Base de Datos

Práctica de Transacciones

1.1 Ejercicio

Considerando un modelo simple de LOCK/UNLOCK.

a. Determine si es serializable.
b. En caso afirmativo, proponga un *schedule* serial equivalente.

	T1	T2	T3	T4
1	LOCK A			
2		LOCK C		
3	LOCK B			
4		UNLOCK C		
5	UNLOCK A			
6	UNLOCK B			
7		LOCK B		
8		UNLOCK B		
9			LOCK B	
10			UNLOCK B	
11				LOCK C
12				LOCK D
13				LOCK E
14				UNLOCK C
15			LOCK C	
16				UNLOCK D
17			UNLOCK C	
18				UNLOCK E

1.2 Ejercicio

Dadas las siguientes transacciones T1 y T2. Construya un *schedule* legal serializable que no sea serial.

T1	T2
WLOCK A	RLOCK A
A = A + 1	WLOCK C
WLOCK B	C = A + 1
B = A + B	WLOCK D
UNLOCK A	D = 1
UNLOCK B	UNLOCK A
	UNLOCK D
	UNLOCK C

1.3 Ejercicio

Dadas las siguientes transacciones que deben ser ejecutadas en forma concurrente, indicar un *schedule* legal en el cual se produzca un *dirty read*.

T1	T2
LOCK A	LOCK A
A = A + 2	A = A + 1
LOCK B	LOCK E
B = A + 5	E = A + 8
UNLOCK B	LOCK D
UNLOCK A	UNLOCK E
LOCK C	D = D/5
C = 2 * C	UNLOCK A
UNLOCK C	UNLOCK D

Reescriba las transacciones para que adhieran al protocolo Two Phase Locking (TPL).

1.4 Ejercicio

Considerando el modelo de concurrencia con *read* y *write locks*, demuestre que el protocolo TPL es suficiente pero no necesario para garantizar serializabilidad.

1.5 Ejercicio

Suponiendo que se utiliza el modelo *Read/Write Lock* y dado el siguiente *schedule*, determine si es serializable y, en caso afirmativo, proponga los correspondientes *schedules* seriales equivalentes.

	T1	T2	T3	T4
1		RLOCK A		
2			RLOCK A	
3		WLOCK B		
4		UNLOCK A		
5			WLOCK A	
6		UNLOCK B		
7	RLOCK B			
8			UNLOCK A	
9				RLOCK B
10	RLOCK A			
11				UNLOCK B
12	WLOCK C			
13	UNLOCK A			
14				WLOCK A
15				UNLOCK A
16	UNLOCK B			
17	UNLOCK C			

1.6 Ejercicio

Suponiendo que se utiliza el modelo *Read/Write Lock* y dado el siguiente *schedule*, determine si es serializable y, en caso afirmativo, proponga los correspondientes *schedules* seriales equivalentes.

	T1	T2	T3	T4	T5
1	RLOCK A				
2		RLOCK A			
3			RLOCK A		
4	UNLOCK A				
5					RLOCK B
6					UNLOCK B
7	WLOCK B				
8		WLOCK C			
9	UNLOCK B				
10			RLOCK B		
11			UNLOCK A		
12		UNLOCK A			
13				WLOCK A	
14				UNLOCK A	
15	RLOCK D				
16	UNLOCK D				
17				WLOCK D	
18				UNLOCK D	
19					WLOCK A

	T1	T2	T3	T4	T5
20					UNLOCK A
21		UNLOCK C			
22			UNLOCK B		

1.7 Ejercicio

En el modelo de *LOCKs/UNLOCKs*, suponga una transacción T1 que no cumple con el protocolo TPL. Construya otra transacción T2 - en forma genérica - y un *schedule* S de {T1, T2} tal que S sea legal pero no serializable.

1.8 Ejercicio

Responda verdadero o falso a las siguientes afirmaciones. Justifique.

a. Puede construirse un *schedule* legal no serializable aunque las transacciones cumplan TPL.
b. Puede construirse un *schedule* legal no serializable cuando, por lo menos, una de las transacciones no cumple TPL.
c. No pueden producirse *deadlocks* en un *schedule* legal.
d. Cuando las transacciones cumplen con TPL no hay *dirty reads*.
e. Para manejar concurrencia, la base de datos tiene que estar particionada en ítems.

1.9 Ejercicio

Dadas T_1, T_2 y T_3, 3 transacciones sobre el modelo *RLOCK/WLOCK* y el siguiente *schedule* serializable *S*:

	T_1	T_2	T_3
1			WLOCK B
2			RLOCK A
3	RLOCK A		
4			RLOCK C
5			UNLOCK B
6	RLOCK B		
7	UNLOCK B		
8		WLOCK B	
9			UNLOCK C
10		RLOCK C	
11			UNLOCK A
12	UNLOCK A		
13		WLOCK A	
14		UNLOCK B	
15		UNLOCK A	
16	RLOCK C		
17			RLOCK D
18			UNLOCK D
19	UNLOCK C		
20		WLOCK C	
21		UNLOCK C	

a. Indicar a qué *schedule* serial es equivalente S. Justificar la respuesta.
b. Existe algún *schedule* legal no serializable que se pueda construir con T_1, T_2 y T_3? Justificar la respuesta. En caso afirmativo, mostrar un ejemplo y justificar.
c. Es posible identificar en S un caso en el que pueda producirse un *dirty read*? Justificar la respuesta. En caso afirmativo, mostrar un ejemplo y justificar.

1.10 Ejercicio

Dadas T1, T2 y T3 transacciones sobre el modelo *RLOCK/WLOCK*:

T1: RLOCK A WLOCK B RLOCK C UNLOCK A UNLOCK B UNLOCK C
T2: RLOCK B WLOCK B WLOCK A UNLOCK A UNLOCK B
T3: WLOCK C RLOCK A WLOCK B UNLOCK C UNLOCK B UNLOCK A

a. Construir un *shcedule* legal serializable no serial, equivalente a la siguiente ejecución serial de las transacciones: T2 T3 T1. Justificar la respuesta.
b. Existe algún *schedule* legal no serializable que se pueda construir con T1, T2 y T3? Justifique su respuesta.

1.11 Ejercicio

Dadas las transacciones:

T_1: RL(A); WL(B); UL(A); UL(B)

T_2: RL(A); UL(A); RL(B); UL(B)

T_3: WL(A); UL(A); WL(B); UL(B)

T_4: RL(B); UL(B); WL(A); UL(A)

a) Arme un *schedule* legal y serializable y dé el correspondiente *schedule* serial.

b) Dibuje el Grafo de Precedencia del siguiente *schedule* y determine si es serializable:

S: $WL_3(A)$; $RL_4(B)$; $UL_3(A)$; $RL_1(A)$; $UL_4(B)$; $WL_3(B)$; $RL_2(A)$; $UL_3(B)$;
$WL_1(B)$; $UL_2(A)$; $UL_1(A)$; $WL_4(A)$; $UL_1(B)$; $RL_2(B)$; $UL_4(A)$; $UL_2(B)$

c) Reescriba las transacciones para que adhieran al protocolo TPL.

NOTA:
$RL_i(X)$: READ-LOCK de la transacción T_i sobre el recurso X
$WL_i(X)$: WRITE-LOCK de la transacción T_i sobre el recurso X
$UL_i(X)$: UNLOCK de la transacción T_i sobre el recurso X

1.12 Ejercicio

Dadas las siguientes historias (*schedules*) en el modelo *read/write*:

H1= r1[X] r4[Y] w1[X] w4[X] r3[X] c1 w4[Y] w3[Y] w4[Z] c4 W3[X] c3.
H2= r1[X] w2[X] r1[Y] w1[X] w1[Y] c1 r2[Z] w2[Y] c2.
H3= w1[X] w2[X] w2[Y] c2 w1[Y] c1 w3[X] w3[Y] c3.
H4= w2[X] r1[X] r2[Z] r1[Y] w2[Y] w1[Y] c2 w1[X] c1.
H5= r1[X] r4[Y] r3[X] w1[X] w4[Y] w3[X] c1 w3[Y] w4[Z] c3 W4[X] c4.

a) Indicar para cada una si es NoRC, RC, ACA o ST.
b) Indicar cuáles podrían producir un *dirty read* o un *lost update*.

1.13 Ejercicio

Dada la siguiente historia en el modelo *ReadLock/WriteLock/Unlock*:

H6= rl2[A] rl3[C] wl2[B] ul3[C] ul2[A] wl3[A] ul2[B] rl1[B] ul3[A] rl4[B] rl1[A] ul4[B] wl1[C] ul1[A] wl4[A] ul4[A] ul1[B] ul1[C].

a) Indicar si es Legal.
b) Indicar si es 2PL.
c) Dibujar el Grafo de Precedencia, indicar si es SR y dar las historias seriales equivalentes.

1.14 Ejercicio

a) Dadas las siguientes transacciones en el modelo read/write:

T1= w[B] r[C] w[C] c.
T2= r[D] r[B] w[C] c.

i) dar una historia H1 que no sea RC.
ii) dar una historia H2 que sea ACA pero no ST.
iii) dar una historia H3 que sea ST.

b) Dada la siguiente historia H4, en el modelo ReadLock/WriteLock/Unlock, sobre el conjunto de transacciones {T1, T2, T3, T4, T5} y el conjunto de items {A, B, C, D, E}:

H4 = wl1[A] ul1[A] rl2[A] rl1[B] ul2[A] rl3[A] wl3[D] rl3[C] ul3[A] ul3[D] ul3[C] rl3[B] wl2[C] rl4[D] ul3[B] ul2[C] ul1[B] wl5[B] wl5[C] ul4[D] ul5[C] ul5[B] wl1[E] ul1[E].

i) Indicar si es Legal.
ii) Indicar si es 2PL.
iii) Dibujar el Grafo de Precedencia, indicar si es SR y dar las historias seriales equivalentes.

1.15 Ejercicio

Clasificar según recuperabilidad las siguientes historias:

H1= r1[X] r2[Z] r1[Z] r3[X] r3[Y] w1[X] c1 w3[Y] c3 r2[Y] w2[Z] w2[Y] c2.
H2= r1[X] r2[Z] r1[Z] r3[X] r3[Y] w1[X] w3[Y] r2[Y] w2[Z] w2[Y] c1 c2 c3.
H3= r1[X] r2[Z] r3[X] r1[Z] r2[Y] r3[Y] w1[X] c1 w2[Z] w3[Y] w2[Y] c3 c2.

1.16 Ejercicio

Detalle que ocurre en las siguientes secuencias, el orden es el orden temporal en el que ocurren los eventos. El planificador usa *timestamping* y no es multiversión.

a) st_1 , st_2, r_1 (A), r_2 (B), w_2 (A), w_1(B)
b) st_1 , r_1 (A), st_2, r_2 (B), r_2 (A), w_1(B)
c) st_1 , st_2, st_3, r_1(A), r_2(B), w_1(C), r_3(B), r_3(C), w_2(B),w_3(B)

Notación:

st_1	Inicia transacción T_1
r_1(A)	la transacción T_1 lee el elemento A
w_1(A)	la transacción T_1 escribe el elemento A

R_1 (X) para transacciones con validación indica que inicia la transacción T_1 y su conjunto de lectura es la lista de elementos X.

V_1 la transacción T_1 intenta validar

W_1 (X) la transacción T_1 finaliza y su conjunto de escritura es la lista de elementos X

1.17 Ejercicio

Detalle que ocurre en las siguientes secuencias, el orden es el orden temporal en el que ocurren los eventos. El planificador usa *timestamping* multiversión.

a) st_1 , st_2, st_3, st_4, $w_1(A)$, $w_2(A)$, $w_3(A)$, $r_2(A)$, $r_4(A)$;
b) st_1 , st_2, st_3, st_4, $w_1(A)$, $w_3(A)$, $r_4(A)$, $r_2(A)$;
c) st_1 , st_2, st_3, st_4, $w_1(A)$, $w_4(A)$, $r_3(A)$, $w_2(A)$;
d) st_1 , st_2, st_3, $r_2(C)$, $r_2(B)$, $w_2(B)$, $r_3(B)$, $r_3(C)$, $r_1(X)$, $w_1(X)$, $w_3(B)$, $w_3(C)$, $r_2(A)$,$r_1(B)$,$w_1(B)$,$w_2(A)$

1.18 Ejercicio

Detalle que ocurriría en las secuencias del ejercicio anterior si el planificador no utilizará multiversión.

1.20 Ejercicio

Describa que ocurre en las siguientes secuencias si el planificador está basado en validación

a) $R_1(A,B)$; $R_2(B,C)$; V_1; R_3 (C,D);V_3;W_1 (A);V_2;$W_2(A)$,$W_3(B)$;
b) $R_1(A,B)$; R_2 (B,C); V_1; R_3 (C,D);V_3;W_1 (A);V_2;$W_2(A)$,$W_3(D)$;
c) $R_1(A,B)$; R_2 (B,C); V_1; R_3 (C,D);V_3;W_1 (C);V_2;$W_2(A)$,$W_3(D)$;
d) $R_1(A,B)$; R_2 (B,C); R_3 (C); V_1;V_2;V_3;$W_1(A)$;$W_2(B)$,$W_3(C)$;
e) $R_1(A,B)$; R_2 (B,C); R_3 (C); V_1;V_2;V_3;$W_1(C)$;$W_2(B)$,$W_3(A)$;
f) $R_1(A,B)$; R_2 (B,C); R_3 (C); V_1;V_2;V_3;$W_1(A)$;$W_2(C)$,$W_3(B)$;

1.21 Ejercicio

Dada la siguiente Historia (para planificador basado en validación):

$R_1(A,B)$; $R_2(B,F)$; V_2; V_1; $R_3(B,D)$; $W_2(D)$; V_3; $W_1(A,C)$; $W_3(D,E)$

a) Indique qué ocurre en cada momento de validación y en caso de no validar cuál es el problema preciso que presenta.

b) Realice un cambio en la Historia. Como resultado de ese cambio todas las validaciones deben ser exitosas. Justifique.

1.22 Ejercicio

Dada la siguiente historia en un planificador con *timestamp*

st_1, st_2, $r_2(X)$, st_3, st_4, $r_1(Y)$, $r_4(Z)$, $w_3(X)$, $w_3(Y)$, $w_4(Z)$, $w_2(X)$, $w_1(Y)$, $r_3(Z)$

en donde los valores iniciales de X, Y, Z son 0 y pasa que:
t_1 escribe Y = 1
t_2 escribe X=2
t_3 escribe X=3, Y=30, Z=300

t_4 escribe Z = 4

a) Decir que pasa en cada acción y que valores quedan en X,Y,Z si el planificador no usa *multiversión*
b) Decir que pasa en cada acción X,Y,Z y que valores se tendrían para X,Y,Z si el planificador usa *multiversión*

1.23 Ejercicio

Suponga que se crea una tabla con la siguiente sentencia:

```
CREATE TABLE Alumnos(idAlumno  INT NOT NULL PRIMARY KEY, nombre
                                VARCHAR(90));
```

Luego se ejecutan las siguientes sentencias SQL:

```
INSERT INTO Alumnos (idAlumno, nombre) VALUES (1, 'Jhon Doe');
INSERT INTO Alumnos (idAlumno, nombre) VALUES (1, 'Don Nadie);
ROLLBACK;
SELECT * FROM Alumnos;
```

Diga que ocurre con la ejecución si:

a) Están habilitadas las transacciones implícitas
b) No están habilitadas las transacciones implícitas

1.24 Ejercicio

En la misma estructura que el ejercicio anterior se ejecutan las siguientes sentencias SQL

```
START TRANSACTION;
INSERT INTO Alumnos (idAlumno, nombre) VALUES (4, 'cuatro');
SELECT * FROM Alumnos ;
ROLLBACK;
SELECT * FROM Alumnos;
```

Describa que ocurre.

1.25 Ejercicio

Considere dos clientes diferentes del motor de base de datos trabajando concurrentemente. Se realiza la siguiente secuencia (el orden de arriba abajo es el orden temporal), la tabla Alumnos tiene la previamente la tupla: (101, 'Horacio')

Cliente1:

```
START TRANSACTION;
SELECT NOMBRE  FROM Alumnos WHERE idAlumno = 101 ;
```

Cliente2:

```
START TRANSACTION;
SELECT NOMBRE  FROM Alumnos WHERE idAlumno = 101;
UPDATE Alumnos SET Nombre= 'Pepe'  WHERE idAlumno = 101;
COMMIT;
```

Cliente1:

```
SELECT NOMBRE  FROM Alumnos WHERE idAlumno = 101 ;
COMMIT;
```

Describa que ocurre con cada SELECT en el caso de que el nivel de aislamiento de las transacciones sea:

a) REPEATABLE READ
b) READ COMMITTED