

SERVICIOS DE BACKEND (TRANSACCIONES)

Sistemas informáticos I

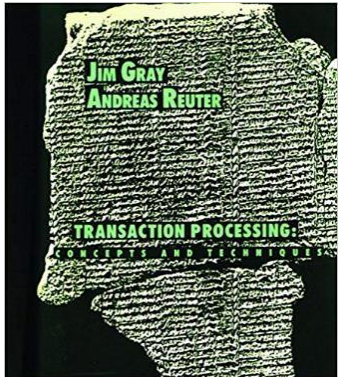
ÍNDICE

- Introducción al proceso de transacciones
- Definiciones y propiedades de las transacciones
- Modelos de transacciones
- Aislamiento.
- Control de la concurrencia



REFERENCIAS BIBLIOGRÁFICAS

GRAY, J. y REUTER, A., Transaction Processing Concepts and Techniques.



TRANSACCIONES

- Concepto orientado a proveer tolerancia a fallos y permitir la concurrencia en sistemas distribuidos (en particular C/S)
- Una transacción es una colección de operaciones de lectura y escritura de datos que están relacionadas a nivel lógico (p.ej.: transferencia bancaria) que:
 - deben ocurrir en su totalidad o no ocurrir en absoluto (**atomicidad**)
 - si la transacción se ejecuta, los efectos de las operaciones de escritura deben persistir; y si no se completa, la transacción no debe producir ningún efecto
 - debe implementarse de forma que estos efectos se garanticen incluso si se produce un fallo del sistema (p.ej.. rotura del disco duro, corte en la red, etc.)




QUÉ PUEDE IR MAL EN CONCURRENCIA

- Supongamos que tenemos una tabla de cuentas bancarias que contiene el saldo y el número de cuenta. Un depósito en la cuenta No. 1234 puede escribirse como:

UPDATE Cuenta SET saldo = saldo + 50 WHERE No_cuenta= '1234';
- Esta sentencia conlleva la lectura y escritura secuencial del atributo saldo:

read(X.saldo)
X.saldo := X.saldo + 50
write(X.saldo)
- ¿Qué ocurre si dos titulares de la cuenta intentan hacer sendos depósitos concurrentemente?



DEPÓSITOS CONCURRENTES

- En la arquitectura más sencilla, sólo una "acción" (esto es, una lectura o escritura) puede suceder en un instante dado
- Hay varias formas en las cuales una transacción puede ser ejecutada simultáneamente:

Depósito 1
read(X.saldo)

X.saldo := X.saldo + 50


write(X.saldo)

Depósito 2
read(X.saldo)

X.saldo:= X.saldo + 10

write(X.saldo)

tiempo
↓



DEPÓSITOS CONCURRENTES

- El resultado hubiera sido correcto si las transacciones se hubieran realizado sobre cuentas diferentes

Depósito 1

```
read(X.saldo)

X.saldo := X.saldo + 50

write(X.saldo)
```


Depósito 3

```
read(Y.saldo)

Y.saldo:= Y.saldo + 10

write(Y.saldo)
```

tiempo



DEPÓSITOS CONCURRENTES

- Una solución para nuestro caso es ejecutar las transacciones en serie, esto es, primero una y luego otra:


Depósito 1


```
read(X.saldo)
X.saldo := X.saldo + 50
write(X.saldo)
```

Depósito 2

```
read(X.saldo)
X.saldo:= X.saldo + 10
write(X.saldo)
```


tiempo






OBJETIVO

- Uno de los objetivos de una gestión correcta de transacciones es ejecutar las transacciones "de forma equivalente" a ejecutarlas en serie, pero no necesariamente en serie (de lo contrario no aprovechamos los recursos computacionales del sistema)
- Una ejecución es correcta si es **en serie** (las transacciones se ejecutan secuencialmente una detrás de otra) o **seriabilizable** (esto es, equivalente a una ejecución en serie)



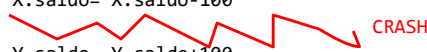


QUÉ PUEDE IR MAL, ATOMICIDAD


- Otro problema asociado a las transacciones es garantizar la atomicidad
 - qué ocurre si se produce un fallo antes de ejecutar completamente la transacción:

Transferencia

```
read(X.saldo)
read(Y.saldo)
X.saldo= X.saldo-100
Y.saldo= Y.saldo+100
```



- Necesitamos garantizar que la escritura a X desaparece de la base de datos
- Los problemas asociados a la atomicidad son especialmente significativos si los datos están distribuidos en distintos nodos





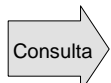

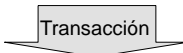


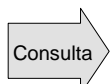
PROCESOS TRANSACCIONALES


- El cliente solicita la ejecución de un procedimiento remoto especial en el servidor: una **transacción**
- El servidor garantiza la atomicidad de la transacción → el sistema nunca queda en un estado inconsistente aunque haya un fallo
- El servidor garantiza la correcta ejecución concurrente de las transacciones → no hay problemas producidos por accesos múltiples a los datos
- El proceso de transacciones ha sido la herramienta más eficaz en el desarrollo de aplicaciones para entornos distribuidos desde sus comienzos



The diagram illustrates the transaction process flow. On the left, a computer icon represents the 'Aplicación' (Application). On the right, a server rack icon represents the 'Servidor DBMS' and 'Servidor OLTP'. A yellow arrow labeled 'transacción' points from the application to the server. A yellow arrow labeled 'resultado' points from the server back to the application. A small logo with the letters 'EPS' is in the bottom left corner.

DEFINICIONES Y PROPIEDADES DE LAS TRANSACCIONES

	Realidad	Abstracción		
Recurso				Resultado
				
Recurso				Resultado



INVARIANTES DE UN SISTEMA

- Relaciones que se deben cumplir entre los componentes de un sistema informático. Ejemplos:
 - El campo de suma de verificación de una página P debe valer MD5(P)
 - Para cada elemento en una lista doblemente enlazada se debe verificar que $\text{prev}(\text{next}(x))=x$
 - La tabla EMP1 es una réplica de la tabla EMP
- También hay invariantes definidos sobre el modo en que el sistema cambia de estado. Ejemplos:
 - Todo cambio en una página debe actualizar su suma de verificación
 - Al insertar un elemento en una lista se deben actualizar los punteros de sus elementos anterior y posterior
 - Cuando se inserta un registro en la tabla EMP se debe insertar también en la tabla EMP1



ESTADOS CONSISTENTES

- El sistema se encuentra en un **estado consistente** si satisface todos sus invariantes
- Cualquier cambio puede hacer pasar al sistema por un estado transitorio inconsistente
 - Cambio página: $P \Rightarrow P'$ y suma de verificación continúa siendo MD5(P)
 - Lista doblemente encadenada: $w \leftrightarrow y$; Inserción x,
 - Paso 1: $w \leftrightarrow x, x \rightarrow y, w \leftarrow y$; Paso 2: $w \leftrightarrow x \leftrightarrow y$
 - Insert into EMP (Reg1) $\rightarrow \text{EMP} \neq \text{EMP1}$
Insert into EMP1 (Reg1) $\rightarrow \text{EMP} = \text{EMP1}$
- Las transacciones pasan al sistema de un estado consistente a otro estado consistente pudiendo pasar por un estado inconsistente
- Si la transacción falla, el sistema debe quedar en un estado consistente



QUÉ SON LAS TRANSACCIONES

- Programas que realizan cambios de estado en recursos (no necesariamente uno)
- Representan una unidad básica de trabajo
- Los cambios se realizan mediante acciones
 - Inserción y borrado de información
 - Consultas y actualizaciones de información
 - Cualquier otro proceso, normalmente con una coherencia interna dentro de la lógica de la aplicación
- La transacción es indivisible: se ejecuta completa o no se ejecuta
- Modeliza el concepto real de contrato para la realización de transacciones reales:
 - Si todo proceso siempre fuera bien, supone un trabajo innecesario
 - Si algo va mal, necesario para saber cómo arreglar la situación



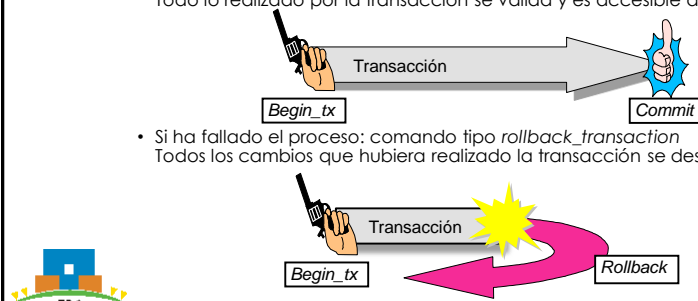
PROPIEDADES DE LAS TRANSACCIONES

- **ACID:** *Atomicity, Consistency, Isolation and Durability*
- **Atomicidad:**
 - La transacción es una unidad indivisible de trabajo
 - Definida con respecto al usuario de la transacción (aplicación que la activa)
- **Consistencia:**
 - Al finalizar su ejecución, el sistema debe quedar en estado estable consistente
 - Si no puede realizarla, debe devolver el sistema a su estado inicial (*rollback*)
- **Aislamiento:**
 - Una transacción no se ve afectada por otras que se ejecuten concurrentemente aunque utilicen los mismos recursos
 - Los cambios que introduzca en recursos compartidos no deben ser visibles hasta que la transacción finalice
 - Necesario que la transacción "bloquee" los recursos que tiene que actualizar
- **Durabilidad:**
 - Sus efectos son permanentes una vez que ha finalizado correctamente (*commit*)



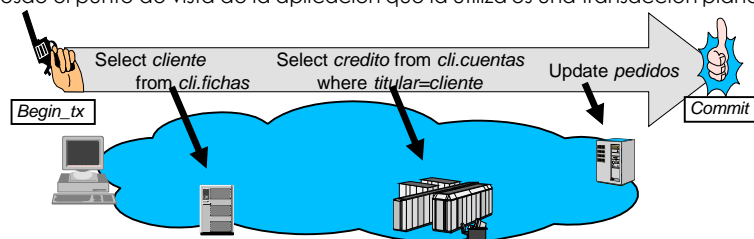
MODELOS DE TRANSACCIONES

- Transacciones Planas (*flat transactions*)
 - Todo el proceso realizado en su interior se encuentra al mismo nivel de jerarquía (no hay llamadas a otras transacciones)
 - Comienzan con un comando tipo *begin_transaction*
 - Finalizan:
 - Si el proceso fue correcto: comando tipo *commit_transaction*
Todo lo realizado por la transacción se valida y es accesible al exterior.
 - Si ha fallado el proceso: comando tipo *rollback_transaction*
Todos los cambios que hubiera realizado la transacción se deshacen.



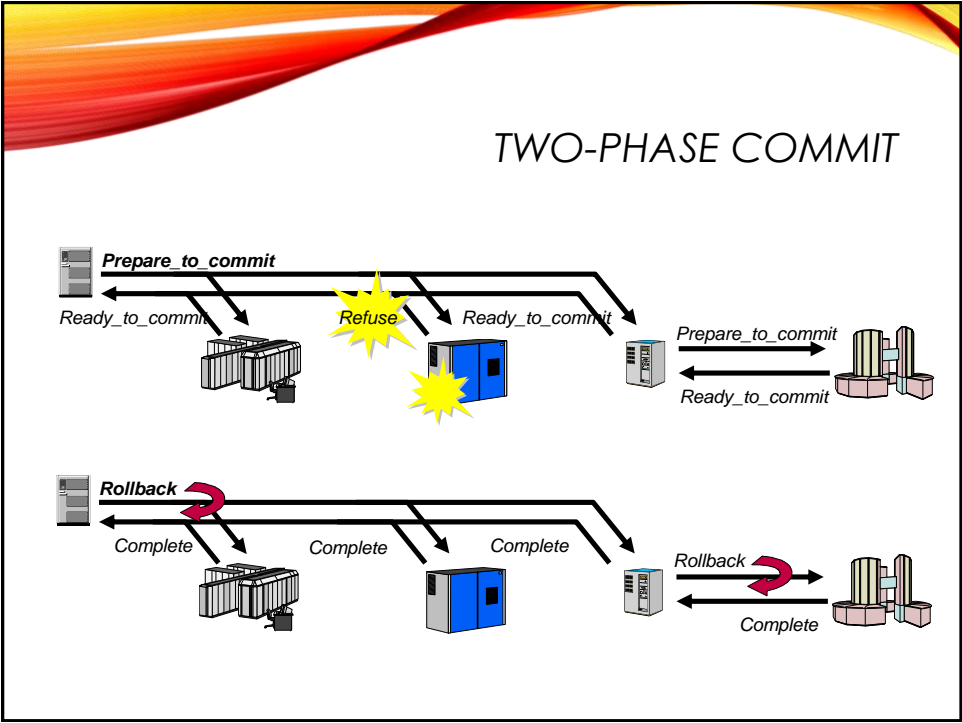
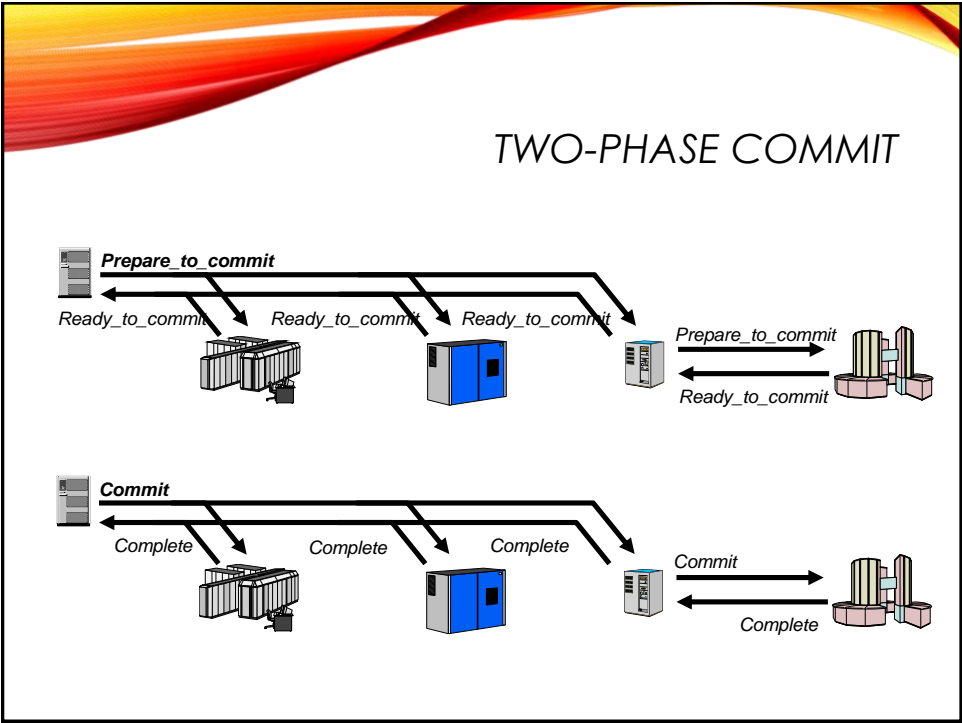
TRANSACCIONES PLANAS DISTRIBUIDAS

- Trabajan con recursos ubicados en diferentes sistemas
- Desde el punto de vista de la aplicación que la utiliza es una transacción plana



- El monitor de proceso transaccional que la controla debe garantizar que la transacción continúa siendo ACID:
 - Necesario garantizar la destrucción de las operaciones en todos los sistemas afectados en caso de que algo falle
 - Requiere nuevo proceso de validación de la transacción: *Two-Phase Commit*





TRANSACCIONES ANIDADADAS

- Llamada a una transacción desde otra (*modelo subrutinas*)

```
graph LR; T1[Transacción] -- Call --> T2[Transacción]; T1 --> C1[Commit]; T2 --> C2[Commit]; T1 --> B1[Begin_tx]; T2 --> B2[Begin_tx];
```

- Un *commit* de una transacción hace sus cambios visibles a todas las transacciones precedentes en la jerarquía de llamadas
- Un *rollback* de una transacción realiza un *rollback* de todas las subtransacciones que haya ejecutado, aunque estas ya hayan realizado un *commit*

TRANSACCIONES CON SAVEPOINTS

- Para evitar el efecto “todo-o-nada”, también se ha propuesto utilizar *savepoints*
 - Registran estados consistentes dentro de la transacción
- Cuando sea necesario un *rollback*, en principio no es necesario deshacer toda la transacción
- Los efectos de la transacción sólo serán permanentes y visibles “desde fuera” con el *commit* de la transacción


AISLAMIENTO. CONTROL DE LA CONCURRENCIA

- Aislamiento (*isolation*): Propiedad de los sistemas de proceso transaccional por la cual una transacción no se ve afectada por otras que se ejecuten concurrentemente aunque utilicen los mismos recursos
- Leyes de la concurrencia:
 - Primera ley: La ejecución concurrente no debe causar que los programas de aplicación funcionen incorrectamente.
Consecuencias:
 - Todo programa debe ver un estado consistente de los recursos al comienzo de su ejecución.
 - Toda modificación del estado consistente de los recursos que vea un programa debe ser motivada por él mismo.
 - Segunda ley: La ejecución concurrente no debe tener menor rendimiento o tiempos de respuesta mucho mayores que la ejecución en serie.
Consecuencia:
 - Los algoritmos de control de la concurrencia deben ser sencillos y eficientes.



DEFINICIONES SOBRE EL AISLAMIENTO

- Historia de un conjunto de transacciones: Secuencia que preserva la mezcla de acciones que tiene lugar al ejecutarse un conjunto de transacciones.
$$H = \langle \langle t, a, o \rangle_i \mid i = 1, \dots, n \rangle$$
- Versiones de objetos:
 - Cada variación de un objeto del sistema se puede ver como la creación de una nueva versión del objeto: <o, 1>, <o, 2>, <o, 3>...
 - Las versiones de los objetos se crean por acciones de escritura en las transacciones.
 - La versión de un objeto "o" en un paso k de una historia se denota por V(o,k), y representa el número de escrituras realizadas sobre el objeto en la historia H hasta llegar al paso k.
$$V(o, k) = \left| \left\{ \langle t, a, o \rangle_j \in H \mid (j < k) \wedge (a_j = WRITE) \wedge (o_j = o) \right\} \right|$$



EJEMPLO

- Acciones:
T1 READ A
T2 WRITE C
T3 WRITE C
T2 READ B
- ¿Cuál sería la historia?
H = < <T1, READ, A>, <T2, WRITE, C>, <T3, WRITE, C>, <T2, READ, B> >
- ¿Cuál sería la evolución de las versiones de los objetos?
V(A,1)=0 V(A,2)=0 V(A,3)=0 V(A,4)=0
V(B,1)=0 V(B,2)=0 V(B,3)=0 V(B,4)=0
V(C,1)=0 V(C,2)=0 V(C,3)=1 V(C,4)=2



FORMAS DE DEPENDENCIAS ENTRE TRANSACCIONES

- Lectura / escritura:

T1 READ <0,1>
T2 WRITE <0,2>
- Escritura / escritura:

T1 WRITE <0,2>
T2 WRITE <0,3>
- Escritura / lectura:

T1 WRITE <0,2>
T2 READ <0,2>




TIPOS DE VIOLACIONES DEL AISLAMIENTO

- Actualización perdida (*lost update*)
→ La escritura de una transacción es ignorada por otra, que realiza una nueva escritura basada en la versión previa del objeto.
T1 READ <o,1>
T2 READ <o,1>
T1 WRITE <o,2>
T2 WRITE <o,3>
- Lectura sucia (*dirty read*) → Una transacción lee un objeto escrito por otra en un estado intermedio.
T2 READ <o,1>
T2 WRITE <o,2>
T1 READ <o,2>
T2 WRITE <o,3>

- Lectura no repetible (*unrepeatable read*) → Una transacción lee un objeto dos veces con valores distintos, por haber una actualización intermedia realizada por otra transacción.
T1 READ <o,1>
T2 WRITE <o,2>
T1 READ <o,2>

- Estas tres son las únicas formas de inconsistencia. Si se pueden prevenir las tres, se resuelve el problema del aislamiento.



BLOQUEO (LOCK)

- Acción de una transacción por la cual se señala el uso de un objeto.
- Dos tipos de bloqueos:
 - Bloqueo compartido (*Shared Lock*): No se requiere el uso exclusivo del objeto.
 - Bloqueo exclusivo (*Exclusive Lock*): Se requiere el uso exclusivo del objeto.
- Compatibilidad de bloqueos de un mismo objeto entre transacciones:

Compatibilidad		Modo de bloqueo existente	
		Compartido	Exclusivo
Modo bloqueo solicitado	Compartido	Compatible	Prohibido
	Exclusivo	Prohibido	Prohibido



ACCIONES DENTRO DE UNA TRANSACCIÓN

- Acciones sobre objetos:
 - **READ**: Lectura de un objeto.
 - **WRITE**: Escritura de un objeto.
 - **XLOCK** (*eXclusive LOCK*): Solicitud uso exclusivo de un objeto.
 - **SLOCK** (*Shared LOCK*): Solicitud de uso compartido de un objeto.
 - **UNLOCK**: Liberación de una solicitud de uso.
- Acciones genéricas:
 - **BEGIN**
 - **COMMIT**: Equivale a la liberación de todos los bloqueos realizados por la transacción.
 - **ROLLBACK**: Equivale a deshacer todas las escrituras realizadas por la transacción y liberar todos sus bloqueos.
- Modelo equivalente simple de una transacción: Contiene la descomposición de las acciones genéricas en sus acciones simples equivalentes.



EJEMPLOS

- Transacción:


T1	BEGIN	
	SLOCK	A
	XLOCK	B
	READ	A
	WRITE	B
	COMMIT	

- Modelo equivalente simple:

T1	SLOCK	A
	XLOCK	B
	READ	A
	WRITE	B
	UNLOCK	A
	UNLOCK	B

T2	BEGIN	
	SLOCK	A
	READ	A
	XLOC	B
	WRITE	B
	ROLLBACK	

T2	SLOCK	A
	READ	A
	XLOCK	B
	WRITE	B
	WRITE (UNDO)	B
	UNLOCK	A
	UNLOCK	B



TRANSACCIONES BIEN FORMADAS Y TRANSACCIONES EN DOS FASES

- Transacción bien formada: Todas sus lecturas y escrituras están cubiertas por bloqueos (precedidas por un *lock* del tipo adecuado y no realizado un *unlock*).
 - READ: cubierta (al menos) por SLOCK.
 - WRITE: cubierta por XLOCK.
- Transacción de dos fases: Todos los bloqueos preceden a todos los desbloques.
 - Fase de crecimiento: adquiere los bloqueos.
 - Fase de contracción: libera los bloqueos.



AISLAMIENTO COMPLETO

- Para garantizar el aislamiento de cualquiera de las combinaciones posibles de un conjunto de transacciones se deben programar atendiendo a los siguientes criterios:
 - Escribir siempre transacciones bien formadas. Proteger todas las acciones con bloqueos.
 - Establecer bloqueos exclusivos en los objetos que se vayan a actualizar.
 - Escribir transacciones en dos fases: No comenzar a liberar los bloqueos hasta que no se necesite realizar más (bloqueos).
 - Mantener los bloqueos exclusivos hasta que se realice el *Commit* o el *Rollback*.
- El aislamiento completo no evita las lecturas fantasma asociadas a inserciones y borrado de objetos que deberían estar bloqueados



GRADOS DE AISLAMIENTO

- El aislamiento completo es costoso.
- Muchos fabricantes permiten activar selectivamente el **grado de aislamiento** deseado para el sistema:
 - 0° (grado 0) → caos. Las transacciones no sobrescriben datos sucios de transacciones de nivel 1° o superior.
 - 1° (grado 1) → lecturas no comprometidas. No hay pérdida de actualizaciones en el ámbito de la transacción (hasta el COMMIT).
 - 2° (grado 2) → lecturas comprometidas. No hay pérdida de actualizaciones en el ámbito de la transacción ni lectura de datos sucios.
 - 3° (grado 3) → lectura repetible. No hay pérdida de actualizaciones ni lectura de datos sucios y hay lecturas repetitivas. Es el aislamiento completo garantizado.
- Menor grado de aislamiento facilita la concurrencia y evita esperas en las transacciones, pero puede producir violaciones de aislamiento.



GRADO 1

```
T1 READ <o,1>
T2 READ <o,1>
T1 WRITE <o,2>
T2 WRITE <o,3>
```

- Transacciones bien formadas respecto a escrituras y XLOCKS en dos fases:

```
T1 READ <o,1>
T2 READ <o,1>
T1 XLOCK o
T1 WRITE <o,2>
T2 XLOCK o
T2 WRITE<o,3> → no hay actualización perdida
```




GRADO 2

T2 READ <o,1>
T2 WRITE <o,2>
T1 READ <o,2>
T2 WRITE <o,3>

- Bien formada respecto a lectura/escrituras y XLOCKS en dos fases:

T2 SLOCK o
T2 READ <o,1>
T2 UNLOCK o
T2 XLOCK o
T2 WRITE <o,2>
T1 SLOCK o
T1 READ <o,2> → no hay lectura de datos sucios




GRADO 3

T1 READ <o,1>
T2 WRITE<o,2>
T1 READ <o,2>

- Bien formada respecto a lectura/escrituras y bloqueos en dos fases:

T1 SLOCK o
T1 READ <o,1>
T2 XLOCK o
T2 WRITE <o,2>
T1 READ <o,2> → no hay lectura no repetible



COMPARACIÓN DE LOS GRADOS DE AISLAMIENTO

Aspecto	Grado 0	Grado 1	Grado 2	Grado 3
Nombre	Caos	Read Uncommitted Consulta (Browse)	Read Committed	Repeatable Read
Dependencias consideradas	Ninguna	WRITE → WRITE	1º y WRITE → READ	2º y READ → WRITE
Protección proporcionada	Permite los niveles superiores	0º y no se pierden actualizaciones	1º y no hay lecturas sucias	2º y hay lecturas repetibles
Datos validados (committed)	Escrituras visibles inmediatamente	Escrituras visibles al fin de la transacción	Igual que 1º	Igual que 1º
Datos sucios	No se sobrescriben datos sucios ajenos	Nadie sobrescribe datos sucios	1º y no se leen datos sucios	2º y no se ensucian datos ya leídos
Protocolo de bloqueo	Bloqueos exclusivos cortos en escrituras	Bloqueos exclusivos largos en escrituras	1º y bloqueos cortos compartidos lectura	1º y bloqueos largos compartidos lectura
Estructura de la transacción	Bien formadas (Wrt)	Bien formadas (Wrt) Dos fases (Wrt)	Bien formadas. Dos fases (Wrt)	Bien formadas Dos fases.
Concurrencia	Muy alta. Casi no hay esperas	Alta: Sólo espera bloqueos escritura	Media: 1º y bloqueo en algunas lecturas	Baja: bloqueo se mantiene hasta EOT
Sobrecarga (overhead)	Mínima.	Pequeña	Media	Media
Rollback	No existe	Funciona	Igual que 1º	Igual que 1º
Recuperación del sistema	Peligroso. Pérdidas y violaciones 3º	Aplicar log en orden 1º	Igual que 1º	Aplicar log en orden <<<



GRADOS DE AISLAMIENTO EN SQL

- SQL y SQL2 permiten establecer el grado de aislamiento en el que trabajan las transacciones mediante la instrucción (dependiente del SGBD)

```
SET TRANSACTION ISOLATION LEVEL
[READ UNCOMMITTED | READ COMMITTED |
 REPEATABLE READ | SERIALIZABLE]
```

- El grado de aislamiento también se pueden especificar a nivel de transacción específica.



PROBLEMAS CON GRADOS DE AISLAMIENTO INFERIORES AL 3

- En estos casos, se pueden producir actualizaciones perdidas tras realizar una transacción el COMMIT. Ejemplo con grado 2:

```
BEGIN
SLOCK A
READ A
UNLOCK A
XLOCK A
WRITE A
COMMIT
```

```
exec sql SELECT balance into :balance FROM account
WHERE account_id = :id;

balance = balance + 10;
exec sql UPDATE account SET balance = :balance
WHERE account_id = :id;
```
- La siguiente historia produce una actualización perdida:

T1 BEGIN		T2 UNLOCK A
T1 SLOCK A		T2 XLOCK A
T1 READ A	// Ejp: A=10	T2 WRITE A
T1 UNLOCK A		T2 COMMIT
T2 BEGIN		T1 XLOCK A
T2 SLOCK A		T1 WRITE A
T2 READ A	// Ejp: A=10	T1 COMMIT
- Para garantizar el aislamiento, es preciso mantener bloqueos largos en lectura.



GRADO DE AISLAMIENTO ESTABILIDAD DE CURSORES

- El grado de aislamiento “estabilidad de cursores” (*cursor stability*, no disponible en todos los SGBD) mejora el grado 2 para resolver el problema anterior
- Se utiliza en sentencias con cursores
- Mantiene el bloqueo compartido sobre el registro al que accede el cursor hasta que se pase al siguiente registro
 - Si se realiza una actualización, el bloqueo se convierte en exclusivo, y, por tanto, se mantiene
 - Si se lee el siguiente registro, el bloqueo se libera

```
exec sql DECLARE c CURSOR FOR
SELECT balance FROM account WHERE account_id = :id;
exec sql open c;
exec sql fetch c into :balance;
balance = balance + 10;
exec sql UPDATE account SET balance = :balance
WHERE account_id = :id;
exec sql close c;
```

```
BEGIN
SLOCK A
READ A
XLOCK A
WRITE A
COMMIT
```



BLOQUEOS FOR UPDATE

```
exec sql DECLARE c CURSOR FOR
SELECT * FROMSELECT balance FROM account WHERE account_id = :id
FOR UPDATE;
```

- Bloqueo para cursores declarados para realizar actualizaciones sobre los registros encontrados:
 - Si se realiza una actualización, el bloqueo se convierte en exclusivo, y, por tanto, se mantiene hasta el final de la transacción.
 - Si no se actualiza y se pasa a leer el siguiente registro...
 - ... el bloqueo se mantiene hasta el final de la transacción si ésta es de grado 3.
 - ... el bloqueo se libera si la transacción es de grado 2.

Compatibilidad		Modo de bloqueo existente		
		Compartido	Update	Exclusivo
Modo bloqueo solicitado	Compartido	Compatible	Prohibido	Prohibido
	Update	Compatible	Prohibido	Prohibido
	Exclusivo	Prohibido	Prohibido	Prohibido



INTERBLOQUEO (DEADLOCK)

- Situación de bloqueo recíproco de recursos, que producen espera ilimitada:

T1 XLOCK A
T1 XLOCK B
...

Deadlock

T2 XLOCK B
T2 XLOCK A
...
- Solución más simple: nunca parar. En caso de que se deniegue un bloqueo, ejecutar rollback. Intentar de nuevo la transacción.
 - Puede producir situaciones de livelock.
- Detección por timeout: cancelar la transacción tras un tiempo de espera.
 - Última solución para liberar algunos interbloqueos.
 - La espera en un bloqueo es una situación muy rara. Un interbloqueo es aún mucho más raro.
⇒ Los mecanismos de detección deben ser sencillos, baratos y no deben sobrecargar la ejecución normal.



DETECCIÓN DE INTERBLOQUEOS POR GRAFO DE ESPERAS

- **Grafo de esperas:** Transacciones del sistema en los nodos. Existe un arco entre las transacciones T y T' si:
 - T está esperando a un recurso bloqueado por T'.
 - T y T' están ambas en espera de un recurso, T está detrás de T' en la lista de espera y sus peticiones son incompatibles.
- Si existe un ciclo en el grafo de esperas, es un interbloqueo.

T1 XLOCK A
T2 XLOCK B
T3 XLOCK C
T4 XLOCK D
T2 XLOCK C
T3 XLOCK D
T4 XLOCK B
T1 XLOCK D

