2nd LAB SESSION ON TRANSACTIONS

Given Name: Miquel **Family name:** Perelló Rodríguez
Given Name: Laia **Family name:** Ondoño Pujol

1) (40%) Consider the No Steal / Force policy:

a) Provide the pseudo code of the *read*, *write*, *commit* and *abort* operations, so that we guarantee recoverability in **case of power failure**. Use as basis those in pages 19 and 20 for the steal / no force policy.

```
procedure read(t: transaction_id, p: page_id, v: page_value)
  if search(p) = 0 then
    fetch(p);
    pin(p) := 1;
  endIf
  v := content(p);
endProcedure
```

```
procedure write(t: transaction_id, p: page_id, v: page_value)
  var w: page_value;
  if search(p) = 0 then
    fetch(p);
    pin(p) := 1;
  endIf
  w := content(p);
  write_log('u', t, p, w);
  content(p) := v;
  dirty(p) := 1;
endProcedure
```

```
procedure commit(t: transaction_id)
  for p in t: //per a cada pagina amb id p que t té al buffer pool
    pin(p) := 0;
    flush(p);
  write_log('c', t);
endProcedure
```

```
procedure abort(t: transaction_id)
  for p in t: //per a cada pagina amb id p que t té al buffer pool
    pin(p) := 0;
  write_log('a', t);
endProcedure
```

b) Under what circumstances that policy may be interesting (e.g., What are its **cons and pros**? **What kind of systems** can you think of that would suit it?)

Pros:
- Fàcil recuperació de les dades.

Cons:
- Ineficiència quan es fan commits.
- Locking de transaccions.

- Problema amb la memoria (molts pins)

Funcionaria en un sistema on es fessin poques escriptures, ja que la Force policy obliga a que, cada canvi al data object, ha de ser escrit a disc, el que genera un nombre elevat d'I/O a aquest.

2) (30%) Given a DBMS without any concurrency control mechanism, let's suppose that we have the following history (actions have been numbered just to facilitate referencing them):

#Acc	T1	T2	T3
10			BoT
20		BoT	
30	BoT		
40		R(E)	
50	R(A)		
60	W(A)		
70			R(A)
80			W(A)
90	R(F)		
100	R(D)		
110	R(E)		
120	W(E)		
130		R(C)	
140		W(C)	
150		R(E)	
160			R(F)
170			W(F)
180		COMMIT	
190	COMMIT		
200			COMMIT

Let's suppose now that the DBMS is based on an **optimistic technique** that validates readings at commit time. How would result the same history? **Is any transaction cancelled?**

- Mirant el contingut de RS i WS a cada acció:
  10 -> RS(T3) = Ø, WS(T3) = Ø
  20-> RS(T2) = Ø, WS(T2) = Ø
  30-> RS(T1) = Ø, WS(T1) = Ø
  40-> RS(T2) = {E}
  50-> RS(T1) = {A}
  60-> WS(T1) = {A}
  …

  Un cop executada l'acció 170, obtenim el següent:
  RS(T1) = {A, F, D, E} WS(T1) = {A, E}
  RS(T2) = {E, C} WS(T2) = {C}
  RS(T3) = {A ,F} WS(T3) ={A, F}

Un cop **T2 fa commit**, s'afegeix aquesta al set of committed transactions. Per tant, tenim que:

RS(T1) = {A, F, D, E} WS(T1) = {A, E} setOfCommited(T1) = {T2}

RS(T2) = {E, C} WS(T2) = {C}

RS(T3) = {A, F} WS(T3) ={A, F} setOfCommited(T3) = {T2}

Quan T1 vol fer commit, cal mirar si hi ha algun conflicte amb alguna transacció que ja ha finalitzat exitosament. Per això, mirem si el resultat de la següent intersecció: RS(T1) ∩ WS(T2) és un conjunt buit. Tenim que:

RS(T1) ∩ WS(T2) = Ø

i, per tant, **T1 fa commit**. Passem a tenim això:

RS(T1) = {A, F, D, E} WS(T1) = {A, E} setOfCommited(T1) = {T2}

RS(T2) = {E, C} WS(T2) = {C}

RS(T3) = {A, F} WS(T3) ={A, F} setOfCommited(T3) = {T2, T1}

Finalment, repetim el pas anterior per a T3, però en aquest cas hem de tenir en compte 2 transaccions finalitzades: T2 i T1. Com que:

RS(T3) ∩ WS(T1) = {A}

hi ha conflicte amb el gränul A i, per tant, **T3 es cancel·la**.

3) (30%) Given a DBMS without any concurrency control mechanism, let's suppose that we have the following history (actions have been numbered just to facilitate referencing them):

#Acc	T1	T2	T3
10			BoT
20		BoT	
30	BoT		
40		R(E)	
50	R(A)		
60	W(A)		
70			R(A)
80			W(A)
90	R(F)		
100	R(D)		
110	R(E)		
120	W(E)		
130		R(C)	
140		W(C)	
150		R(E)	
160			R(F)
170			W(F)
180		COMMIT	
190	COMMIT		
200			COMMIT

Let's suppose now that the DBMS is based on a **dynamic timestamping** technique. How would result the same history? **Is any transaction cancelled?**

- Mirant el contingut de RS i WS a cada acció:
  10->
  20 ->
  30 ->
  40 -> RS(T2) = {E}
  50 -> RS(T1) = {A}
  60-> WS(T1) = {A}
  ...

  A l'acció número 70 trobem un possible conflicte entre dues transaccions (T1 i T3), i cal que assignem un timestamp dinàmicament. T3 vol llegir el gränul A que T1 ha modificat prèviament. Com que tant TS(T1) com TS(T3) tenen valors nuls, s'assignen valors que compleixin amb TS(T1) < TS(T3). Per tant, assignem:
  70-> TS(T1)=1, TS(T3)=2

  Si passem a l'acció 150, tornem a tenir el mateix escenari però, en aquest cas, T2 vol llegir un valor que T1 ha modificat (gränul E). TS(T1) no té valor null, però TS(T2) si, se li assigna un valor tal que TS(T1) < TS(T2). Finalment, obtenim que:
  150-> TS(T1) = 1, TS(T2) = 3

  Com que sempre es compleix que TS(T) > TS(Ti) on T és la transacció que està executant una acció en la unitat de temps en què es comprova això i Ti són la resta de transaccions actives, **no es cancel·la cap transacció**.