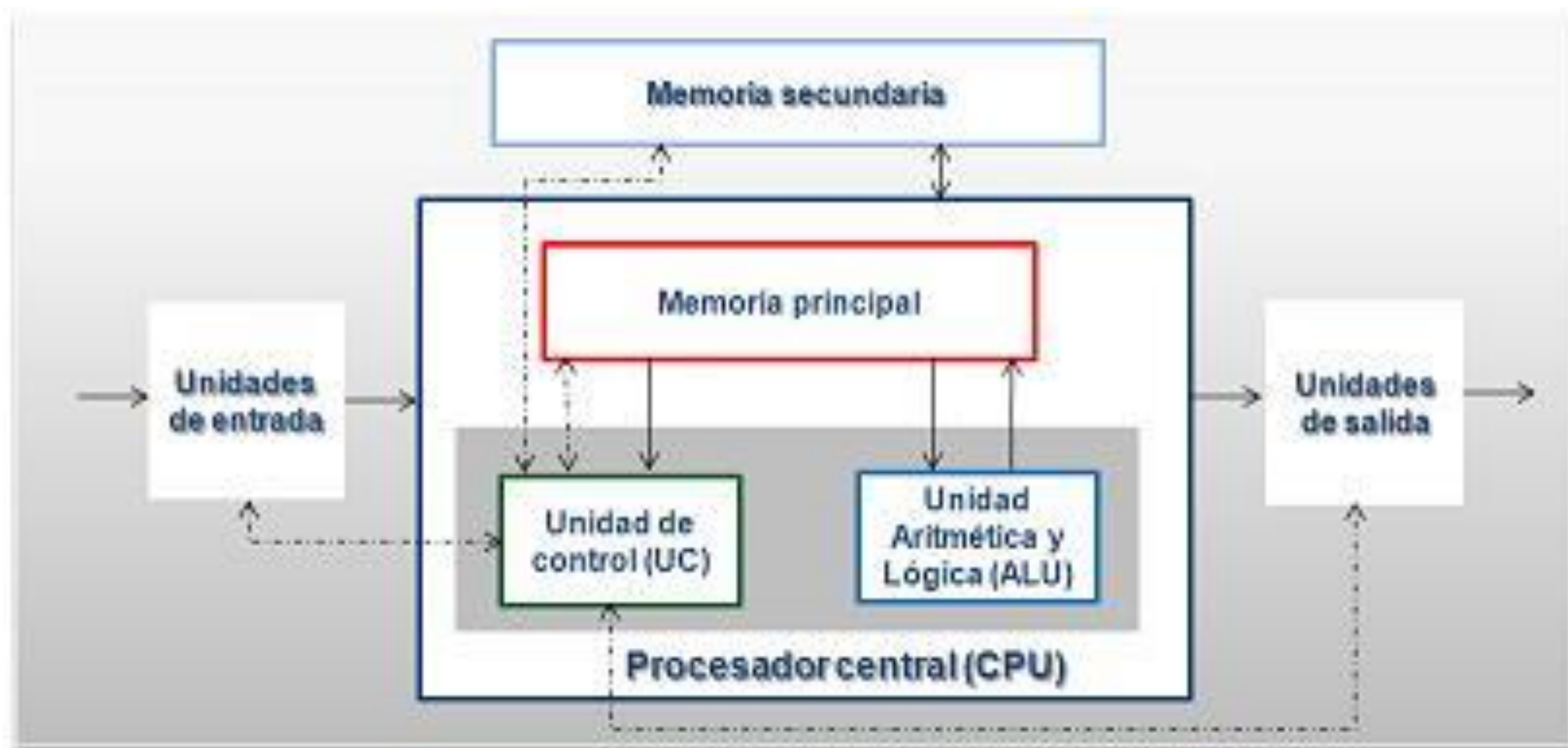


# Bases de Datos

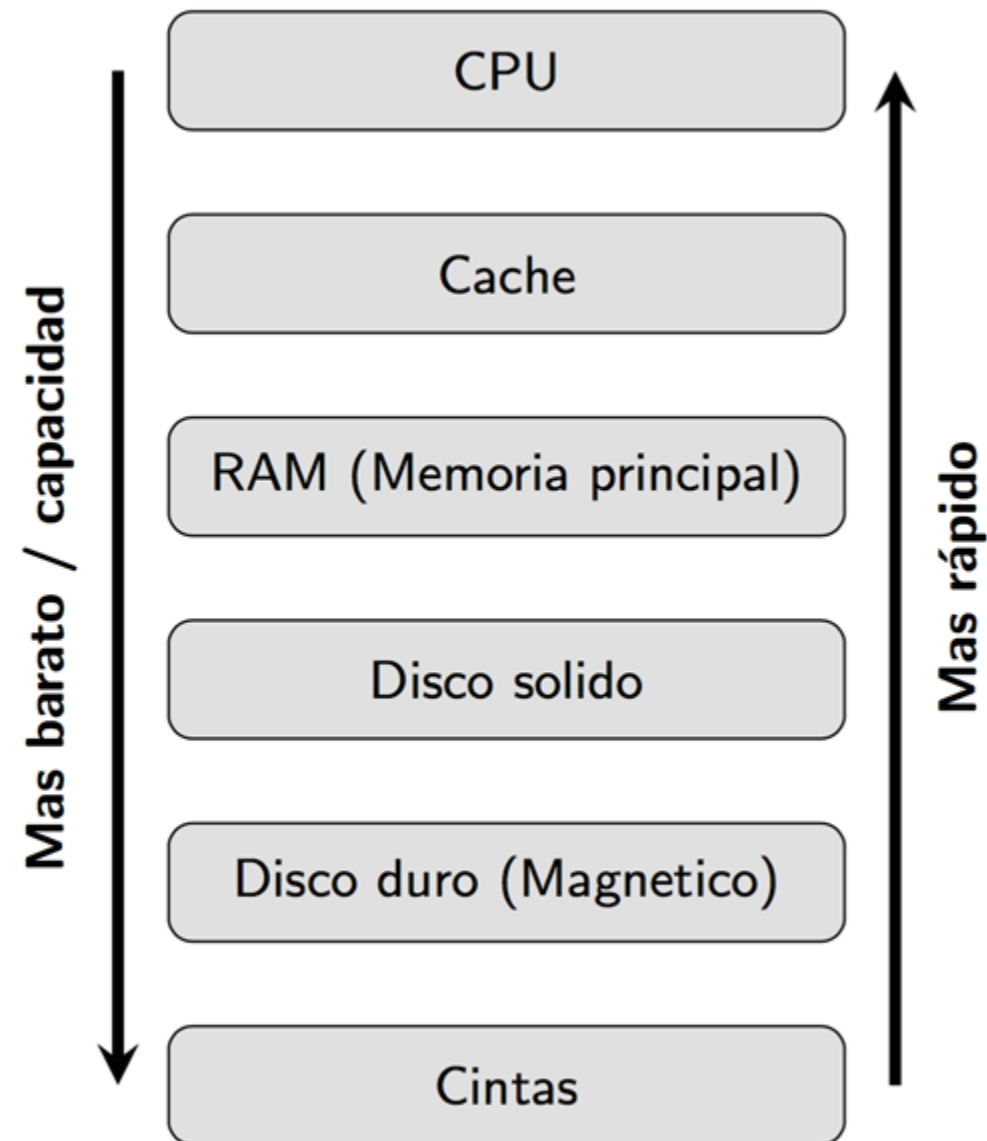
Clase 10: Arquitectura de Computadores y S.O, y  
Recuperación de Fallas

# Veamos un poco de Arquitectura de computadores y Sistemas Operativos

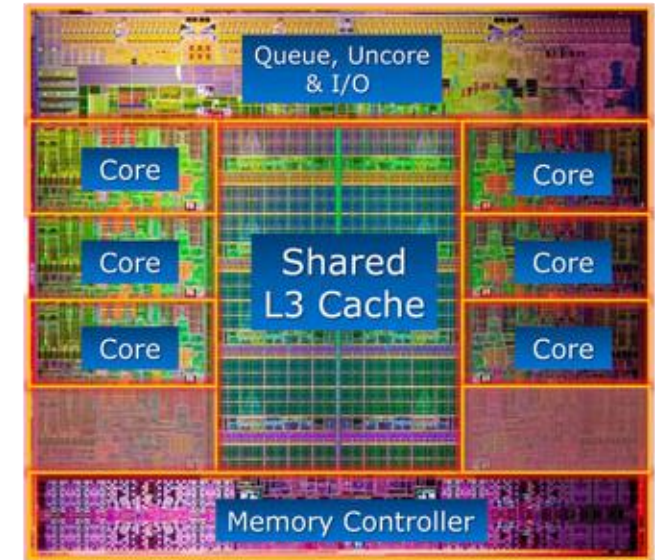


# Almacenamiento

## Jerarquía de Memoria



## Intel Core i7-3960X Processor Die Detail



ComputerHope.com



# Disco Duro

**Sector:** unidad física mínima de almacenamiento para el Disco (definido por hardware)

**Página:** unidad lógica mínima de almacenamiento para el Sistema de Archivos (definido por OS/DBMS)

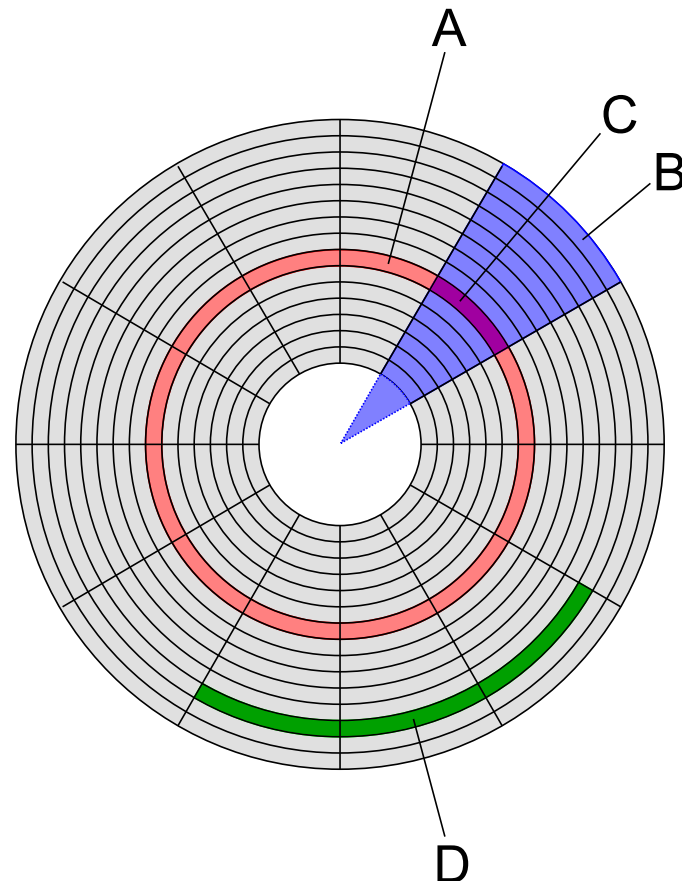
Estructura de disco magnético:

A es una **pista del disco** (roja),

B es un sector geométrico (azul),

C es un sector de una pista (magenta),

D es un grupo de sectores o clúster (verde).



**¡Un DBMS trabaja al nivel de página!**

# Implicancias en rendimiento al usar un disco Duro

**1. Operación de la DBMS:** Los datos deben estar en memoria para que la DBMS los procese.

**2. Transferencia de Datos:** La unidad de transferencia entre el disco y la memoria principal es un bloque. Si se necesita un elemento del bloque, se transfiere el bloque completo.

**3. Tiempo de Operación de I/O:** El tiempo para leer o escribir un bloque varía según la ubicación de los datos.

Tiempo de acceso = tiempo de búsqueda + retraso rotacional + tiempo de transferencia.

# Disco y RAM

**RAM**

so	so						
----	----	--	--	--	--	--	--

# Disco y RAM

## RAM

so	so						
----	----	--	--	--	--	--	--

swap	swap	Word	Word	Word	Spotify	Spotify	Fifa	Fifa	Fifa
------	------	------	------	------	---------	---------	------	------	------

# Disco y RAM

## RAM

so	so						
----	----	--	--	--	--	--	--

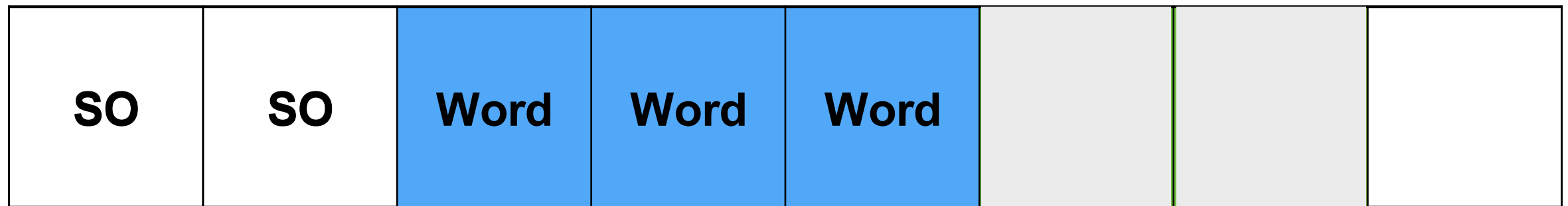
## Disco

...	Word	Word	Word	...
-----	------	------	------	-----

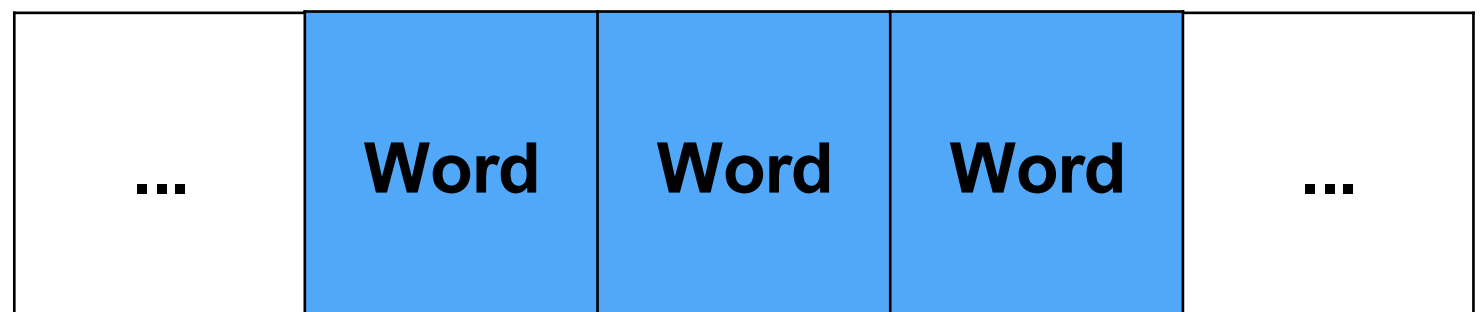


# Disco y RAM

## RAM



## Disco



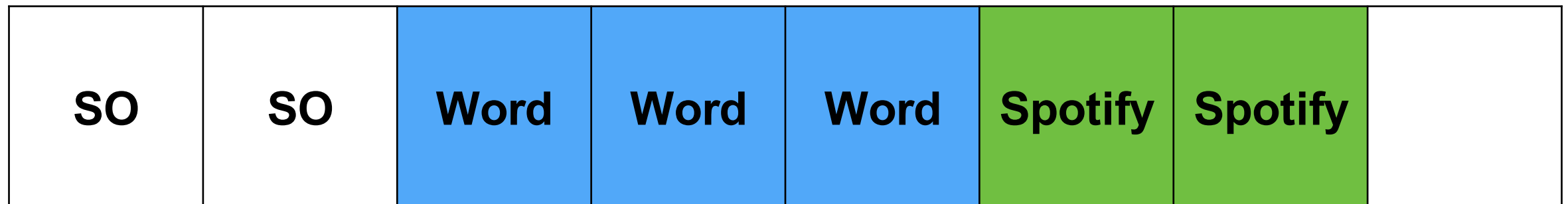
# Disco y RAM

**RAM**

SO	SO	Word	Word	Word	Spotify	Spotify	
----	----	------	------	------	---------	---------	--

# Disco y RAM

## RAM

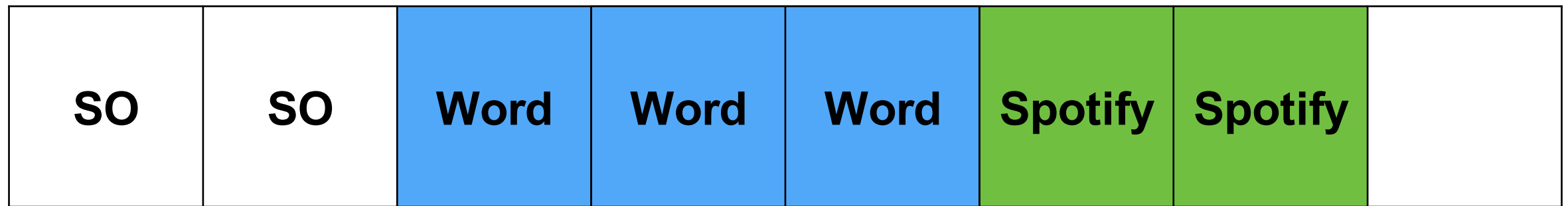


## Disco



# Disco y RAM

## RAM



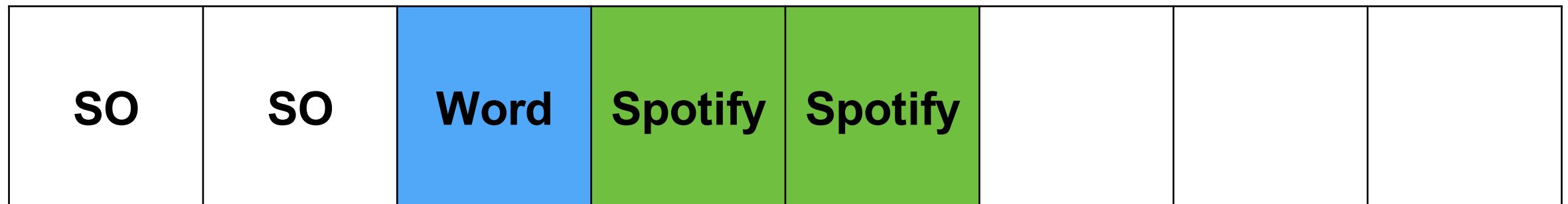
(espacio de swap)

## Disco



# Disco y RAM

## RAM



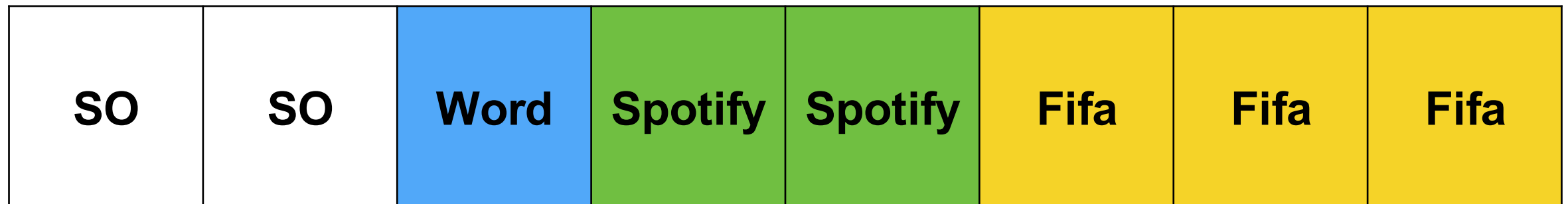
(espacio de swap)

## Disco Duro



# Disco y RAM

## RAM



(espacio de swap)

## Disco Duro

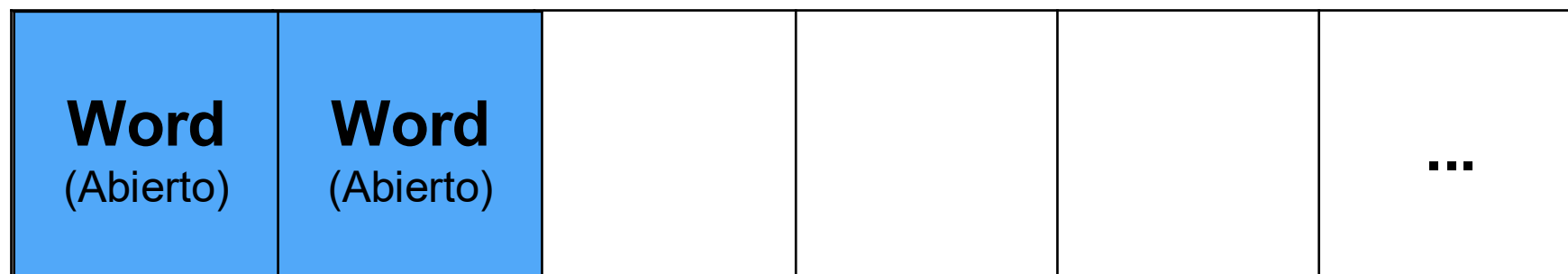


# Disco y RAM

## RAM



## Disco (espacio de swap)



# Disco y DBMS

Los records de las bases de datos se almacenan en **páginas** de disco.

A medida que se hace necesario, las páginas son traídas a memoria principal (**buffer**)



# Páginas, disco y buffer

Para trabajar con las tuplas de una relación, la base de datos carga la página con la tupla desde el disco

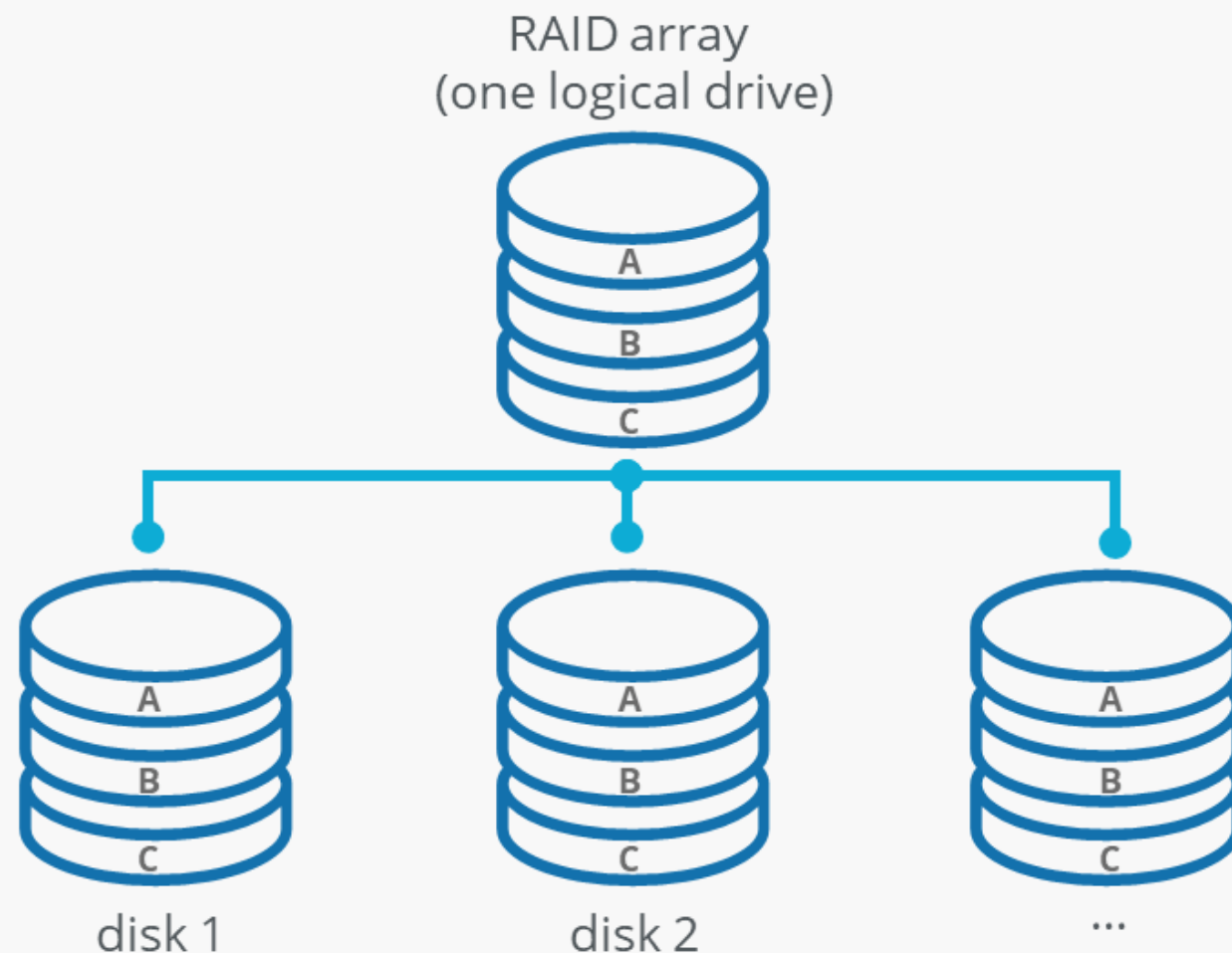
Para cargar estas páginas, la base de datos reserva un espacio en RAM llamado **Buffer**

RAID (Matriz redundante  
de discos independientes)

# RAID

Objetivo del RAID: Incrementar el rendimiento y la fiabilidad de sistemas de almacenamiento a través de un arreglo de múltiples discos que operan como uno solo.

## RAID (Redundant Array of Independent Disks)



# RAID - Técnicas

Técnicas Principales de un RAID:

- 1. Segmentación de Datos (Data Striping):** Los datos se dividen en bloques (unidades de segmentación) distribuidos en varios discos.
- 2. Redundancia:** Uso de discos adicionales para permitir la reconstrucción de datos en caso de fallos.

# RAID - Niveles

## Niveles de RAID:

- **Nivel 0:** Sin redundancia, solo segmentación.
- **Nivel 1:** Espejado, cada disco tiene una copia idéntica.
- **Nivel 0+1:** Combinación de segmentación y espejado.
- **Nivel 3:** Paridad entrelazada por bit.
- **Nivel 4:** Paridad entrelazada por bloque.
- **Nivel 5:** Paridad distribuida, similar al Nivel 4 pero con bloques de paridad distribuidos entre todos los discos.

# Disco Solido

**Celda:** Unidad mínima que puede almacenar bits (definida por el tipo de SSD, como SLC, MLC, TLC, QLC)

**Bloque:** Conjunto de páginas que es la unidad mínima de borrado

**Plano:** Conjunto de bloques gestionados como una unidad

**IDE:** Un chip de memoria en el SSD que puede contener múltiples planos

# Disco Duro vs Disco Solido



## SSD vs HDD

Usually 10 000 or 15 000 rpm SAS drives

**0.1 ms**

### Access times

SSDs exhibit virtually no access time

**5.5 ~ 8.0 ms**

SSDs deliver at least

**6000 io/s**

### Random I/O Performance

SSDs are at least 15 times faster than HDDs

HDDs reach up to

**400 io/s**

SSDs have a failure rate of less than

**0.5 %**

### Reliability

This makes SSDs 4 - 10 times more reliable

HDD's failure rate fluctuates between

**2 ~ 5 %**

SSDs consume between

**2 & 5 watts**

### Energy savings

This means that on a large server like ours, approximately 100 watts are saved

HDDs consume between

**6 & 15 watts**

SSDs have an average I/O wait of

**1 %**

### CPU Power

You will have an extra 6% of CPU power for other operations

HDDs' average I/O wait is about

**7 %**

the average service time for an I/O request while running a backup remains below

**20 ms**

### Input/Output request times

SSDs allow for much faster data access

the I/O request time with HDDs during backup rises up to

**400 ~ 500 ms**

SSD backups take about

**6 hours**

### Backup Rates

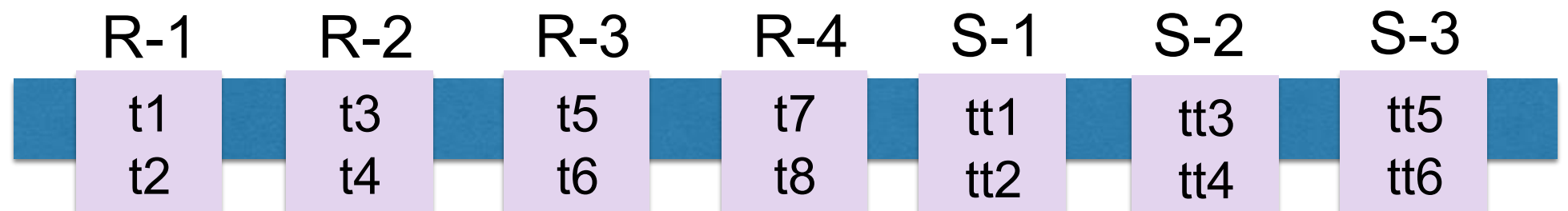
SSDs allows for 3 - 5 times faster backups for your data

HDD backups take up to

**20 ~ 24 hours**

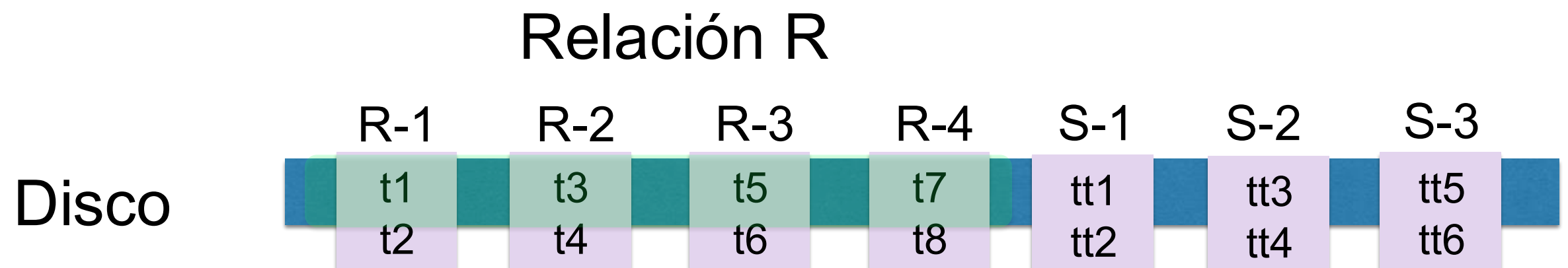
# Páginas, disco y buffer

Disco

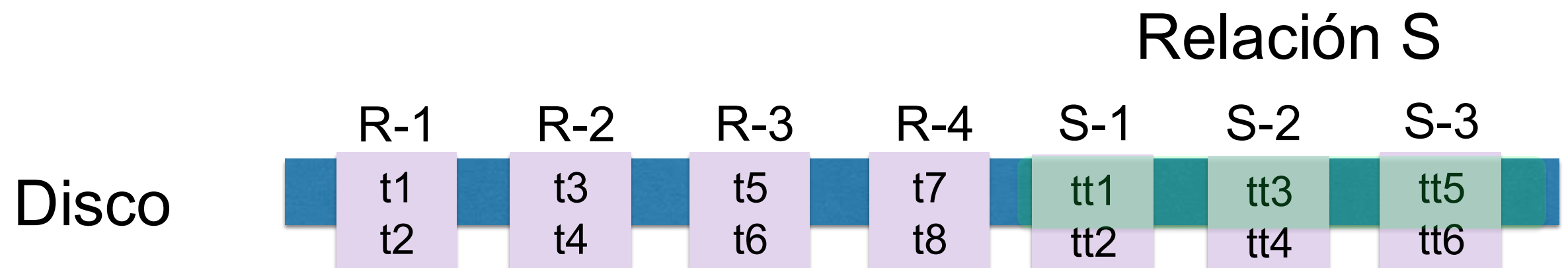




# Páginas, disco y buffer

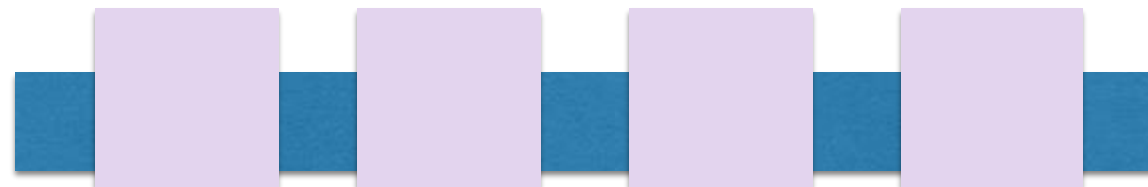


# Páginas, disco y buffer

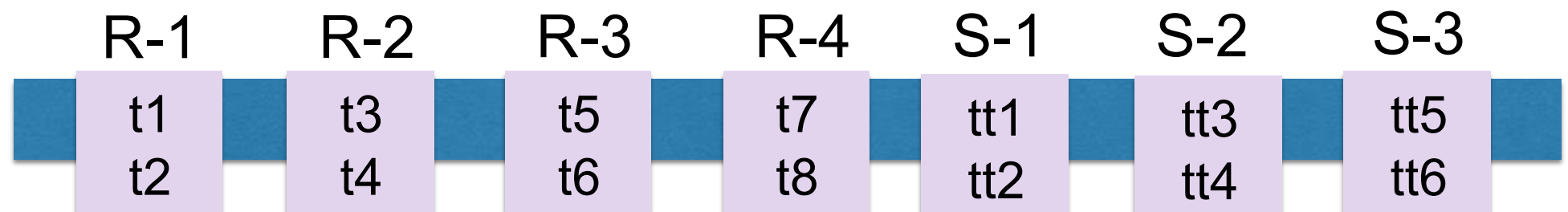


# Páginas, disco y buffer

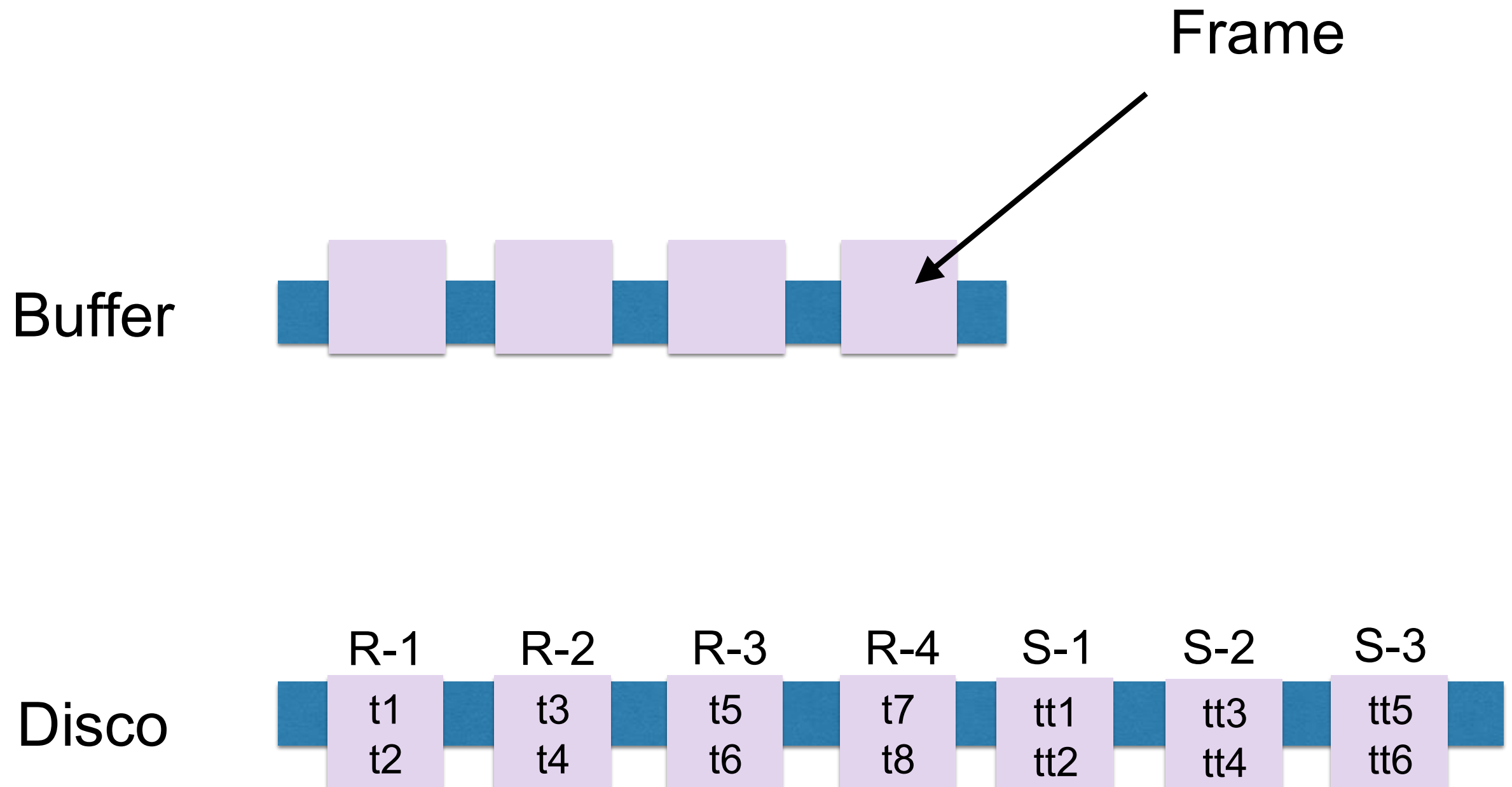
Buffer



Disco



# Páginas, disco y buffer

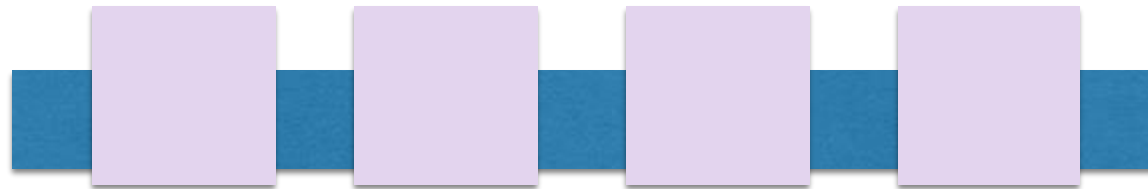


# Páginas, disco y buffer

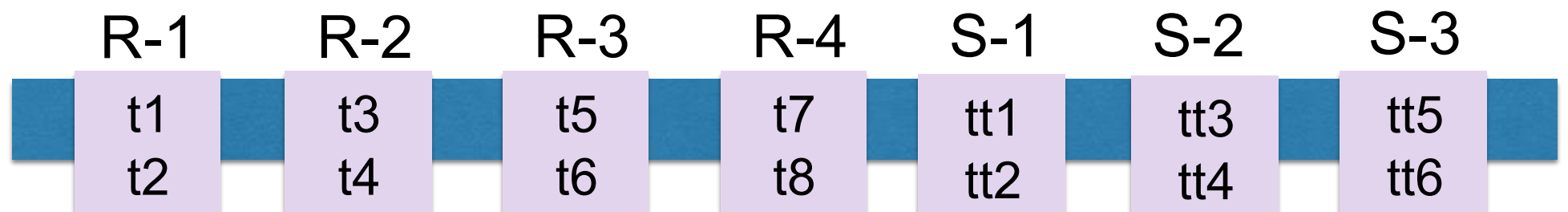
DB

Necesito tupla t3 de R!

Buffer



Disco



# Páginas, disco y buffer

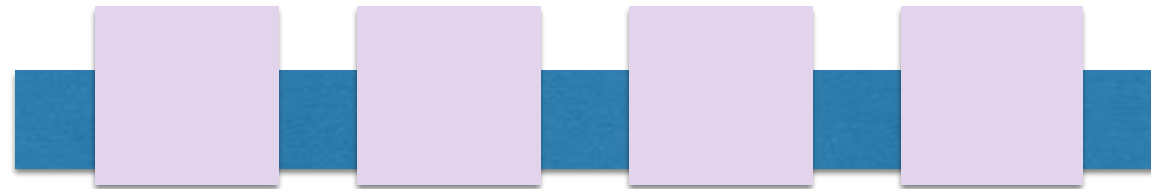
DB

Necesito tupla t3 de R!

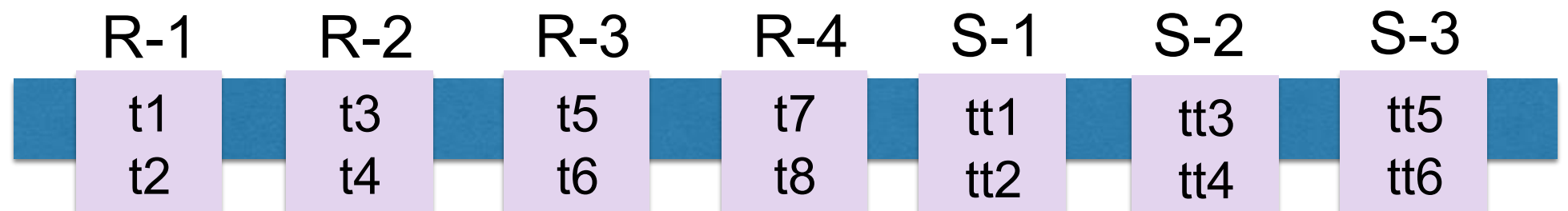


Cargar página con t3

Buffer



Disco



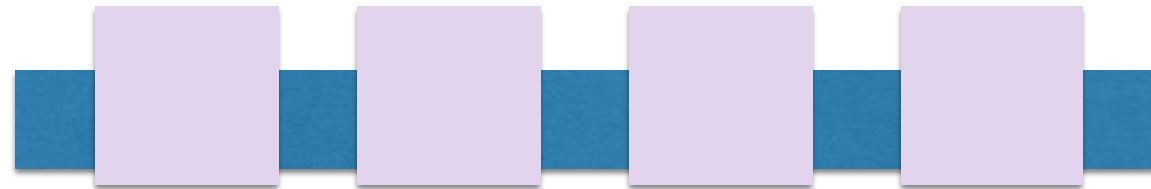
# Páginas, disco y buffer

DB

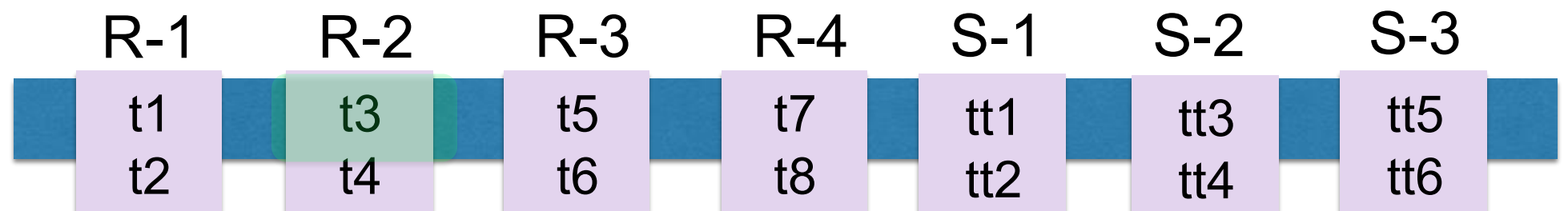
Necesito tupla t3 de R!

↓ Cargar página con t3

Buffer



Disco



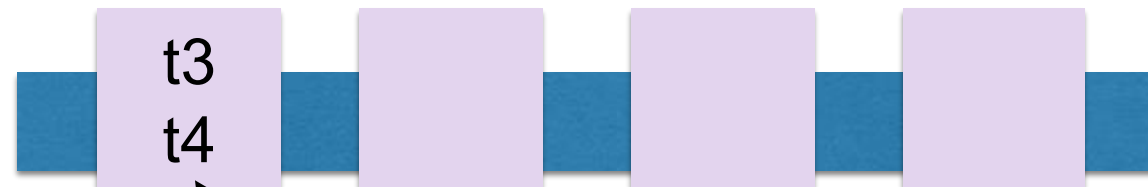
# Páginas, disco y buffer

DB

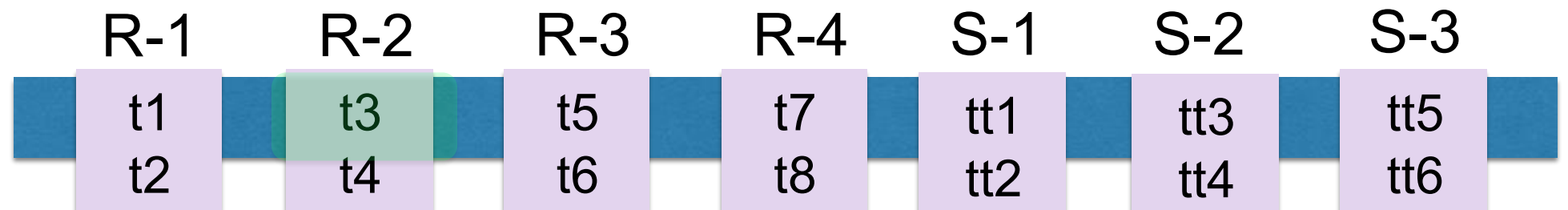
Necesito tupla t3 de R!

↓ Cargar página con t3

Buffer



Disco

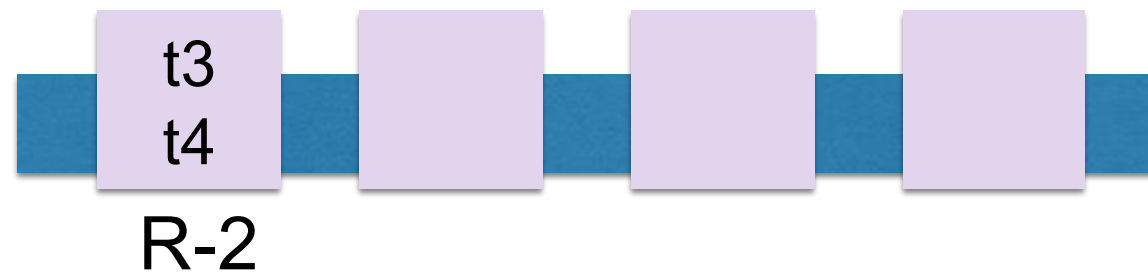




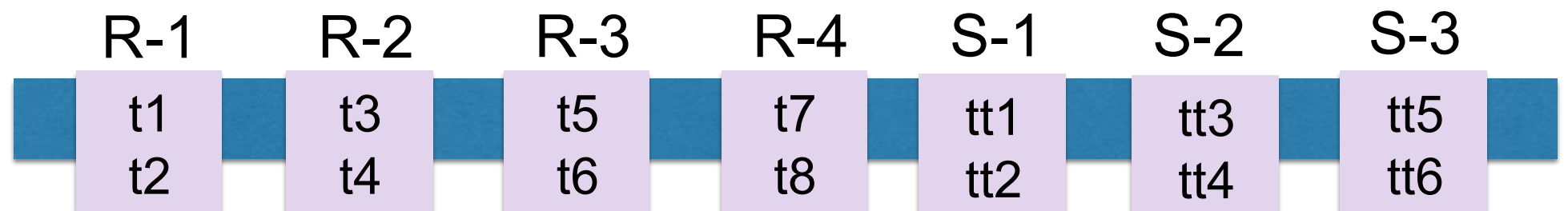
# Páginas, disco y buffer

DB

Buffer



Disco

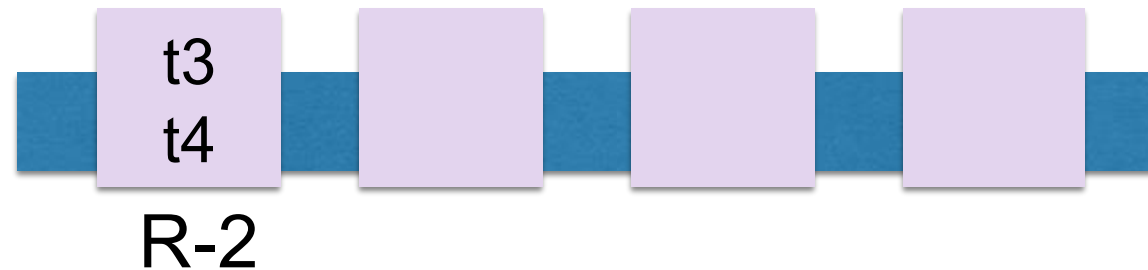


# Páginas, disco y buffer

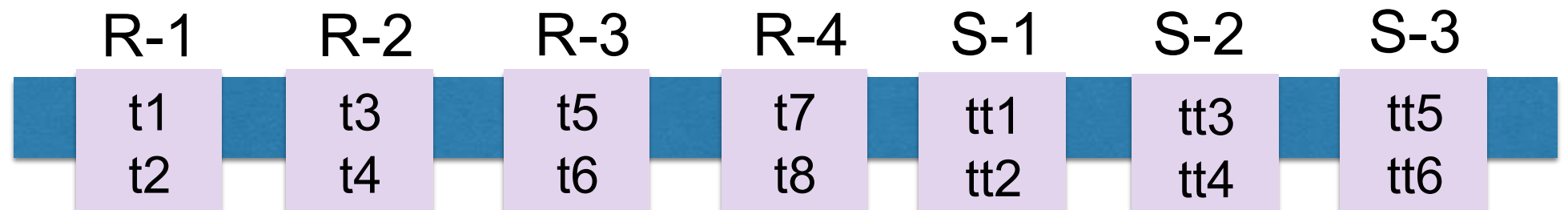
DB

Necesito tupla t4 de R!

Buffer



Disco



# Páginas, disco y buffer

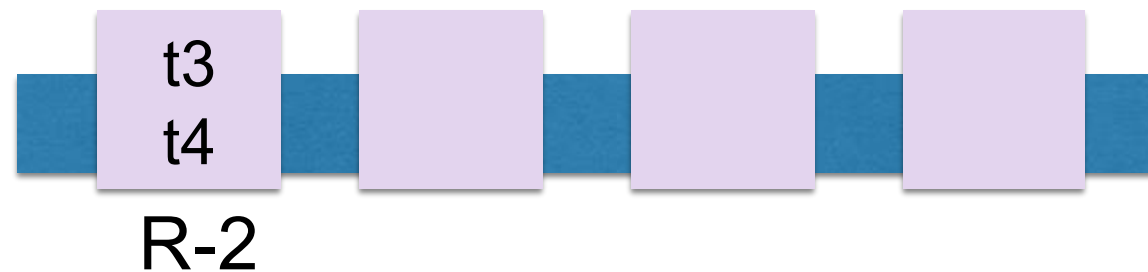
DB

Necesito tupla t4 de R!

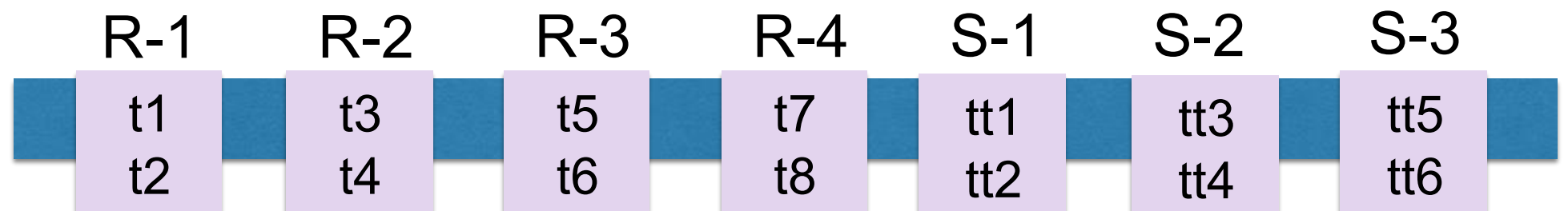


Cargar página con t4

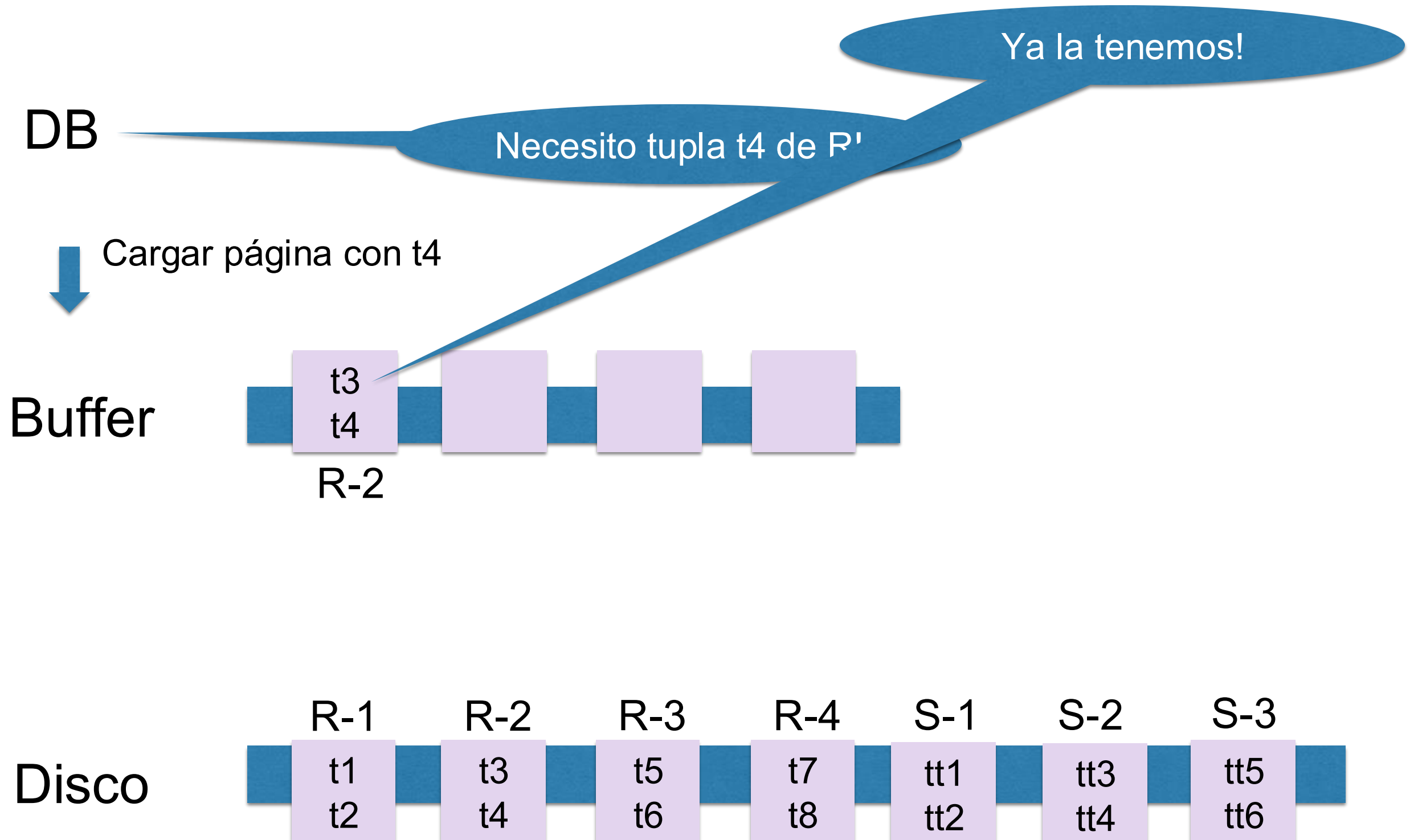
Buffer



Disco



# Páginas, disco y buffer



# Fallas en Transacciones

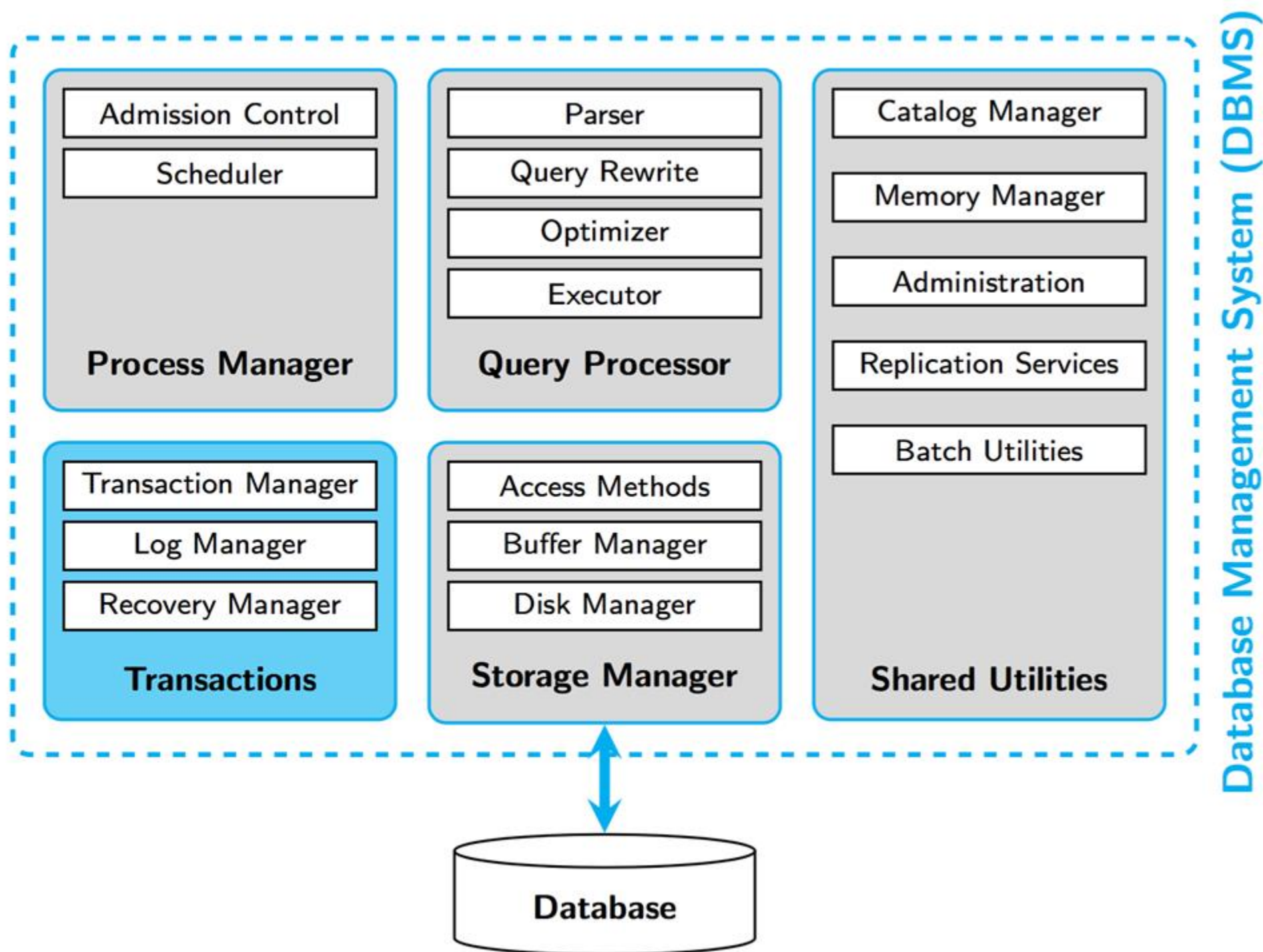
**EL SERVIDOR HA FALLADO**

**¿DÓNDE ESTÁ LA COPIA DE SEGURIDAD?**

**EN EL SERVIDOR....**



# Repaso: Transacciones



# Transacciones

Es un componente que asegura las propiedades **ACID**



**Atomicity**  
**Consistency**  
**Isolation**  
**Durability**

**Atomicidad**  
**Consistencia**  
**aislamiento**  
**Durabilidad**



# ACID



**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.

**En esta clase nos centraremos en Atomicity y Durability.**

# Transacciones



**Transaction Manager** se encarga de asegurar Isolation y Consistency

**Log y Recovery Manager** se encargan de asegurar **Atomicity y Durability**

# Tipos de fallas en Transacciones

# Fallas en la ejecución de transacciones

**1. Datos Erróneos:** pueden ingresar en el sistema debido a errores humanos, fallas en la validación de los datos o problemas en la integración de sistemas externos. Soluciones:

- **Restricciones de integridad:** Implementar restricciones a nivel de base de datos como claves primarias, claves foráneas, y restricciones de unicidad y verificación.
- **Limpieza de datos (Data Cleaning):** Usar procesos para detectar y corregir registros erróneos o incompletos. Esto incluye la normalización de datos, la eliminación de duplicados y la corrección de formatos incorrectos.

# Fallas en la ejecucion de transacciones

**2. Fallas en el Almacenamiento:** pueden ocurrir debido a fallos físicos en los dispositivos de almacenamiento, lo que puede resultar en la pérdida de datos críticos. Soluciones:

- **RAID (Redundant Array of Independent Disks):** Utilizar diferentes configuraciones de RAID para proteger los datos. Por ejemplo, RAID 1 duplica los datos en dos discos duros, mientras que RAID 5 distribuye los datos junto con la paridad a través de tres o más discos.
- **Copias Redundantes:** Mantener copias de seguridad regulares en dispositivos de almacenamiento separados.

# Fallas en la ejecucion de transacciones

**3. Catástrofes:** Eventos imprevistos y graves como incendios, inundaciones, terremotos o ataques cibernéticos que pueden destruir los equipos físicos y los datos almacenados. Soluciones:

- **Copias Distribuidas:** Implementar un sistema de copias de datos distribuidas geográficamente. Esto asegura que, si una ubicación es afectada por una catástrofe, los datos pueden ser recuperados desde otra ubicación intacta.
- **Respaldos Totales e Incrementales:** Realizar copias de seguridad completas de toda la base de datos a intervalos regulares. También se deben tener respaldos incrementales, los cuales solo guardan los cambios desde el último respaldo, sea total o incremental.

# Fallas en la ejecucion de transacciones

**4. Fallas del Sistema:** Interrupciones en el funcionamiento normal del sistema de base de datos, ya sea por fallos de software, hardware o errores humanos que afectan la integridad de las transacciones. Solución:

- **Log y Recovery Manager**



# Log Manager



# Ejemplo de log

2023-10-24 02:39:13.320 UTC [1125037] grupo3@grupo3e2 ERROR: type "varcahr" does not exist at character 60

2023-10-24 02:39:13.320 UTC [1125037] grupo3@grupo3e2 STATEMENT: create table genero\_en\_pelicula(id INT PRIMARY KEY, genero VARCAHR(40), pid INT, titulo VARCHAR(60));

2023-10-24 03:42:54.934 UTC [1134253] grupo3@grupo3e2 ERROR: column "pid" of relation "visualizaciones\_series" does not exist

2023-10-24 03:42:54.934 UTC [1134253] grupo3@grupo3e2 STATEMENT: COPY visualizaciones\_series ( id\_visualizacion, uid, pid, fecha ) FROM STDIN DELIMITER ',' CSV HEADER;

2023-10-24 04:09:19.361 UTC [1136729] grupo3@grupo3e2 ERROR: column "id" of relation "cancelaciones" does not exist

2023-10-24 04:09:19.361 UTC [1136729] grupo3@grupo3e2 STATEMENT: COPY cancelaciones ( id,estado, fecha\_inicio, pro\_id,uid, proveedor ) FROM STDIN DELIMITER ',' CSV HEADER;

2023-10-24 04:09:35.542 UTC [1136729] grupo3@grupo3e2 ERROR: relation "subscripciones" does not exist

2023-10-24 04:09:35.542 UTC [1136729] grupo3@grupo3e2 STATEMENT: COPY subscripciones ( id,estado, fecha\_inicio, pro\_id,uid, proveedor ) FROM STDIN DELIMITER ',' CSV HEADER;

2023-10-24 04:12:00.863 UTC [1136729] grupo3@grupo3e2 ERROR: relation "subscripciones" does not exist

2023-10-24 04:12:00.863 UTC [1136729] grupo3@grupo3e2 STATEMENT: COPY subscripciones ( id,estado, fecha\_inicio, pro\_id,uid, proveedor ) FROM STDIN DELIMITER ',' CSV HEADER;

# Log Manager

Registra todas las acciones de las transacciones

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 Records

Los *logs* comunes son:

- <START T>
- <COMMIT T>
- <ABORT T>
- <T UPDATE>

¿Cómo los usamos?, con **Loggins**.

# Undo Logging

# Undo Logging

Forma de escribir los *logs* para poder hacer *recovery* del sistema

# Undo Logging

Los *logs* son:

- **<START T>**
- **<COMMIT T>**
- **<ABORT T>**
- **<T, X, t>** donde t es el valor **antiguo** de X

# Undo Logging - Reglas

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  $\langle T, X, t \rangle$
  - Escribir todos logs  $\langle T, X, t \rangle$  al disco
  - Escribir valor nuevo de  $X$  a disco
  - Escribir  $\langle \text{COMMIT } T \rangle$
- } repeat
- } write-ahead logging



# Ejemplo de log de transacción

Start transaction T1

Reemplace el valor 31 de R.a por 99

Reemplace el valor 99 de R.a por 23

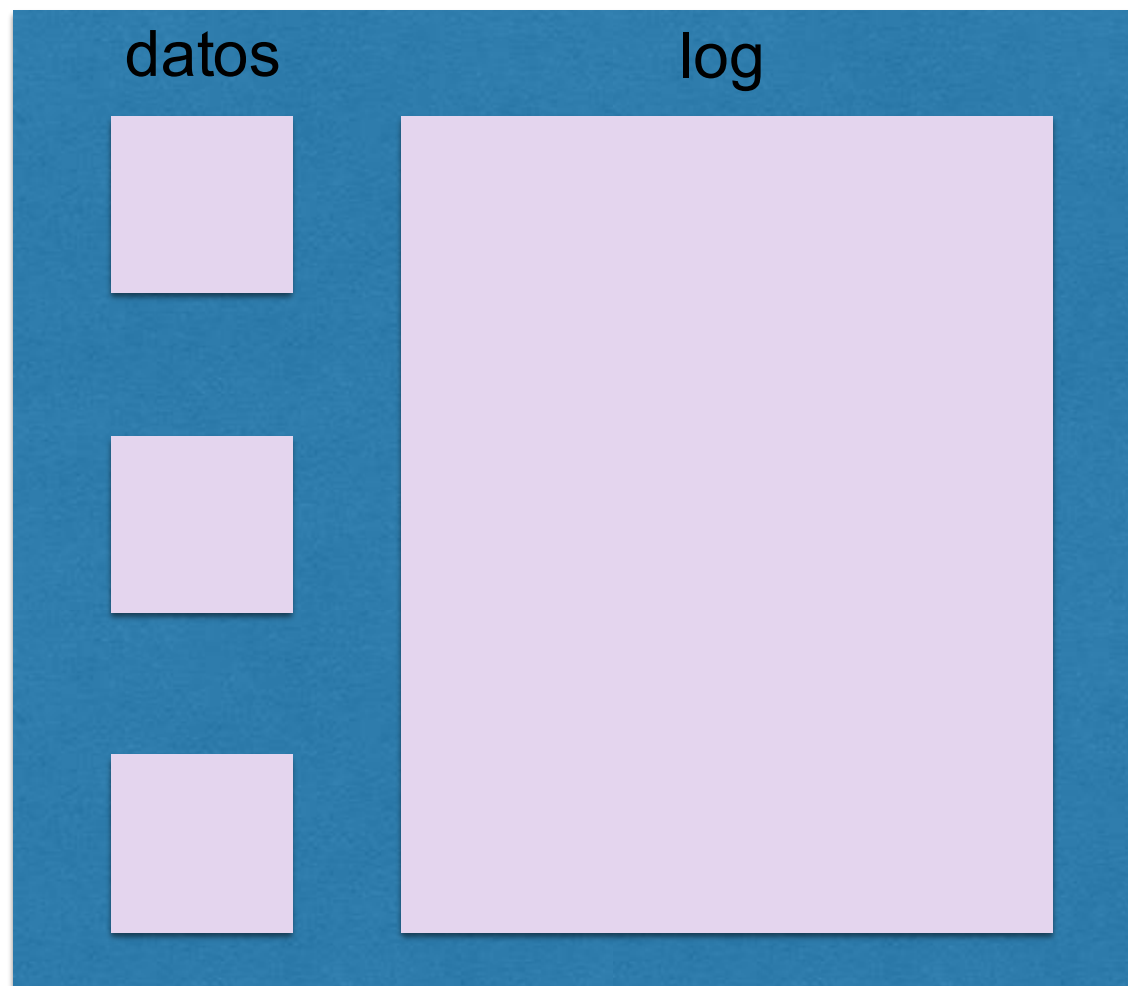
Commit T1

# Undo Logging – en la BD

DBMS

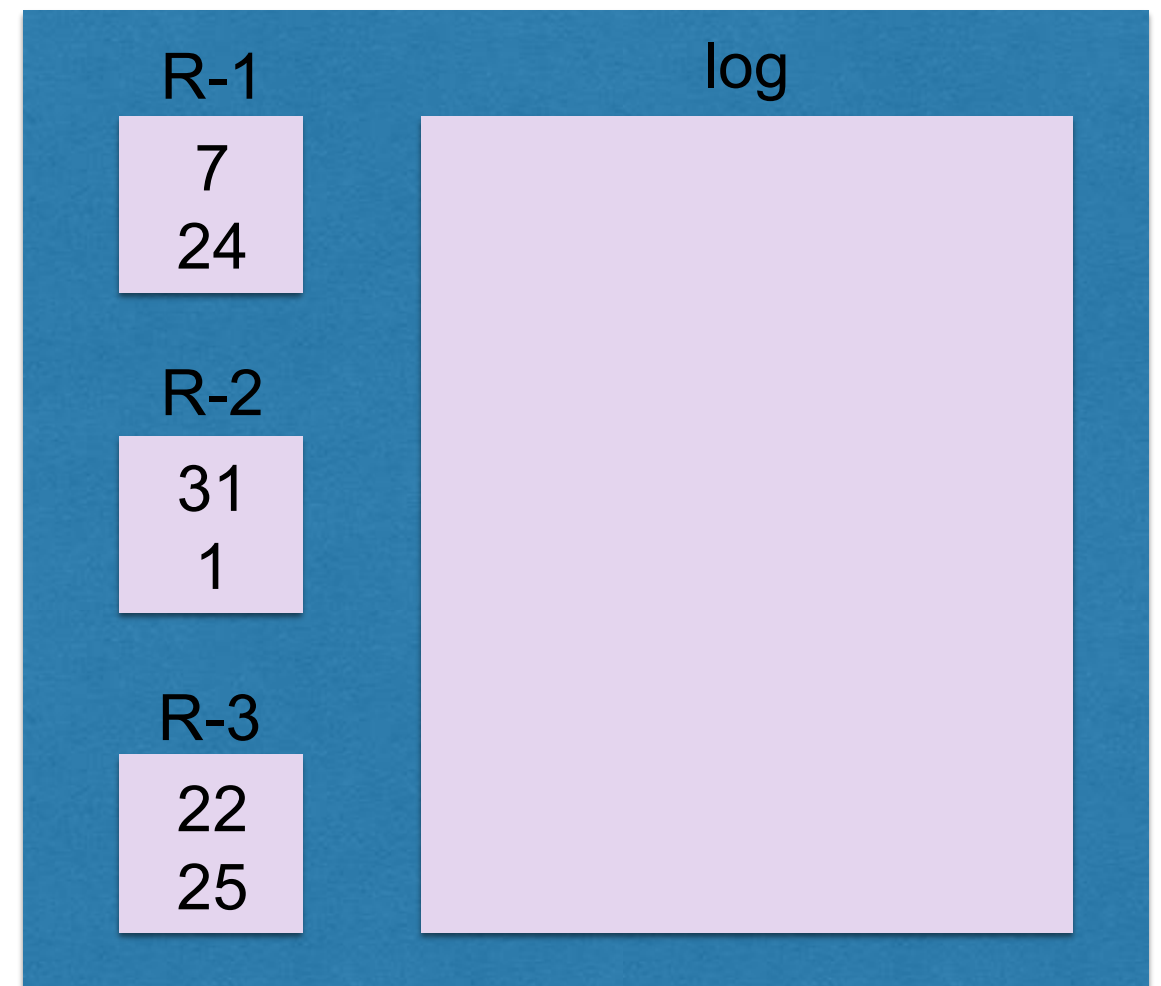
T1: voy a empezar

Buffer



R(A int)

Disco

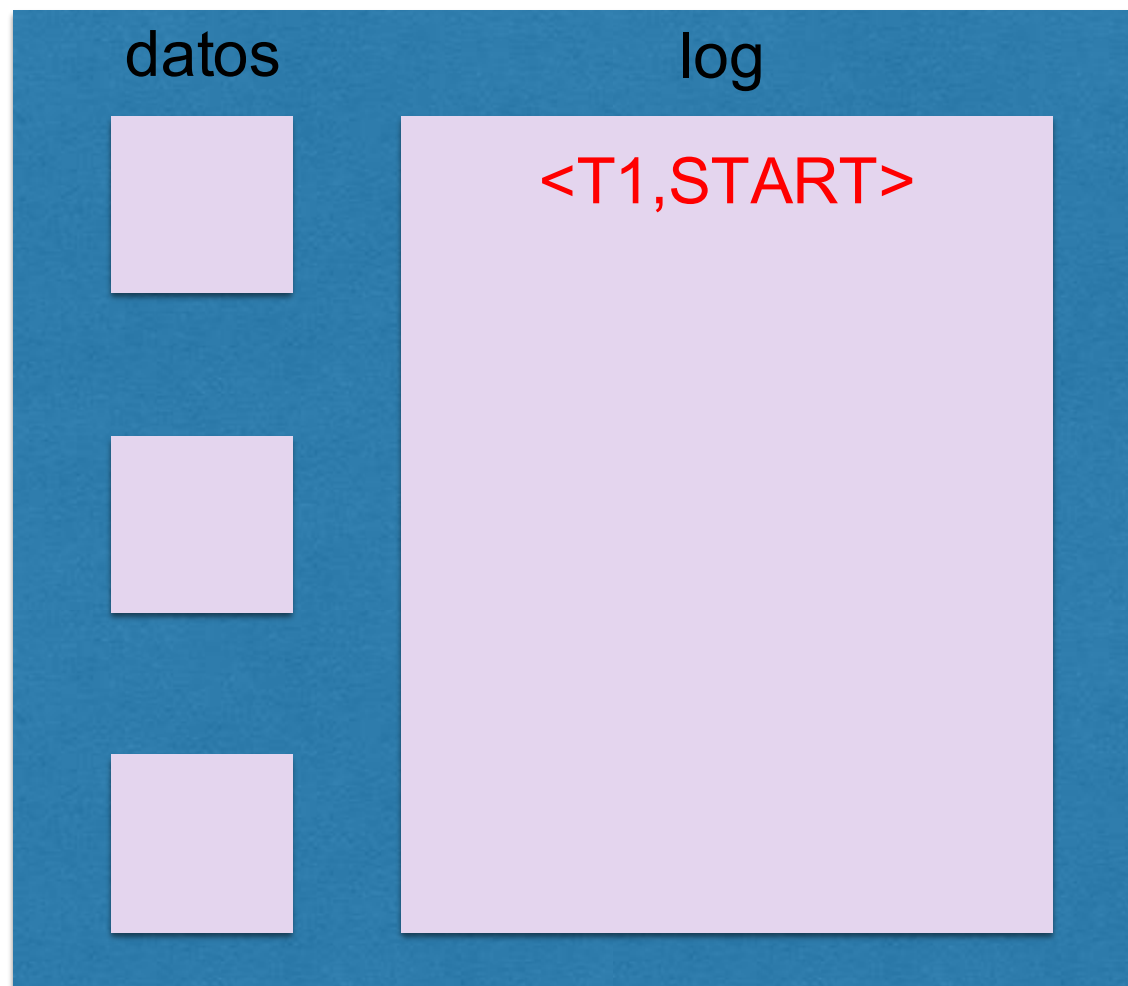


# Undo Logging – en la BD

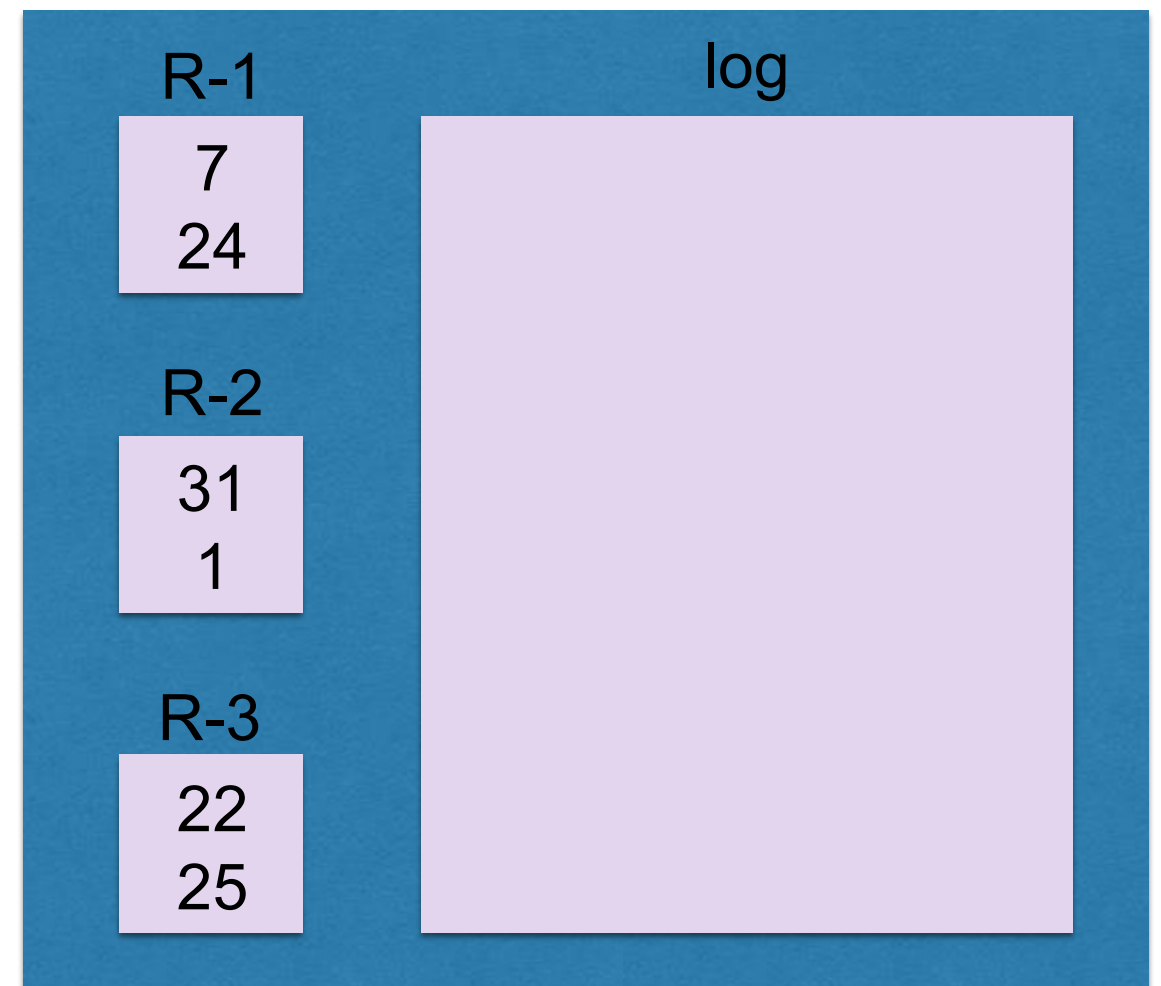
DBMS

T1: voy a empezar

Buffer



Disco

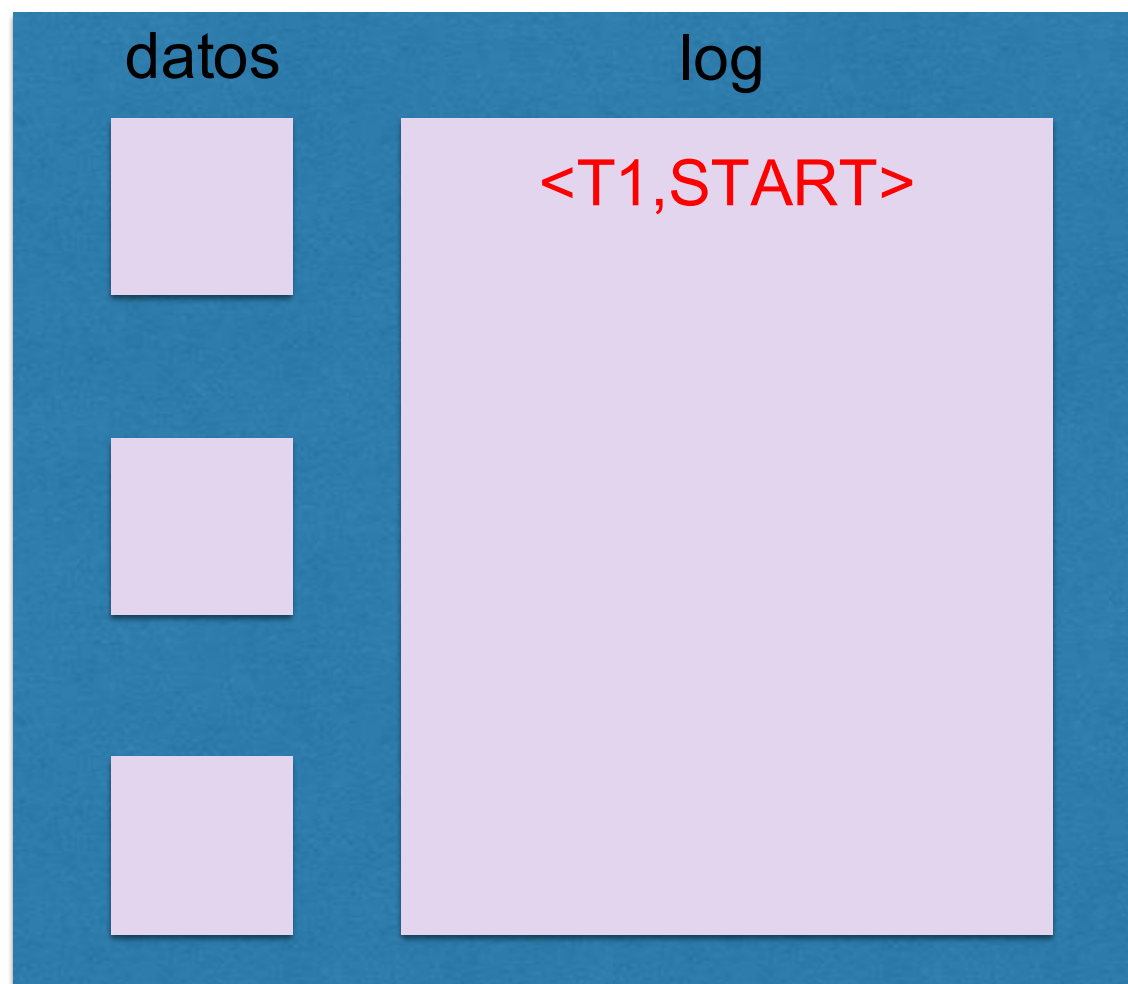


# Undo Logging – en la BD

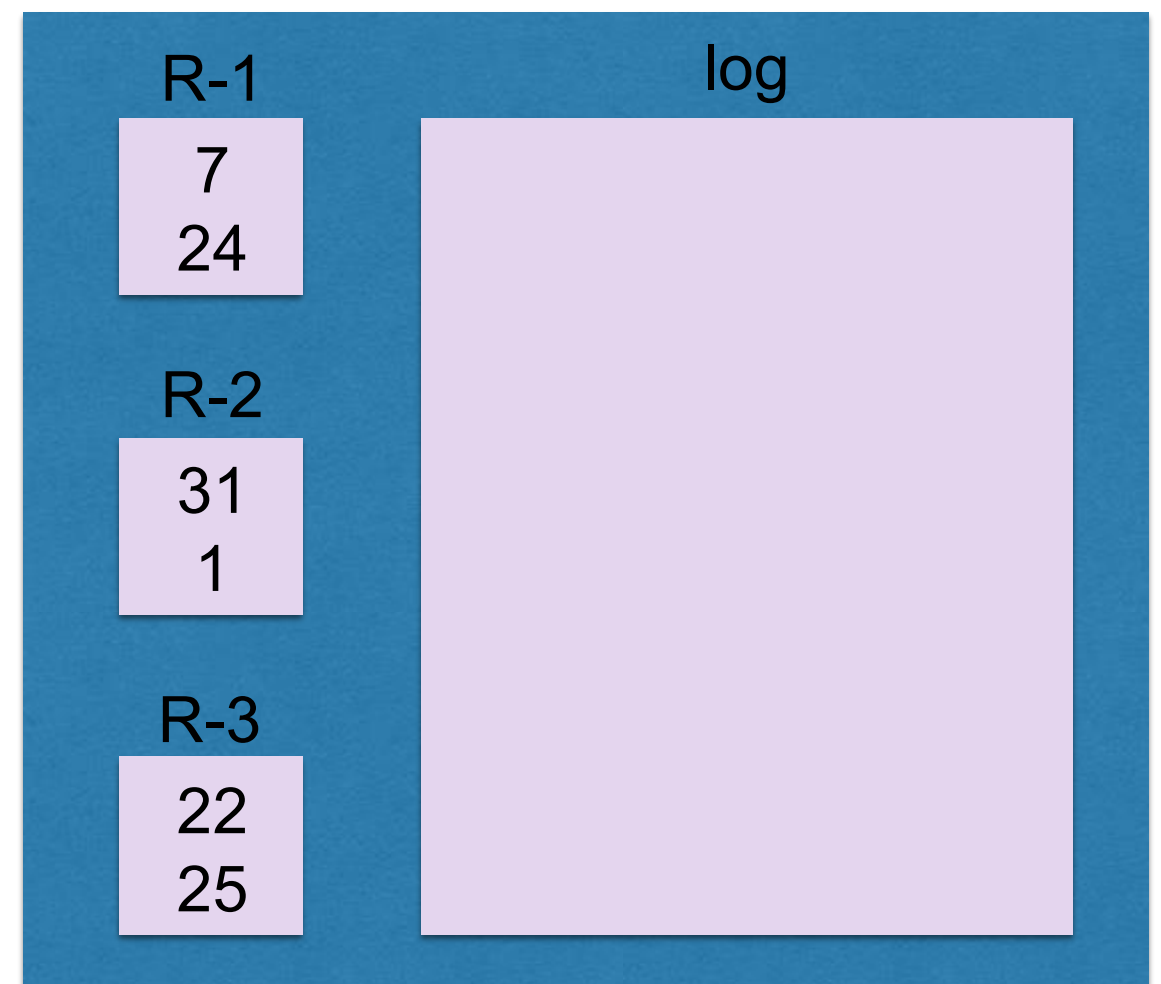
DBMS

T1: Necesito primera  
tupla de página 2 de R

Buffer



Disco

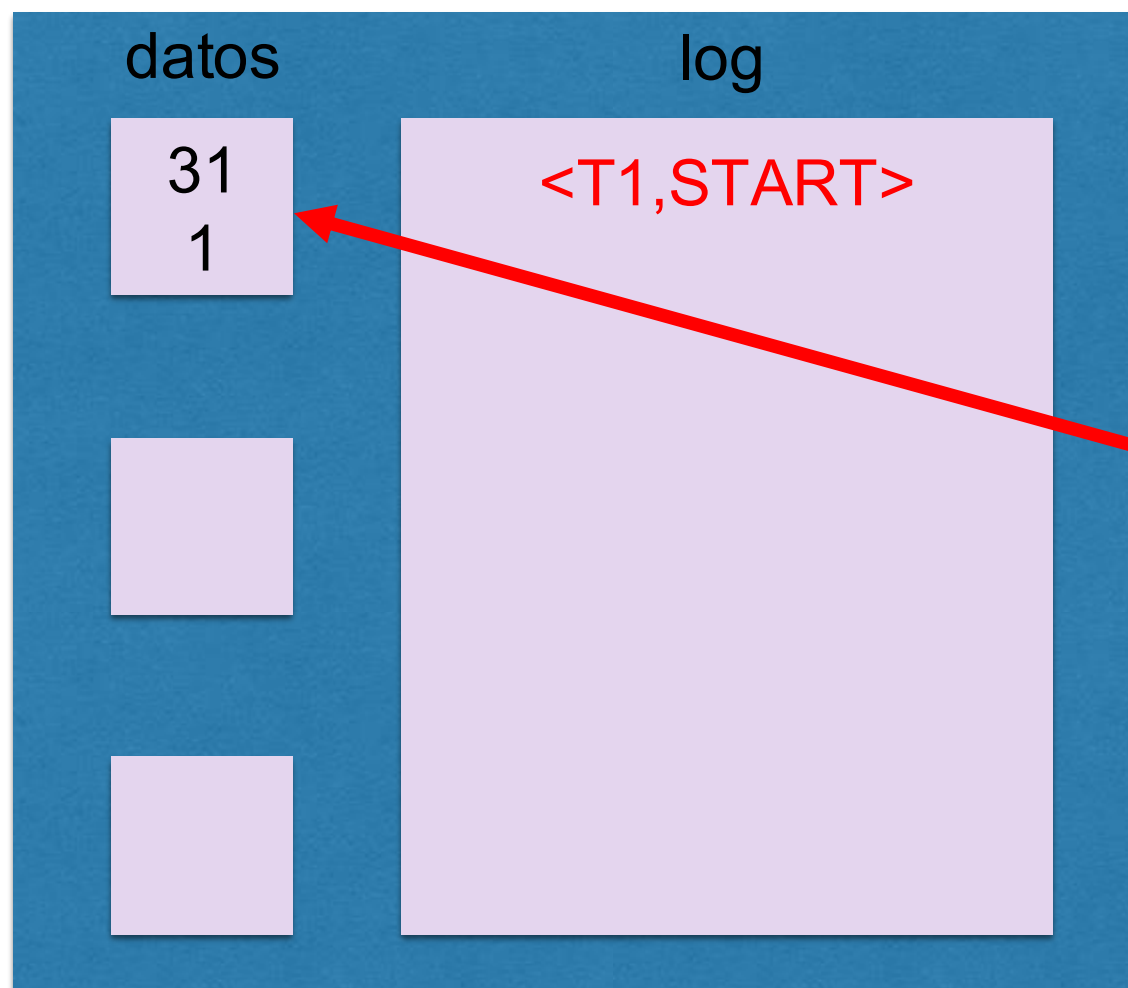


# Undo Logging – en la BD

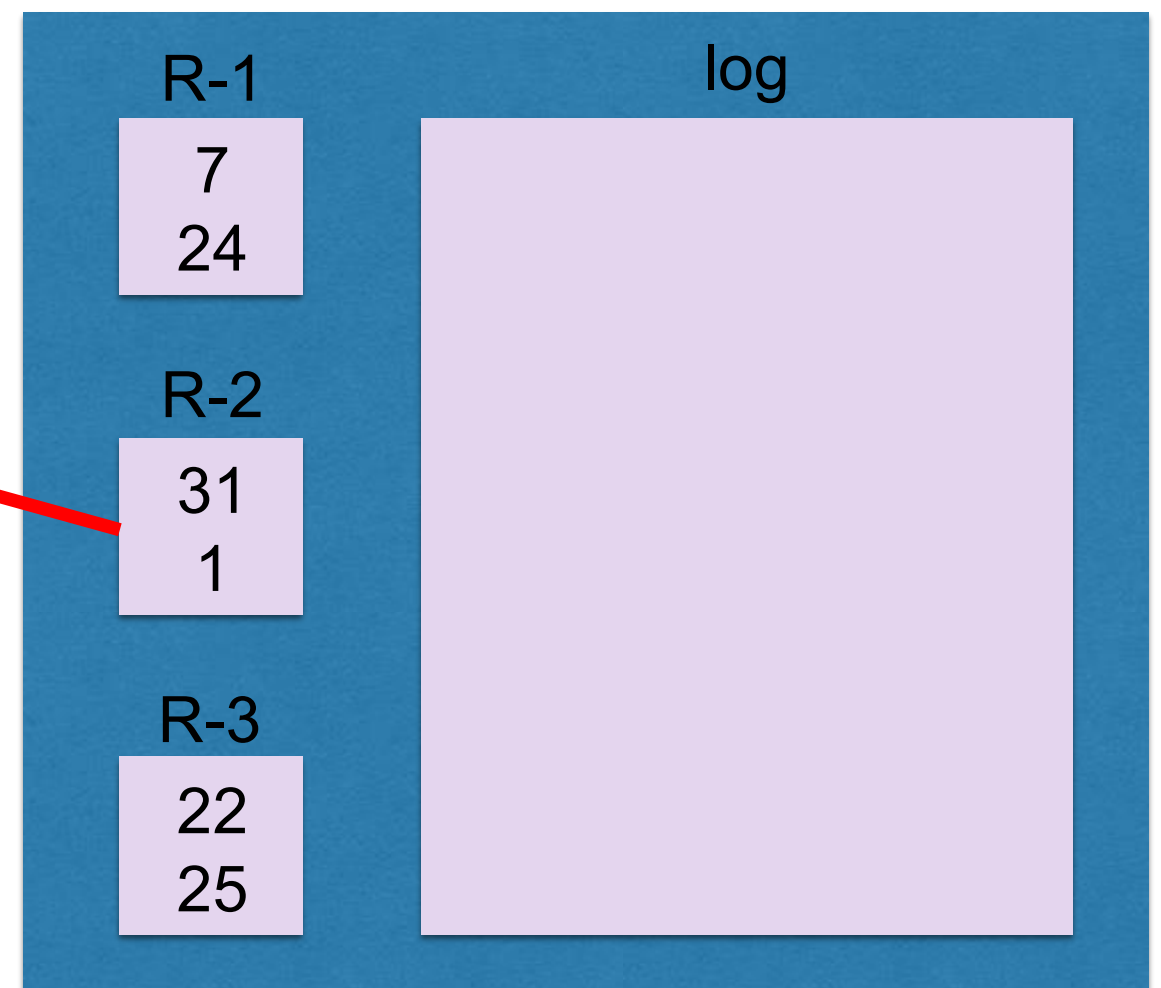
DBMS

T1: Necesito primera  
tupla de página 2 de R!

Buffer



Disco

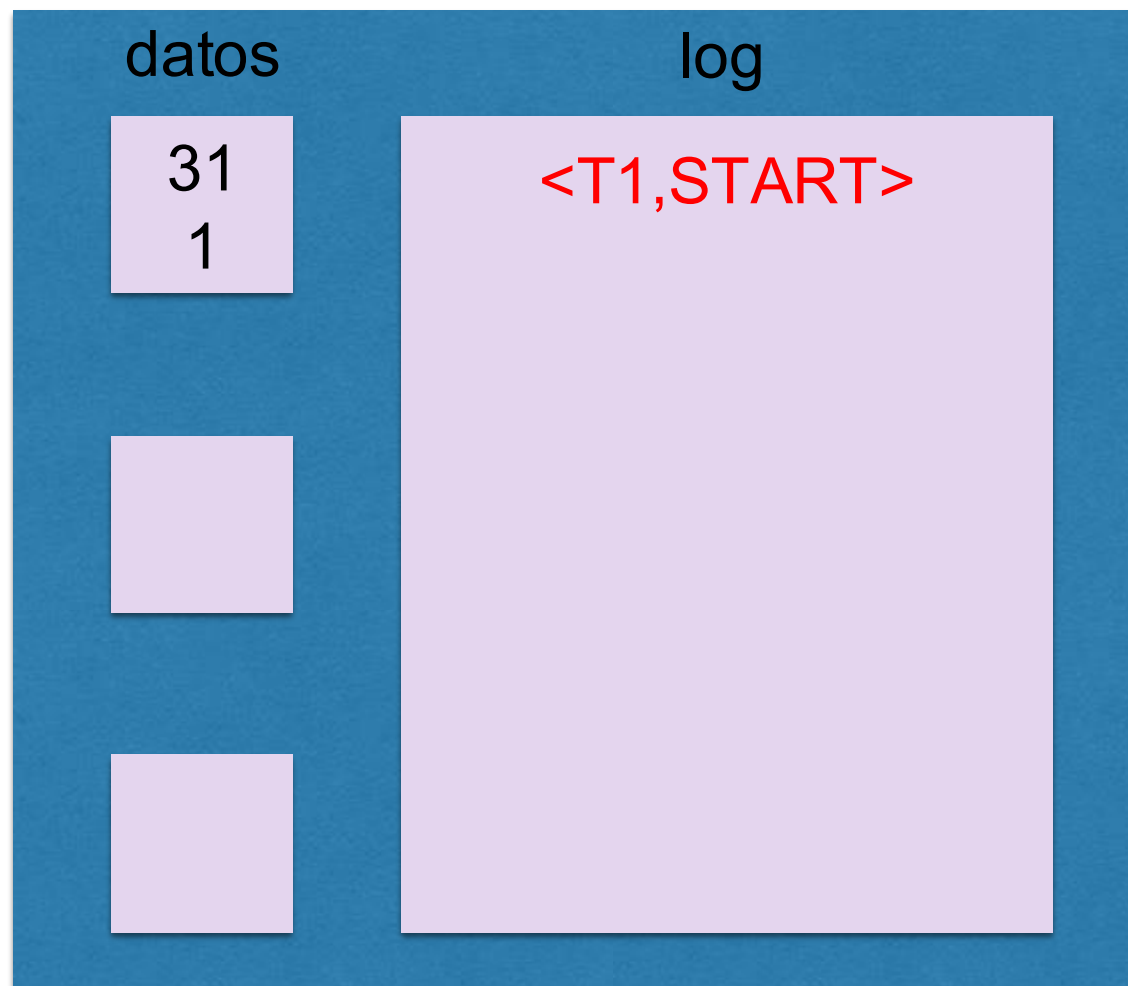


# Undo Logging – en la BD

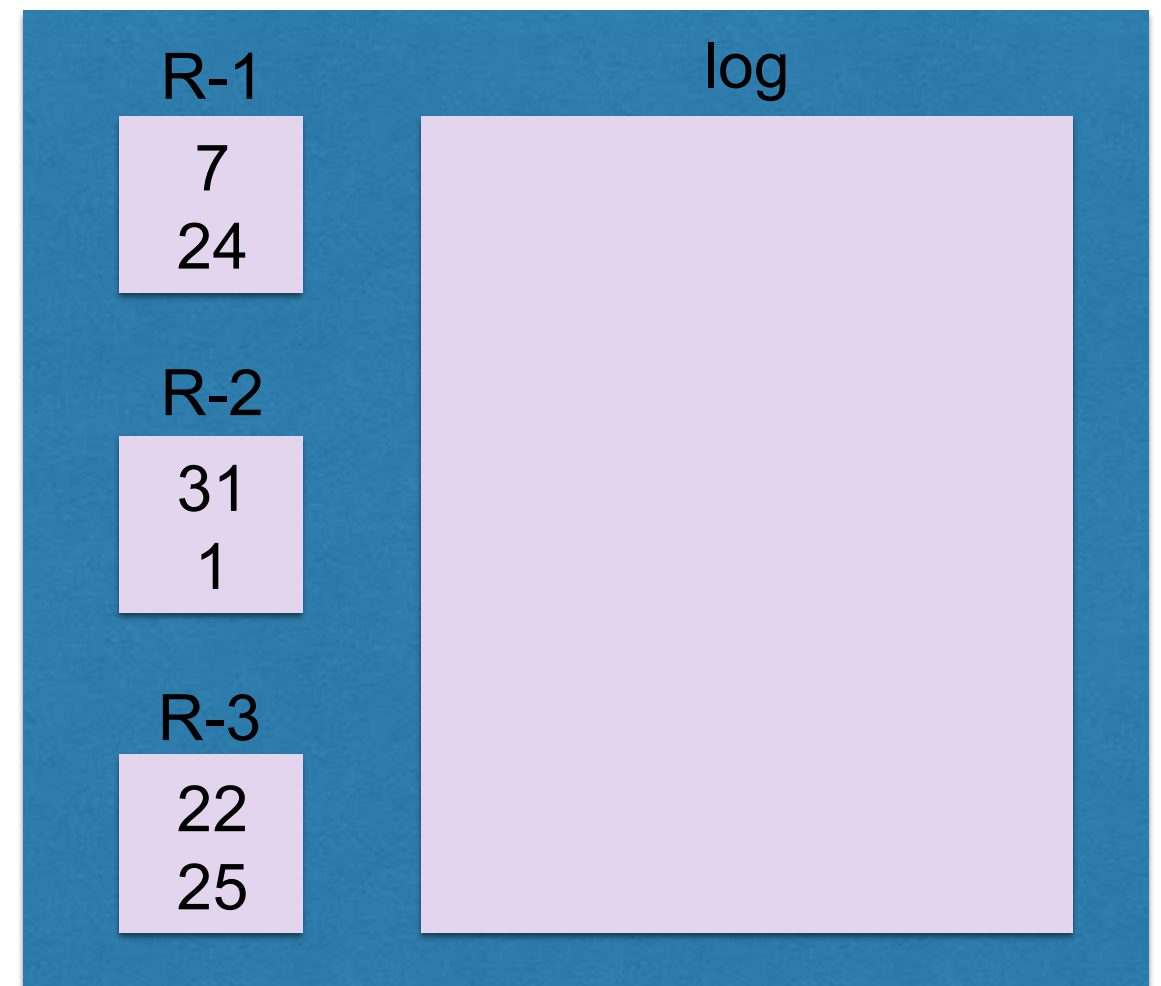
DBMS

T1: Cambio 31 a 99

Buffer



Disco

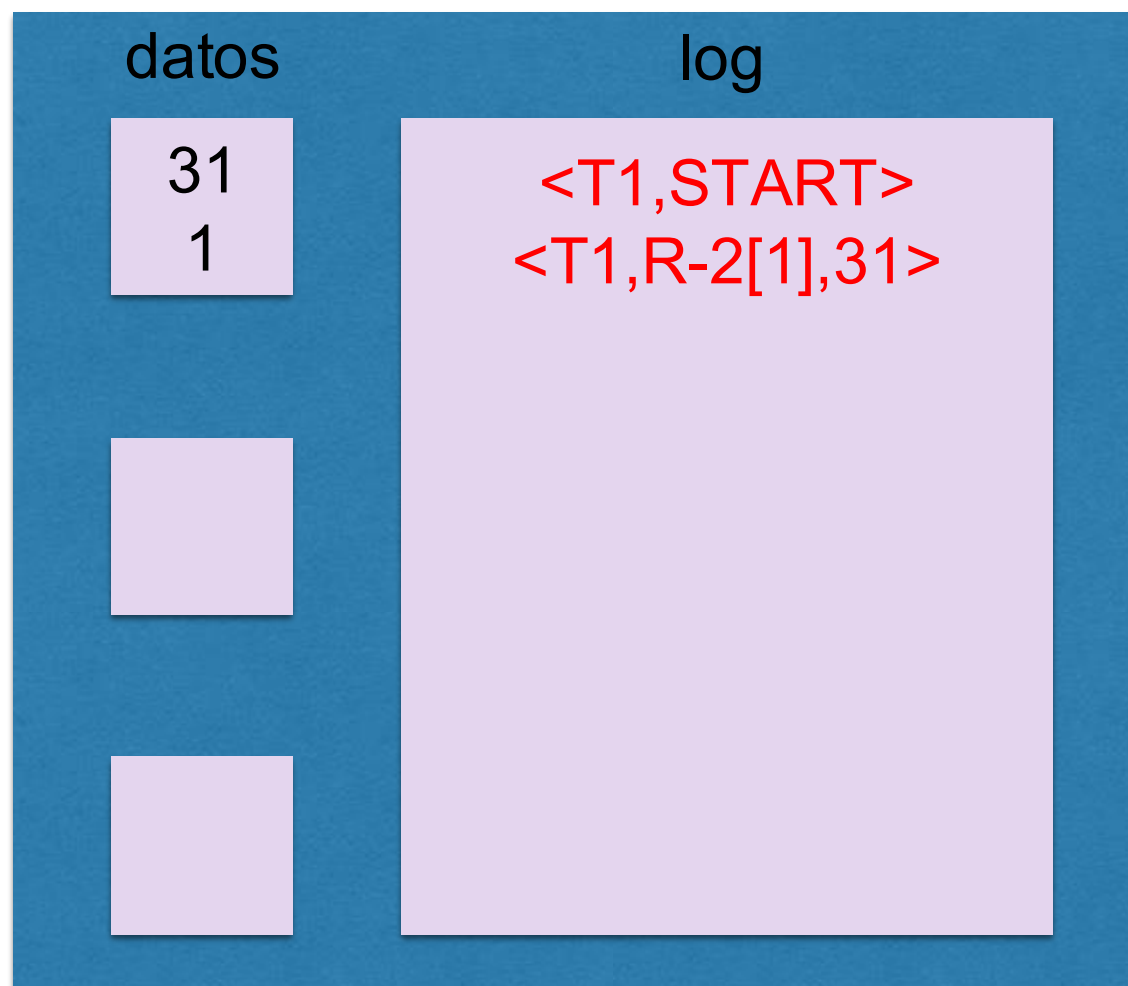


# Undo Logging – en la BD

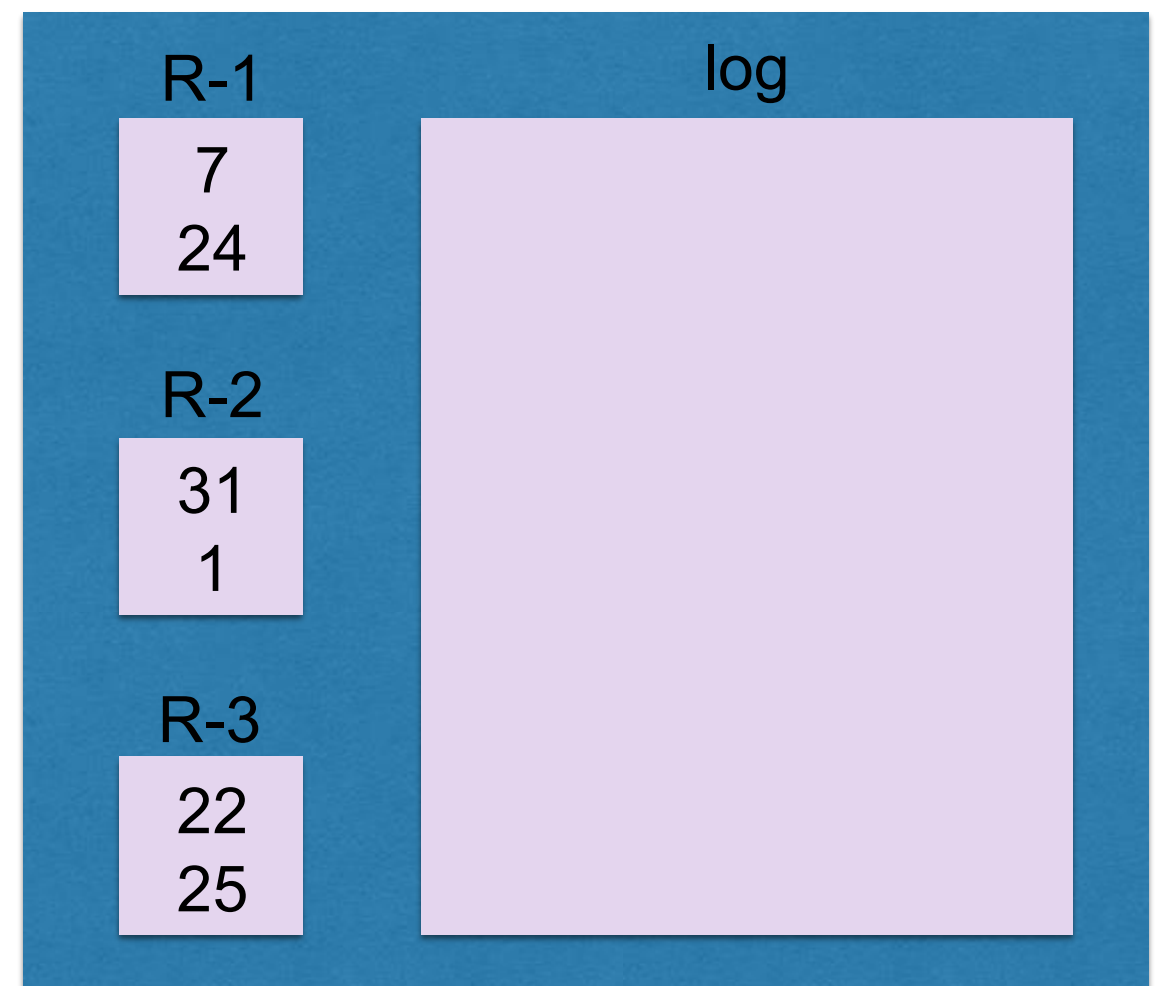
DBMS

T1: Cambio 31 a 99

Buffer



Disco

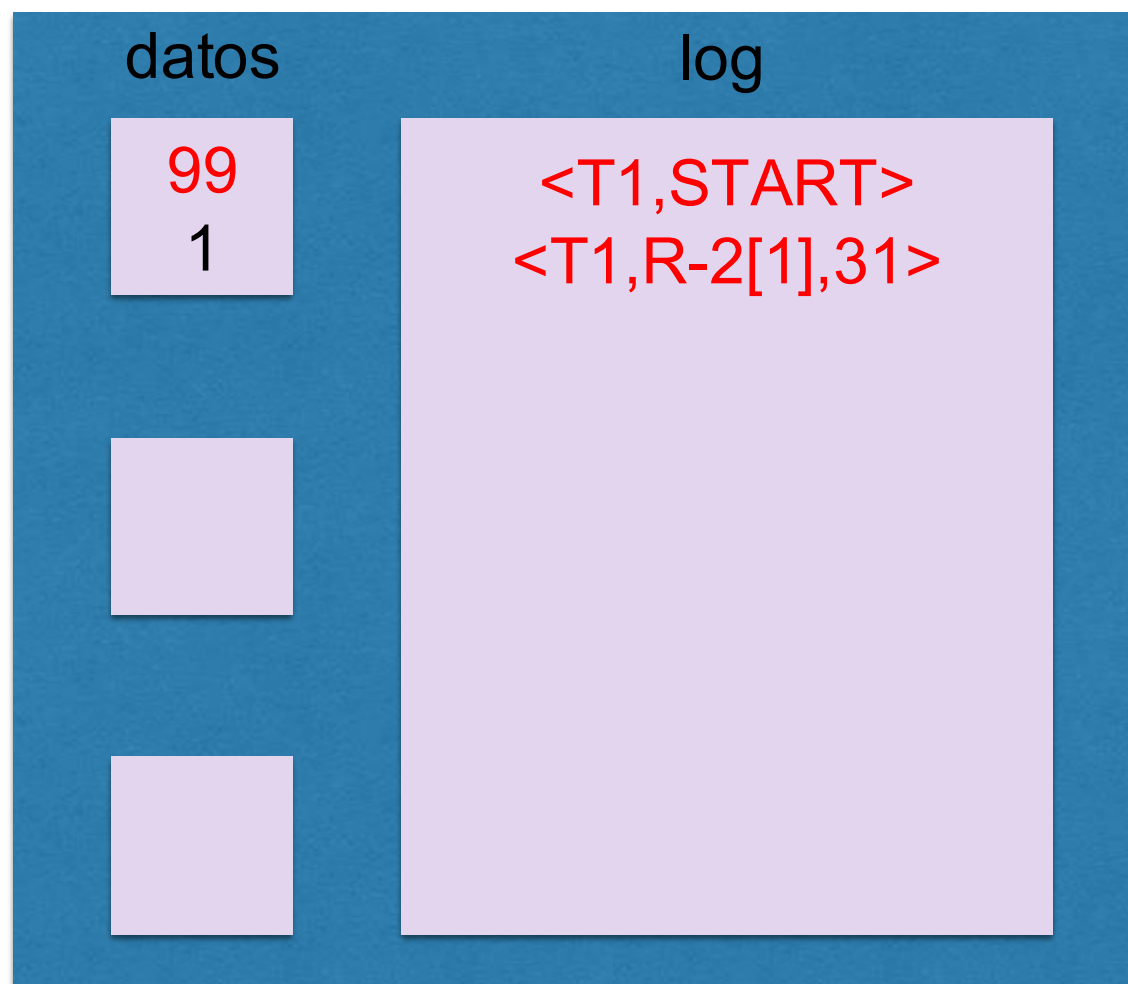


# Undo Logging – en la BD

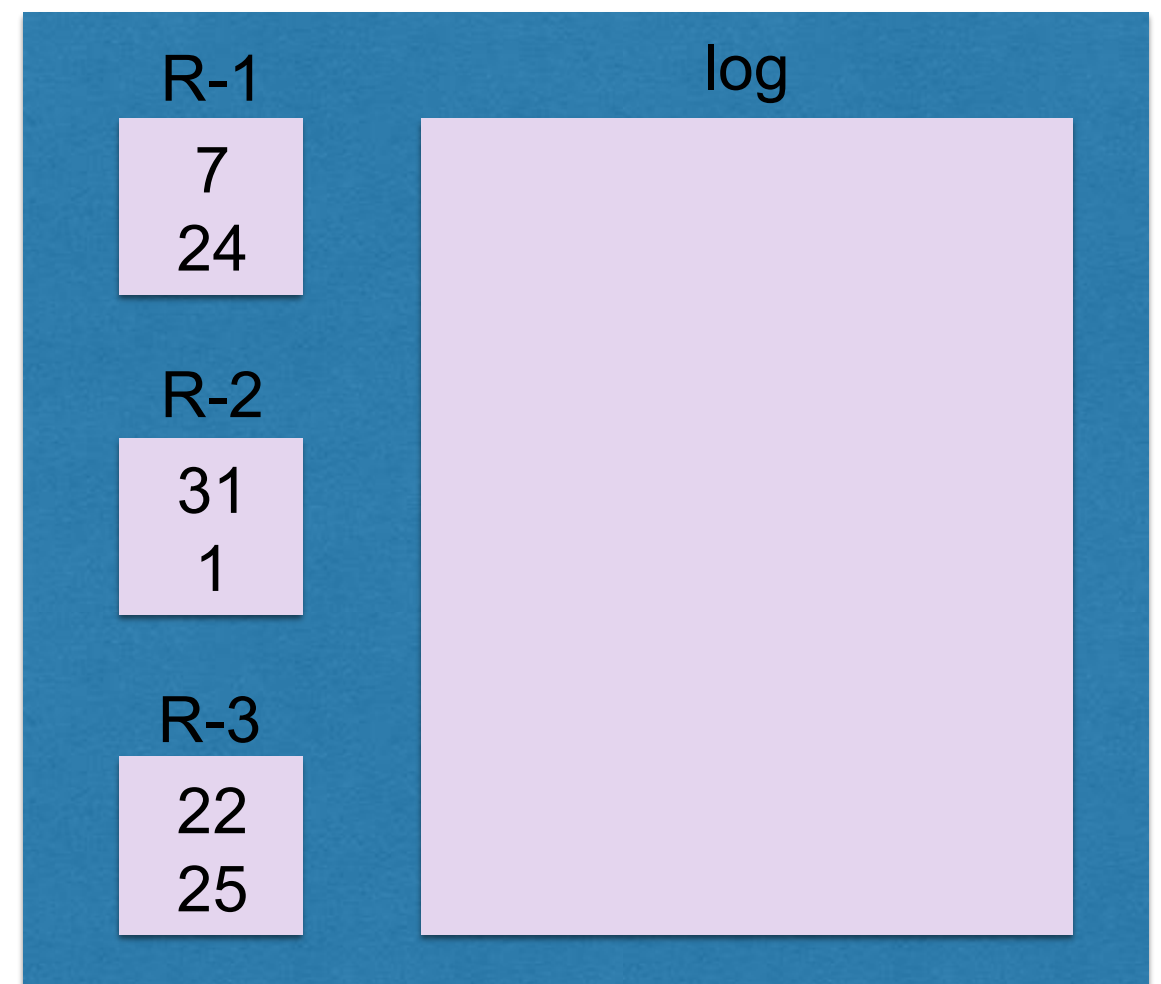
DBMS

T1: Cambio 31 a 99

Buffer



Disco



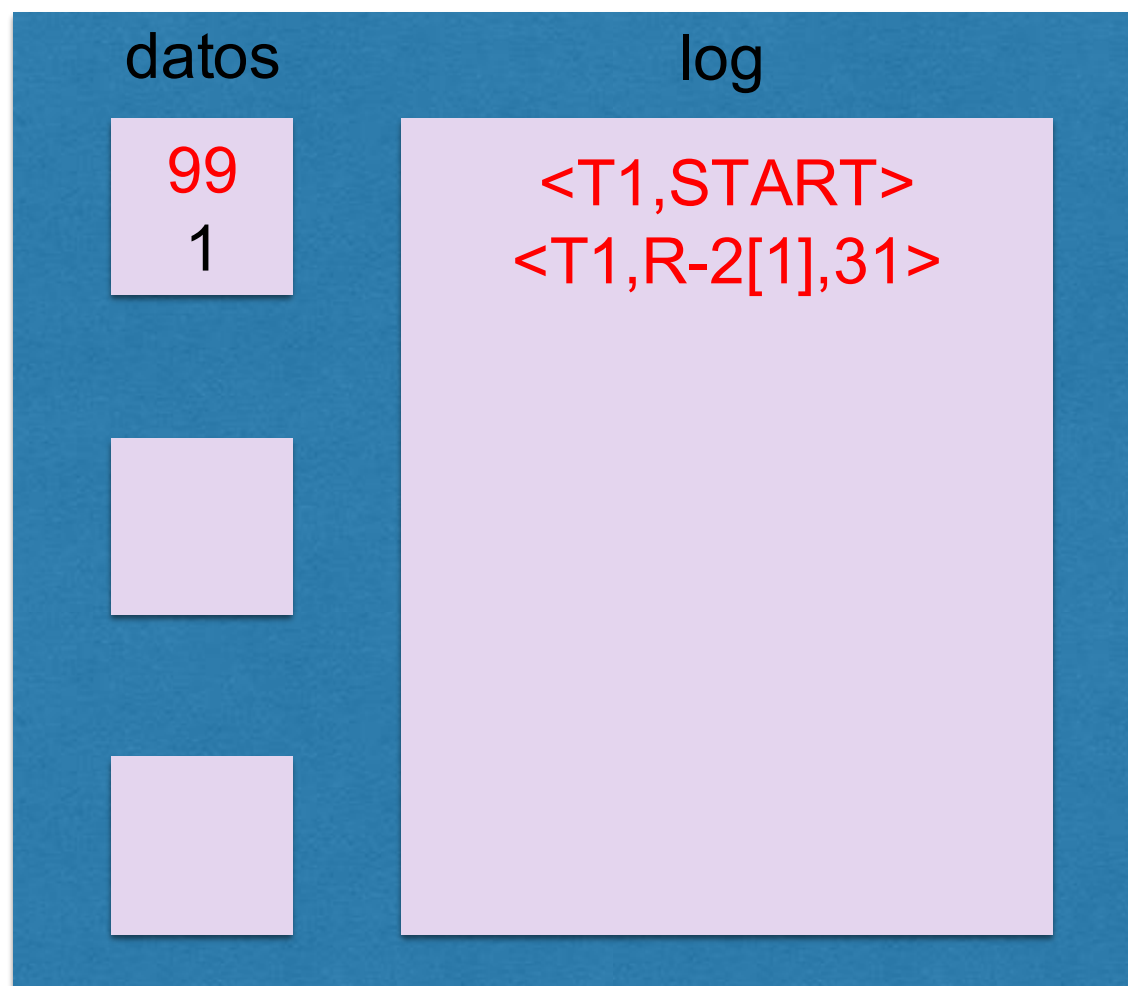


# Undo Logging – en la BD

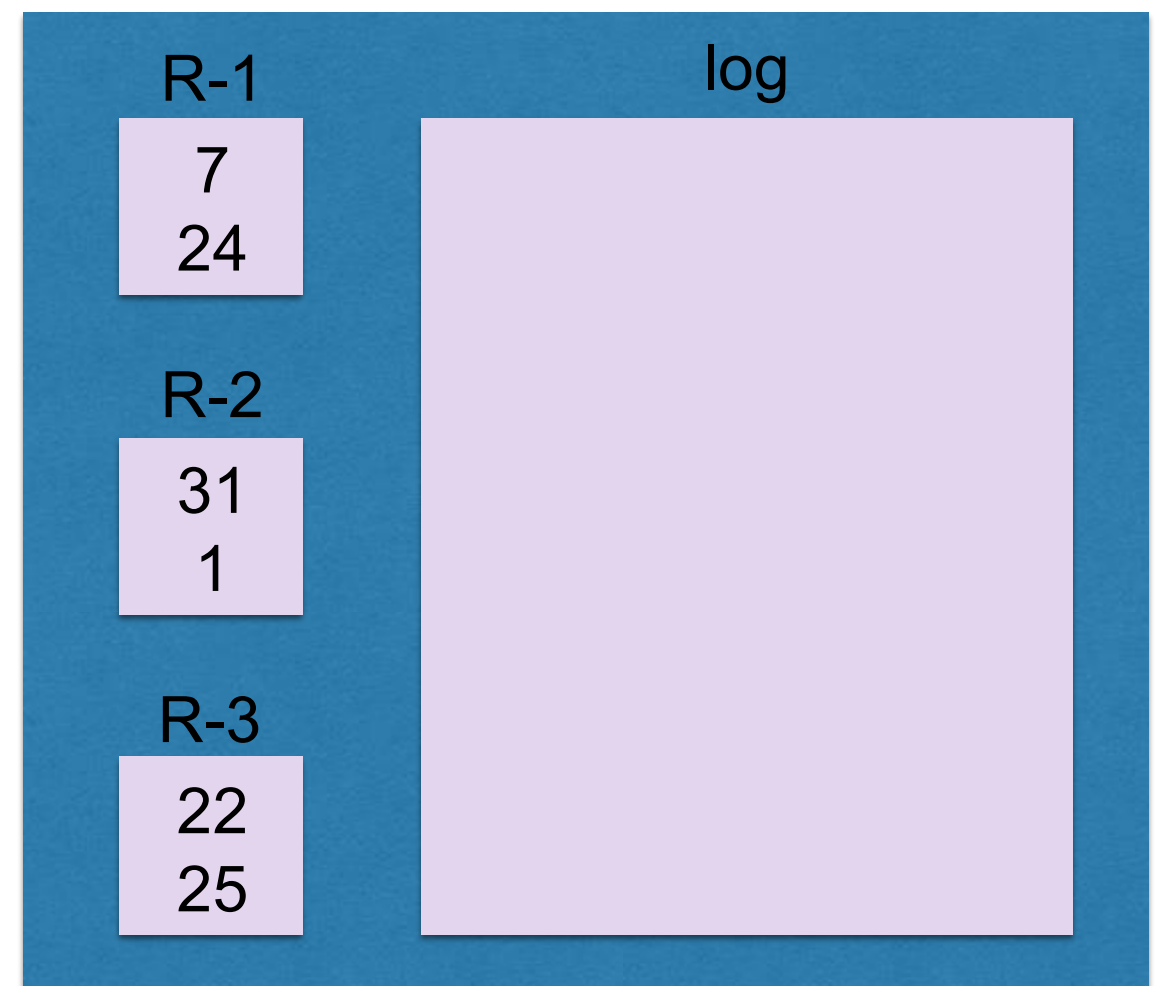
DBMS

T1: Cambio 99 a 23

Buffer



Disco

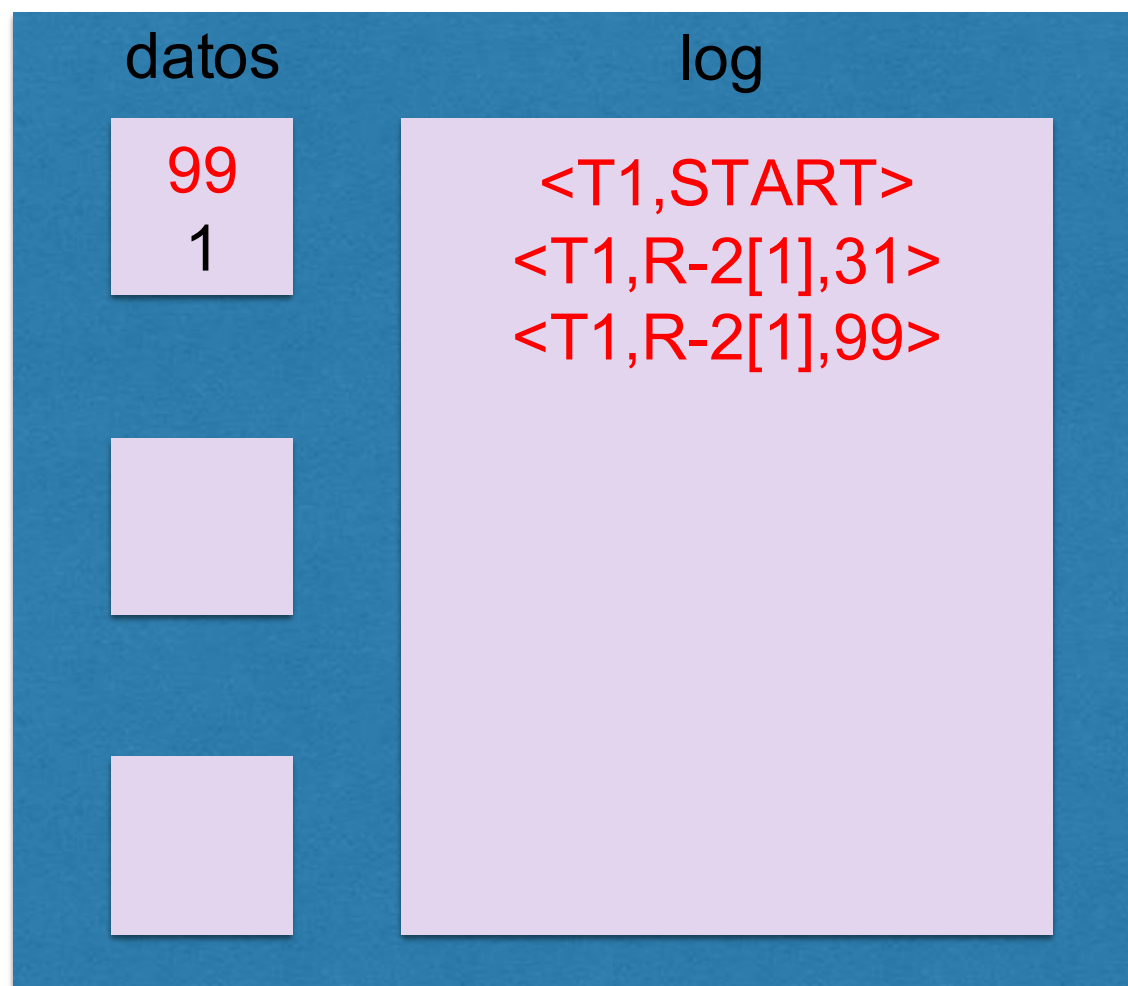


# Undo Logging – en la BD

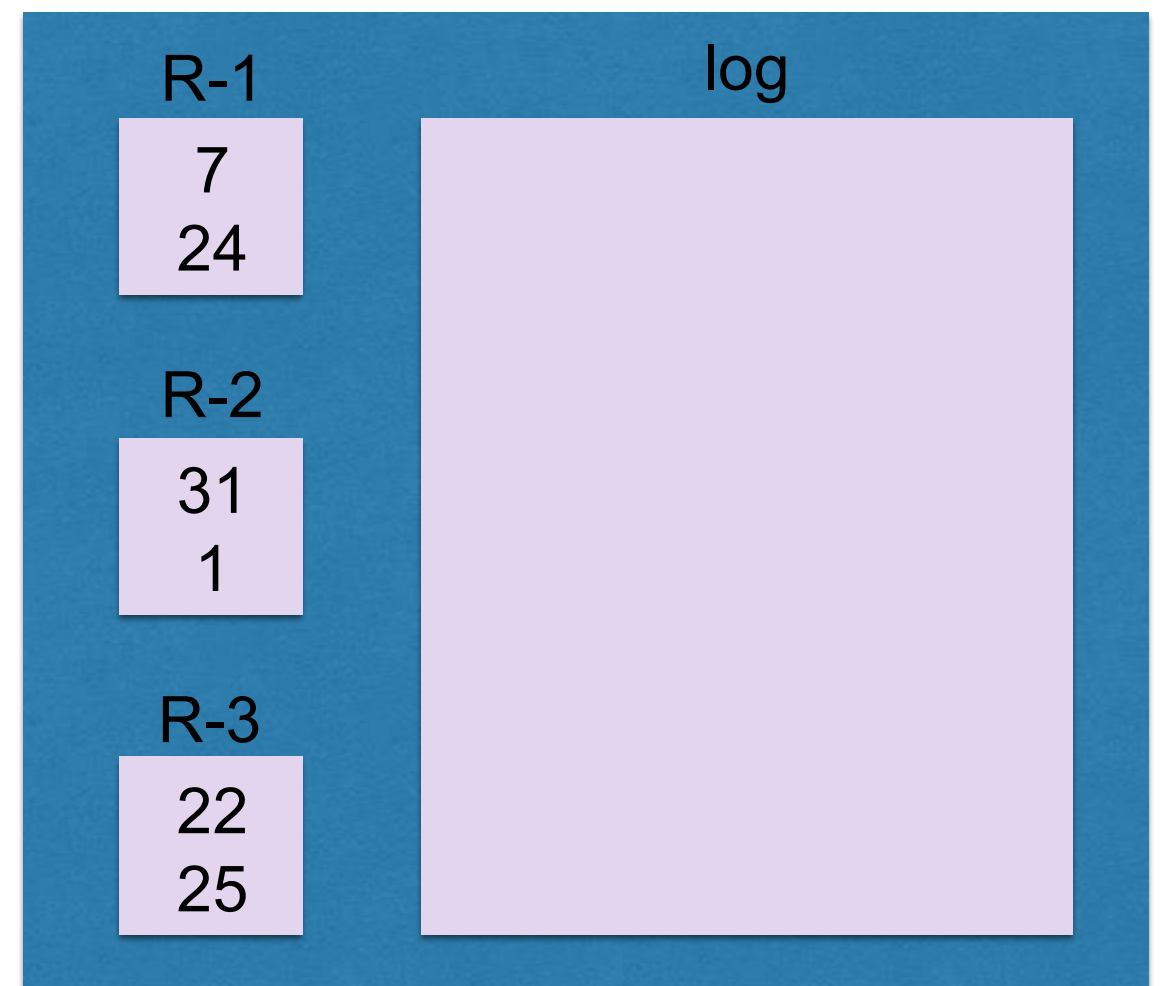
DBMS

T1: Cambio 99 a 23

Buffer



Disco

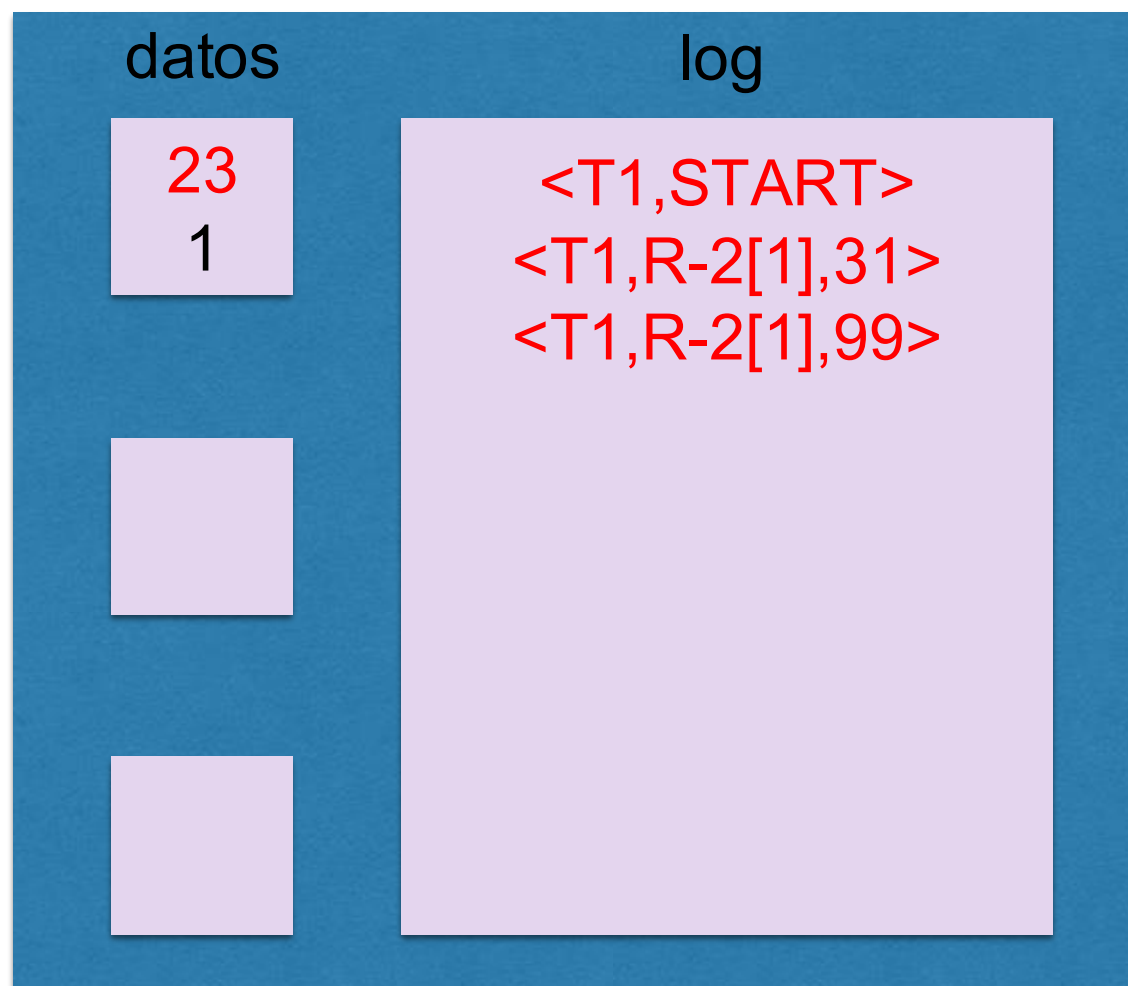


# Undo Logging – en la BD

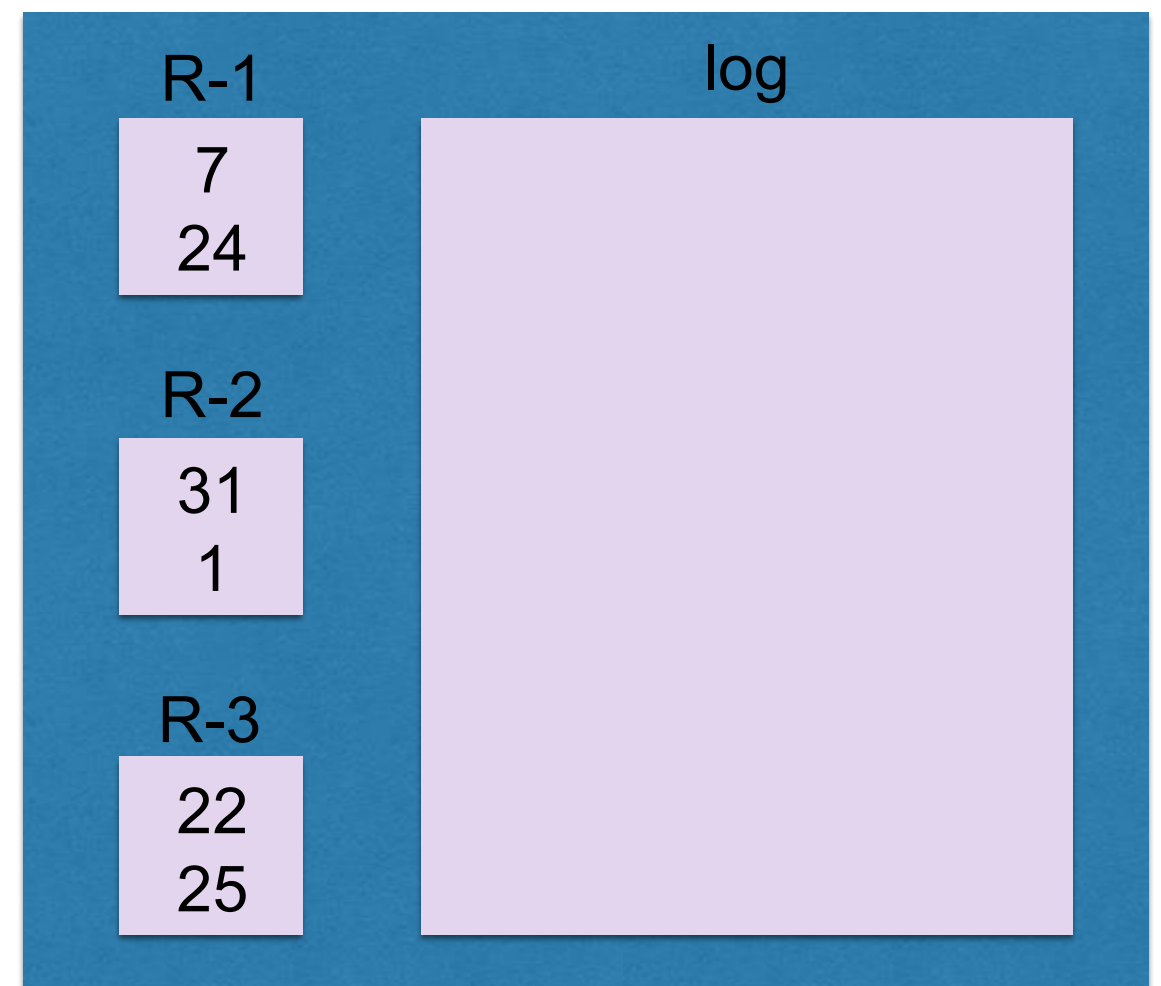
DBMS

T1: Cambio 99 a 23

Buffer



Disco

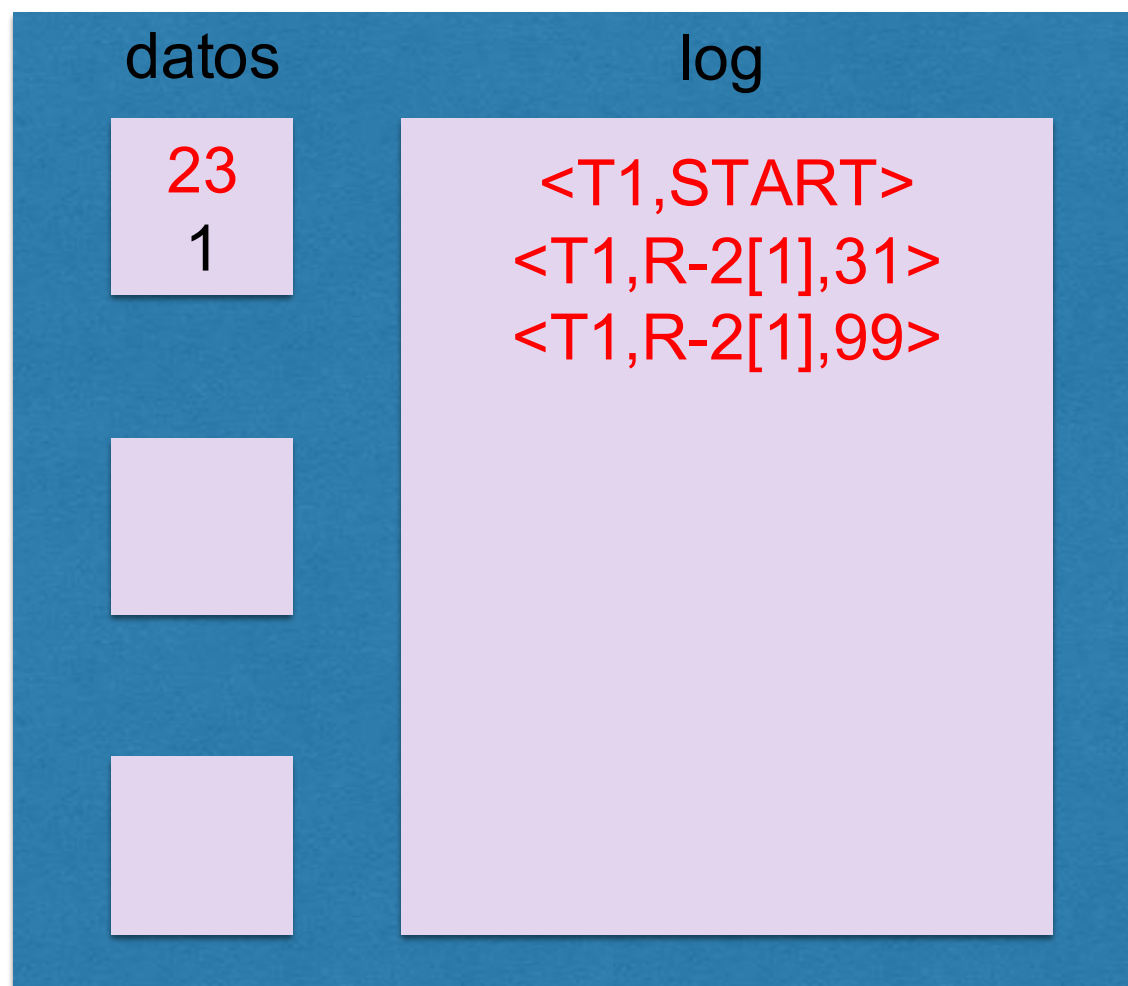


# Undo Logging – en la BD

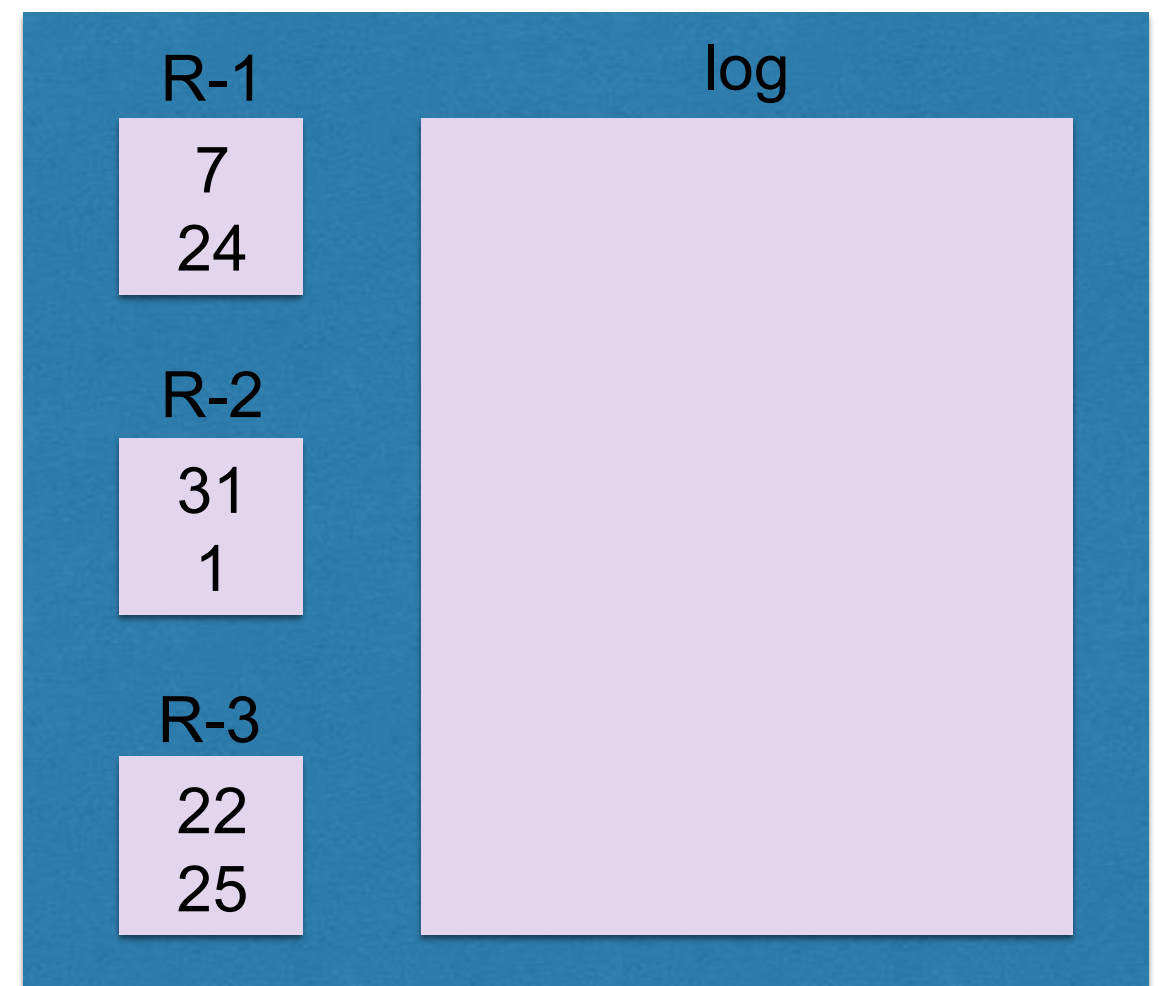
DBMS

T1: estoy listo

Buffer



Disco

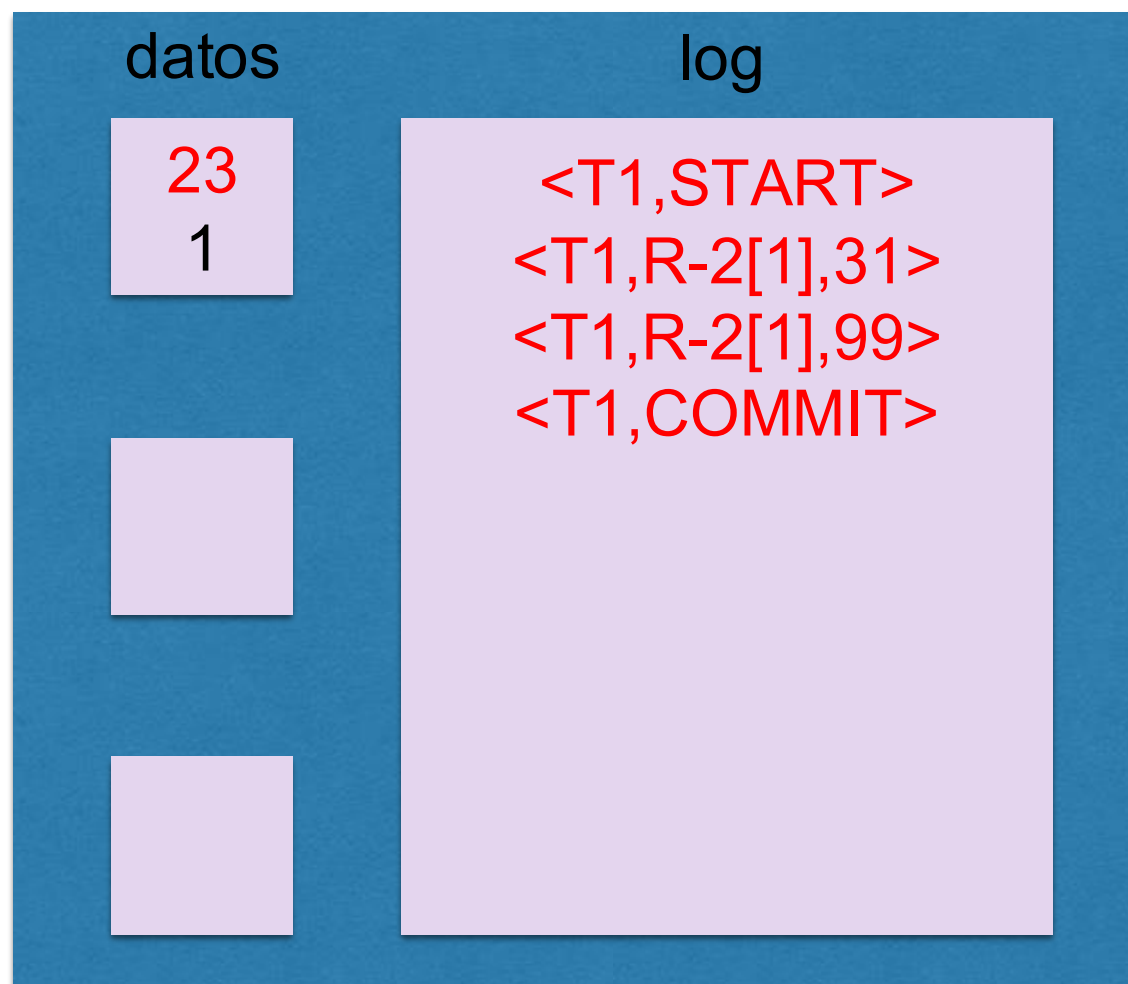


# Undo Logging – en la BD

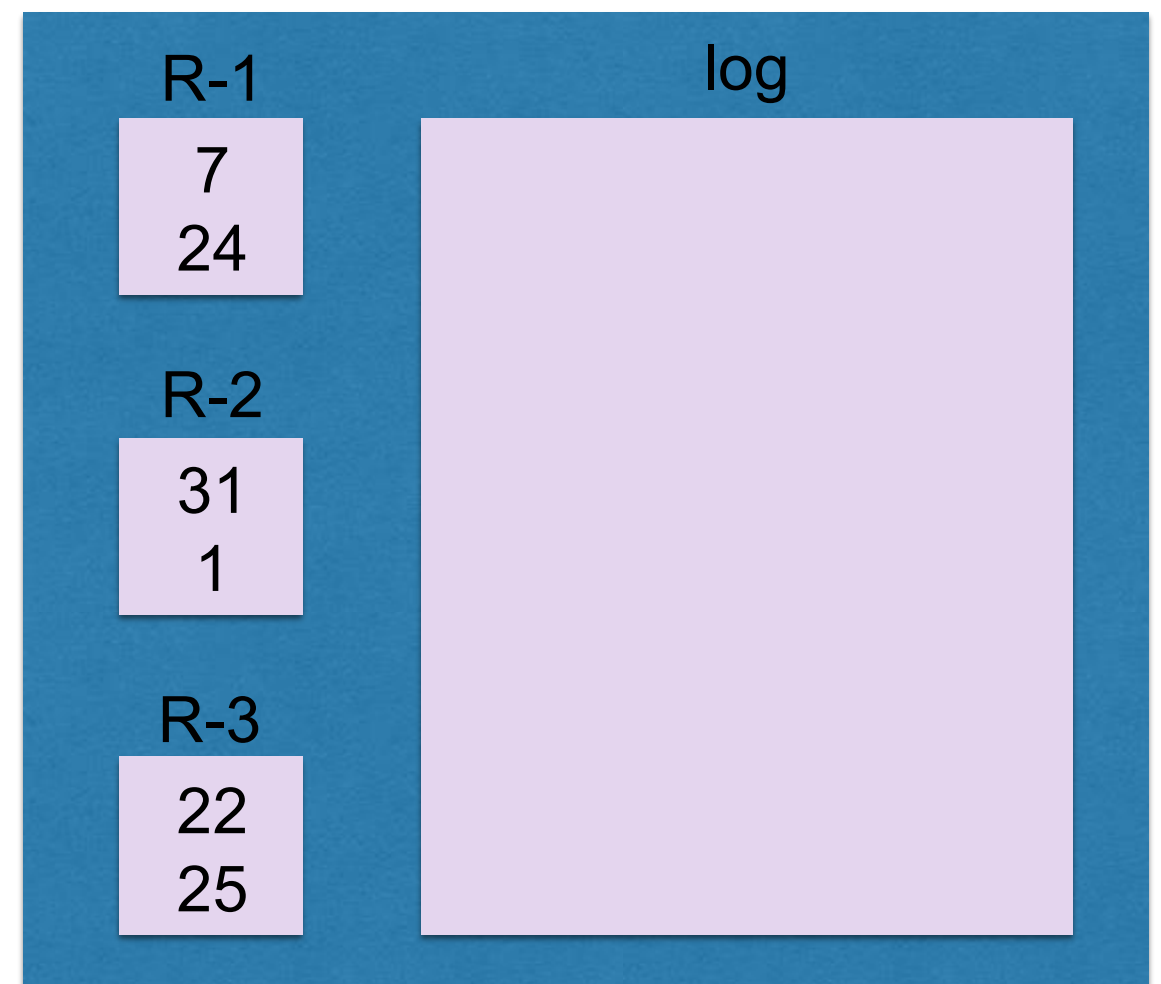
DBMS

T1: estoy listo

Buffer



Disco

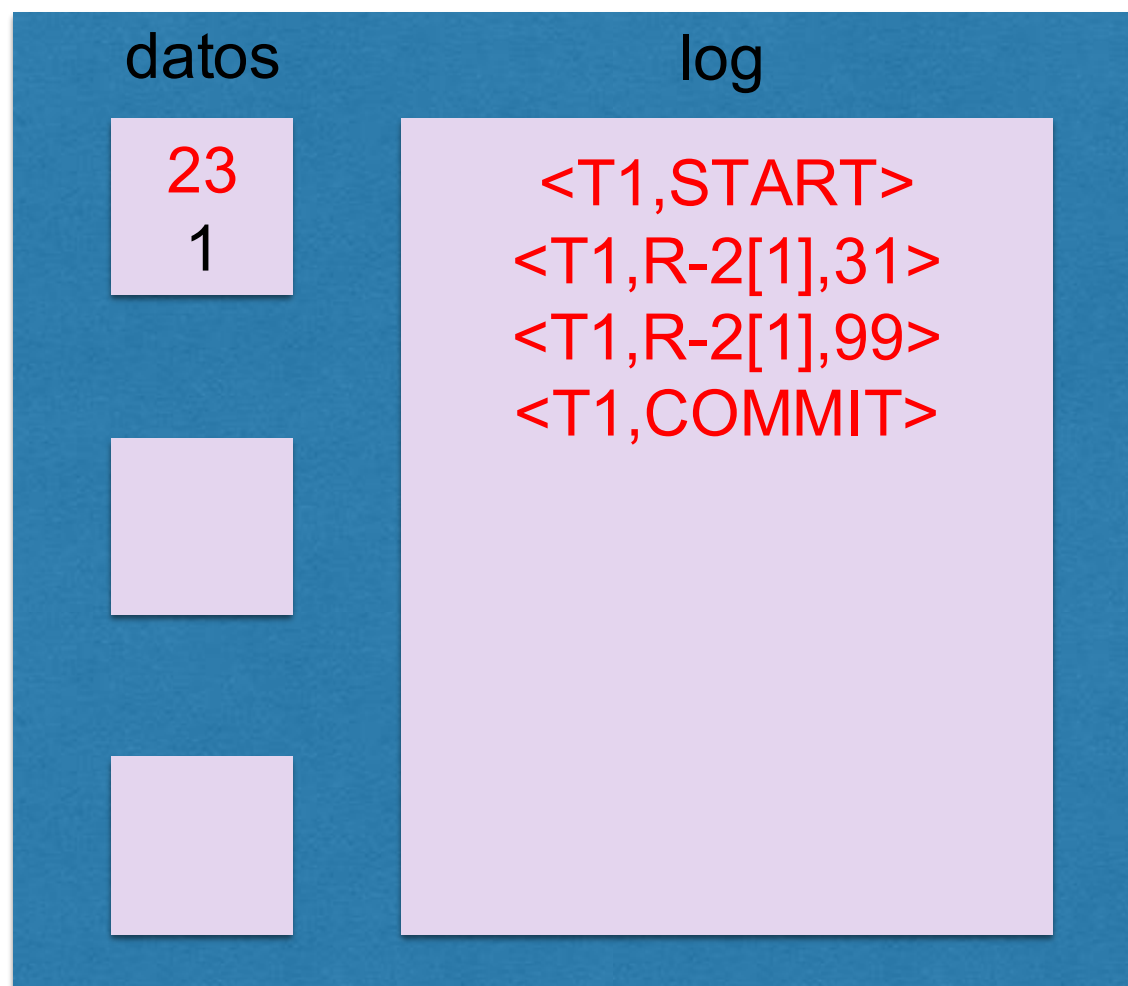


# Undo Logging – en la BD

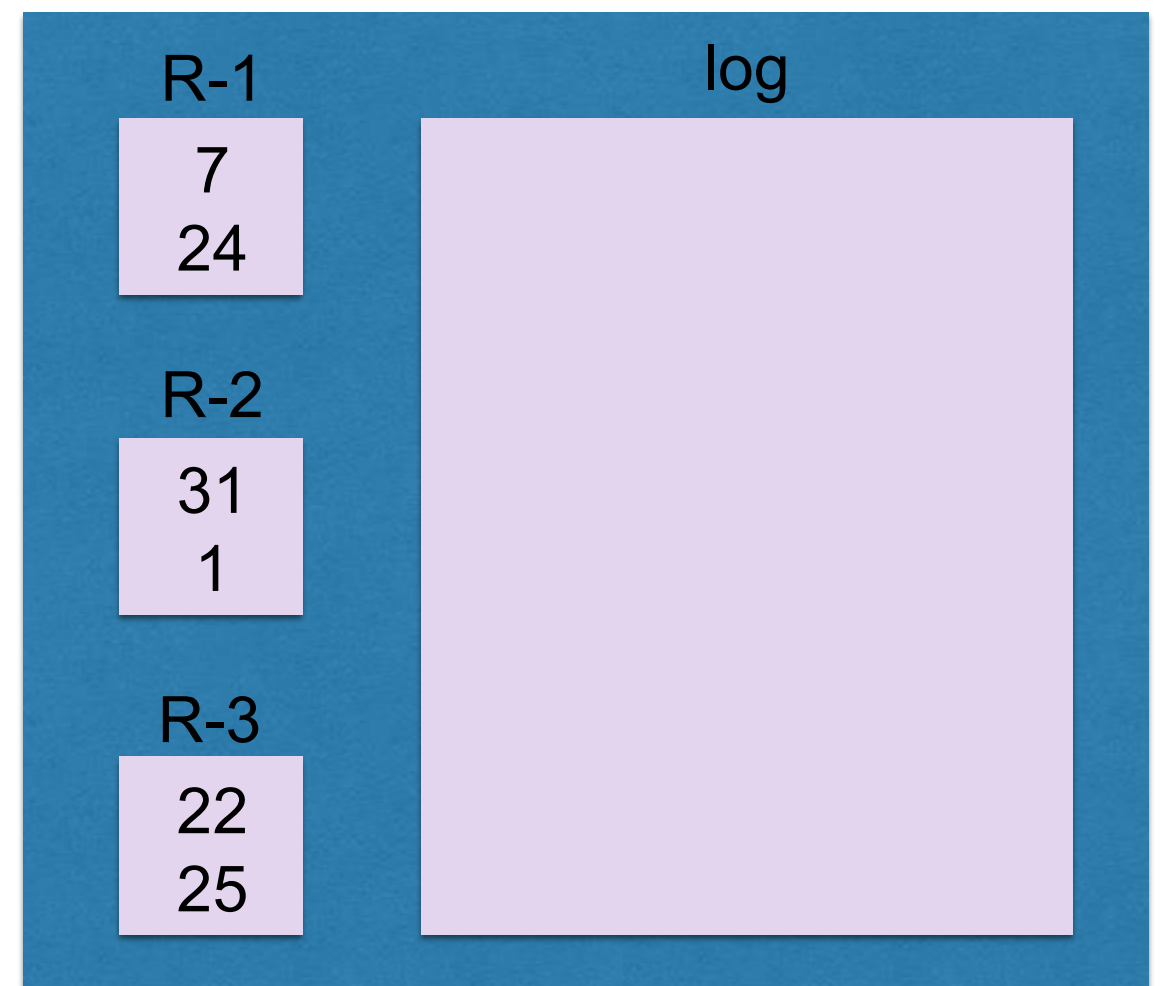
DBMS

Vamos al disco

Buffer



Disco

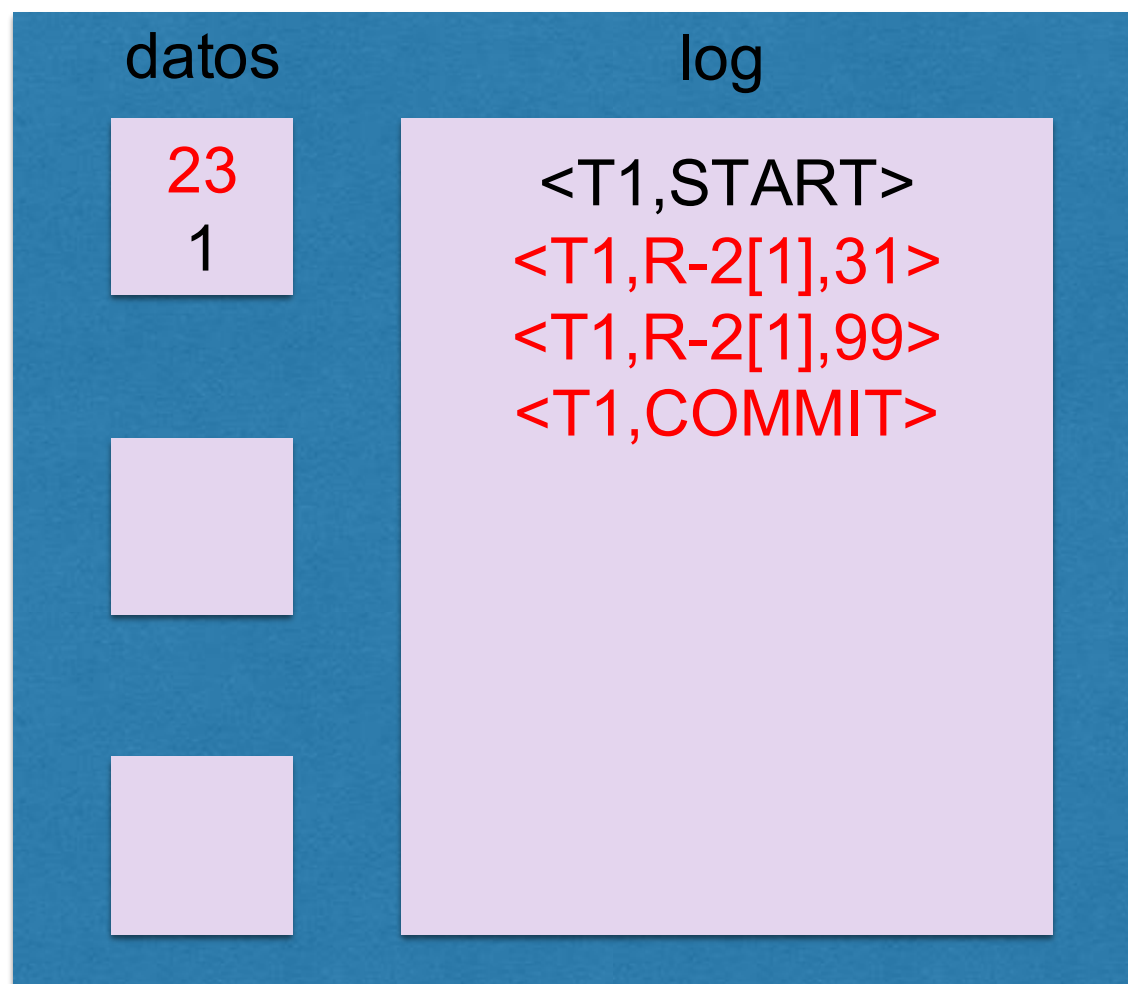


# Undo Logging – en la BD

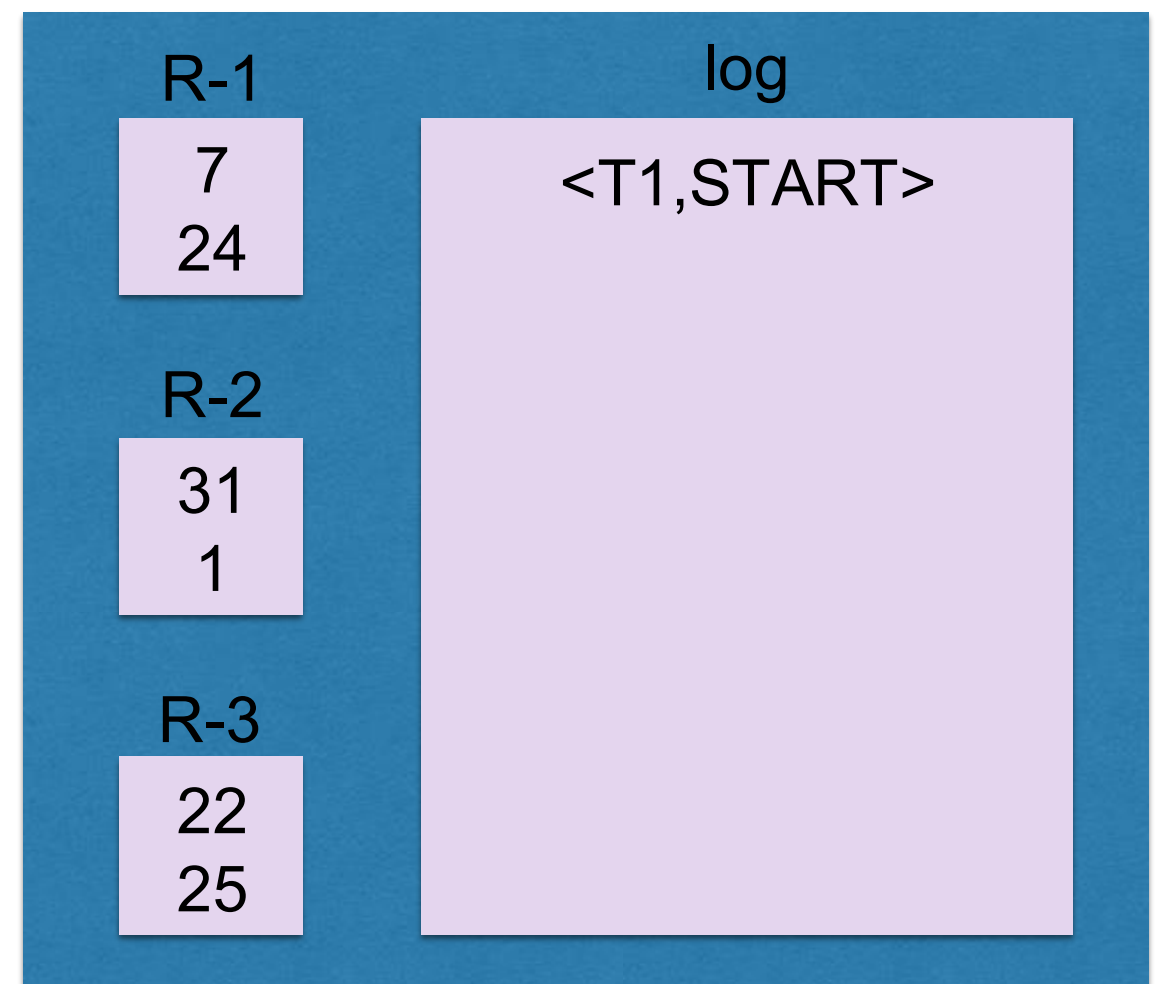
DBMS

Vamos al disco!

Buffer



Disco

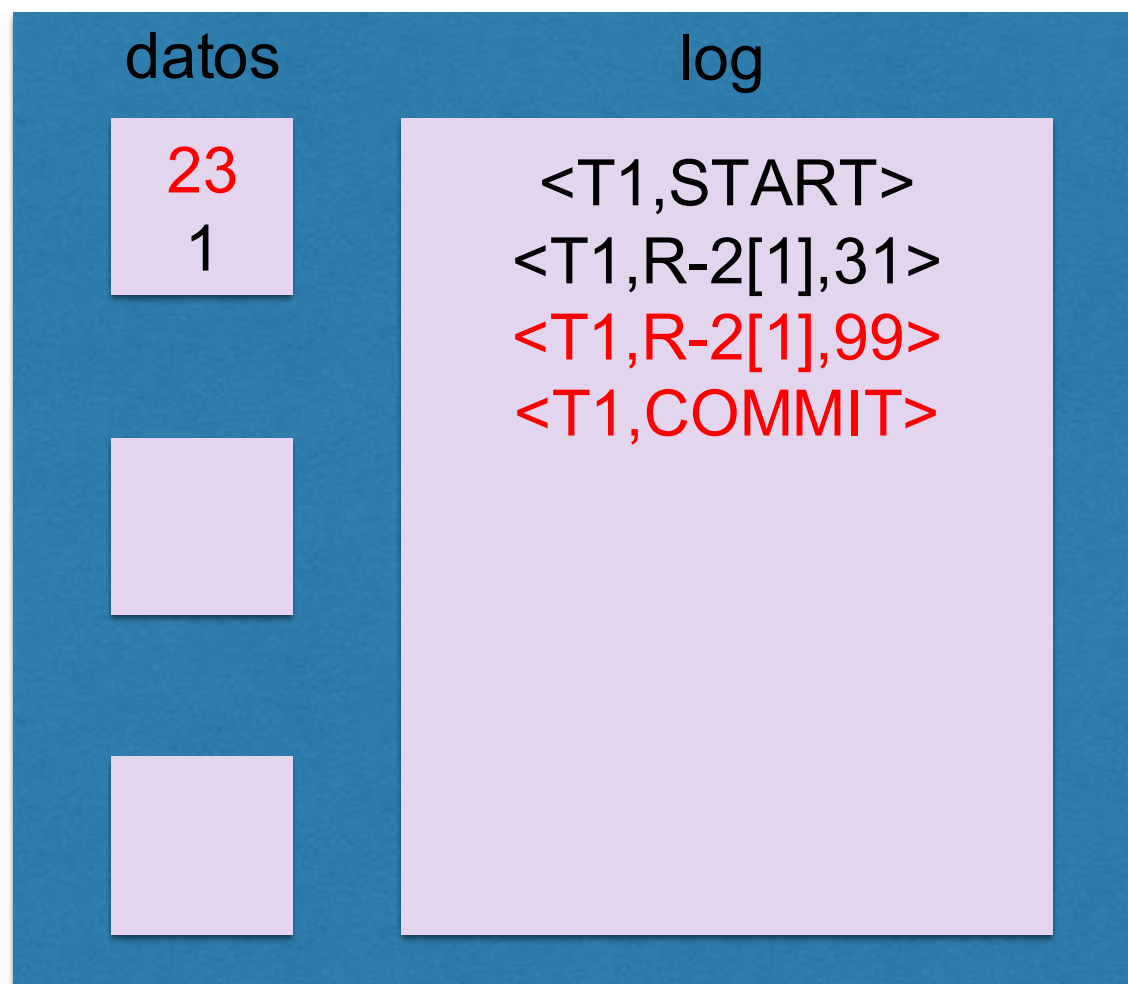


# Undo Logging – en la BD

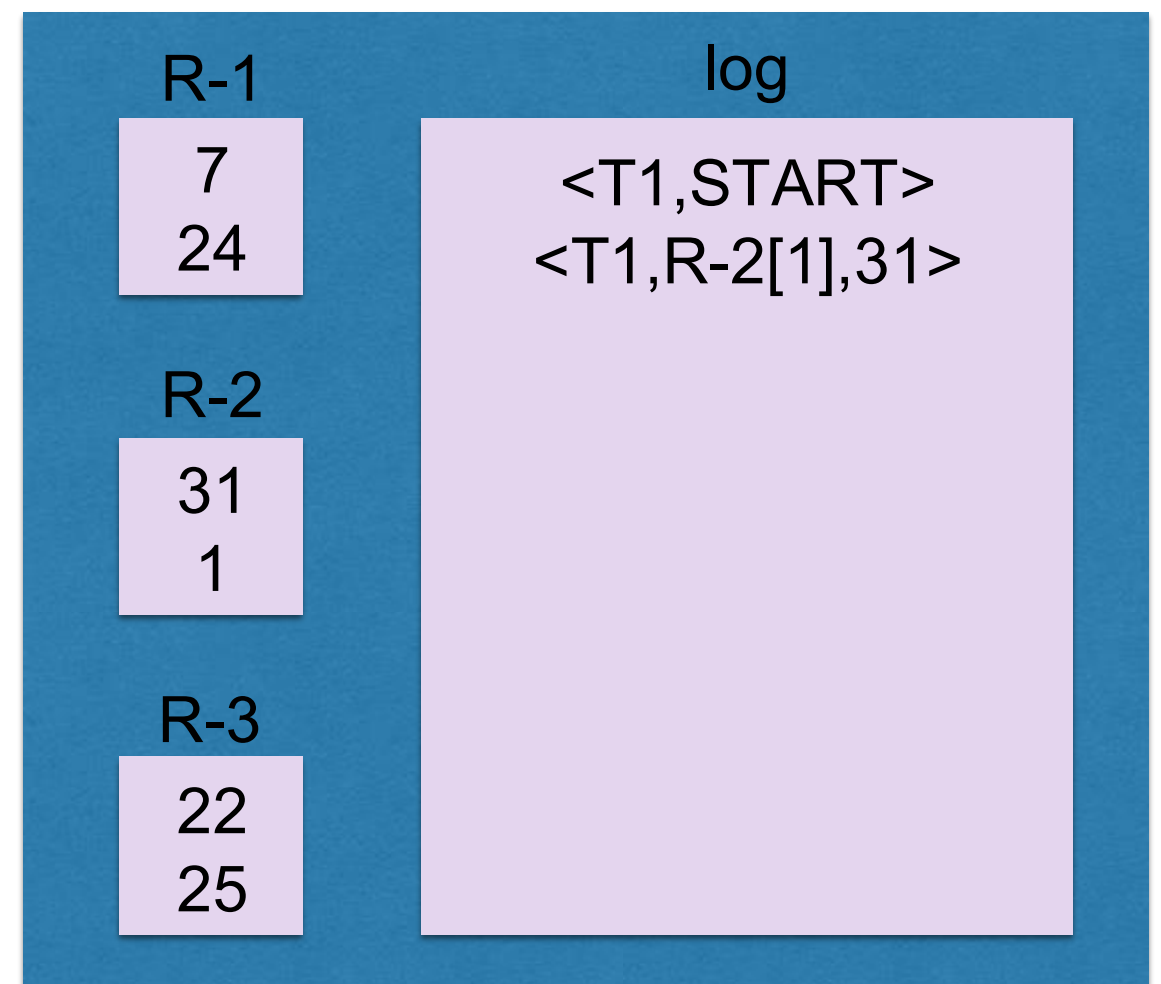
DBMS

Vamos al disco!

Buffer



Disco



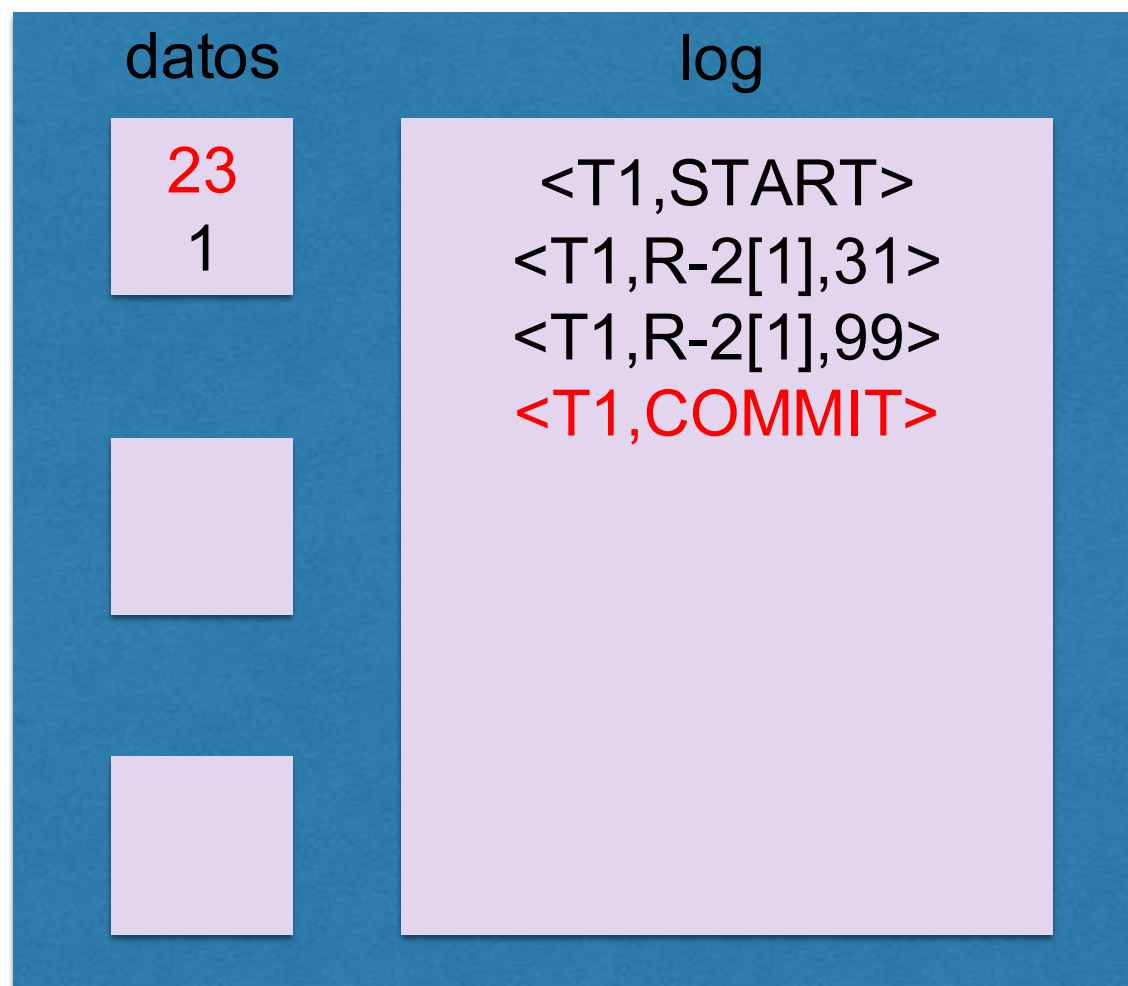


# Undo Logging – en la BD

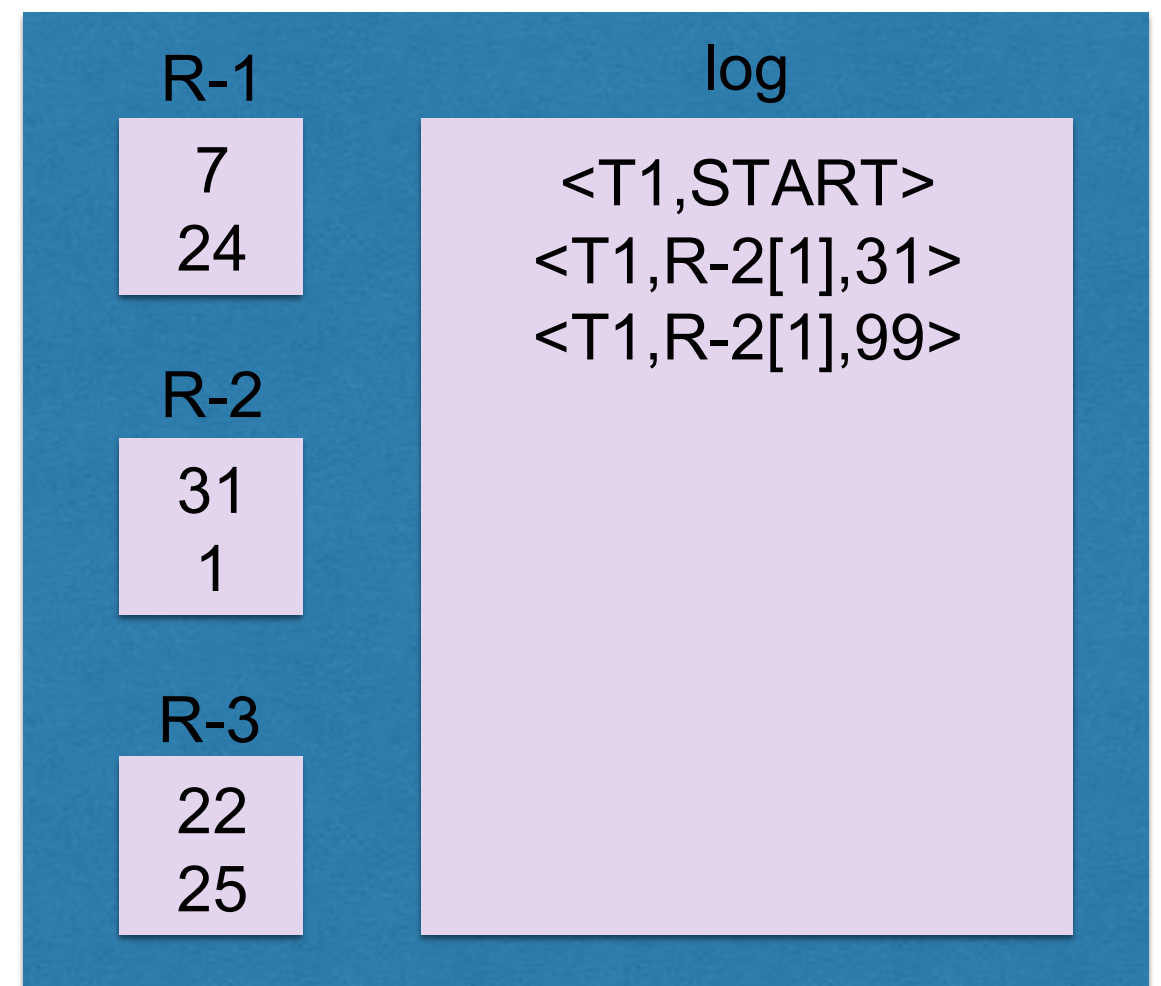
DBMS

Vamos al disco!

Buffer



Disco

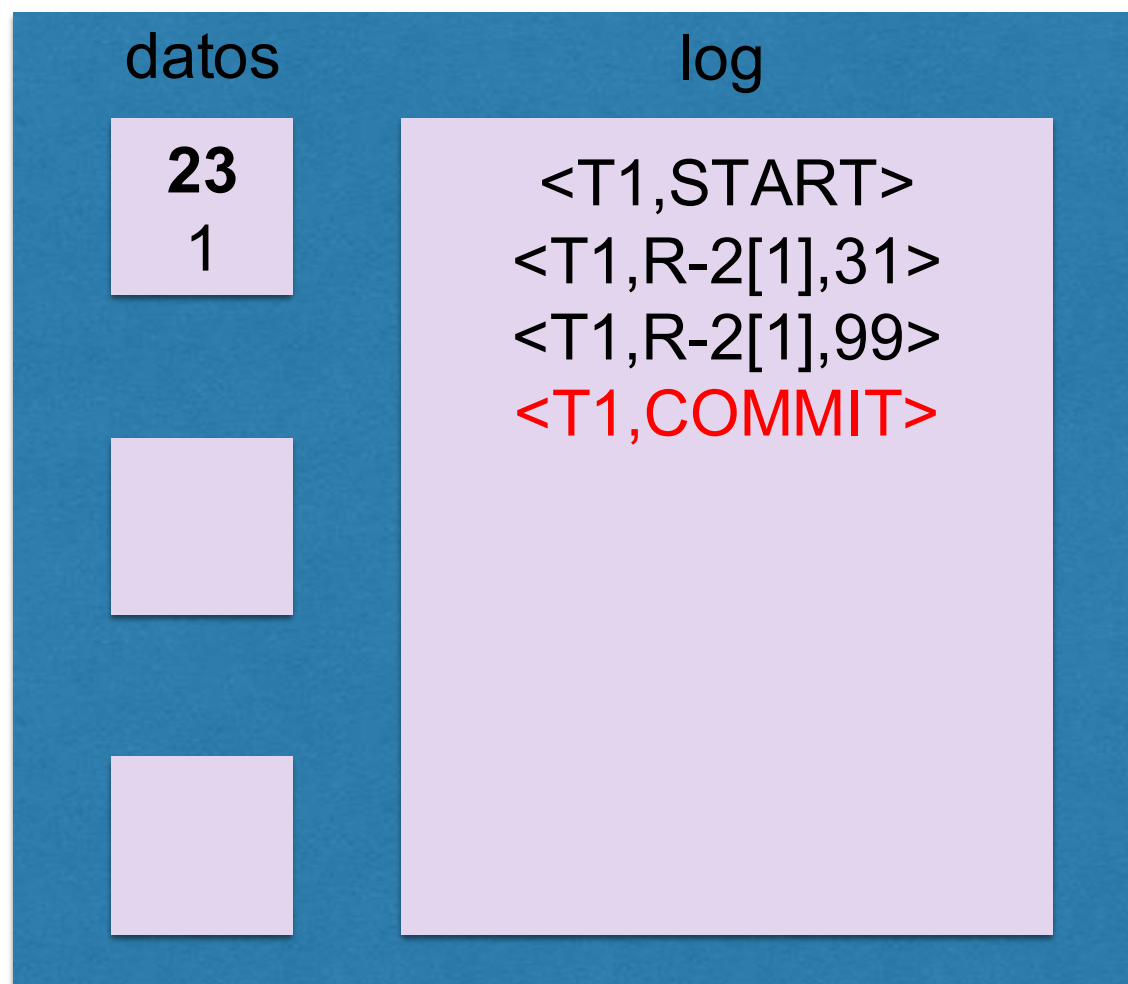


# Undo Logging – en la BD

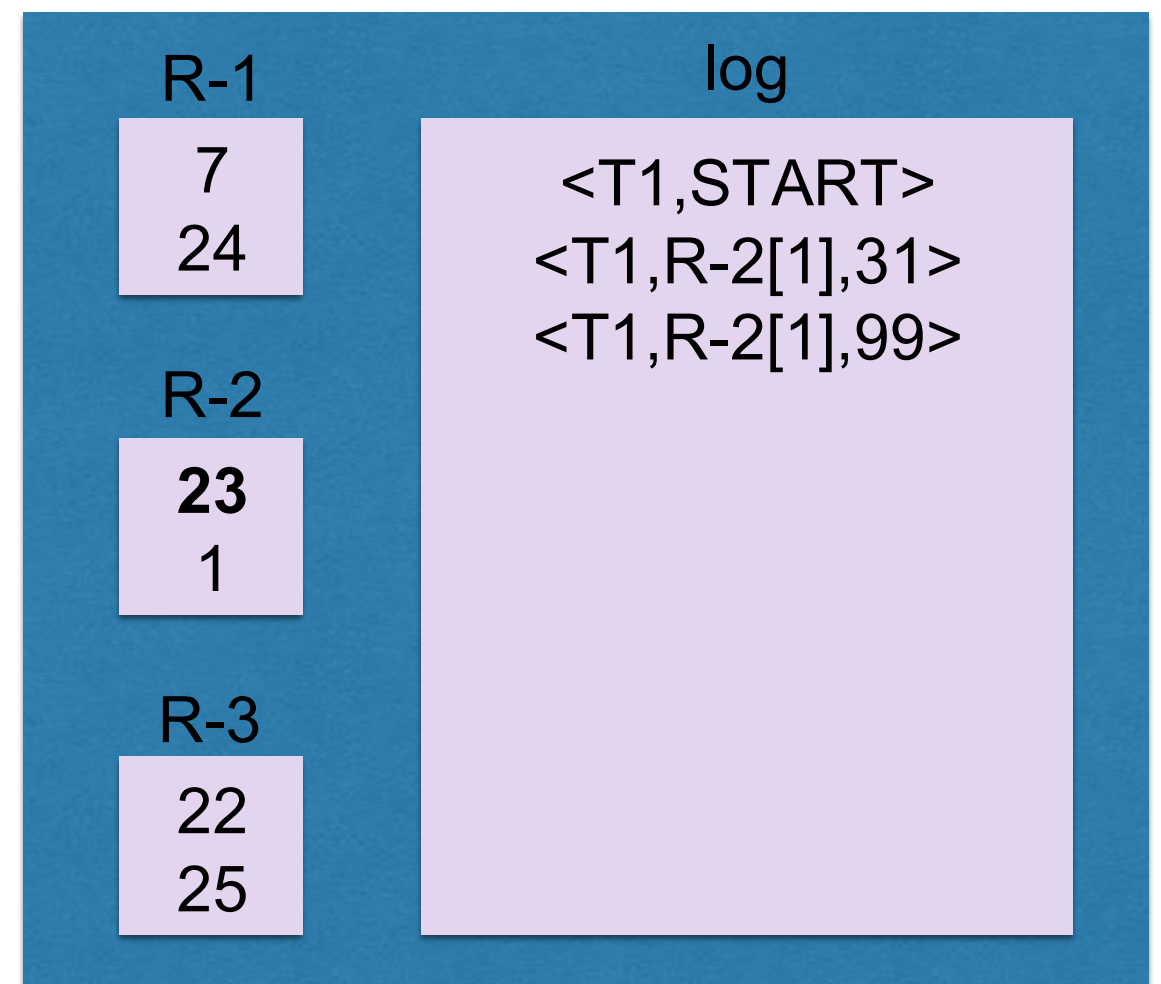
DBMS

Vamos al disco!

Buffer



Disco

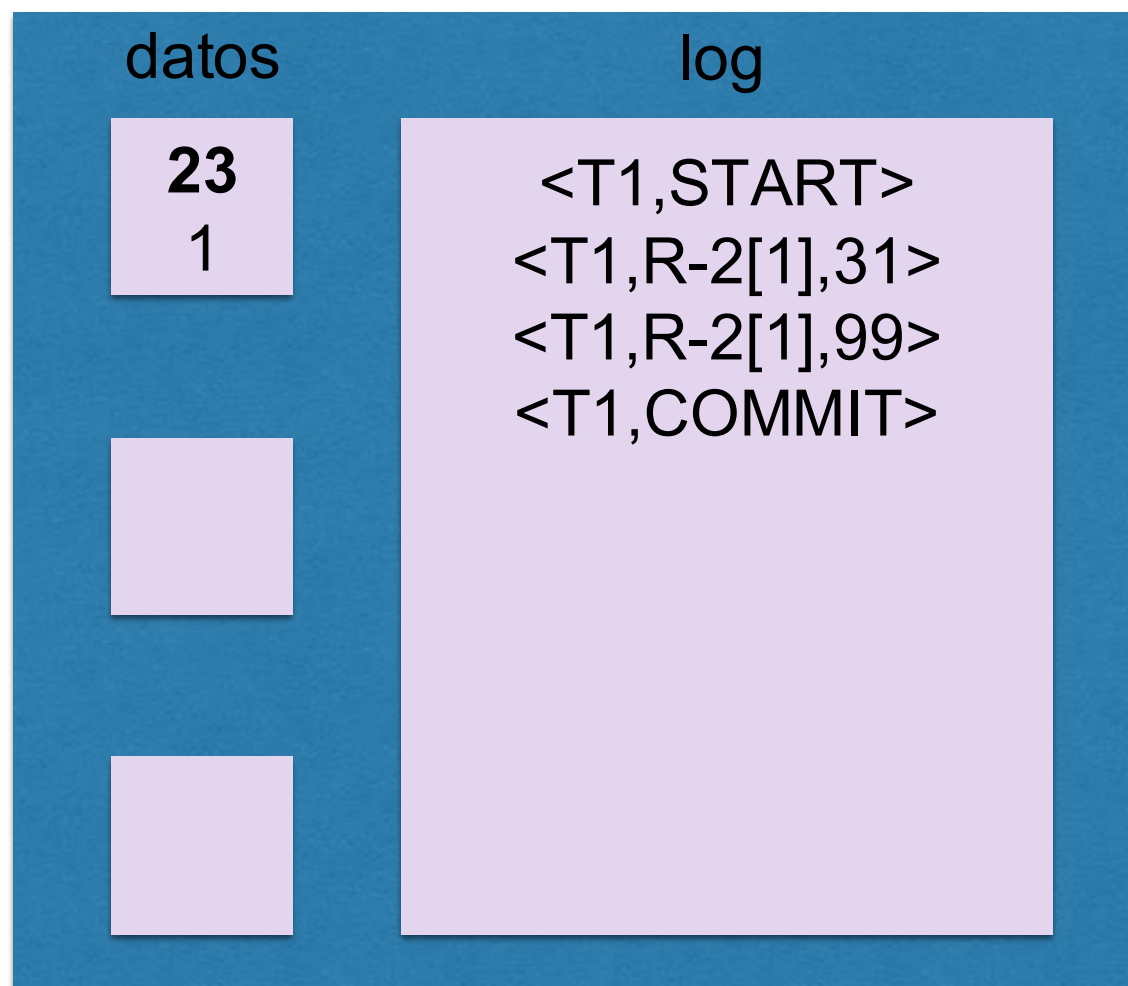


# Undo Logging – en la BD

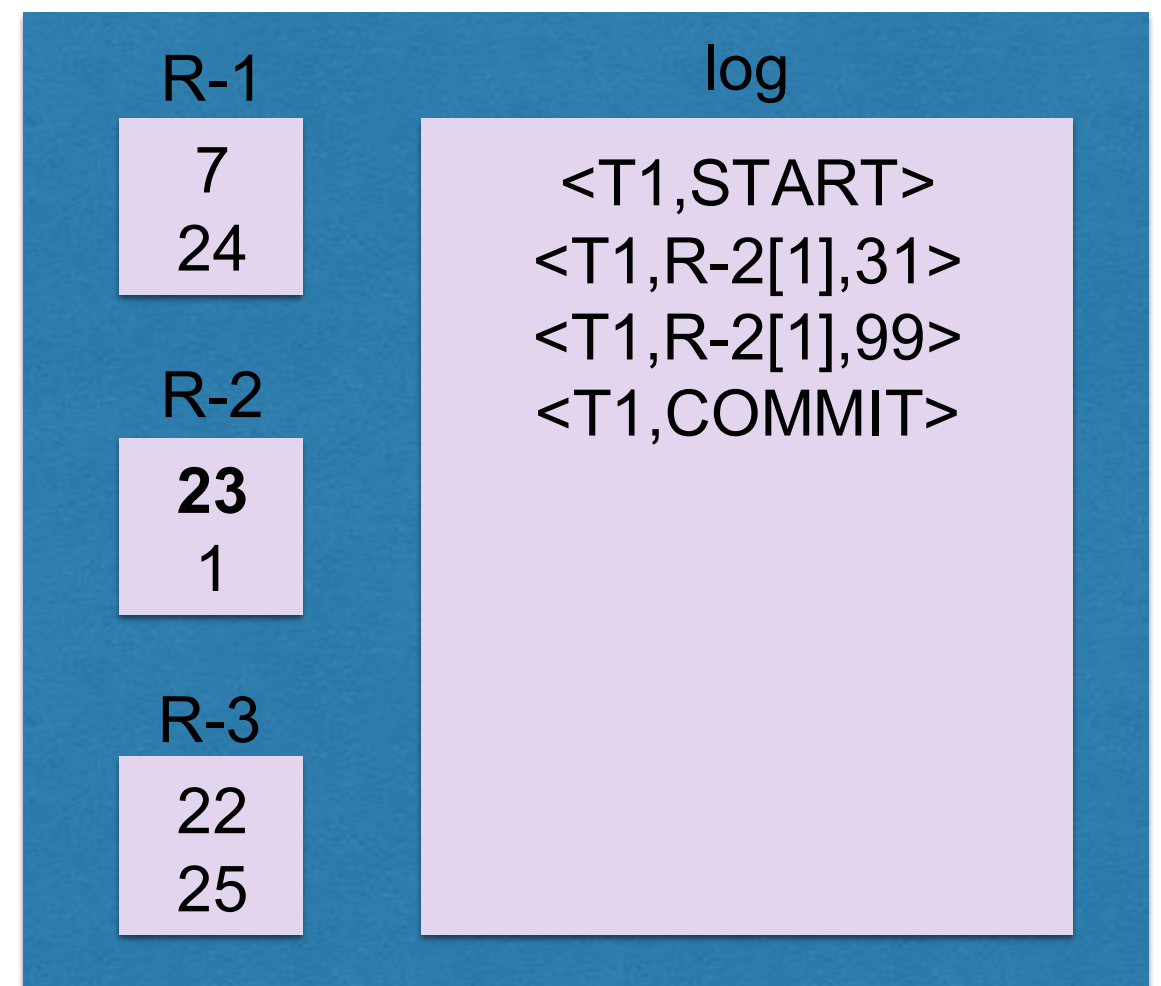
DBMS

Vamos al disco!

Buffer



Disco



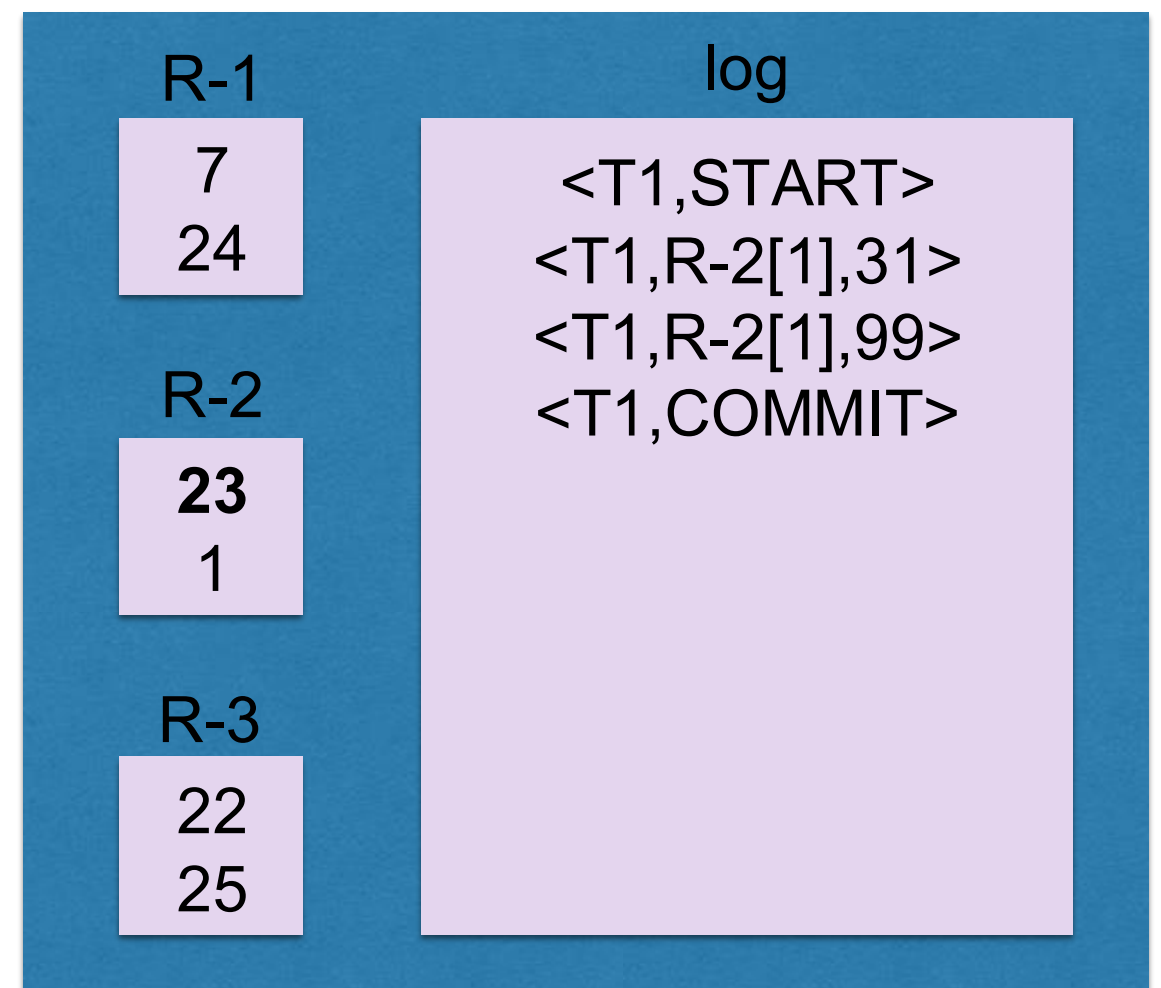
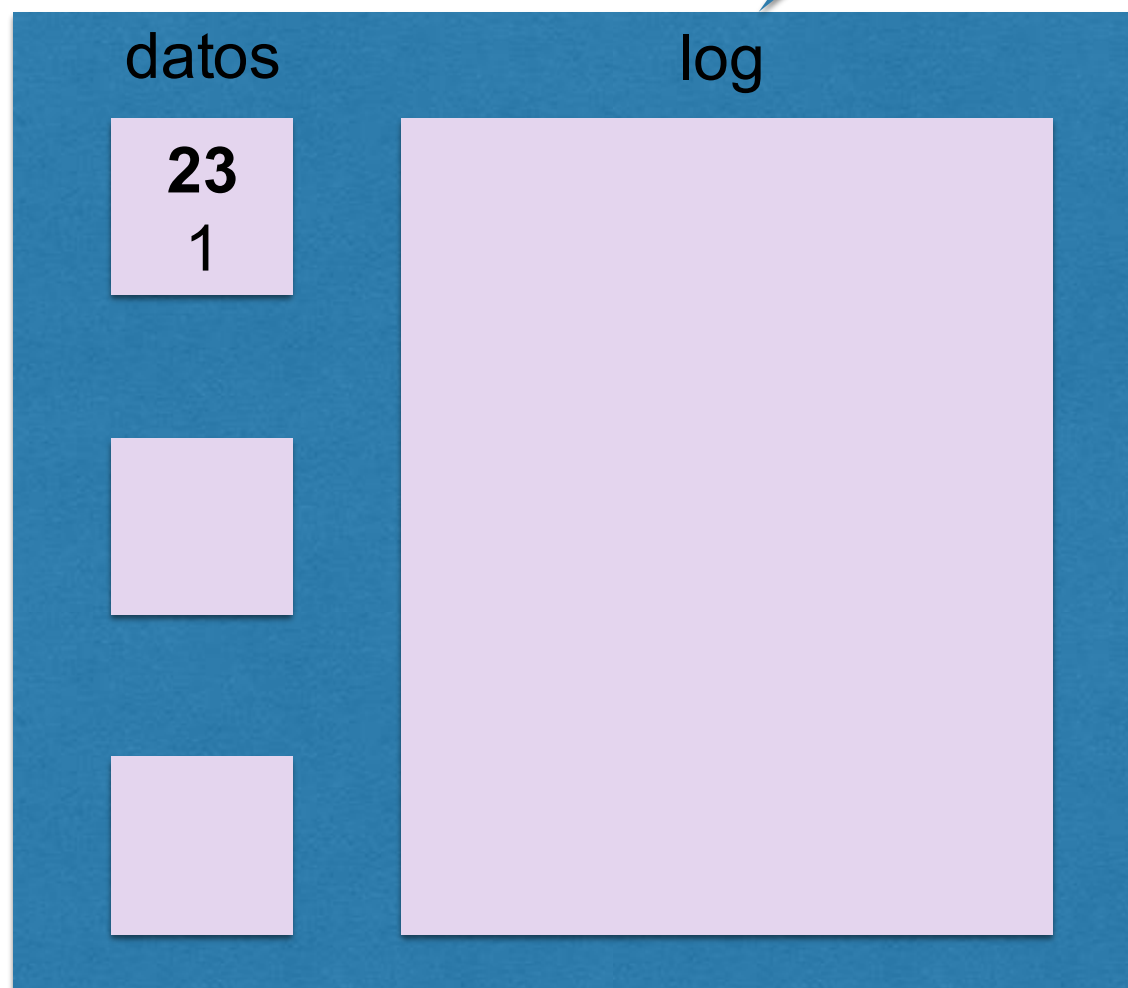
# Undo Logging – en la BD

DBMS

Buffer

Libera el buffer

Disco

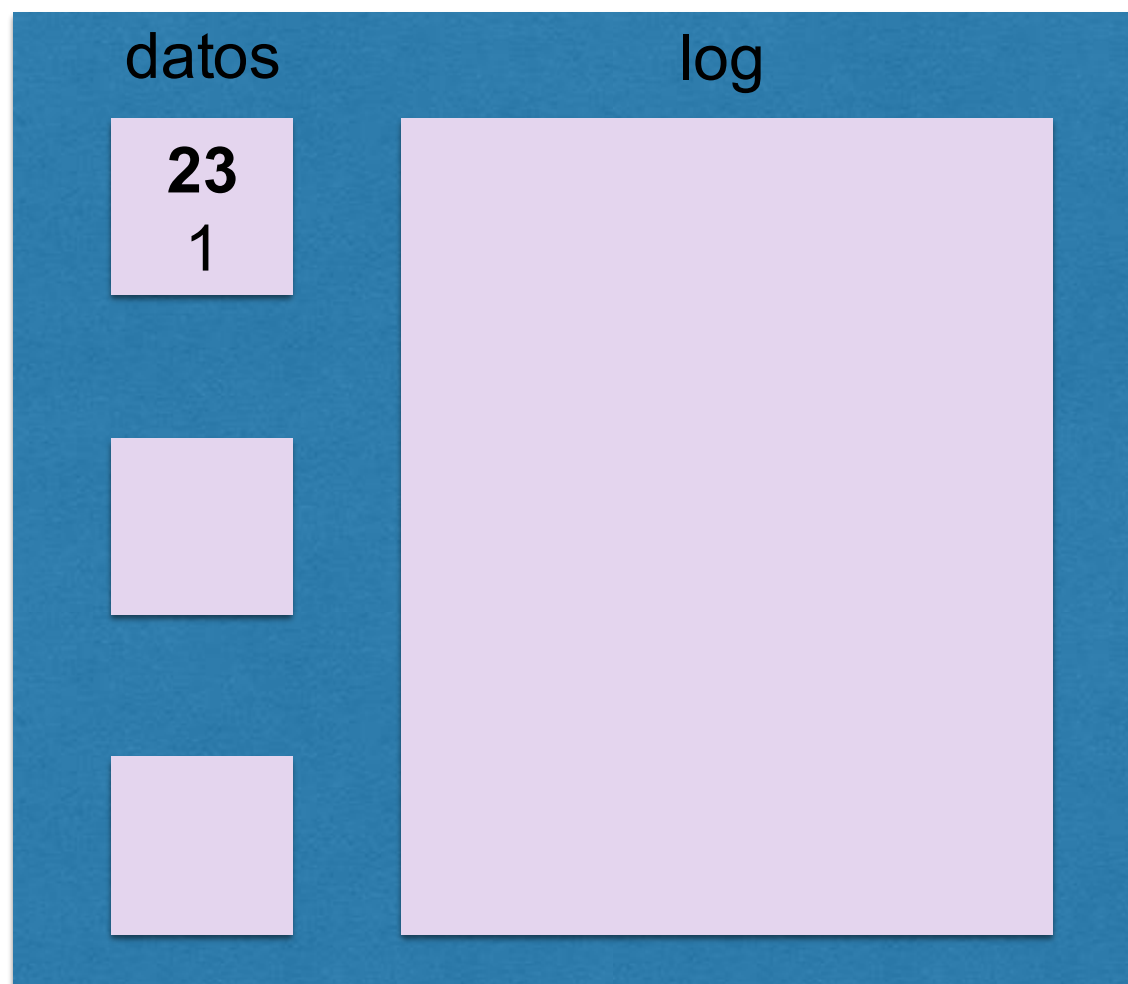


# Undo Logging – en la BD

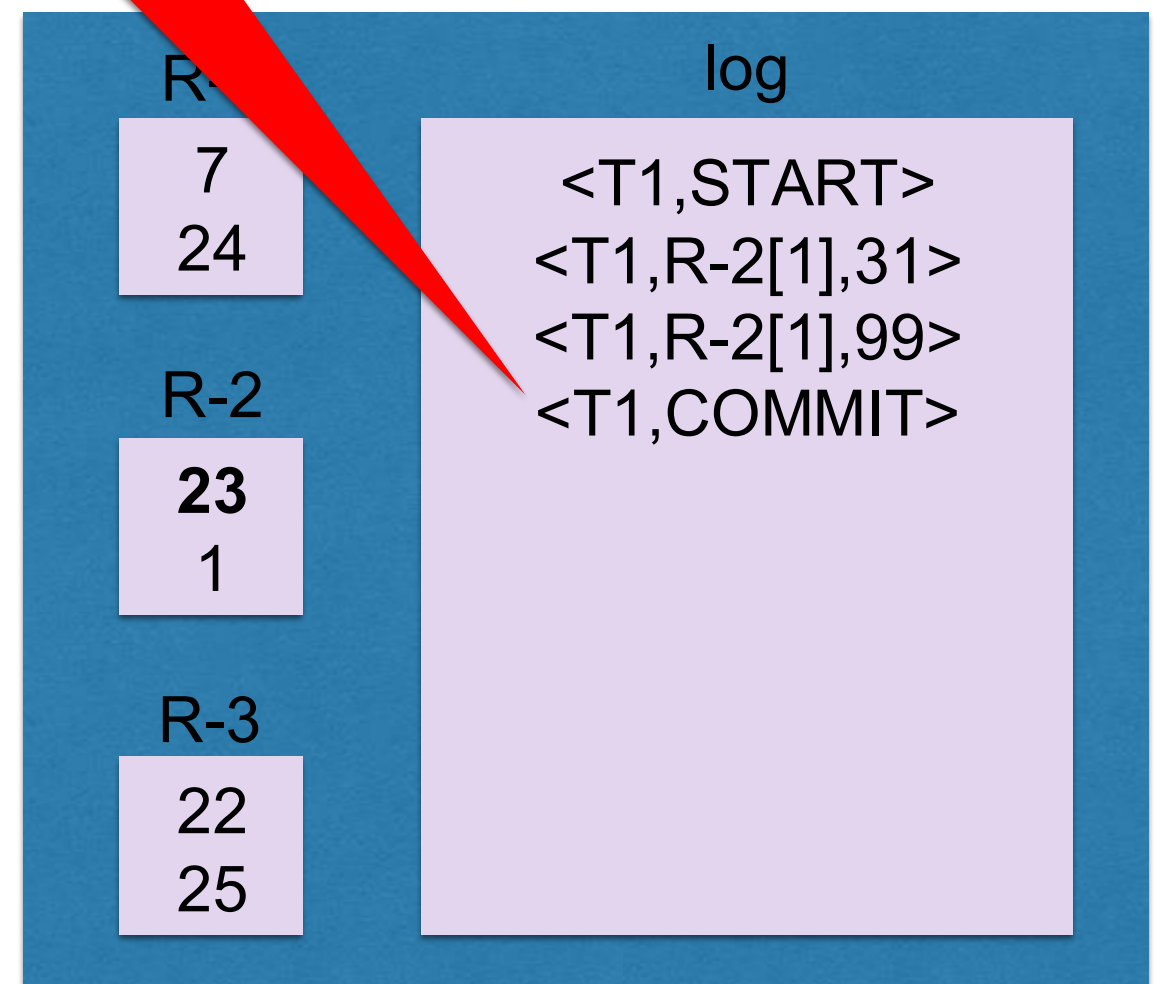
DBMS

COMMIT =  
datos están en disco

Buffer



Disco



# Recuperación con Undo Logging

Detectando fallas en el *log*:

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



Veamos un caso donde ocurre una falla...

# Ejemplo de log de transacción abortada

Start transaction T1

Reemplace el valor 31 de R.a por 99

T1 se cae

Reemplace el valor 99 de R.a por 23

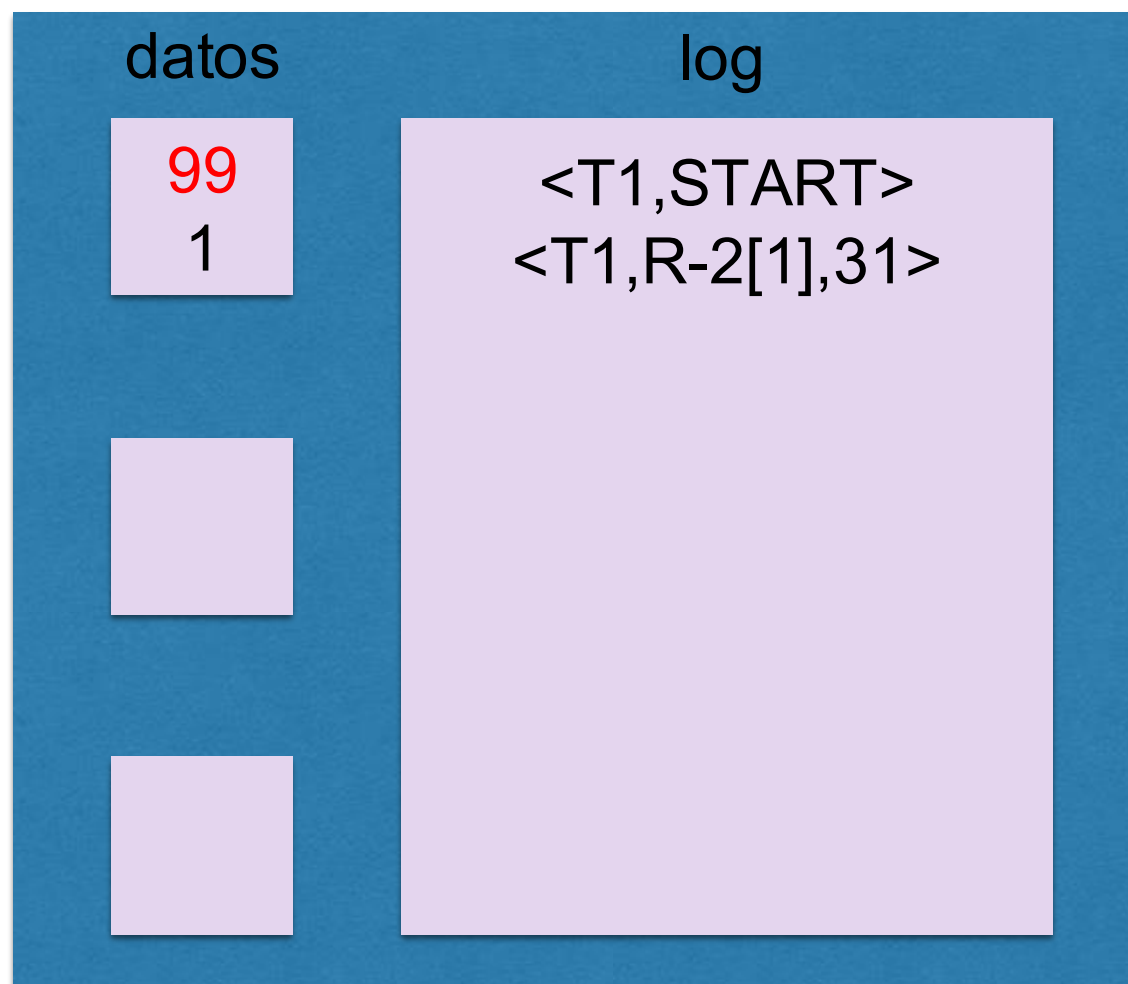
Commit T1

# Undo Logging – en la BD

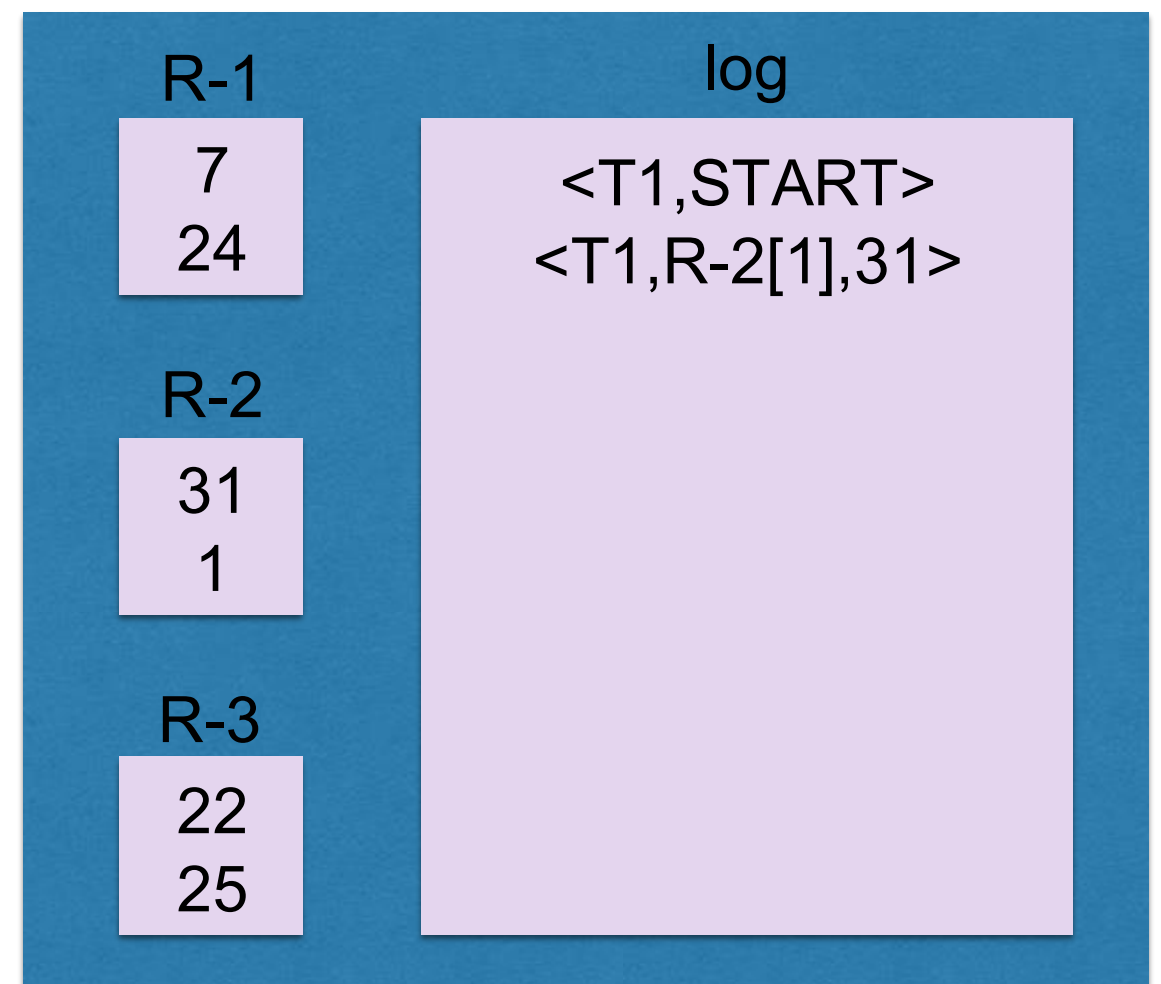
DBMS

T1: Cambio 31 a 99

Buffer



Disco



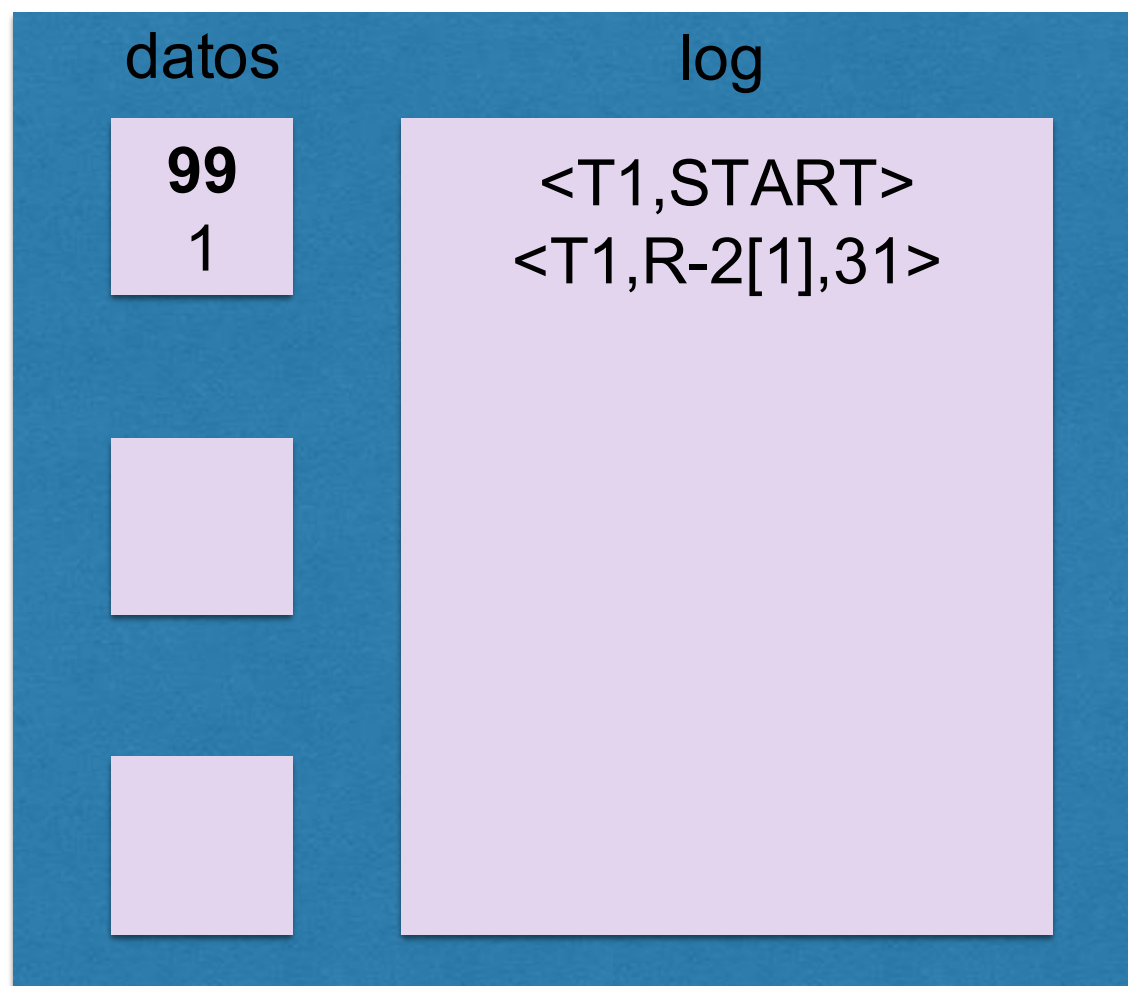


# Undo Logging – en la BD

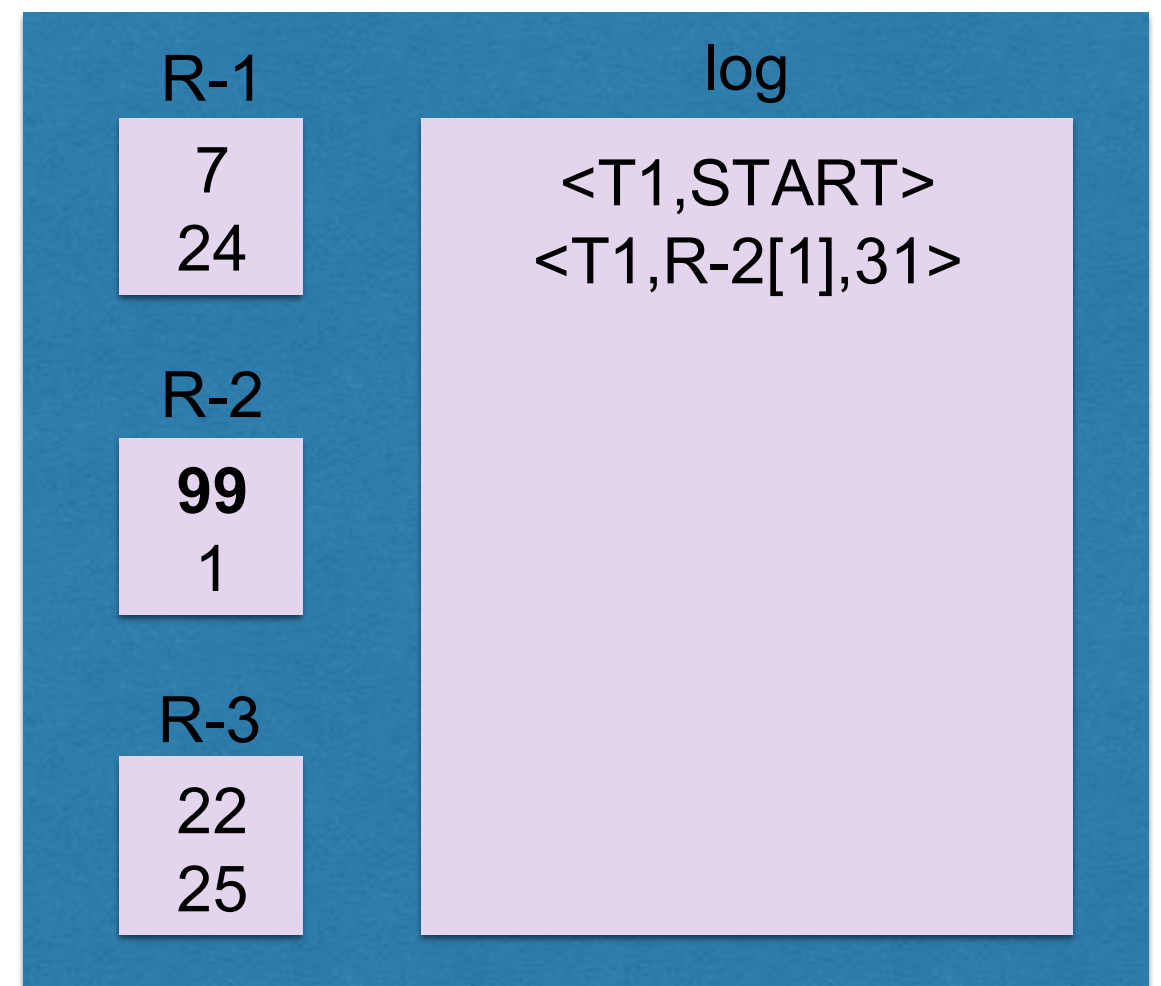
DBMS

T1: Cambio 31 a 99

Buffer



Disco

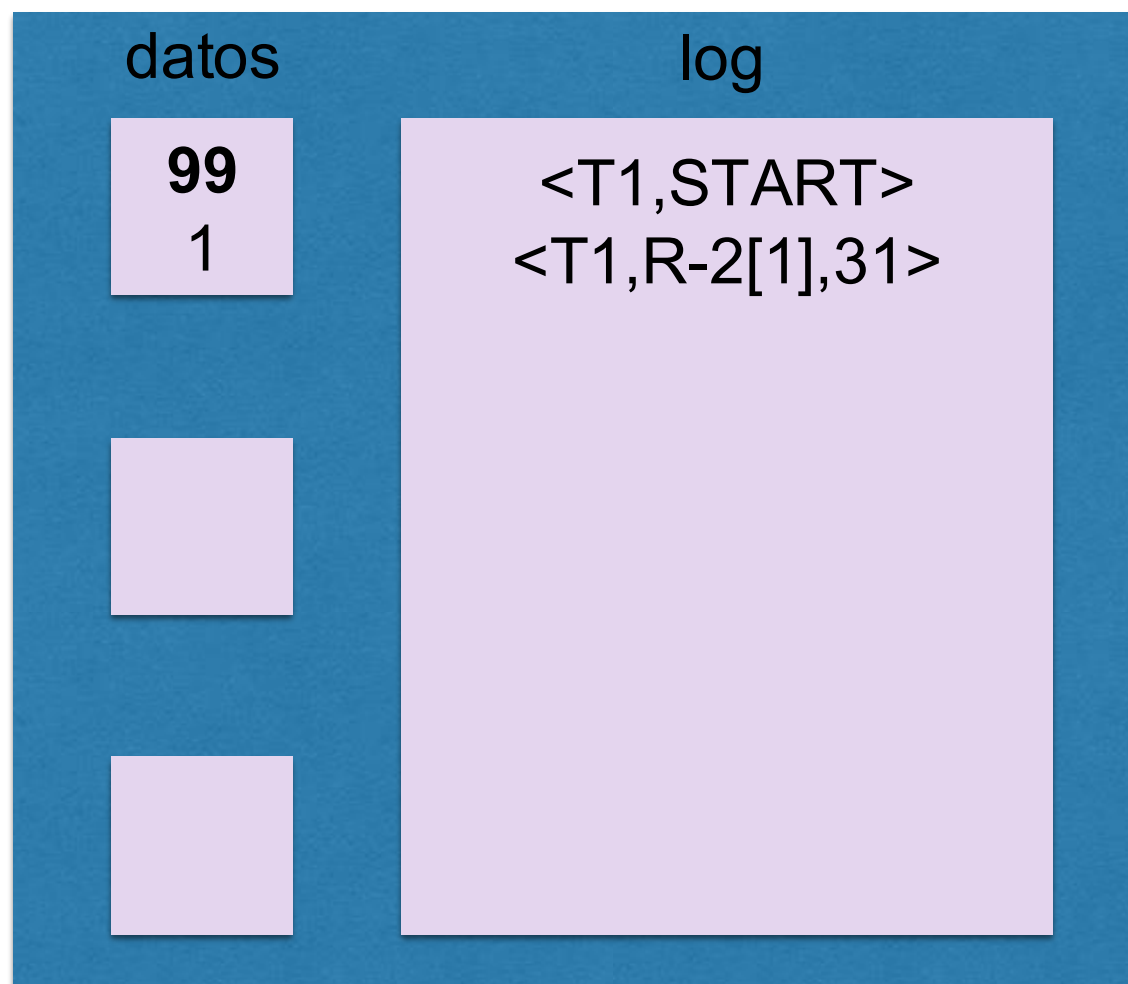


# Undo Logging – en la BD

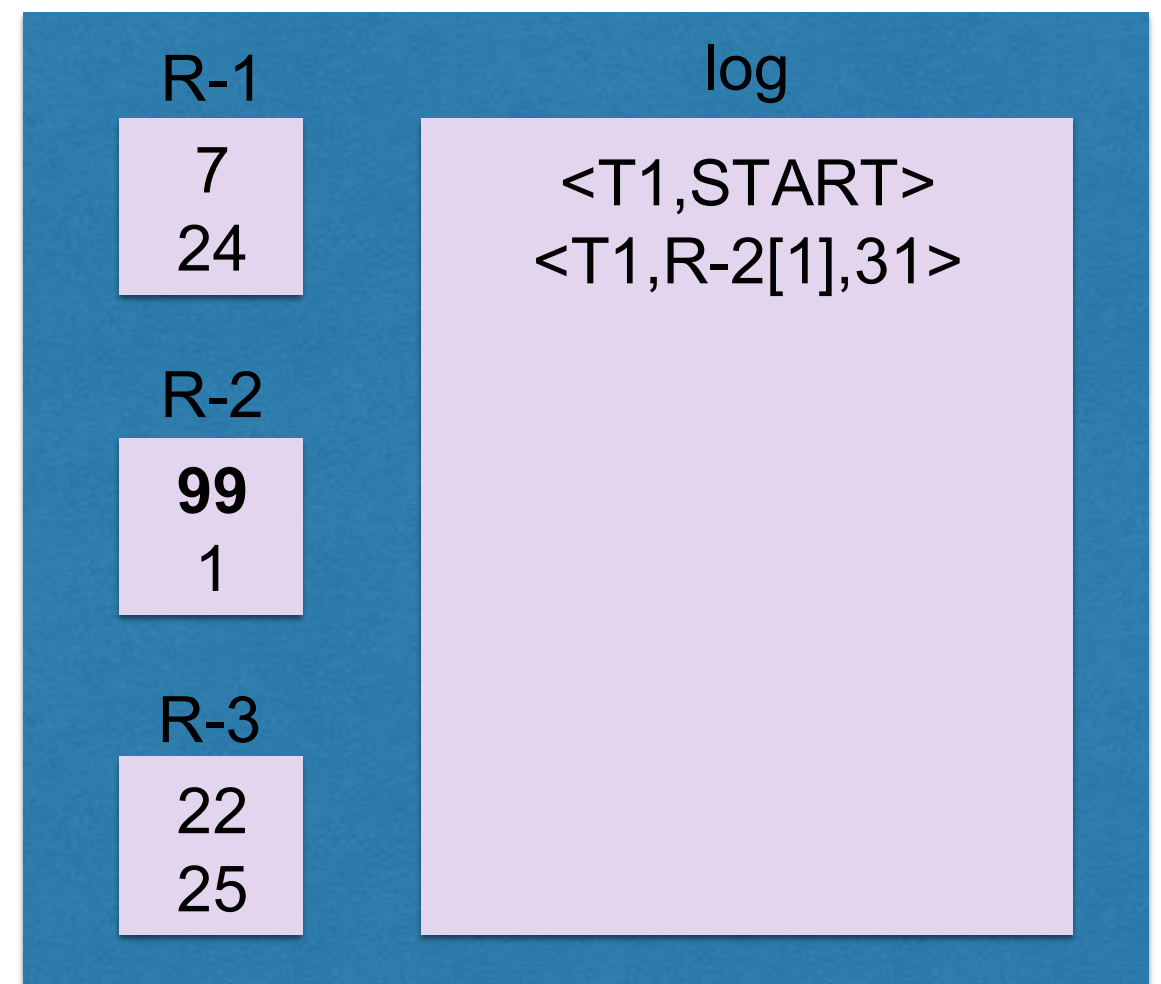
DBMS

T1: bota la ejecución

Buffer



Disco

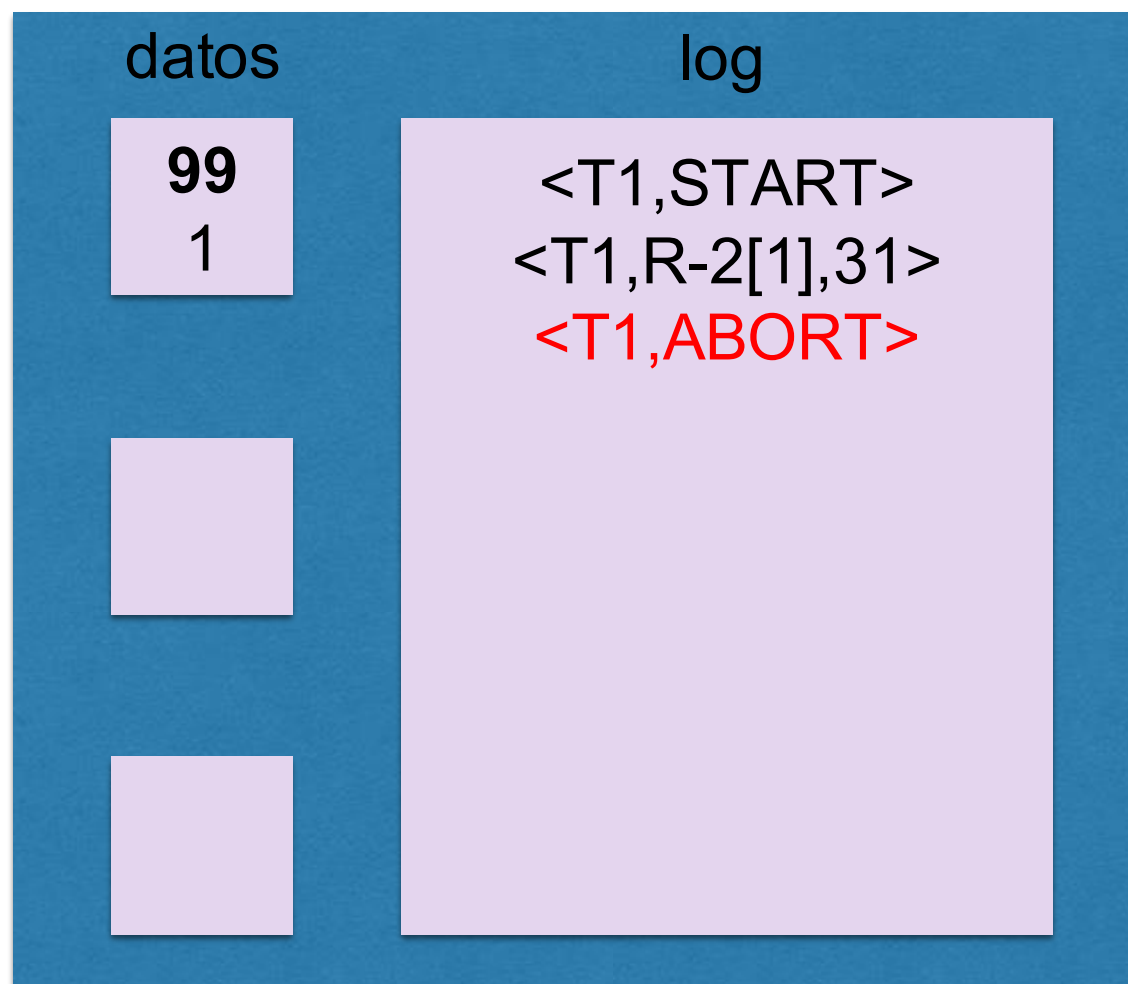


# Undo Logging – en la BD

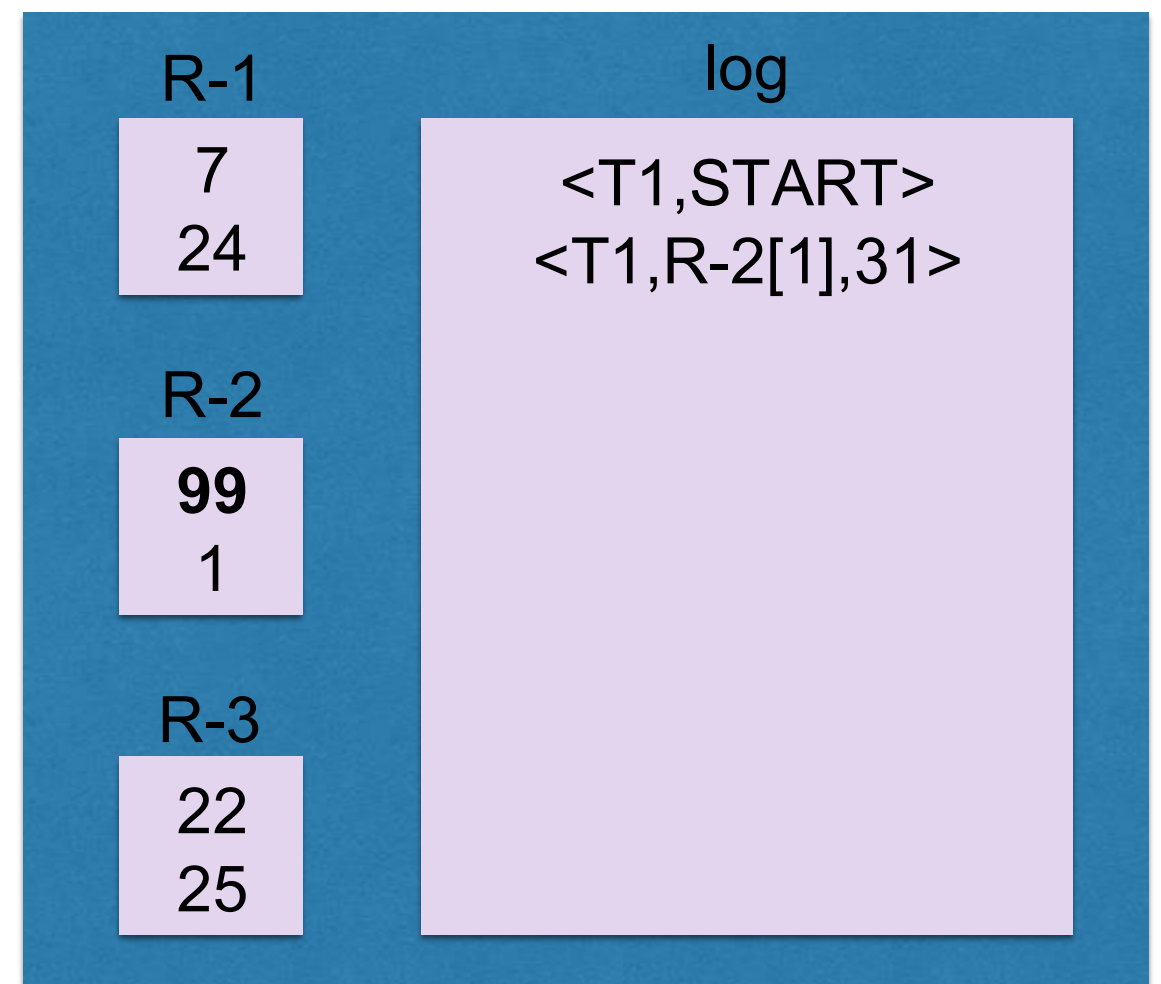
DBMS

T1: bota la ejecución

Buffer



Disco



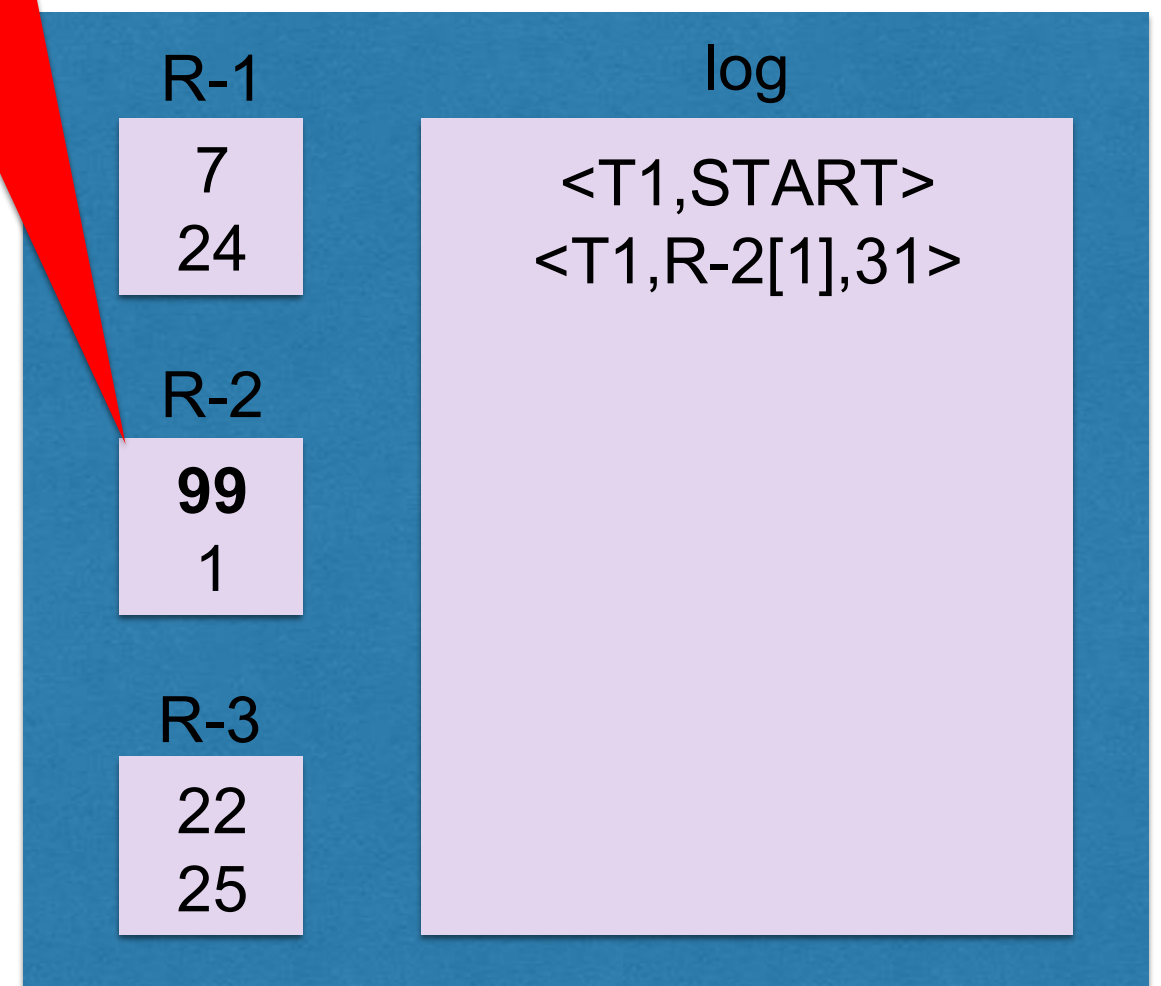
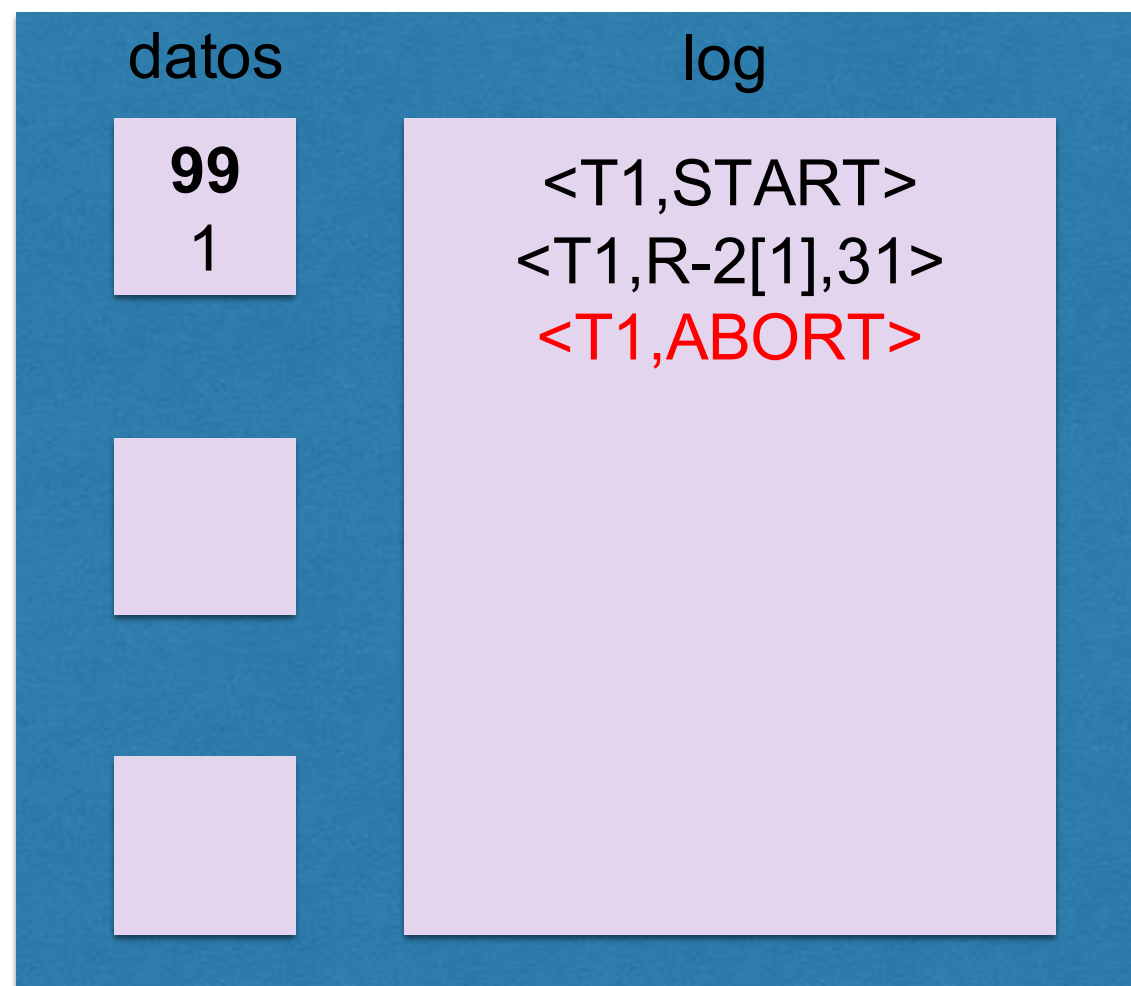
# Undo Logging – en la BD

DBMS

Asegurate deshacer los  
cambios!

Buffer

Disco

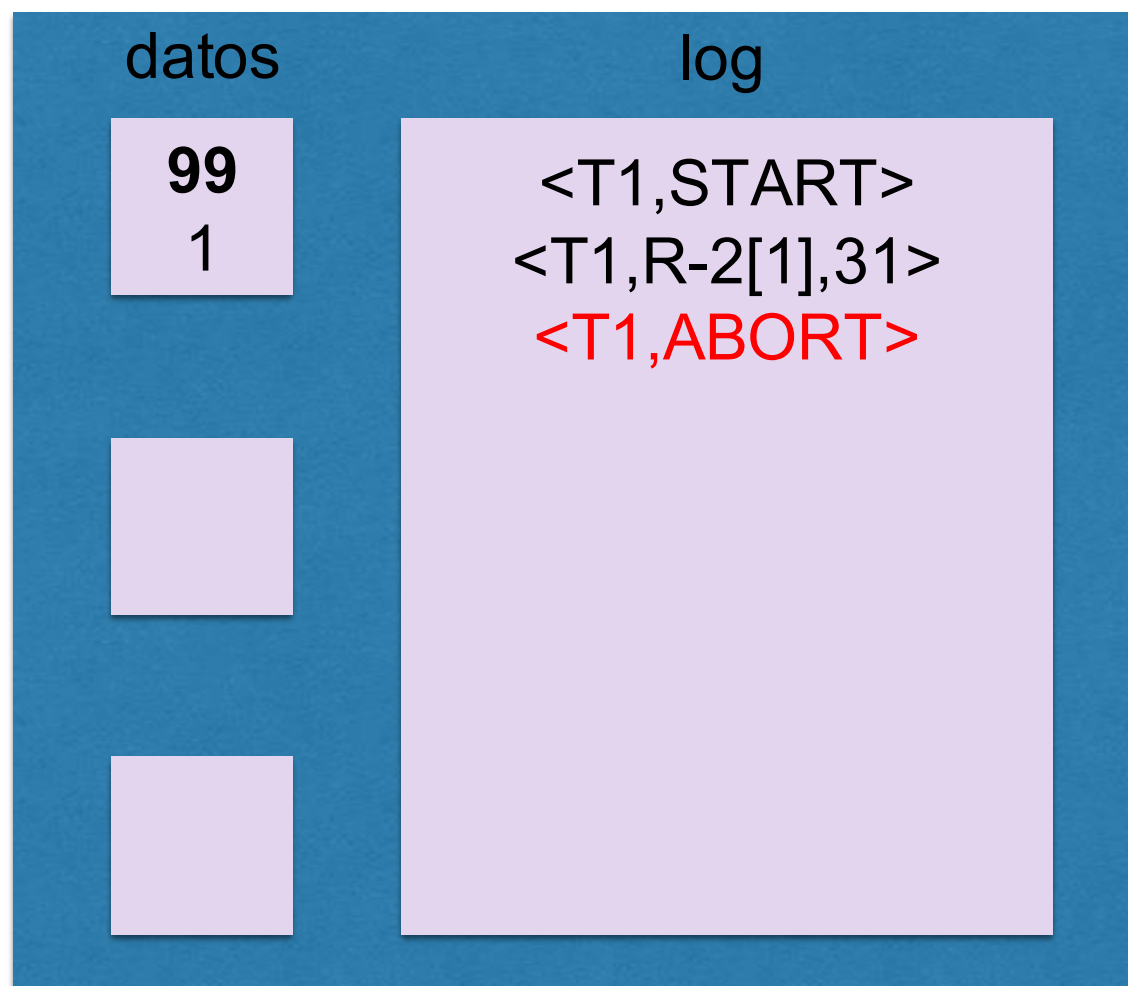


# Undo Logging – en la BD

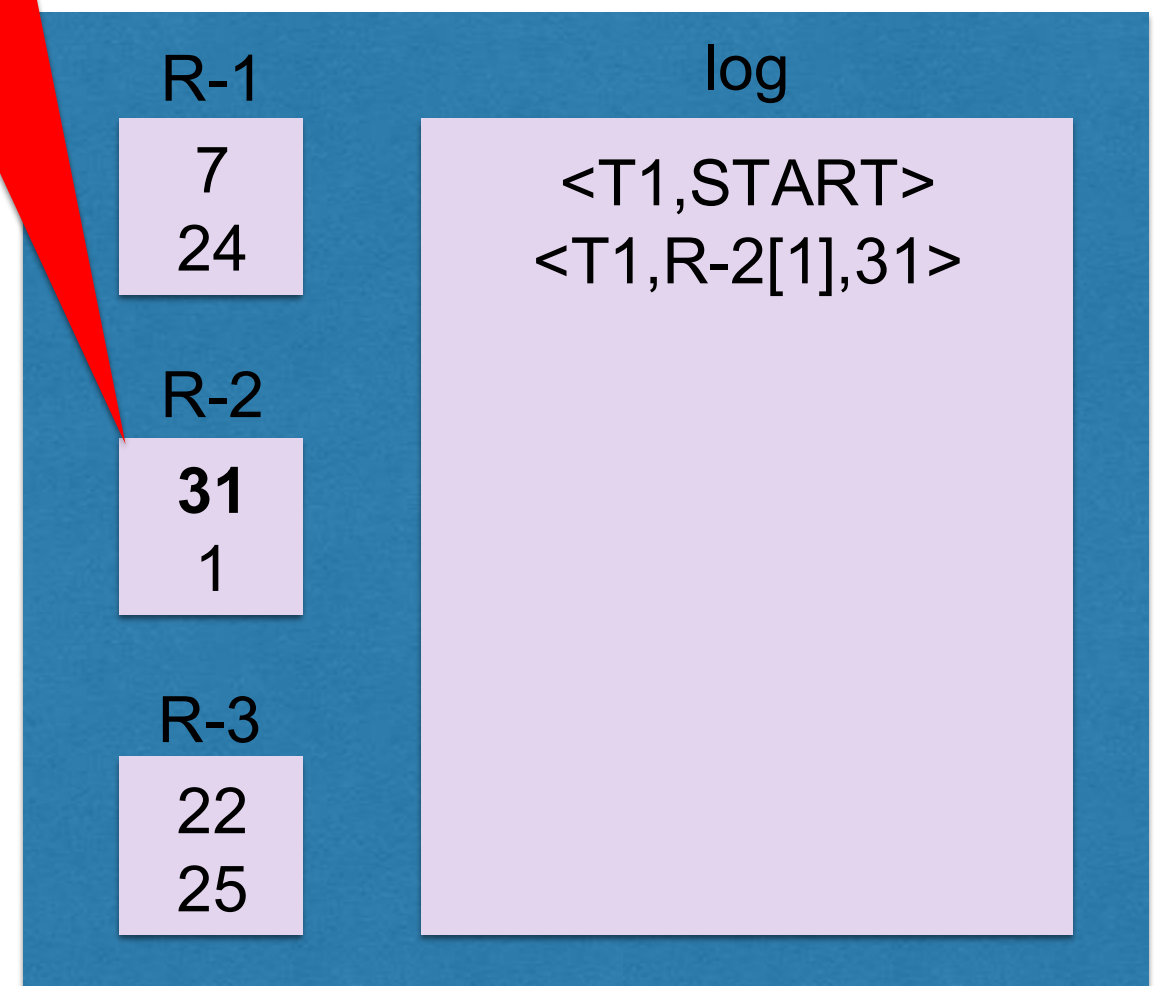
DBMS

Asegurate deshacer los  
cambios!

Buffer



Disco

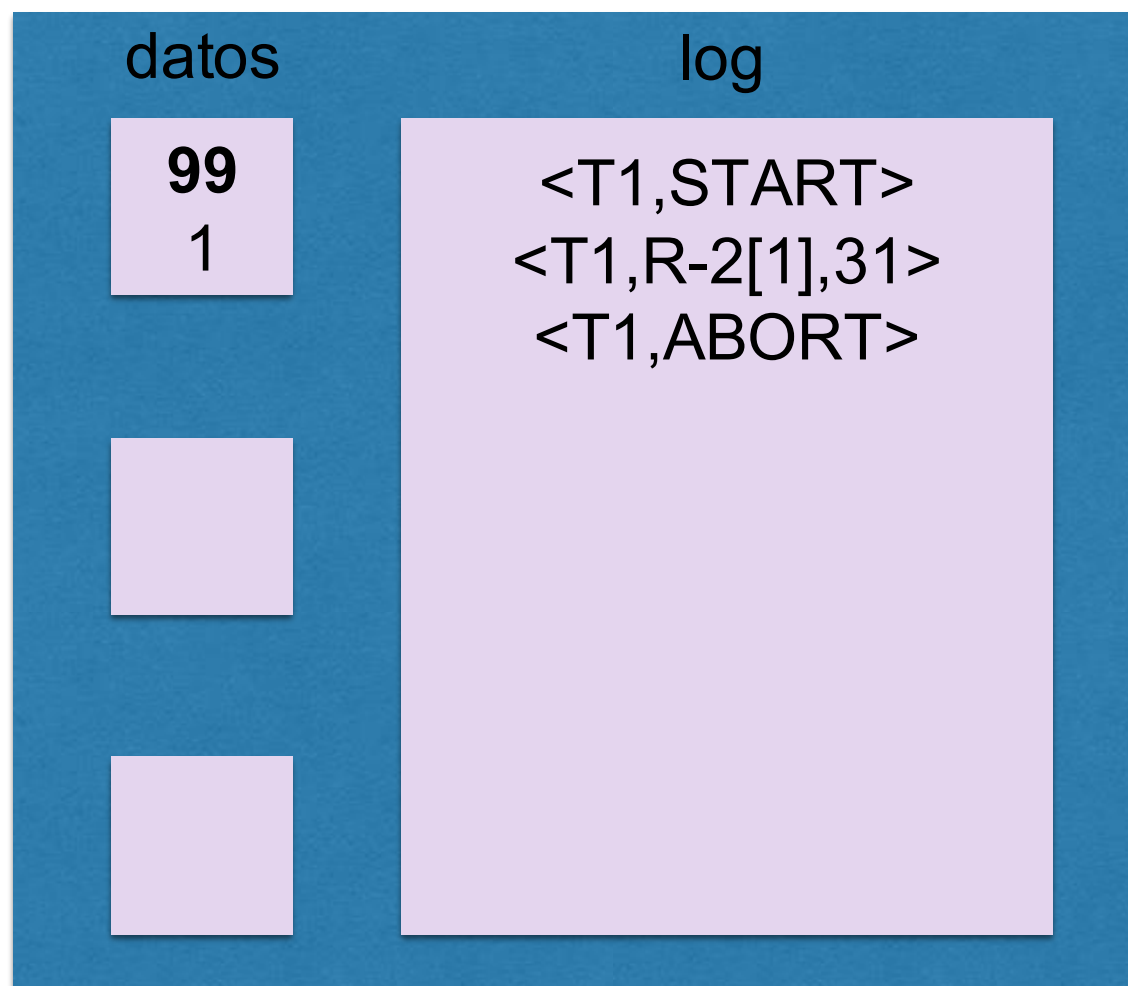


# Undo Logging – en la BD

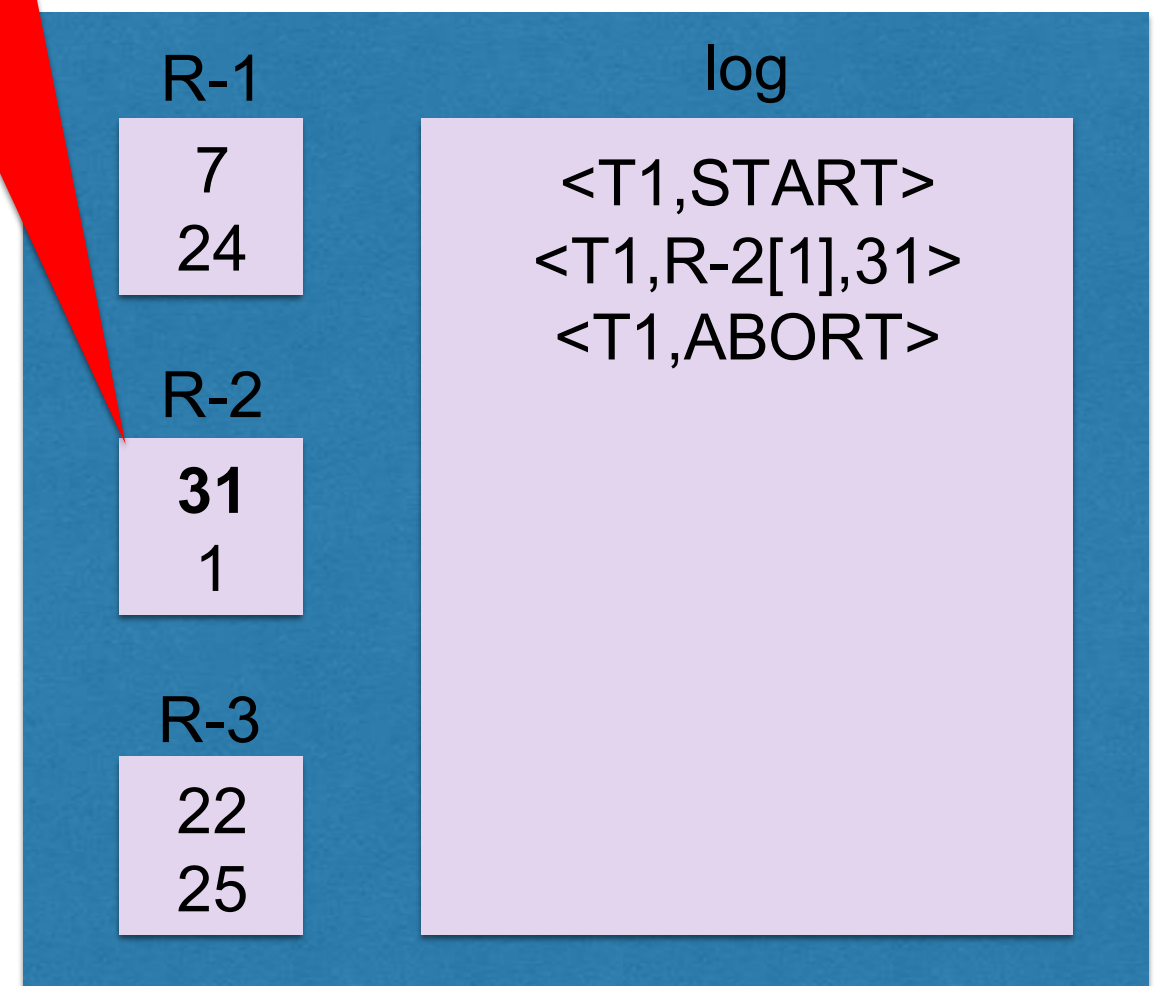
DBMS

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> ...



¡Encontramos una falla! Recuperemos con Undo Logging:

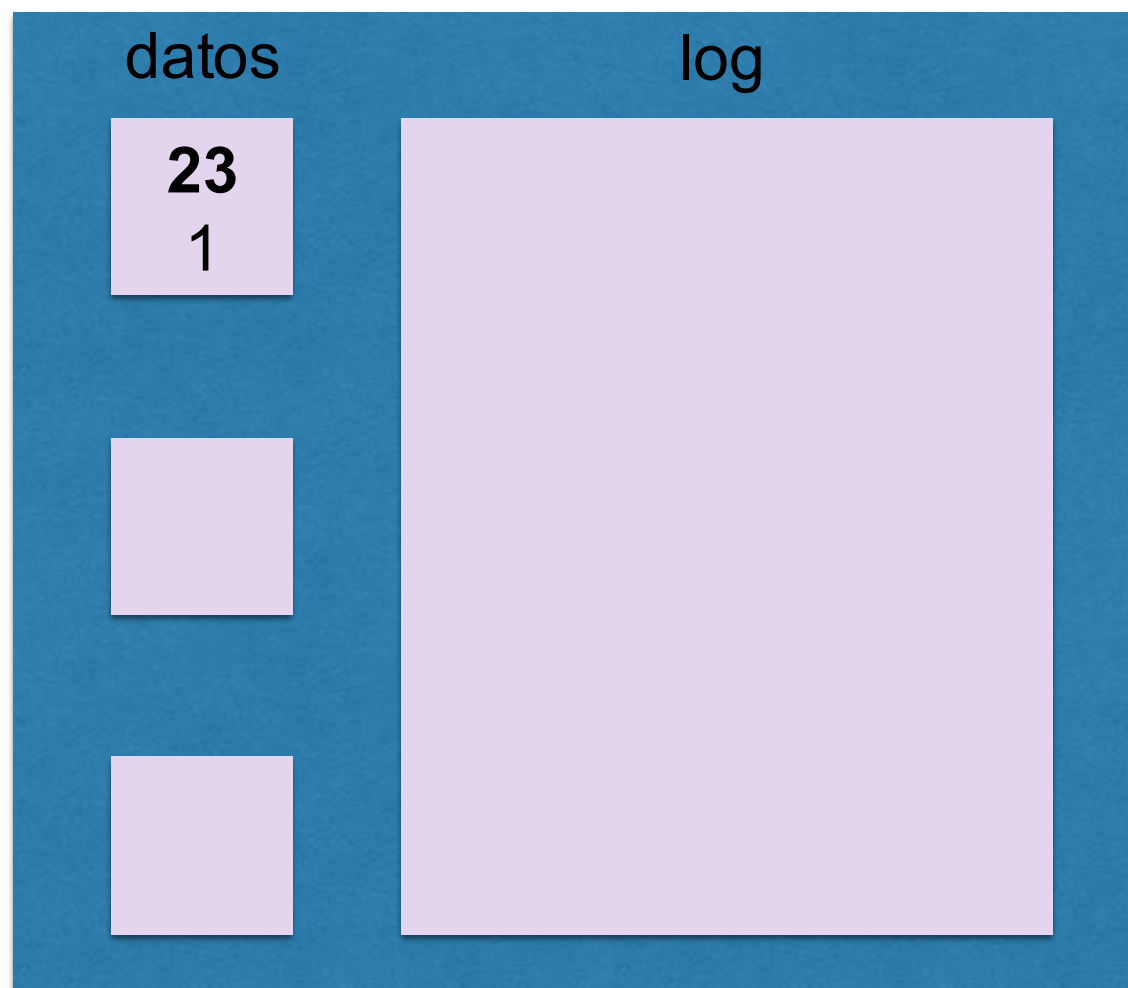


# Undo Logging – en la BD

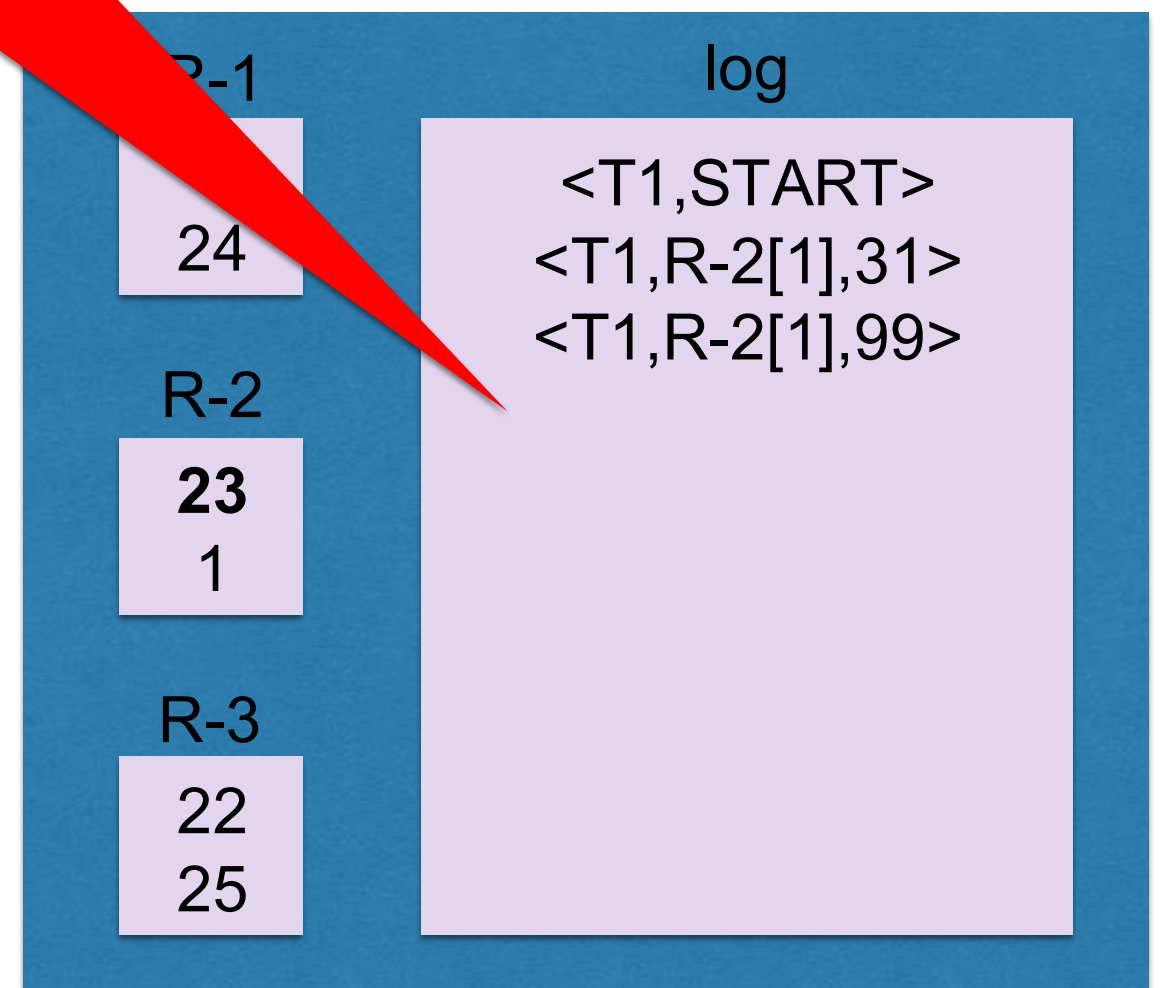
DBMS

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*?

¡Con **Checkpoints**!

# 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>

**Pero hay un gran problema:** es prácticamente necesario apagar el sistema para guardar un *checkpoint*.

Para solucionarlo, usaremos

**Nonquiescent Checkpoints**

# 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 <END CKPT>, hacemos *undo* de todo lo que haya *empezado* después del inicio del *checkpoint*
- Si encontramos un <START CKPT ( $T_1, \dots, T_n$ )> sin su <END CKPT>, 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*

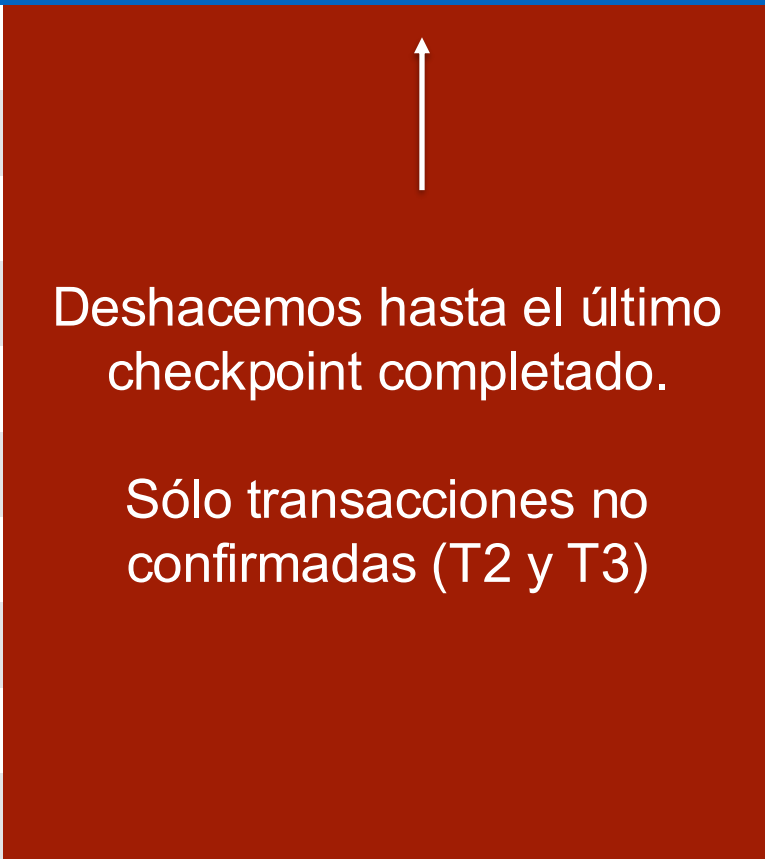
Considere este *log* después de una falla:

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

**Pero tenemos un problema:** con Undo logging no es posible hacer COMMIT antes de almacenar los datos en disco.

Por lo tanto las transacciones se toman más tiempo en terminar.

¿Existe alguna manera de hacer commit antes de almacenar los datos en el disco? ¡Si!, con **Redo Logging**.

# Redo Logging

# Redo Logging

Los *logs* son:

- **<START T>**
- **<COMMIT T>**
- **<ABORT T>**
- **<T, X, v>** donde v es el valor **nuevo** de X
- **<T,END>**

# Redo Logging - Reglas

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:

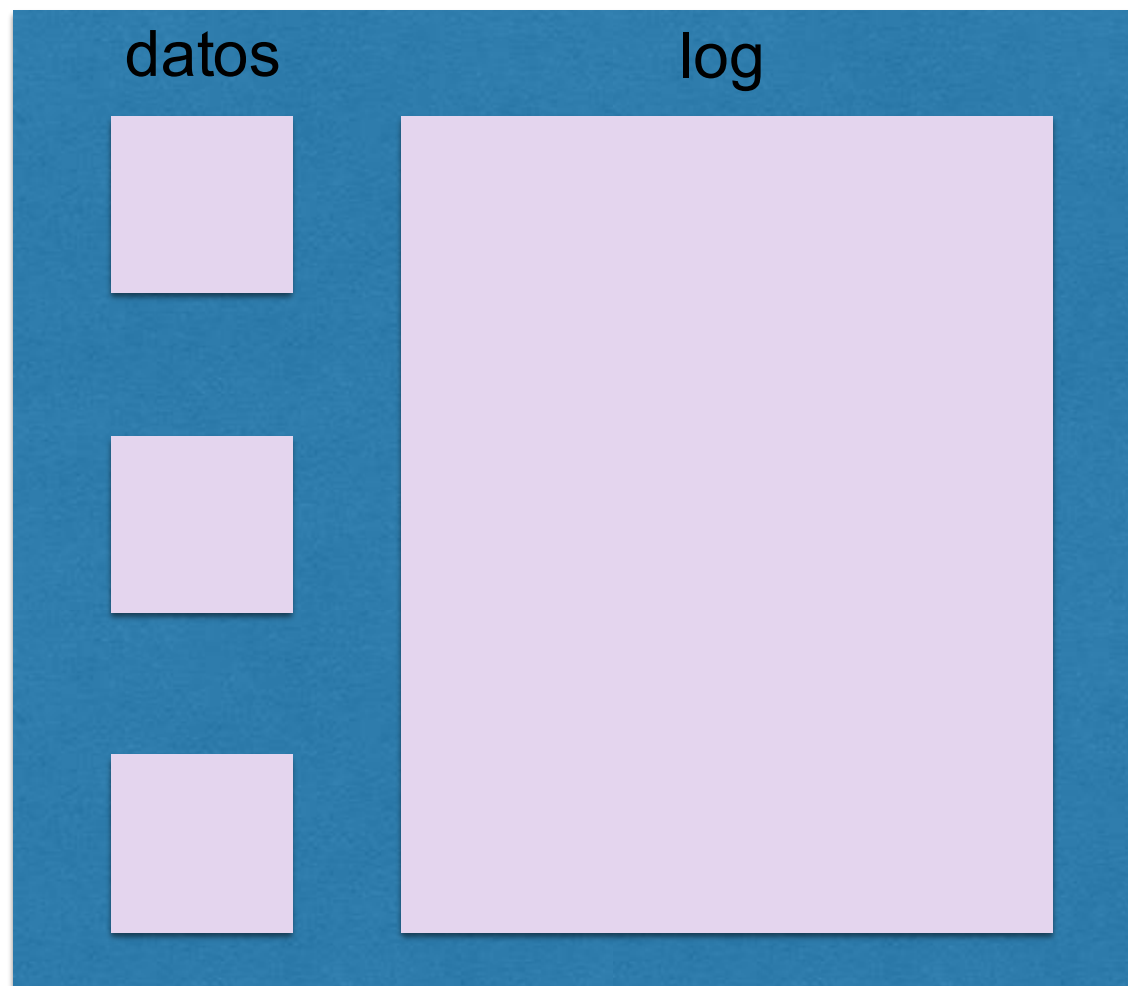
- Escribir el log  $\langle T, X, v \rangle$
- Escribir  $\langle \text{COMMIT } T \rangle$
- Escribir los datos en disco
- Escribir  $\langle T, \text{END} \rangle$  en log (en disco)

# Redo Logging – en la BD

DBMS

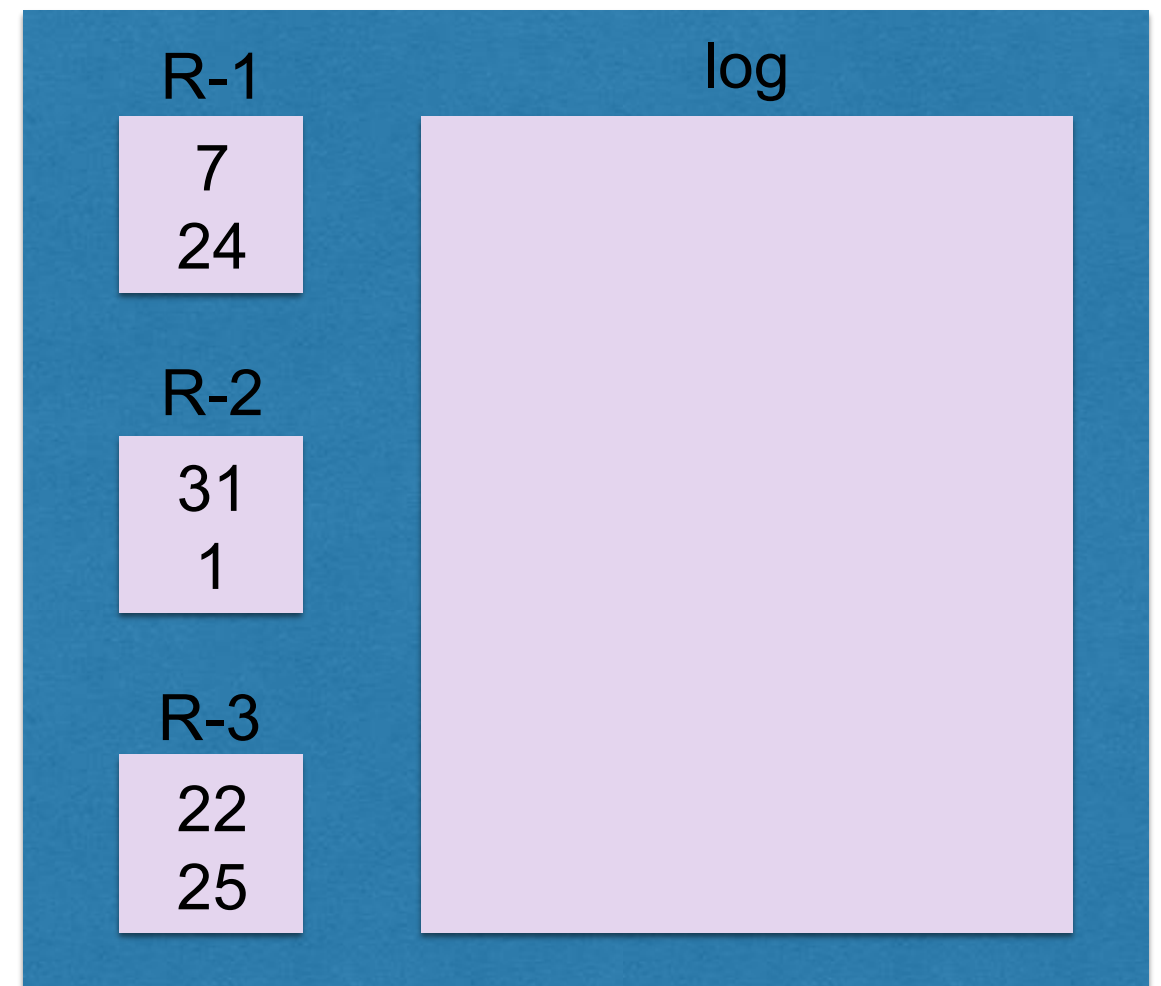
T1: voy a empezar

Buffer



R(A int)

Disco

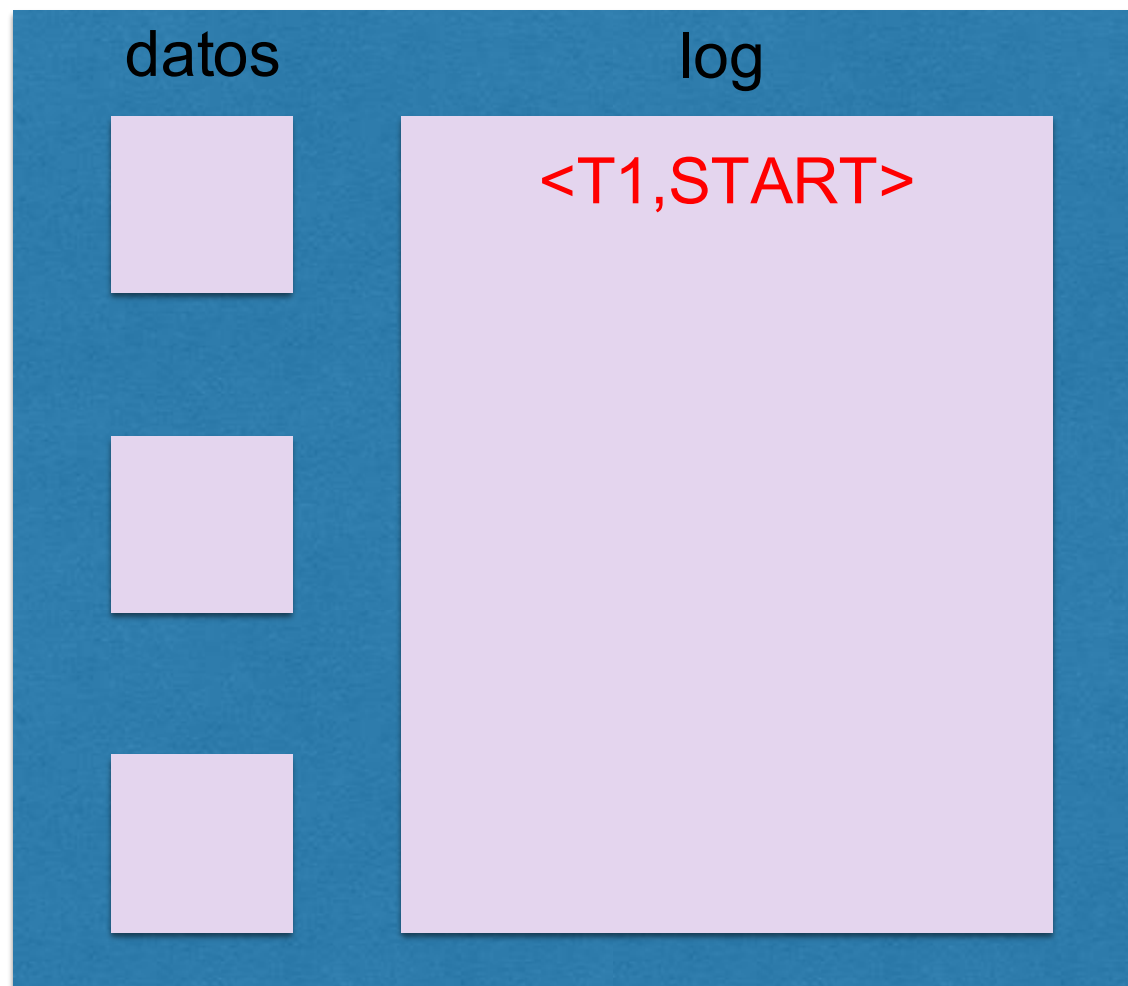


# Redo Logging – en la BD

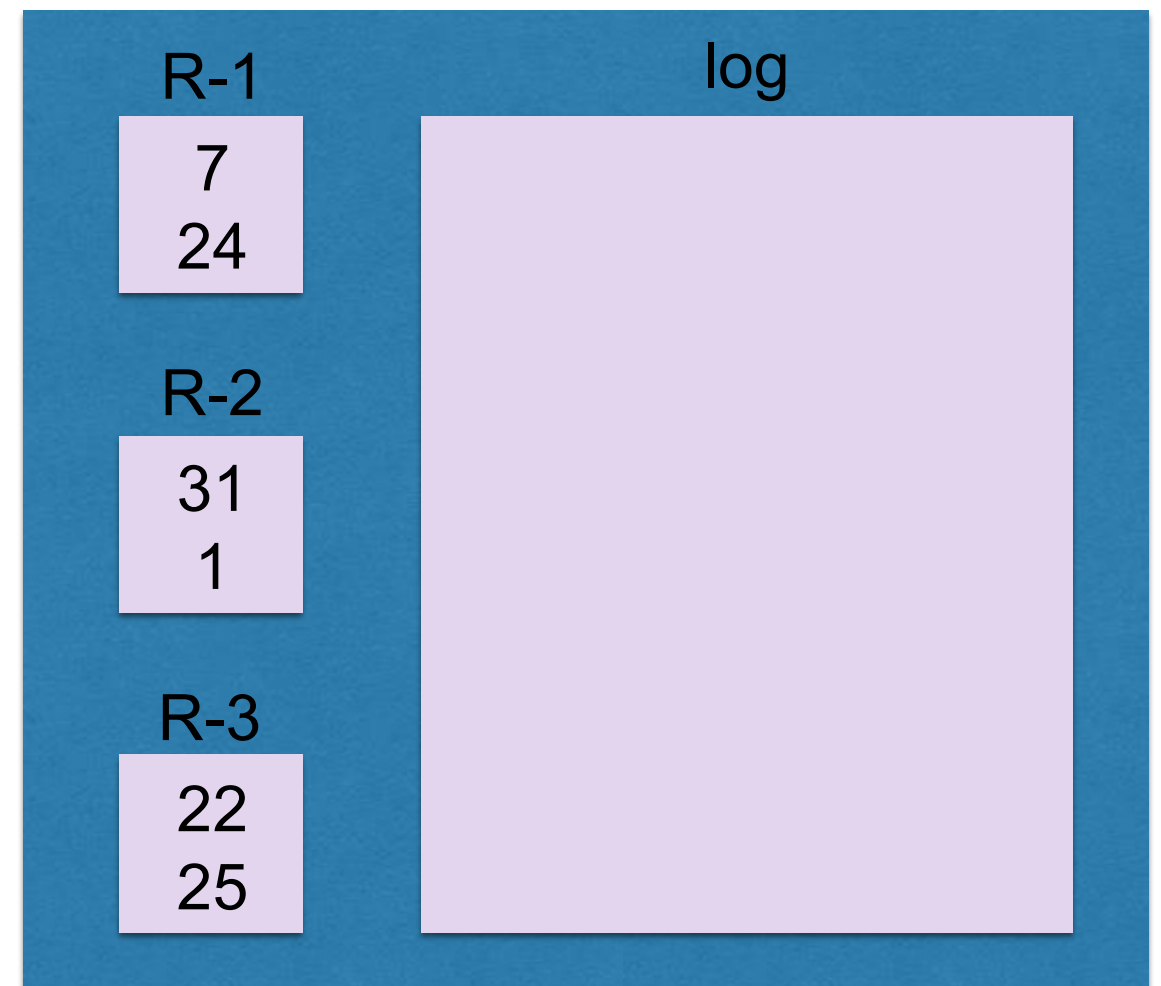
DBMS

T1: voy a empezar

Buffer



Disco



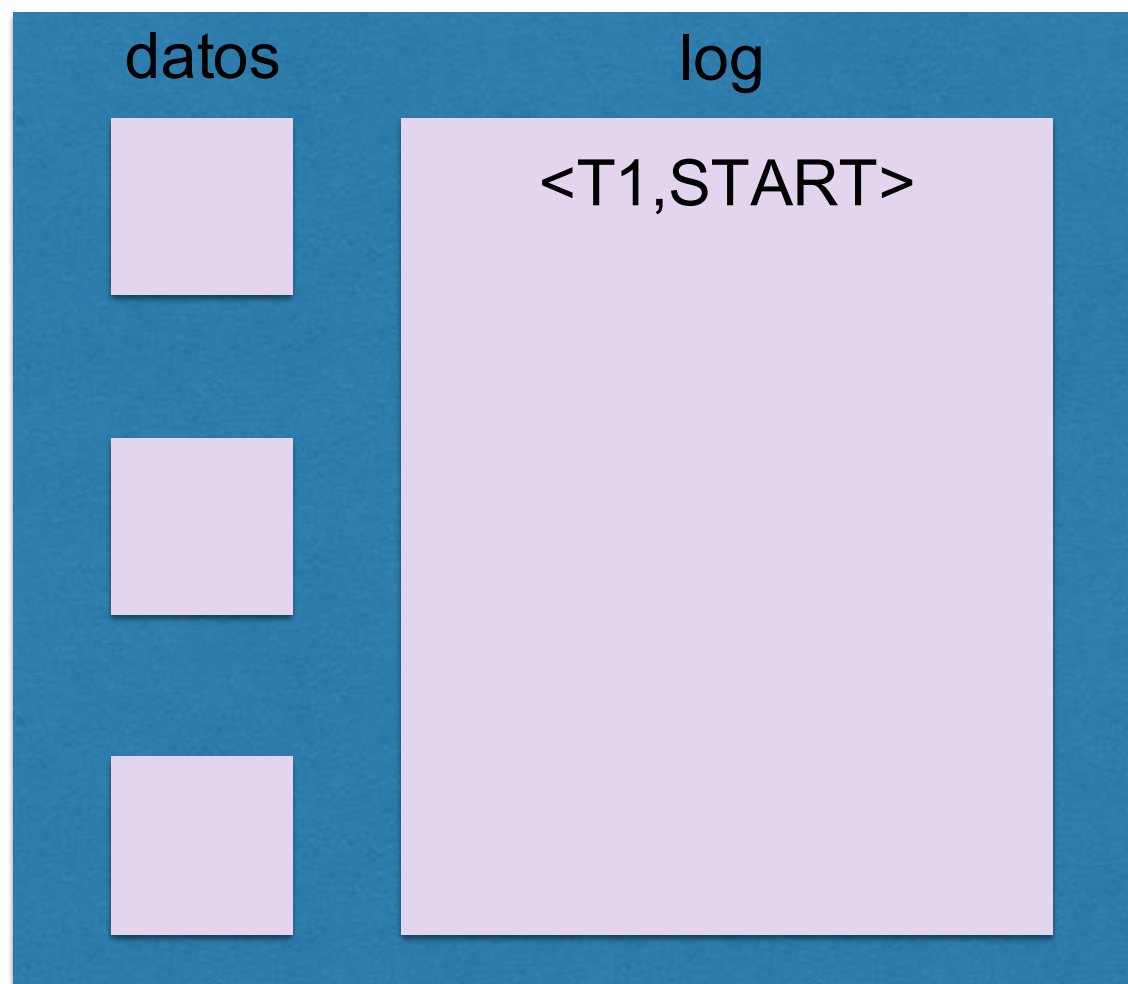
# Redo Logging – en la BD

DBMS

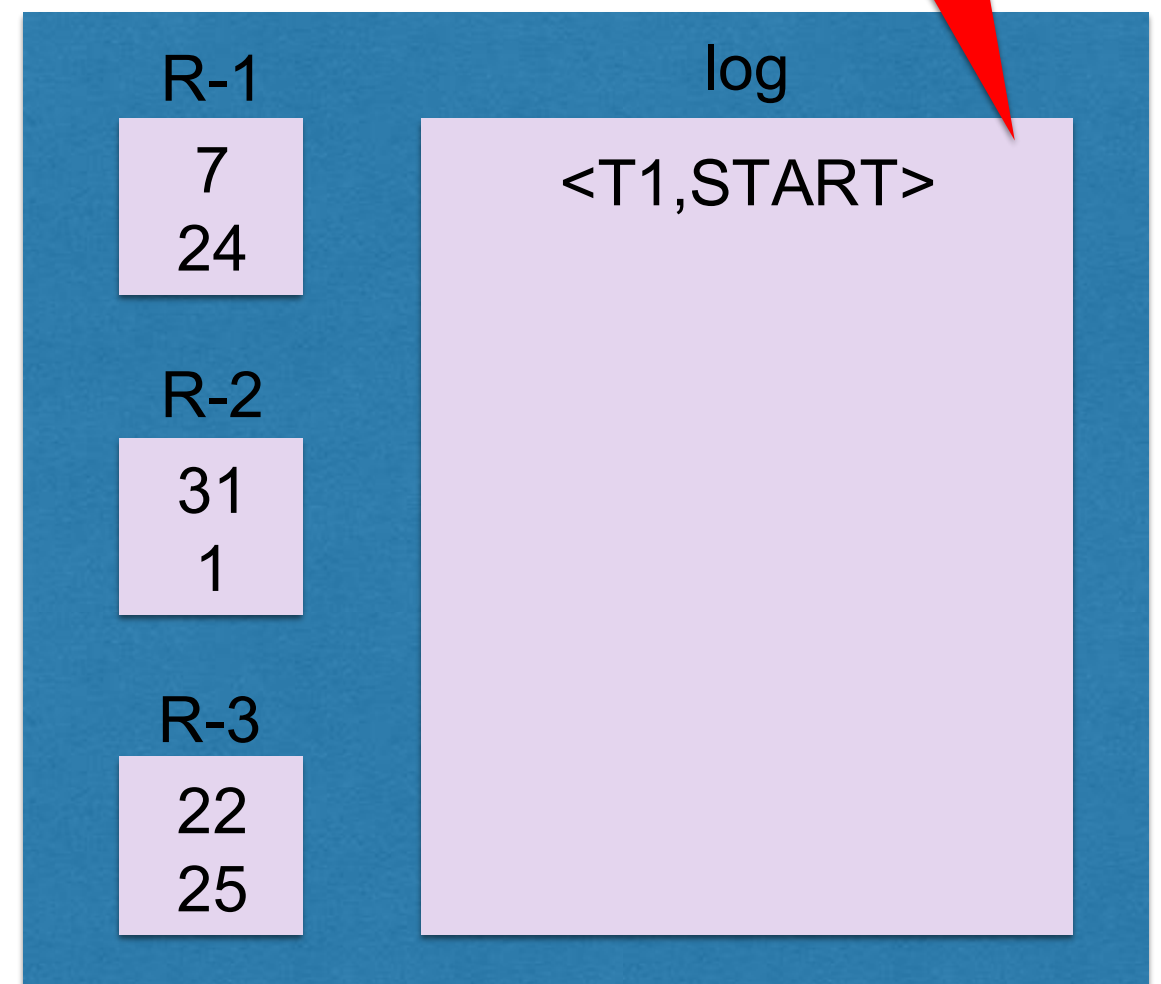
T1: voy a empezar

Incluso puedo escribir  
log al disco al tiro

Buffer



Disco

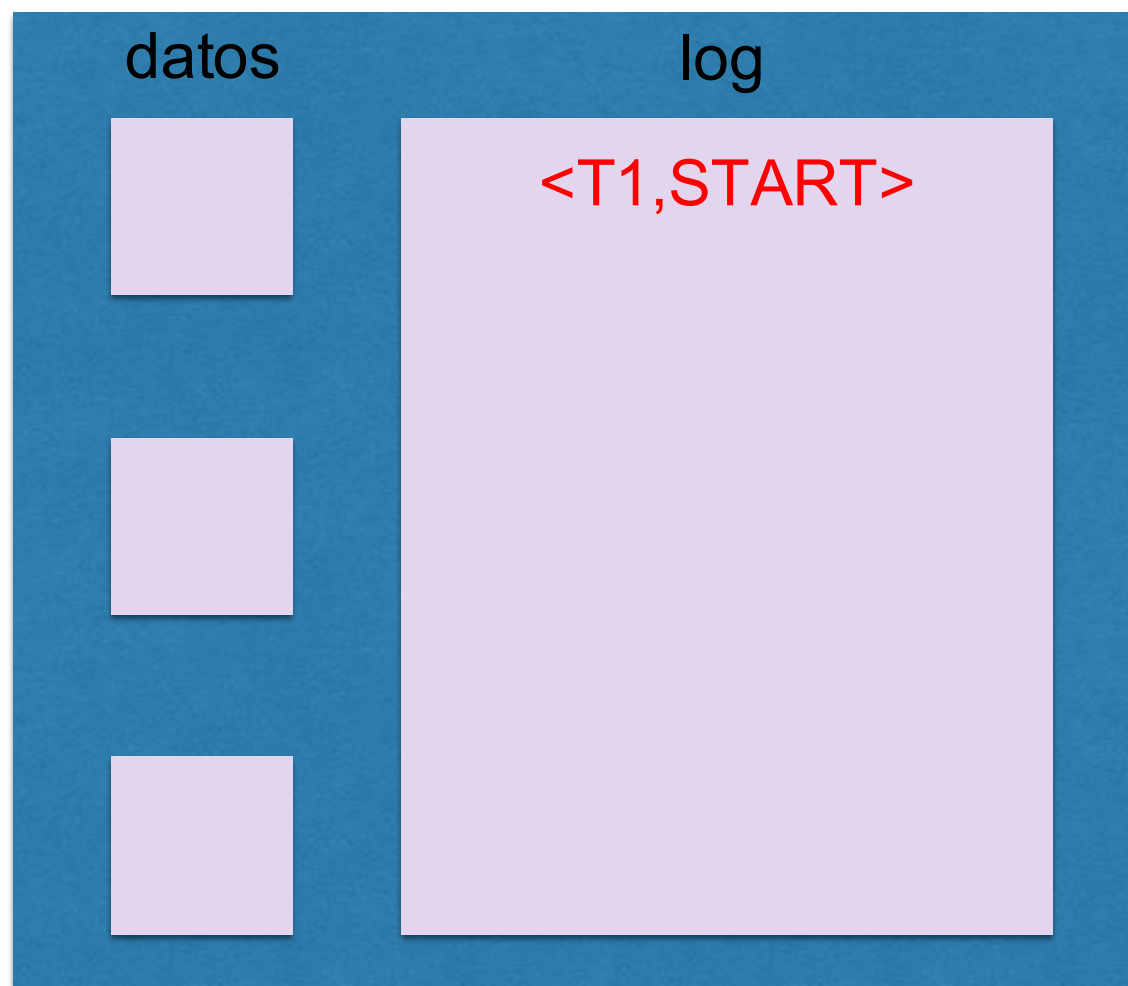


# Redo Logging – en la BD

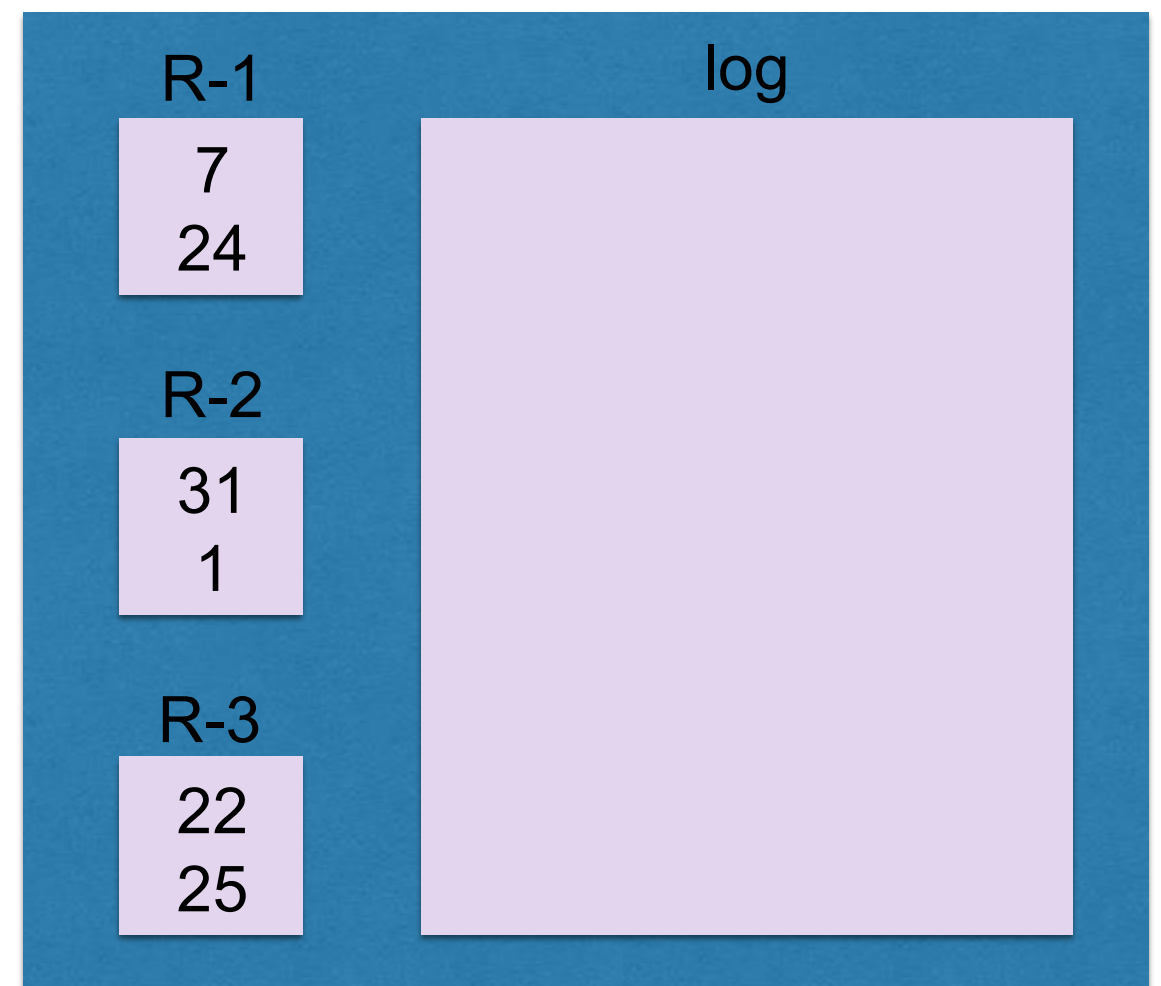
DBMS

T1: Necesito primera  
tupla de página 2 de R!

Buffer



Disco

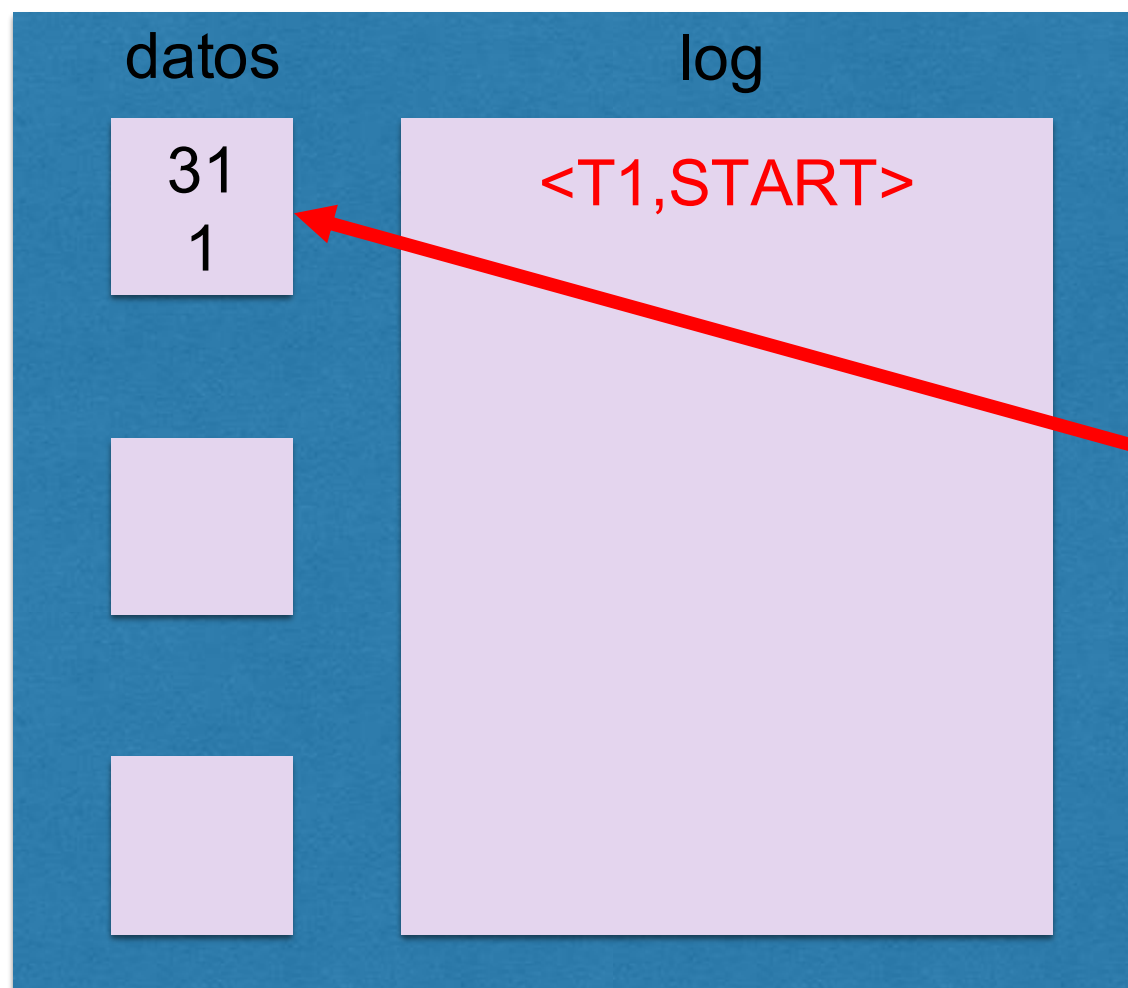


# Redo Logging – en la BD

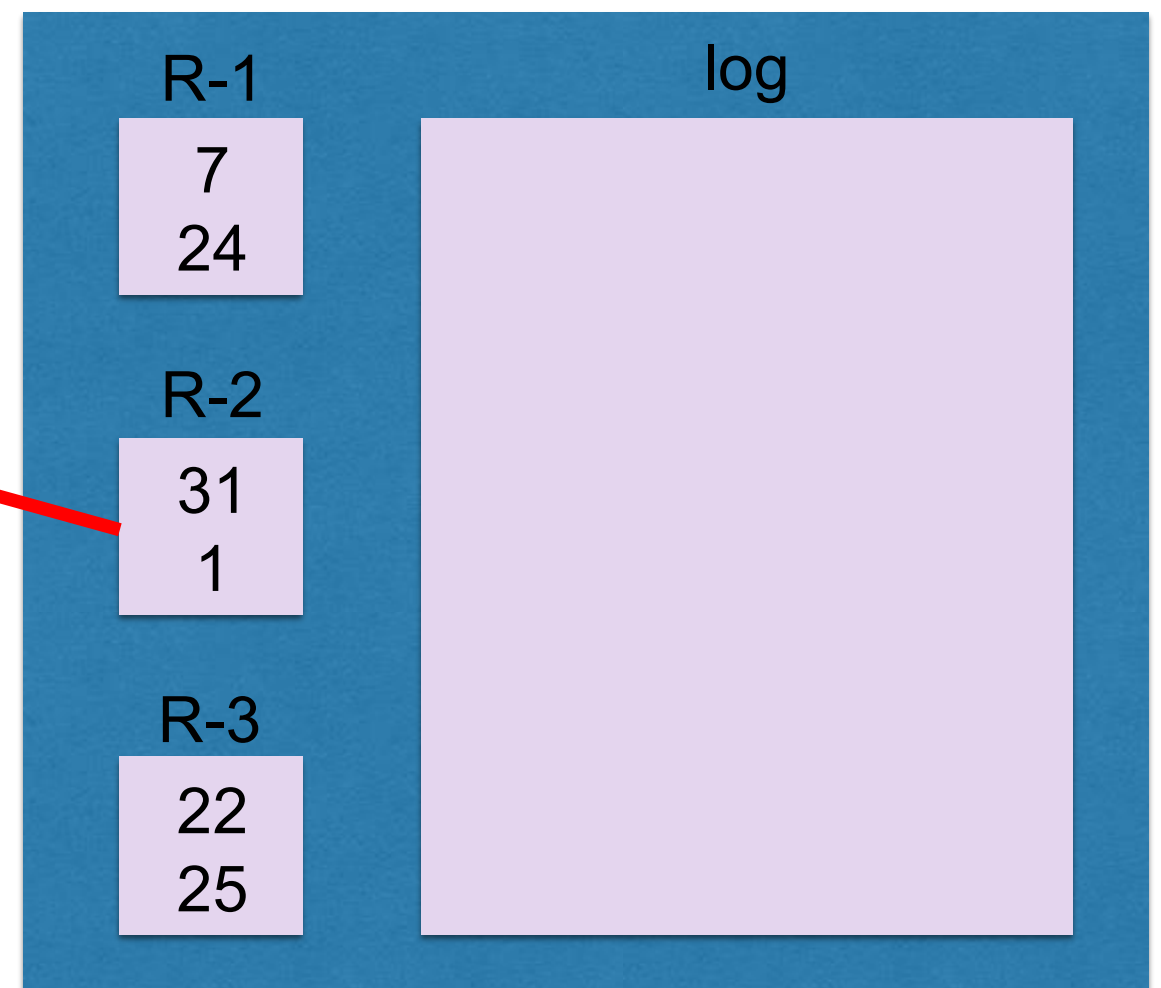
DBMS

T1: Necesito primera  
tupla de página 2 de R!

Buffer



Disco

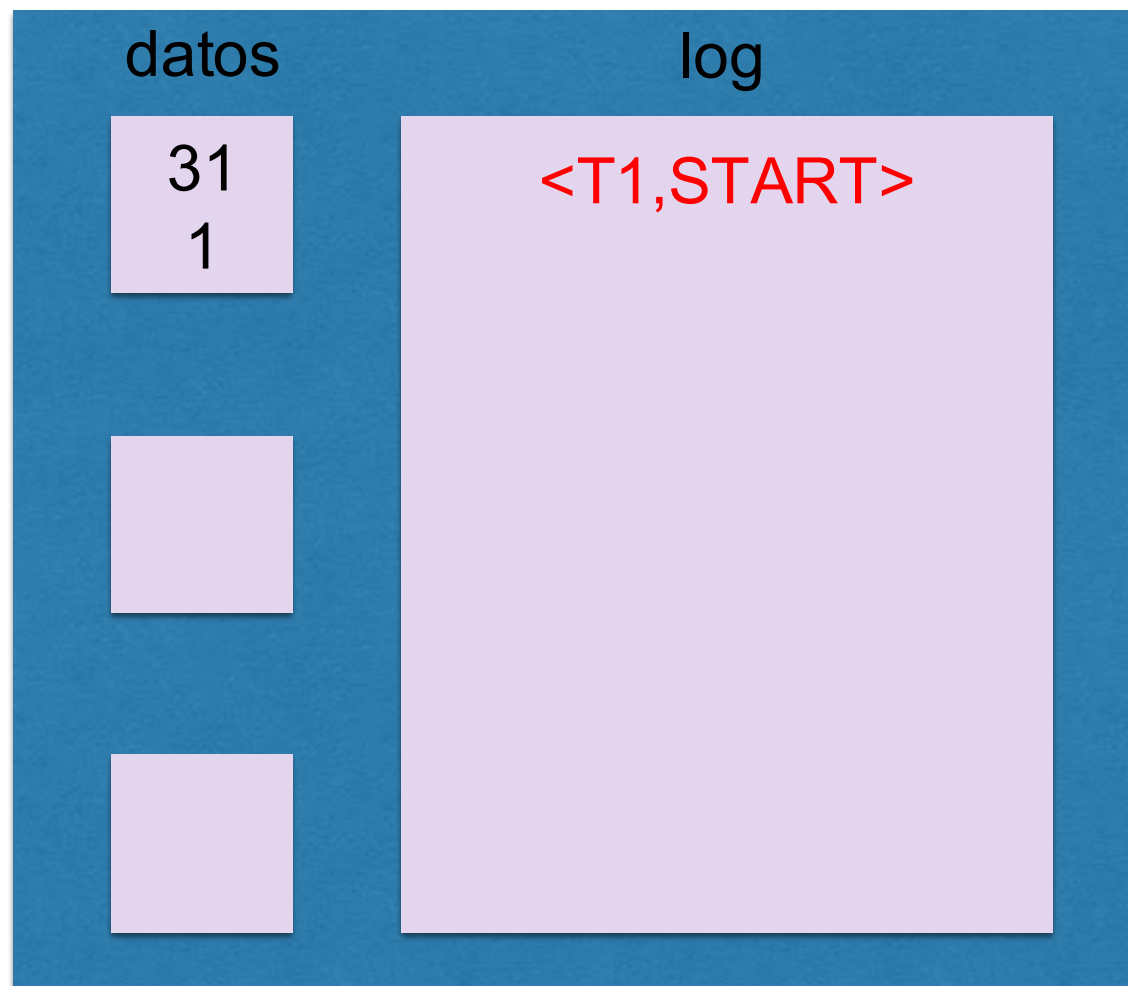


# Redo Logging – en la BD

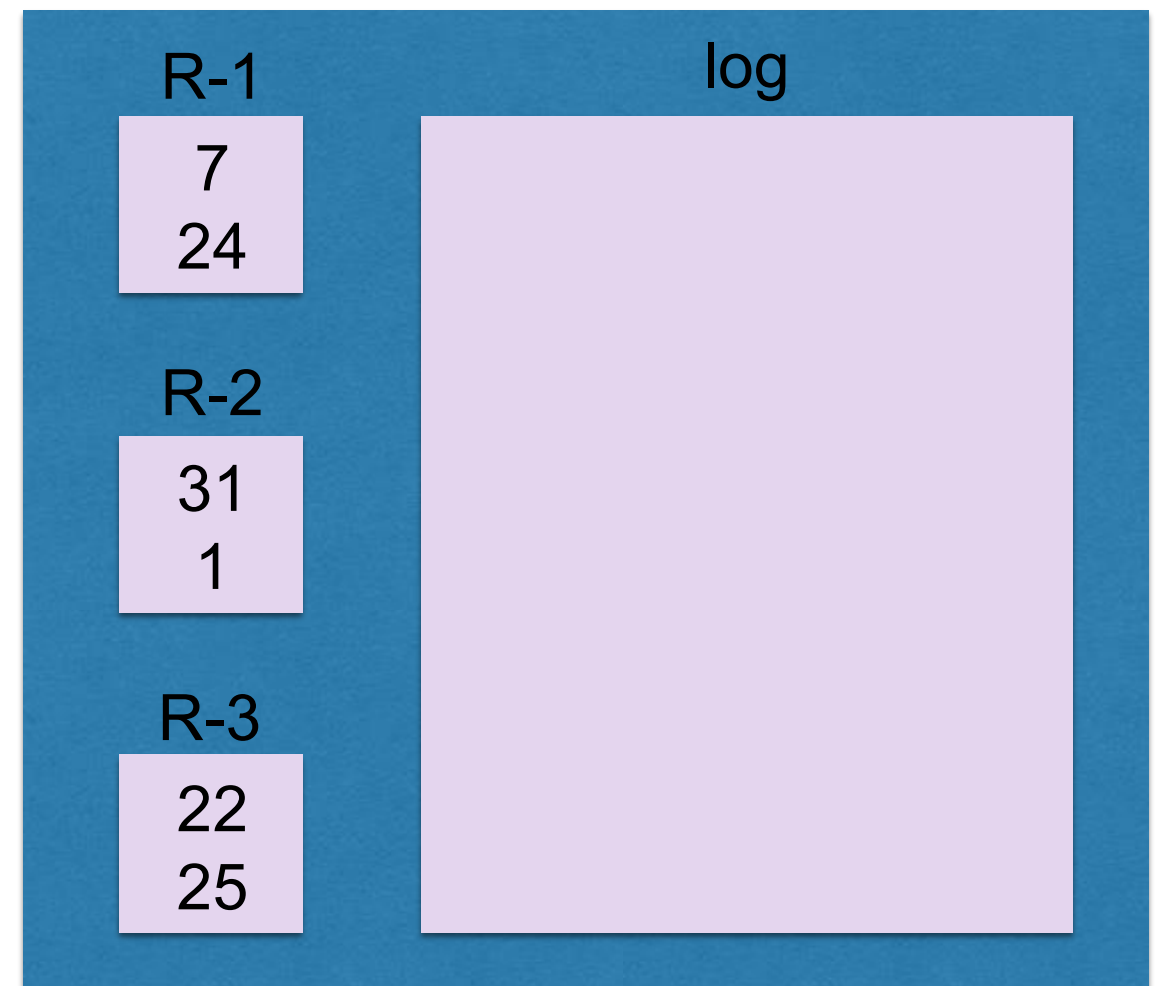
DBMS

T1: Cambio 31 a 99!

Buffer



Disco

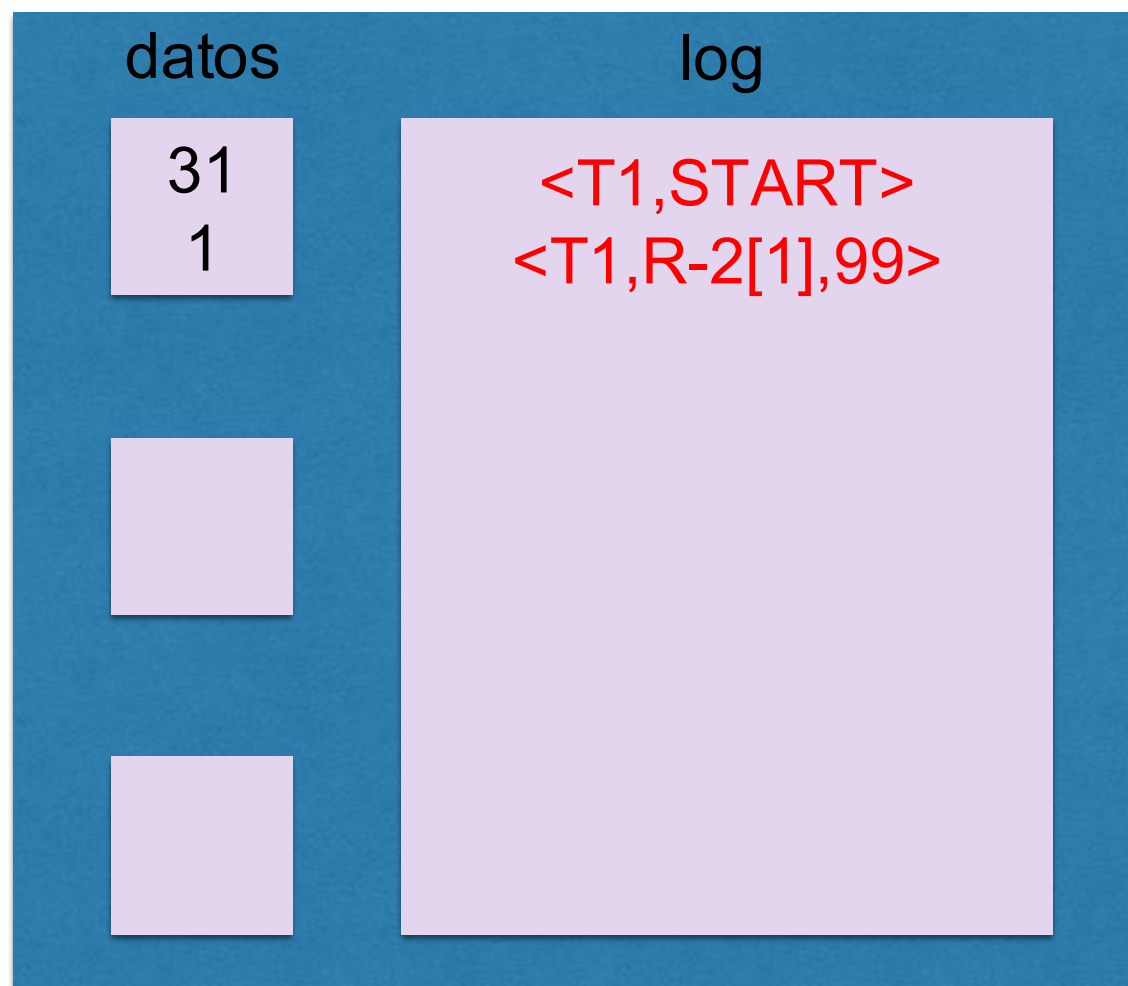


# Redo Logging – en la BD

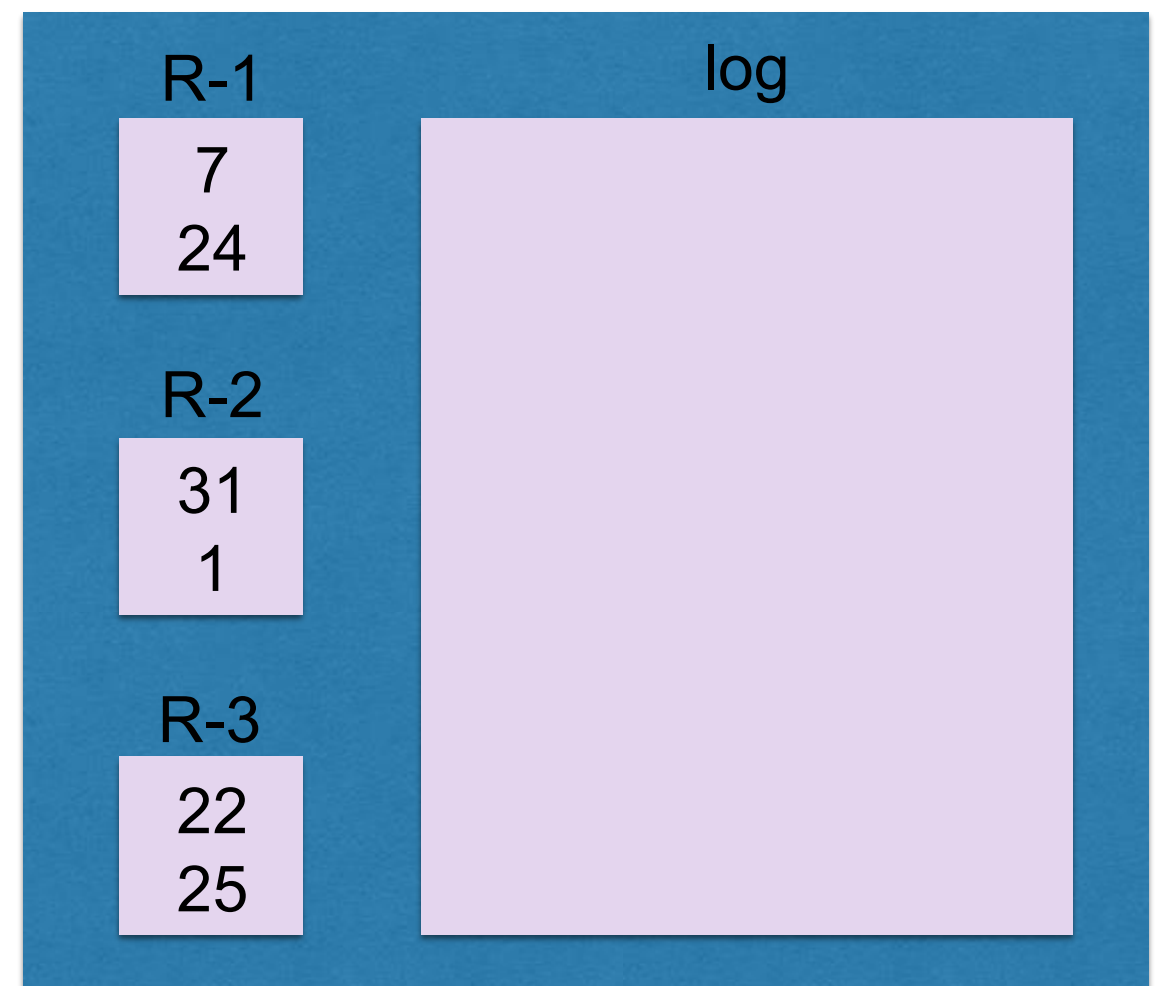
DBMS

T1: Cambio 31 a 99!

Buffer



Disco



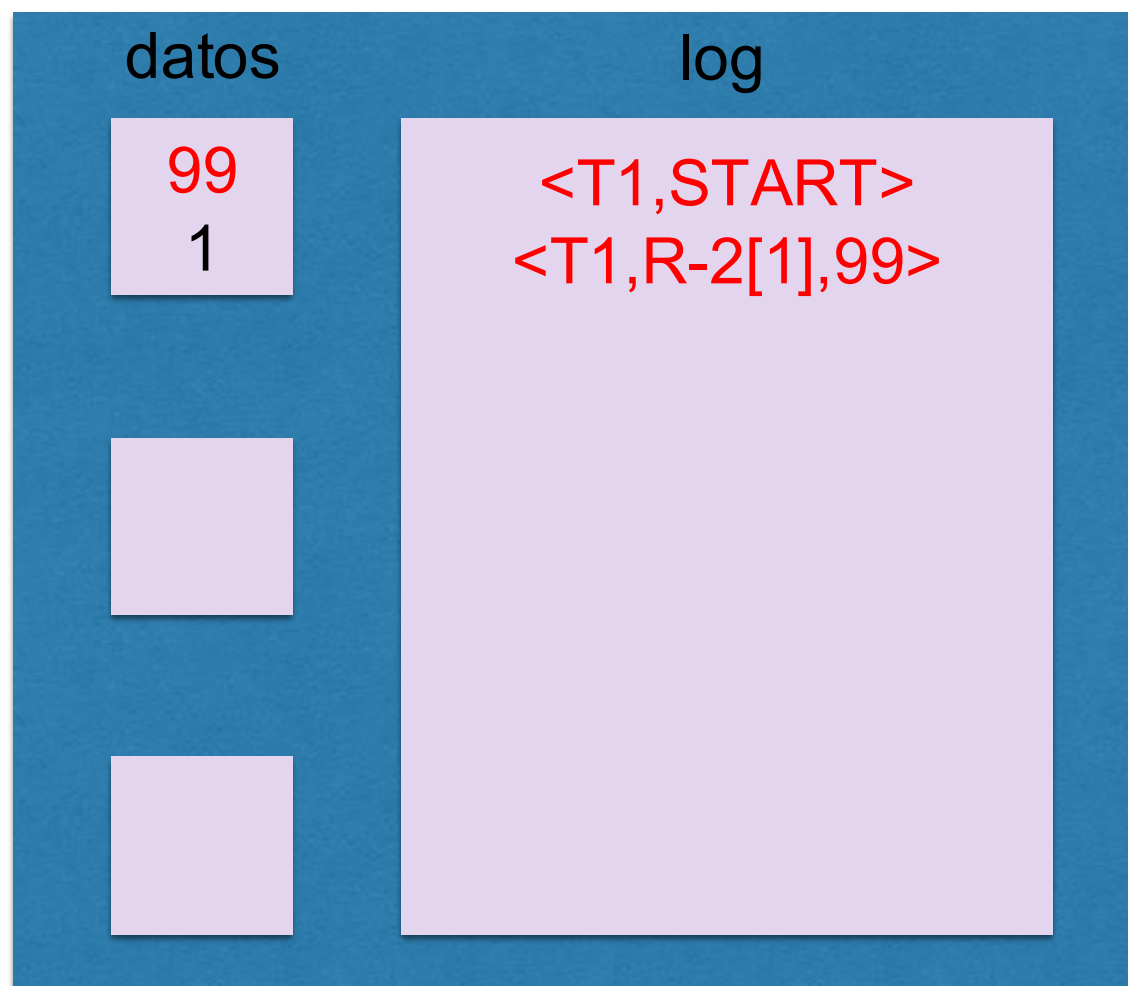


# Redo Logging – en la BD

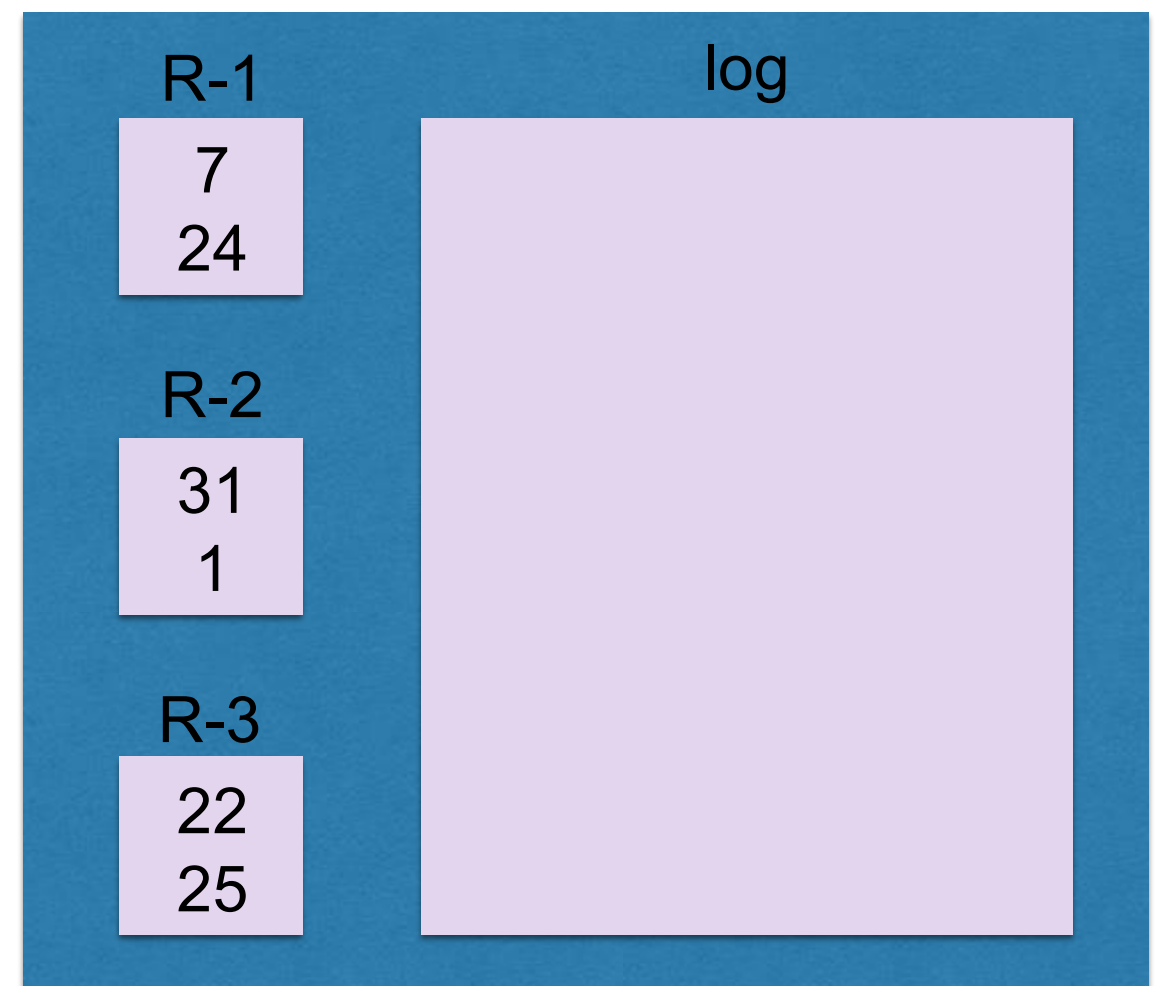
DBMS

T1: Cambio 31 a 99!

Buffer



Disco

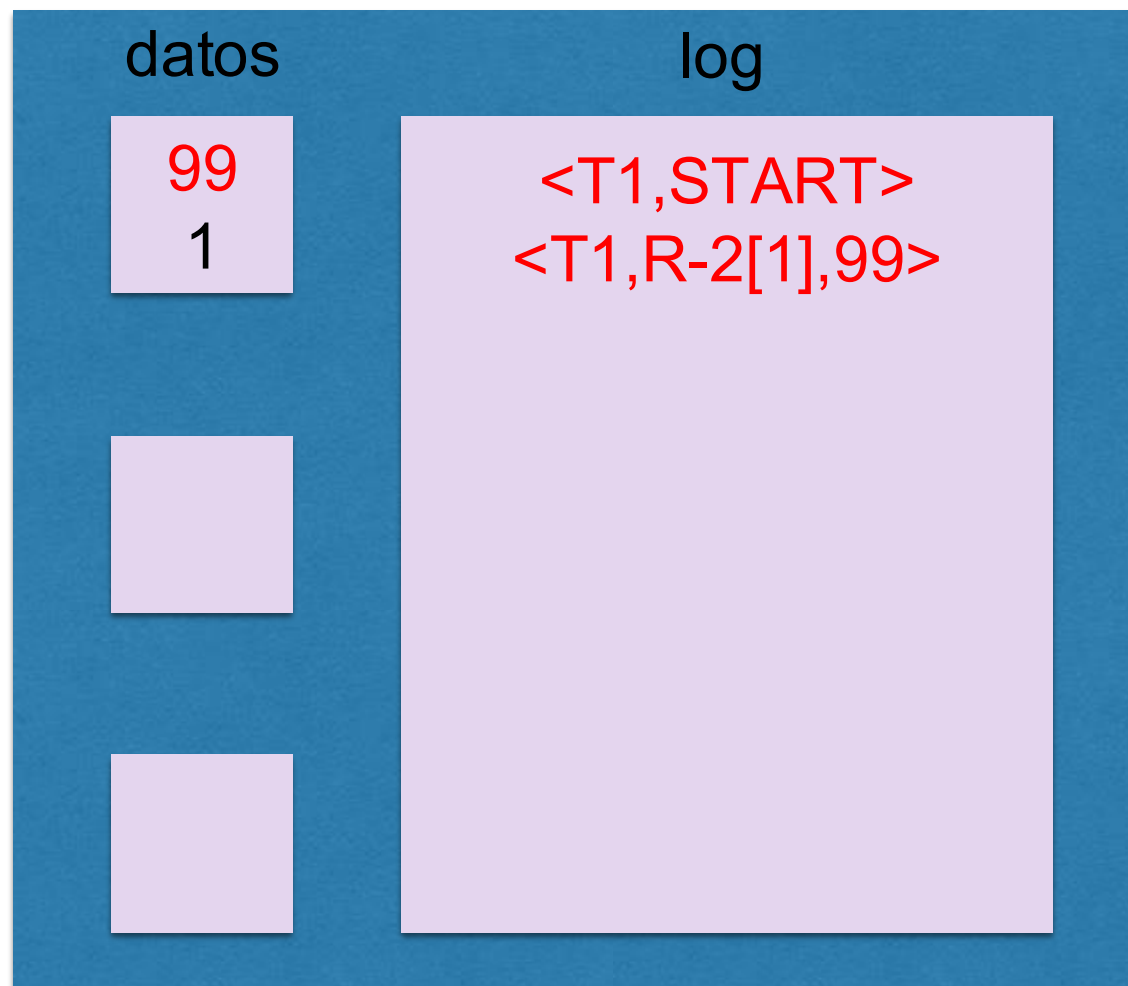


# Redo Logging – en la BD

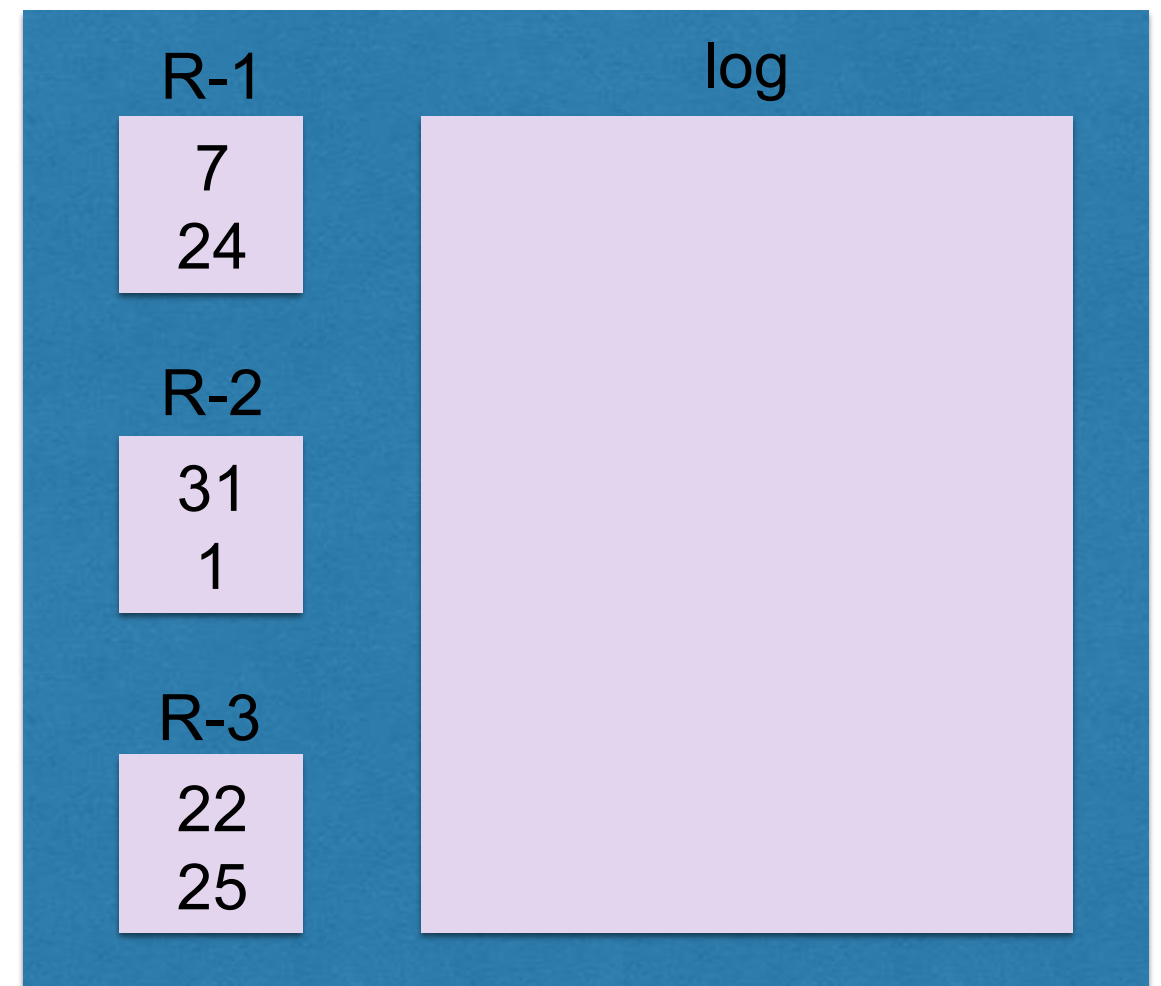
DBMS

T1: Cambio 99 a 23!

Buffer



Disco

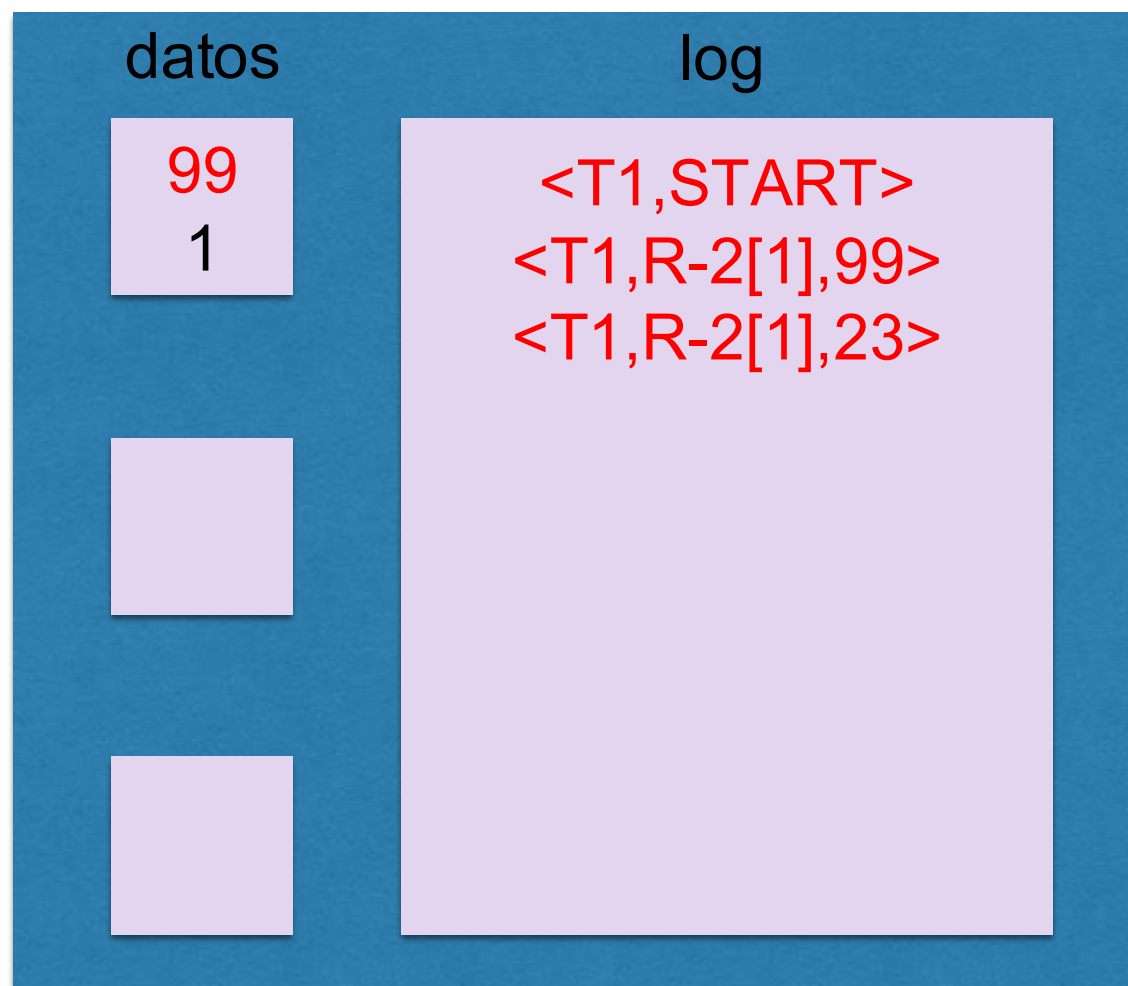


# Redo Logging – en la BD

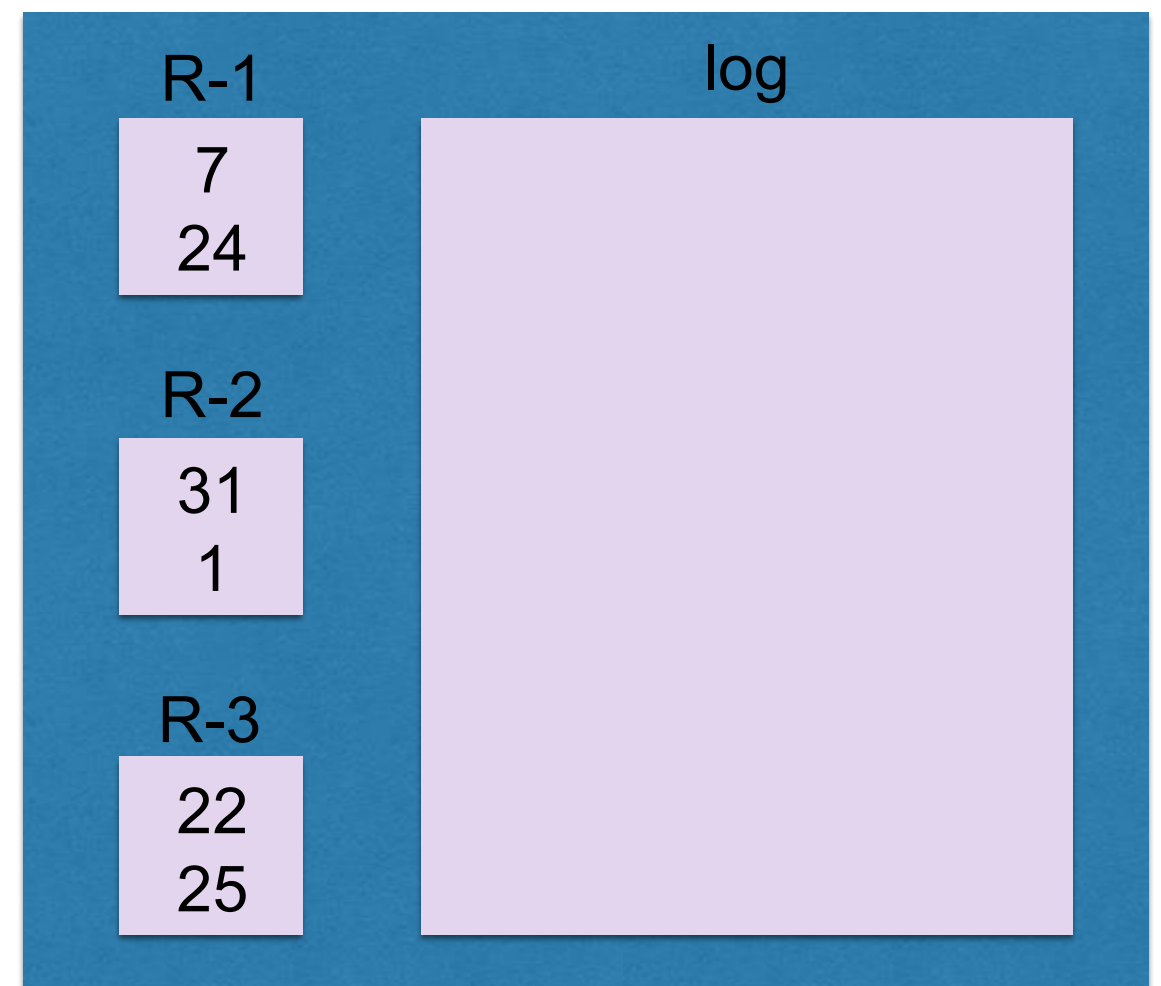
DBMS

T1: Cambio 99 a 23!

Buffer



Disco

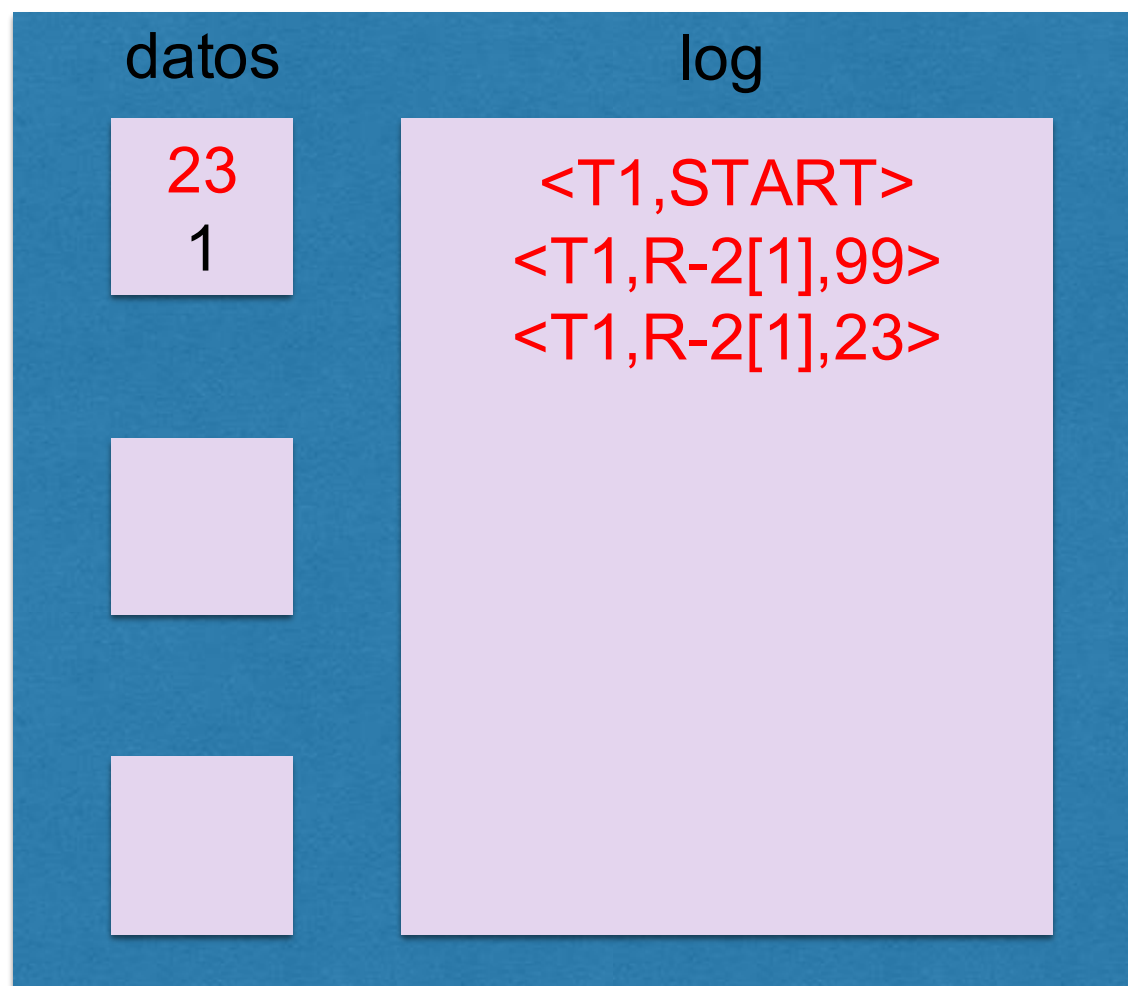


# Redo Logging – en la BD

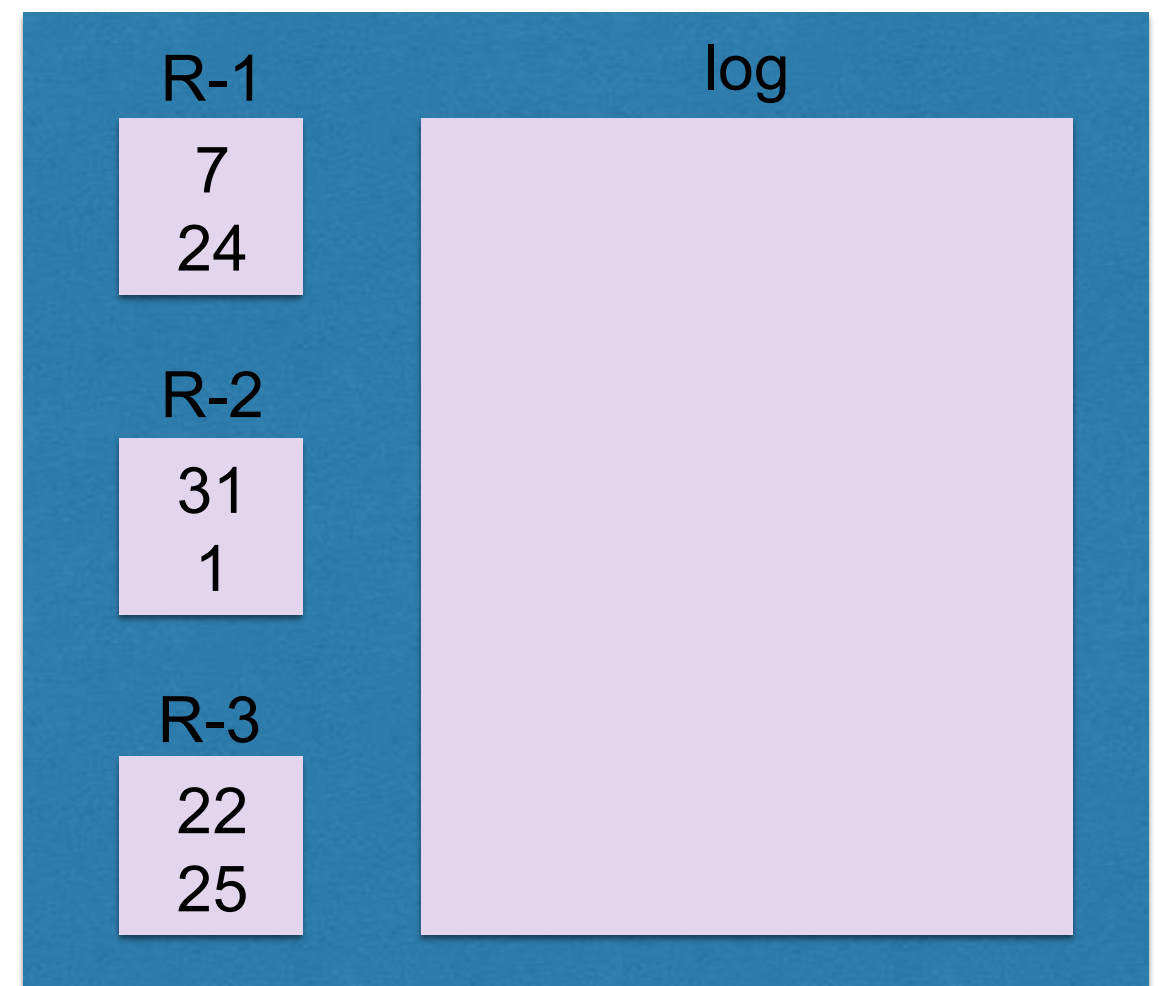
DBMS

T1: Cambio 99 a 23!

Buffer



Disco

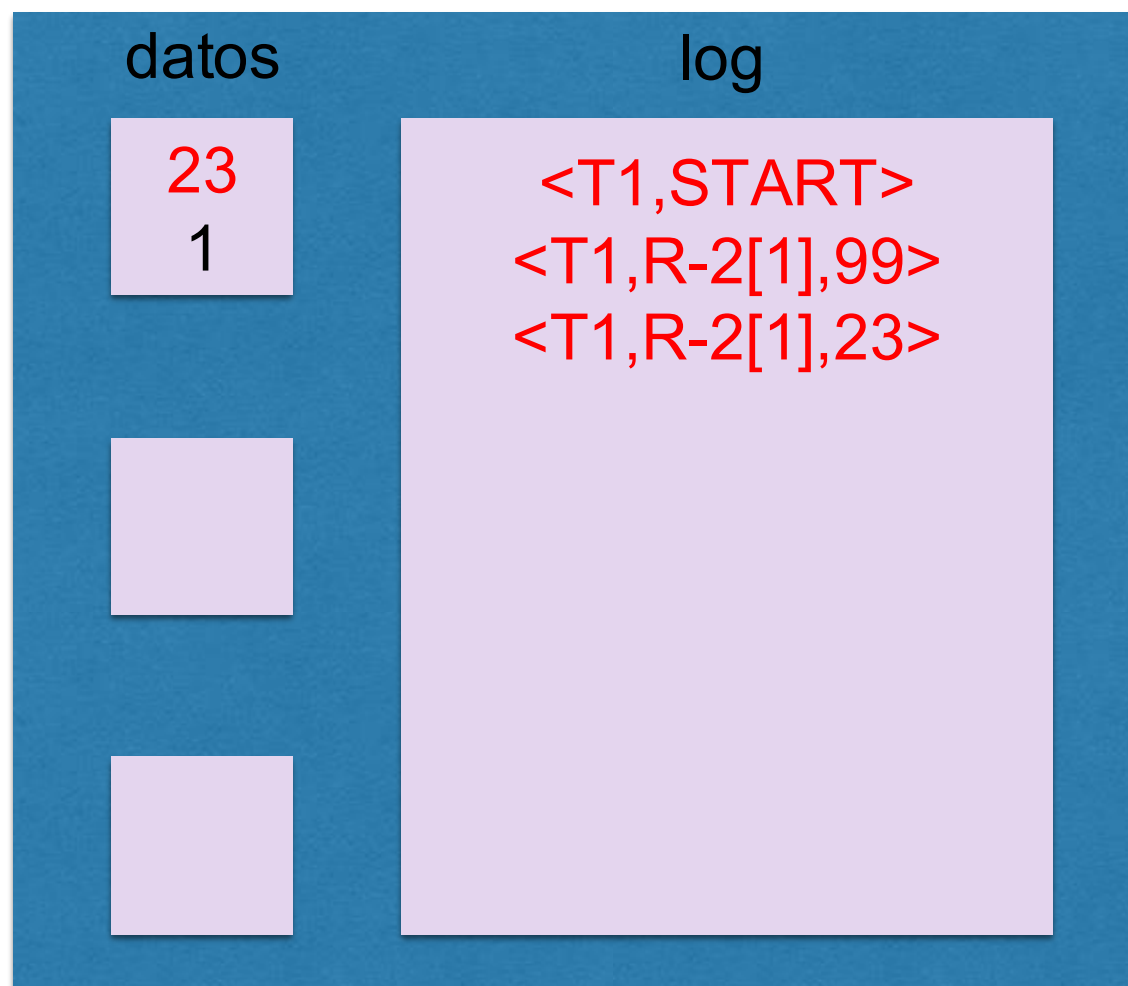


# Redo Logging – en la BD

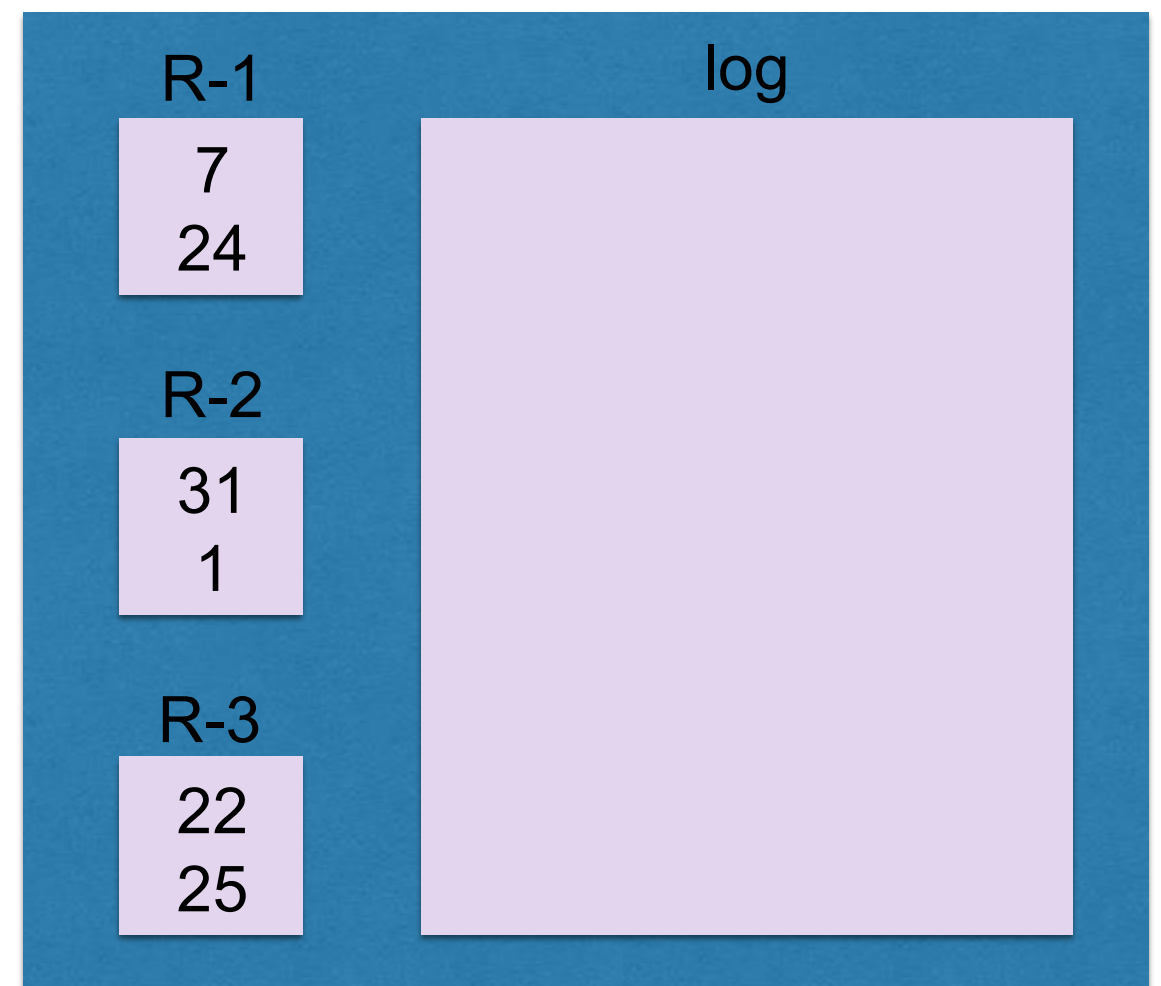
DBMS

T1: estoy listo!

Buffer



Disco

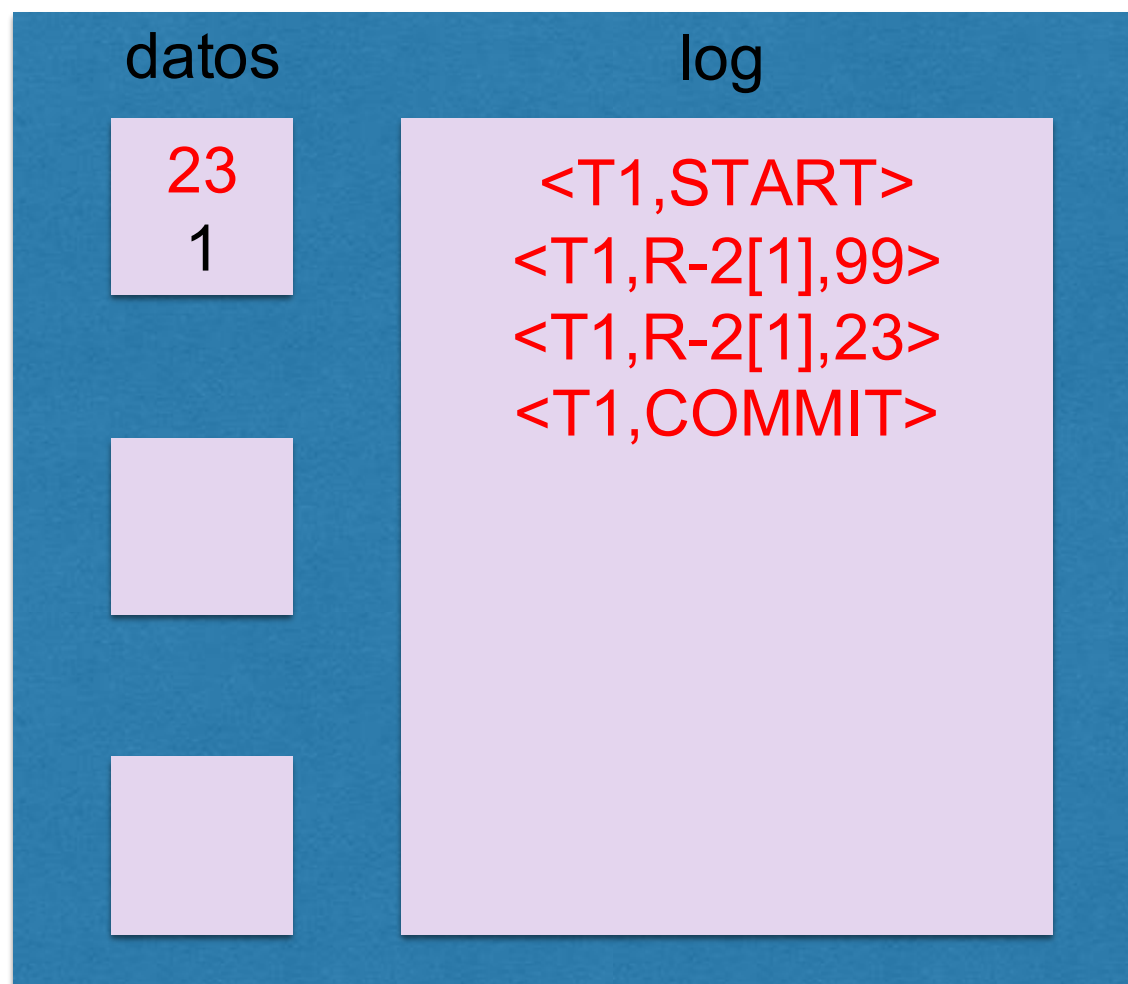


# Redo Logging – en la BD

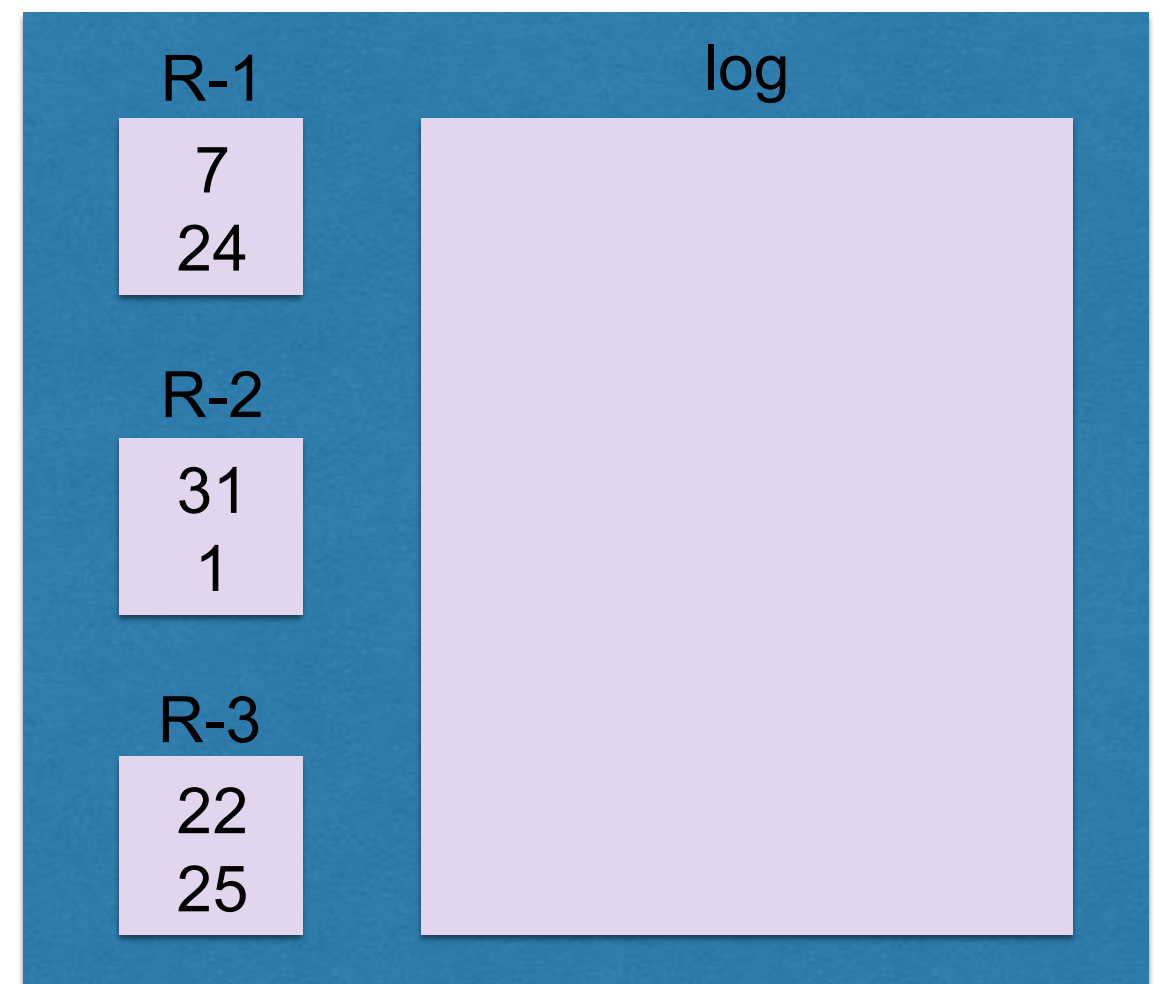
DBMS

T1: estoy listo!

Buffer



Disco

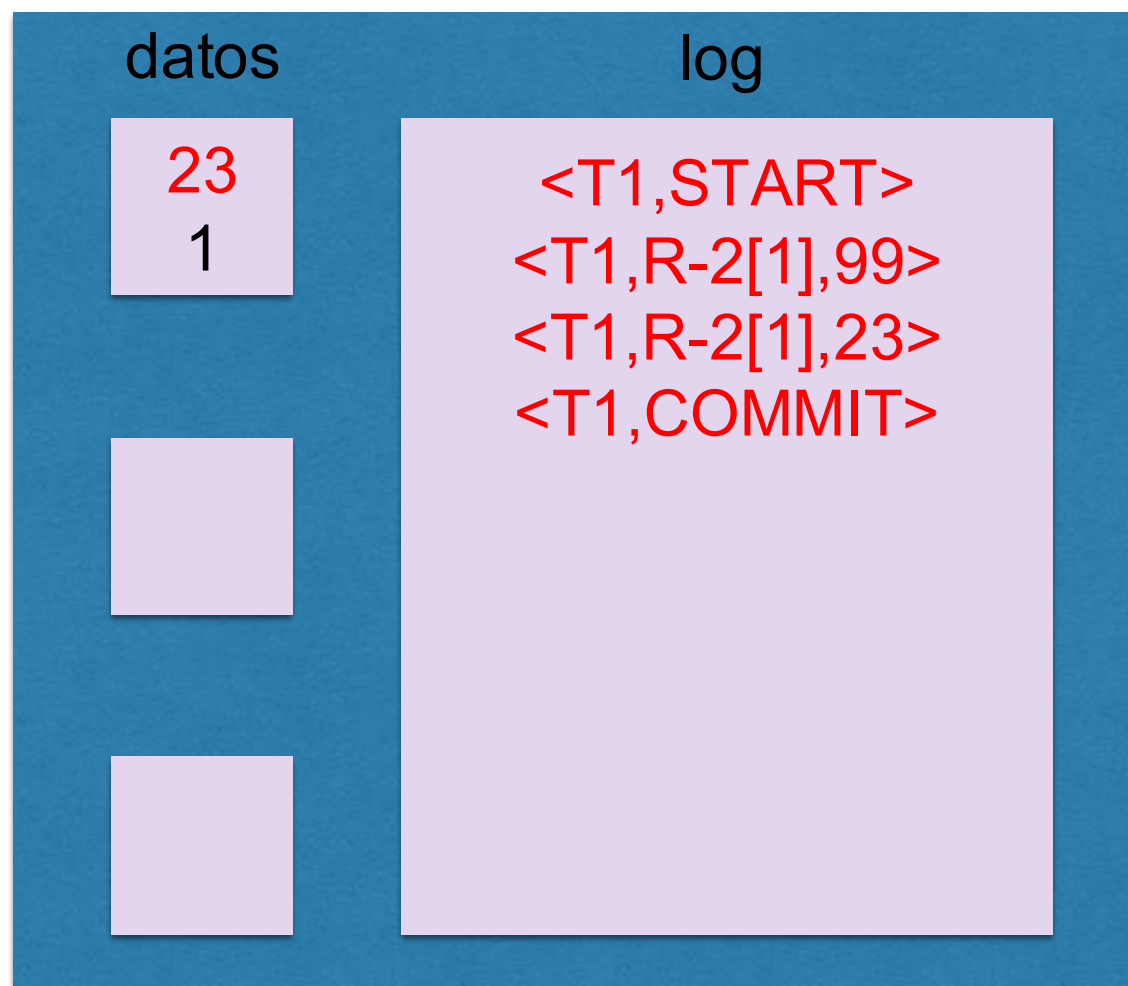


# Redo Logging – en la BD

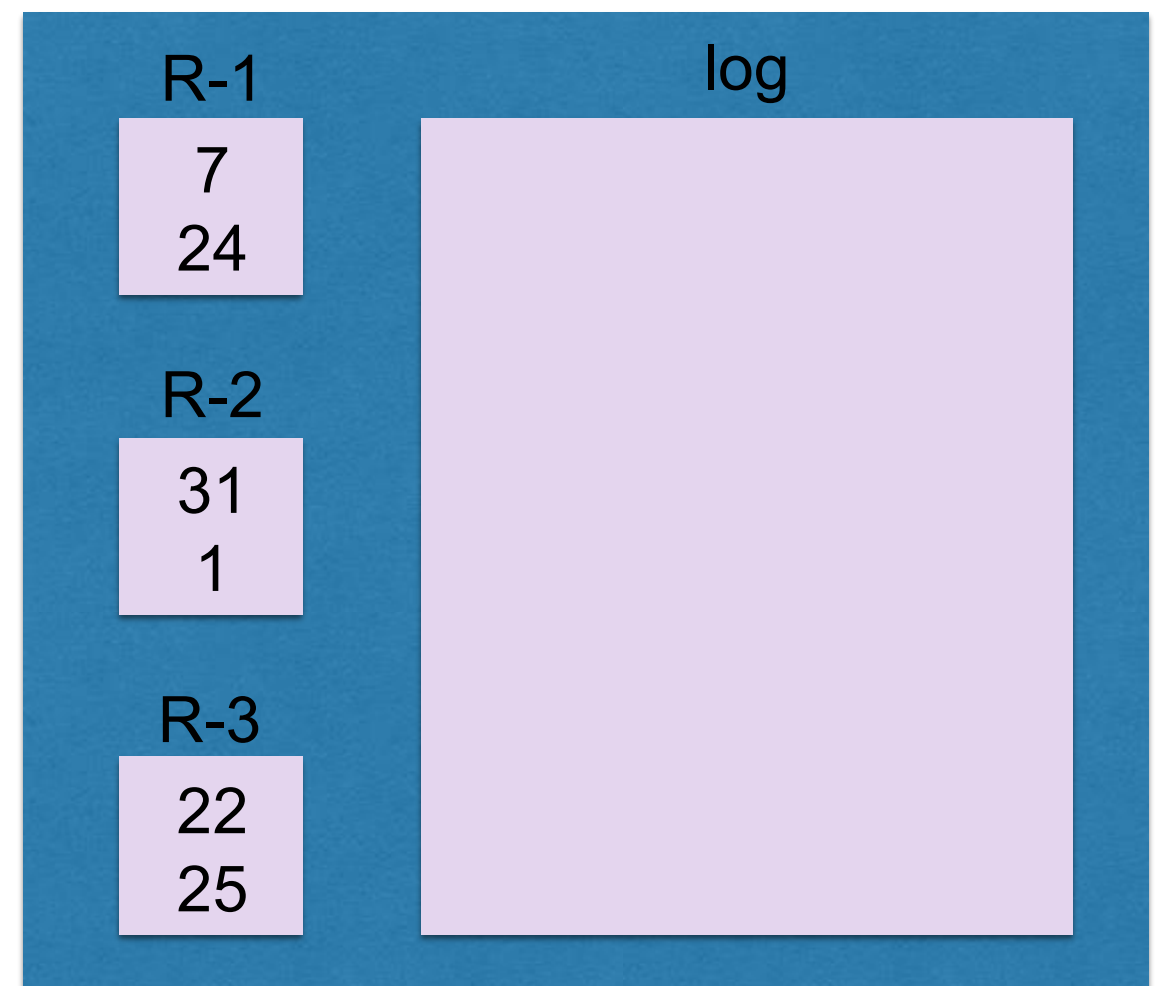
DBMS

Vamos al disco!

Buffer



Disco

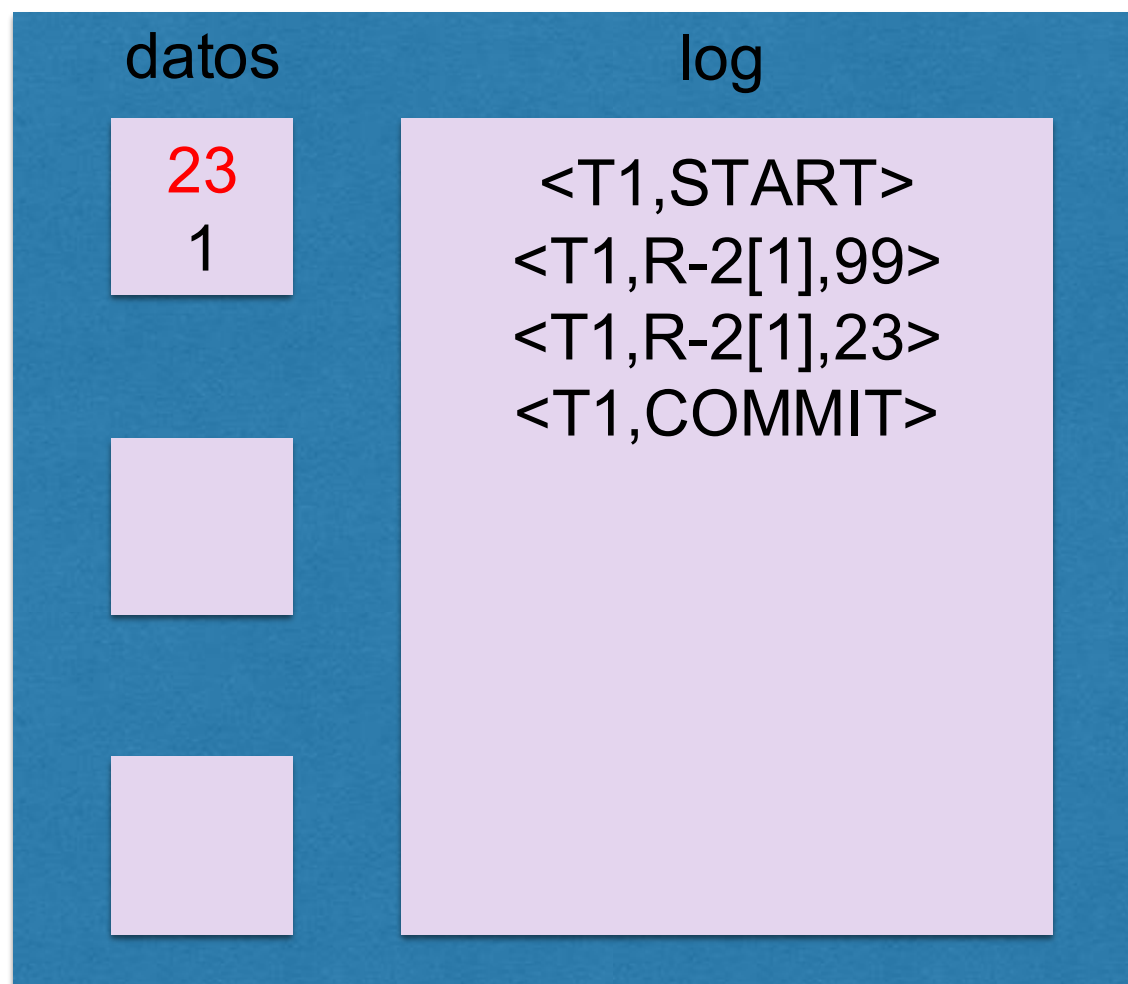


# Redo Logging – en la BD

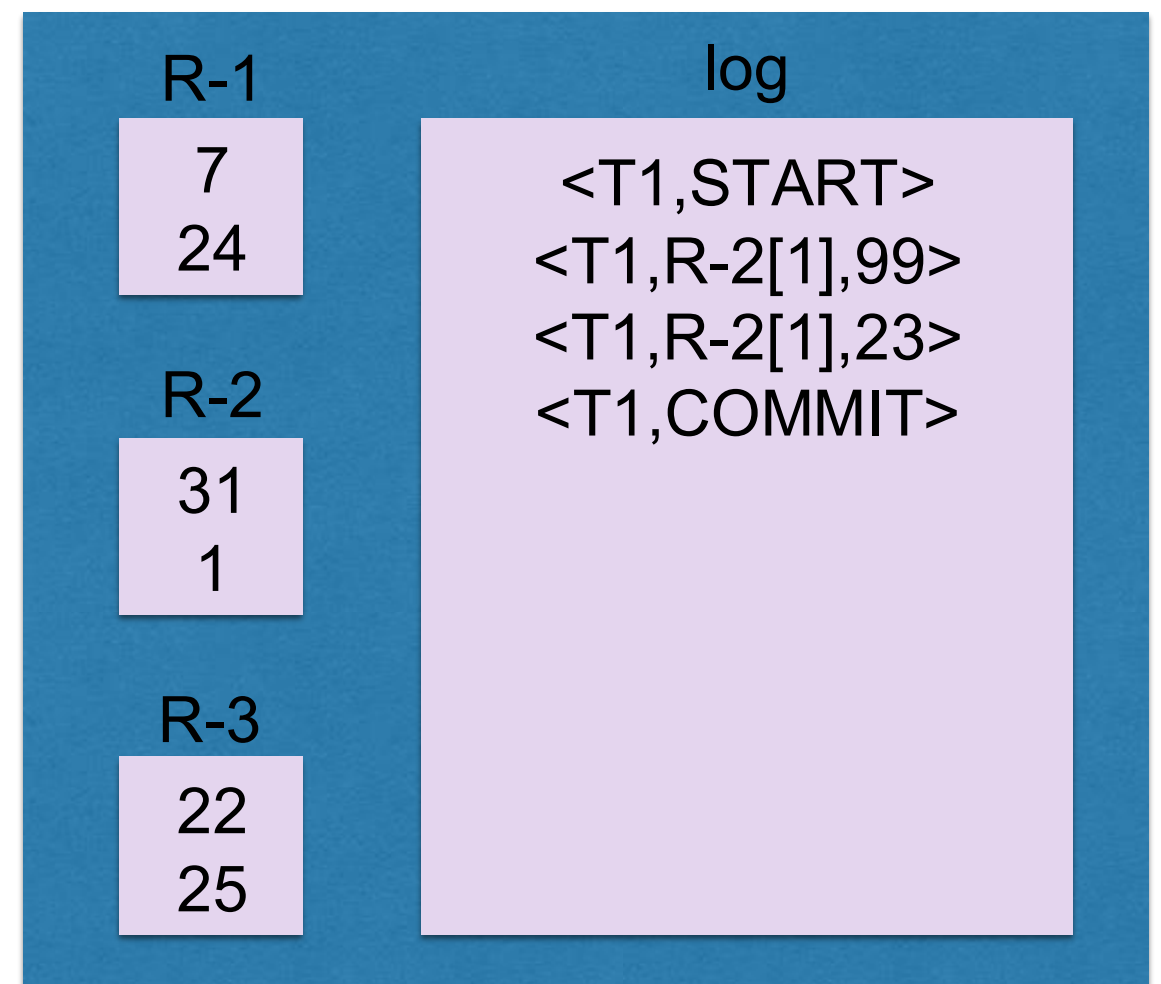
DBMS

Vamos al disco!

Buffer



Disco



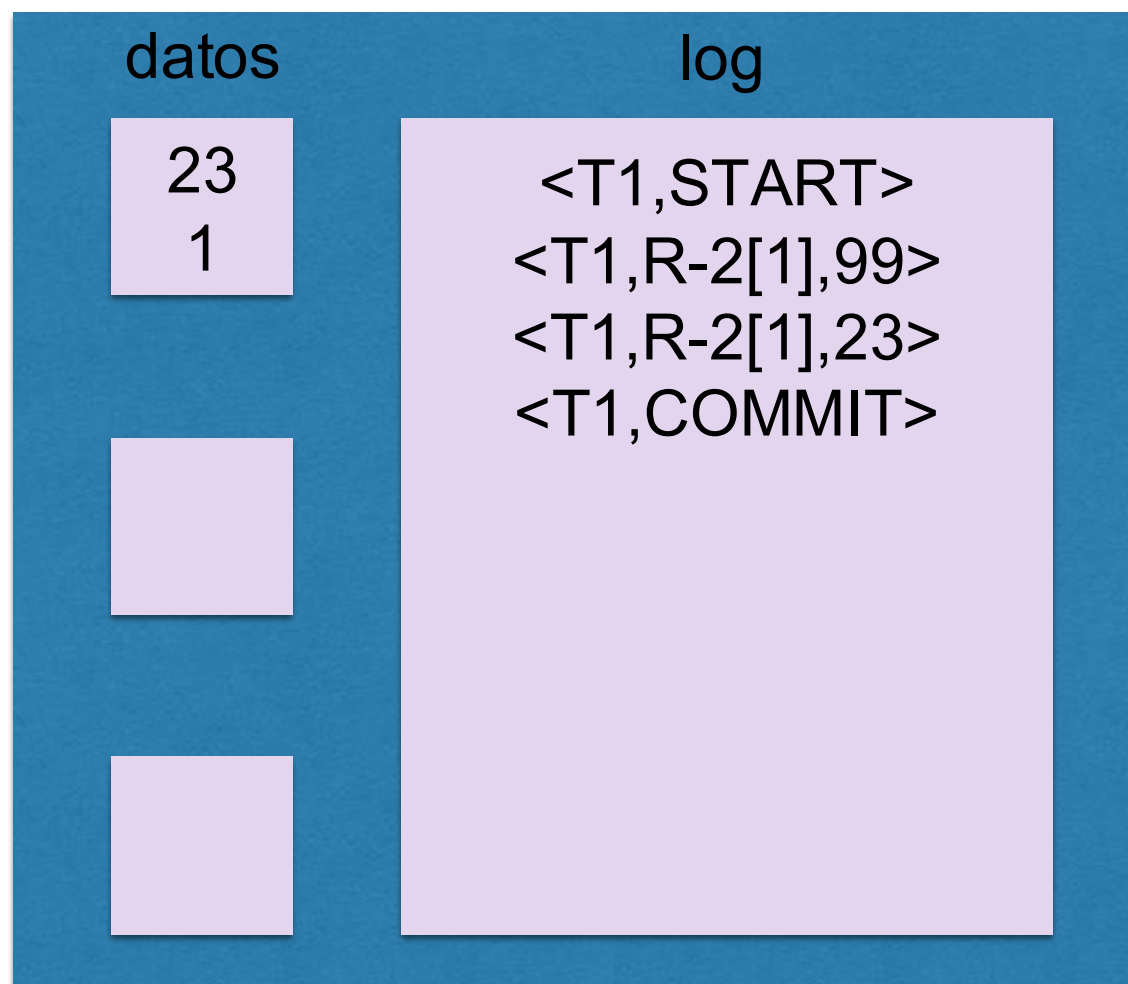


# Redo Logging – en la BD

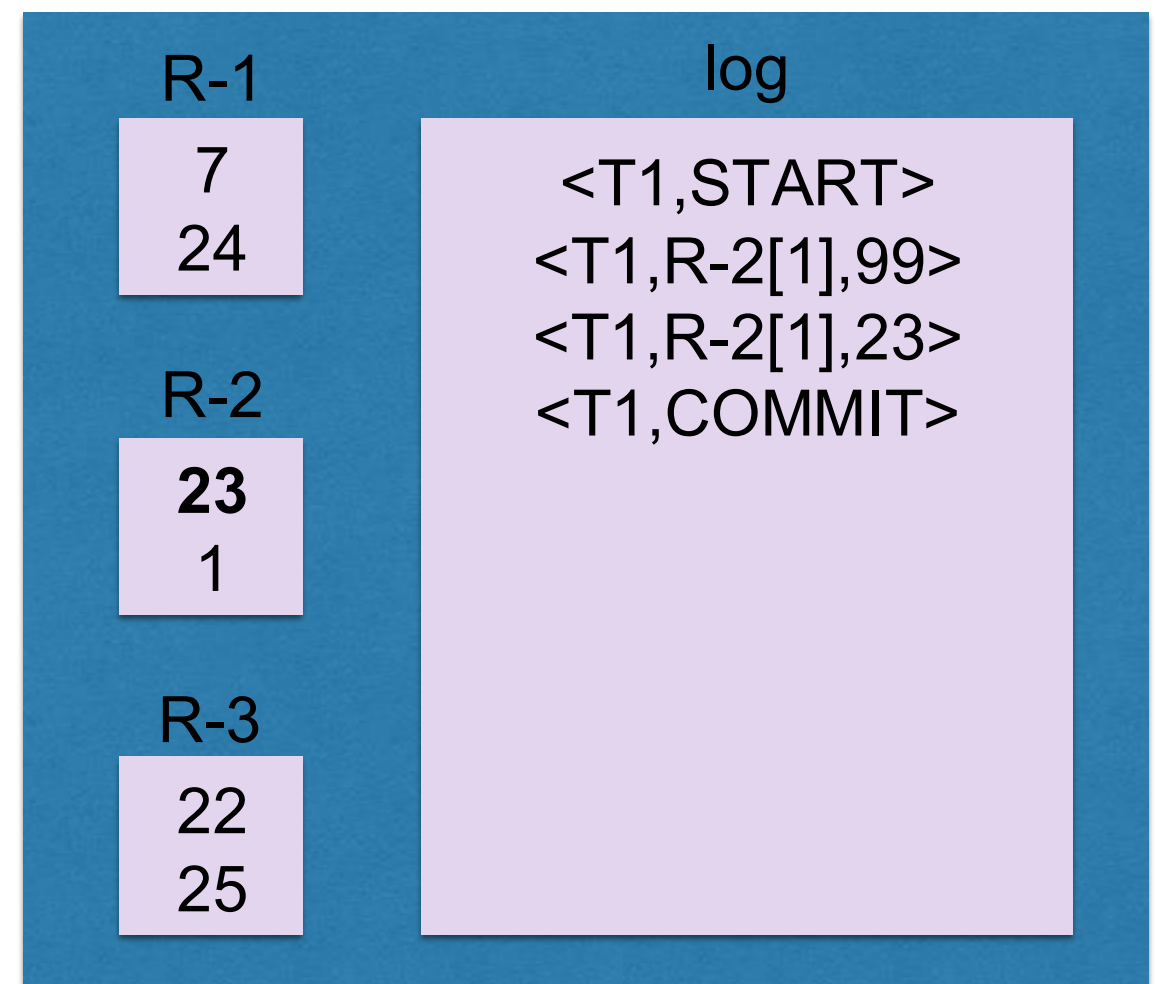
DBMS

Actualizo los datos

Buffer



Disco

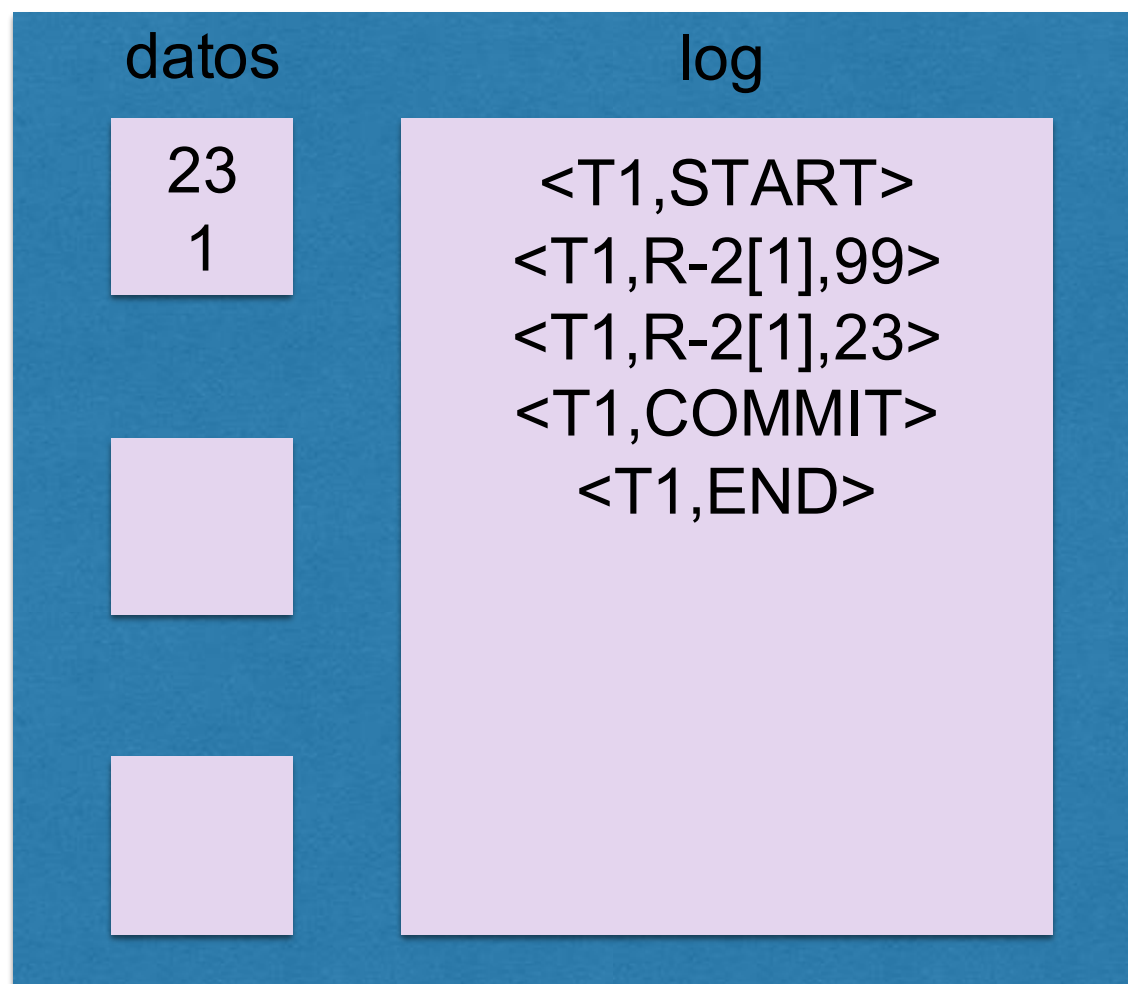


# Redo Logging – en la BD

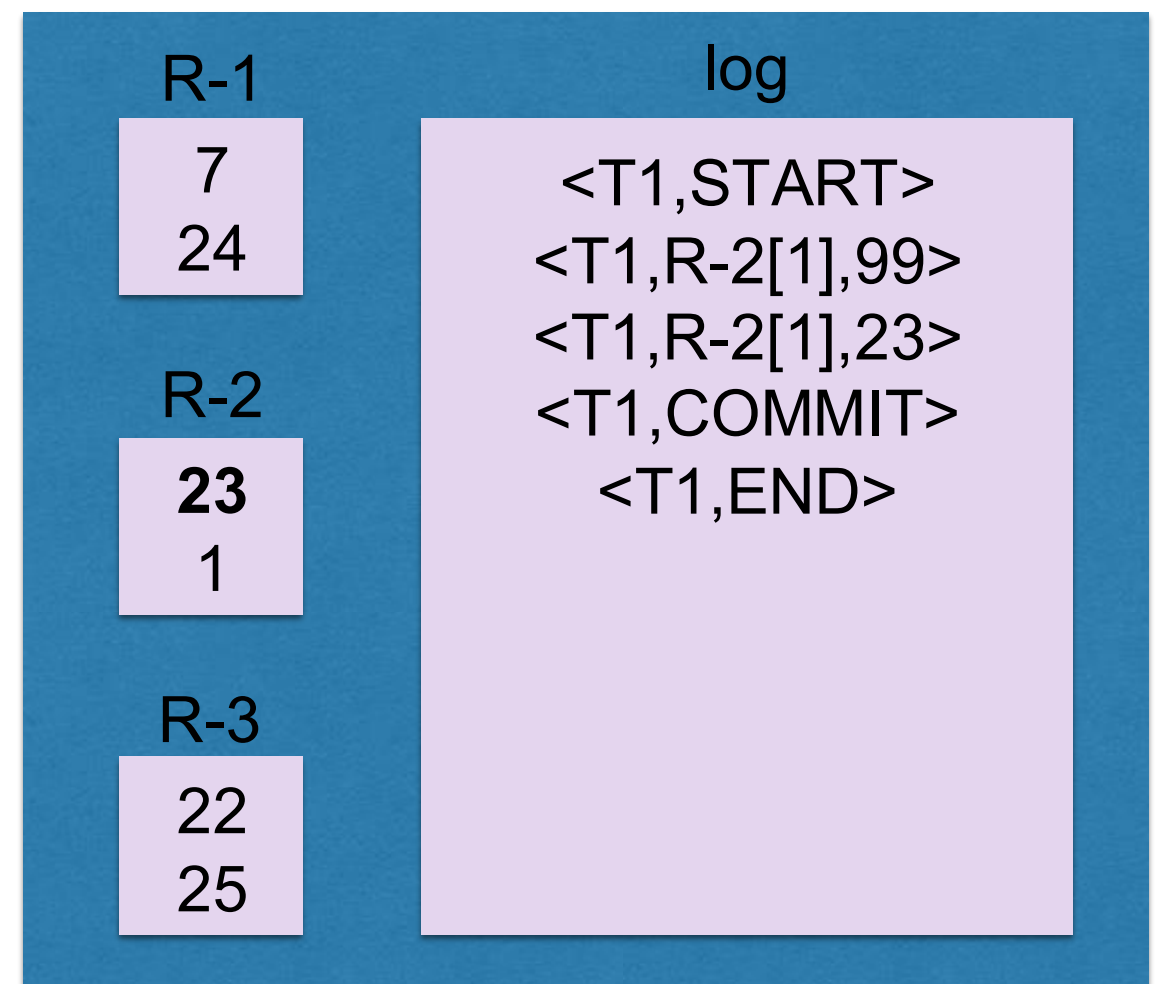
DBMS

100% OK

Buffer



Disco



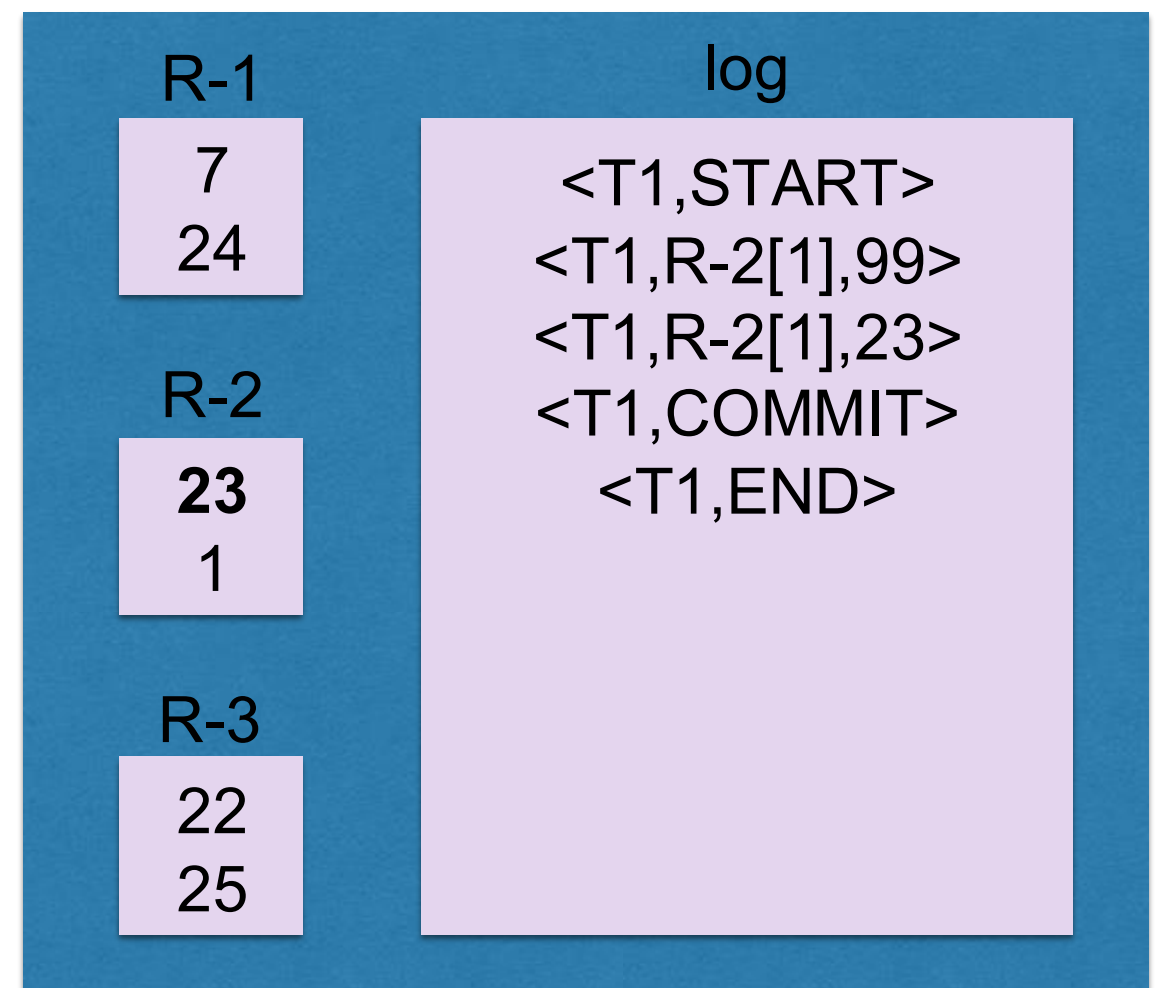
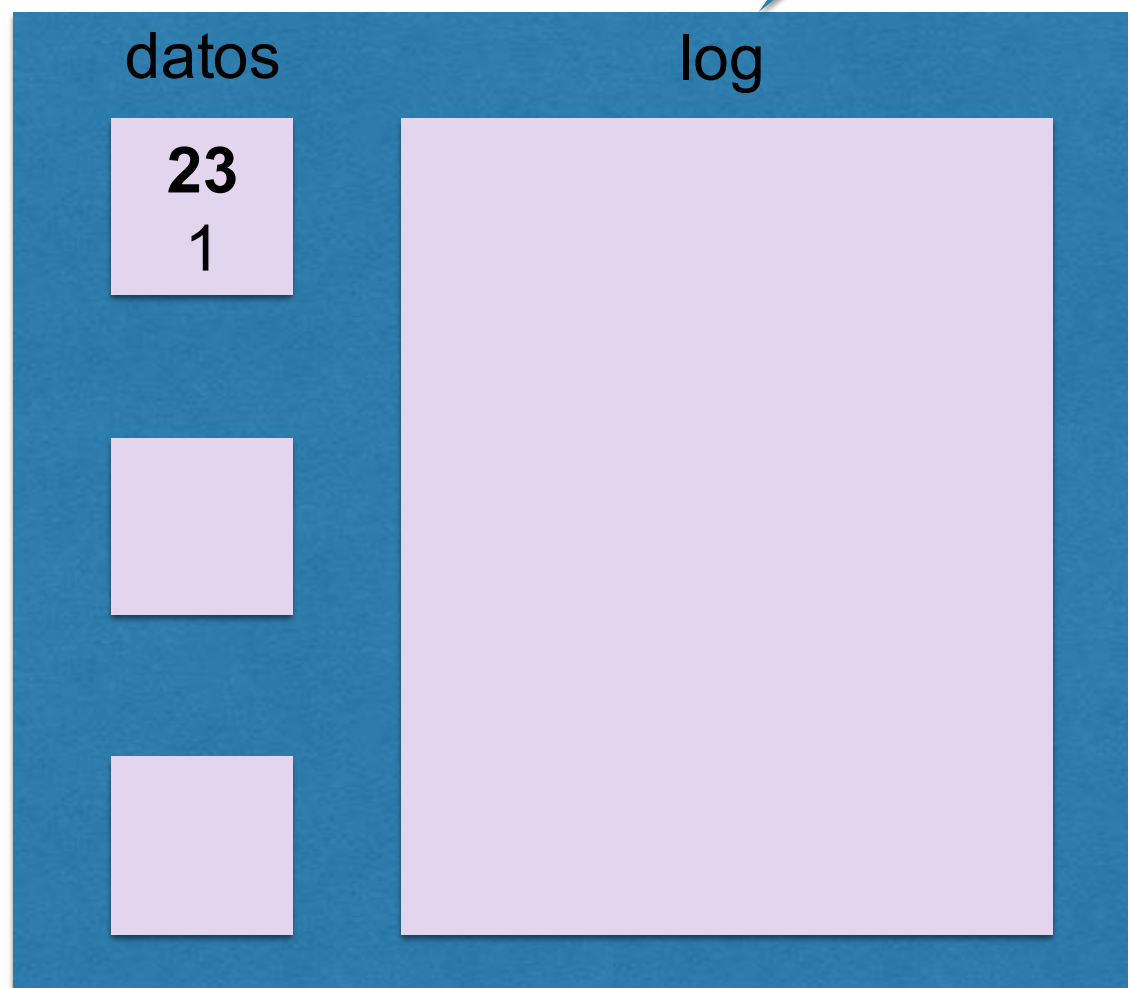
# Redo Logging – en la BD

DBMS

Libera el buffer

Buffer

Disco

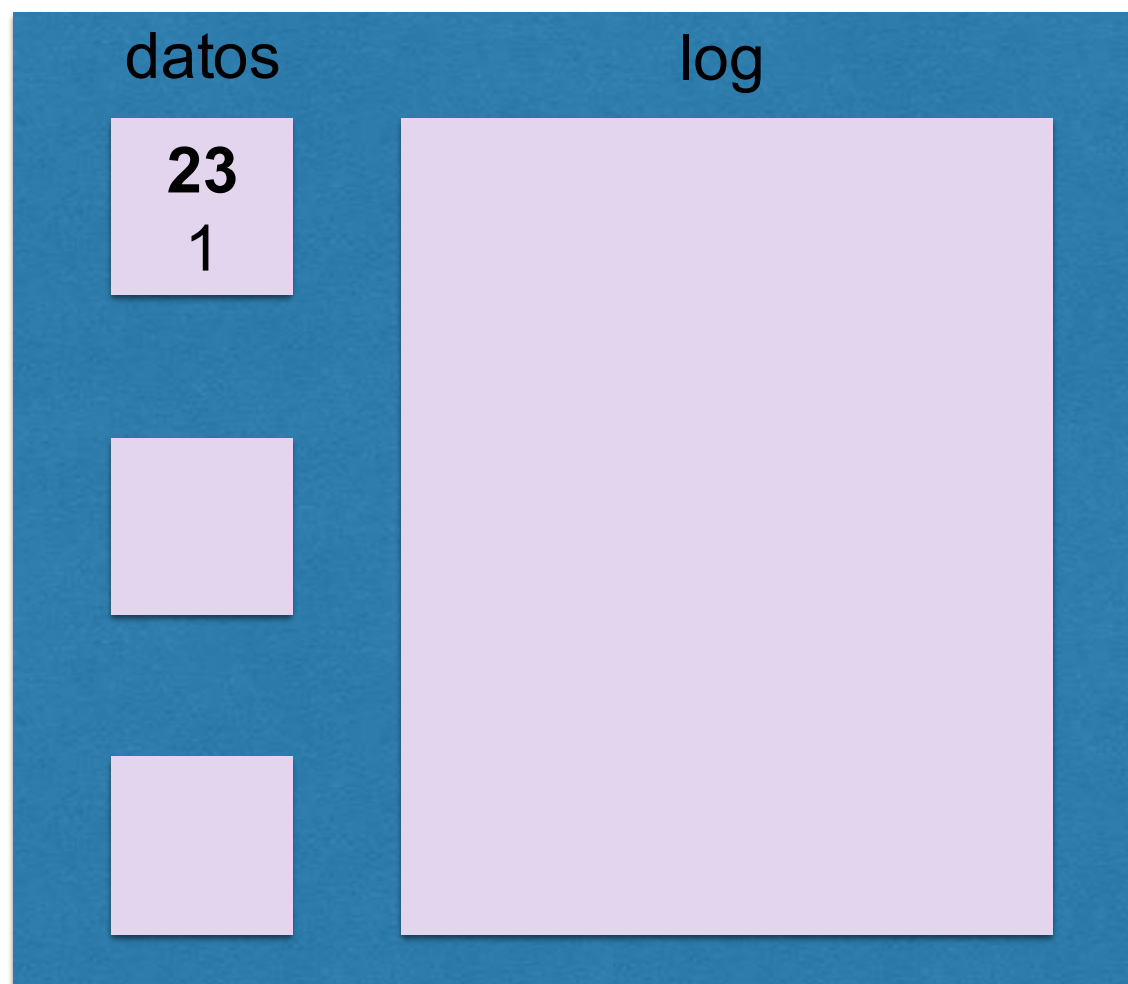


# Redo Logging – en la BD

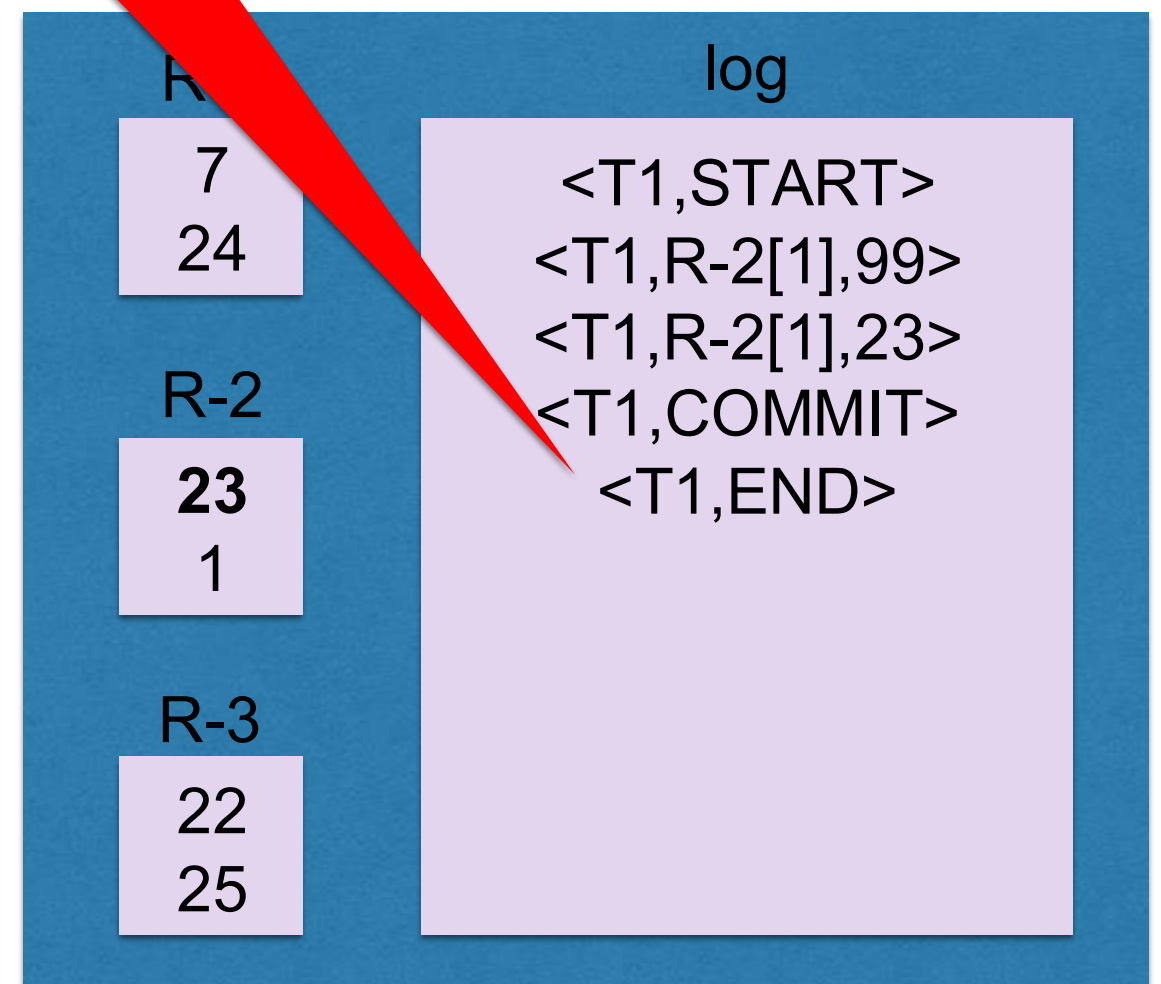
DBMS

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> ...

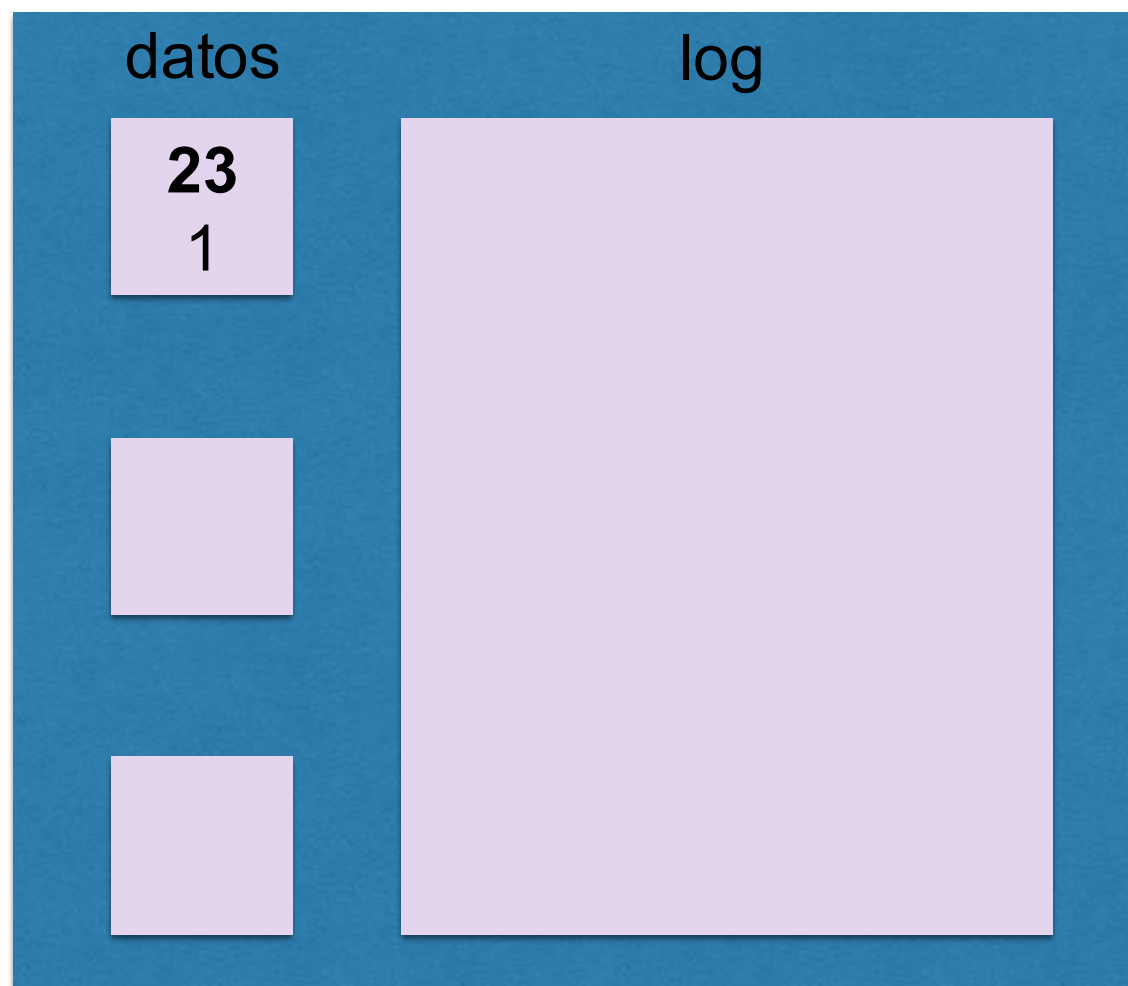


# Redo Logging – en la BD

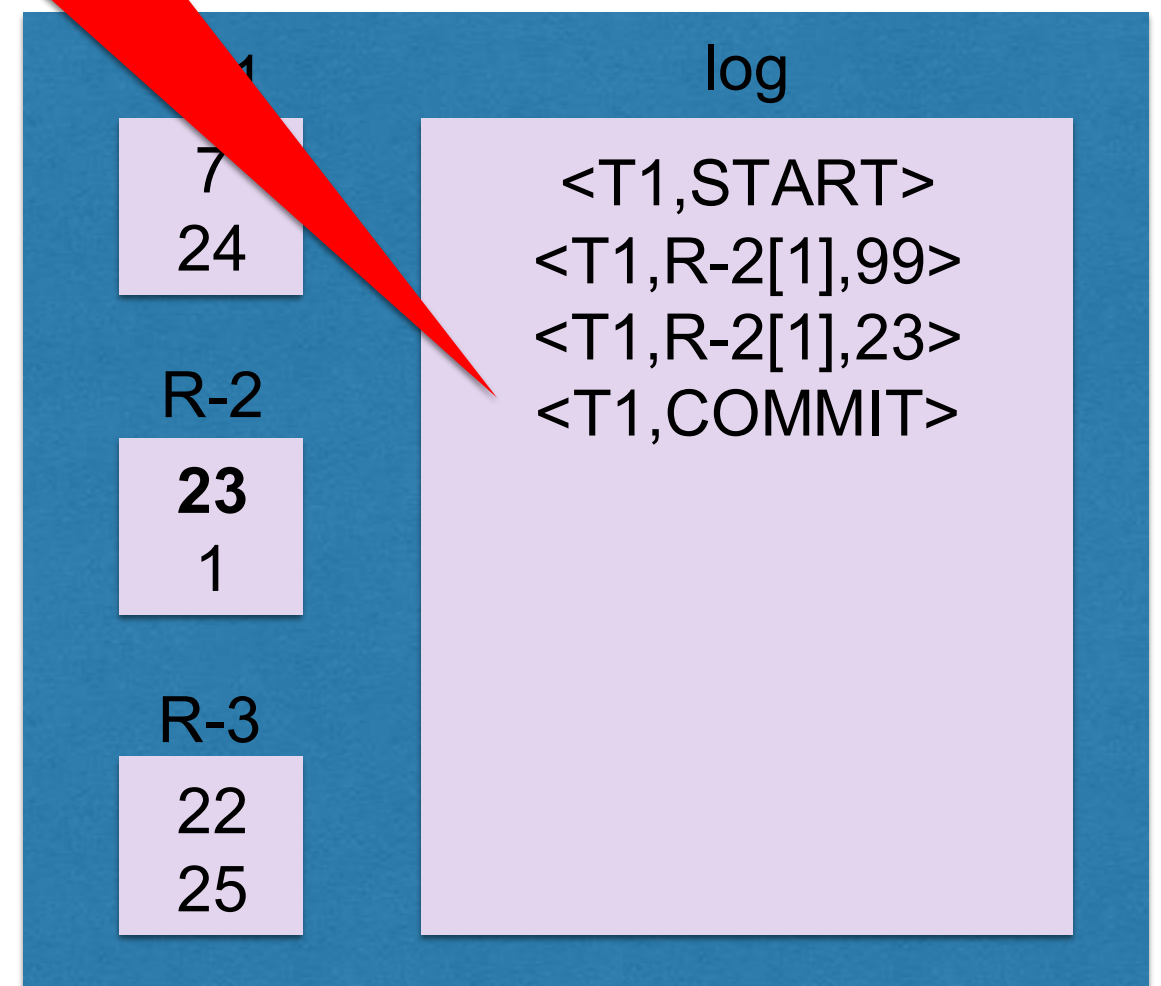
DBMS

No hay END  
(No sé si escribí los datos)

Buffer



Disco



# Recuperación con Redo Logging

Detectando fallas en el *log*:

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



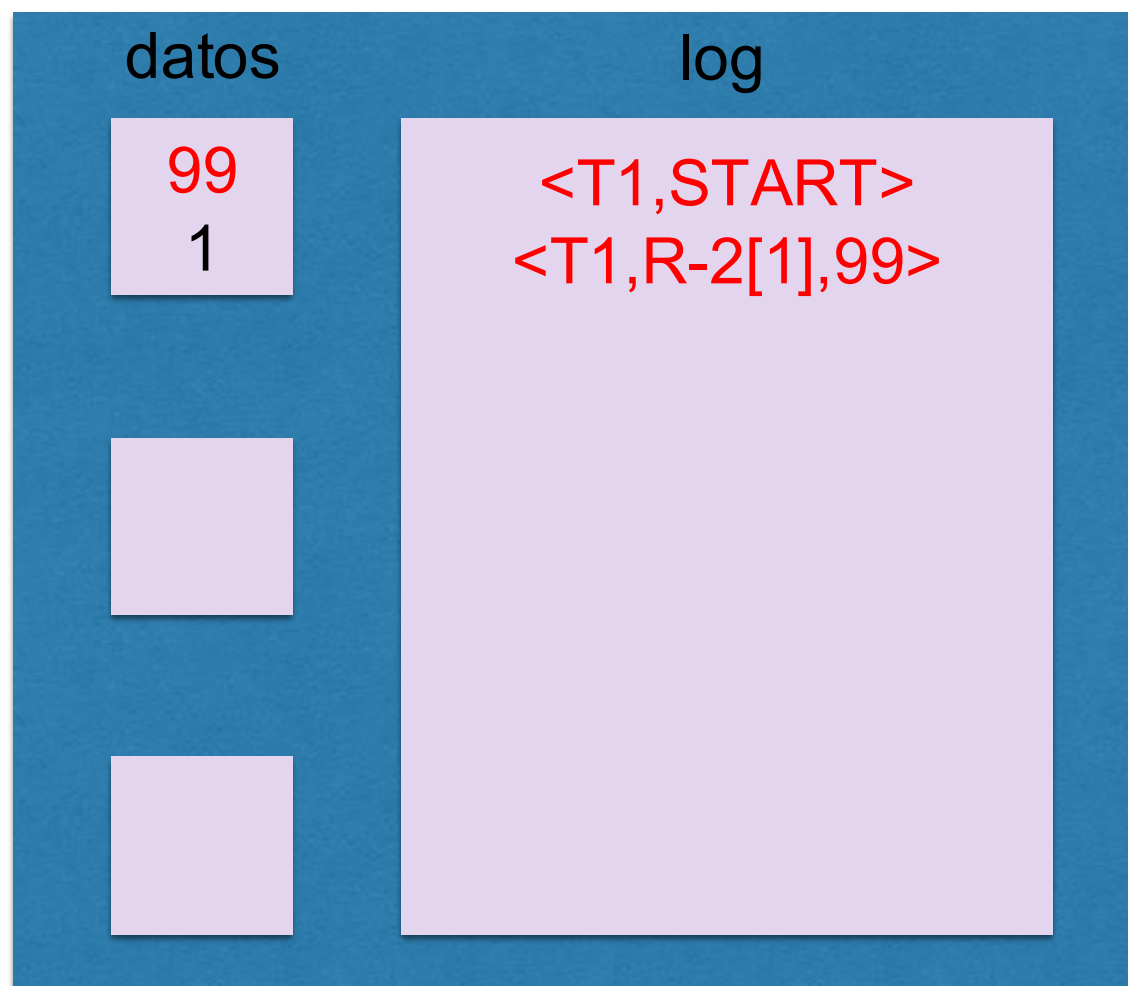


# Redo Logging – en la BD

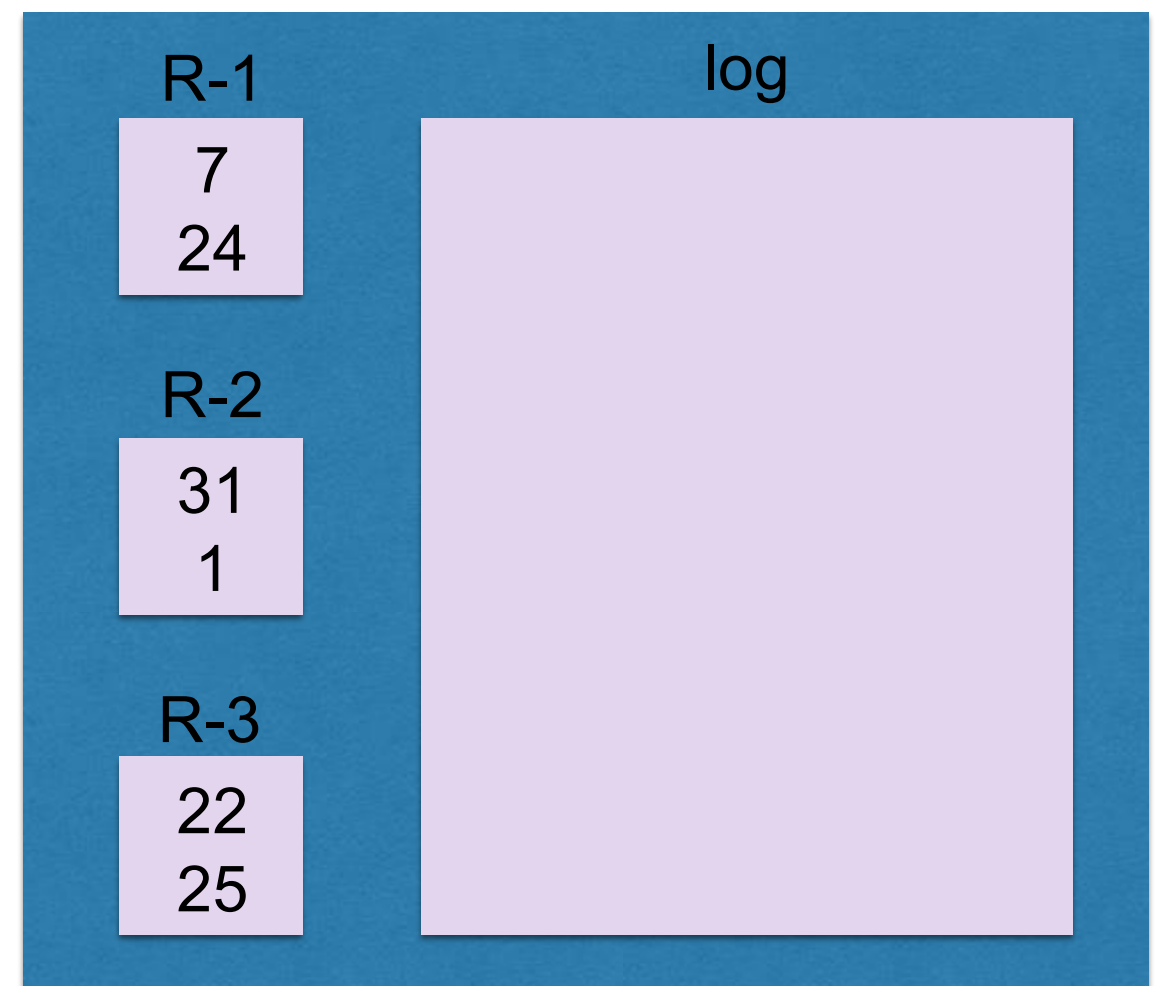
DBMS

T1: Cambio 31 a 99!

Buffer



Disco

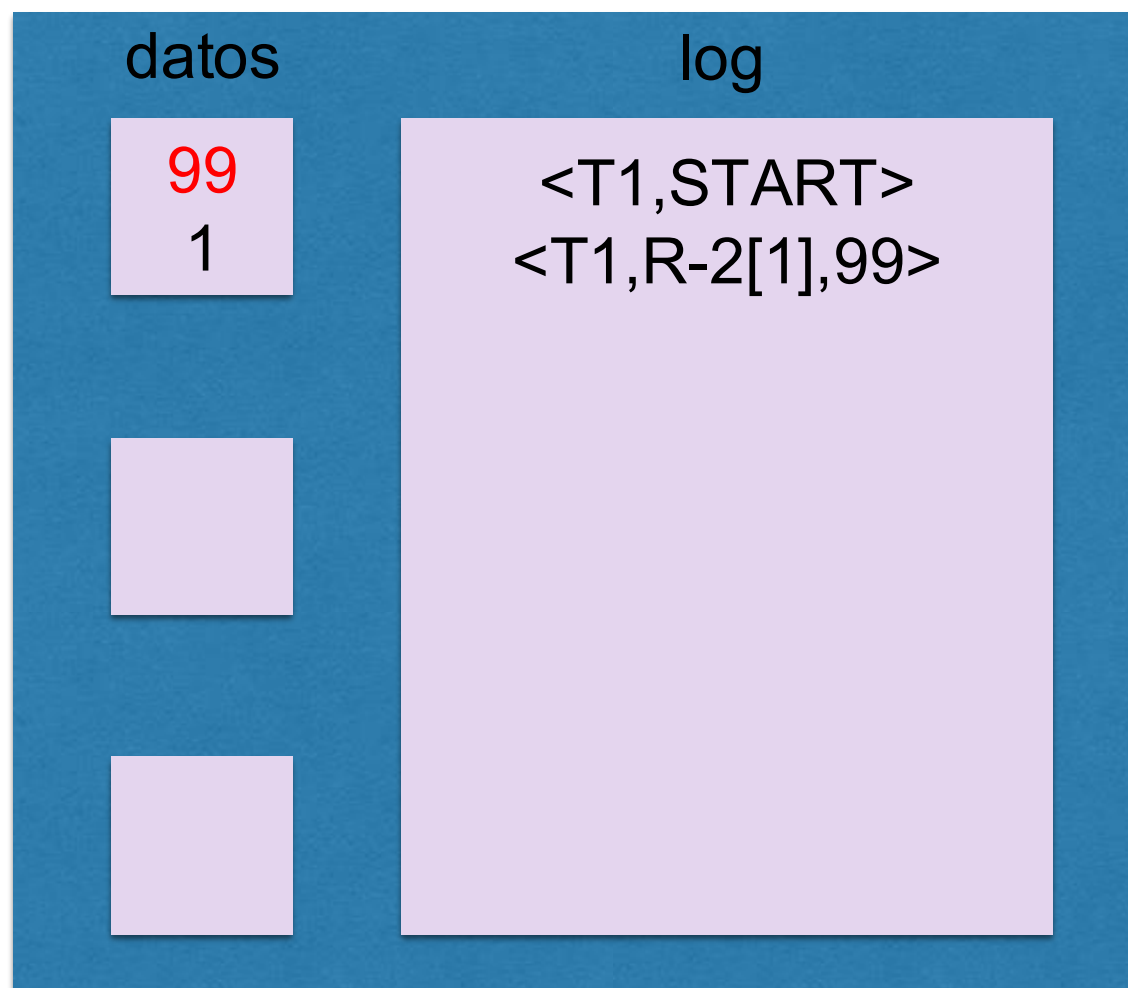


# Redo Logging – en la BD

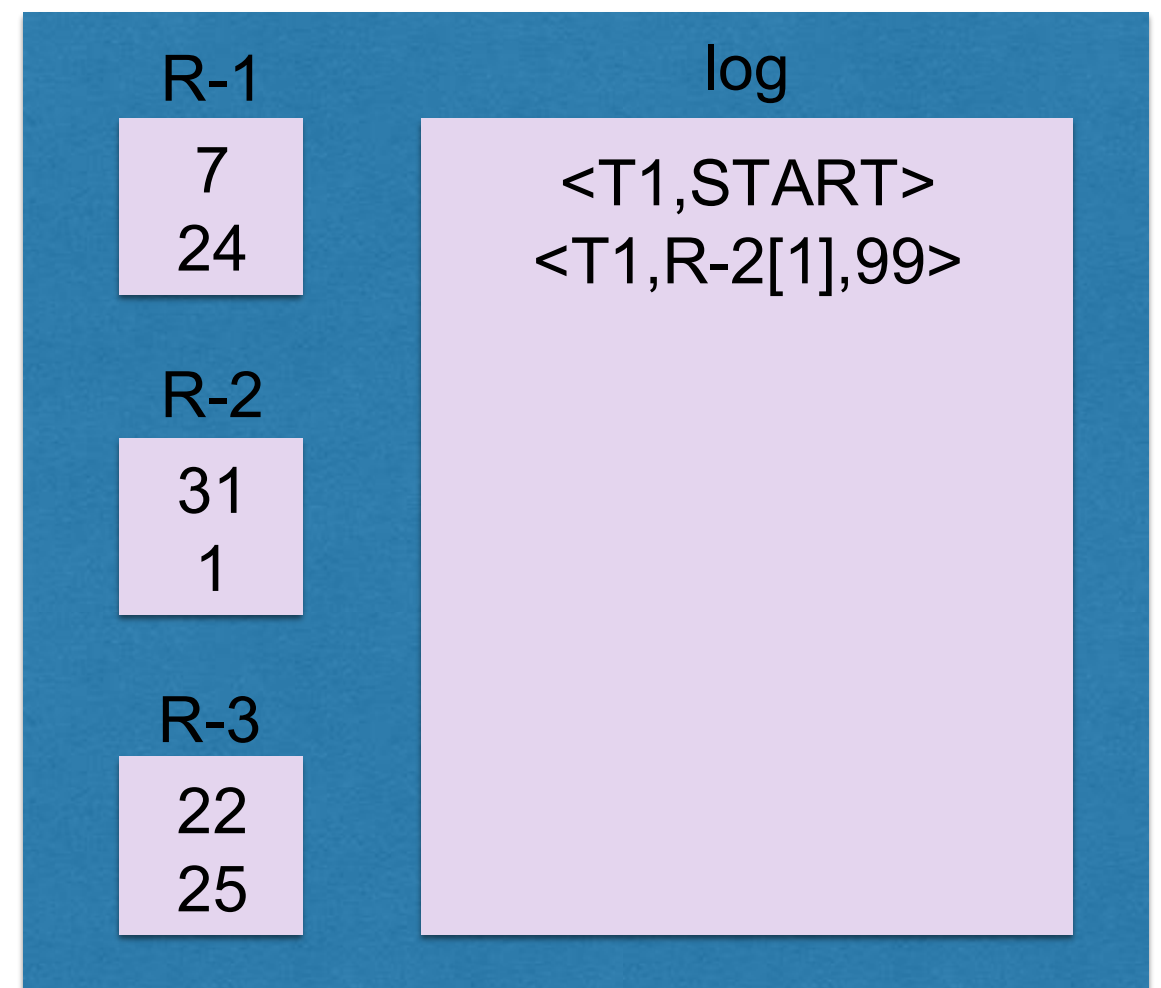
DBMS

T1: Cambio 31 a 99!

Buffer



Disco

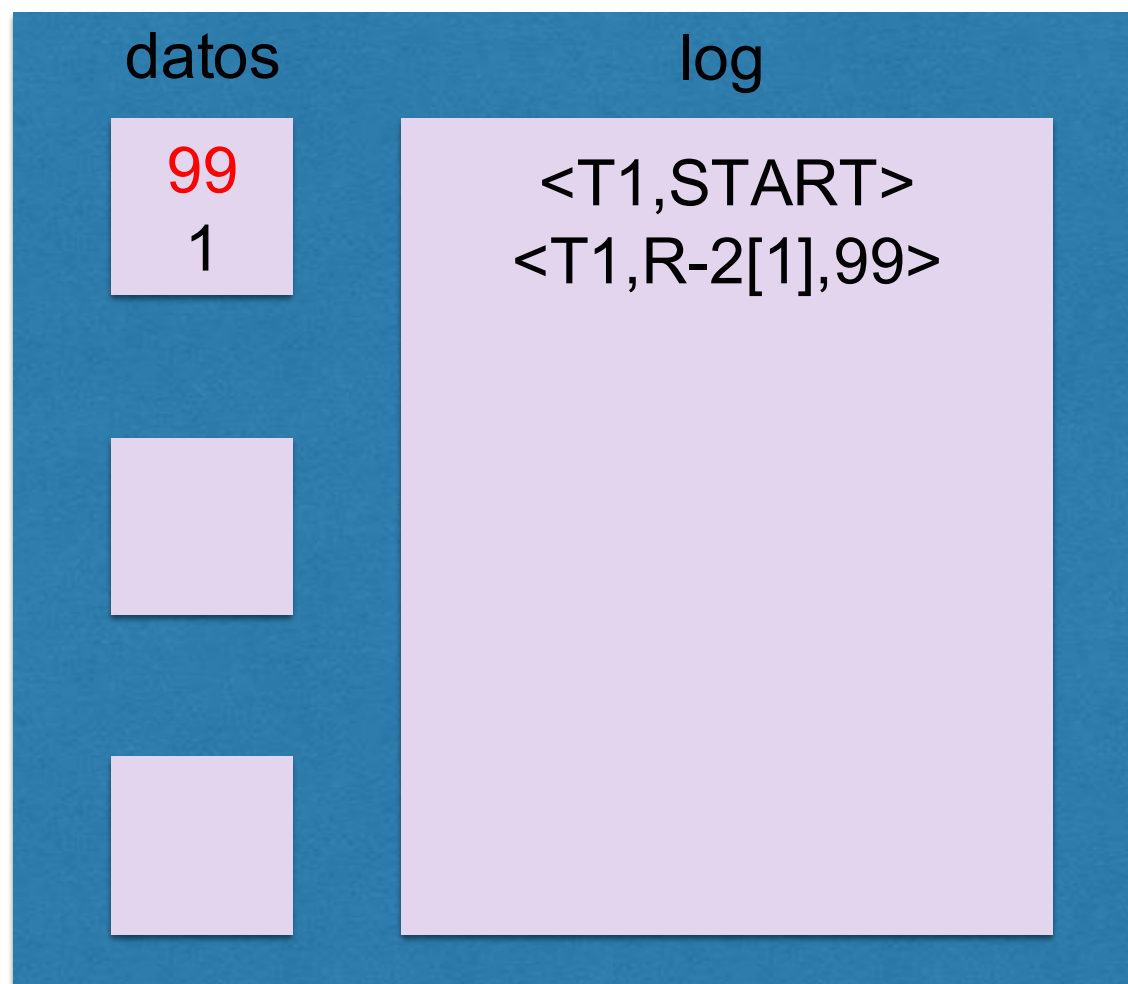


# Redo Logging – en la BD

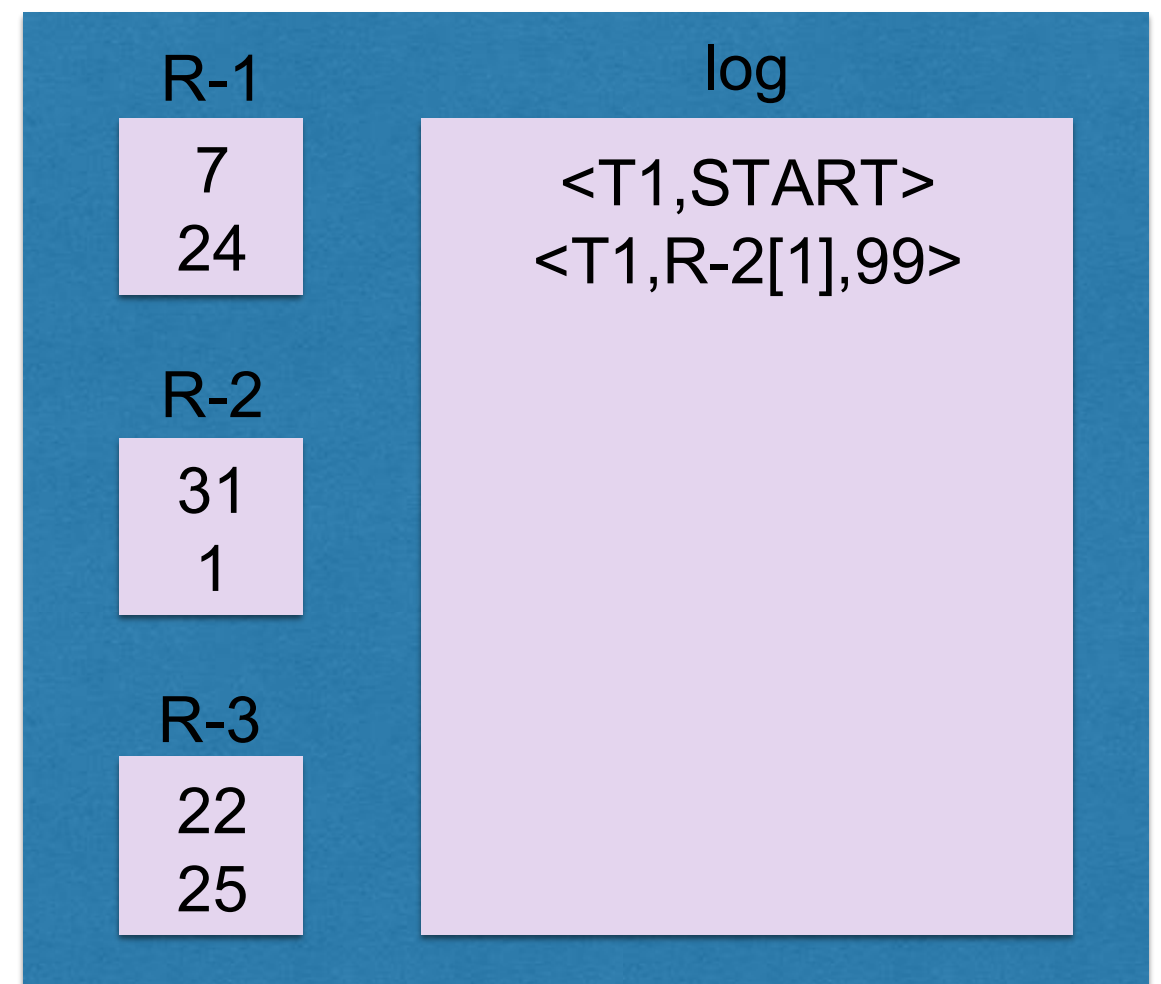
DBMS

T1: bota la ejecución

Buffer



Disco

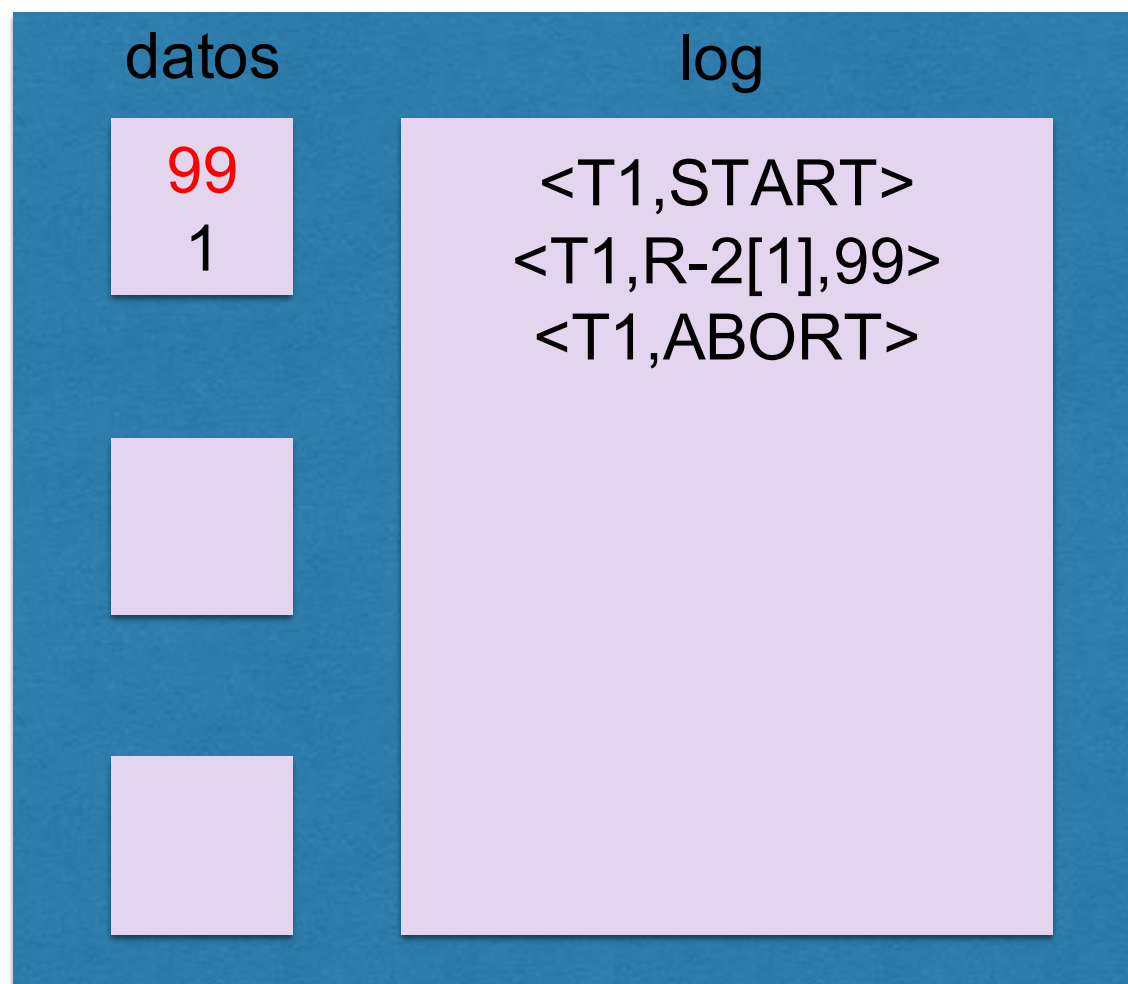


# Redo Logging – en la BD

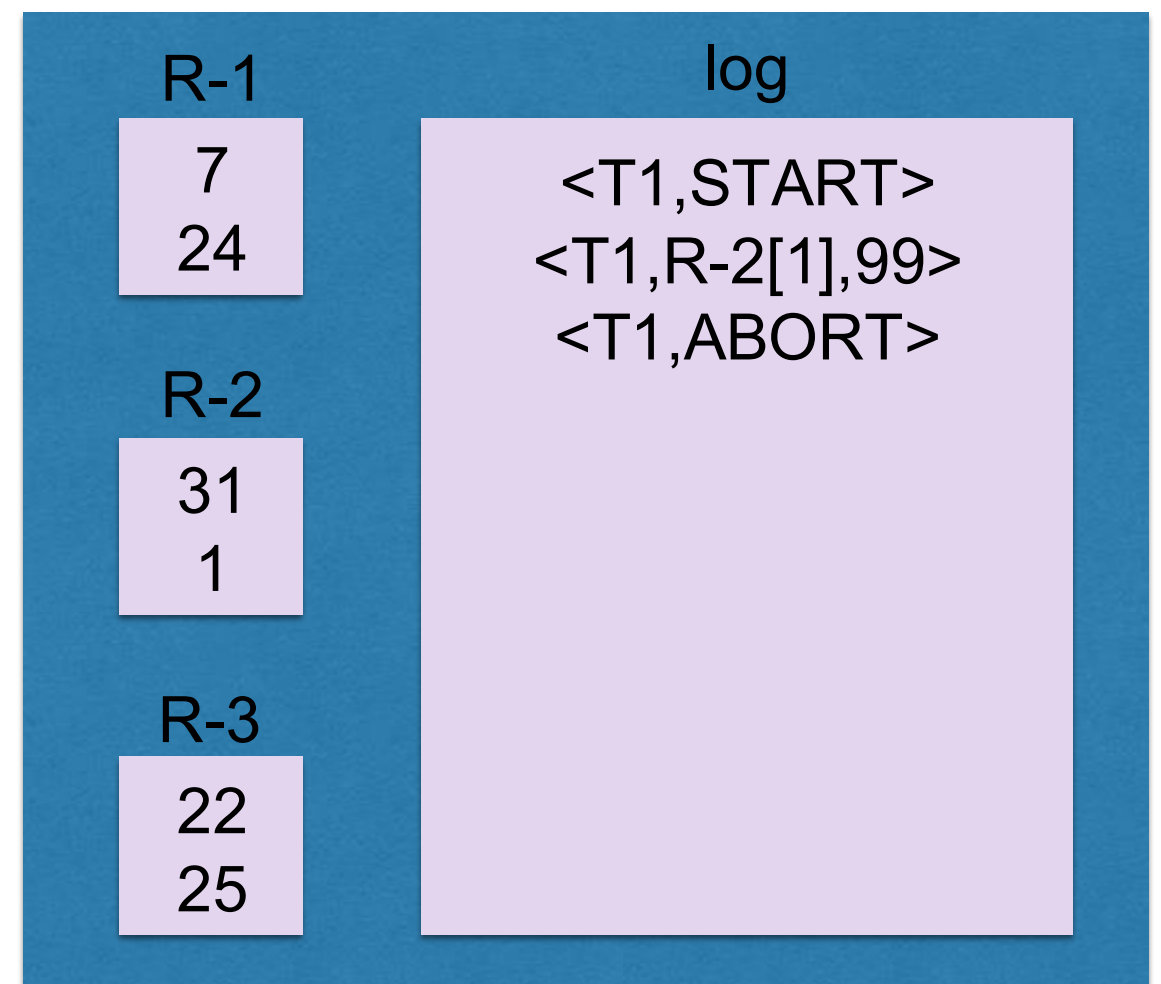
DBMS

T1: bota la ejecución

Buffer



Disco



# Recuperación con Redo Logging

Detectando fallas en el *log*:

... <START T> ...

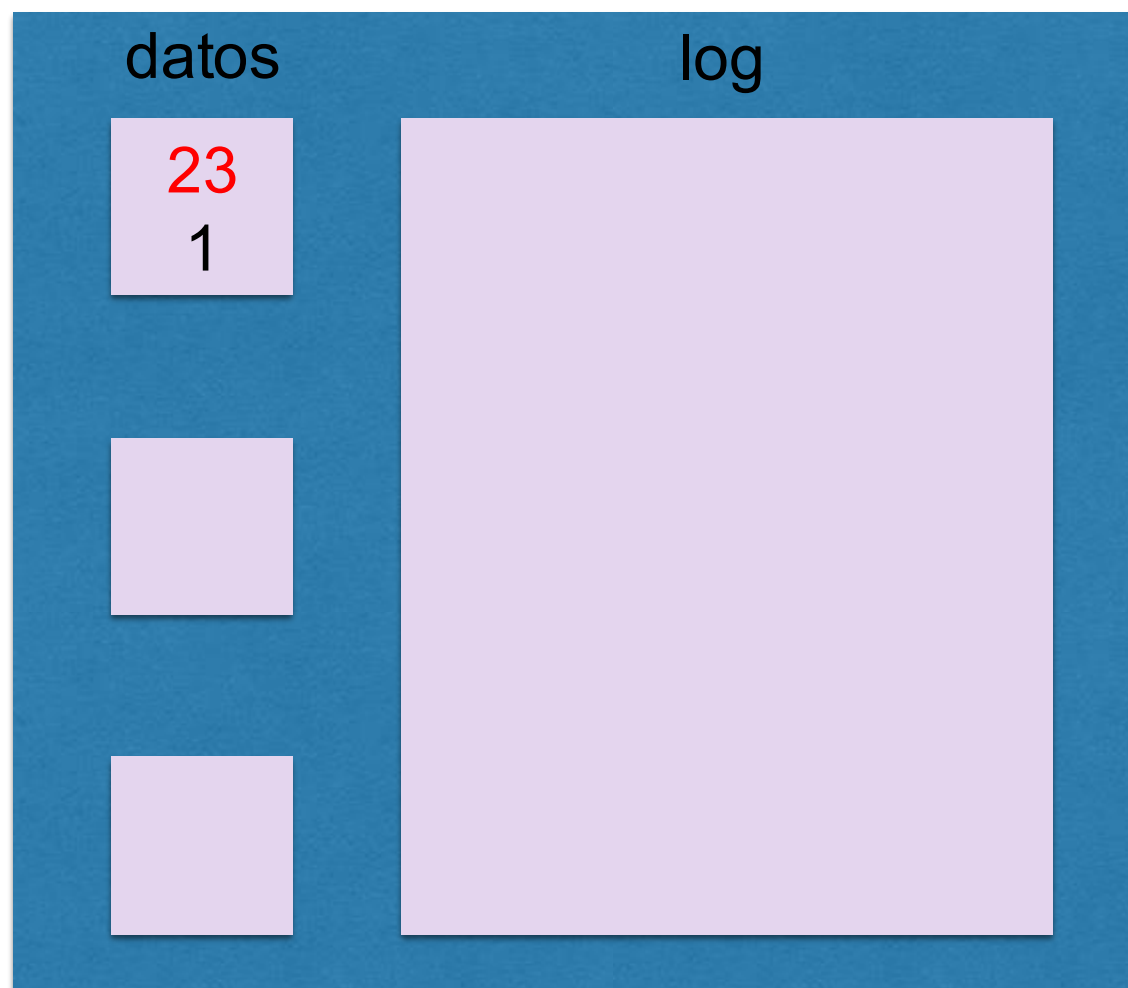


# Redo Logging – en la BD

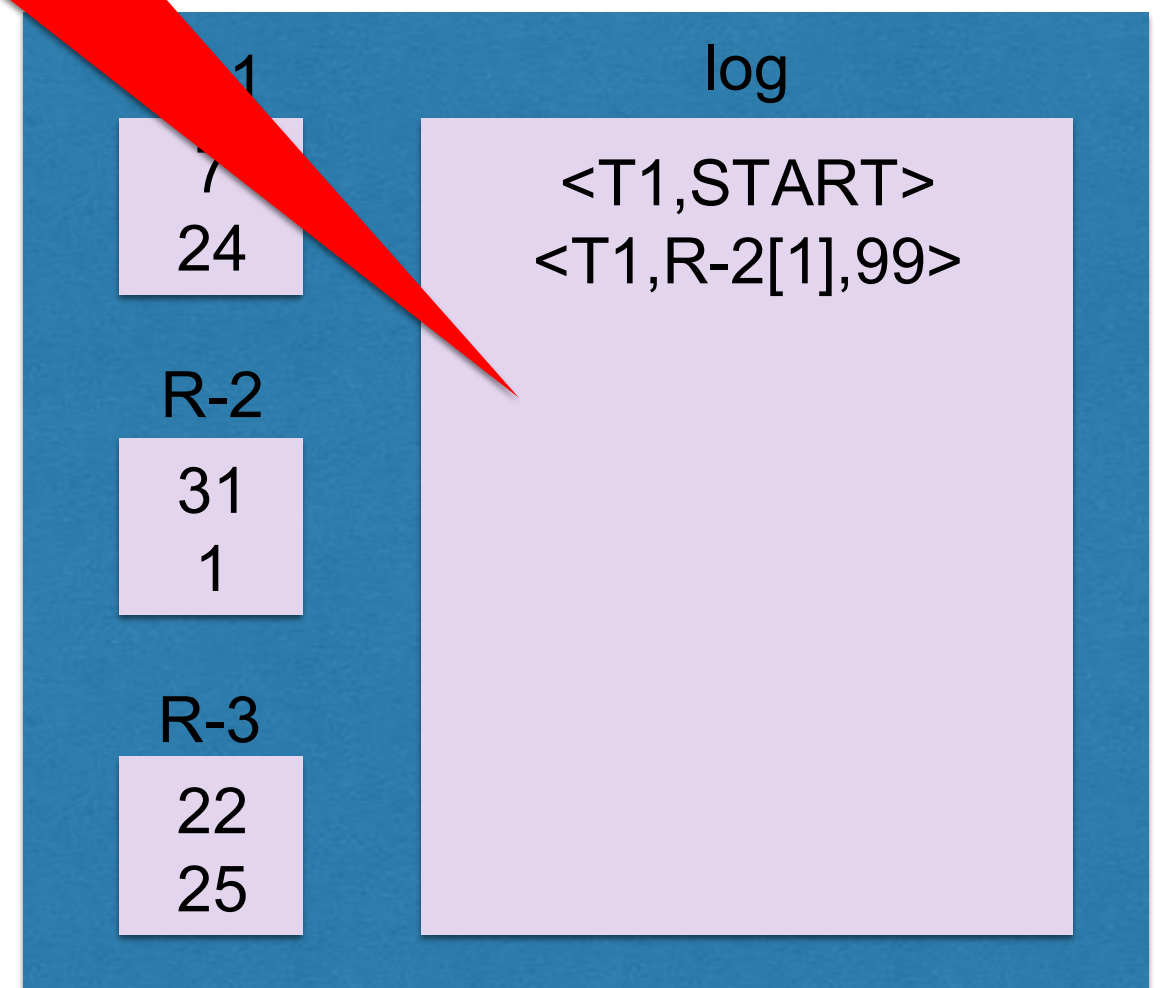
DBMS

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  $\langle T, X, v \rangle$ :
  - Si  $T$  no hizo COMMIT, no hacer nada
  - Si  $T$  hizo COMMIT, reescribir con el valor  $v$
- Para cada transacción sin COMMIT, escribir  $\langle \text{ABORT } T \rangle$

# Recovery

Uso de *Checkpoints* en *Redo Logging*

¿Cómo utilizamos los checkpoints en el Redo Logging?



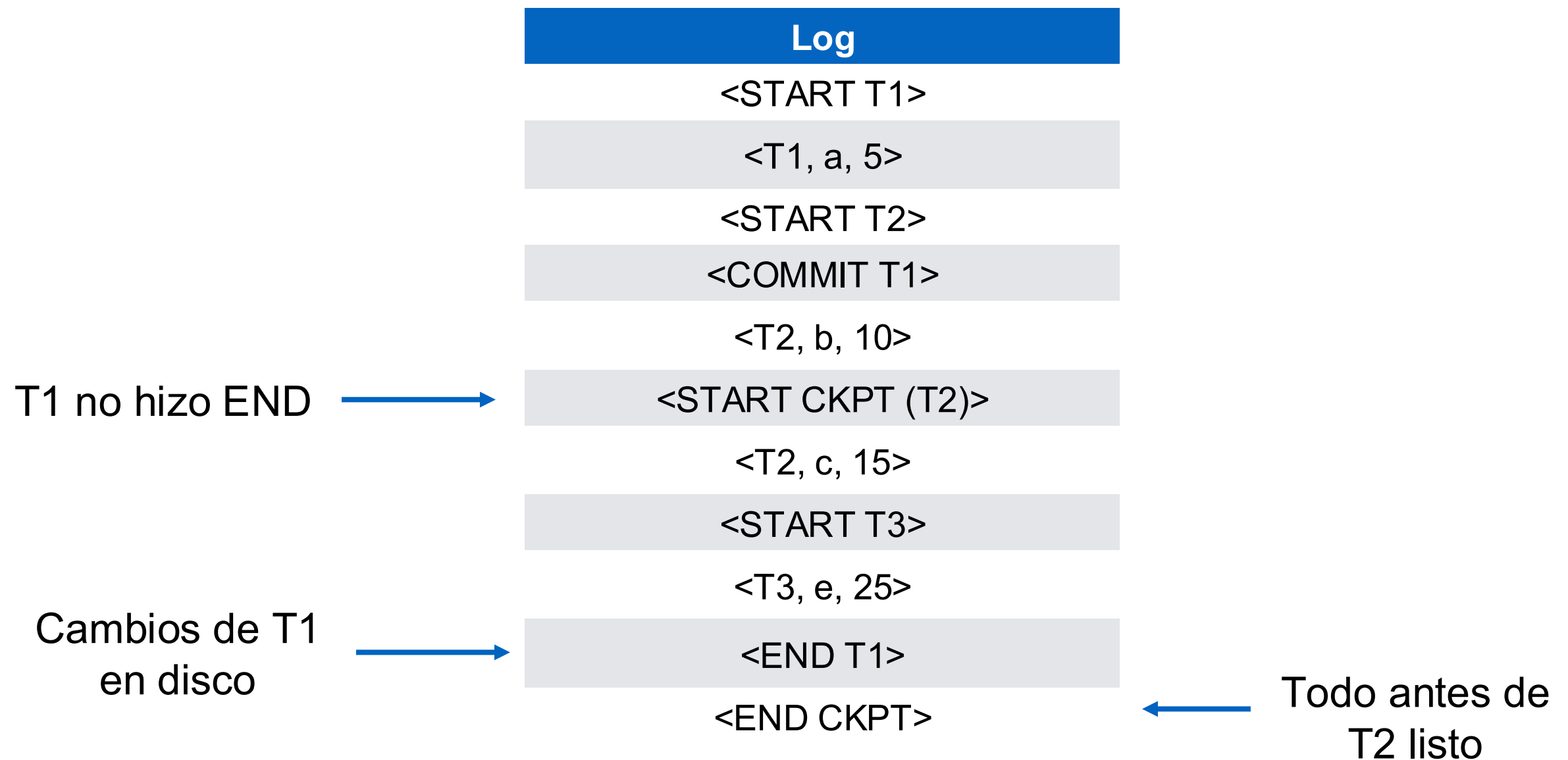
# Recovery

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 `<END CKPT (T1, ..., Tn)>`, retroceder hasta su respectivo `<START CKPT (T1, ..., Tn)>`, y comenzar a hacer *redo* desde la transacción más antigua entre T<sub>1</sub>, ..., T<sub>n</sub>
- No se hace *redo* de las transacciones con COMMIT antes del `<START CKPT (T1, ..., Tn)>`

Su END aparece antes de  
<END CKPT>

# Redo Recovery

Uso de *Checkpoints* en Redo Logging

- Revisar el *log* desde el final
- Si encontramos un `<END CKPT (T1, ..., Tn)>`, retroceder hasta su respectivo `<START CKPT (T1, ..., Tn)>`, y comenzar a hacer *redo* desde la transacción más antigua entre T<sub>1</sub>, ..., T<sub>n</sub>
- No se hace *redo* de las transacciones con COMMIT antes del `<START CKPT (T1, ..., Tn)>`

En realidad END de estas transacciones es redundante

# Redo Recovery

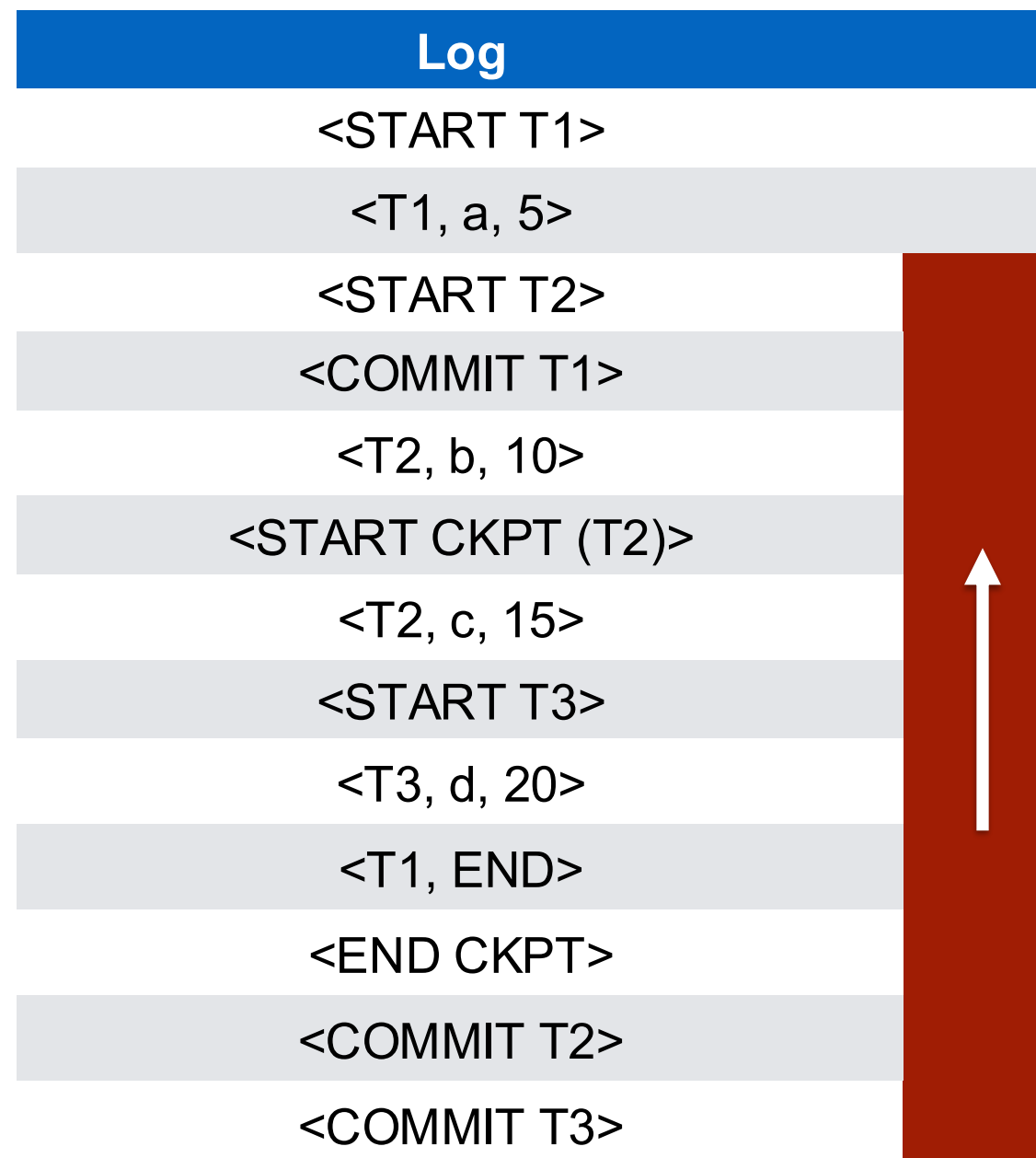
Uso de *Checkpoints* en *Redo Logging*

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

**Pero hay un 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,  $v_{\text{antiguo}}$ ,  $v_{\text{nuevo}}$ >**
- **<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

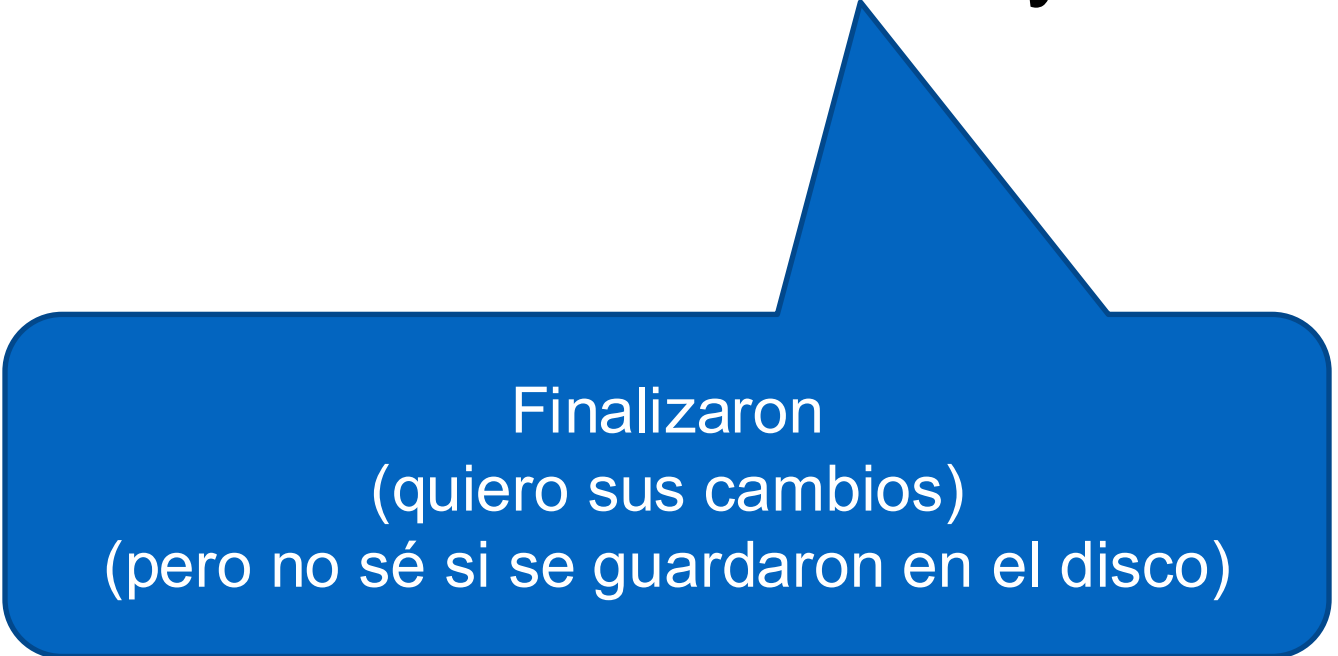
# Undo/Redo Logging



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



# Recursos para estudiar

[http://mlwiki.org/index.php/Database\\_Transaction\\_Log](http://mlwiki.org/index.php/Database_Transaction_Log)

[Database Management Systems, 3rd edition, de Raghu Ramakrishnan y Johannes Gehrke. Cap 18](#)