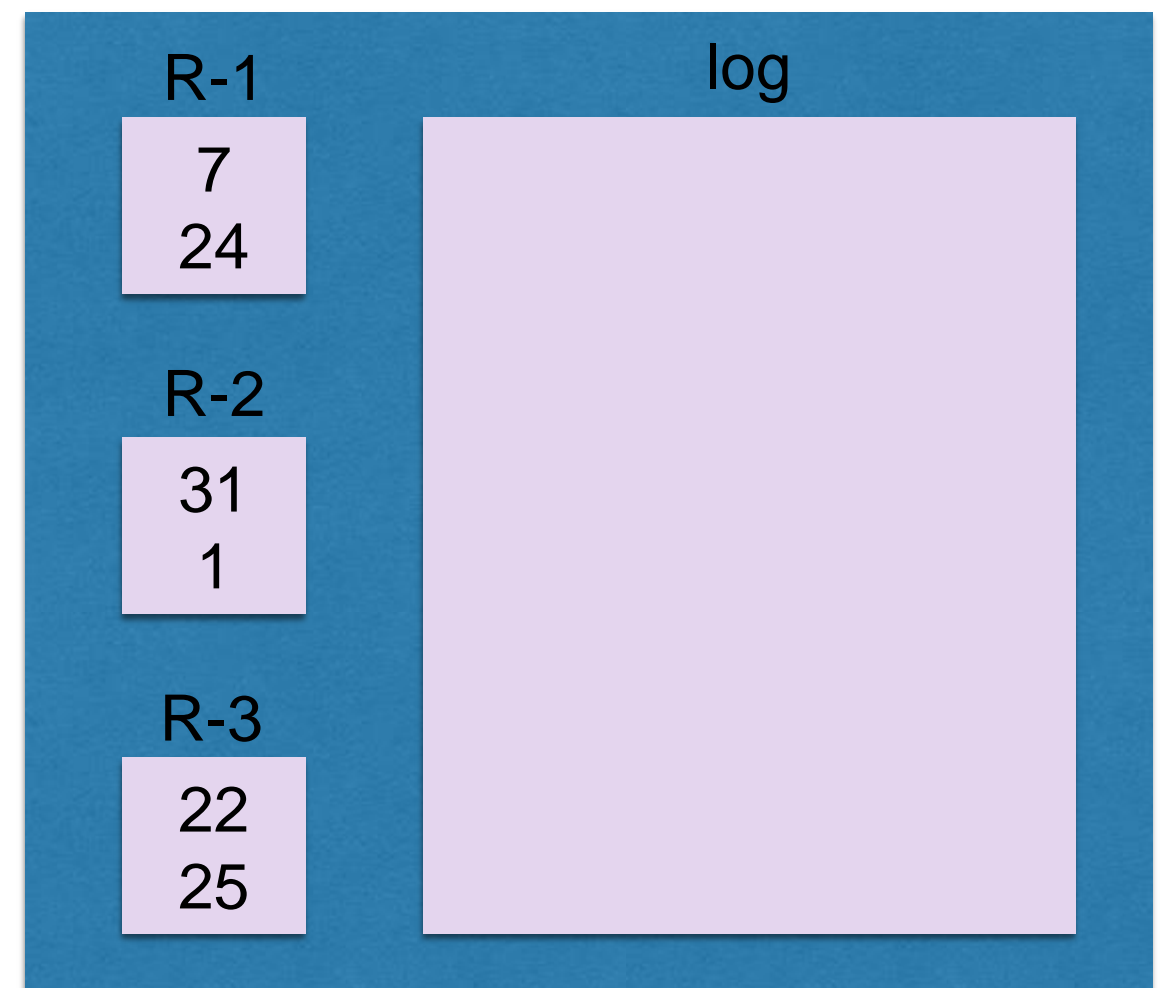
DB

T1: Cambio 99 a 23!

Buffer



Disco

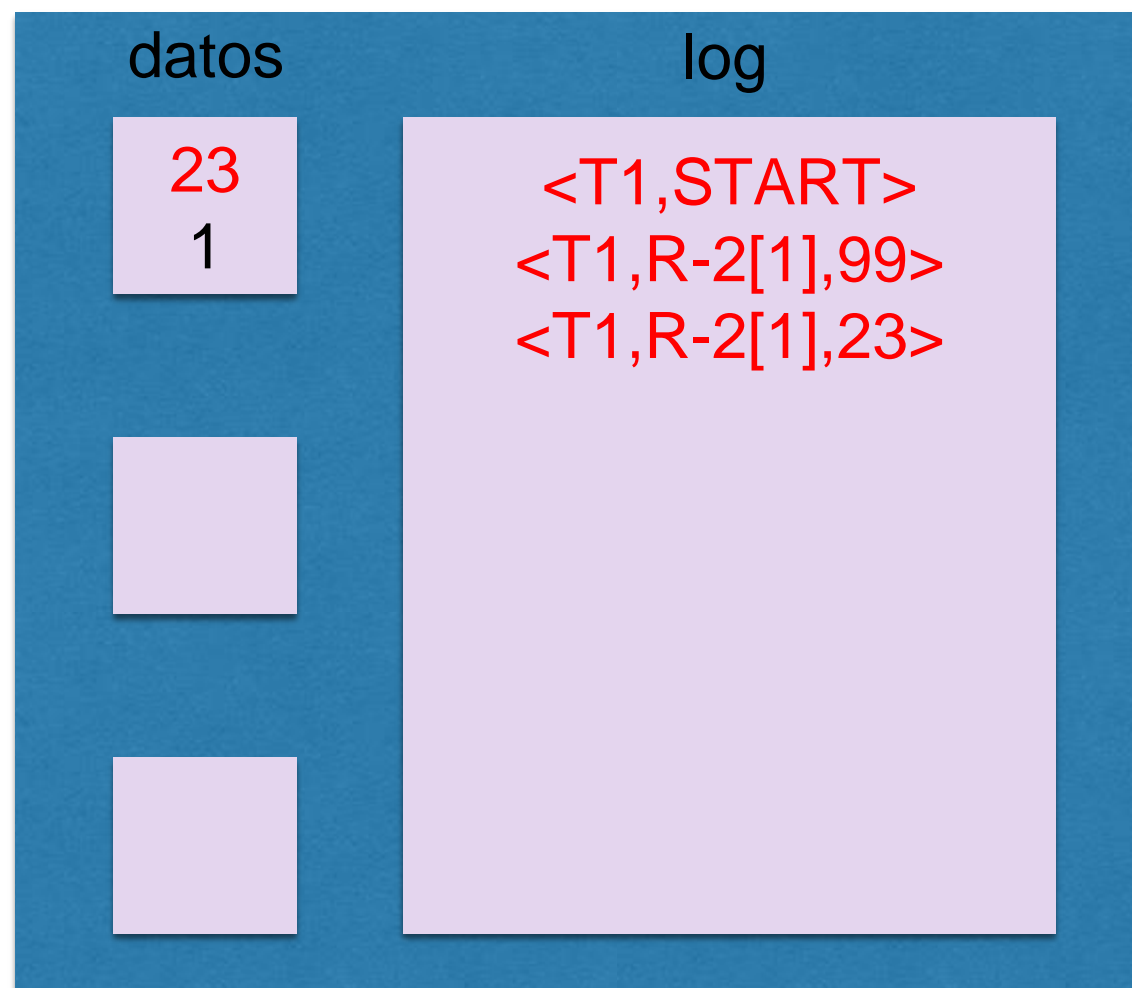


Redo Logging – en la BD

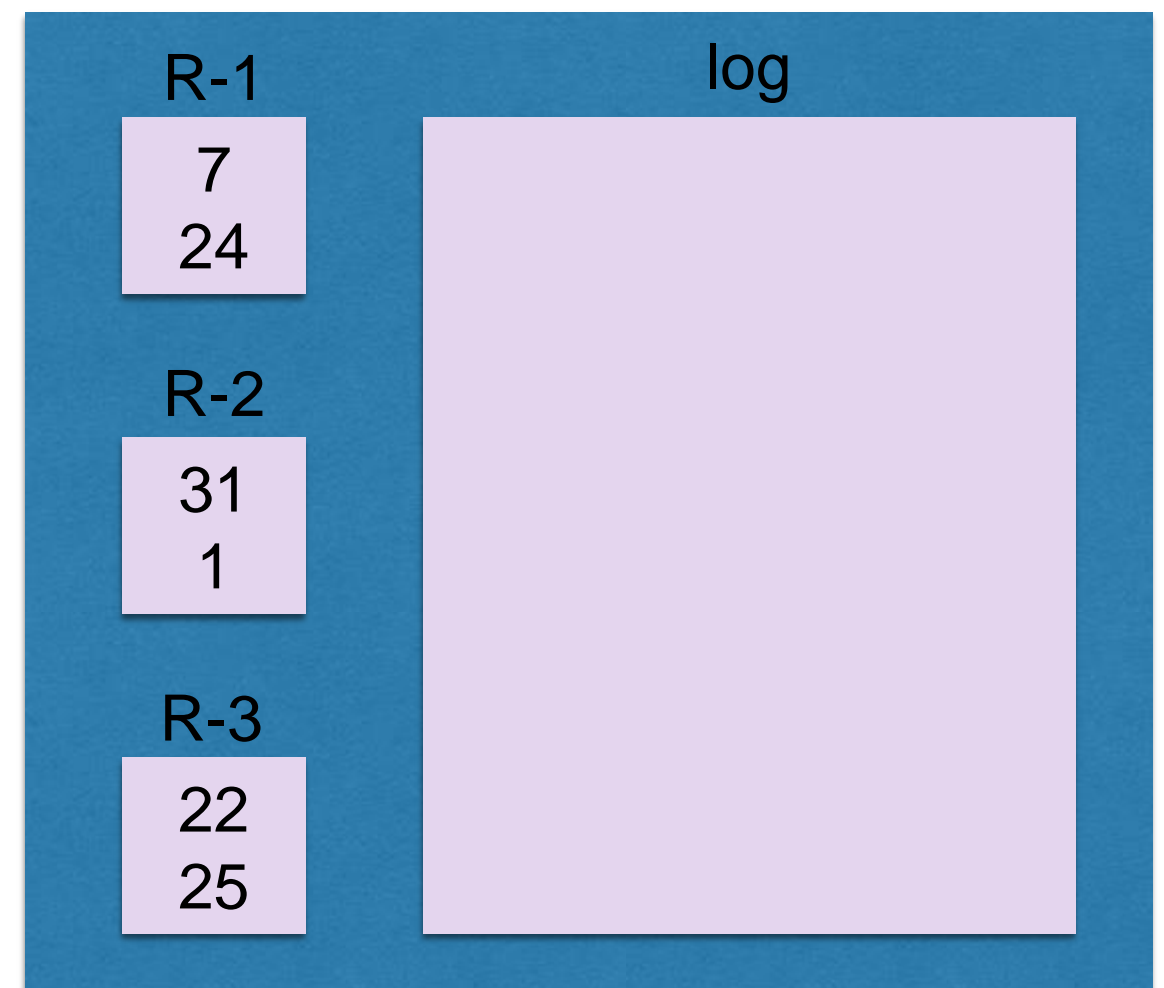
DB

T1: Cambio 99 a 23!

Buffer



Disco

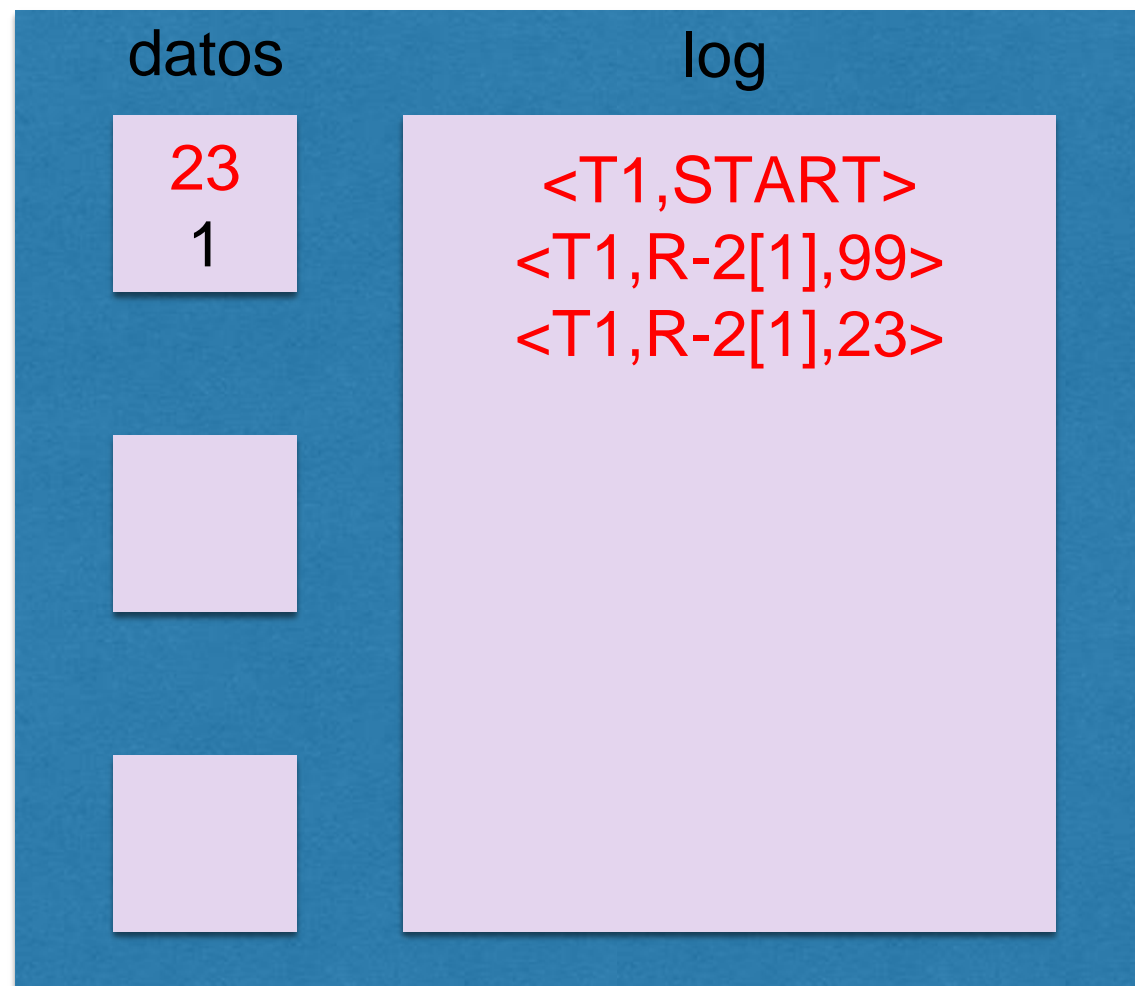


Redo Logging – en la BD

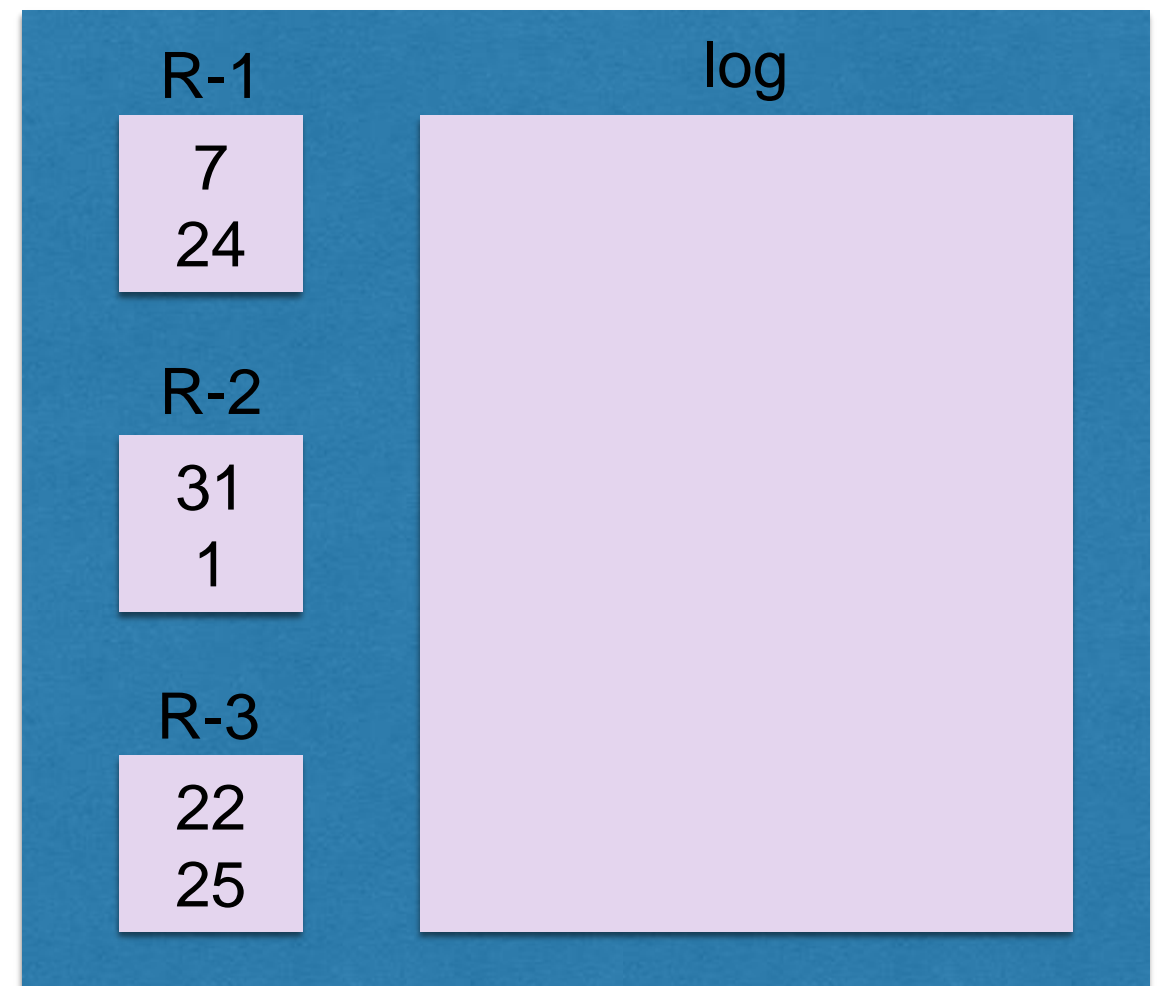
DB

T1: estoy listo!

Buffer



Disco

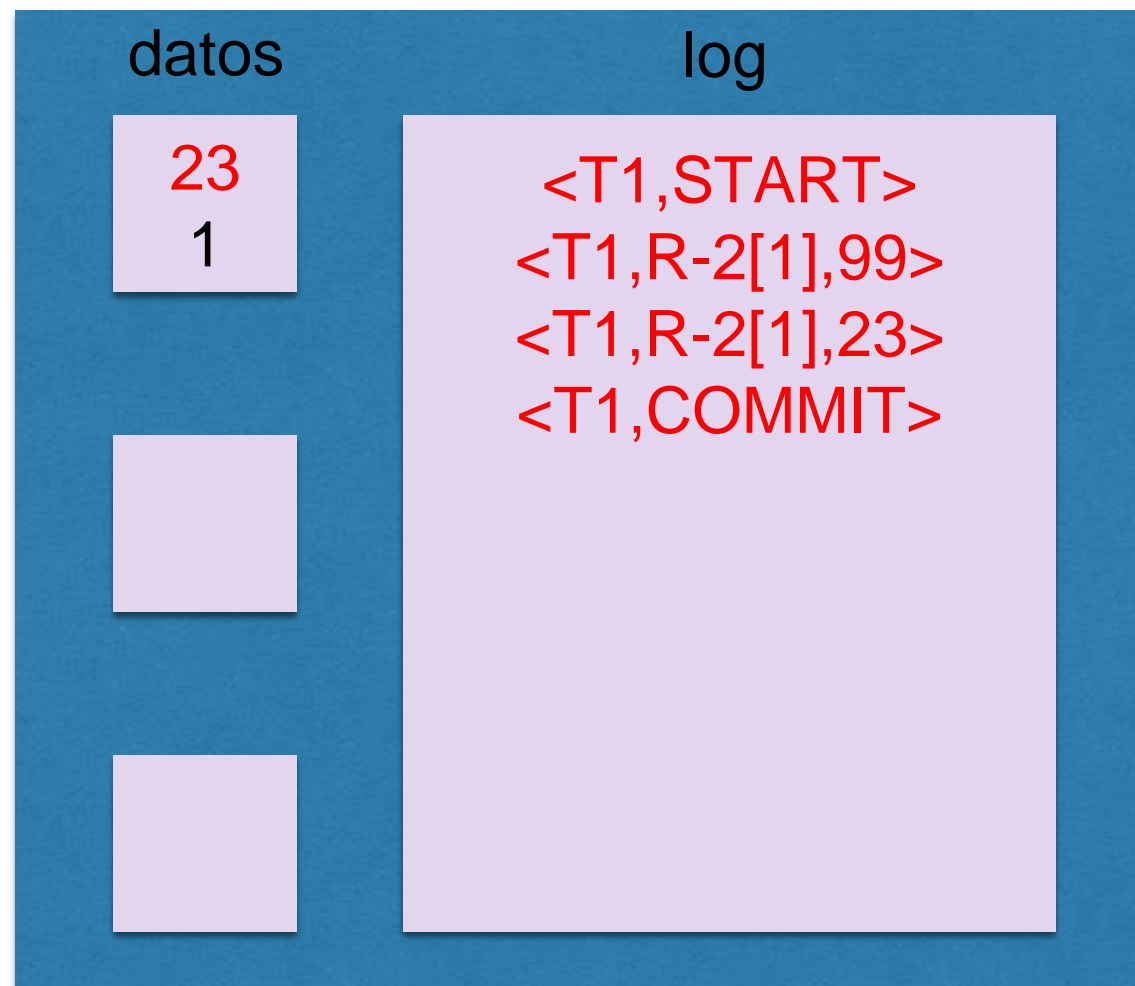


Redo Logging – en la BD

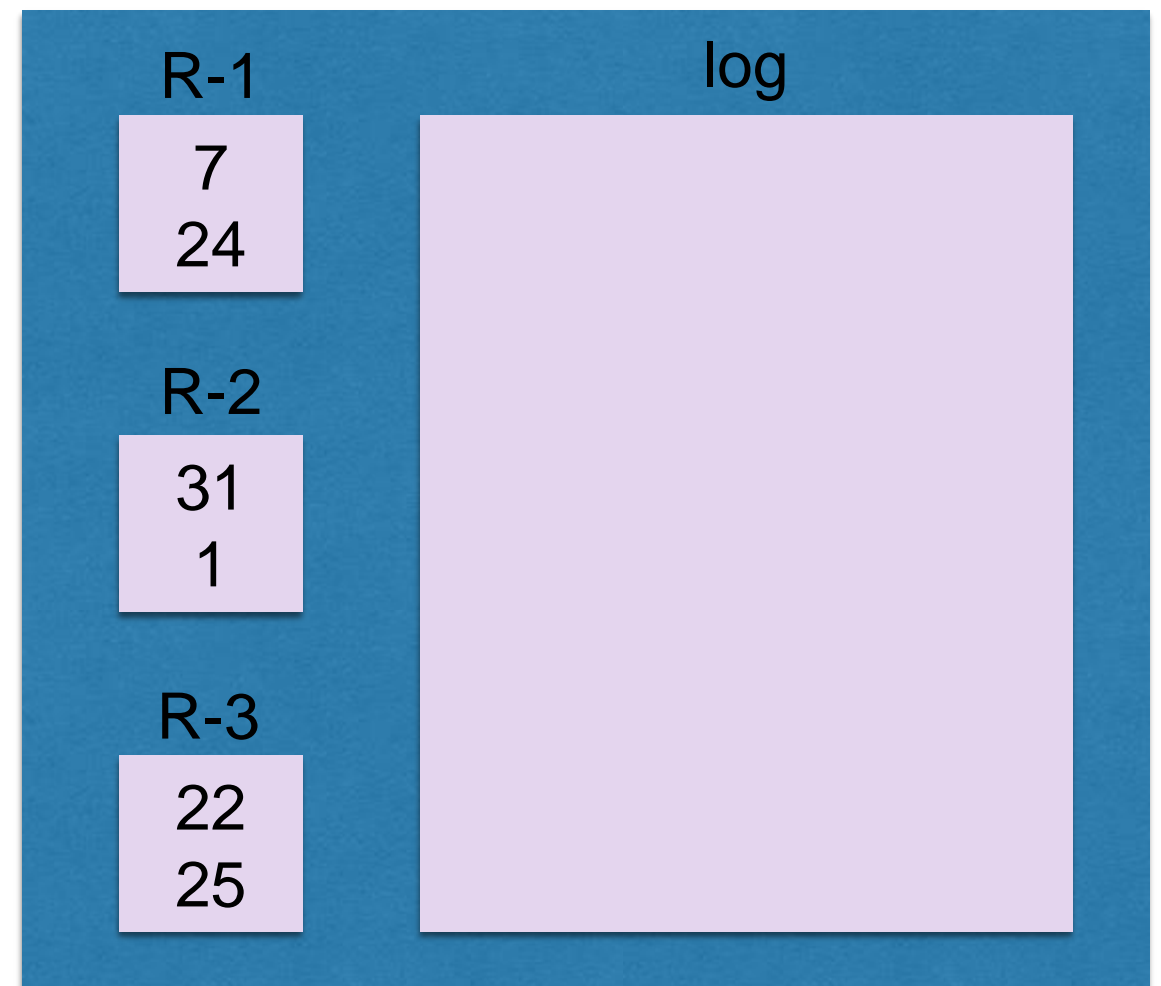
DB

T1: estoy listo!

Buffer



Disco

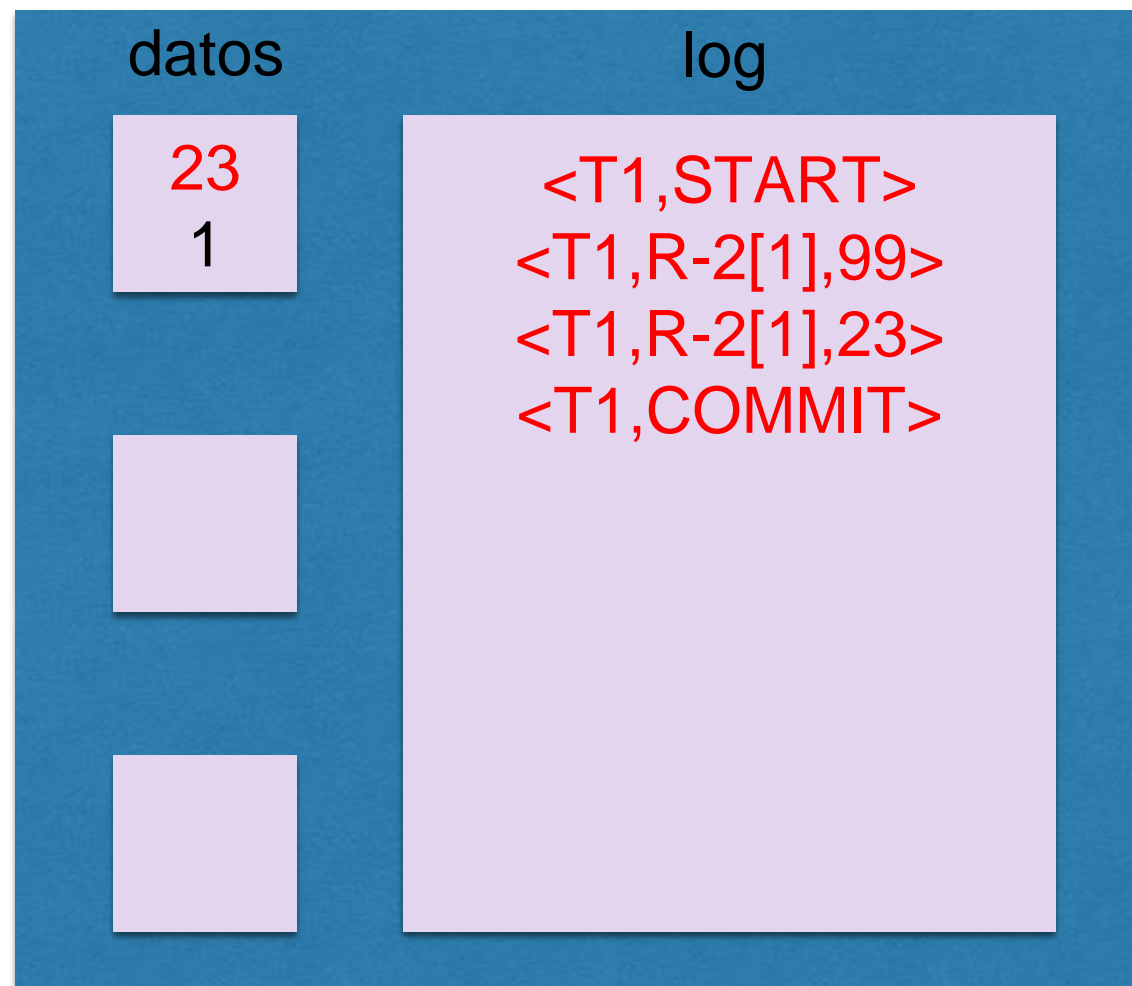


Redo Logging – en la BD

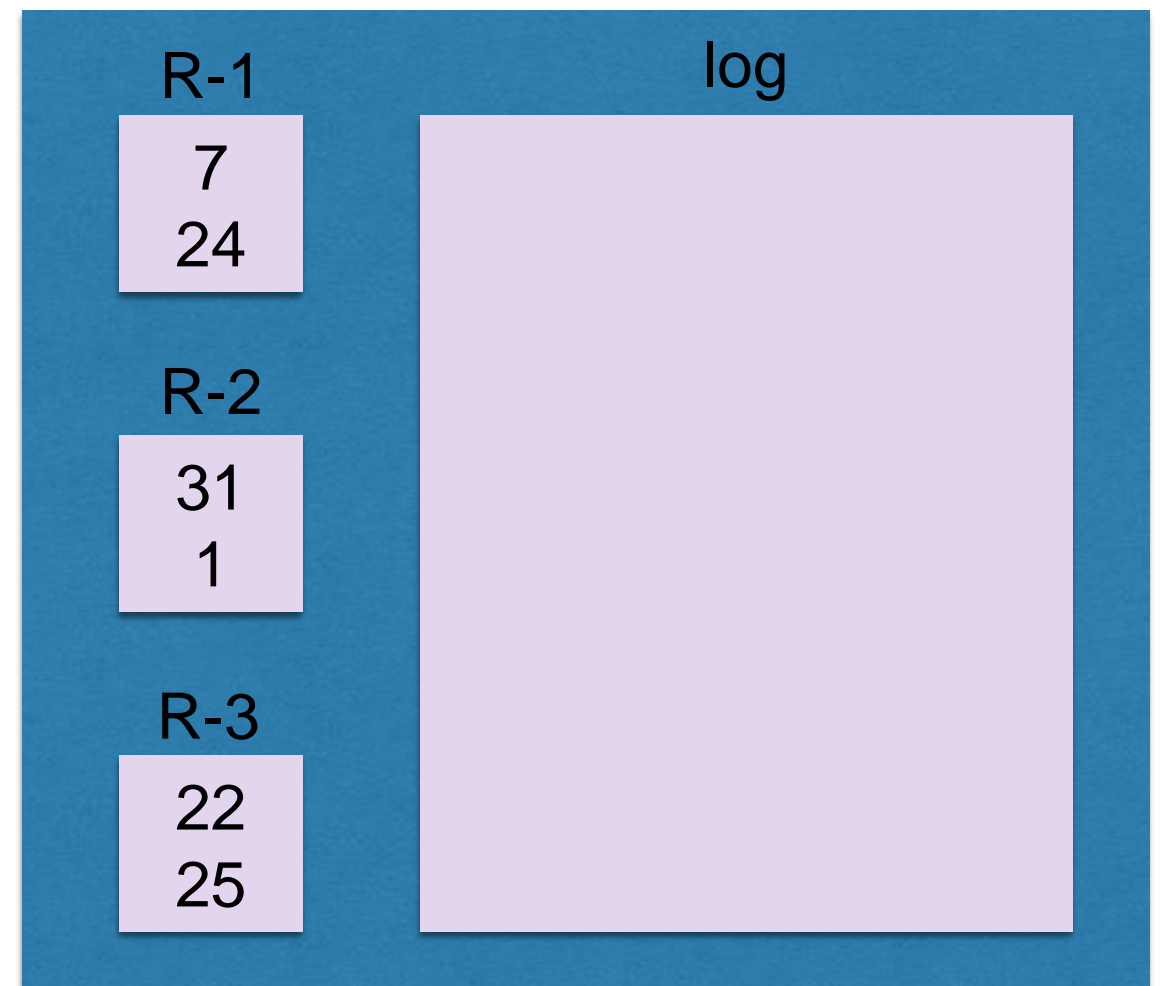
DB

Vamos al disco!

Buffer



Disco

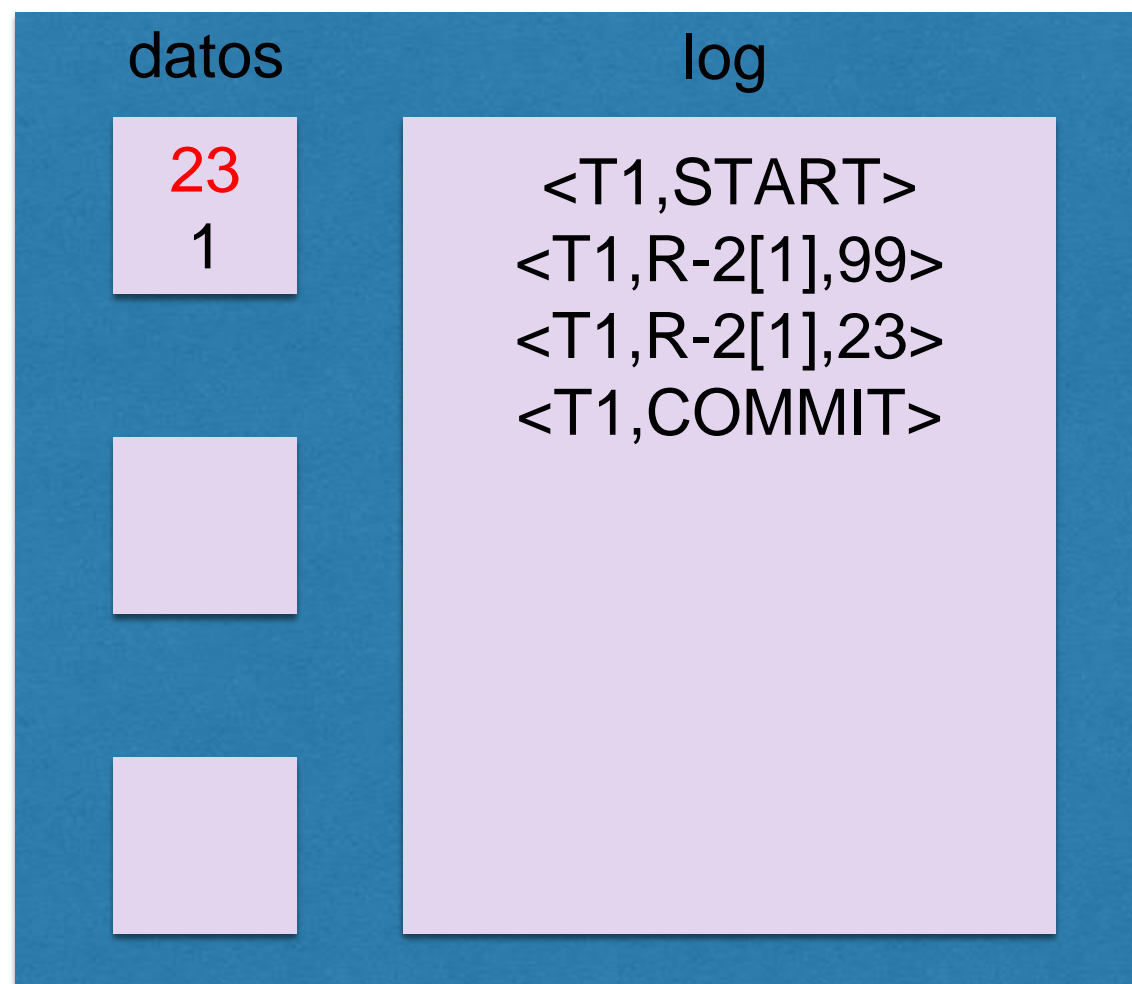


Redo Logging – en la BD

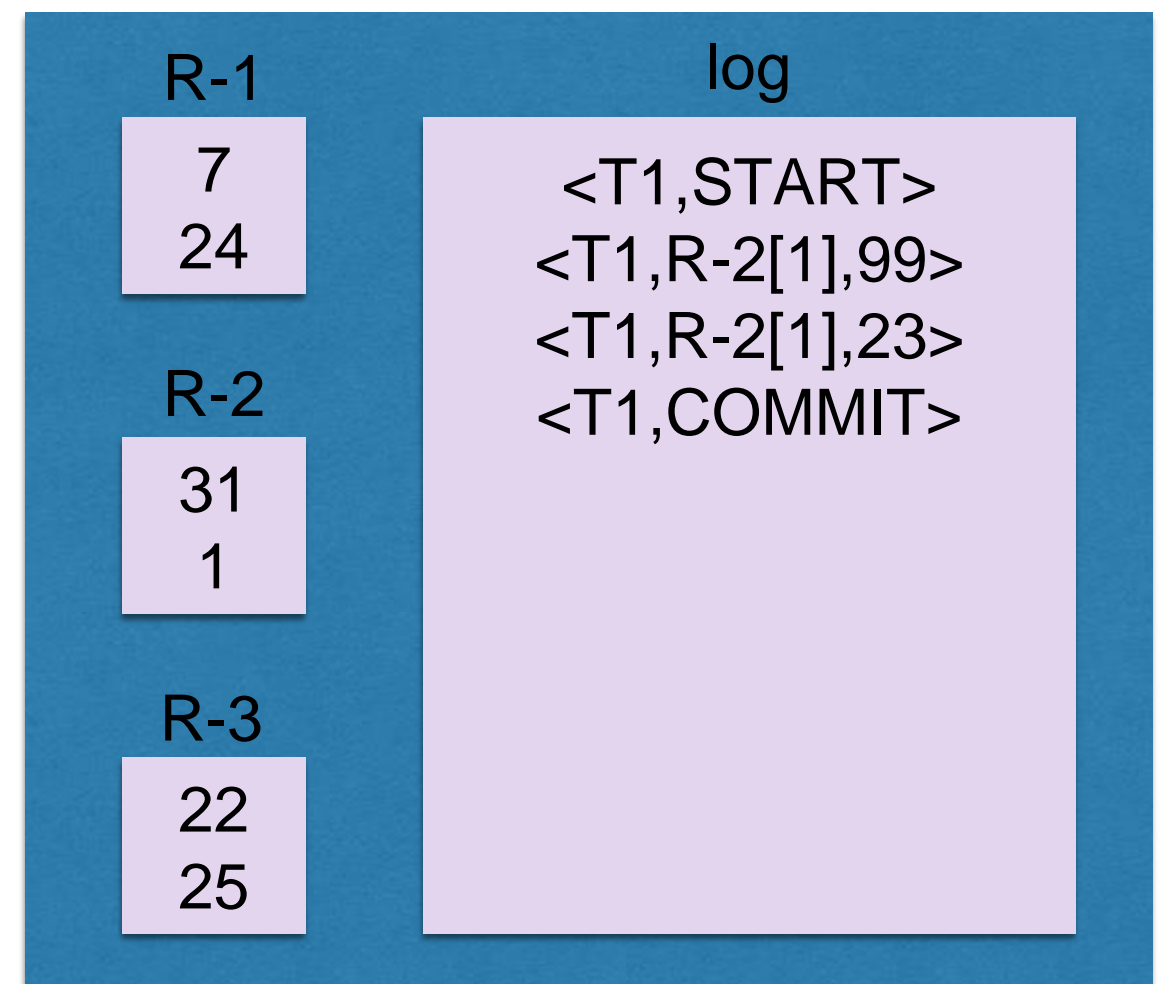
DB

Vamos al disco!

Buffer



Disco

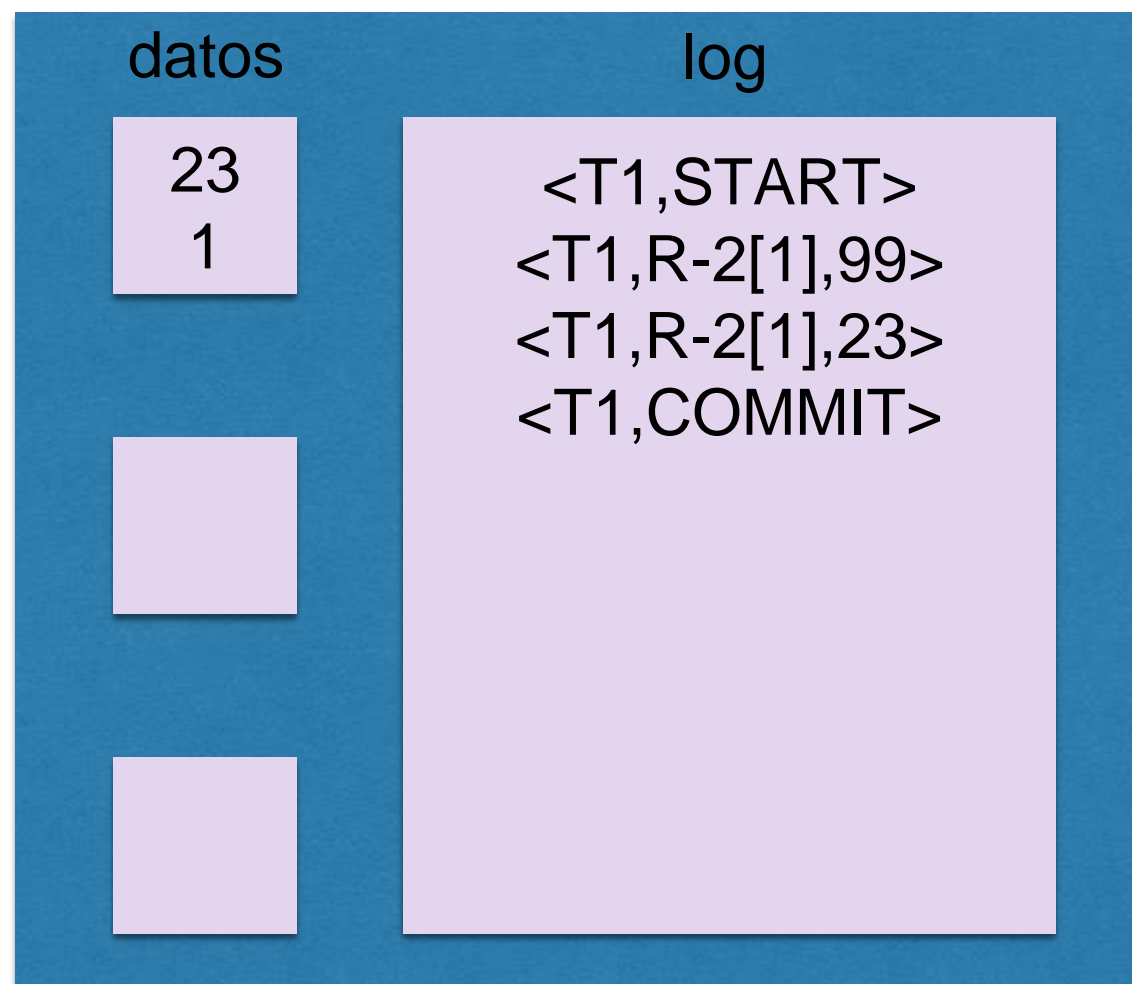


Redo Logging – en la BD

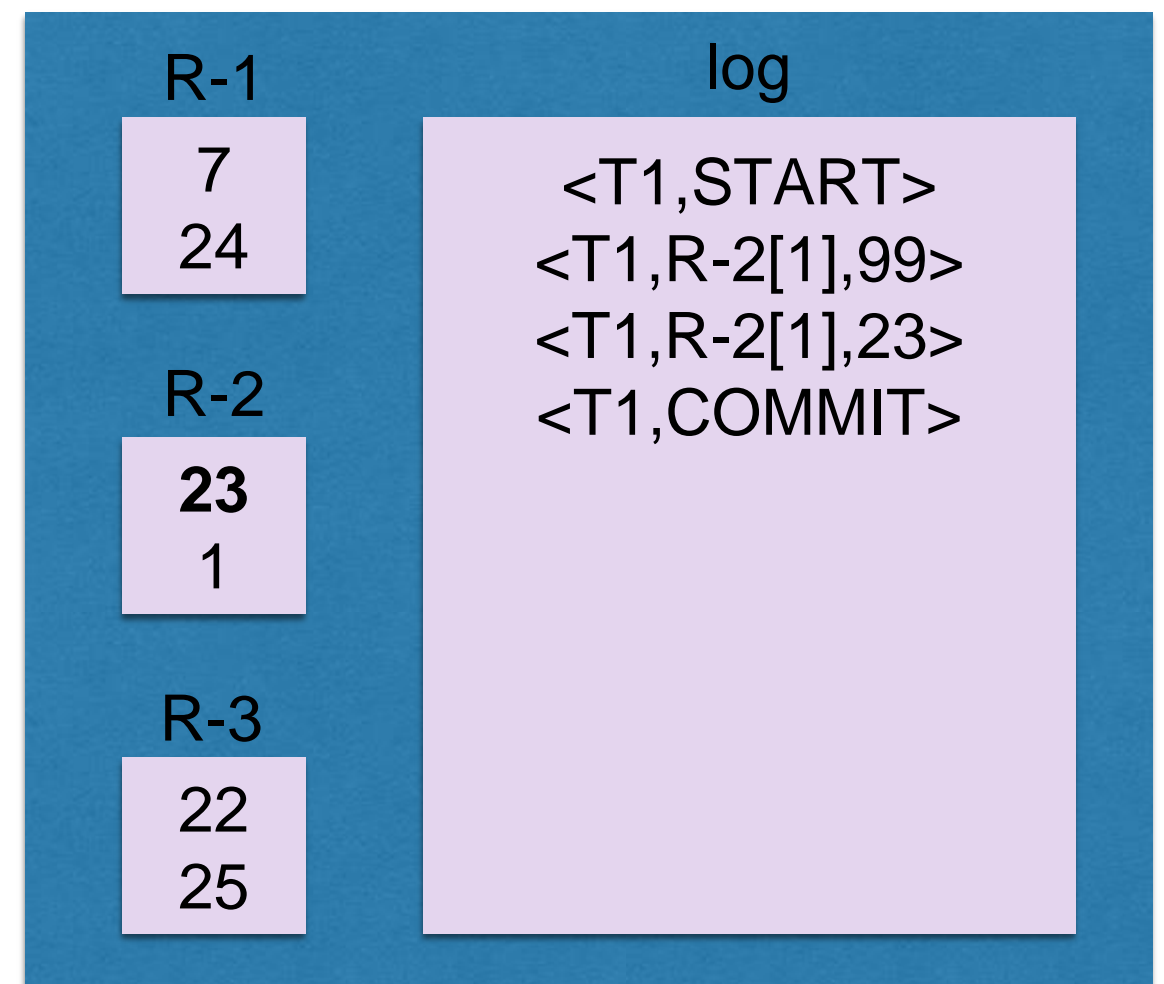
DB

Actualizo los datos

Buffer



Disco

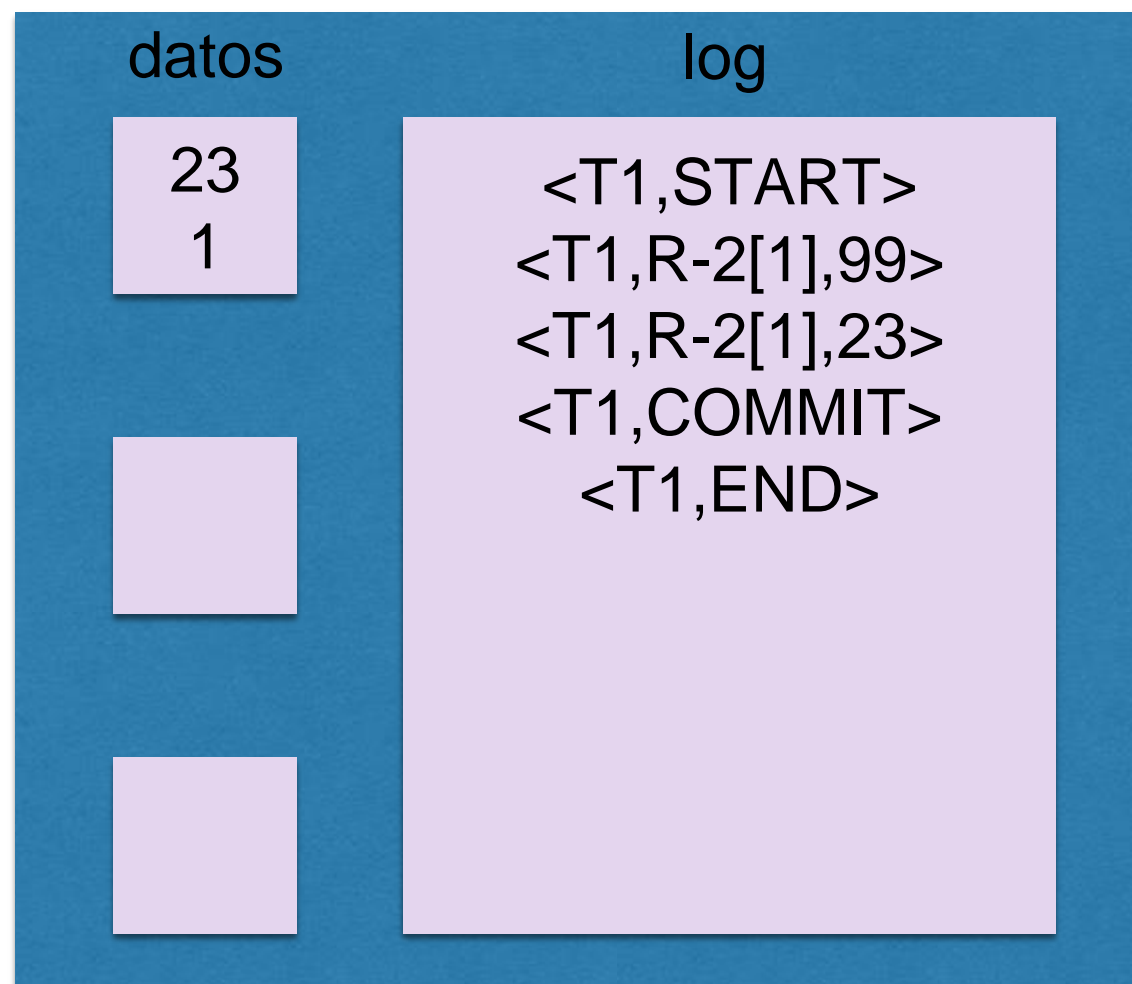


Redo Logging – en la BD

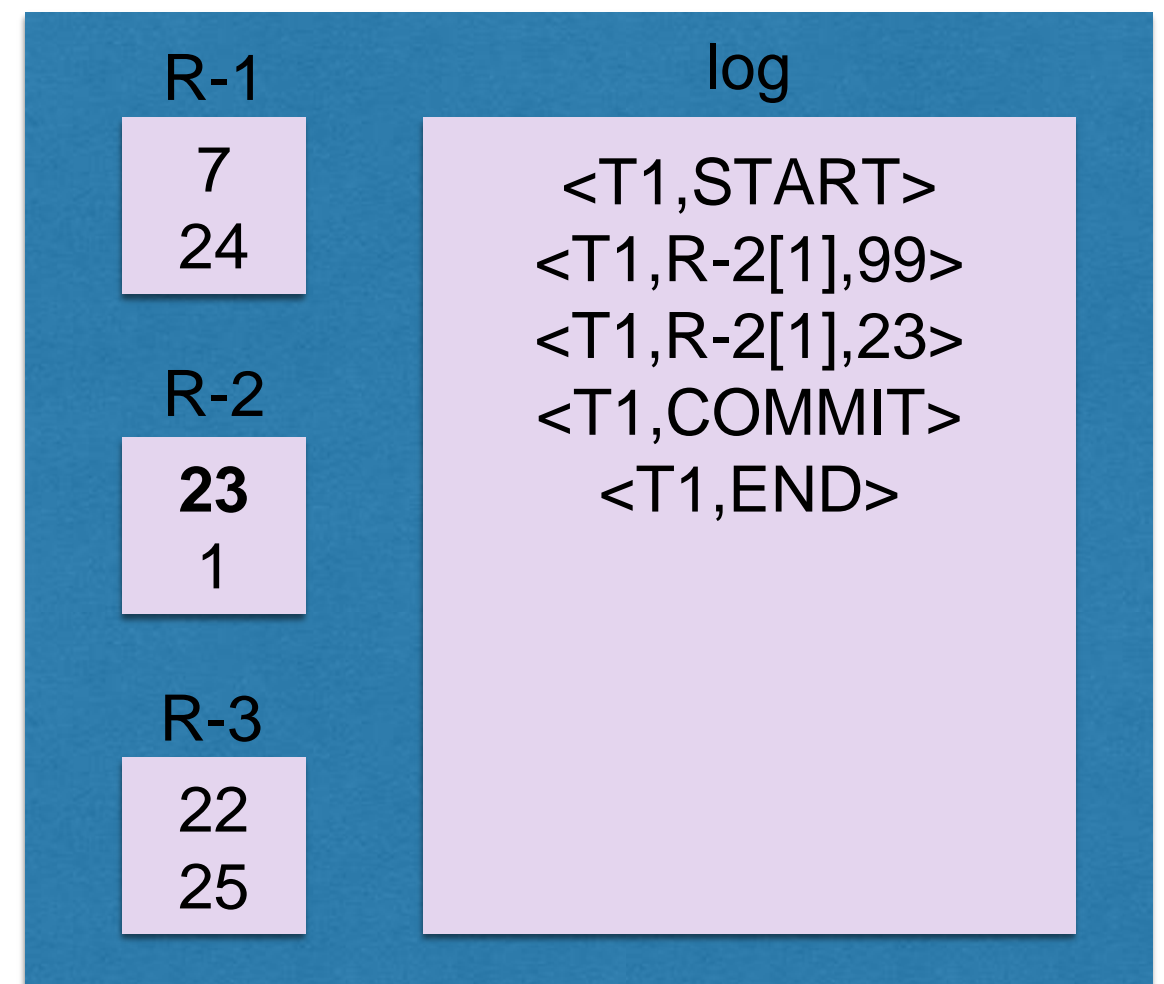
DB

100% OK

Buffer



Disco



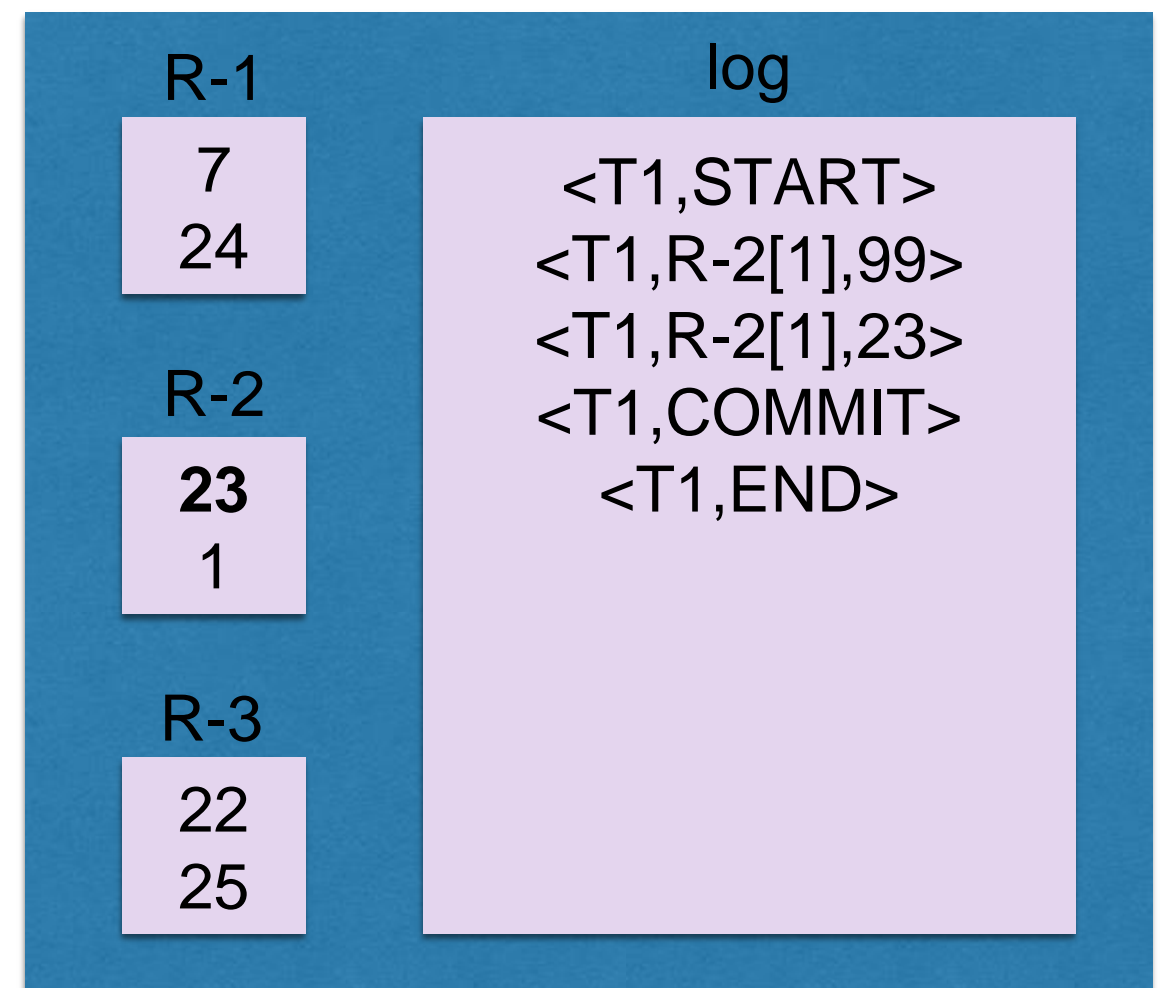
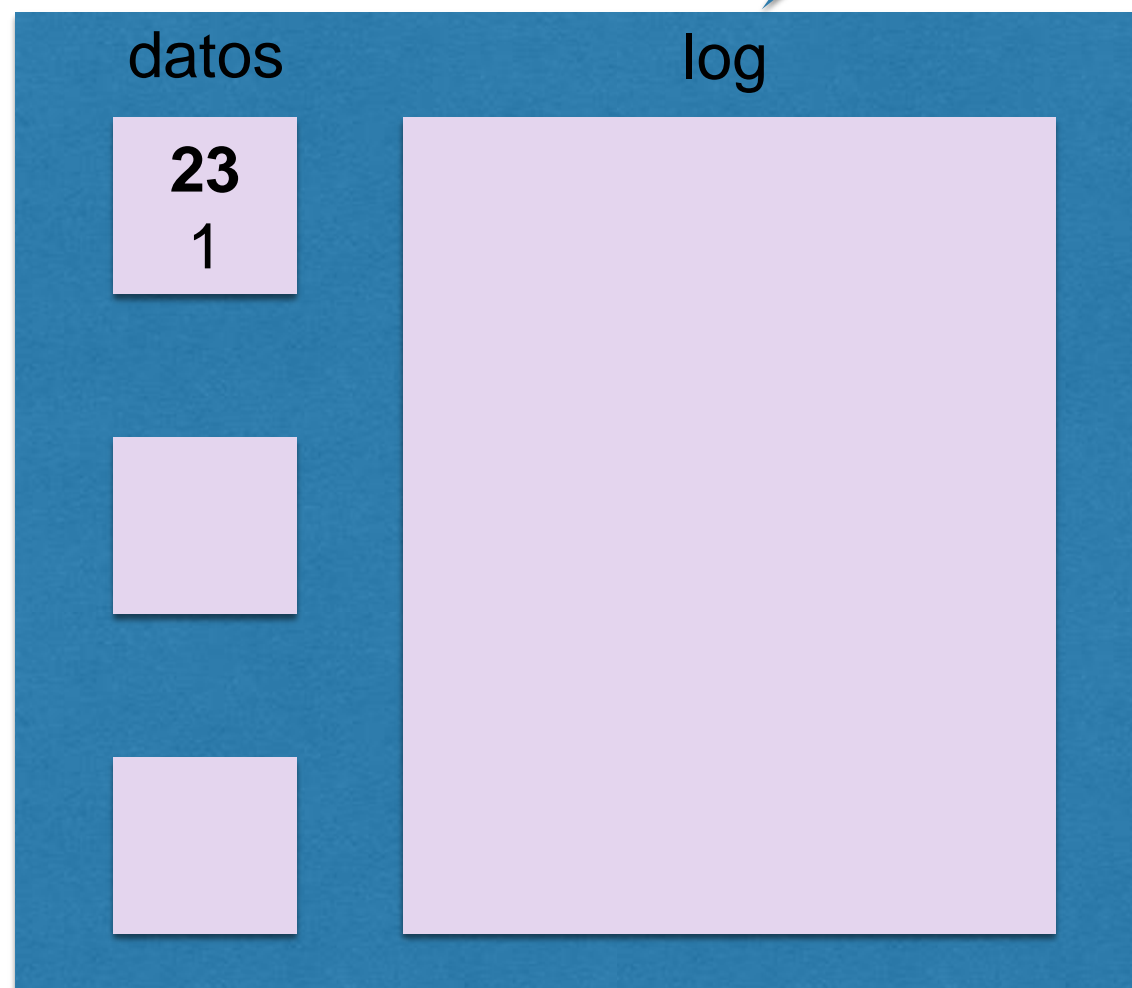
Redo Logging – en la BD

DB

Libera el buffer

Buffer

Disco

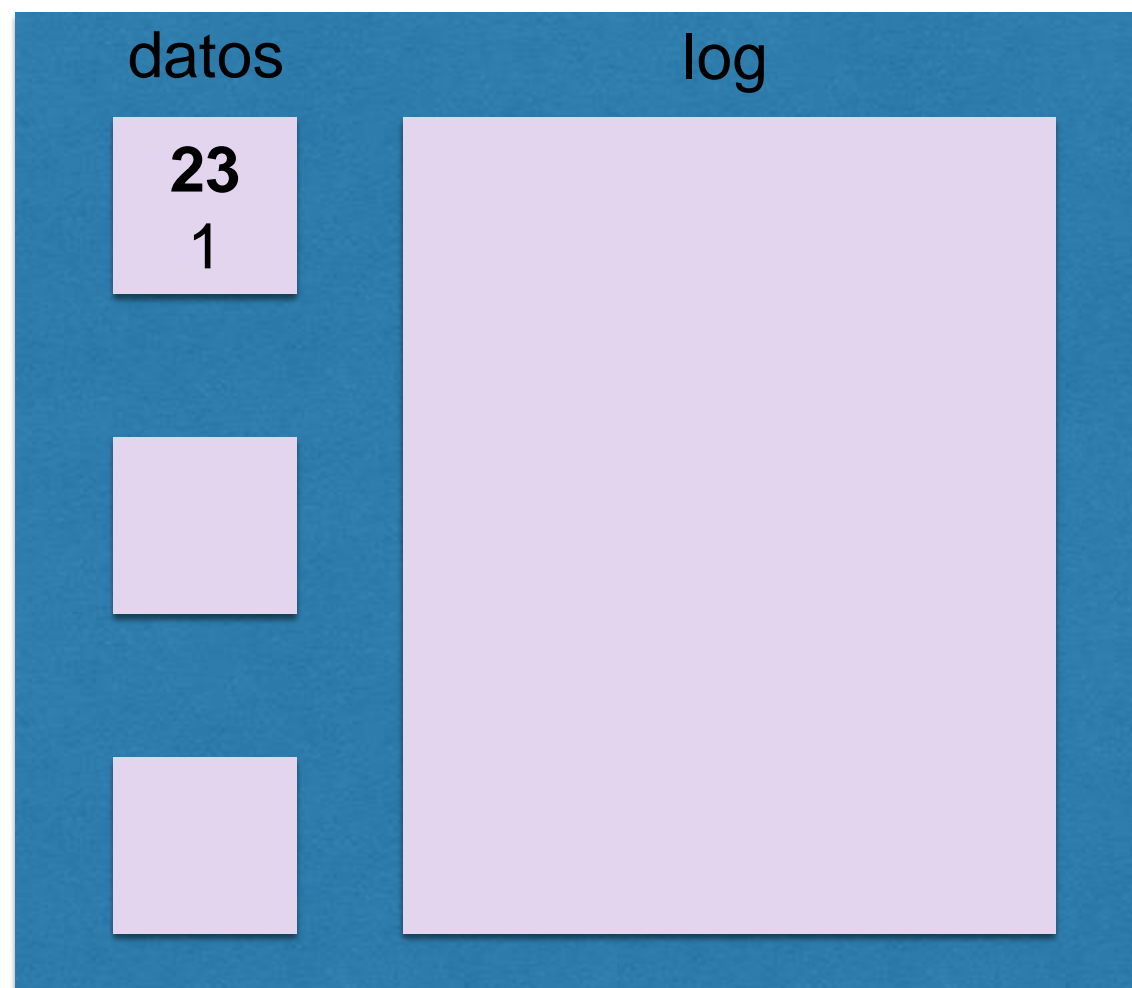


Redo Logging – en la BD

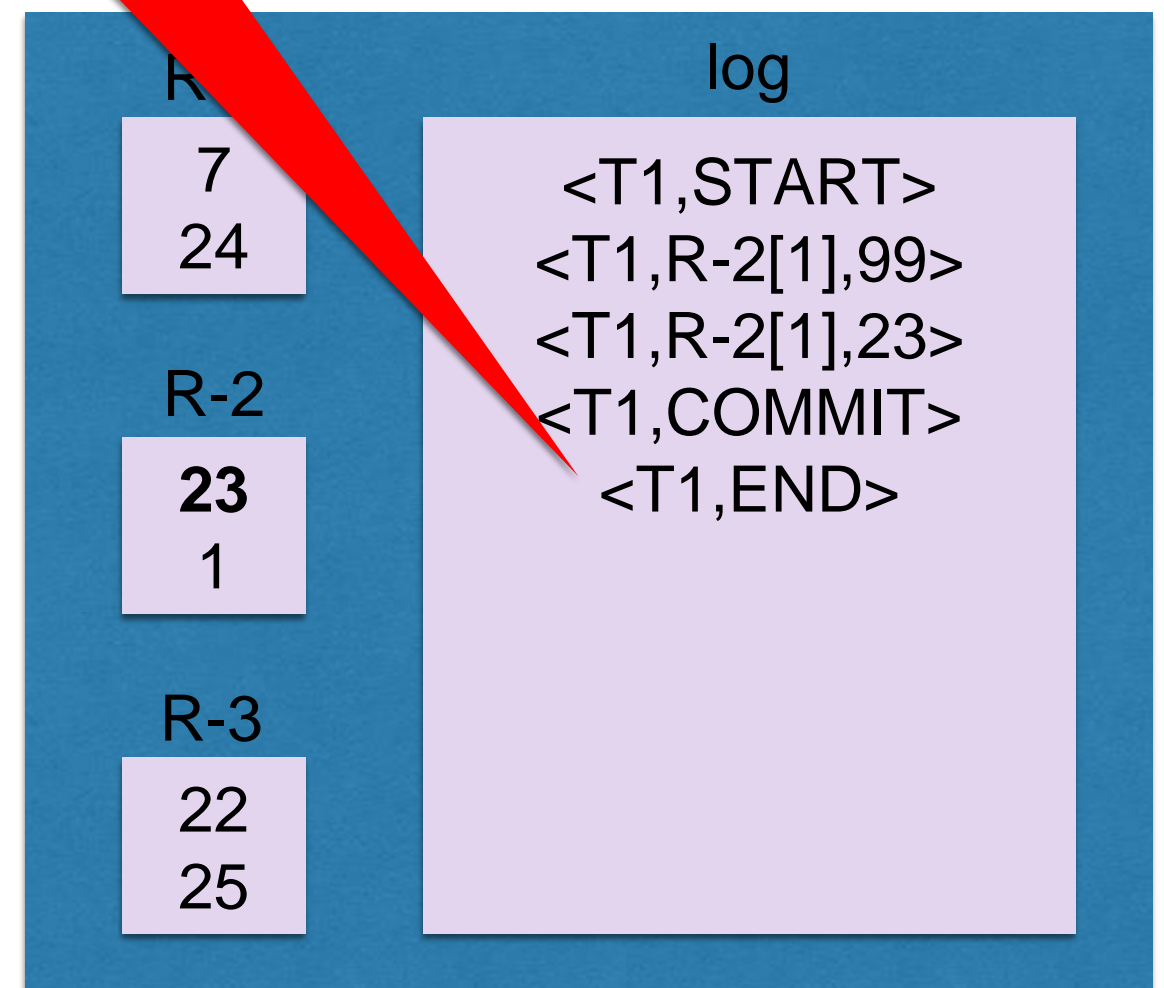
DB

END =
datos están en disco

Buffer



Disco



Recuperación con Redo Logging

Detectando fallas en el *log*:

... <START T> ... <END T> ...



Recuperación con Redo Logging

Detectando fallas en el *log*:

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

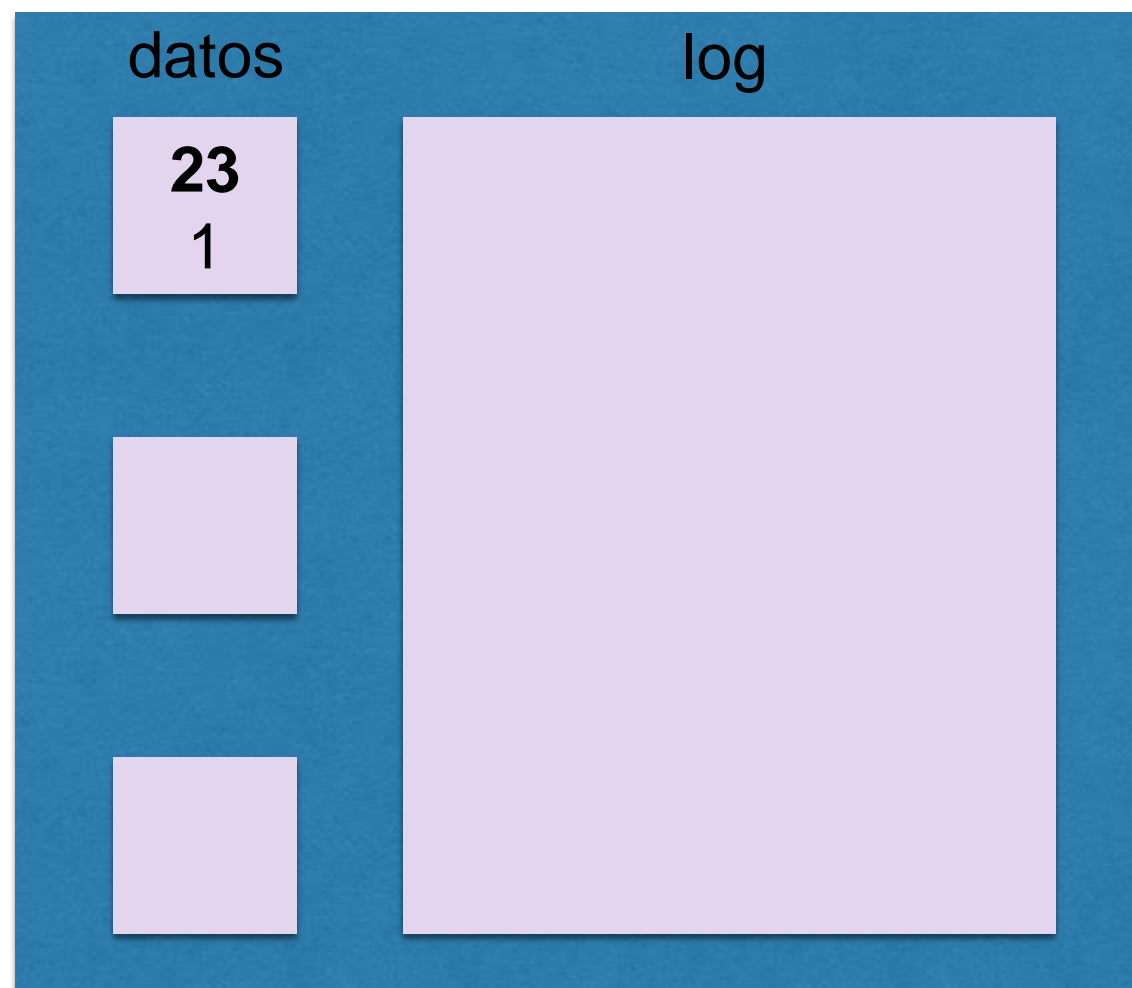


Redo Logging – en la BD

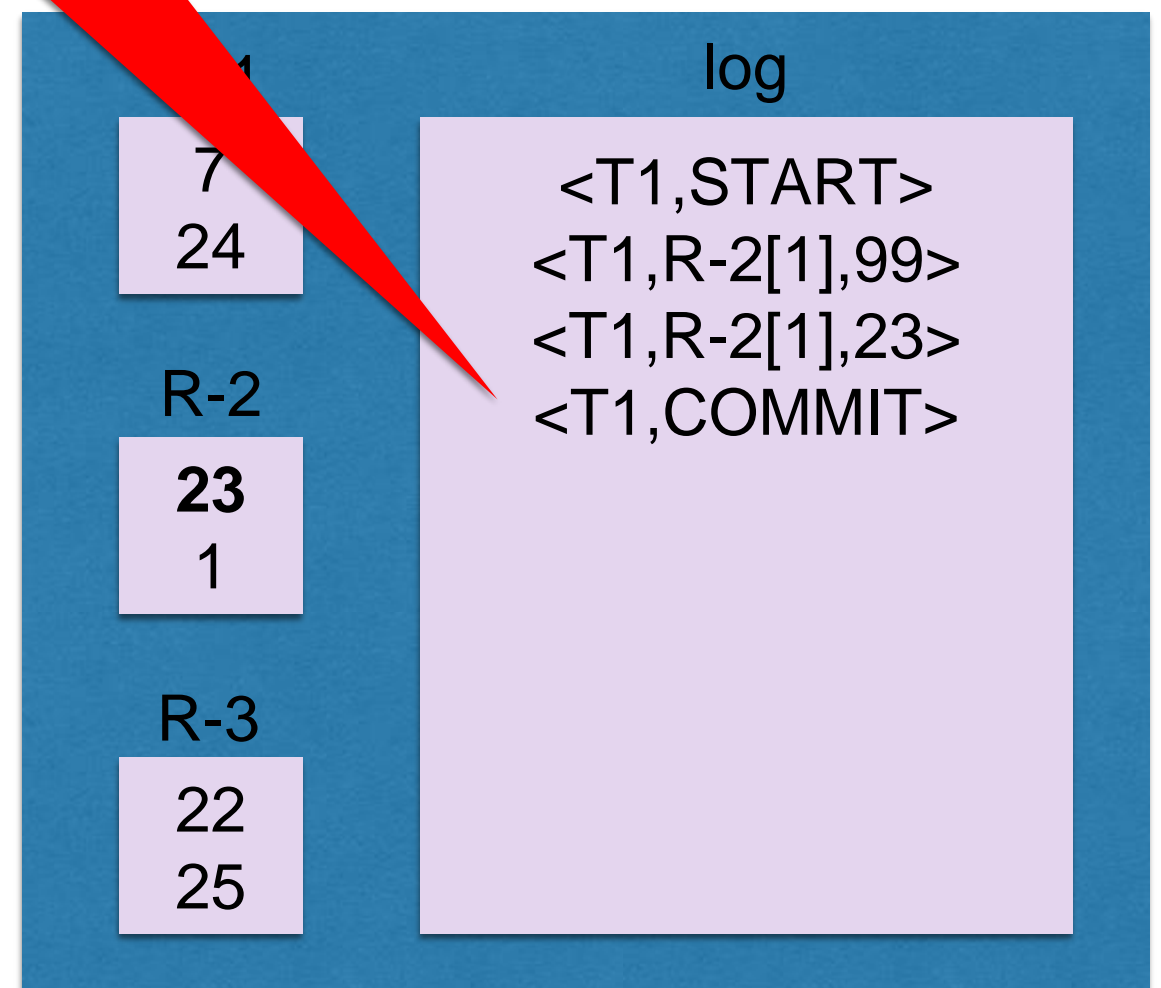
DB

No hay END
(No sé si escribí los datos)

Buffer



Disco



Recuperación con Redo Logging

Detectando fallas en el *log*:

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

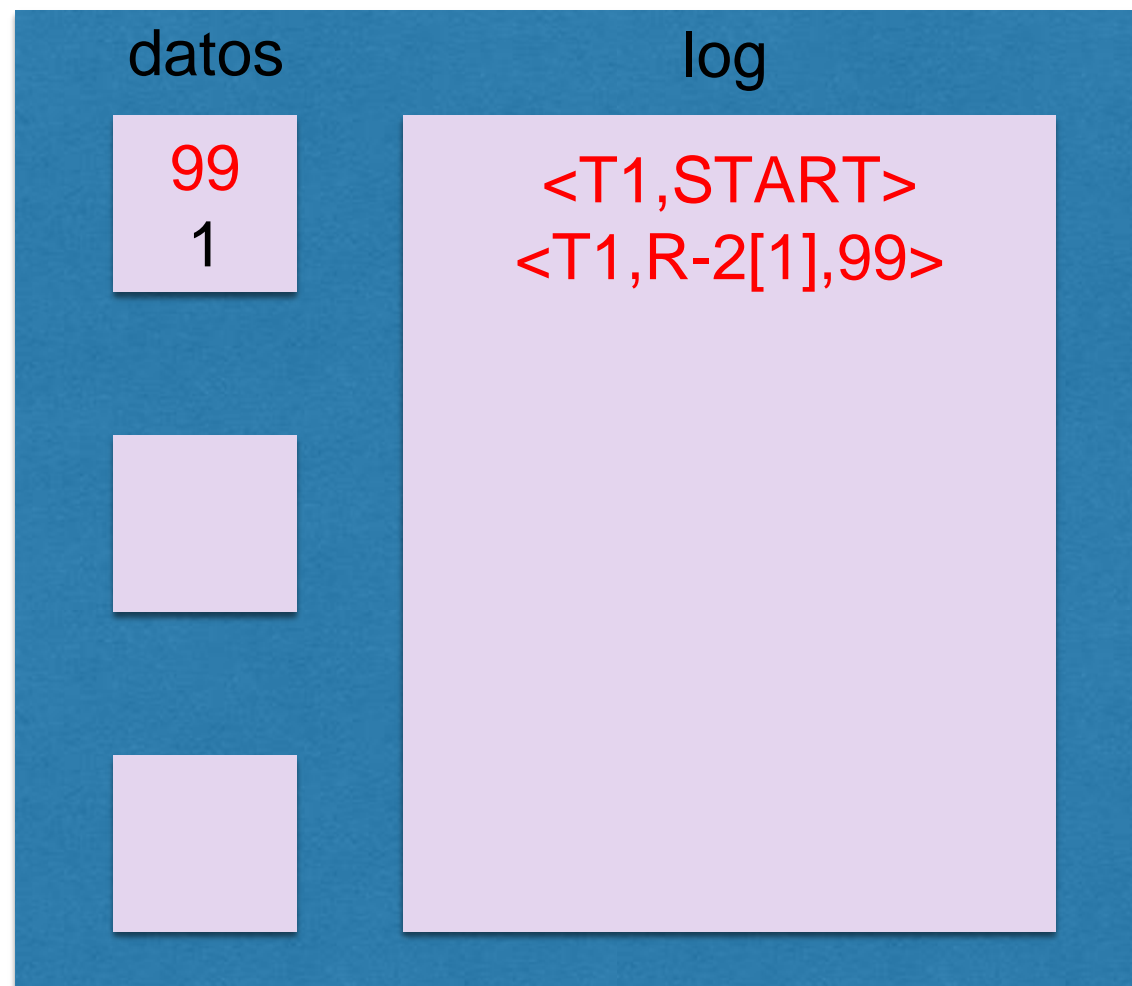


Redo Logging – en la BD

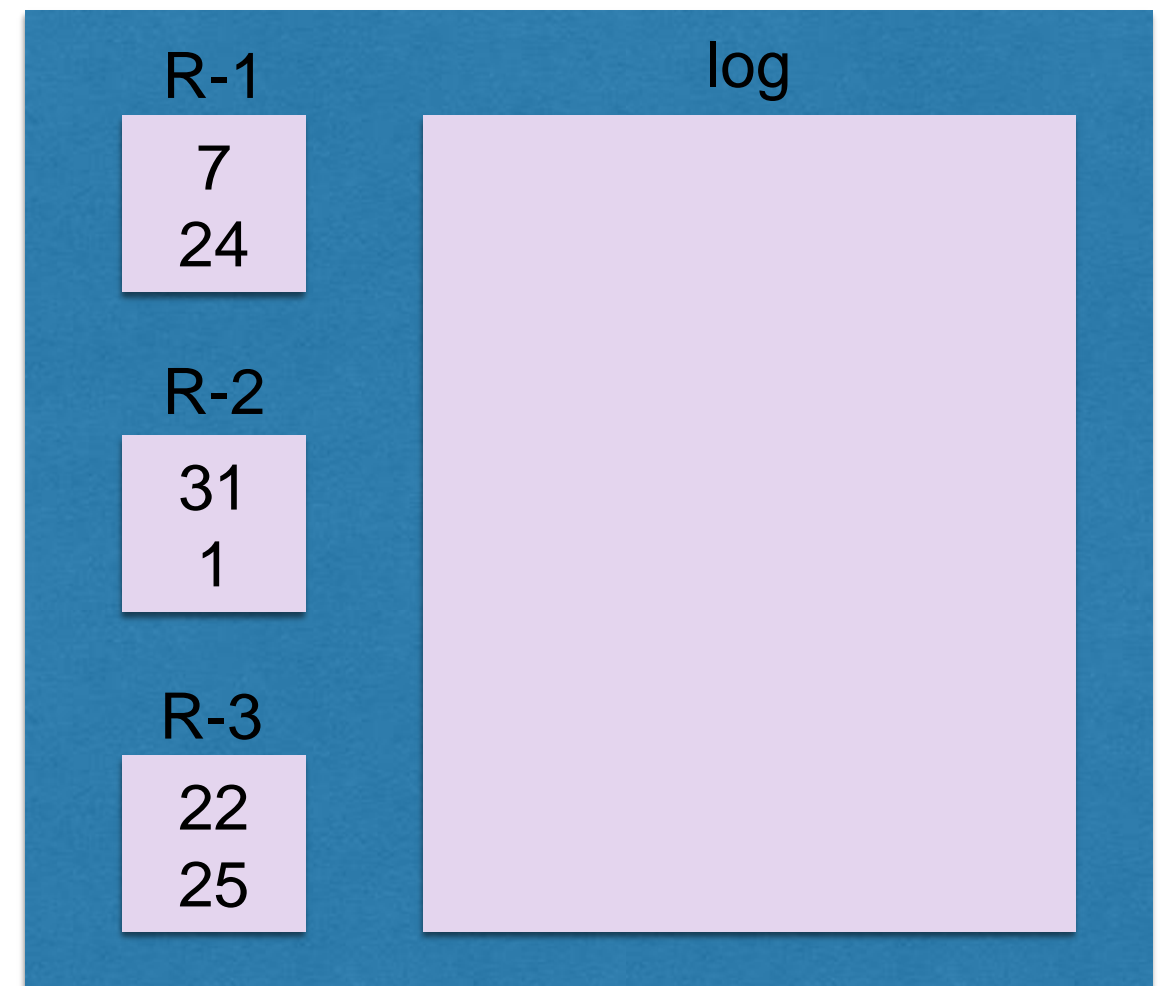
DB

T1: Cambio 31 a 99!

Buffer



Disco

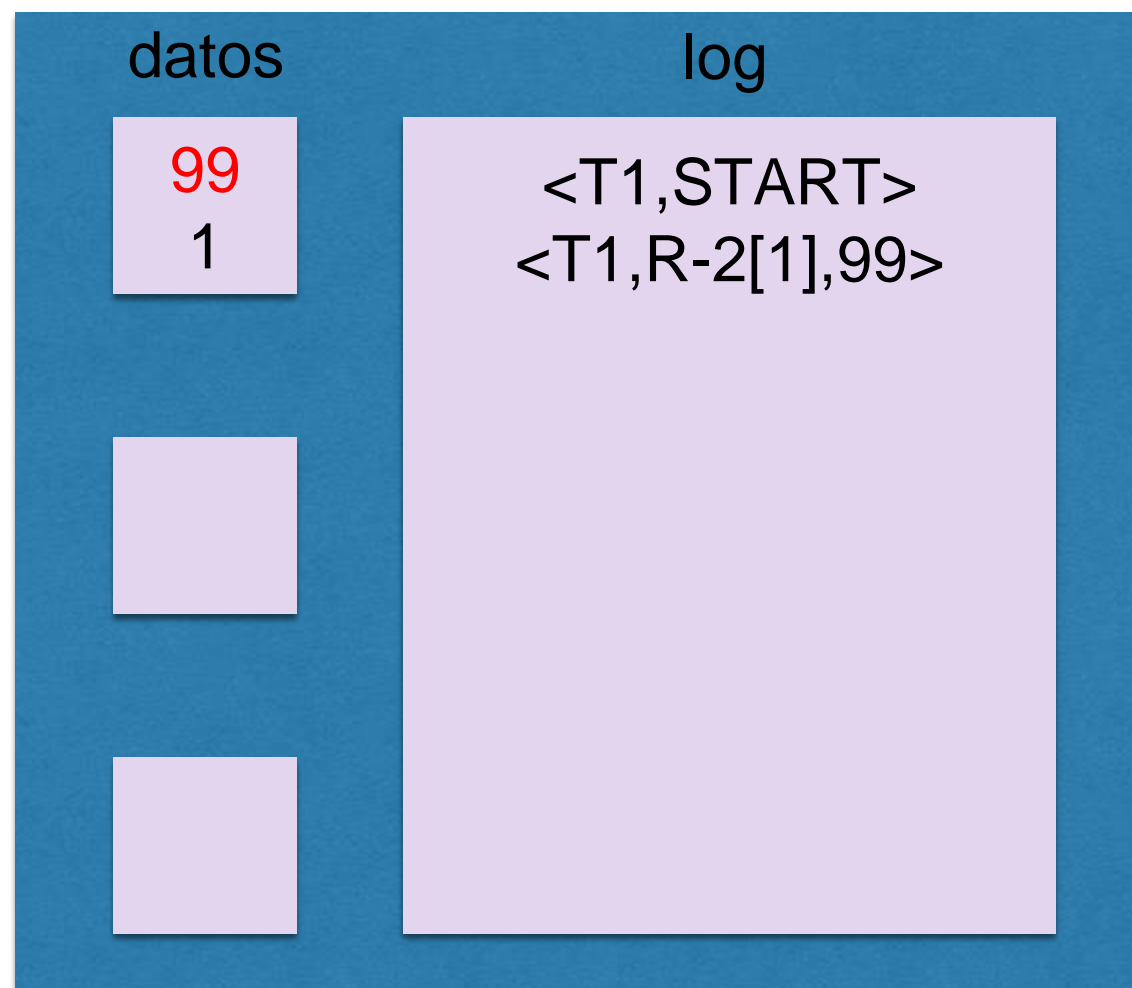


Redo Logging – en la BD

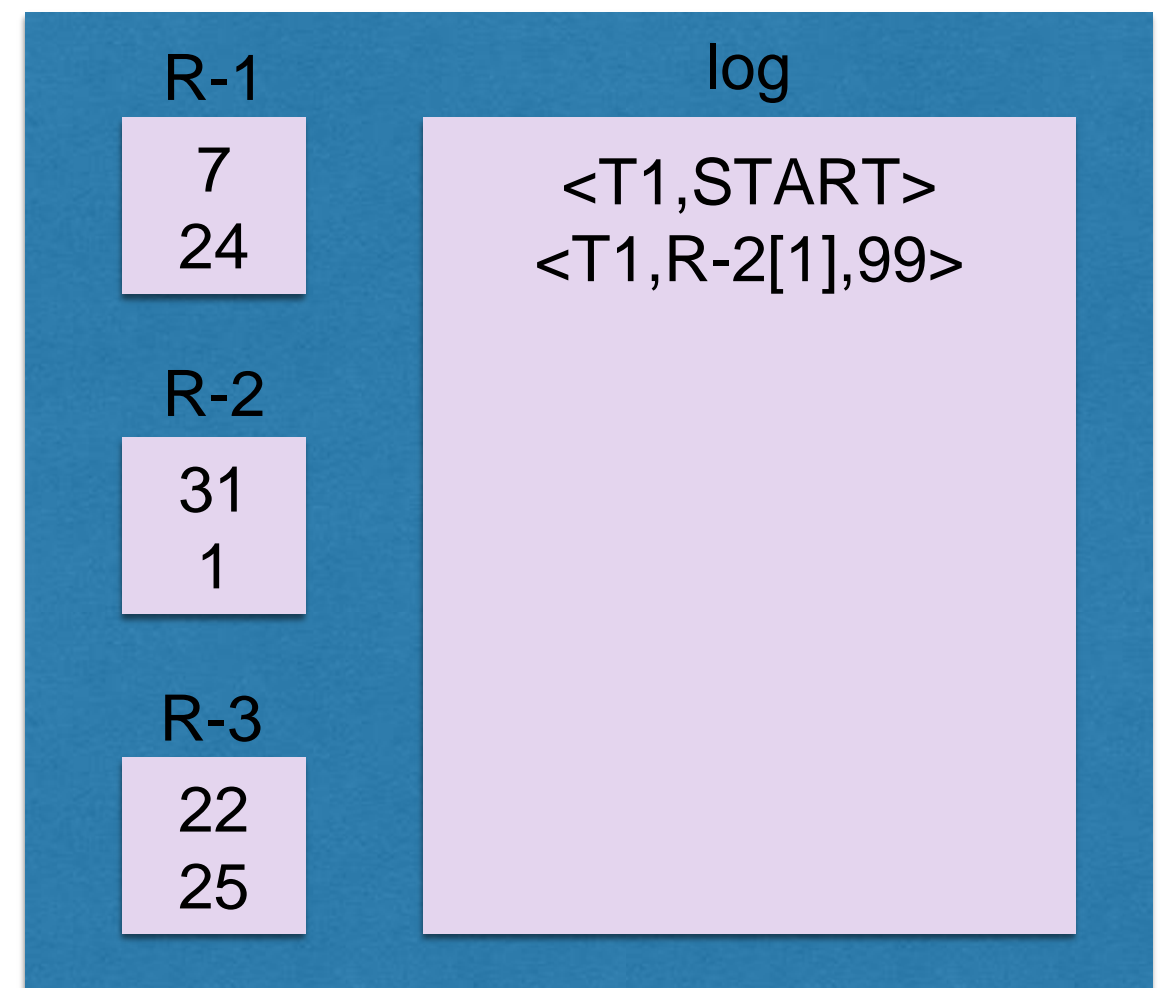
DB

T1: Cambio 31 a 99!

Buffer



Disco

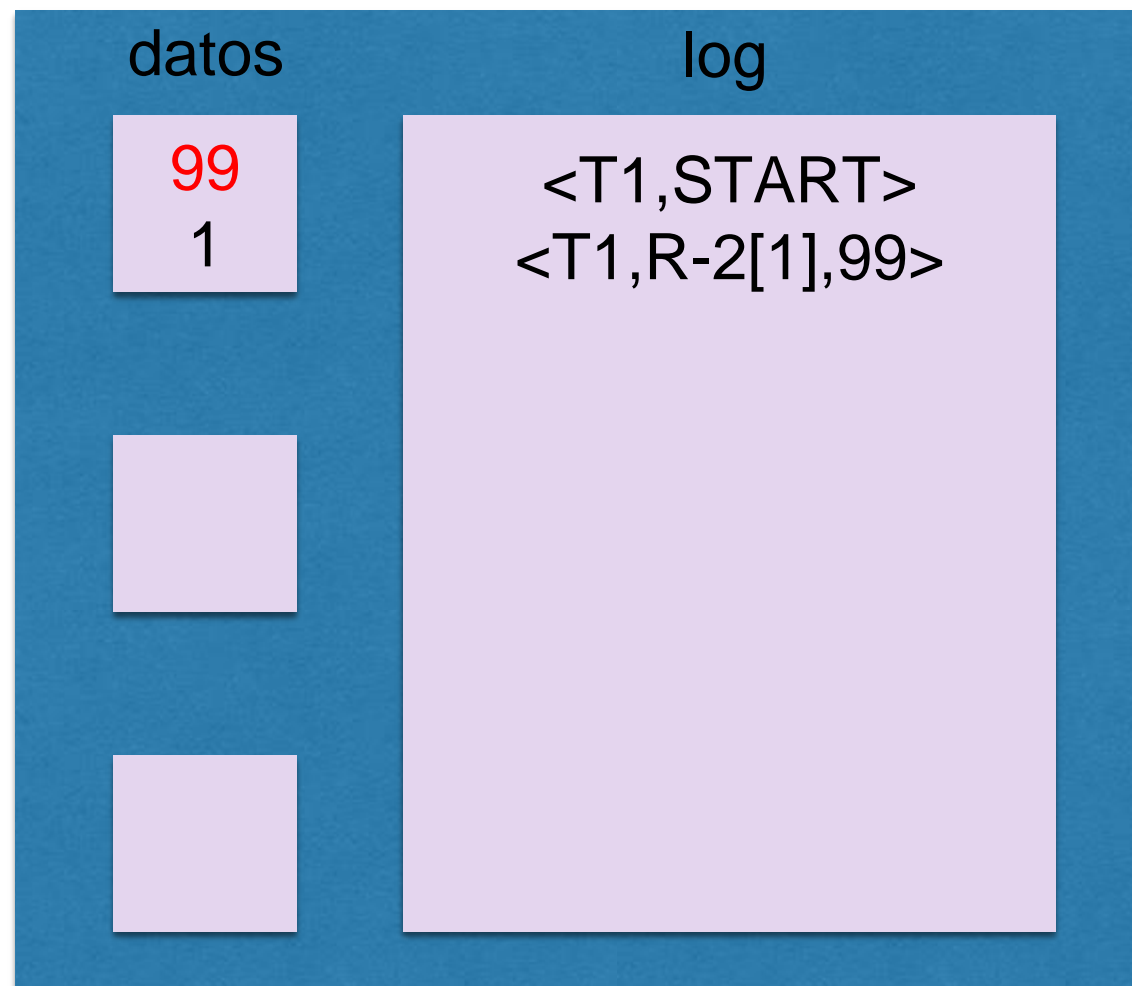


Redo Logging – en la BD

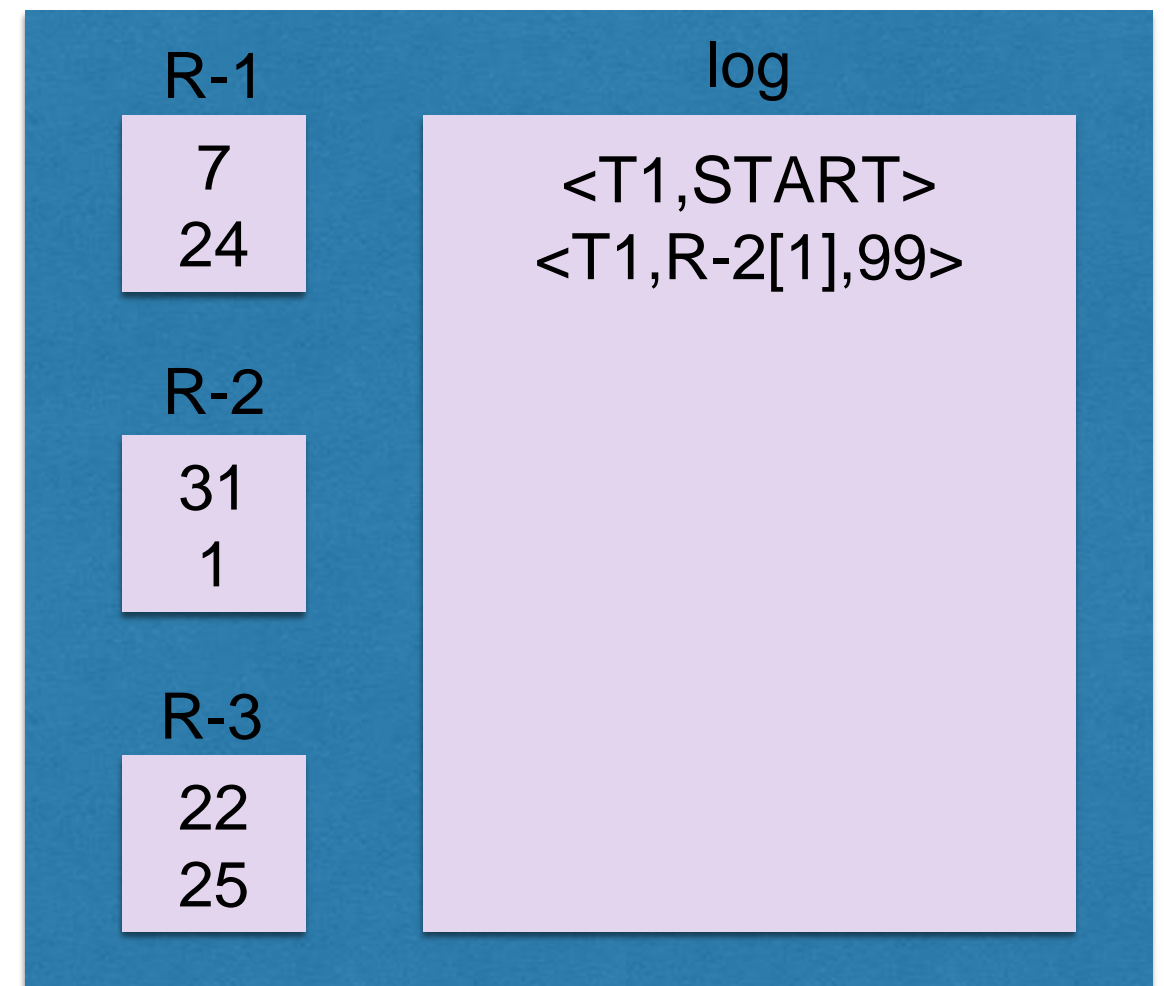
DB

T1: bota la ejecución

Buffer



Disco

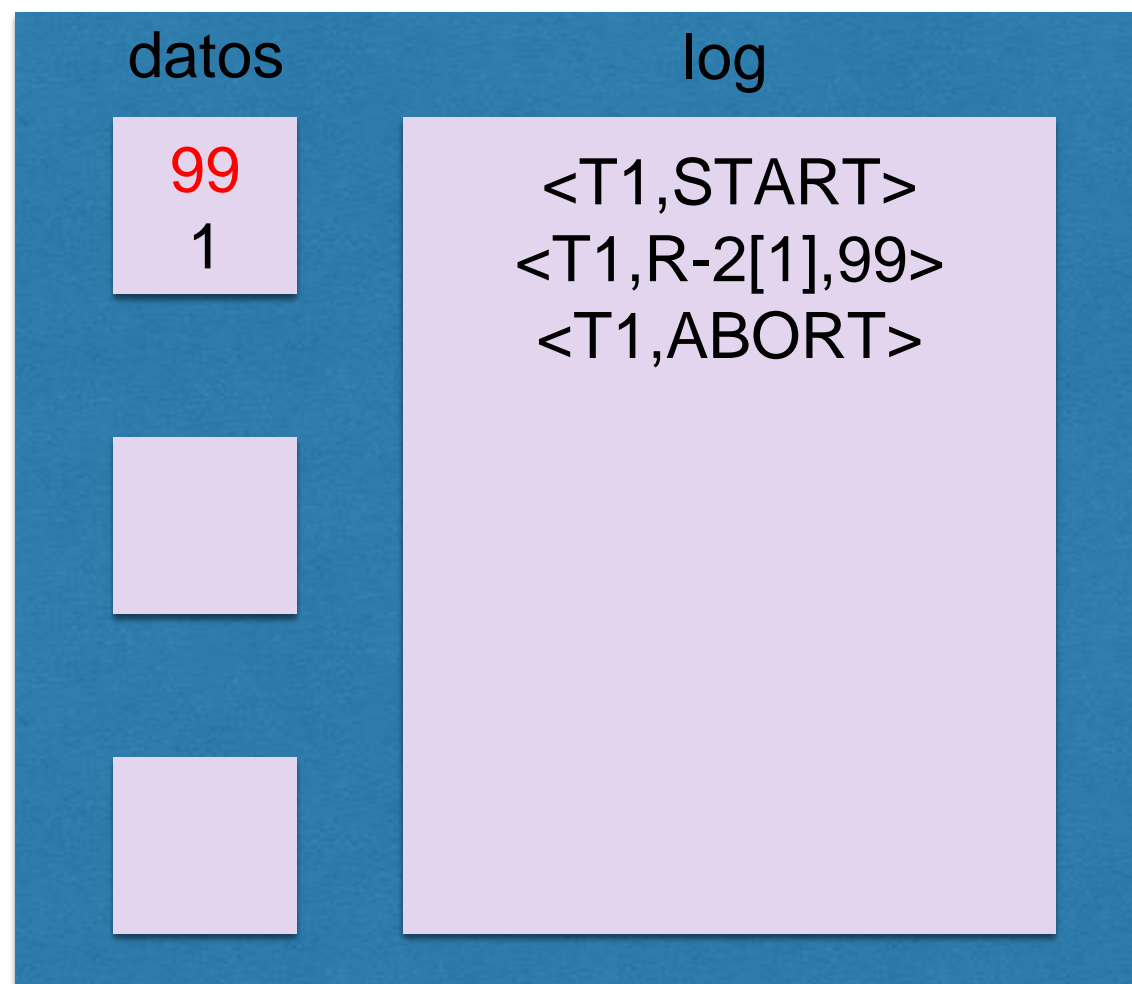


Redo Logging – en la BD

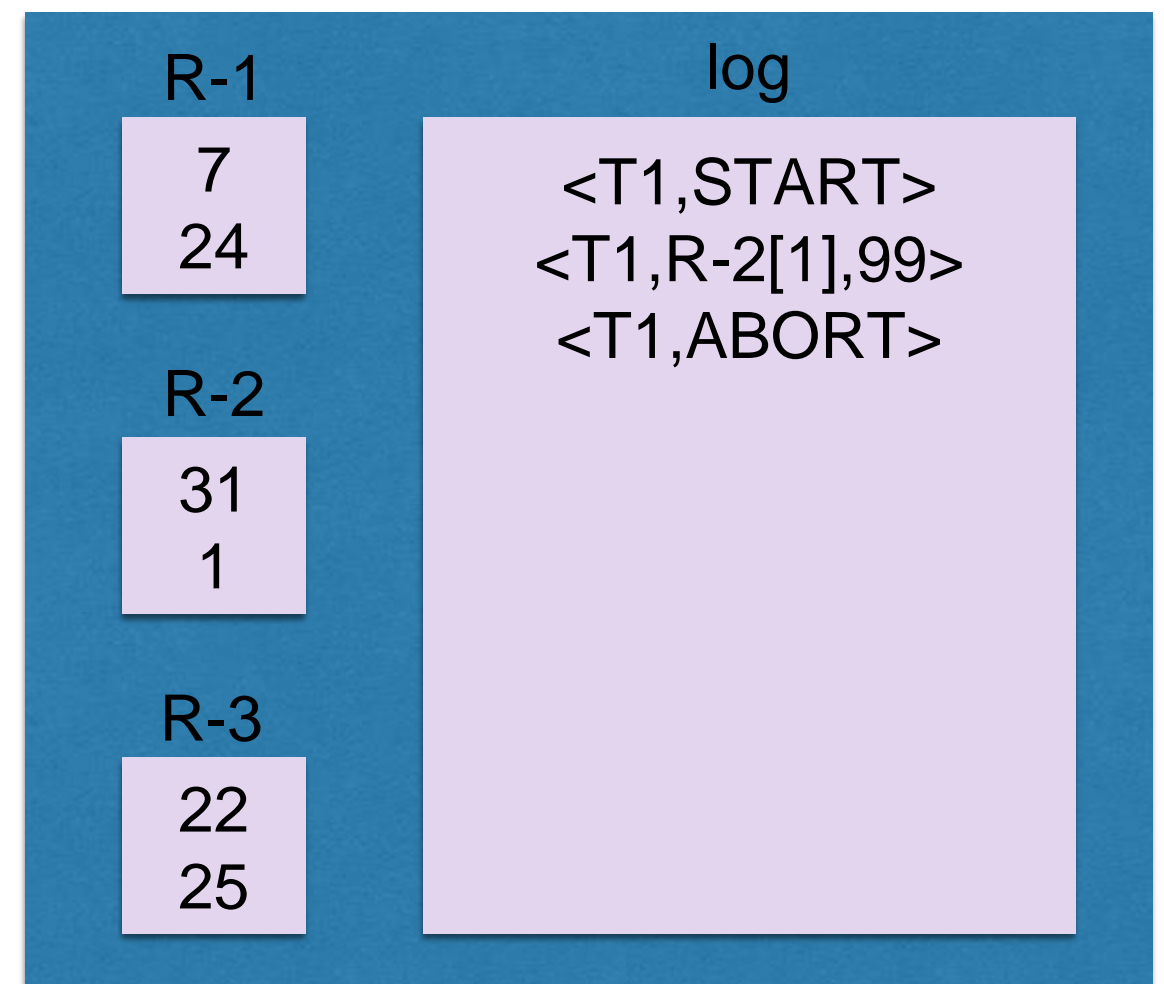
DB

T1: bota la ejecución

Buffer



Disco



Recuperación con Redo Logging

Detectando fallas en el *log*:

... <START T> ...

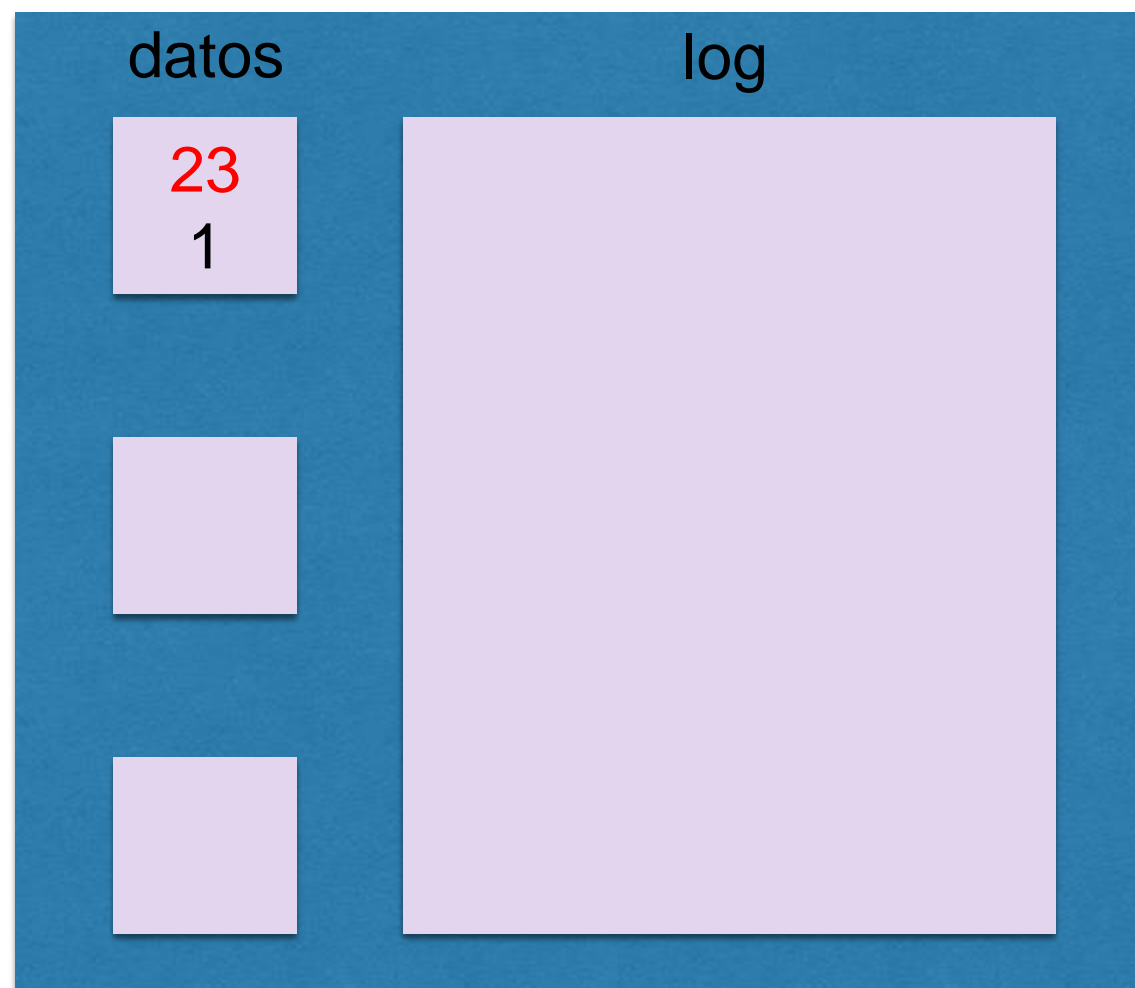


Redo Logging – en la BD

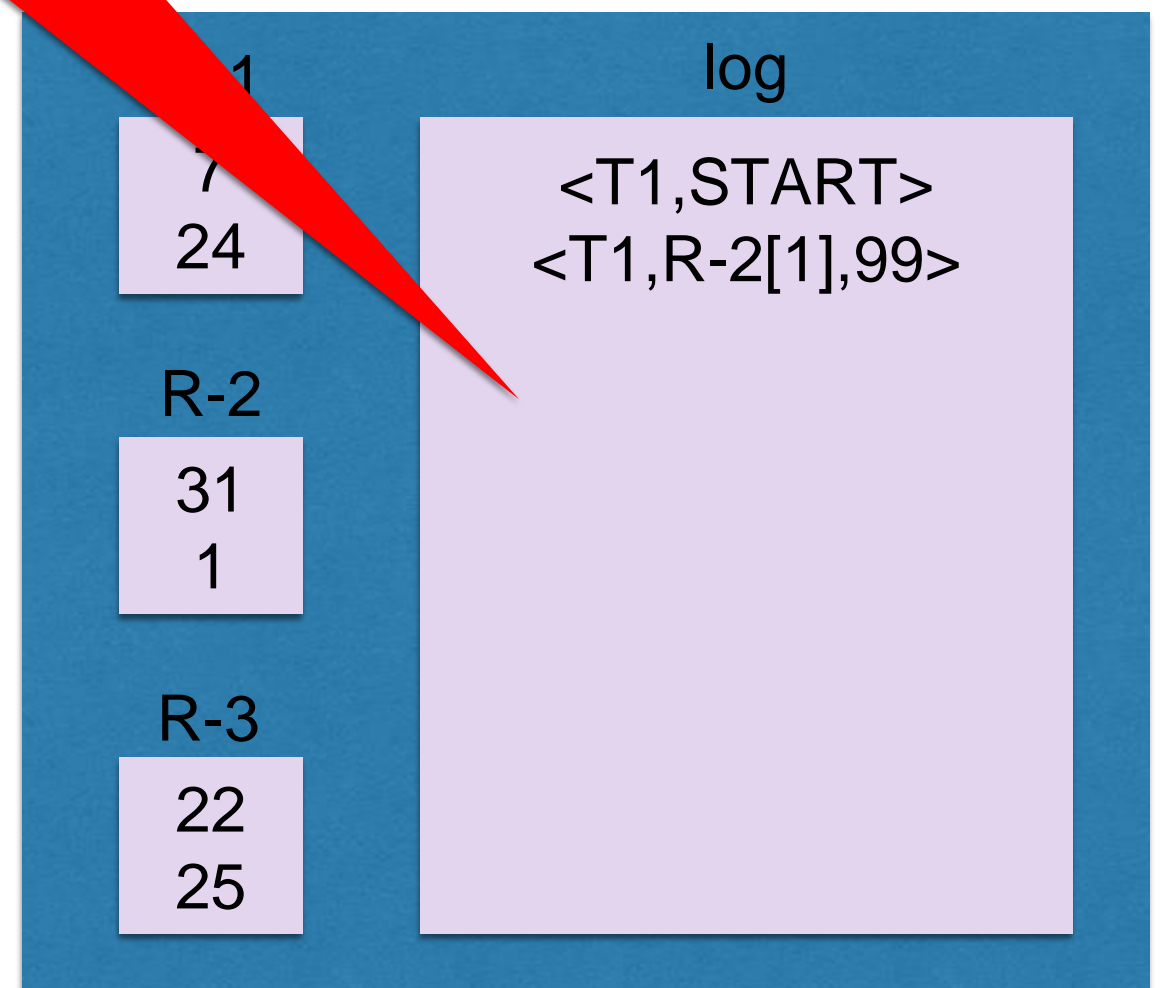
DB

No hay COMMIT
(No empecé escribir los datos)

Buffer



Disco



Recovery

Algoritmo para un *Redo Logging*

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

- Identificamos las transacciones que hicieron COMMIT *sin hacer END* (si hicieron END todo OK)
- Hacemos un *scan* desde el principio
- Si leo $\langle \mathbf{T}, X, v \rangle$:
 - Si \mathbf{T} no hizo COMMIT, no hacer nada
 - Si \mathbf{T} hizo COMMIT, reescribir con el valor v
- Para cada transacción sin COMMIT, escribir $\langle \text{ABORT } \mathbf{T} \rangle$

Recovery

Uso de *Checkpoints* en *Redo Logging*

¿Cómo utilizamos los checkpoints en el Redo Logging?

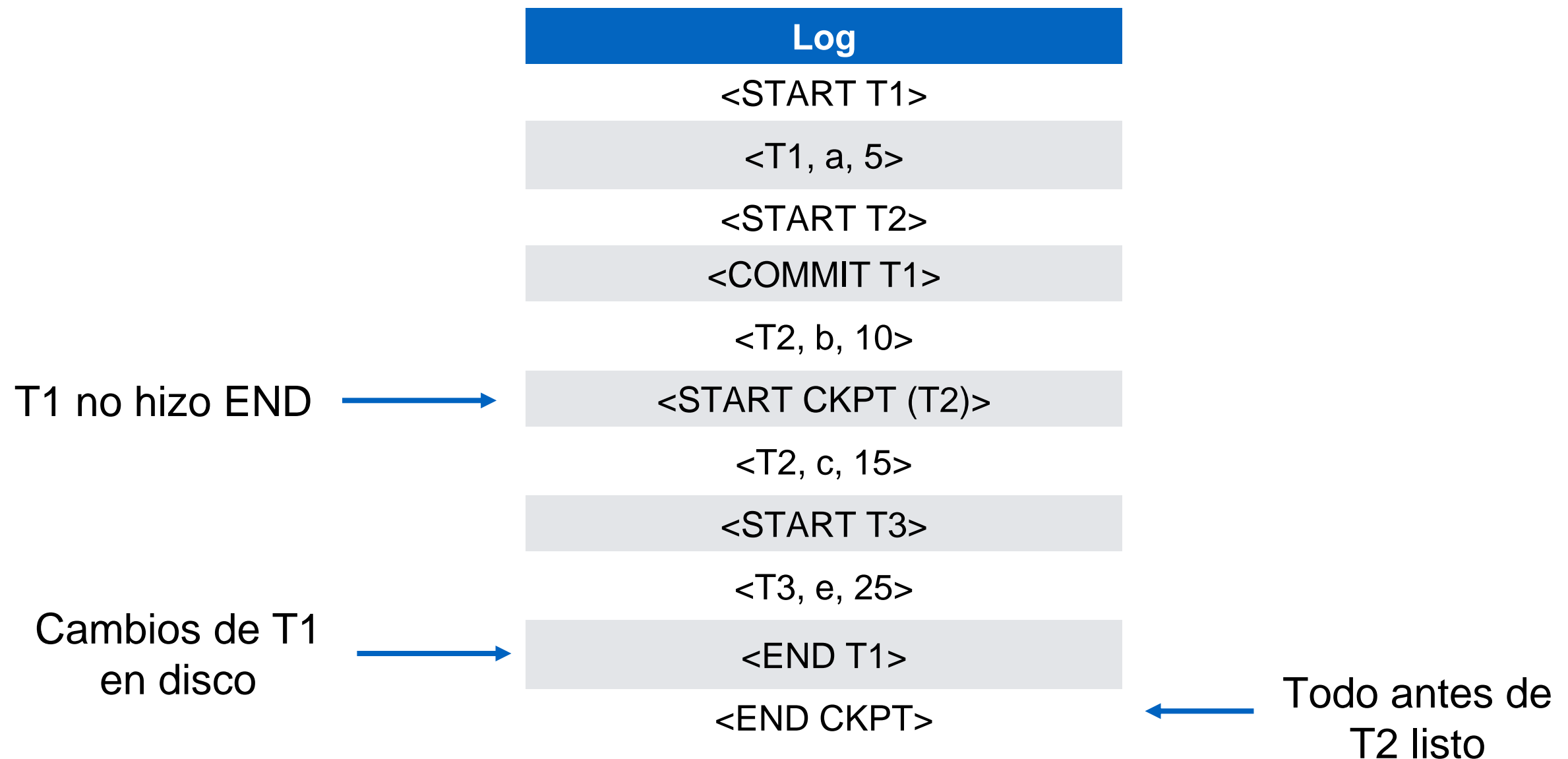
Recovery

Uso de *Checkpoints* en *Redo Logging*

- Escribimos un *log* $\langle \text{START CKPT } (T_1, \dots, T_n) \rangle$, donde T_1, \dots, T_n son transacciones activas y sin COMMIT
- Guardar todo el log en el disco
- Guardar en disco todo lo que haya hecho COMMIT hasta ese punto; escribir en log END al finalizar
- Una vez hecho, escribir $\langle \text{END CKPT} \rangle$

Ejemplo

Cuand finalizamos un checkpoint



Redo Recovery

Uso de *Checkpoints* en *Redo Logging*

- Revisar el *log* desde el final al inicio
- Si encontramos un $\langle \text{END CKPT} \rangle$, debemos retroceder hasta su respectivo $\langle \text{START CKPT} (T_1, \dots, T_n) \rangle$, y comenzar a hacer *redo* desde la transacción más antigua entre T_1, \dots, T_n – las *sin* END
- No se hace *redo* de las transacciones con COMMIT antes del $\langle \text{START CKPT} (T_1, \dots, T_n) \rangle$

Redo Recovery

Uso de *Checkpoints* en *Redo Logging*

- Revisar el *log* desde el final
- Si encontramos un $\langle \text{END CKPT} \rangle$, retroceder hasta su respectivo $\langle \text{START CKPT} (T_1, \dots, T_n) \rangle$, y comenzar a hacer *redo* desde la transacción más antigua entre T_1, \dots, T_n
- No se hace *redo* de las transacciones con COMMIT antes del $\langle \text{START CKPT} (T_1, \dots, T_n) \rangle$

Su END aparece antes de
 $\langle \text{END CKPT} \rangle$

Redo Recovery

Uso de *Checkpoints* en *Redo Logging*

- Revisar el *log* desde el final
- Si encontramos un $\langle \text{END CKPT } (T_1, \dots, T_n) \rangle$, retroceder hasta su respectivo $\langle \text{START CKPT } (T_1, \dots, T_n) \rangle$, y comenzar a hacer *redo* desde la transacción más antigua entre T_1, \dots, T_n
- No se hace *redo* de las transacciones con COMMIT antes del $\langle \text{START CKPT } (T_1, \dots, T_n) \rangle$

En realidad END de estas transacciones es redundante

Redo Recovery

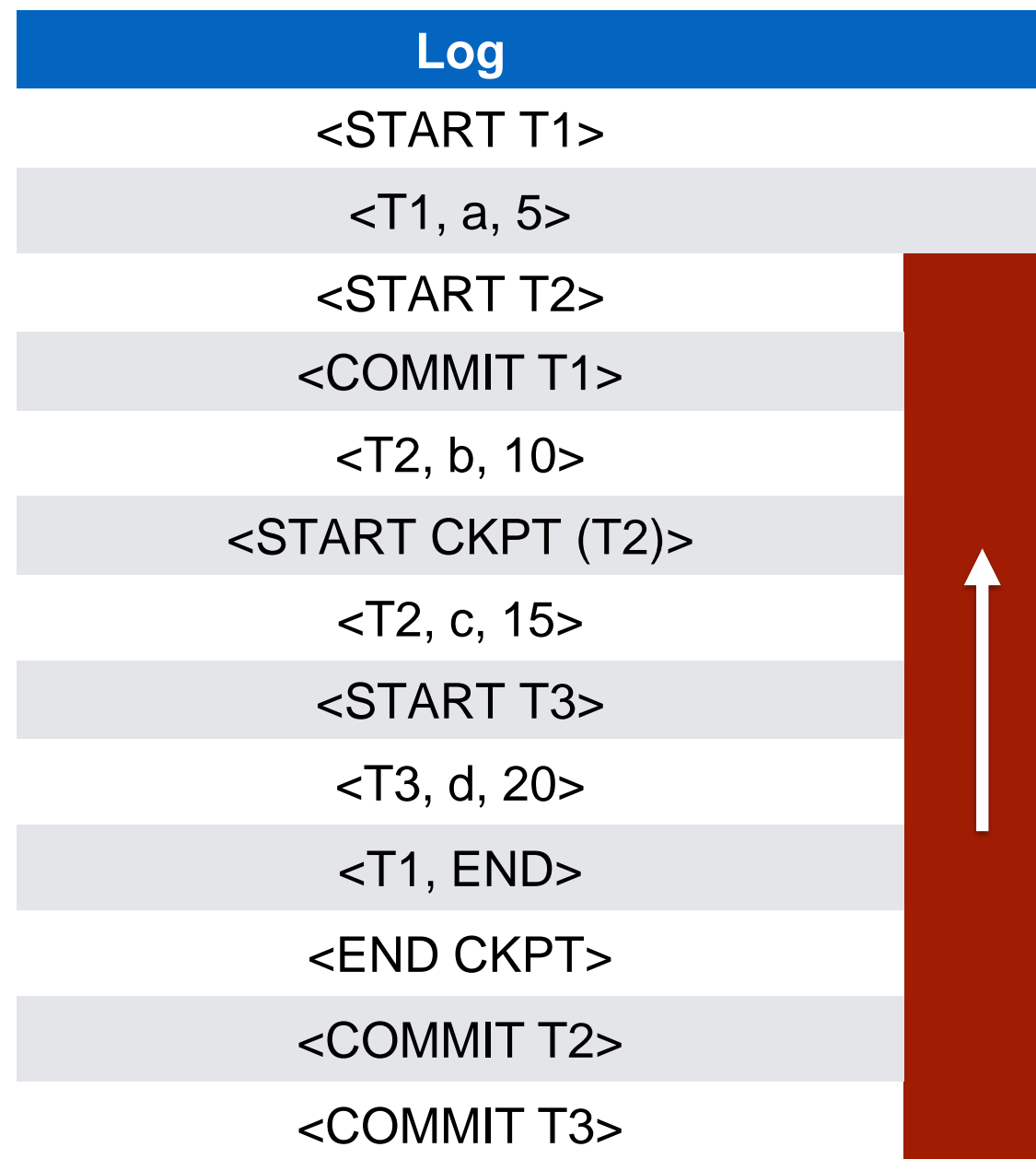
Uso de *Checkpoints* en *Redo Logging*

Si encontramos un $\langle \text{START CKPT } (T_1, \dots, T_n) \rangle$ sin su $\langle \text{END CKPT} \rangle$, debemos retroceder hasta encontrar un $\langle \text{END CKPT} \rangle$ más antiguo

Ejemplo

Uso de *Checkpoints* en *Redo Logging*

Considere este *log* después de una falla:



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, d, 20>
<T1, END>
<END CKPT>
<COMMIT T2>
<COMMIT T3>

Ahora REDO

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

Undo/Redo Logging

Los *logs* son:

- **<START T>**
- **<COMMIT T>**
- **<ABORT T>**
- **<T, X, v_{antiguo} , v_{nuevo} >**
- **<T,END>** (todo en disco – COMMIT y valor)

Undo/Redo Logging

El flujo:

- Log $\langle T, X, v_{\text{antiguo}}, v_{\text{nuevo}} \rangle$ va al disco antes de X
- (write-ahead logging)
- $\langle T, \text{COMMIT} \rangle$ va de manera arbitraria al disco
- (antes o después de X)
- $\langle T, \text{END} \rangle$ va cuando escribimos y datos y COMMIT

Undo/Redo Logging

Recuperación:

- Undo de transacciones sin COMMIT
- Redo transacciones con COMMIT y sin END

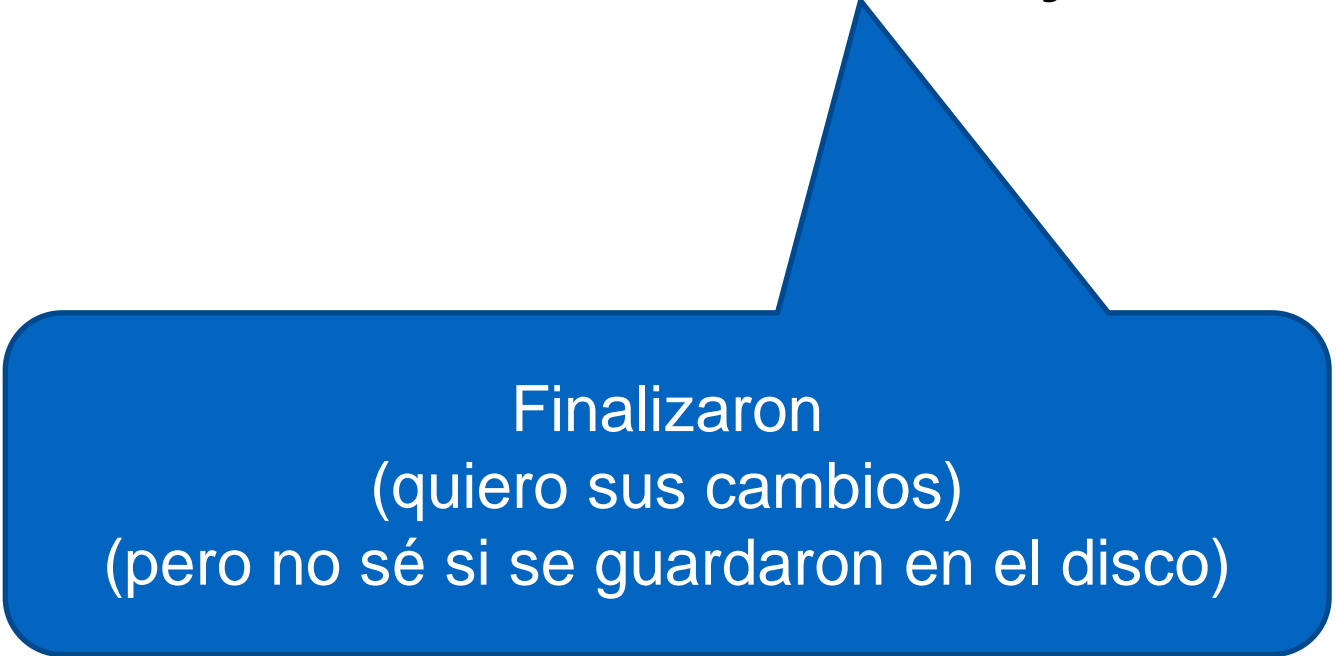
Undo/Redo Logging



No finalizaron
(no quiero tener cambios)

Recuperación:

- Undo de transacciones sin COMMIT
- Redo transacciones con COMMIT y sin END



Finalizaron
(quiero sus cambios)
(pero no sé si se guardaron en el disco)

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

Comentario importante

Para que todo esto funcione:

- Asumimos ACID
- En particular, que no hay conflictos RW, WR, o WW

Un buen recurso sobre logging

http://mlwiki.org/index.php/Database_Transaction_Log