

Índice

- 3.1 Concepto de transacción y estados
- 3.2 Operaciones de almacenamiento
- 3.3 Concurrency
- 3.4 Serializabilidad
- 3.5 Técnicas de control de concurrencia
- 3.6 Concurrency: Oracle
- 3.7 Ejercicios

Bibliografía

- Fundamentals of Database Systems. 6th Edition
R. Elmasri y S.B. Navathe. Addison Wesley, 2010
Capítulo 20 y 21
- Database: Principles, Programming, and Performance, 2ª Edición
P. O'Neil y E. O'Neil. Morgan Kaufmann, 2000
Capítulo 10
- Fundamentos de bases de datos. 4ª edición
A. Silberschatz, H.F. Korth y S. Sudarshan. McGraw-Hill, 2002
Capítulo 15
- Oracle 10g: Manual del administrador
K. Loney. McGraw-Hill, 2005
Capítulo 7



3.1 Concepto de Transacción y Estados

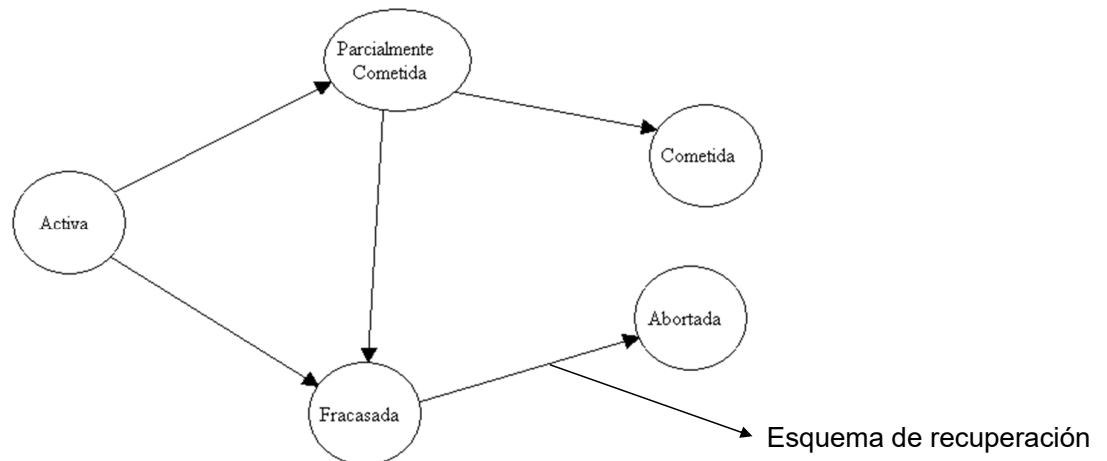
- Transacción:
 - Unidad de programa que accede a la base de datos y que mantiene la consistencia
 - Unidad lógica de procesamiento de base de datos
 - Modelo lógico de un cambio de estado en el universo modelado
- Propiedades (conocidas como propiedades ACID):
 - **Atomicidad:** Se deben ejecutar todas las operaciones de la transacción o ninguna. Esta propiedad es responsabilidad del DBMS y para garantizarla debe proporcionar un esquema de recuperación de caídas.
 - **Consistencia:** Las operaciones que realiza la transacción deben mantener la consistencia de la base de datos. Es responsabilidad del programador.
 - **Aislamiento:** Una transacción debe parecer que se está ejecutando de forma aislada respecto al resto de transacciones. Esta propiedad es responsabilidad del DBMS y para garantizarla debe proporcionar algoritmos de procesamiento de transacciones concurrentes.
 - **Durabilidad:** Los cambios en la base de datos realizados por una transacción cometida deben permanecer en la base de datos aunque se produzcan fallos posteriores. Esta propiedad es responsabilidad del DBMS y para garantizarla debe proporcionar un esquema de recuperación de caídas.



3.1 Concepto de Transacción y Estados

Tema 3: Gestión de transacciones. Parte I.

- Estados de una transacción:
 - Activa: La transacción ha comenzado su ejecución
 - Parcialmente cometida: Justo después de ejecutar la última operación de la transacción
 - Fracasada: La transacción no puede seguir con la ejecución normal de sus operaciones
 - Abortada: La base de datos se deja como estaba justo antes de empezar la ejecución de la transacción. Después se reiniciará (fallo de hw, red, ...) o se eliminará (fallo en la programación de la transacción, ...).
 - Cometida: Las modificaciones realizadas por la transacción han quedado almacenadas en almacenamiento estable



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.1. Concepto de Transacción y Estados. Soporte SQL

Tema 3: Gestión de transacciones. Parte I.

- Cometer/iniciar una transacción

- COMMIT;

- Deshacer una transacción

- ROLLBACK;

- Ejemplo:

```
INSERT INTO mecanico
VALUES ('5555', 'LUIS', 'MOTOR');
INSERT INTO mecanico
VALUES ('6666', 'PEPE', 'CHAPA');
SELECT * FROM mecanico;
ROLLBACK;
INSERT INTO mecanico
VALUES ('6666', 'PEPE', 'CHAPA');
COMMIT;
ROLLBACK;
SELECT * FROM mecanico;
```

DNI Nombre Puesto

5555 LUIS MOTOR
6666 PEPE CHAPA

DNI Nombre Puesto

6666 PEPE CHAPA



Grado en Informática

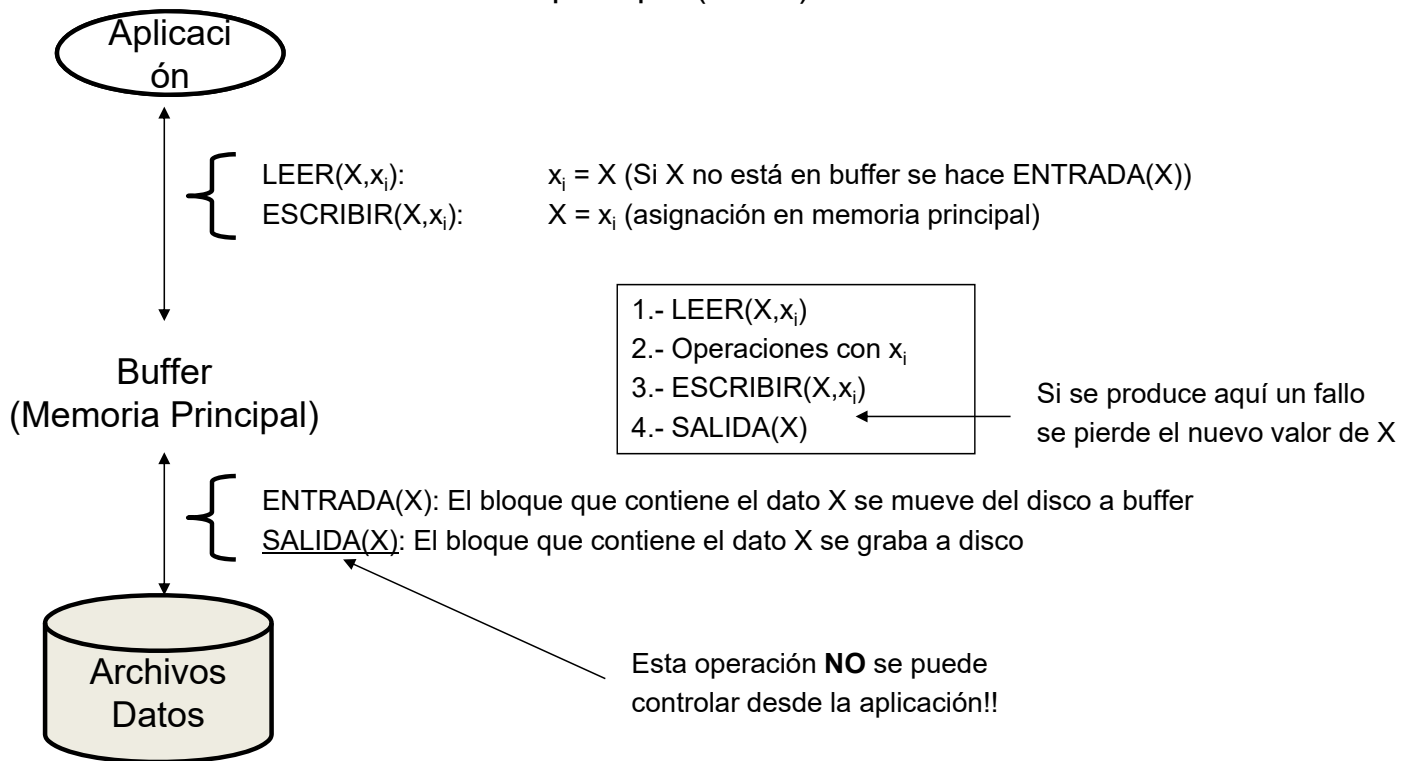
Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.2 Operaciones de Almacenamiento

Tema 3: Gestión de transacciones. Parte I.

- El transvase de datos entre el disco y la aplicación se realiza a través de bloques
- Se utiliza una zona de memoria principal (buffer) como almacenamiento intermedio



3.3 Concurrencia

Tema 3: Gestión de transacciones. Parte I.

- Varias transacciones introducidas por usuarios, que se ejecutan de manera concurrente, pueden leer/modificar los mismos elementos almacenados en la base de datos
- Razones para permitir la concurrencia:
 - Aumentar la productividad: número de transacciones ejecutadas por minuto.
 - Aumentar la utilización de la CPU (menos tiempo ociosa) y Control del disco.
 - Reducir el tiempo medio de respuesta de transacciones (las 'pequeñas' no esperan a las 'grandes').



3.3 Concurrencia: problemas potenciales

Tema 3: Gestión de transacciones. Parte I.

Dos operaciones entran en **conflicto** si (i) pertenecen a dos transacciones distintas, (ii) acceden al mismo elemento X y (iii) al menos una de las dos operaciones es de escritura

Ejemplo: sistema de reservas de vuelos. Sean dos transacciones T1 y T2 concurrentes:

- T1 transfiere N reservas realizadas en un vuelo X a otro vuelo Y
- T2 reserva M plazas en el vuelo X

Transacción T1

```
LEER (X, xi) ;  
xi := xi -N;  
ESCRIBIR (X, xi) ;  
LEER (Y, yi) ;  
yi := yi +N;  
ESCRIBIR (Y, yi) ;
```

Transacción T2

```
LEER (X, xi) ;  
xi := xi +M;  
ESCRIBIR (X, xi) ;
```



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.3 Concurrencia: algunos problemas potenciales

Tema 3: Gestión de transacciones. Parte I.

- actualización
perdida

T1
LEER (X, x_i) ;
x_i := x_i -N;

ESCRIBIR (X, x_i) ;
LEER (Y, y_i) ;

y_i := y_i +N;
ESCRIBIR (Y, y_i) ;

T2
LEER (X, x_j) ;
x_j := x_j +M;

ESCRIBIR (X, x_j) ;

- resumen
incorrecto

T1
LEER (X, x_i) ;
x_i := x_i -N;
ESCRIBIR (X, x_i) ;

LEER (Y, y_i) ;
y_i := y_i +N;
ESCRIBIR (Y, y_i) ;

T3
suma:=0;
LEER (A, a_j) ;
suma+=a_j;
...
...
LEER (X, x_j) ;
suma+=x_j;
LEER (Y, y_j) ;
suma+=y_j;

- actualización
temporal
(lectura sucia)

T1
LEER (X, x_i) ;
x_i := x_i -N;
ESCRIBIR (X, x_i) ;

LEER (Y, y_i) ;



T2
LEER (X, x_j) ;
x_j := x_j +M;

ESCRIBIR (X, x_j) ;

- lectura no
repetible

T1
LEER (X, x_i) ;
x_i := x_i -N;

ESCRIBIR (X, x_i) ;
LEER (Y, y_i) ;

LEER (Y, y_i) ;
ESCRIBIR (Y, y_i) ;

T4

LEER (X, x_j) ;

...
LEER (X, x_j) ;
...



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.3 Concurrencia: concepto de planificación

Tema 3: Gestión de transacciones. Parte I.

- Una planificación P de n transacciones concurrentes $T_1, T_2 \dots T_n$ es una secuencia de las operaciones realizadas por dichas transacciones, sujeta a la restricción de que
 - (i) para cada transacción T_i que participa en P ,
 - (ii) sus operaciones aparecen en P
 - (iii) en el mismo orden en el que ocurren en T_i

operación	abreviatura
LEER	l
ESCRIBIR	e
commit	c
rollback	r

Ejemplos de planificaciones de transacciones(*)

$P_A: l_1(X); e_1(X); l_1(Y); e_1(Y); c_1; l_2(X); e_2(X); c_2;$
 $P_B: l_2(X); e_2(X); c_2; l_1(X); e_1(X); l_1(Y); e_1(Y); c_1;$
 $P_C: l_1(X); l_2(X); e_1(X); l_1(Y); e_2(X); c_2; e_1(Y); c_1;$
 $P_D: l_1(X); e_1(X); l_2(X); e_2(X); c_2; l_1(Y); e_1(Y); c_1;$
 $P_E: l_1(X); e_1(X); l_2(X); e_2(X); c_2; l_1(Y); r_1;$

(*) El subíndice de cada operación indica la transacción que la realiza



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.3 Concurrencia: problemas potenciales

Tema 3: Gestión de transacciones. Parte I.

- ¿Cómo evitar planificaciones “problemáticas”?
- Estrategias basadas en seriabilidad:
 - Planificaciones que dejan la base de datos en un estado consistente
- Estrategia basada en la recuperabilidad:
 - Si el DBMS tiene la capacidad de deshacer el efecto de una transacción. Las transacciones pueden ser:
 - **No recuperable:** permite confirmar datos leídos de transacciones no confirmadas
 $P_A: e_1(A); e_1(B); e_2(A); l_2(B); c_2$
 - **Recuperable:** permite leer datos de transacciones no confirmadas, pero no confirmarlos
 $P_A: e_1(A); e_1(B); e_2(A); l_2(B); c_2$
 - **Evita borrado en cascada:** no permite leer datos de transacciones no confirmadas
 $P_A: e_1(A); e_1(B); e_2(A); c_1; l_2(B); c_2$
 - **Estricta:** no permite leer ni escribir datos de transacciones no confirmadas
 $P_A: e_1(A); e_1(B); c_1; e_2(A); l_2(B); c_2$



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

- Objetivo de un protocolo de control de concurrencia:
 - Planificar las transacciones de forma que no ocurran interferencias entre ellas

Solución trivial e ineficiente: prohibir intercalar operaciones (planificación serie)

<u>T1</u>	<u>T2</u>
LEER (X, x_1);	
$x_1 := x_1 - N$;	
ESCRIBIR (X, x_1);	
LEER (Y, y_1);	
	LEER (X, x_2);
	$x_2 := x_2 + M$;
	ESCRIBIR (X, x_2);



3.4 Serializabilidad: planificación de transacciones

- **Planificación correcta:** si deja la base de datos en un estado consistente (evita problemas de concurrencia)
- **Planificación serie:** sólo una transacción por vez. Es trivialmente correcta e ¡ineficiente!
- **Planificación serializable:** planificación no serie que obtiene igual resultado que un planificación serie → es correcta
- **Planificaciones equivalentes:** si se mantiene el orden de cada operación → objetivo de la **serializabilidad** es buscar una planificación equivalente y serializable:
 - Equivalencia por conflictos
 - Equivalencia de vistas

$P_C: I_1(X); I_2(X); e_1(X); I_1(Y); e_2(X); c_2; e_1(Y); c_1;$

$P_D: I_1(X); e_1(X); I_2(X); e_2(X); c_2; I_1(Y); e_1(Y); c_1; ✓$



3.4 Serializabilidad: planificación de transacciones

Tema 3: Gestión de transacciones. Parte I.

- **Equivalencia por conflictos:** se mantiene el orden de aquellas operaciones en conflicto
- Una planificación P es **serializable por conflictos** si equivale por conflictos a alguna planificación serie S

$P_C: l_1(X); l_2(X); e_1(X); l_1(Y); e_2(X); c_2; e_1(Y); c_1;$
 $P_D: l_1(X); e_1(X); l_2(X); e_2(X); c_2; l_1(Y); e_1(Y); c_1;$

$P_D: l_1(X); e_1(X); l_2(X); e_2(X); c_2; l_1(Y); e_1(Y); c_1;$

$P_{D1}: l_1(X); e_1(X); l_2(X); e_2(X); l_1(Y); c_2; e_1(Y); c_1;$

$P_{D2}: l_1(X); e_1(X); l_2(X); e_2(X); l_1(Y); e_1(Y); c_2; c_1;$

...

$P_{DS}: l_1(X); e_1(X); l_1(Y); e_1(Y); c_1; l_2(X); e_2(X); c_2;$

¡Es una planificación serie!
PD es serializable



Grado en Informática

Gestión y Administración de Bases de Datos

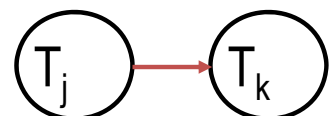
Fernando Martínez Santiago

3.4 Serializabilidad: planificación de transacciones

Tema 3: Gestión de transacciones. Parte I.

Detección de la serializabilidad por conflictos

- Construcción del **grafo de precedencia** (o de serialización)
 - Es un grafo dirigido $G = (N, A)$
 - N es un conjunto de **nodos** y A es un conjunto de **aristas dirigidas**
 - Algoritmo:
 - Crear **un nodo por** cada **transacción** T_i en P
 - Crear una arista $T_j \rightarrow T_k$ si T_k **lee** el valor de un elemento **después** de que T_j lo haya **escrito**
 - Crear una arista $T_j \rightarrow T_k$ si T_k **escribe** el valor de un elemento **después** de que T_j lo haya **leído**
 - Crear una arista $T_j \rightarrow T_k$ si T_k **escribe** el valor de un elemento **después** de que T_j lo haya **escrito**



Grado en Informática

Gestión y Administración de Bases de Datos

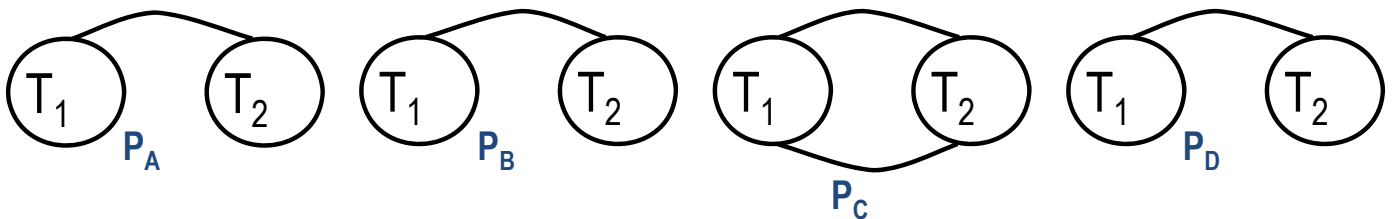
Fernando Martínez Santiago

3.4 Serializabilidad: planificación de transacciones

Tema 3: Gestión de transacciones. Parte I.

Detección de la serializabilidad por conflictos (y 2)

- Una arista $T_j \rightarrow T_k$ indica que T_j debe aparecer antes que T_k en una planificación serie equivalente a P , pues dos operaciones en conflicto aparecen en dicho orden en P
- Si el grafo contiene un **ciclo**, P no es serializable por conflictos
 - Un **ciclo** es una secuencia de aristas $C = ((T_j \rightarrow T_k), (T_k \rightarrow T_p), \dots, (T_i \rightarrow T_j))$
- Si no hay ciclos en el grafo, P es serializable
 - 4 Es posible obtener una planificación serie S equivalente a P , mediante una ordenación topológica de los nodos



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.4 Serializabilidad: planificación de transacciones

Tema 3: Gestión de transacciones. Parte I.

Ejemplo de planificación no serializable

Transacción T1

leer(X);
escribir(X);
leer(Y);
escribir(Y);

Transacción T2

leer(Z);
leer(Y);
escribir(Y);
leer(X);
escribir(X);

Transacción T3

leer(Y);
leer(Z);
escribir(Y);
escribir(Z);

T1

leer(X);
escribir(X);

leer(Y);
escribir(Y);

T2

leer(Z);
leer(Y);
escribir(Y);

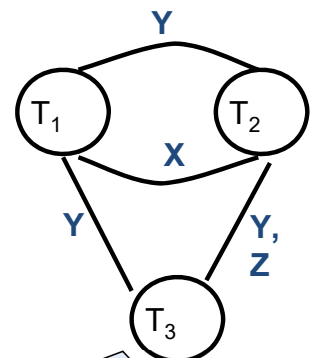
leer(X);

escribir(X);

T3

leer(Y);
leer(Z);

escribir(Y);
escribir(Z);



Hay dos ciclos:

$T_1 \rightarrow T_2 \rightarrow T_1$ y
 $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$

Planificación E



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

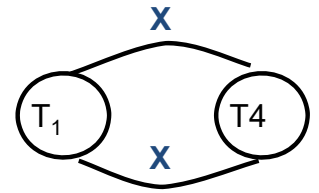
3.4 Serializabilidad: planificación de transacciones

Tema 3: Gestión de transacciones. Parte I.

Ejemplos de planificación no serializable

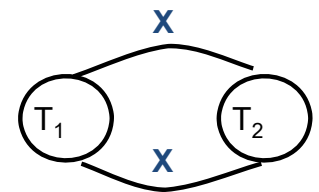
- lectura no repetible

<u>T1</u>	<u>T4</u>
LEER(X, x _i);	
x _i := x _i - N;	
ESCRIBIR(X, x _i);	LEER (X, x _j);
LEER(Y, y _i);	...
	LEER (X, x_j);
	...
LEER(Y, y _i);	
ESCRIBIR(Y, y _i);	



- actualización perdida

<u>T1</u>	<u>T2</u>
LEER(X, x _i);	
x _i := x _i - N;	
ESCRIBIR(X, x _i);	LEER (X, x _j);
LEER(Y, y _i);	x _j := x _j + M;
	ESCRIBIR(X, x_j);
Y _i := Y _i + N;	
ESCRIBIR(Y, y _i);	



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.4 Serializabilidad: planificación de transacciones

Tema 3: Gestión de transacciones. Parte I.

Ejemplo de planificación serializable

Transacción T1

leer(X);
escribir(X);
leer(Y);
escribir(Y);

Transacción T2

leer(Z);
leer(Y);
escribir(Y);
leer(X);
escribir(X);

Transacción T3

leer(Y);
leer(Z);
escribir(Y);
escribir(Z);

T1

leer(X);
escribir(X);

leer(Y);
escribir(Y);

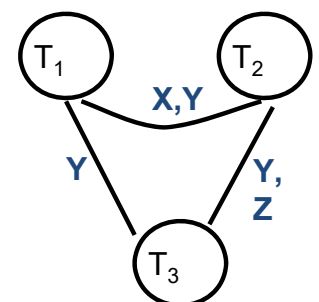
T2

leer(Z);

leer(Y);
escribir(Y);
leer(X);
escribir(X);

T3
leer(Y);
leer(Z);

escribir(Y);
escribir(Z);



Planificación F

La planificación serie equivalente es T3 → T1 → T2



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.5 Técnicas de control de concurrencias

Tema 3: Gestión de transacciones. Parte I.

- Un DBMS no puede anticipar cómo se intercalarán las operaciones de las transacciones → ¿Cómo usar la teoría de la serializabilidad?
- Protocolos más usuales:
 - Métodos de bloqueo
 - Métodos de marca de tiempo
 - Técnicas de multiversión
 - Métodos optimistas



3.5 Técnicas de control de concurrencias: métodos de bloqueo

Tema 3: Gestión de transacciones. Parte I.

- Dada una transacción T que opera sobre un elemento X puede obtener:
 - Bloqueo compartido (`bloquear_lectura(X)`):
 - permisos de lectura sobre X
 - varias transacciones acceden a X simultáneamente
 - Bloqueo exclusivo (`bloquear_escritura(X)`):
 - permisos de lectura y escritura sobre X
 - Sólo una transacción T accede a X en un instante dado
 - El nuevo valor de T sólo será visible por las demás transacciones en el momento que finalice el bloqueo exclusivo



3.5 Técnicas de control de concurrencias: métodos de bloqueo

Tema 3: Gestión de transacciones. Parte I.

Reglas de uso de los bloqueos

1. T debe emitir `bloquear_lectura(X)` o `bloquear_escritura(X)` antes de ejecutar una operación `leer(X)`
2. T debe emitir `bloquear_escritura(X)` antes de realizar una operación `escribir(X)` en T
3. T debe emitir `desbloquear(X)` una vez completadas todas las operaciones `leer(X)` y `escribir(X)`
4. Si T ya posee un bloqueo, compartido o exclusivo, sobre X no emitirá `bloquear_lectura(X)` ni `bloquear_escritura(X)`
5. T no emitirá `desbloquear(X)` salvo si posee un bloqueo, compartido o exclusivo, sobre X



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.5 Técnicas de control de concurrencias: métodos de bloqueo

Tema 3: Gestión de transacciones. Parte I.

- El uso de bloqueos para la programación de transacciones no garantiza la serializabilidad de las planificaciones

Transacción T4
`bloquear_lectura(Y);`
`leer(Y);`
`desbloquear(Y);`
`bloquear_escritura(X);`
`leer(X);`
`X:=X+Y;`
`escribir(X);`
`desbloquear(X);`

Transacción T5
`bloquear_lectura(X);`
`leer(X);`
`desbloquear(X);`
`bloquear_escritura(Y); leer(Y);`
`Y:=X+Y;`
`escribir(Y);`
`desbloquear(Y);`

Valores iniciales: X=20, Y=30
Resultados de las planificaciones serie:
T4→T5: X=50, Y=80
T5→T4: X=70, Y=50

T4
`bloquear_lectura(Y);`
`leer(Y);`
`desbloquear(Y);`

`bloquear_escritura(X);`
`leer(X);`
`X:=X+Y;`
`escribir(X);`
`desbloquear(X);`

T5

`bloquear_lectura(X);`
`leer(X);`
`desbloquear(X);`
`bloquear_escritura(Y); leer(Y);`
`Y:=X+Y;`
`escribir(Y);`
`desbloquear(Y);`

Resultado de la planificación G:
X=50, Y=50 (¡no serializable!)

Planificación G



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

3.5 Técnicas de control de concurrencias: métodos de bloqueo

Tema 3: Gestión de transacciones. Parte I.

- **Bloqueo en dos fases (B2F):**
 - Fase de expansión (o crecimiento)
 - T puede adquirir bloqueos
 - T no puede liberar ningún bloqueo
 - Fase de contracción
 - T puede liberar bloqueos existentes
 - T no puede adquirir ningún bloqueo
- Garantiza la serializabilidad...
- ... pero limita el grado de concurrencia y puede aparecer **interbloqueo** o **bloqueo indefinido**

Transacción T4'
bloquear_lectura(Y);
leer(Y);
bloquear_escritura(X);
desbloquear(Y);
leer(X);
X:=X+Y;
escribir(X);
desbloquear(X);

Transacción T5'
bloquear_lectura(X);
leer(X);
bloquear_escritura(Y);
desbloquear(X); leer(Y);
Y:=X+Y;
escribir(Y);
desbloquear(Y);



3.5 Técnicas de control de concurrencias: métodos de bloqueo

Tema 3: Gestión de transacciones. Parte I.

Ejemplos de problemas de concurrencia resueltos con B2F

- lectura no repetible

<u>T1</u>	<u>T4</u>
LEER (X, x_i);	
$x_i := x_i - N$;	
ESCRIBIR (X, x_i);	LEER (X, x_j);
LEER (Y, y_i);	...
	LEER (X, x_j);
	...
LEER (Y, y_i);	
ESCRIBIR (Y, y_i);	

Planificación H



<u>T1'</u>	<u>T4'</u>
BLOQUEAR_LECTURA (X) ;	
LEER (X, x_i);	
$x_i := x_i - N$;	
	BLOQUEAR_LECTURA (X) ;
	LEER (X, x_j);
	LEER (X, x_j);
	DESBLOQUEAR (X)
BLOQUEAR_ESCRITURA (X) ;	
ESCRIBIR (X, x_i);	
BLOQUEAR_LECTURA (Y) ;	
LEER (Y, y_i);	
LEER (Y, y_i);	
BLOQUEAR_ESCRITURA (Y) ;	
ESCRIBIR (Y, y_i);	
DESBLOQUEAR (X) ;	
DESBLOQUEAR (Y) ;	

Planificación I



3.5 Técnicas de control de concurrencias: métodos de bloqueo

Tema 3: Gestión de transacciones. Parte I.

- **Bloqueo en dos fases (B2F):**
 - Conservador o estático: T bloquea todos los elementos antes de comenzar
 - Si no puede bloquear algún elemento, no bloquea a ninguno hasta poder bloquearlos todos
 - ¿Sufrirá de interbloqueo? ¿de bloqueo indefinido? ¿recuperable?
 - Estricto: T no libera ningún bloqueo exclusivo hasta terminar
 - Cualquier transacción que presente conflicto con T debe esperar a que T se haya completado
 - ¿Sufrirá de interbloqueo? ¿de bloqueo indefinido? ¿recuperable?
 - Riguroso: T no libera ningún bloqueo exclusivo o compartido hasta terminar
 - Puede bloquear otras transacciones incluso sin que haya conflicto
 - Más fácil de implementar que el estricto
 - ¿Sufrirá de interbloqueo? ¿de bloqueo indefinido? ¿recuperable?



3.5 Técnicas de control de concurrencias: interbloqueo

Tema 3: Gestión de transacciones. Parte I.

- **Interbloqueo:** Situación en la que cada una de dos (o más) transacciones está esperando a que se libere un bloqueo establecido por la otra transacción

<u>T6</u>	<u>T7</u>
bloquear_escritura(X);	bloquear_escritura(Y);
leer(X);	leer(Y);
X:=X-10;	Y:=Y+100;
escribir(X);	escribir(Y);
bloquear_escritura(Y);	bloquear_escritura(X);
[... en espera ...]	[... en espera ...]

¿qué hacer?

- Temporizaciones de interbloqueos
- Prevención de interbloqueos
- Detección de interbloqueos



3.5 Técnicas de control de concurrencias: interbloqueo

Tema 3: Gestión de transacciones. Parte I.

Temporizaciones de bloqueos:

- Una transacción que solicita un bloqueo sólo esperará durante un período de tiempo predefinido por el sistema
- Si no se concede el bloqueo durante ese tiempo, se producirá un 'fin de temporización': el SGBD asumirá que la transacción está interbloqueada (aunque puede que no), la abortará y la reiniciará automáticamente



3.5 Técnicas de control de concurrencias: interbloqueo

Tema 3: Gestión de transacciones. Parte I.

Prevención de interbloqueos:

- Ordenar las transacciones usando **marcas temporales** de transacción $MT(T)$
- Sea T_j que intenta bloquear el elemento de datos X , pero X ya está bloqueado por T_k con un candado en conflicto
- Algoritmo **Esperar - Morir**
 - si $MT(T_j) < MT(T_k)$ entonces T_j puede esperar
 - si no, se aborta T_j (T_j muere) y se reinicia después con la misma marca de tiempo
- Algoritmo **Herir - Esperar**
 - si $MT(T_j) < MT(T_k)$ entonces se aborta T_k (T_j hiere a T_k) y se reinicia después con la misma MT
 - si no, T_j puede esperar

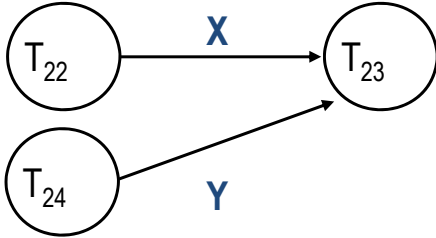


3.5 Técnicas de control de concurrencias: interbloqueo

Tema 3: Gestión de transacciones. Parte I.

Ejemplo:

- Transacciones: T_{22} , T_{23} , T_{24}
- Timestamps: $M(T_{22})=5$, $M(T_{23})=10$, $M(T_{24})=15$



- Algoritmo **wait-die**:

- X: T_{22} espera
- Y: T_{24} rollback

- Algoritmo **wound-wait**:

- X: T_{23} rollback
- Y: T_{24} espera

- Algoritmo **wait-die**

if $M(T_j) < M(T_k)$ then T_j can wait
if not, T_j is aborted (T_j dies) and
 T_j will be restarted with the same timestamp later

- Algoritmo **wound-wait**

if $M(T_j) < M(T_k)$ then aborts T_k (T_j hurts T_k) and
 T_k will be restarted with the same timestamp later
if not, T_j can wait



Grado en Informática

Gestión y Administración de Bases de Datos

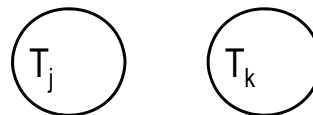
Fernando Martínez Santiago

3.5 Técnicas de control de concurrencias: interbloqueo

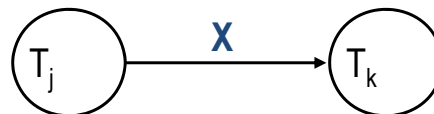
Tema 3: Gestión de transacciones. Parte I.

Detección de interbloqueos:

- Verificación periódica del estado del sistema ¿está en un bloqueo mortal?
- Creación de un **grafo de espera** que muestra las dependencias entre transacciones
 - Crear un nodo por cada transacción en ejecución, etiquetado con el identificador de la transacción, T



- Si T_j espera para bloquear el elemento X , ya bloqueado por T_k , crear una arista dirigida desde T_j a T_k



- Cuando T_k libera el candado sobre X , borrar la arista correspondiente
- Si existe un ciclo en el grafo de espera, entonces se ha detectado un interbloqueo entre las transacciones



Grado en Informática

Gestión y Administración de Bases de Datos

Fernando Martínez Santiago

Transacción T1

```

LEER (X, xi) ;
xi := xi -N;
ESCRIBIR (X, xi) ;
LEER (Y, yi) ;
yi := yi +N;
ESCRIBIR (Y, yi) ;

```

Transacción T2

```

LEER (X, xi) ;
xi := xi +M;
ESCRIBIR (X, xi) ;

```

MT(T1)
MT(T2)

T1

T2

Transacción T1

```

1 BL (X)
2 LEER (X, xi) ;
3 xi := xi -N;

8 BE (X) ← BLOQUEO

```

Transacción T2

```

4 BL (X)
5 LEER (X, xi) ;
6 xi := xi +M;
7 BE (X) ← BLOQUEO

```

MT(T1)=1
MT(T2)=4

```

ESCRIBIR (X, xi) ;
LEER (Y, yi) ;
yi := yi +N;
ESCRIBIR (Y, yi) ;

```

```

ESCRIBIR (X, xi) ;

```

T1

T2



Transacción T1

```

1 BL (X)
2 LEER (X, xi) ;
3 xi := xi -N;
8 BE (X) ← BLOQUEO

```

Transacción T2

```

4 BL (X)
5 LEER (X, xi) ;
6 xi := xi +M;
7 BE (X) ← BLOQUEO

```

MT(T1)=1
MT(T2)=4

```

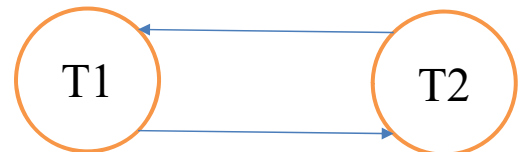
ESCRIBIR (X, xi) ;
LEER (Y, yi) ;
yi := yi +N;
ESCRIBIR (Y, yi) ;

```

```

ESCRIBIR (X, xi) ;

```



Esperar morir:

- T1: T1 espera
- T2 aborta

Herir esperar:

- T1 aborta a T2
- T2 espera

ROLLBACK T2 en el instante 8 o posterior, que es la transacción más joven



3.5 Técnicas de control de concurrencias: interbloqueo

El problema del bloqueo indefinido

- El protocolo de control de concurrencia nunca selecciona a una transacción que está esperando para establecer un bloqueo, mientras otras transacciones continúan ejecutándose con normalidad
 - Ocurre si el esquema de espera da más prioridad a unas transacciones que a otras ➔ esquema de espera injusto
- Dos algoritmos de prevención de bloqueo indefinido
 - Consiguen un esquema de espera justo
 - **Estrategia FIFO**
 - Las transacciones pueden bloquear el elemento X en el orden en que solicitaron su bloqueo
 - **Aumento de prioridad en la espera**
 - Cuanto más espera T, mayor es su prioridad
 - Cuando T tiene la prioridad más alta de todas, obtiene el bloqueo y continúa su ejecución



Niveles de aislamiento en SQL-92:

- Si alguna transacción se ejecuta en algún nivel menor al **SERIALIZABLE**, la seriabilidad puede ser incumplida:

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma (*)
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITTED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

(*)Lectura fantasma:

Transacción 1

```
SELECT * FROM usuarios WHERE edad BETWEEN 10 AND 30;
```

Transacción 2

```
INSERT INTO usuarios VALUES ( 3, 'Bob', 27 );  
COMMIT;
```

```
SELECT * FROM usuarios WHERE edad BETWEEN 10 AND 30;
```



3.6 Concurrencia: Oracle

Niveles de aislamiento:

1. SERIALIZABLE/READ ONLY

- Si T2 serializable intenta ejecutar una sentencia DML que actualiza un dato que puede haber sido modificado por T1 no confirmada en el momento de comenzar T2, entonces dicha sentencia DML **falla**
- Una transacción serializable sólo ve los cambios confirmados en **el instante en que se inicia la transacción**, más los cambios realizados por la propia transacción mediante INSERT, UPDATE, DELETE.
- READ ONLY es un subtipo de SERIALIZABLE en el que solo se permiten consultas. No está definido en SQL-92

2. READ COMMITTED (defecto)

- Si T2 read-committed intenta ejecutar una sentencia DML que necesita filas bloqueadas por T1, entonces **espera** hasta que se liberen los bloqueos de las filas
- Cada consulta ejecutada por una transacción sólo ve los datos confirmados **antes de comenzar la consulta** (no la transacción)

Nótese que en ambos casos los cambios no confirmados no son visibles por el resto de las transacciones → se garantiza que en ORACLE nunca se dará una lectura sucia



3.6 Concurrency: Oracle

Ejemplo de transacción serializable o de solo lectura:

T1

```
t1 commit;
t2 set transaction isolation level serializable;
t3 create table b ( x int );
t4 select count(*) from b -- devuelve "0";
t5
t6
t7

t8 select count(*) from b -- devuelve "0";
t9 commit;
t10 select count(*) from b -- devuelve "1";
```

T2

```
insert into b values(1);
commit;
select count(*) from b --
    devuelve "1";
```

Transacciones serializables son apropiadas si:

- Se prevee que nadie va a modificar datos que están en el mismo bloque concurrentemente
- Se necesita consistencia de lectura a nivel de transacción
- Es una transacción corta (para confirmar que no han cambiado demasiado las cosas)

Precio:

- Datos desfasados
- Posible excepción en tiempo de ejecución (intentas actualizar unos datos que ya fueron actualizados)



3.6 Concurrency: Oracle

- Bloqueos
 - **Gestión Automática** de los bloqueos
 - Bloqueos **exclusivos** y **compartidos**
 - Permiten a otras transacciones leer los datos bloqueados, pero no modificarlos
 - Bloqueos de **tabla** o de **fila** (una o más)
 - Los bloqueos sólo **se liberan al finalizar la transacción** (COMMIT o ROLLBACK)
 - **Gestión Manual**
 - Se superpone al bloqueo automático
 - Sentencia **LOCK TABLE** (no existe **UNLOCK**)



3.7 Ejercicios

1. ¿Por qué la concurrencia es necesaria? ¿qué ocurriría si un DBMS no la soportara?
2. ¿Qué propiedad ACID se ve afectada en sistemas concurrentes?
3. ¿Es siempre deseable que las transacciones sean serializables? ¿por qué?
4. Piensa un escenario donde cada uno de los niveles de aislamiento que define SQL-92 sea admisible, pero no en el nivel justamente anterior.

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma (*)
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

1. Usando el esquema UNIVERSIDAD, escribe en PL/SQL dos transacciones T1 y T2 y una traza de ejecución de estas transacciones que generen:
 1. Actualización perdida
 2. Lectura sucia
 3. Lectura no repetible
 4. Lectura fantasma



3.7 Ejercicios

3. Dada la siguiente situación en ORACLE:

T1

```
t1 commit; -- empieza transacción T1.1
t2 create table b ( x int );
t3 insert into b values(1);
t4 commit; -- empieza transacción T1.2
t5
t6
t7 update b set x=11 where x=1;
t8
```

T2

```
commit;
update b set x=2 where x=1;
commit;
```


- i. ¿en qué momento se ejecutará t7?
- ii. ¿qué ocurrirá cuando se confirme T1.2 tal como está en el ejercicio? ¿y en caso que la transacción se hubiera establecido el nivel serializable?



3.7 Ejercicios

7. ¿Cómo garantiza ORACLE que nunca habrá una lectura sucia?

<u>T1</u> LEER (X, x _i); x _i := x _i -N; ESCRIBIR (X, x _i); LEER (Y, y _i);	<u>T2</u> LEER (X, x _j); x _j := x _j +M; ESCRIBIR (X, x_j);
---	--



8. Un sistema que garantice el nivel de aislamiento serializable, ¿puede tener actualizaciones perdidas?

<u>T1</u> LEER (X, x _i); x _i := x _i -N; ESCRIBIR (X, x _i); LEER (Y, y _i); y _i := y _i +N; ESCRIBIR (Y, y _i);	<u>T2</u> LEER (X, x _j); x _j := x _j +M; ESCRIBIR (X, x_j);
---	--



3.7 Ejercicios

9. ¿Cuál de las siguientes planificaciones es serializable? Demuéstralo usando grafos de precedencias. Para aquellos que sean serializables, ¿Cuál sería la planificación equivalente en serie?

- a. r₁(X); r₃(X); w₁(X); r₂(X); w₃(X);
- b. r₁(X); r₃(X); w₃(X); w₁(X); r₂(X);
- c. r₃(X); r₂(X); w₃(X); r₁(X); w₁(X);
- d. r₃(X); r₂(X); r₁(X); w₃(X); w₁(X);

10. Dado T₁, T₂ y T₃ ¿Cuál de las siguientes planificaciones es serializable? Demuéstralo usando grafos de precedencias. Para aquellos que sean serializables, ¿Cuál sería la planificación equivalente en serie?

T1: r₁(X); r₁(Z); w₁(X);

T2: r₂(Z); r₂(Y); w₂(Z); w₂(Y);

T3: r₃(X); r₃(Y); w₃(Y);

- a. r₁(X); r₂(Z); r₁(Z); r₃(X); r₃(Y); w₁(X); w₃(Y); r₂(Y); w₂(Z); w₂(Y);
- b. r₁(X); r₂(Z); r₃(X); r₁(Z); r₂(Y); r₃(Y); w₁(X); w₂(Z); w₃(Y); w₂(Y);

11. Crea un planificación para las transacciones del ejercicio 10 siguiendo el protocolo B2F estricto



3.7 Ejercicios

12. Dado el siguiente código SQL, ¿qué devolverá la sentencia **select** de la línea 8 (transacción T2) Tema 6: gestión de transacciones

T1

T2

```
1      create table a ( x1 number, x2 number);
2      insert into a values (1, 10);
3      commit;
4
5
6      update x2=20 where x1=1;
7      commit;
8
```

```
set transaction isolation level serializable;
select x2 from a where x1=1;

select x2 from a where x1=1;
```

- a. 10
- b. 20
- c. se generará una excepción en T1
- d. se generará una excepción en T2

13. Dadas dos transacciones T1 y T2 que se ejecutan concurrentemente como a continuación se indica, podemos afirmar

T1

T2

```
1      LEER (X, xi);
2      xi := xi-N;
3      ESCRIBIR (X, xi);
4
5
6
7      LEER (Y, yi);
```

(se genera una excepción en T1)

```
LEER (X, xj);
xj := xj+M;
ESCRIBIR (X, xj);
```

- a. Con un nivel de aislamiento READ UNCOMMITTED se generaría una lectura sucia
- b. Con un nivel de aislamiento READ UNCOMMITTED se generaría una lectura fantasma
- c. Con un nivel de aislamiento READ COMMITTED se generaría una lectura sucia
- d. Con un nivel de aislamiento READ COMMITTED se generaría una lectura fantasma

14. En Oracle, una transacción serializable:

- a. Puede generar una excepción si intenta modificar un dato que ha sido modificado en otra transacción T2, tras comenzar T1
- b. Puede realizar lecturas de datos no actualizadas
- c. Es el mínimo nivel de aislamiento necesario para garantizar que no habrá lecturas sucias
- d. Es el mínimo nivel de aislamiento necesario para garantizar que no habrá lecturas fantasma



3.7 Ejercicios

15. Dadas dos transacciones T1 y T2 y una planificación P, se puede garantizar que:

Tema 6: gestión de transacciones

- a) Si P es serializable, se garantiza que no hay estados conflictivos
- b) Si P es serie, se garantiza que no hay estados conflictivos
- c) Si P2 es una planificación alternativa a P, y P2 es serie, se garantiza que ambas dejarán la base de datos en el mismo estado
- d) Si P es serie entonces es estricta

13. Tomando como ejemplo el siguiente caso escribe planificaciones que sean:

- a) Serie y estricta
- b) Serializable y no serie, recuperable y que evita el borrado en cascada
- c) Serializable y no serie, estricta
- d) No serializable que no evita el borrado en cascada
- e) Serializable y no recuperable

Transacción T1

```
LEER (X, xi);
xi := xi -N;
ESCRIBIR (X, xi);
LEER (Y, yi);
yi := yi +N;
ESCRIBIR (Y, yi);
```

Transacción T2

```
LEER (X, xi);
xi := xi +M;
ESCRIBIR (X);
```



3.7 Ejercicios

Tema 6: gestión de transacciones

17. Un planificador que implemente el protocolo de bloqueo B2F estricto

- a) Garantiza que cualquier planificación será serializable
- b) Garantiza que cualquier planificación será recuperable
- c) Garantiza que no habrá interbloqueos
- d) Garantiza que ninguna transacción quedará en espera indefinida

18. Un planificador que garantice concurrencia, la serializabilidad, esté libre de borrados en cascada, no esté libre de interbloqueos y no conozca los datos que va a modificar cada transacción al inicio de ésta, podría implementar una estrategia:

- a) Serie
- b) Estrategia B2F conservadora
- c) Estrategia B2F estricta
- d) Estrategia B2F rigurosa

19. Sobre el nivel de aislamiento SERIALIZABLE de Oracle podemos afirmar:

- a) Está libre de actualizaciones perdidas
- b) Está libre de borrados en cascada
- c) Está libre de interbloqueos
- d) Sigue una estrategia de bloqueos B2F

20. Sobre el nivel de aislamiento READ COMMITED de Oracle podemos afirmar:

- a) Está libre de actualizaciones perdidas
- b) Está libre de borrados en cascada
- c) Está libre de interbloqueos
- d) Está libre de lecturas sucias

