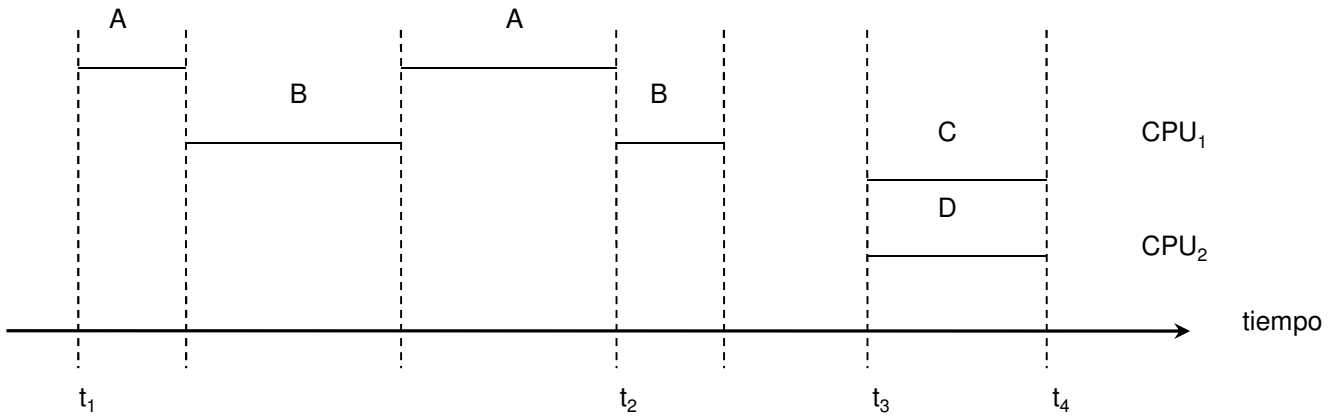
Transacciones y concurrencia

ITAM - Bases de Datos

Dr. Felipe López G.

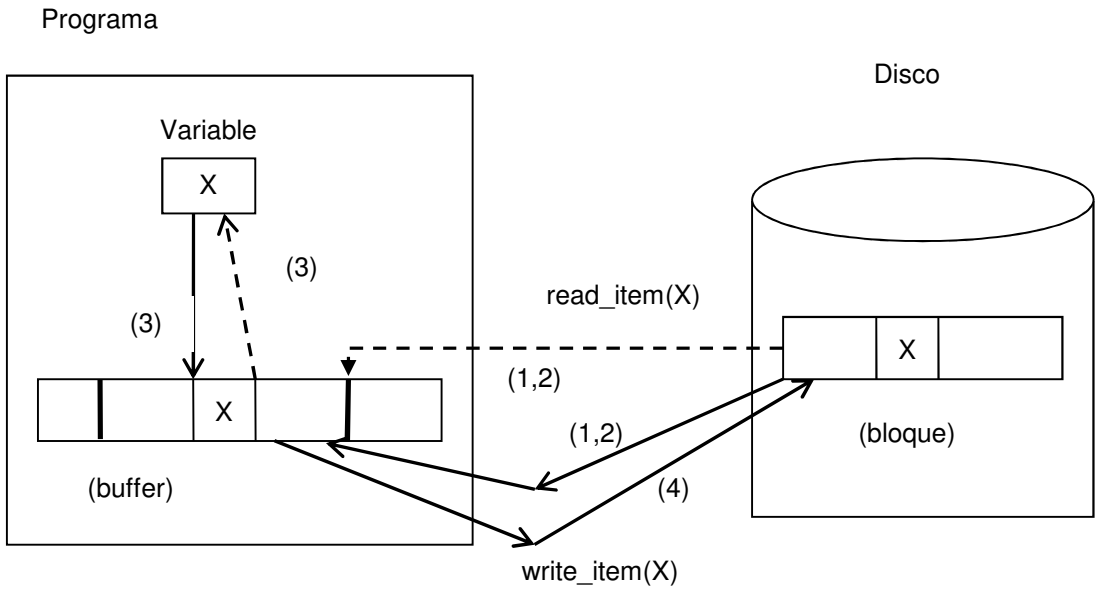
Introducción: multiprogramación

- Procesamiento *concurrente* de varios programas.
- Un solo procesador: ejecución concurrente *intercalada.*
- Un programa se puede suspender por varios motivos.
- Varios procesadores: ejecución *simultánea.*
- Recursos primarios solicitados en BD: los datos.
- En BD se le llama **transacción** a la ejecución de un programa que lee o modifica la base de datos.



Introducción: lectura/escritura de una transacción

- Operaciones de acceso: a nivel de *elementos de datos* y de *bloques de disco*. Se pueden representar como:
  - **read_item(X)**
  - **write_item(X)**



Introducción: ejemplo de transacciones

- La primera transacción podría representar una transferencia monetaria entre dos cuentas dadas por X y Y.
- La segunda, un depósito a la cuenta X.

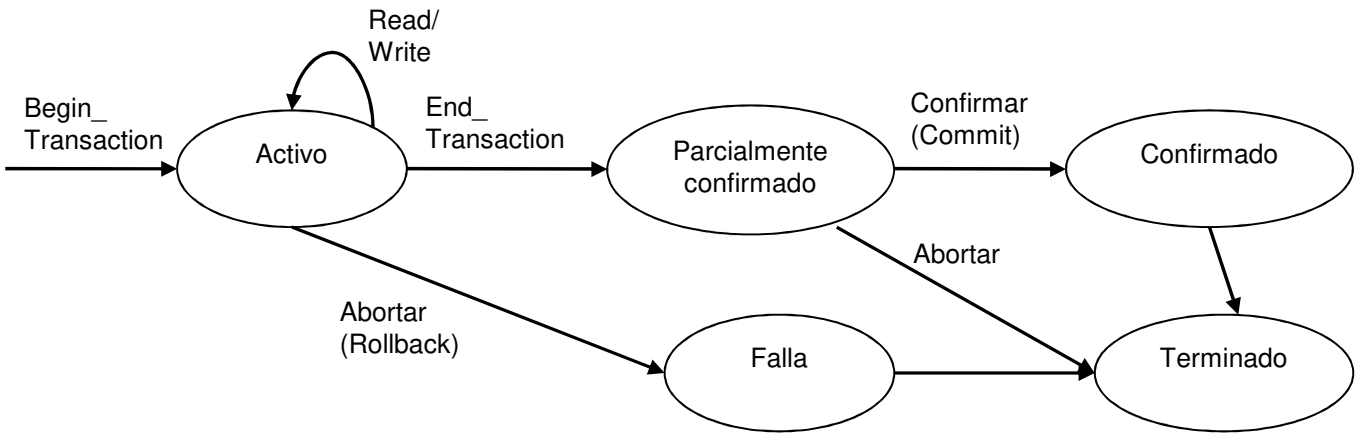
T_1: read_item(X); X= X-N;
write_item(X);
read_item(Y); Y= Y+N;
write_item(Y);

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

Nota: Los mismos programas pueden ser usados para ejecutar muchas de estas transacciones T_1 y T_2.

Conceptos de transacciones

- Una **transacción** es una unidad atómica de trabajo que se realiza por completo o bien, no se efectúa en absoluto.
- **Recuperación**: es un proceso que debe efectuar el DBMS para llevar a la BD hasta el último estado consistente que tenía antes de una falla.
- Para fines de recuperación, se necesita llevar el registro de las siguientes operaciones:



Conceptos de transacciones: propiedades deseables de una transacción

- Propiedades **ACID**
  - **Atomicity**: una transacción es una unidad atómica de procesamiento; se ejecuta totalmente o no se ejecuta en absoluto.
  - **Consistency preservation**: una ejecución correcta de la transacción debe llevar a la base de datos de un estado consistente a otro.
  - **Isolation**: una transacción no debe hacer visibles sus actualizaciones a otras transacciones hasta que sea confirmada.
  - **Durability**: una vez que una transacción cambia la base de datos y los cambios son confirmados, éstos nunca deben perderse por fallas subsecuentes.

Conceptos de transacciones: seriabilidad

- Un **plan** (o **historia**) S de n transacciones T_1, T_2, . . ., T_n es un ordenamiento de las operaciones de las transacciones sujeto a la restricción de que, para cada transacción T_i que participe en el plan S, las operaciones de T_i en S deben aparecer en el mismo orden en el cual ellas ocurren en T_i.
- Un plan S es **serial** si, para cada transacción T participante en el plan, todas las operaciones de T son ejecutadas consecutivamente en el plan; en caso contrario, el plan es llamado **no serial** .
- Ejemplo serial:

T_1	T_2
read_item(X); X=X-N;	
write_item(X); read_item(Y);	
Y= Y+N; write_item(Y);	
	read_item(X); X= X+M;
	write_item(X);

- Ejemplos no seriales: actualización perdida o lectura sucia.

Conceptos de transacciones: seriabilidad

- Se puede suponer razonablemente, si se considera que las transacciones son *independientes*, que *cada plan serial es considerado correcto*.
- Un plan S de *n* transacciones es **seriable** si es *equivalente a algún plan serial* de las mismas *n* transacciones.

  Ejemplo: plan no serial que es seriable

T_1	T_2
read_item(X); X=X-N;	
write_item(X);	
	read_item(X); X= X+M;
	write_item(X);
read_item(Y);	
Y= Y+N; write_item(Y);	

  Decir que un plan S no serial es ***seriable*** es equivalente a decir que es ***correcto***.

Transacciones en Oracle

- Estándar SQL92 (ver notas): Read uncommitted, Read committed, Repeatable read, Serializable.
- En Oracle: el segundo (default), el cuarto y Read-only.
- Consultas y actualizaciones individuales: se ejecutan como transacciones atómicas.
- Lecturas consistentes a nivel de transacción: se tienen con el modo *serializable*.
- Transacciones Read-only: no permiten actualizaciones.
- Establecimiento del nivel de aislamiento:
  - Para transacciones individuales:

    Set transaction isolation level {read committed | serializable | read only}
  - Para un conjunto de transacciones:

    Alter session set isolation_level {read committed | serializable | read only}

Control de concurrencia: granularidad de los datos

- Elementos de datos: una tupla, un conjunto de tuplas, un bloque de disco, una tabla, un conjunto de tablas o toda la base de datos.
- Mayor tamaño del elemento → menos concurrencia
- Menor tamaño del elemento → más concurrencia, pero también más candados.

Control de concurrencia: técnicas de bloqueo

- Un **candado** o **bloqueo (lock)** es una variable asociada con un elemento de datos en la base de datos y describe el estatus de ese elemento con respecto a posibles operaciones que pueden ser aplicadas al mismo.
- En general, puede existir un candado por cada elemento de datos en la base de datos.
- Tipos de candados:
  - Binarios
  - Compartidos y exclusivos

Control de concurrencia: candados binarios

- Pueden tener dos estados o valores: bloqueado (1) y desbloqueado (0).
- Un candado distinto puede ser asociado con cada elemento X de la base de datos:

  Lock(X)- da el valor del candado asociado a X.
- Operaciones representativas (se implementan como unidades indivisibles, o sea, no pueden interrumpirse):
  - lock_item(X)
  - unlock_item(X)
- El candado binario fuerza la **exclusión mutua** sobre los elementos de datos.
- Hay restricciones en el uso de estas operaciones dentro de una transacción (**ver notas: pág. 8**).

Control de concurrencia: candados compartidos y exclusivos

- Pueden tener tres tipos de valores: “read-locked”, “write-locked” o “unlocked”.
- Operaciones representativas (se implementan como unidades indivisibles):
  - read_lock(X)
  - write_lock(X)
  - unlock(X)
- Un elemento *read-locked* permite que otras transacciones lo lean.
- Un elemento *write-locked* es de uso exclusivo por una transacción.
- Estos candados no garantizan la seriabilidad.

Protocolo de bloqueo de dos fases

- *Todas* las operaciones de bloqueo (read_lock, write_lock) preceden a la *primera* operación de desbloqueo en la transacción.
- Existen dos fases:
  - **Expansión**: se adquieren candados y ninguno se libera.
  - **Contracción**: se liberan candados y ninguno se adquiere.
- Ejemplo: en las notas (**pág. 9**).
- **Se puede probar** que: si *cada* transacción en un plan sigue este protocolo, se garantiza que el plan es **seriable** (y por lo tanto, correcto).
- El empleo de este protocolo no evita los **bloqueos mortales (deadlocks)**. Ejemplo: en las notas (**pág. 10**).

Bloqueos en Oracle

- Oracle **automáticamente obtiene los bloqueos necesarios** cuando ejecuta las instrucciones de SQL. usando el nivel más bajo de bloqueo (Read committed).
- Modos de bloqueo:
  - Bloqueo compartido (share lock).
  - Bloqueo exclusivo (exclusive lock).
- Todos los candados adquiridos dentro de una transacción son mantenidos hasta su terminación (sea con *commit* o con *rollback*); esto es, no hay una operación equivalente a unlock(X).
- Oracle también permite que el usuario bloquee manualmente los datos (ver notas después de candados internos).

Categorías de candados

- **Candados DML**. Permiten proteger la integridad de los datos; son de dos tipos:
  - **Row lock**: se aplica al nivel de filas de una tabla.
  - **Table lock**: permite reservar una tabla dentro de una transacción.
- El primer tipo se adquiere **en modo exclusivo** para cada fila modificada por un *insert, delete, update* o *select… for update*.
- El segundo tipo permite bloquear una tabla en varios modos (ver notas).
- **Candados DDL**. Protegen la estructura de la BD.
- **Candados internos y latches**. Protegen estructuras internas (catálogo) de la BD. Sólo los usa el DBMS.