

Procesamiento de transacciones, concurrencia y recuperación

¿Qué es una transacción?

- El concepto de **transacción** describe unidades lógicas de procesamiento de bases de datos
- Los **sistemas de procesamiento de transacciones** son sistemas con bases de datos grandes y cientos, miles o millones de usuarios concurrentes
- ¿Cuales son ejemplos de sistemas de procesamiento de transacciones?

¿Qué es una transacción?

- El concepto de **transacción** describe unidades lógicas de procesamiento de bases de datos
- Los **sistemas de procesamiento de transacciones** son sistemas con bases de datos grandes y cientos, miles o millones de usuarios concurrentes
- ¿Cuales son ejemplos de sistemas de procesamiento de transacciones?

Reservaciones de aerolíneas

¿Qué es una transacción?

- El concepto de **transacción** describe unidades lógicas de procesamiento de bases de datos
- Los **sistemas de procesamiento de transacciones** son sistemas con bases de datos grandes y cientos, miles o millones de usuarios concurrentes
- ¿Cuales son ejemplos de sistemas de procesamiento de transacciones?

Sistemas bancarios

ones de
neas

¿Qué es una transacción?

- El concepto de **transacción** describe unidades lógicas de procesamiento de bases de datos
- Los **sistemas de procesamiento de transacciones** son sistemas con bases de datos grandes y cientos, miles o millones de usuarios concurrentes
- ¿Cuales son ejemplos de sistemas de procesamiento de transacciones?

Sistemas bancarios

El sistema que
usan en walmart!

iones de
neas

¿Qué es una transacción?

- El concepto de **transacción** describe unidades lógicas de procesamiento de bases de datos
- Los **sistemas de procesamiento de transacciones** son sistemas con bases de datos grandes y cientos, miles o millones de usuarios concurrentes
- ¿Cuales son ejemplos de sistemas de transacciones?



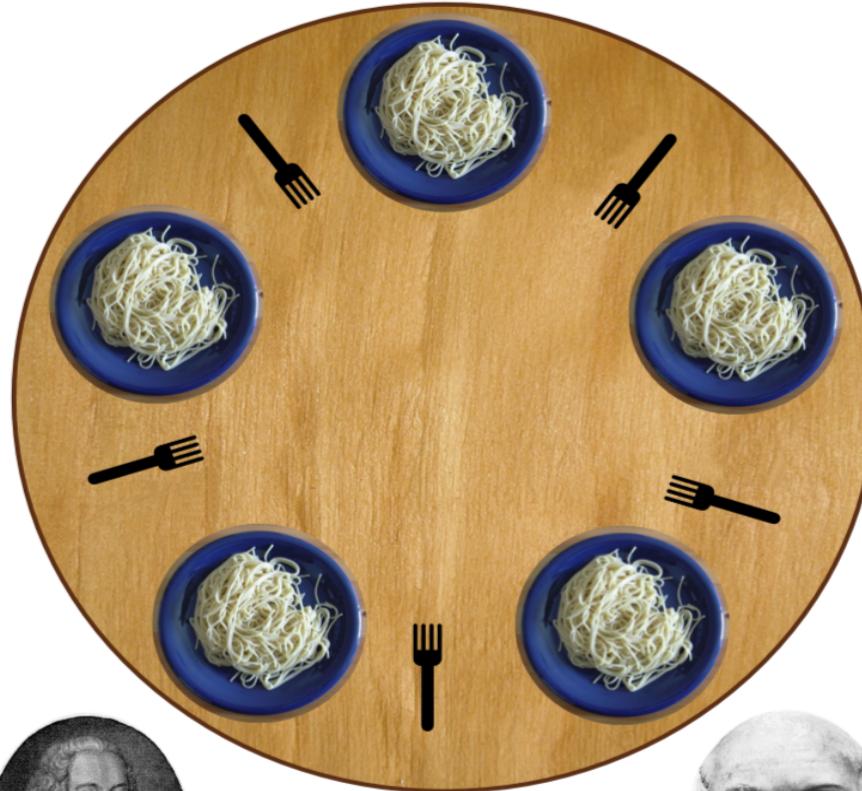
¿Qué es una transacción?

- El concepto de **transacción** describe unidades lógicas de procesamiento de bases de datos
- Los **sistemas de procesamiento de transacciones** son sistemas con bases de datos grandes y cientos, miles o millones de usuarios concurrentes
- ¿Cuales son los sistemas de procesamiento de transacciones?

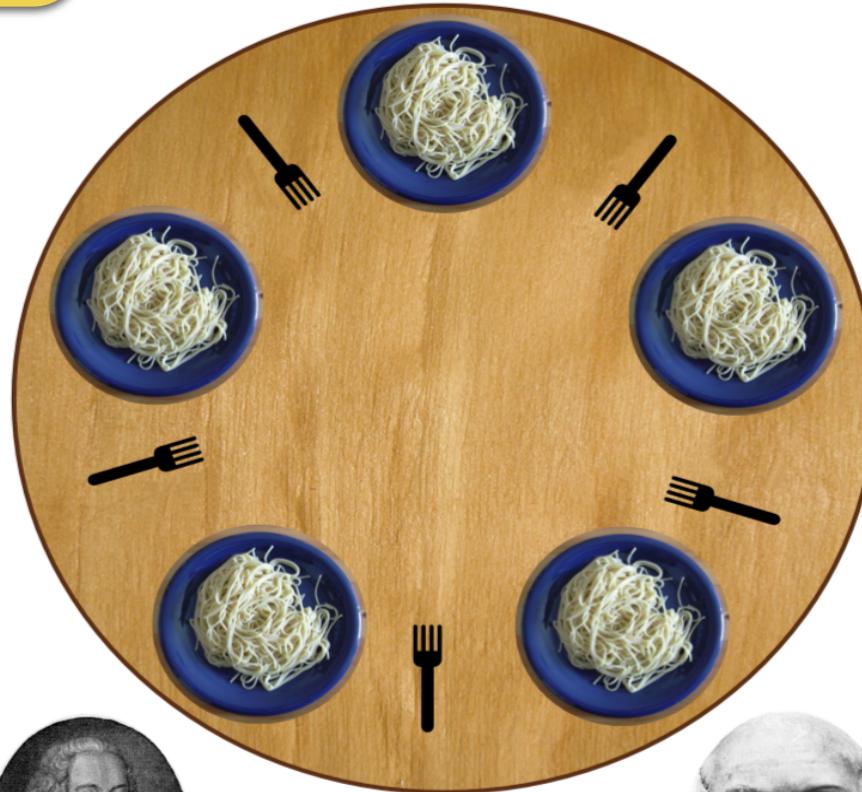
¿Qué es
conurrencia?



Cada uno de
estos viejos son
“filósofos” ...

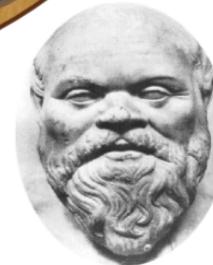
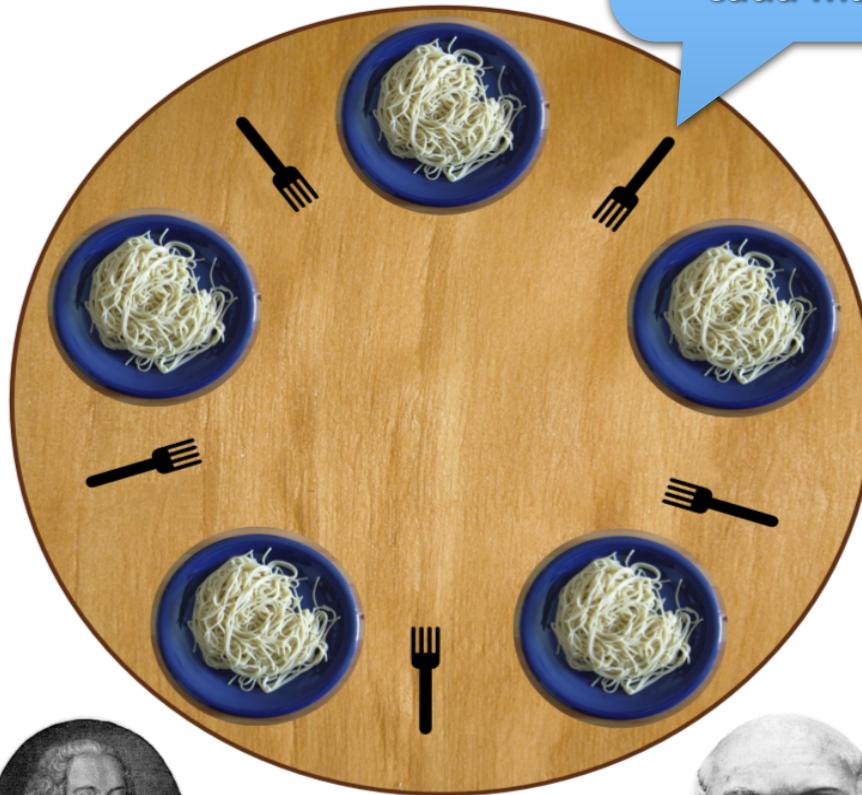


Solo saben pensar
y comer....

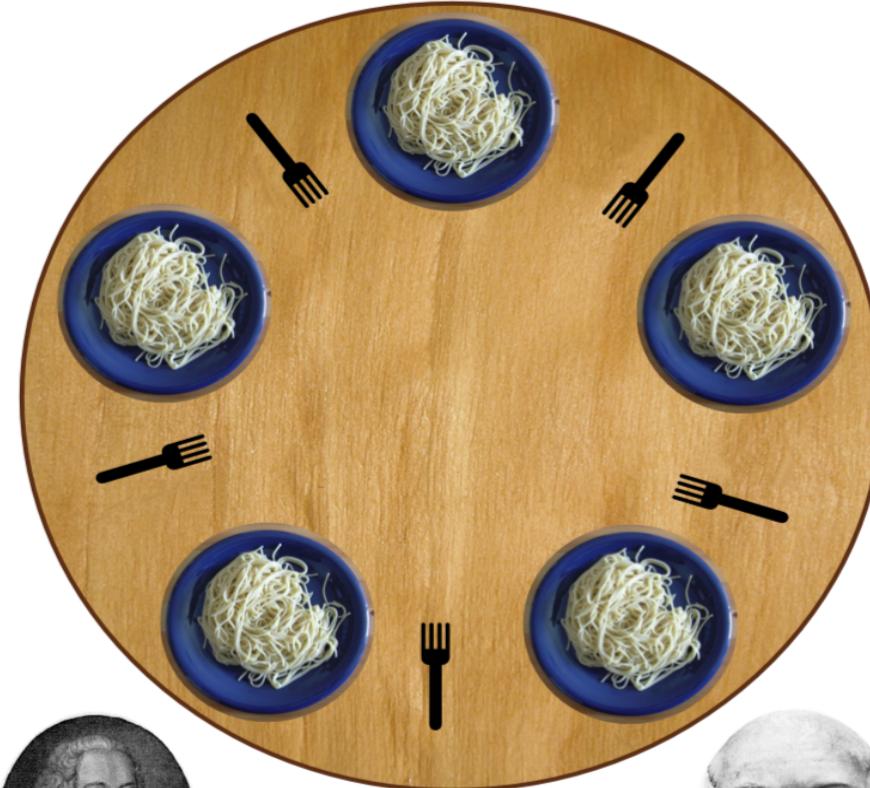




Cada filosofo
come con dos
tenedores...uno en
cada mano...

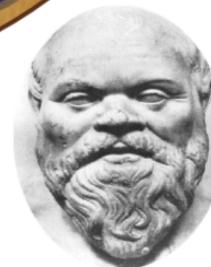
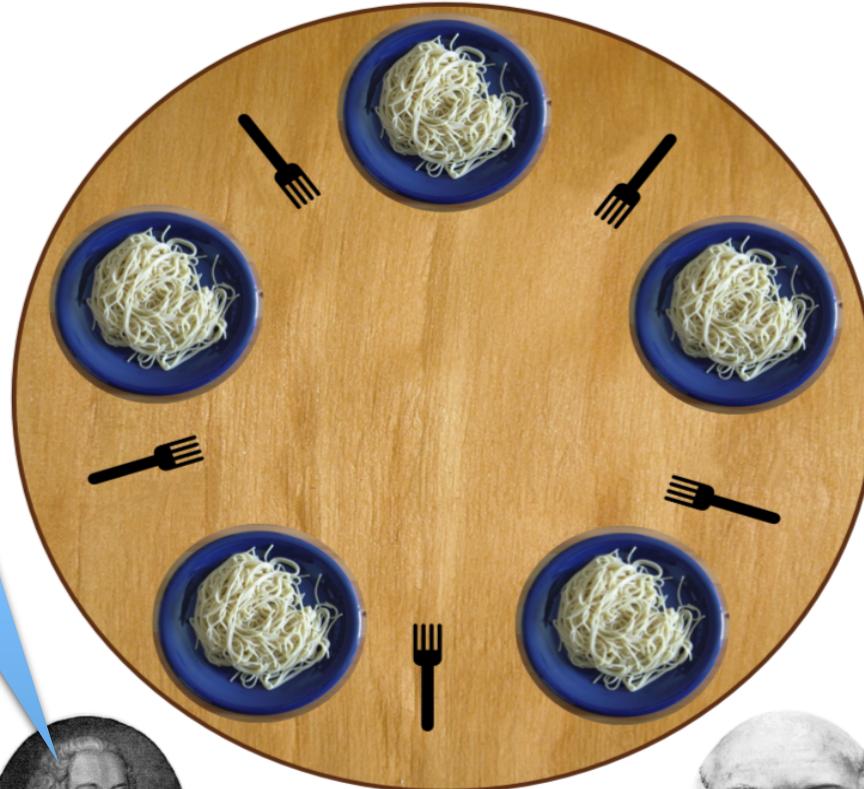


Un filósofo solo
puede tomar los
tenedores a su
izquierda y
derecha

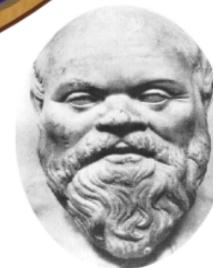
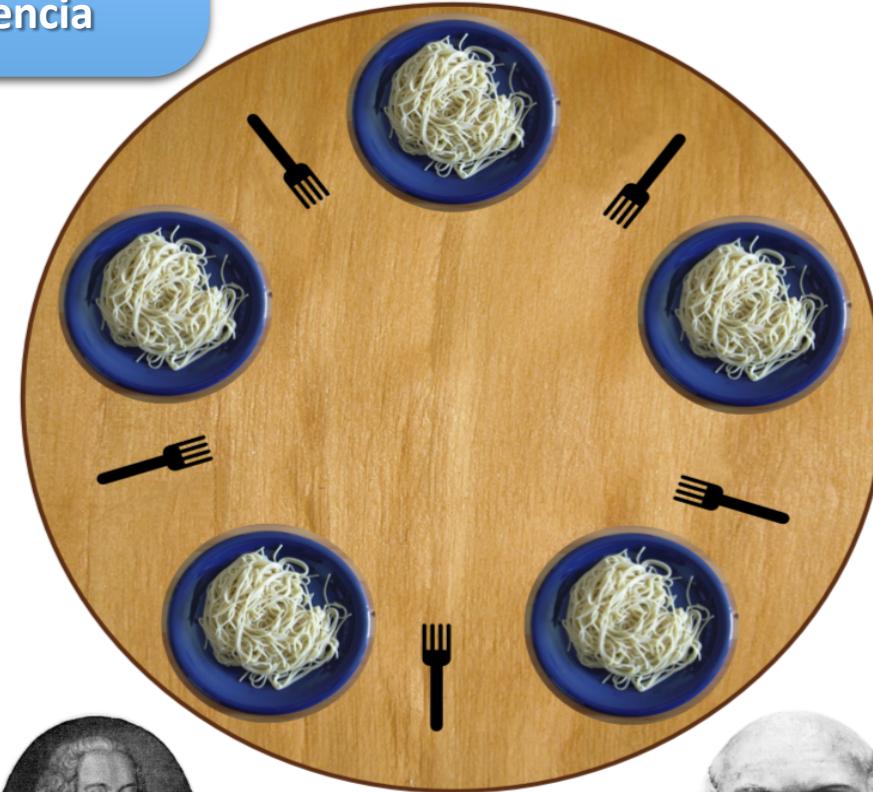




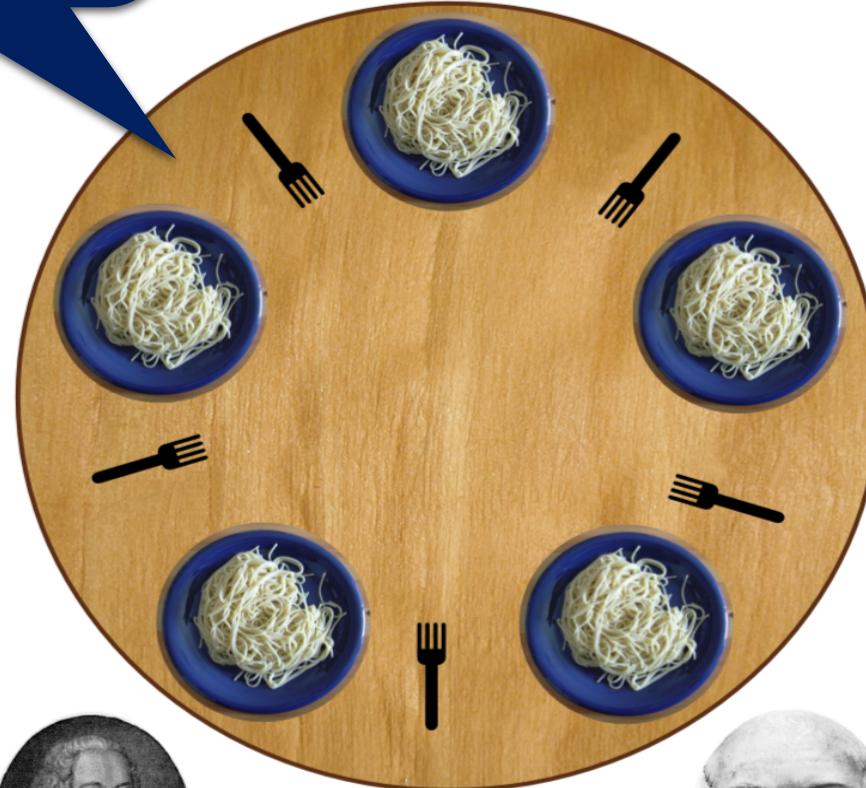
Si un filósofo toma un tenedor y el otro no está disponible, se queda esperando con el otro tenedor agarrado, hasta que pueda tomar los dos



Hay una situación de competencia por los tenedores...es un problema de concurrencia

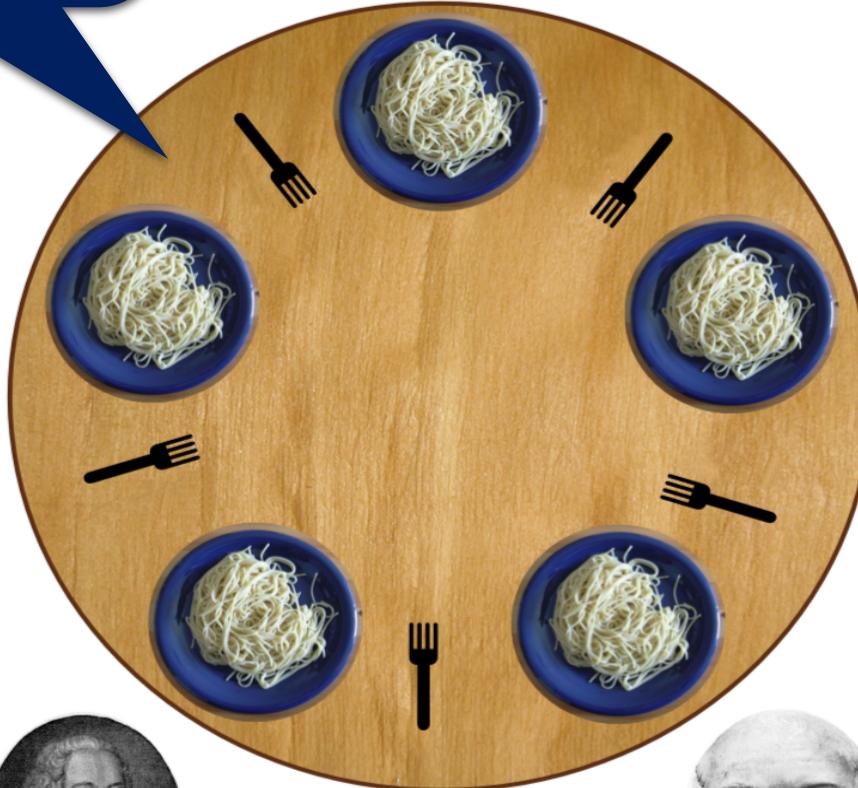
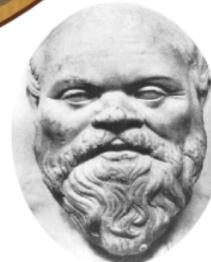
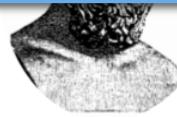


Este es un problema
clásico de concurrencia:
“Problema de la cena de
los filósofos”



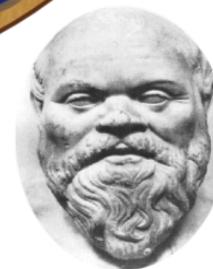
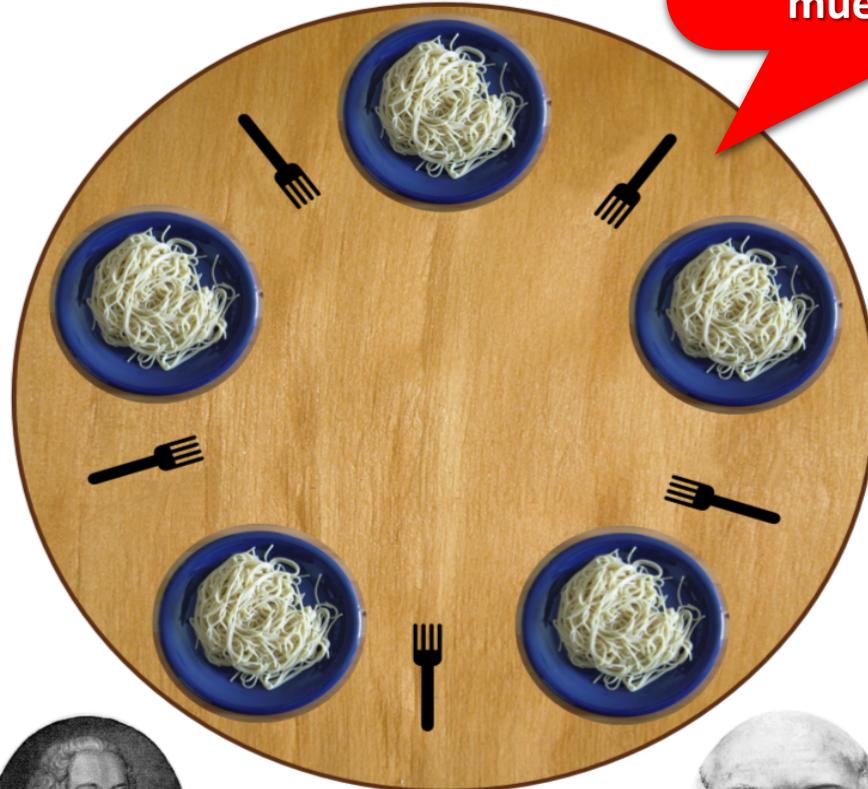
Este es un problema
clásico de concurrencia:
“Problema de la cena de
los filósofos”

Propuesto por
Dijkstra en 1965





Volviendo al tema...el ejercicio consiste en diseñar un algoritmo en el que ningún filósofo se muera de hambre..



¿Qué es una transacción?

- El tema de la concurrencia está presente en muchas áreas de la computación...
- Concurrencia es acceso de múltiples usuarios/procesos/hilos simultáneamente
- En el contexto de bases de datos, se puede clasificar un DBMS por la cantidad de usuarios que pueden usarlo concurrentemente.

¿Qué es una transacción?

- El tema de la concurrencia está presente en muchas áreas de la computación...
- Concurrencia es acceso de múltiples usuarios/procesos/hilos simultáneamente
- En el contexto de bases de datos, se puede clasificar un DBMS por la cantidad de usuarios que pueden usarlo concurrentemente.

Single-user: es un DBMS que a lo sumo puede ser utilizado por un usuario a la vez

¿Qué es una transacción?

- El tema de la concurrencia está presente en muchas áreas de la computación...
- Concurrencia es acceso de múltiples usuarios/procesos/hilos simultáneamente
- En el contexto de bases de datos, se puede clasificar un DBMS por la cantidad de usuarios que pueden usarlo concurrentemente.

Multiuser: es un DBMS que puede ser utilizado por muchos usuarios a la vez

¿Qué es una transacción?

- Este concepto de múltiples usuarios a la vez, se da gracias al concepto de multiprogramación.
- Multiprogramación se refiere a la capacidad de un sistema operativo de ejecutar múltiples procesos o programas “a la vez”
- Piense en un momento en lo que esto significa...

¿Qué es una transacción?



¿Qué es una transacción?



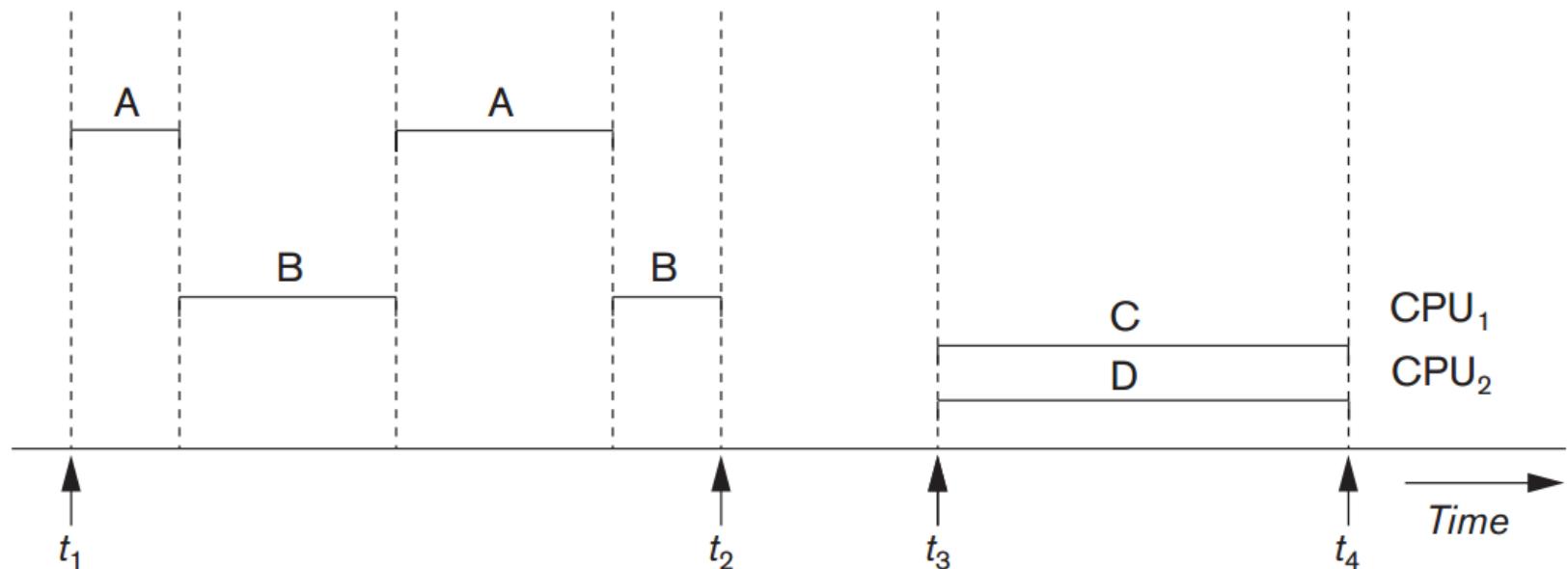
- ¿Cuántos programas se están ejecutando en t_4 ?
- Depende de la cantidad de procesadores que tenga la máquina.
- Suponiendo que solo tiene uno, en un tiempo t_4 **solo se está ejecutando un programa / proceso a la vez**
- Por así decirlo el sistema operativo intercala la ejecución de los programas

¿Qué es una transacción?



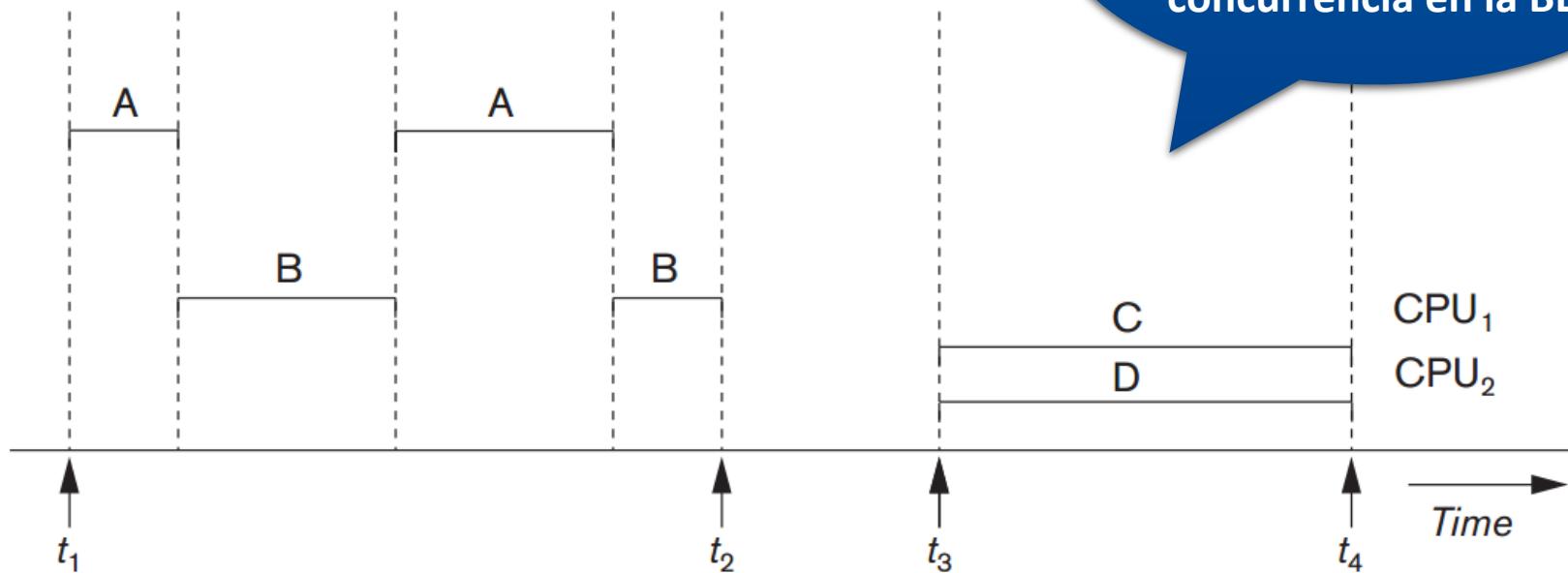
- El sistema operativo le da la impresión al usuario de que todos se ejecutan simultáneamente
- El intercalado se hace tan rápido que al ojo humano todo se ejecuta a la vez
- Esto implica que el sistema operativo “le da” el procesador a un programa, se lo quita, se lo da a otro, se lo regresa...
- Hay un algoritmo de scheduling....

¿Qué es una transacción?



¿Qué es una transacción?

Es clave entender este modelo, para entender concurrencia en la BD



¿Qué es una transacción?

- Una transacción incluye una o más operaciones en la base de datos (INSERT / UPDATE / DELETE / SELECT)
- Estas operaciones pueden estar en un script SQL o empotradas en un programa
- Un programa puede tener más de una transacción

¿Qué es una transacción?

- Una transacción incluye una o más operaciones en la base de datos (INSERT / UPDATE / DELETE / SELECT)
- Estas operaciones pueden estar en un script SQL o empotradas en un programa
- Un programa puede tener más de una transacción
- Una transacción es entonces, una unidad atómica de trabajo que se complete en su totalidad o no se hace nada.

Características de una transacción

- Las transacciones deben poseer una serie de características conocidas como ACID

Atomicidad: una transacción es una unidad atómica de procesamiento, se hace completamente o no se hace nada

Consistente (preservación de la consistencia): si una transacción se ejecuta completamente de principio a fin sin interferencias, debe llevar la BD de un estado consistente a otro estado consistente.

Isolation (Aislamiento): una transacción debe parecer que se ejecuta asiladamente de otras transacciones en ejecución concurrentes. Una transacción por tanto, no debe verse interferida por otra transacción concurrente

Durable: los cambios hechos por una transacción deben persistir

Características de una transacción

- Las transacciones d
conocidas como AC

Es responsabilidad del motor
(subsistema de recuperación de
transacciones)

Atomicidad: una transacción es una unidad atómica de procesamiento, se hace completamente o no se hace nada

Consistente (preservación de la consistencia): si una transacción se ejecuta completamente de principio a fin sin interferencias, debe llevar la BD de un estado consistente a otro estado consistente.

Isolation (Aislamiento): una transacción debe parecer que se ejecuta aisladamente de otras transacciones en ejecución concurrentes. Una transacción por tanto, no debe verse interferida por otra transacción concurrente

Durable: los cambios hechos por una transacción deben persistir

Características de una transacción

- Las transacciones deben poseer una serie de características conocidas como ACID

Atomicidad: una transacción es una completamente o no se hace nada

Es responsabilidad del programador. El motor no obliga a nadie a usar integridad referencial

Consistente (preservación de la consistencia): una transacción se ejecuta completamente de principio a fin sin interferencias, debe llevar la BD de un estado consistente a otro estado consistente.

Isolation (Aislamiento): una transacción debe parecer que se ejecuta asiladamente de otras transacciones en ejecución concurrentes. Una transacción por tanto, no debe verse interferida por otra transacción concurrente

Durable: los cambios hechos por una transacción deben persistir

Características de una transacción

- Las transacciones deben poseer una serie de características conocidas como ACID

Atomicidad: una transacción es una unidad atómica de procesamiento, se hace completamente o no se hace nada

Consistente (preservar la consistencia): una transacción debe dejar la base de datos en un estado consistente con respecto a las demás transacciones que están ejecutándose.

Es responsabilidad del motor (subsistema de control de concurrencia) y del programador

acción se ejecuta
levar la BD de un

Isolation (Aislamiento): una transacción debe parecer que se ejecuta aisladamente de otras transacciones en ejecución concurrentes. Una transacción por tanto, no debe verse interferida por otra transacción concurrente

Durable: los cambios hechos por una transacción deben persistir

Características de una transacción

- Las transacciones deben poseer una serie de características conocidas como ACID

Atomicidad: una transacción es una unidad atómica de procesamiento, se hace completamente o no se hace nada

Consistente (preservación de la consistencia): si una transacción se ejecuta completamente de principio a fin sin interferencias, debe llevar la BD de un estado consistente a otro estado consistente.

Isolation (Aislamiento): una transacción debe funcionar aisladamente de otras transacciones. Una transacción por tanto, no debe interferir con otra transacción concurrente

Es responsabilidad del motor
(subsistema de recuperación)

Durable: los cambios hechos por una transacción deben persistir

Modelo de una transacción

- Algunos conceptos esenciales para entender lo que veremos más adelante...
- En este modelo, una **base de datos** es un conjunto de ítems de datos con nombre
- Un **ítem de dato** puede ser un atributo de una tabla, un registro de una tabla o inclusive un bloque del disco
- **Granularidad** es el tamaño de un ítem de datos

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una variable del programa de nombre 'x' en el ítem de la base de datos nombrado 'x'

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una variable del programa de nombre 'x' en el ítem de la base de datos nombrado 'x'

1. Busca la dirección del bloque de disco que contiene el ítem 'x'
2. Copia el bloque de disco en un buffer en la memoria principal
3. Copia el ítem 'x' del buffer a la variable del programa llamada 'x'

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una variable del programa de nombre 'x' en el ítem de la base de datos nombrado 'x'

1. Busca la dirección del bloque de disco que contiene el ítem 'x'
2. Copia el bloque de disco en un buffer en la memoria principal (si el bloque no está ya en la memoria)
3. Copia el ítem 'x' de la variable 'x' del programa a su ubicación correcta en el buffer
4. Almancena el bloque del buffer al disco (inmediatamente o en algún momento posterior)

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una variable del programa de nombre 'x' en el ítem de la base de datos nombrado 'x'

(a)	T_1	(b)	T_2
	<pre>read_item(X); X := X - N; write_item(X); read_item(Y); Y := Y + N; write_item(Y);</pre>		<pre>read_item(X); X := X + M; write_item(X);</pre>

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una el ítem de la base de datos nombrado

¿Por qué se necesita control de concurrencia?

(a)

T_1
read_item(X); $X := X - N;$ write_item(X); read_item(Y); $Y := Y + N;$ write_item(Y);

(b)

T_2
read_item(X); $X := X + M;$ write_item(X);

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una el ítem de la base de datos nombrado

¿Sobre qué se ejecuta el DBMS?

(a)

T_1
read_item(X); $X := X - N;$ write_item(X); read_item(Y); $Y := Y + N;$ write_item(Y);

(b)

T_2
read_item(X); $X := X + M;$ write_item(X);

Modelo de una transacción

- Dentro de este modelo hay dos operaciones:

Read_item(x): lee un ítem de la base de datos que tiene por nombre 'x' en una variable de un programa

Write_item(x): escribe el valor de una
el ítem de la base de datos nombrado

El DBMS es software. Es un
programa que corre sobre el
SO..entonces se ejecuta
intercalado...

(a)

T_1
read_item(X); $X := X - N;$ write_item(X); read_item(Y); $Y := Y + N;$ write_item(Y);

(b)

T_2
read_item(X); $X := X + M;$ write_item(X);

Problemas ocasionados por la concurrencia

- **Actualizaciones perdidas**

Ocurre cuando dos transacciones que acceden los mismos ítems de datos son intercalados ocasionando que los valores de los items de datos sean incorrectos

The diagram illustrates two transactions, T_1 and T_2 , with a vertical arrow labeled "Time" pointing downwards, indicating the sequence of events over time.

T_1	T_2
<pre>read_item(X); X := X - N; write_item(X); read_item(Y); Y := Y + N; write_item(Y);</pre>	<pre>read_item(X); X := X + M; write_item(X);</pre>

Problemas ocasionados por la concurrencia

- **Lecturas sucias**

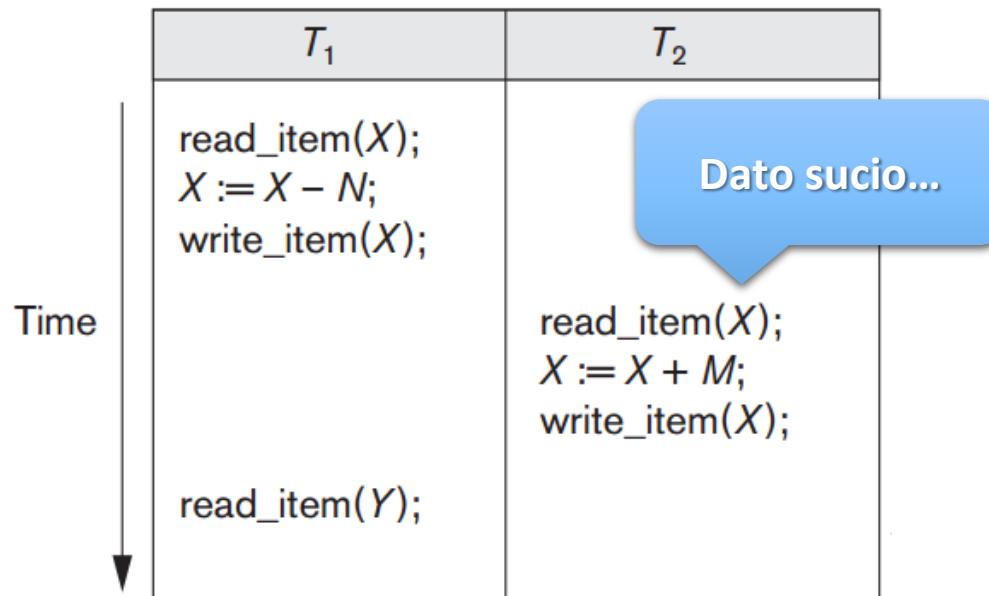
Ocurre cuando una transacción actualiza un ítem y falla por alguna razón. Mientras tanto, otra transacción lee el ítem actualizado antes de que se revierta a su valor original

T_1	T_2
<pre>read_item(X); X := X - N; write_item(X);</pre>	
Time ↓	<pre>read_item(X); X := X + M; write_item(X);</pre>

Problemas ocasionados por la concurrencia

• Lecturas sucias

Ocurre cuando una transacción actualiza un ítem y falla por alguna razón. Mientras tanto, otra transacción lee el ítem actualizado antes de que se revierta a su valor original



Problemas ocasionados por la concurrencia

- **Totales incorrectos**

Si una transacción está calculando una función agregada sobre un número determinado de ítems mientras que otra transacción está actualizando algunos de estos ítems, la función agregada va a calcular algunos valores antes de que sean actualizados y otros después que son actualizados

T_1	T_3
	$sum := 0;$ $read_item(A);$ $sum := sum + A;$ ⋮ $read_item(X);$ $X := X - N;$ $write_item(X);$
	$read_item(X);$ $sum := sum + X;$ $read_item(Y);$ $sum := sum + Y;$

¿Por qué se necesita la recuperación?

- El DBMS es responsable de asegurar que todas las operaciones en la transacción se completen exitosamente y su efecto sea permanente o que ninguna transacción tenga algún efecto
- Una transacción es una sola unidad atómica. El DBMS no permite que algunas operaciones se apliquen mientras que otras no. Si una transacción falla, se deshacen las operaciones aplicadas

¿Por qué se necesita la recuperación?

- El DBMS es responsable de asegurar que todas las operaciones en la transacción se completen exitosamente y su efecto sea permanente o que ninguna transacción tenga algún efecto
- Una transacción es una sola unidad atómica. El DBMS no permite que algunas operaciones se apliquen mientras que otras no. Si una transacción falla, se deshacen las operaciones aplicadas

Si la transacción se completa, está comprometida (committed)

¿Por qué se necesita la recuperación?

- El DBMS es responsable de asegurar que todas las operaciones en la transacción se completen exitosamente y su efecto sea permanente o que ninguna transacción tenga algún efecto
- Una transacción es una sola unidad atómica. El DBMS no permite que algunas operaciones se apliquen mientras que otras no. Si una transacción falla, se deshacen las operaciones aplicadas

Si la transacción falla, esta se aborta y debe hacerse un rollback

¿Por qué se necesita la recuperación?

- El DBMS es responsable de asegurar que todas las operaciones en la transacción se completen exitosamente y su efecto sea permanente o que ninguna transacción tenga algún efecto
- Una transacción es una sola unidad atómica. El DBMS no permite que algunas operaciones se apliquen mientras que otras no. Si una transacción falla, se deshacen las operaciones aplicadas

Pero, ¿qué puede salir mal?

¿Por qué se necesita la recuperación?

- **Falla computacional**

Un error de hardware, software o de la red puede ocurrir en el sistema computacional durante la ejecución de la transacción.

Los errores de hardware más comunes se deben a fallas en el almacenamiento (discos duros, RAM)

- **Un error de la transacción o del sistema**

Overflow, división por cero, parámetros incorrectos, errores de programación, cancelación por parte del usuario.

- **Errores locales o condiciones de excepción detectadas por la transacción**

Datos no encontrados. Ejemplo: fondos insuficiente, detección de un fraude

¿Por qué se necesita la recuperación?

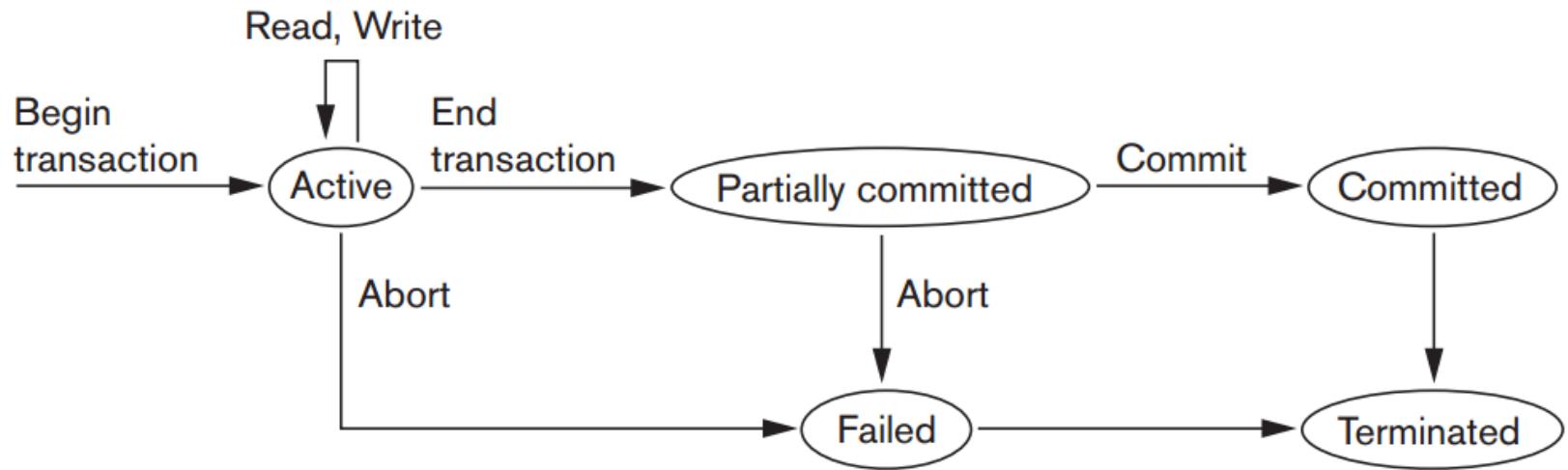
- **Control de concurrencia**

El método de concurrencia utilizado puede decidir abortar una transacción. Por ejemplo, se detecta un deadlock

- **Problema físico y catástrofes**

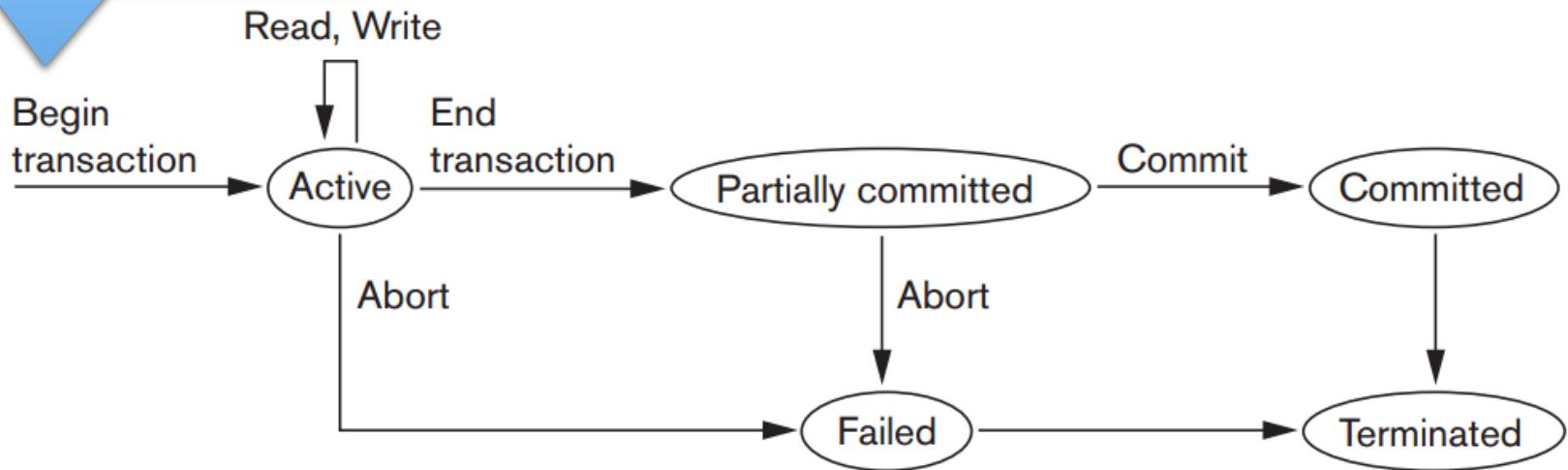
Una lista interminable de problemas (fallas en aires acondicionados, energía, sabotaje, incendio, robo)

Estados de una transacción



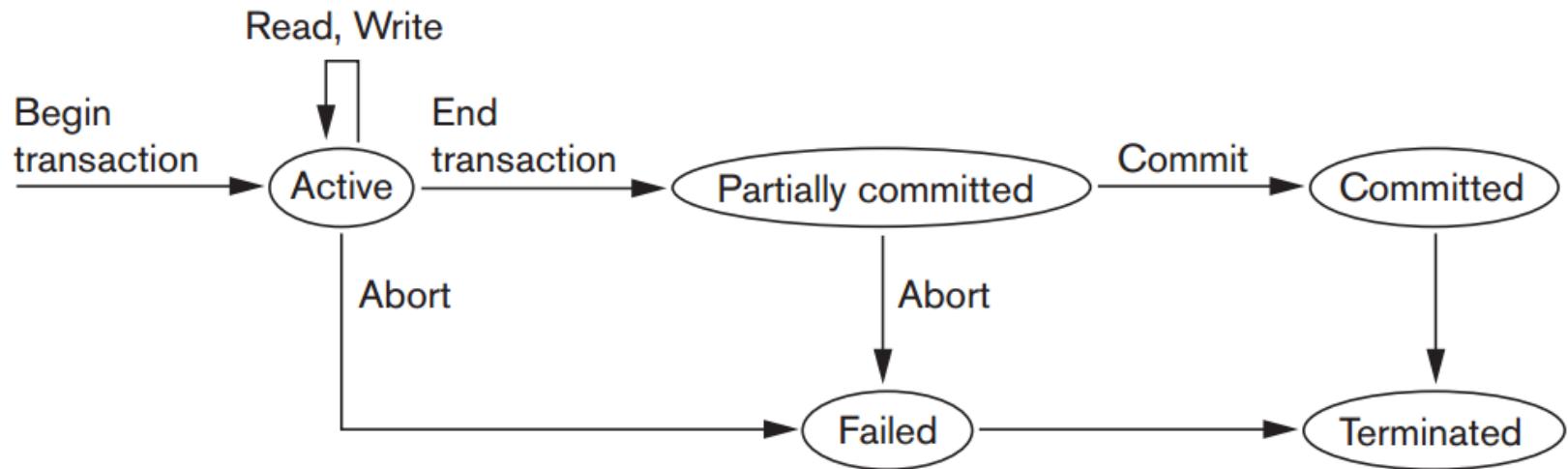
Estados de una transacción

Marca el inicio de la transacción

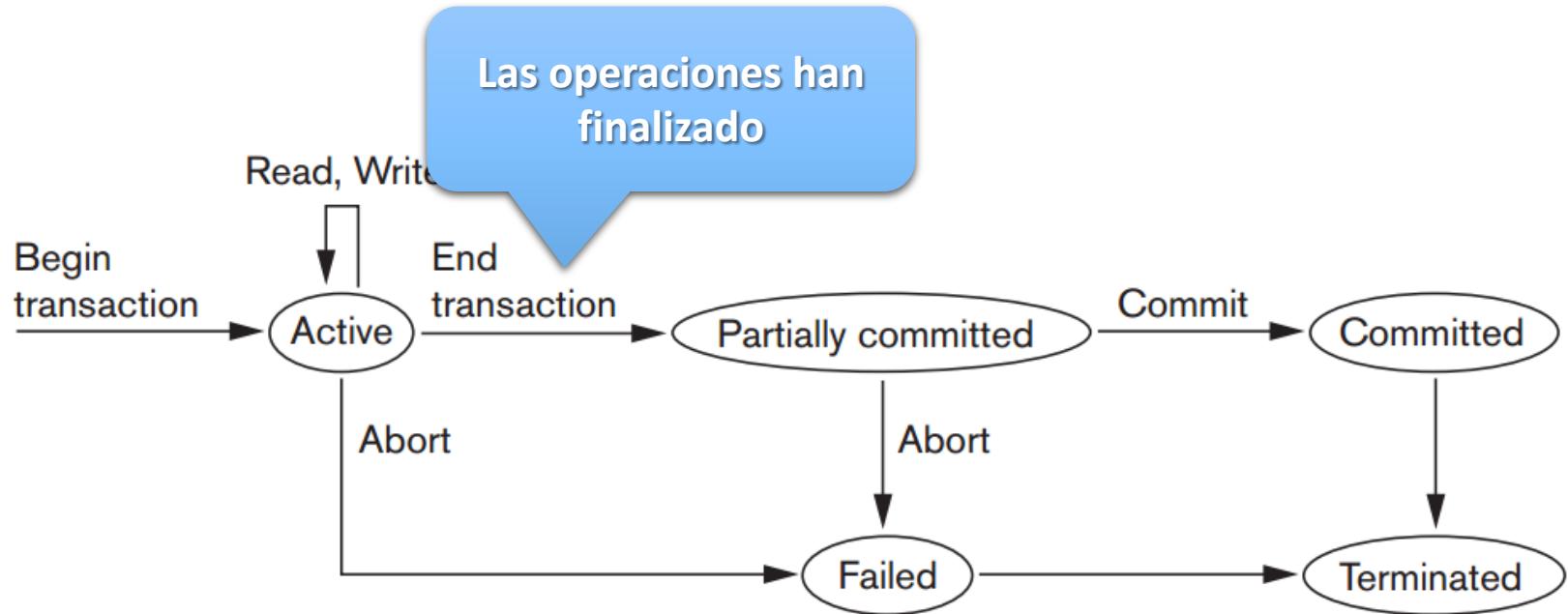


Estados de una transacción

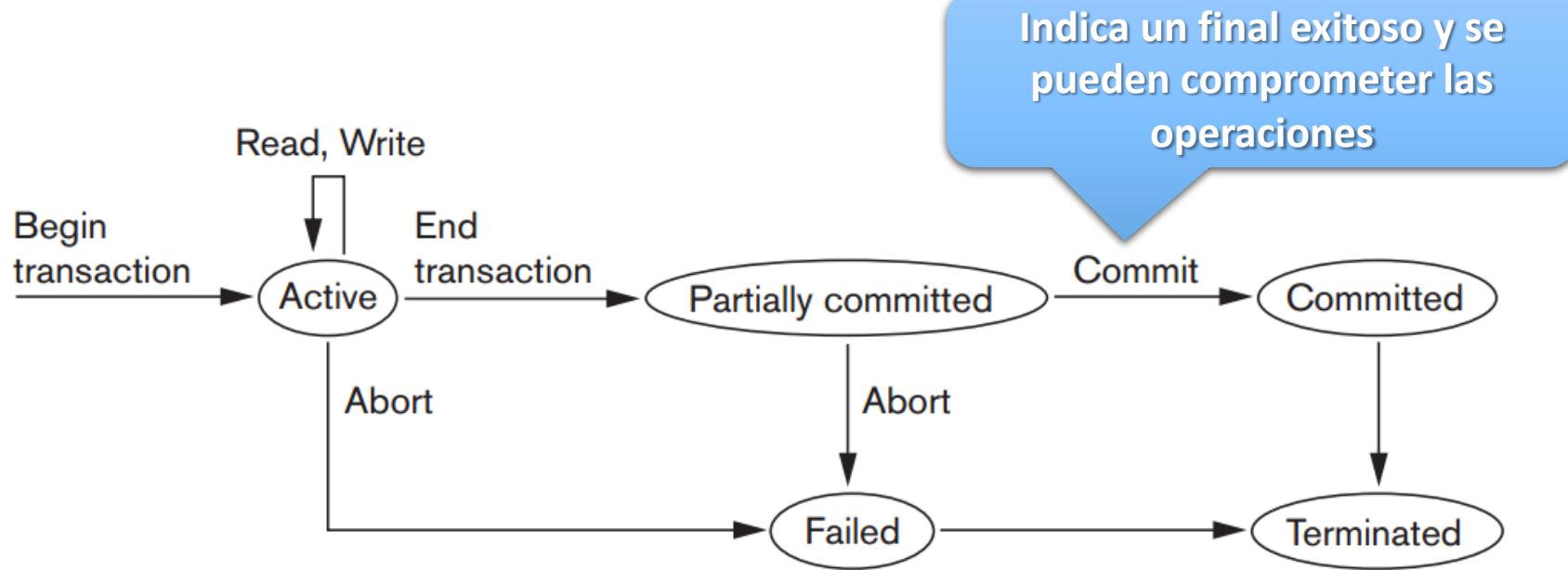
Operaciones
ejecutadas en la base
de datos



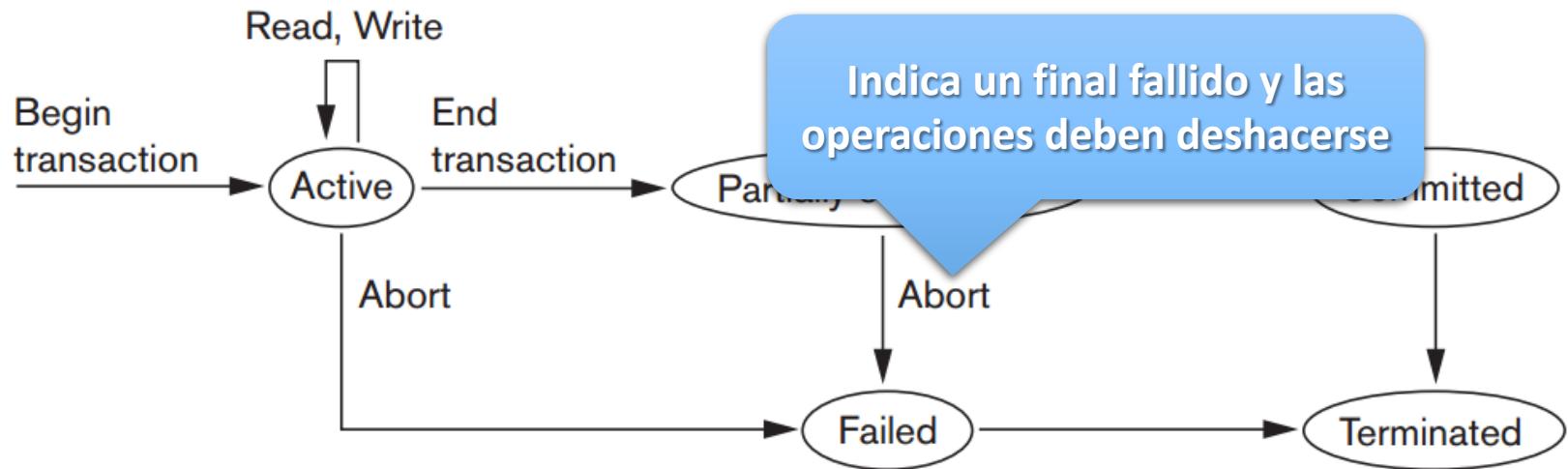
Estados de una transacción



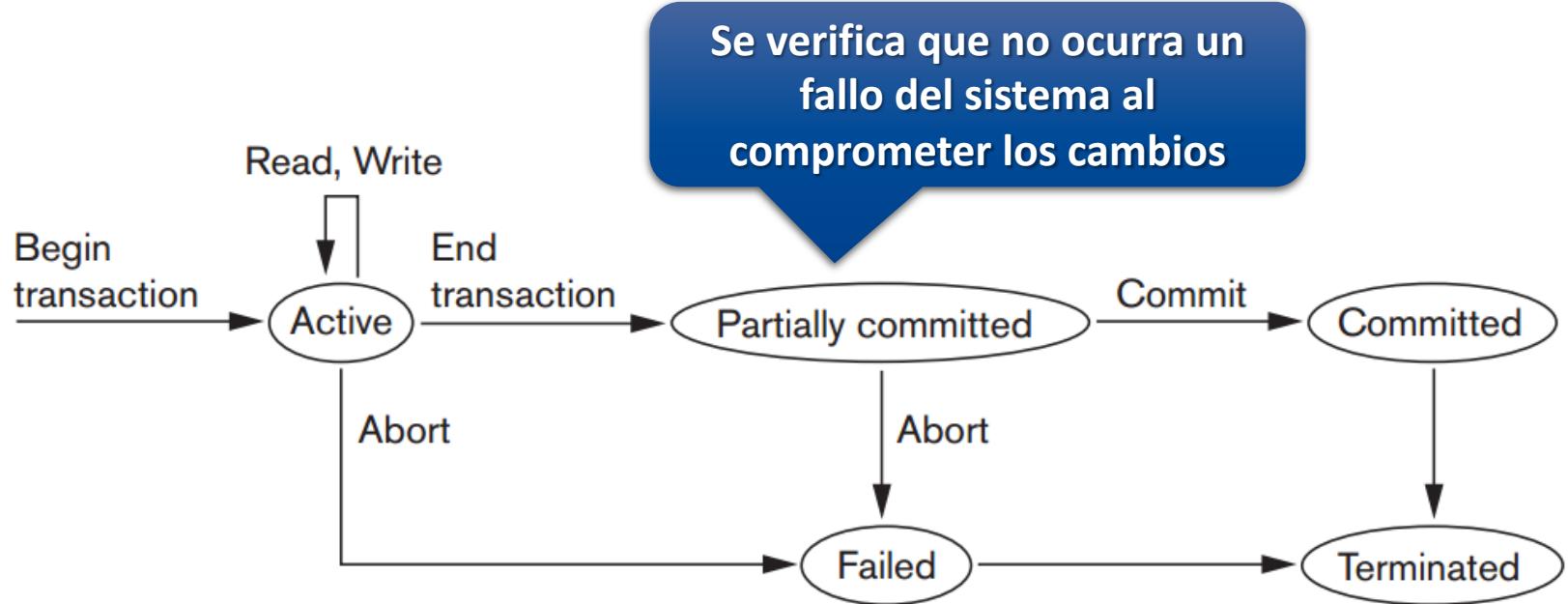
Estados de una transacción



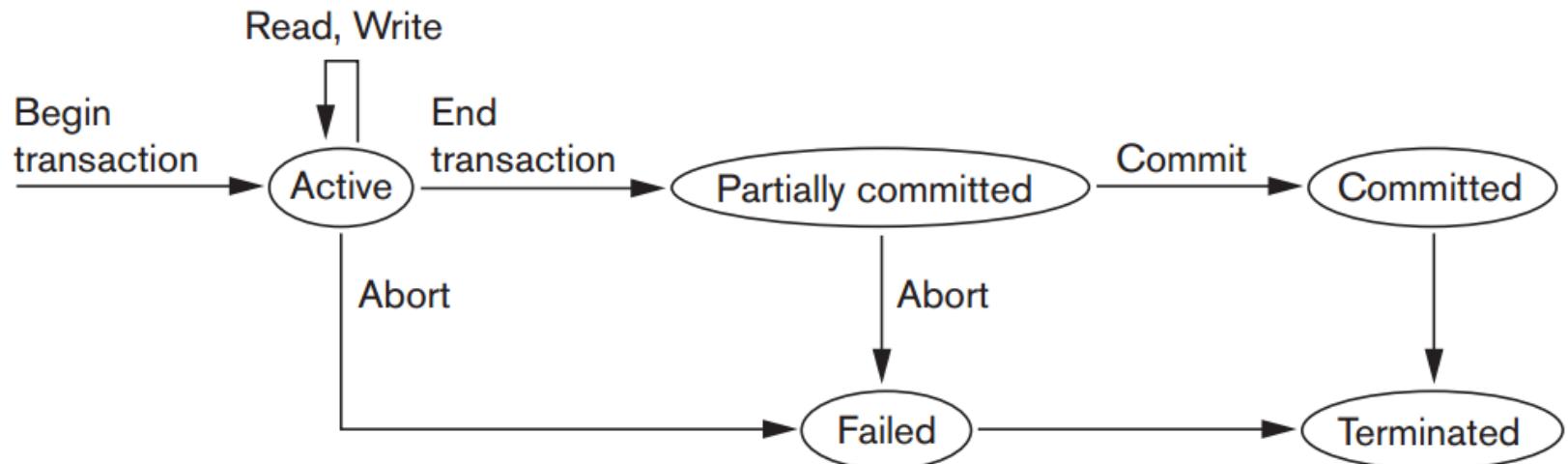
Estados de una transacción



Estados de una transacción

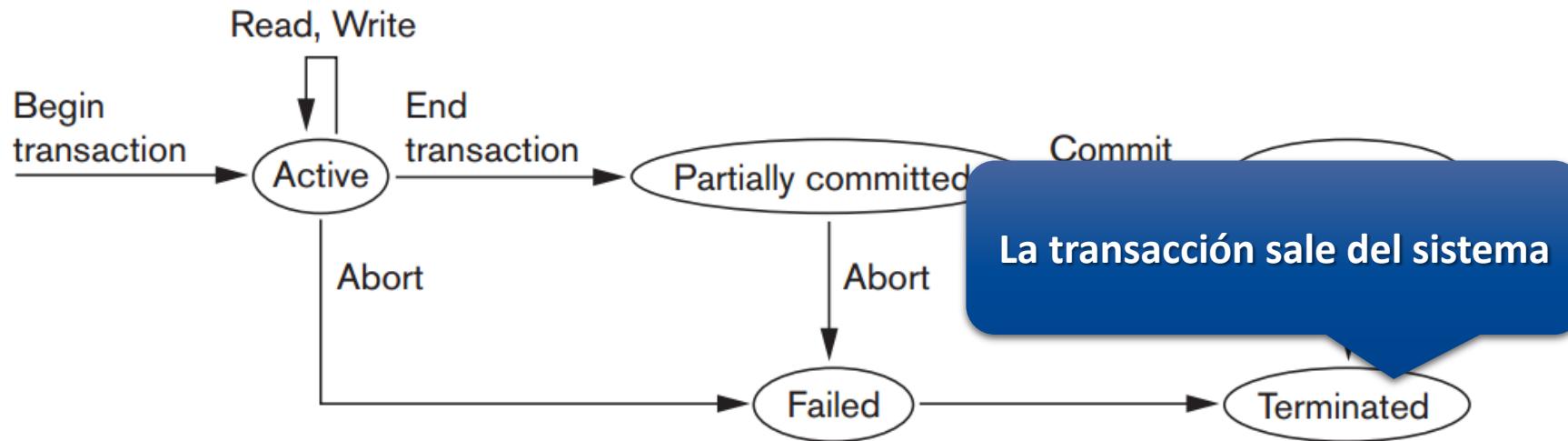


Estados de una transacción

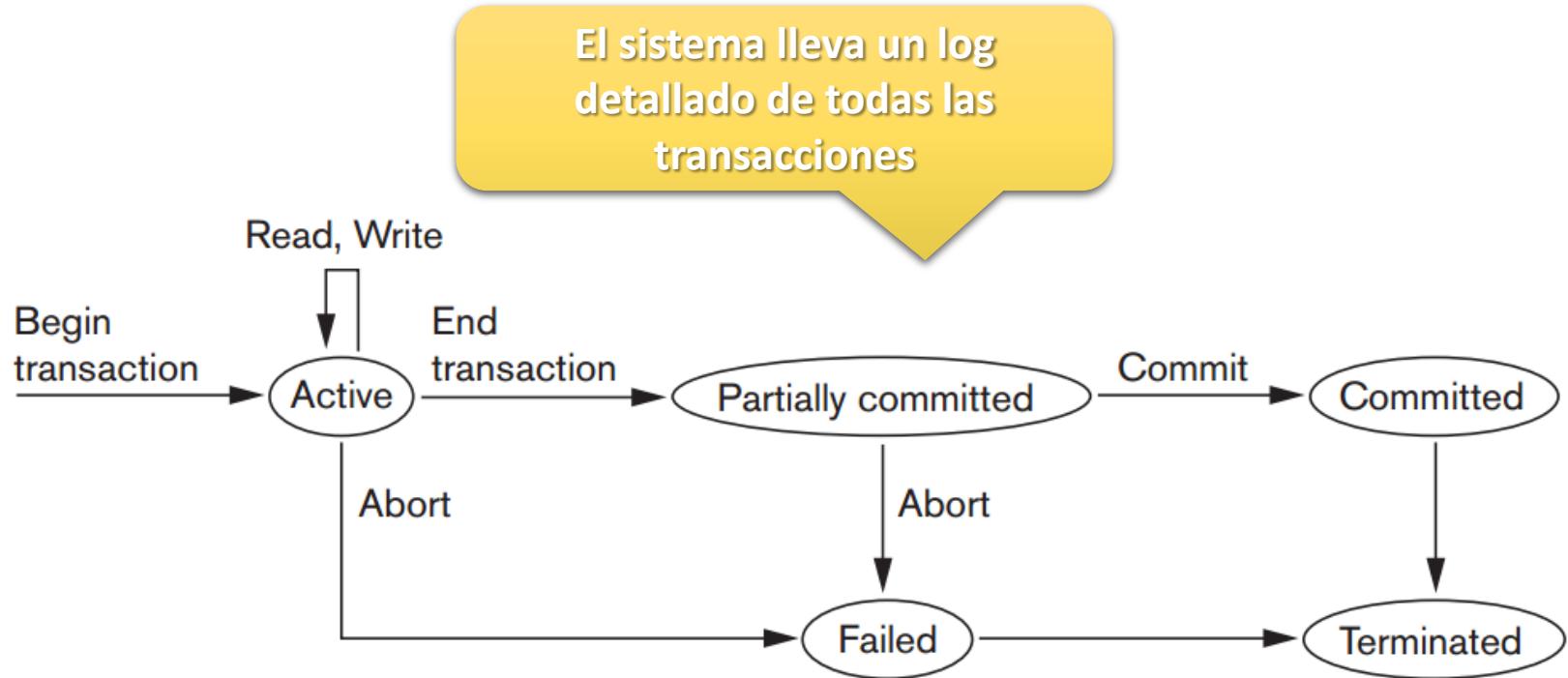


Transacción abortada o no se
pudo comprometer

Estados de una transacción



Estados de una transacción



Estados de una transacción

El sistema lleva un log detallado de todas las transacciones

	Results						
	Current LSN	Operation	Transaction ID	Parent Transaction ID	Begin Time	Transaction Name	Transaction SID
1	00000020:00000055:0001	LOP_BEGIN_XACT	0000:000002cf	NULL	2013/05/06 12:54:04:977	Backup:CommitDifferentialBase	0x01050000000000005150
2	00000020:0000005d:0002	LOP_BEGIN_XACT	0000:000002d0	NULL	2013/05/06 12:54:05:037	CREATE TABLE	0x01050000000000005150
3	00000020:0000005d:0003	LOP_BEGIN_XACT	0000:000002d1	0000:000002d0	2013/05/06 12:54:05:037	CMEDCatYukonObject::PreAllocateO	0x01050000000000005150
4	00000020:0000005f:000e	LOP_BEGIN_XACT	0000:000002d2	0000:000002d0	2013/05/06 12:54:05:050	SplitPage	0x01050000000000005150
5	00000020:0000005f:003f	LOP_BEGIN_XACT	0000:000002d3	0000:000002d0	2013/05/06 12:54:05:050	SplitPage	0x01050000000000005150
6	00000020:0000005f:0058	LOP_BEGIN_XACT	0000:000002d4	0000:000002d0	2013/05/06 12:54:05:053	SplitPage	0x01050000000000005150
7	00000020:0000005f:0074	LOP_BEGIN_XACT	0000:000002d5	0000:000002d0	2013/05/06 12:54:05:053	SplitPage	0x01050000000000005150
8	00000020:0000009b:0002	LOP_BEGIN_XACT	0000:000002d6	NULL	2013/05/06 12:54:05:070	INSERT	0x01050000000000005150
9	00000020:0000009b:0004	LOP_BEGIN_XACT	0000:000002d7	NULL	2013/05/06 12:54:05:070	Allocate Root	0x01050000000000005150
10	00000020:0000009b:0006	LOP_BEGIN_XACT	0000:000002d8	0000:000002d7	2013/05/06 12:54:05:070	AllocFirstPage	0x01050000000000005150
11	00000020:000000a:0002	LOP_BEGIN_XACT	0000:000002d9	NULL	2013/05/06 12:54:05:073	INSERT	0x01050000000000005150
12	00000020:000000a1:0001	LOP_BEGIN_XACT	0000:000002da	NULL	2013/05/06 12:54:05:077	INSERT	0x01050000000000005150
13	00000020:000000a2:0001	LOP_BEGIN_XACT	0000:000002db	NULL	2013/05/06 12:54:05:077	INSERT	0x01050000000000005150
14	00000020:000000a3:0001	LOP_BEGIN_XACT	0000:000002dc	NULL	2013/05/06 12:54:05:080	INSERT	0x01050000000000005150
15	00000020:000000a4:0001	LOP_BEGIN_XACT	0000:000002dd	NULL	2013/05/06 12:54:05:090	user_transaction	0x01050000000000005150
16	00000020:000000a7:0001	LOP_BEGIN_XACT	0000:000002de	NULL	2013/05/06 12:54:05:107	UpdateQPStats	0x01050000000000005150
17	00000020:000000aa:0001	LOP_BEGIN_XACT	0000:000002df	NULL	2013/05/06 12:54:05:113	DELETE	0x01050000000000005150

Técnicas de control de conurrencia

Introducción

Existen distintos protocolos de control de concurrencia:

- **Bloqueo de dos fases:** Bloquea los ítems de datos para prevenir que múltiples transacciones accedan los ítems concurrentemente
- **Basados en timestamps:** Generan un *timestamp* único para cada transacción y lo utilizan para determinar el orden en que las transacciones se ejecutan
- **Optimistas:** se basan en el concepto de validación o certificación de una transacción después de que se ejecuta

Introducción

Conjunto de reglas

Existen distintos protocolos de control de concurrencia:

- **Bloqueo de dos fases:** Bloquea los ítems de datos para prevenir que múltiples transacciones accedan los ítems concurrentemente
- **Basados en timestamps:** Generan un *timestamp* único para cada transacción y lo utilizan para determinar el orden en que las transacciones se ejecutan
- **Optimistas:** se basan en el concepto de validación o certificación de una transacción después de que se ejecuta

Bloqueo de dos fases

- Un bloqueo o **lock** es una variable asociada a un ítem de datos que describe el estado del ítem con respectos a posibles operaciones que pueden ser aplicadas sobre el
- Un **lock** permite sincronizar el acceso a un ítem de datos
- El uso de locks tiene dos problemas asociados: interbloqueos (**deadlocks**) e inanición (**starvation**). Más adelante veremos más sobre esto...

Tipos de locks: Binarios

- Un lock binario solo puede tener dos estados o valores: abierto (0) o cerrado (1).
- Si el ítem X tiene asociado un lock cuyo valor es 1, el ítem X **no puede ser accedido** por la operación que está pidiendo el ítem.
- Si el ítem X tiene asociado un lock cuyo valor es 0, el ítem X **puede ser accedido** por la operación que lo solicita.
- Hay dos operaciones asociadas con los locks binarios: `lock_item` y `unlock_item`
- Un lock binario fuerza una exclusión mutua (MUTEX)

Tipos de locks: Binarios

lock_item(X):

B: if $\text{LOCK}(X) = 0$ (* item is unlocked *)
 then $\text{LOCK}(X) \leftarrow 1$ (* lock the item *)
else
 begin
 wait (until $\text{LOCK}(X) = 0$
 and the lock manager wakes up the transaction);
 go to B
 end;

unlock_item(X):

$\text{LOCK}(X) \leftarrow 0$; (* unlock the item *)
if any transactions are waiting
 then wakeup one of the waiting transactions;

Tipos de locks: Binarios

lock_item(X):

B: if $LOCK(X) = 0$ (* item is unlocked *)
 then $LOCK(X) \leftarrow 1$ (* lock the item *)
else
 begin
 wait (until $LOCK(X) = 0$
 and the lock manager wakes up the transaction);
 go to B
 end;

unlock_item(X):

$LOCK(X) \leftarrow 0$;
if any transactions are waiting
 then wakeup one of the waiting transactions;

Ambas operaciones son
atómicas

(* unlock the item *)

Tipos de locks: Binarios

lock_item(X):

B: if $LOCK(X) = 0$ (* item is unlocked *)
then $LOCK(X) \leftarrow 1$ (* lock the item *)

else

begin

wait (until $LOCK(X) = 0$
and the lock manager v

go to B

end;

unlock_item(X):

$LOCK(X) \leftarrow 0;$

if any transactions are waiting

then wakeup one of the waiting transactions;

El scheduler del sistema operativo
no va a interrumpir la operación
hasta que terminen

Ambas operaciones son
atómicas

(* unlock the item *)

Tipos de locks: Binarios

lock_item(X):

B: if $LOCK(X) = 0$ (* item is unlocked *)
 then $LOCK(X) \leftarrow 1$ (* lock the item *)
else
 begin
 wait (until $LOCK(X) = 0$
 and the lock manager wakes up the transaction);
 go to B
 end;

unlock_item(X):

$LOCK(X) \leftarrow 0$; (* unlock the item *)
if any transactions are waiting
 then wakeup one of the waiting transactions;

Encola la transacción hasta
que el recurso esté
desbloqueado

Tipos de locks: Binarios

lock_item(X):

B: if $\text{LOCK}(X) = 0$ (* item *)
 then $\text{LOCK}(X) \leftarrow 1$ (* lock the item *)
else
 begin
 wait (until $\text{LOCK}(X) = 0$ and the lock manager wakes up the transaction);
 go to B
 end;

unlock_item(X):

$\text{LOCK}(X) \leftarrow 0$; (* unlock the item *)
if any transactions are waiting
 then wakeup one of the waiting transactions;

El subsistema de administración de locks del DBMS mantiene el registro de los locks

Encola la transacción hasta que el recurso esté desbloqueado

Tipos de locks: Binarios

lock_item(X):

```
B: if LOCK( $X$ ) = 0
    then LOCK( $X$ ) ← 1      (* item  $X$  is unlocked *)
else
begin
    go to  $X$ 
end;
the transaction);
```

Son muy
restrictivos

unlock_item(X):

```
LOCK( $X$ ) ← 0;
if any transactions are waiting
    then wakeup one of the waiting transactions;
```

(* unlock the item *)

Tipos de locks: Shared / Exclusive

- Permiten que varias transacciones accedan el mismo ítem **si lo acceden únicamente para lectura**
 - Las operaciones de lectura no son conflictivas
- Más flexible que los locks binarios
- Si una transacción quiere escribir el ítem, deberá tener acceso exclusivo al ítem.
- Hay tres operaciones:
 - Read_lock
 - Write_lock
 - unlocked

Tipos de locks: Shared / Exclusive

read_lock(X):

B: if $\text{LOCK}(X) = \text{"unlocked"}$

then **begin** $\text{LOCK}(X) \leftarrow \text{"read-locked"};$
 $\text{no_of_reads}(X) \leftarrow 1$
end

else if $\text{LOCK}(X) = \text{"read-locked"}$

then $\text{no_of_reads}(X) \leftarrow \text{no_of_reads}(X) + 1$

else **begin**

wait (until $\text{LOCK}(X) = \text{"unlocked"}$
and the lock manager wakes up the transaction);
go to B
end;

Tipos de locks: Shared / Exclusive

write_lock(X):

B: if $\text{LOCK}(X) = \text{"unlocked"}$

 then $\text{LOCK}(X) \leftarrow \text{"write-locked"}$

else **begin**

 wait (until $\text{LOCK}(X) = \text{"unlocked"}$

 and the lock manager wakes up the transaction);

 go to B

end;

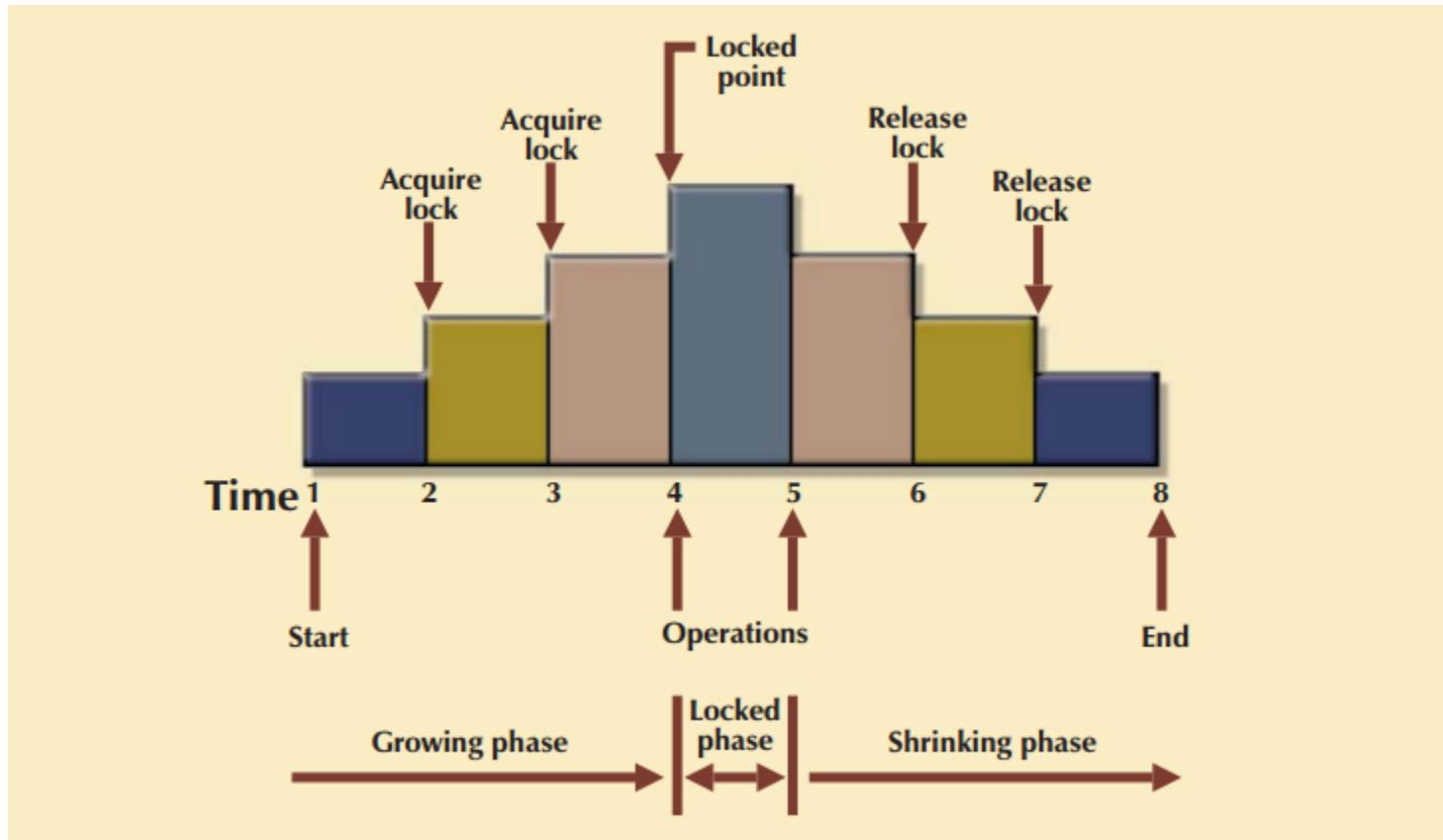
Tipos de locks: Shared / Exclusive

unlock (X):

```
if LOCK(X) = "write-locked"
    then begin LOCK(X) ← "unlocked";
        wakeup one of the waiting transactions, if any
        end
else if LOCK(X) = "read-locked"
    then begin
        no_of_reads(X) ← no_of_reads(X) -1;
        if no_of_reads(X) = 0
            then begin LOCK(X) = "unlocked";
                wakeup one of the waiting transactions, if any
                end
    end;
```

Bloqueo de dos fases

- Las dos fases de este mecanismo de concurrencia son:



Bloqueo de dos fases: Deadlocks

- Un deadlock puede ocurrir cuando dos transacciones esperan indefinidamente por la otra para que liberen un ítem de datos.

TIME	TRANSACTION	REPLY	LOCK STATUS	
			Data X	Data Y
0				
1	T1:LOCK(X)	OK	Unlocked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Unlocked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...
...
...
...



A large black arrow points downwards from the word "Deadlock" to the bottom of the table, indicating the point where the deadlock occurs.





Igual que en Los Yoses a las 5:30!

¿Cómo se controlan los deadlocks?

- **Prevención de deadlocks:** una transacción que solicita un lock se aborta si hay posibilidad de que ocurra un deadlock
 - La transacción se recalendariza para ejecución
 - Evita las condiciones que pueden conducir a un deadlock
- **Detección de deadlocks:** el DBMS periódicamente busca deadlocks en la base de datos. Si se detecta alguno, una de las transacciones se aborta y reinicia posteriormente.
- **Evasión de deadlocks:** Una transacción debe obtener todos los locks que necesita antes de ejecutarse. Esta técnica evita deshacer transacciones conflictivas, pero aumenta el tiempo de respuesta

¿Cómo se controlan los deadlocks?

- **Prevención de deadlocks**: una transacción aborta si hay posibilidad de que ocurra un deadlock.
 - La transacción se recalendariza para ejecución
 - Evita las condiciones que pueden conducir a un deadlock
- **Detección de deadlocks**: el DBMS periódicamente busca deadlocks en la base de datos. Si se detecta alguno, una de las transacciones se aborta y reinicia posteriormente.
- **Evasión de deadlocks**: Una transacción debe obtener todos los locks que necesita antes de ejecutarse. Esta técnica evita deshacer transacciones conflictivas, pero aumenta el tiempo de respuesta

¿Cuál técnica es mejor?

¿Cómo se controlan los deadlocks?

- **Prevención de deadlocks**: una transacción aborta si hay posibilidad de que ocurra un deadlock.
 - La transacción se recalendariza para ejecución
 - Evita las condiciones que pueden conducir a un deadlock
- **Detección de deadlocks**: el DBMS periódicamente busca deadlocks en la base de datos. Si se detecta alguno, una de las transacciones se aborta y reinicia posteriormente.
- **Evasión de deadlocks**: Una transacción debe obtener todos los locks que necesita antes de ejecutarse. Esta técnica evita deshacer transacciones conflictivas, pero aumenta el tiempo de respuesta

Depende de la probabilidad de deadlocks

¿Cómo se controlan los deadlocks?

- **Prevención de deadlocks:** una transacción que solicita un lock se aborta si hay posibilidad de que ocurra un deadlock
 - La transacción se recalendariza para ejecución
 - Evita las condiciones que pueden conducir a un deadlock
- **Detección de deadlocks:** el DBMS periódicamente busca deadlocks en la base de datos. Si se detecta alguno, una de las transacciones se aborta y se reejecuta posteriormente.

Más empleada en los DBMS
- **Ejecución de transacciones serializadas:** una transacción debe obtener todos los locks que necesita antes de ejecutarse. Esta técnica evita deshacer transacciones conflictivas, pero aumenta el tiempo de respuesta

Bloqueo de dos fases: starvation

- **Starvation** (inanición – *morir de hambre*) es una condición que puede ocurrir al usar bloqueos.
- Esta situación ocurre cuando una transacción no puede continuar por un periodo indefinido de tiempo mientras que otras transacciones continúan normalmente
- Esto ocurre si el esquema de locks es injusto: da prioridad a unas transacciones sobre otras.
- Una solución para evitar una condición de *starvation* es implementar un algoritmo que balancee la prioridad

Bloqueo de dos fases: starvation

- **Starvation** (inanición – *morir de hambre*) es una condición que puede ocurrir al usar bloqueos.
 - Esta situación ocurre cuando un periodo indefinido de tiempo continúan normalmente.
 - Una solución para evitar una condición de *starvation* es implementar un algoritmo que balancee la prioridad
- Primero en llegar,
primero en salir (cola)
- es...
sobre otra
- Manejar prioridades pero
incrementar la prioridad de las
transacciones que
esperan...eventualmente
tendrán la máxima prioridad
- puede continuar por
ras transacciones
- prioridad a unas

Protocolos basados en timestamps

- Asigna un *timestamp* único para cada transacción que produce un ordenamiento explícito en el que las transacciones serán ejecutadas por el DBMS
- Todas las operaciones dentro de una transacción deben tener el mismo *timestamp*
- El problema de este protocolo es que cada valor almacenado en la base de datos, **requiere dos campos adicionales**: un *timestamp* para la última vez que se leyó y otro para la última vez que se actualizó
- Hay muchas formas de implementar un protocolo basado en *timestamp*, los más comunes son: wait/die o wound/die

Protocolos basados en timestamps

- **wait/die:**

Si la transacción que solicita el *lock* es la más antigua, esperará hasta que la otra transacción se complete y el *lock* se libere

Si la transacción que solicita el *lock* es la más joven, hará un rollback (die) y se vuelve programar para su ejecución

→ La transacción más vieja espera que la más joven termine y libere los locks

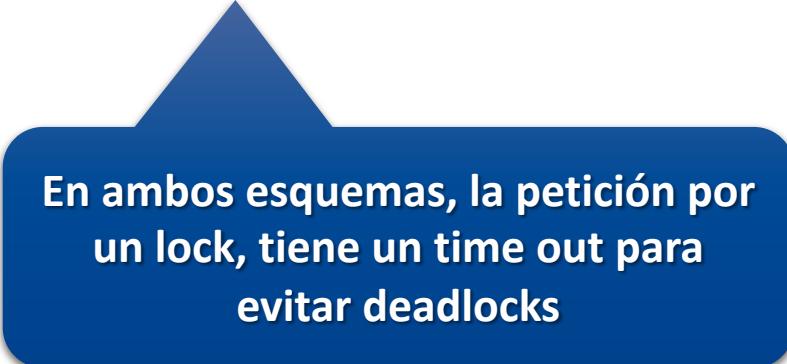
Protocolos basados en timestamps

- **wound/wait:**

Si la transacción que solicita el *lock* es la más antigua de las dos, va a “herir” (expropiar) a la transacción haciéndole rollback. La transacción más joven vuelve a ser programada para su ejecución posterior

Si la transacción que solicita el *lock* es la más joven de las dos, espera hasta que la otra transacción se complete y los *locks* se liberen

→ La transacción más vieja aborta la más joven y la re programa



En ambos esquemas, la petición por un lock, tiene un time out para evitar deadlocks

Protocolos optimistas

- El enfoque optimista se basa en la asunción de que la mayoría de operaciones en la base de datos no son conflictivas
- El enfoque optimista no requiere locks o timestamps, sino que una transacción se ejecuta sin restricciones hasta que sea comprometida
- Cada transacción se mueve por dos o tres fases: lectura, validación y escritura

Protocolos optimistas

- El enfoque optimista se basa en la asunción de que la mayoría de operaciones en la base de datos no son conflictivas
- El enfoque optimista no requiere locks o timestamps, sino que una transacción se ejecuta sin restricciones hasta que sea comprometida
- Cada transacción se mueve por dos o tres fases: lectura, validación y escritura

En la etapa de lectura, la transacción lee la base de datos, ejecuta los cálculos necesarios y hace updates a copias privadas de los valores de la base de datos. Todos los updates se registran en un archivo temporal que no es visible para las demás transacciones

Protocolos optimistas

- El enfoque optimista se basa en la asunción de que la mayoría de operaciones en la base de datos no son conflictivas
- El enfoque optimista no requiere locks o timestamps, sino que una transacción se ejecuta sin restricciones hasta que sea comprometida
- Cada transacción se mueve por dos o tres fases: lectura, validación y escritura

Durante la etapa de validación, la transacción se valida para asegurar que los cambios no van a afectar la integridad y consistencia de la base de datos. Si la validación es positiva, se pasa a la etapa de escritura, si no, la transacción se reinicia y los cambios se descartan

Protocolos optimistas

- El enfoque optimista se basa en la asunción de que la mayoría de operaciones en la base de datos no son conflictivas
- El enfoque optimista no requiere locks o timestamps, sino que una transacción se ejecuta sin restricciones hasta que sea comprometida
- Cada transacción se mueve por dos o tres fases: lectura, validación y escritura

Durante la etapa de escritura, los cambios se aplican permanentemente

Control de concurrencia en SQL

Introducción

- Una transacción en SQL es una unidad lógica de trabajo y que se garantiza ser atómica
- Una sentencia SQL por sí mismo se considera como una transacción
- Dependiendo del motor, una transacción puede o no empezar con un BEGIN TRANSACTION
- Toda transacción SQL tiene que terminar con un COMMIT o con un ROLLBACK

Características de un transacción SQL

- Una transacción SQL tiene características que puede establecerse mediante la sentencia SET TRANSACTIONS
- Las características incluyen:
 - **Modo de acceso:** determina la intención de la transacción. Puede ser READ ONLY o READ WRITE. En READ ONLY solo se permiten recuperaciones de datos, mientras que en READ WRITE se permiten todas las operaciones
 - **Tamaño del área de diagnóstico:** especifica un entero que indica el número de condiciones que se pueden mantener simultáneamente en el área de diagnósticos
 - **Nivel de aislamiento:** especifica el nivel de aislamiento que se utilizará en la transacción, hay varias modalidades que se pueden especificar según la necesidad

Características de un transacción SQL

- Una transacción SQL tiene características que puede establecerse mediante:
 - **No se pueden utilizar ambas modalidades a la vez**
- Las características incluyen:
 - **Modo de acceso**: determina la intención de la transacción. Puede ser READ ONLY o READ WRITE. En READ ONLY solo se permiten recuperaciones de datos, mientras que en READ WRITE se permiten todas las operaciones
 - **Tamaño del área de diagnóstico**: especifica un entero que indica el número de condiciones que se pueden mantener simultáneamente en el área de diagnósticos
 - **Nivel de aislamiento**: especifica el nivel de aislamiento que se utilizará en la transacción, hay varias modalidades que se pueden especificar según la necesidad

Características de un transacción SQL

- Una transacción SQL tiene características que puede establecerse mediante la sentencia SET TRANSACTION
- Las características incluyen:
 - **Modo de acceso:** determina la intención de lectura, es decir, READ ONLY o READ WRITE. En READ ONLY solo se leen los datos, mientras que en READ WRITE se pue
 - **Tamaño del área de diagnóstico:** especifica un entero que indica el número de condiciones que se pueden mantener simultáneamente en el área de diagnósticos
 - **Nivel de aislamiento:** especifica el nivel de aislamiento que se utilizará en la transacción, hay varias modalidades que se pueden especificar según la necesidad

Después de cada statement se establecía el valor de SQLSTATE... ¿Recuerda?

Características de un transacción SQL

- Una transacción SQL tiene características que puede establecerse mediante la sentencia SET TRANSACTIONS
- Las características incluyen:
 - **Modo de acceso:** determina la intención de la transacción. Puede ser READ ONLY o READ WRITE. En READ ONLY solo se permiten recuperaciones de datos, mientras que en READ WRITE se permiten todas las operaciones
 - **Tamaño** de condición diagnóstica
 - **Nivel de aislamiento:** especifica el nivel de aislamiento que se utilizará en la transacción, hay varias modalidades que se pueden especificar según la necesidad

Es responsabilidad del programador especificar el nivel de aislamiento

Características de un transacción SQL

```
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno, Salary)
    VALUES ('Robert', 'Smith', '991004321', 2, 35000);
EXEC SQL UPDATE EMPLOYEE
    SET Salary = Salary * 1.1 WHERE Dno = 2;
EXEC SQL COMMIT;
```

Niveles de aislamiento

- Distintos niveles de aislamiento se pueden especificar según las necesidades del **negocio**
- El nivel de aislamiento **se especifica y aplica** por transacción
- Normalmente **el motor tiene un nivel de aislamiento default** que se aplica a todas las transacciones.

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **READ UNCOMMITTED**: especifica que los *statements* pueden leer filas que han sido modificadas por otras transacciones pero aún no se han comprometido
 - No se adquieren *locks* compartidos ni exclusivos
 - Pueden darse lecturas sucias

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **READ UNCOMMITTED**: especifica que los *statements* pueden leer filas que han sido modificadas por otras transacciones pero aún no se han comprometido
 - No se adquieren *locks* compartidos ni exclusivos
 - Pueden darse lecturas sucias

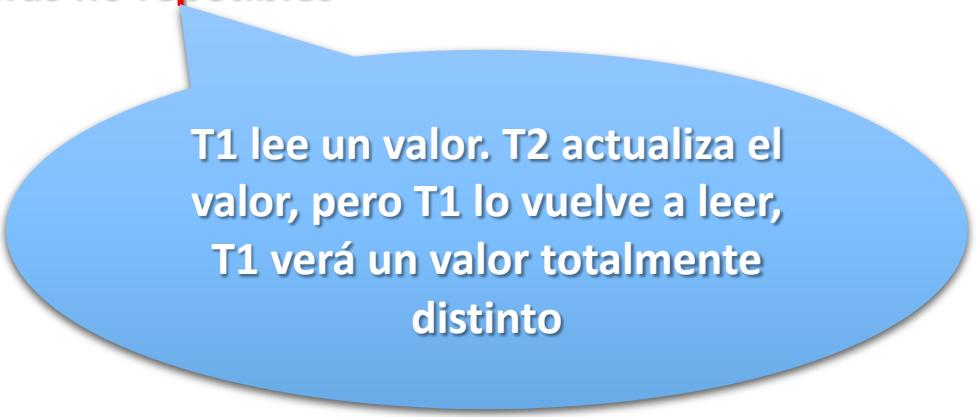
Una transacción T1 lee el valor actualizado por T2, la cual no ha sido comprometida. Si T2 se aborta, T1 leyó un valor que no existe

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **READ COMMITTED**: especifica que los *statements* no pueden leer datos que han sido modificados pero no han sido comprometidos por otras transacciones (Default de SQL SERVER)
 - Los datos pueden ser cambiados por otras transacciones, lo que resulta en lecturas no repetibles o datos fantasmas

Niveles de aislamiento

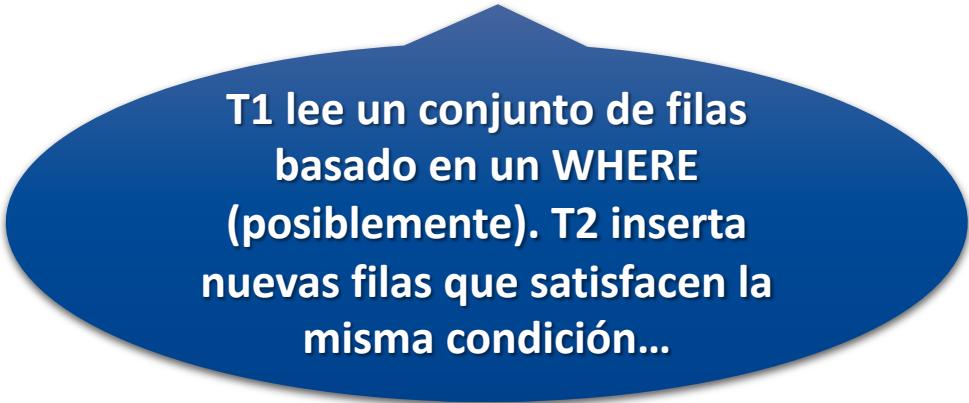
- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **READ COMMITTED**: especifica que los *statements* no pueden leer datos que han sido modificados pero no han sido comprometidos por otras transacciones (Default de SQL SERVER)
 - Los datos pueden ser cambiados por otras transacciones, lo que resulta en **lecturas no repetibles** o datos fantasmas



T1 lee un valor. T2 actualiza el valor, pero T1 lo vuelve a leer, T1 verá un valor totalmente distinto

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **READ COMMITTED**: especifica que los *statements* no pueden leer datos que han sido modificados pero no han sido comprometidos por otras transacciones (Default de SQL SERVER)
 - Los datos pueden ser cambiados por otras transacciones, lo que resulta en lecturas no repetibles o **datos fantasma**



T1 lee un conjunto de filas
basado en un WHERE
(posiblemente). T2 inserta
nuevas filas que satisfacen la
misma condición...

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **READ COMMITTED**: especifica que los *statements* no pueden leer datos que han sido modificados pero no han sido comprometidos por otras transacciones (Default de SQL SERVER)
 - Los datos pueden ser cambiados por otras transacciones, lo que resulta en lecturas no repetibles o **datos fantasma**

...si T1 vuelve a leer el conjunto de filas, encontrará filas que antes no existían

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **REPEATABLE READ**: especifica que los *statements* no pueden leer datos que han sido modificados pero aun no están comprometidos por otras transacciones y que ninguna otra transacción puede modificar el dato leído por la transacción actual hasta que esta termine.
 - Puede leer datos fantasmas
 - Peor desempeño que los otros niveles
 - Úselo solo cuando sea necesario

Niveles de aislamiento

- Los niveles de aislamiento dependen del motor, pero normalmente incluyen al menos:
 - **SERIALIZABLE**: especifica lo siguiente
 - Los *statements* no pueden tener datos que hayan sido modificados pero que aun no se hayan comprometido
 - Ninguna otra transacción puede modificar datos que hayan sido leídos por la transacción actual hasta que termine
 - Otras transacciones no pueden insertar nuevas filas con valores llave que caen en el rango de llaves leídas por cualquier *statement* en la transacción actual hasta que termine.
 - Es el nivel más restrictivo. Pone locks en rangos de filas
 - Úsalo cuando sea estrictamente necesario...

Recuperación de la base de datos

Introducción

- La recuperación de la base de datos se refiere a restaurar la base de datos de un estado (usualmente inconsistente) a un estado consistente previo
- Las técnicas de recuperación se basan en el concepto de transacciones atómicas y se enfocan principalmente en la recuperación de transacciones
- Estas técnicas también aplican para la recuperación de la base de datos o del sistema después de que haya ocurrido algún tipo de error crítico

Introducción

- Eventos críticos incluyen:
 - Fallos de hardware / software
 - Incidentes causados por humanos
 - Desastres naturales
- En el caso de algunos de estos eventos críticos, la única técnica de recuperación prácticamente es restaurar una copia de la base de datos desde una cinta de respaldo almacenada externamente

Conceptos de recuperación

- La recuperación de transacciones utiliza el log de transacciones para recuperar la base de datos de un estado inconsistente a un estado consistente
- Algunos conceptos involucrados:
 - **Protocolo write-ahead**: asegura que el log de transacciones siempre se escriba antes de que cualquier dato de la base de datos se actualice
 - **Logs de transacciones redundantes**: se tiene múltiples copias del log de transacciones para asegurar que una falla en el disco no impida realizar la recuperación de la base de datos

Conceptos de recuperación

- La recuperación de transacciones utiliza el log de transacciones para recuperar la base de datos de un estado inconsistente a un estado consistente
- Algunos conceptos involucrados:
 - **Buffers**: son áreas de almacenamiento temporal en memoria principal (RAM) que aceleran las operaciones a disco. Cuando una transacción actualiza datos, lo que actualiza en realidad es el buffer. Posteriormente los buffers se escriben a disco
 - **Checkpoints**: son operaciones en las que el DBMS escribe todos los buffers a disco. Mientras esto ocurre el DBMS no ejecuta otras peticiones. Estos checkpoints son muy importantes para la recuperación

Escritura suspendida (deferred-write)

- Es una técnica que puede emplear el procedimiento de recuperación. En esta, las transacciones no actualizan directamente los datos físicos
- En cada transacción lo que se actualiza es el log. La BD solo se actualiza físicamente si la transacción alcanza su punto de commit. Si nunca se alcanza el commit no se realiza ningún cambio

Escritura suspendida (deferred-write)

- El proceso de recuperación cuando se utilizó esta técnica es:

Identificar el último *checkpoint* en el log de transacciones

Para una transacción comprometida antes del *checkpoint*, no hay que hacer nada, puesto que ya está escrita físicamente

Si una transacción hizo *commit* después del último *checkpoint*, el DBMS utiliza el log para re hacer la transacción

Si una transacción hizo *rollback* después del último *checkpoint*, nada se necesita hacer puesto que nunca se escribió físicamente

Escritura directa (write-through)

- Mediante esta técnica la BD se actualiza inmediatamente durante la transacción incluso antes de alcanzar un *commit*.
- Si la transacción aborta antes del *commit*, se debe efectuar una operación *rollback* para retornar el estado de la base de datos a un estado consistente.

Escritura directa (write-through)

- El proceso de recuperación cuando se utilizó esta técnica es:

Identificar el último *checkpoint* en el log de transacciones

Para una transacción comprometida antes del *checkpoint*, no hay que hacer nada, puesto que ya está escrita físicamente

Si una transacción hizo *commit* después del último *checkpoint*, el DBMS utiliza el log para re hacer la transacción

Si una transacción hizo *rollback* después del último *checkpoint* o si quedó activa, el DBMS utiliza el log para hacer rollback aplicando los cambios en orden inverso

Procesamiento de transacciones, concurrencia y recuperación