

Apuntes-SI1-Tema-4.pdf

**SalvaGrados**

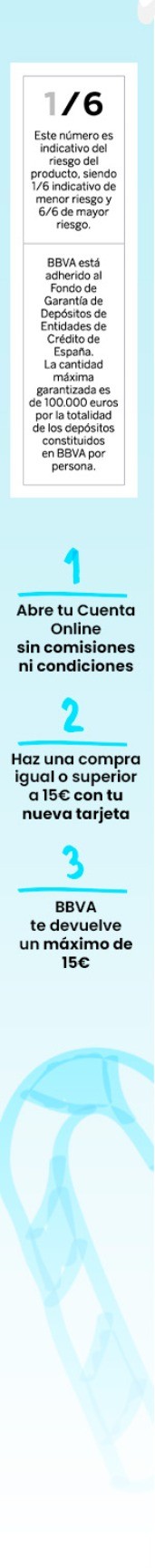
**Sistemas Informaticos I**

**3º Grado en Ingeniería Informática**

**Escuela Politécnica Superior Universidad Autónoma de Madrid**

Reservados todos los derechos.

No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



**SERVICIOS DE BACKEND (TRANSACCIONES)**

**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

Tabla

Descripción generada automáticamente

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:

Tabla

Descripción generada automáticamente con confianza baja

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

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

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

Tabla

Descripción generada automáticamente

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)

• Se pueden obtener ejecuciones serializables a través de mecanismos de sincronización adecuados, como los bloqueos (locks)

TRANSACCIÓN EN SERIE

Tabla

Descripción generada automáticamente

TRANSACCIÓN SERIALIZABLE

Tabla

Descripción generada automáticamente

TRANSACCIÓN SERIALIZABLE

Tabla

Descripción generada automáticamente

QUÉ PUEDE IR MAL, ATOMICIDAD

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**PROCESOS TRANSACCIONALES**

Pasos para una transacción:

1. El cliente solicita la ejecución de un procedimiento remoto especial en el servidor: una transacción.
2. El servidor garantiza la atomicidad de la transacción el sistema nunca queda en un estado inconsistente aunque haya un fallo.
3. El servidor garantiza la correcta ejecución concurrente de las transacciones no hay problemas producidos por accesos múltiples a los datos.

No

# INVARIANTES DE UN SISTEMA

Relaciones que se deben cumplir entre los componentes de un sistema informático. Ejemplo:

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 prev(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.

Ejemplo:

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.Ç

**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 ⇒ P' y suma de verificación continúa siendo MD5(P)

• Lista doblemente encadenada: w ↔ y; Inserción x,

• Paso 1: w ↔ x, x→y, w ← y; Paso 2: w ↔ x ↔ y

• Insert into EMP (Reg1) ◊ EMP ≠ EMP1

Insert into EMP1 (Reg1) ◊ EMP = 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**

• Conjuntos de acciones 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

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente





# PROPIEDADES DE LAS TRANSACCIONES: ACID

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)

Reserv

* 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)

No

# 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.

* Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

  Descripción generada automáticamenteSi ha fallado el proceso: comando tipo rollback\_transaction. Todos los cambios que hubiera realizado la transacción se deshacen.

# Transacciones Planas Distribuidas:

Desde el punto de vista de la aplicación que la utiliza es una transacción plana, pero en realidad trabajan con recursos ubicados en diferentes sistemas.

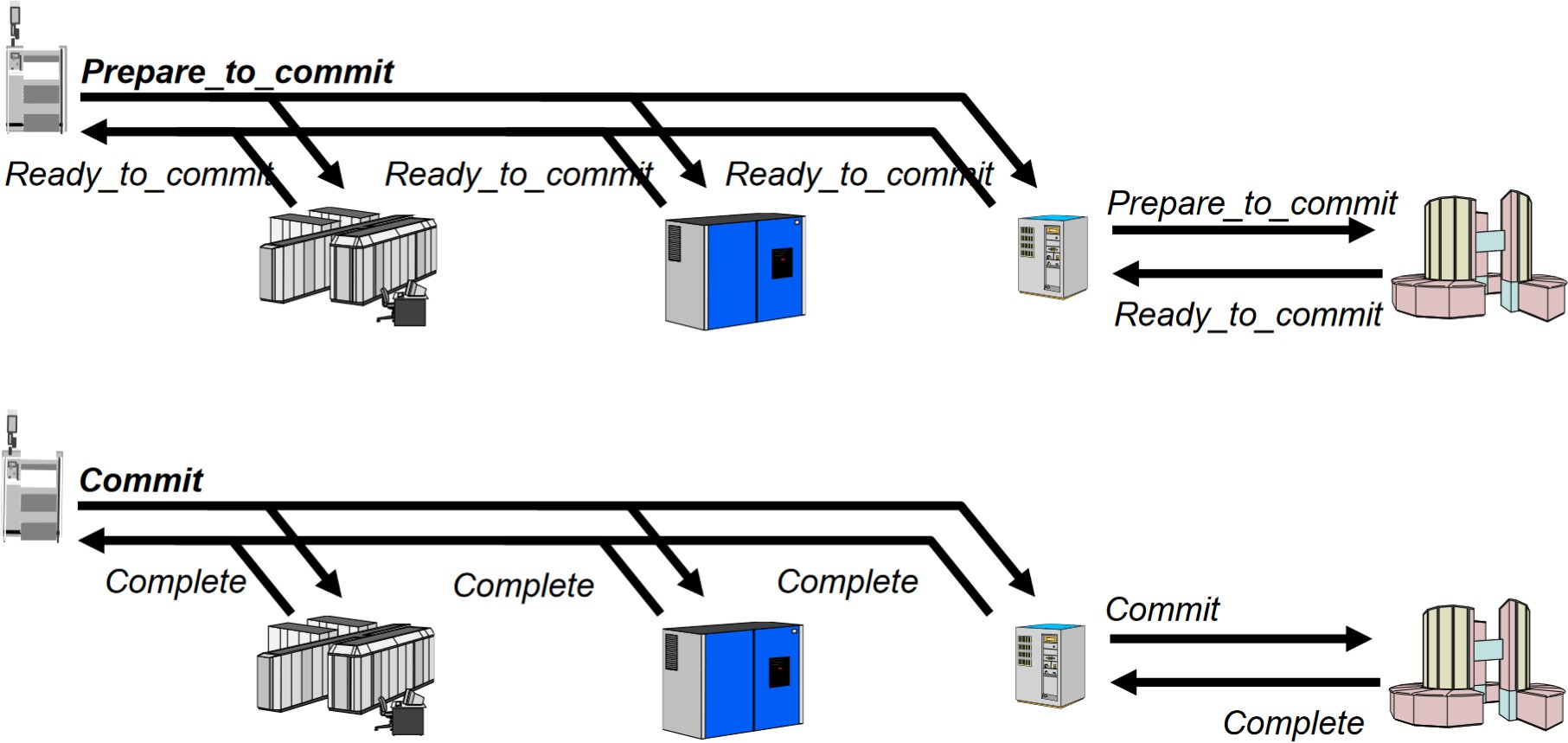
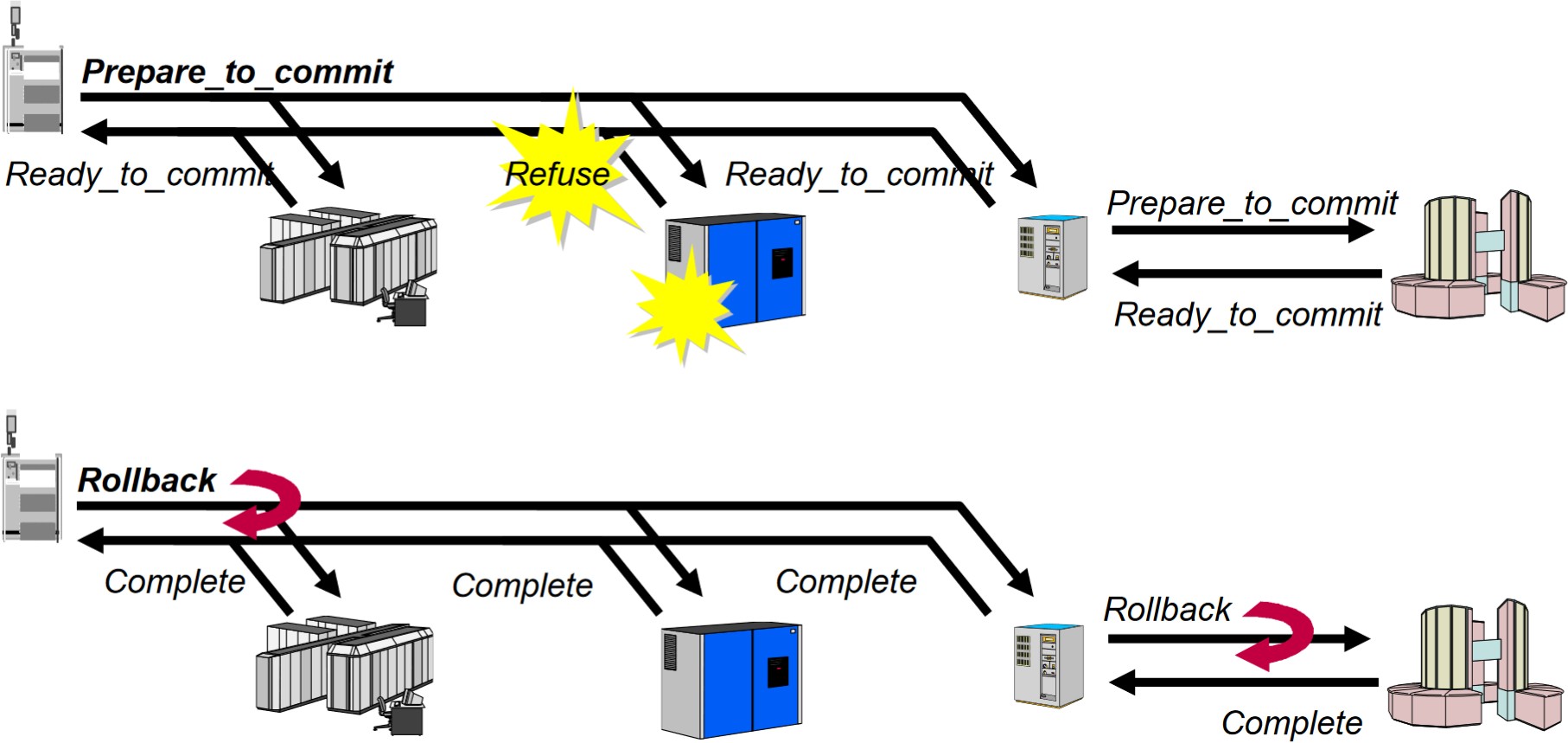
El monitor de proceso transaccional que la controla debe garantizar que la transacción continúa siendo ACID:

* Necesario garantizar la destrucción de las operaciones en todos los sistemas afectados en caso de

que algo falle

* Requiere un nuevo proceso de validación de la transacción: **Two-Phase Commit.** Consiste en lanzar primero un Prepare desde el cliente donde los diferentes servidores devuelven un Ready. La segunda fase es lanzar el Commit desde el servidor, para que finalmente los servidores devuelvan el Complete (Ver imagen).

Si en vez de un Ready un servidor respondiese Refuse, se haría Rollback (Volver a la situación original en todos los servidores).



# Limitaciones de las transacciones planas: planificación viajes

Imaginemos una aplicación de reserva de viajes. Asumamos que se quiere ir desde Madrid a Nueva York

BEGIN;

P1: Compra un vuelo desde Madrid a Londres

P2: Compra un vuelo desde Londres a Pittsburgh, el mismo día por la noche

P3: Compra un vuelo desde Pittsburgh a Nueva York, al día siguiente por la noche ...

Problema: No hay forma de ir desde Pittsburgh a Nueva York, salvo en autobús, y no se quiere utilizar debido al largo periodo de viaje

¿Qué puede hacer el usuario en esta situación?

• Una posibilidad es viajar desde Londres a Nueva York directamente, que es más caro

• O viajar desde Londres a Boston y hacer escala de 3h

• ... Explorar más desde Londres o Madrid Sea cual fuera la opción elegida, únicamente se contemplan dos opciones:

• Cancelar todos los vuelos, lo que puede ser una tarea compleja. Además, es posible que se tenga que abonar una cantidad por la cancelación o directamente perder el coste de los vuelos por no existir la posibilidad de cancelar

• Tratar de cancelar únicamente los vuelos que no sirvan de utilidad en la nueva re-planificación del viaje ¿Es esto posible con las transacciones planas?

# Limitaciones de las transacciones planas: actualización masiva

# • Supongamos un sistema que tiene 5.000.000 de clientes

• Se cuenta con una transacción que procesa ciertos datos de estos clientes, cuyo tiempo de cómputo se estima en 0.005 segundos por cliente (~7h)

• Asumamos que la transacción ha procesado el 99.35% de los clientes (4967500)

• En este momento se produce un error no contemplado en la transacción.

• Debido a atomicidad de las transacciones, toda la información procesada será reestablecida al estado consistente anterior. Por lo que 4967500 actualizaciones serán revertidas

• Despues del rollback, la base de datos del sistema se encontrará en el mismo estado en el que estaba antes de ejecutar la transacción, perdiendo 7h de trabajo (correcto)

# Transacciones Anidadas:

Llamada a una transacción desde otra (modelo subrutinas). Un commit de una transacción hace sus cambios visibles a todas las transacciones precedentes en la jerarquía de llamadas

Diagrama

Descripción generada automáticamenteUn rollback de una transacción realiza un rollback de todas las subtransacciones que haya ejecutado, aunque estas ya hayan realizado un commit.

# Texto Descripción generada automáticamente

# 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, sino que se vuelve al último savepoint.

Los efectos de la transacción sólo serán permanentes y visibles “desde fuera” con el commit de la transacción.

Los Savepoints y las Transacciones Anidadas no resuelven el problema de la caída de un servidor en plena transacción.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Resolviendo el problema de la actualización masiva

• ¿Utilizamos savepoints?

• Si se produce un fallo en algún punto intermedio, se puede realizar rollback a uno de los savepoints hasta completar el proceso

• ¿Utilizamos transacciones anidadas?

• Si se produce algún error en las subtransacciones, se puede gestionar a través del arbol de transacciones principal

• ¿Y si se produce una caída del servidor? ¿Qué ocurre?

• Los savepoints o las transacciones anidadas no son la solución a todos los problemas existentes. Deben utilizarse como mecanismo a la hora de encontrar soluciones, más allá de ser la solución en sí

• Ejercicio: analizar el problema y proponer una solución

• ¿Utilizar savepoints? ¿Transacciones anidadas?

Texto, Carta

Descripción generada automáticamente• ¿Dividir la transacción en varias partes?

• ¿Utilizar transacciones planas distribuidas?

# 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.

Reserv

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.

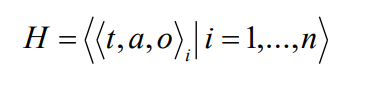
Consecuencia:

* + Los algoritmos de control de la concurrencia deben ser sencillos y eficientes.

No

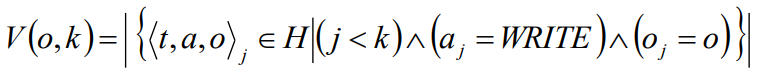
**DEFINICIONES SOBRE EL AISLAMIENTO**

Historia de un conjunto de transacciones: Secuencia que preserva la mezcla de acciones que tiene lugar al ejecutarse un conjunto de transacciones. (Con el ejemplo de abajo se entiende fácil).



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.



Ejemplo 1 Acciones:

T1 READ A T2 WRITE C T3 WRITE C T2 READ B

Historia:

H = < <T1, READ, A>, <T2, WRITE, C>, <T3, WRITE, C>, <T2, READ, B> >

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

Ejemplo 2: Historias en serie y serializable (Ejemplo del inicio del tema) En serie:

H1 = < <T1, READ, A>,<T1, WRITE, A>,<T1, READ, B>,

<T1, WRITE,B>, <T2, READ, A>,<T2, WRITE, A>,<T2, READ, B>,

<T2, WRITE, B> >



No serializable:

H3 = < <T1, READ, A>,<T1, WRITE, A>, <T2, READ,A>,

<T2, WRITE, A>,<T2, READ, B>,<T2, WRITE, B>, <T1, READ, B>,

<T1,WRITE, B> >

# TIPOS DE VIOLACIONES DE 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>

# Lectura Fantasma

Parecido a lectura no repetible pero con un insert.

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

TIPOS DE VIOLACIONES

Diagrama

Descripción generada automáticamente

# EJEMPLO: ACTUALIZACIÓN PERDIDATabla Descripción generada automáticamente

# EJEMPLO: LECTURA SUCIA

# Tabla Descripción generada automáticamente

# EJEMPLO: LECTURA NO REPETIBLE

# Tabla Descripción generada automáticamente

# EJERCICIO DE PROBLEMAS EN EL AISLAMIENTO EN POSTGRES

# • Poned la sesión en read uncommited para que se puedan producir los distintos problemas de lectura:

# • SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

# • Comprobar y analizar los resultados en las siguientes transacciones.

# ACTUALIZACIÓN PERDIDA

# Tabla Descripción generada automáticamente

# LECTURA SUCIA

# Tabla, Escala de tiempo Descripción generada automáticamente

# LECTURA NO REPETIBLE

# Tabla Descripción generada automáticamente

# LECTURA FANTASMA

# Tabla Descripción generada automáticamente

# 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:

**ACCIONES DENTRO DE UNA TRANSACCIÓN**

|  |  |
| --- | --- |
| Acciones sobre objetos | Acciones genéricas: |
| **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. | **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. |

# TRANSACCIONES BIEN FORMADAS

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

# TRANSACCIONES EN DOS FASES

Todos los bloqueos preceden a todos los desbloqueos Fase de crecimiento: adquiere los bloqueos

Fase de contracción: libera los bloqueos

Tabla

Descripción generada automáticamenteEstos dos tipos de transacciones son compatibles y sus propiedades pueden combinarse para aumentar su grado de aislamiento.

TRANSACCIÓN BIEN FORMADA

Tabla

Descripción generada automáticamente TRANSACCIÓN EN DOS PASOS

EJEMPLO AISLAMIENTO

Tabla

Descripción generada automáticamente

# 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.

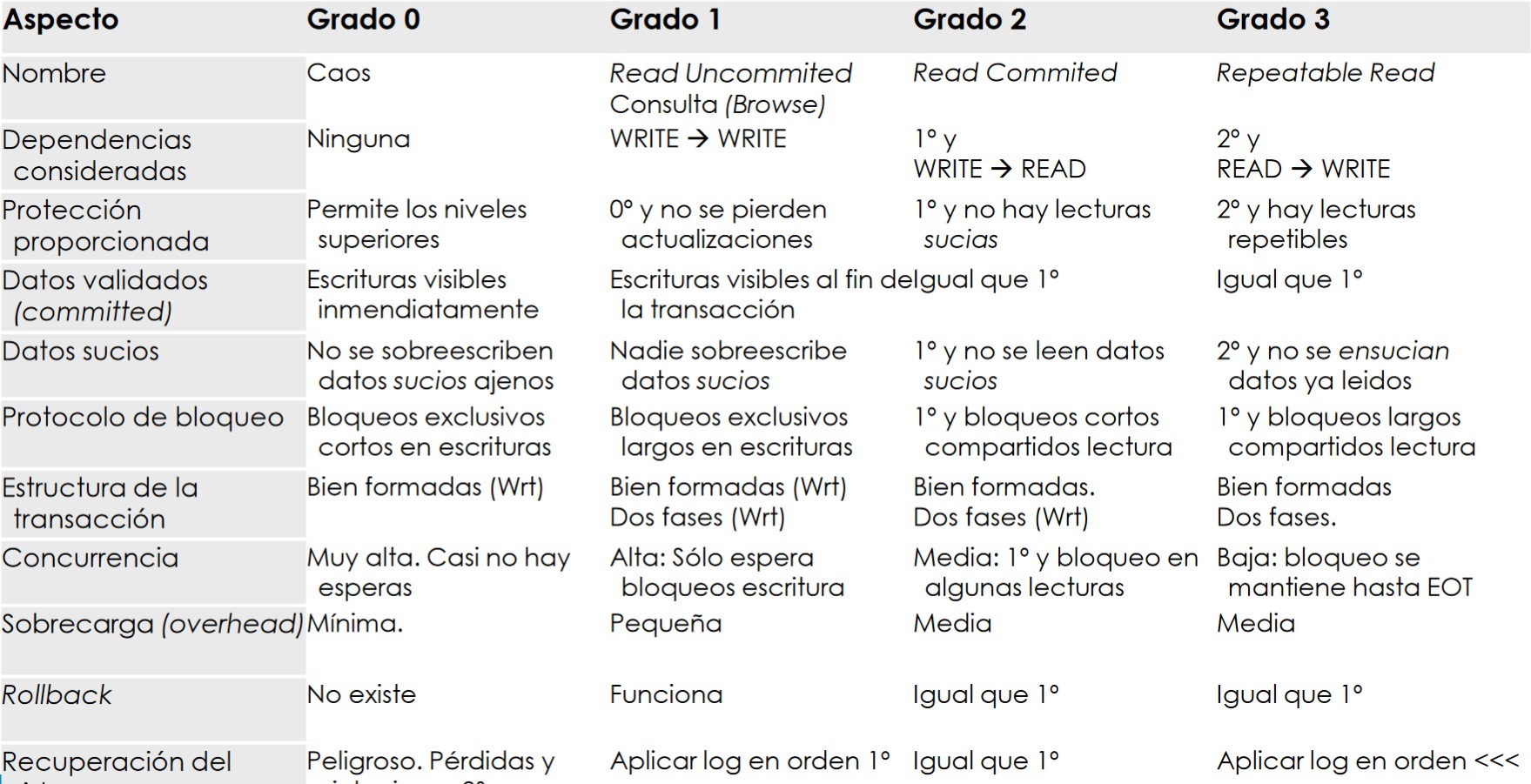
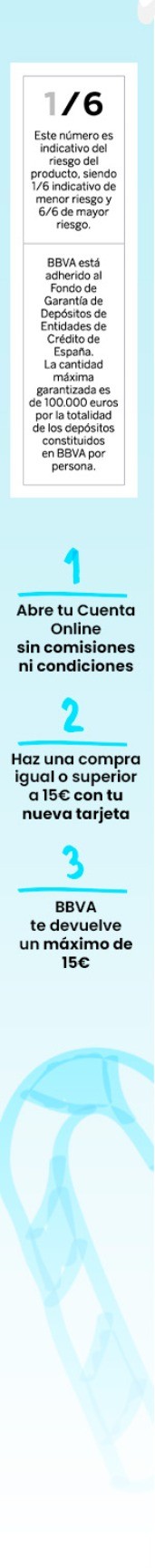
**GRADOS DE AISLAMIENTO**

El aislamiento completo es costoso

Muchos fabricantes permiten activar selectivamente el grado de aislamiento deseado para el sistema:

* 0º (grado 0) caos. Las transacciones no sobreescriben datos sucios de transacciones de nivel 1º o superior
* 1º (grado 1) lecturas no comprometidas. No hay pérdida de actualizaciones en el ámbito de la transacción (hasta el COMMIT)
* 2º (grado 2) lecturas comprometidas. No hay pérdida de actualizaciones en el ámbito de la transacción ni lectura de datos sucios
* 3º (grado 3) lectura repetible. No hay pérdida de actualizaciones ni lectura de datos sucios y hay lecturas repetitivas. Es el aislamiento completo garantizado.
* 4º (grado 4) Serializable. Imita el comportamiento en serie. No hay razón para usarlo.

Menor grado de aislamiento facilita la concurrencia y evita esperas en las transacciones, pero puede producir violaciones de aislamiento.



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

GRADOS DE AISLAMIENTO EN SQL

Texto

Descripción generada automáticamente

PROBLEMAS CON GRADOS DE AISLAMIENTO INFERIORES AL 3

Tabla

Descripción generada automáticamente con confianza baja

**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.

Diagrama

Descripción generada automáticamente

**BLOQUEOS FOR UPDATE**

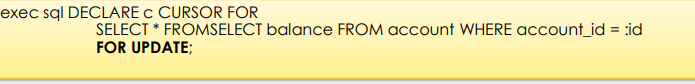
Bloqueo para cursores declarados para realizar actualizaciones sobre los registros encontrados:

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

Tabla

Descripción generada automáticamente

# INTERBLOQUEO (DEADLOCK)

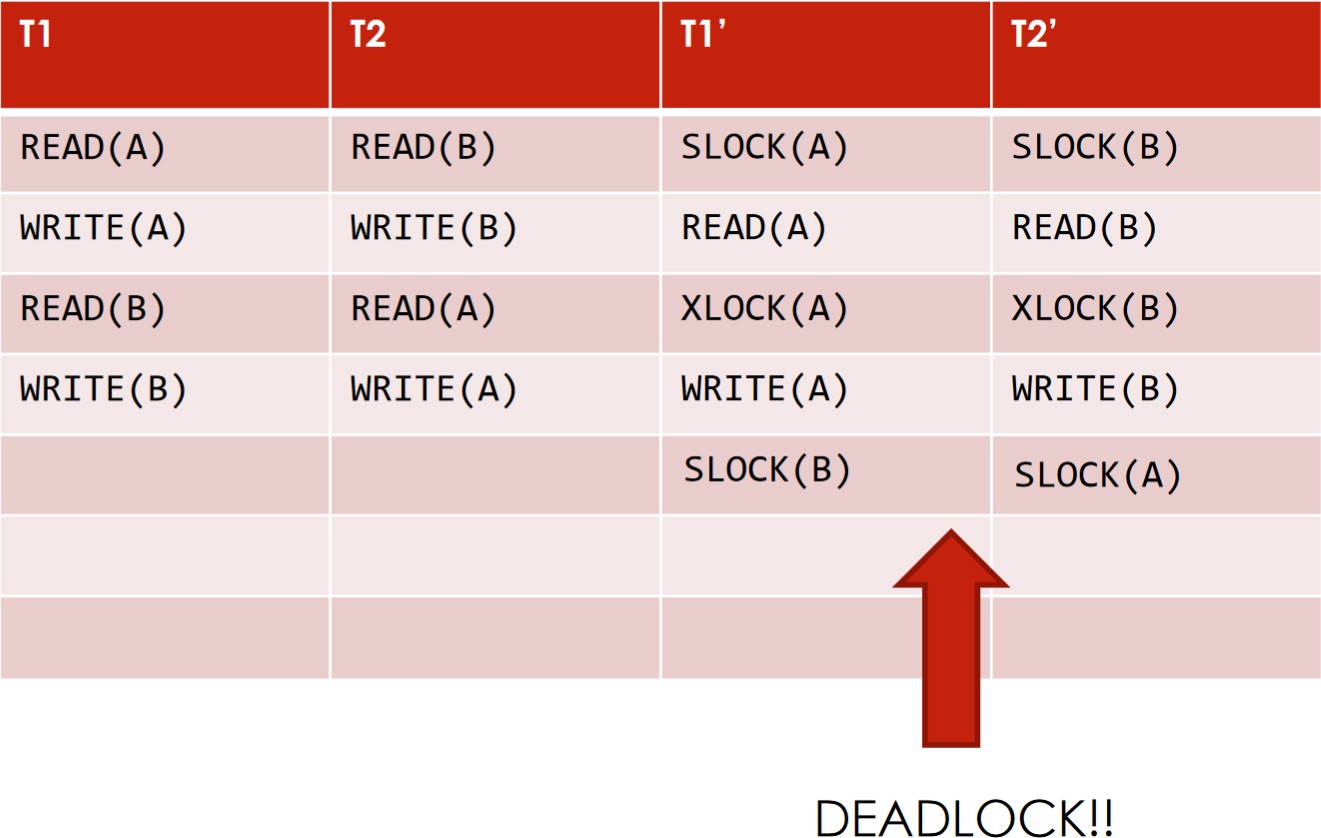
* Situación de bloqueo recíproco de recursos, que producen espera ilimitada:

T1 XLOCK A T2 XLOCK B T1 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

**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.

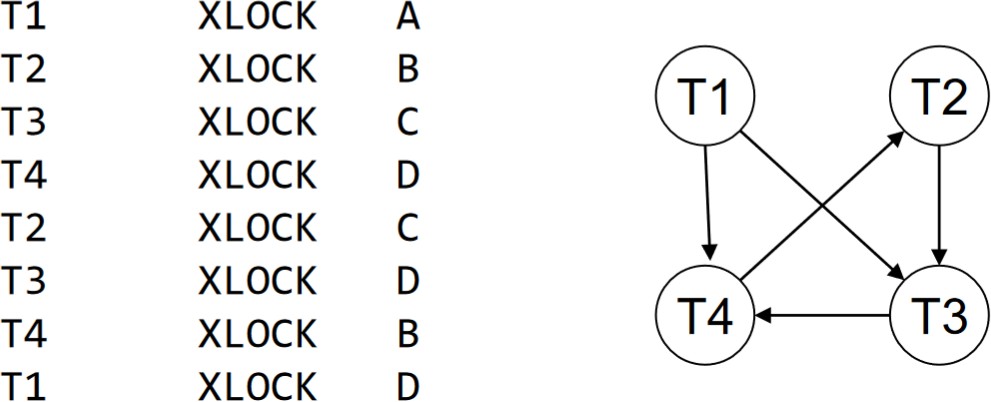


Imagen que contiene Tabla

Descripción generada automáticamente

