2nd LAB SESSION ON TRANSACTIONS

Given Name: Marc **Family name:** Aparicio Arbusà

Given Name: Anna **Family name:** Llanza Carmona

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
```

És el mateix cas que el No Steal/No Force policy. Per tant, si es realitza un fetch de la pàgina, caldrà posar el pin = 1.

```
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
```

És el mateix cas que el No Steal/No Force policy. Per tant, si es realitza un fetch de la pàgina, caldrà posar el pin = 1 i no fa falta guardar la before image (w). A més a més, tampoc és necessari guardar l'after image (v), ja que no cal realitzar cap redo.

```
procedure commit(t: transaction_id)
        var p: page_id;
        pin(p) := 0;
        flush(p);
        write_log('c', t);
endProcedure
```

```
procedure flush(p: page_id)
        var w: page_value;
         if dirty(p) = 1 then
                  w := content(p);
                 write_page (p, w);
                  dirty(p) := 0;
        endIf
        update_directory(p, 0);
endProcedure
```

És el mateix cas que el Steal/No Force policy, però també és necessari fer unpin (pin = 0) de la pàgina i flush del buffer slots. També hem afegit la implementació de la funció flush().

```
procedure abort(t: transaction_id)
        var p: page_id;
        w: page_value;

        read_log_backwards('u', t, p, w);
         while records_remain_in_log ('u', t) do
                  content(p) := w
                  pin(p) := 0;
                  read_log_backwards('u', t, p, w);
        endWhile
        write_log('a', t);
endProcedure
```

És el mateix que el No Steal/ No Force policy, per tant al no ser necessari fer cap undone, només cal posar el pin a 0 i escriure la corresponent entrada al log

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?)

La política de No Steal/ Force no necesita fer undo o redo en cap circunstancia, si hi ha algun problema o fer algun abort és més fàcil de recuperar la situació on estem, pero per altre banda per cada pàgina que es llegeix i/o s'escriu s'ha de fer pin de la pàgina i això pot generar un problema de memoria, ja que en cas de tenir moltes pàgines podriem acabar emplenant tot els buffer slots.

Un dels possibles sistemes on es podria utilitzar aquesta política seria en el cas de disposar d'una memòria suficientment gran per així poder evitar aquest problema. Una altre possibilitat seria sistemes on es realitzin molts pocs accessos a diferents pàgines, és a dir, es podria realitzar sense cap problema molts accessos a una mateixa pàgina, ja que al haver fet el pin d'aquesta, no variaria l'espai a memòria.

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?**

- T2 llegeix E, per tant afegim E a la llista de lectures de T2 RS(T2)= {E}
- T1 llegeix A, per tant afegim A a la llista de lectures de T1 RS(T1)= {A}
- T1 escriu A, per tant afegim A a la llista de escriptures de T1 WS(T1)= {A}
- T3 llegeix A, per tant afegim A a la llista de lectures de T3 RS(T3)= {A}
- T3 escriu A, per tant afegim A a la llista de escriptures de T3 WS(T3)= {A}
- T1 llegeix F, per tant afegim F a la llista de lectures de T1 RS(T1)= {A,F}
- T1 llegeix D, per tant afegim D a la llista de lectures de T1 RS(T1)= {A,F,D}
- T1 llegeix E, per tant afegim E a la llista de lectures de T1 RS(T1)= {A,F,D,E}
- T1 escriu E, per tant afegim E a la llista de escriptures de T1 WS(T1)= {A,E}
- T2 llegeix C, per tant afegim C a la llista de lectures de T2 RS(T2)= {E,C}
- T2 escriu C, per tant afegim C a la llista de escriptures de T2 WS(T2)= {C}
- T2 llegeix E, per tant afegim E a la llista de lectures de T2 RS(T2)= {E,C}
- T3 llegeix F, per tant afegim F a la llista de lectures de T3 RS(T3)= {A,F}
- T3 escriu F, per tant afegim F a la llista de escriptures de T3 WS(T3)= {A,F}
- T2 vol fer commit, per tant fem la validació i com que el setOfCommitedTx(T2)={} complim totes les condicions per a poder fer el commit. A més afegim T2 a la llista de setOfCommitedTx(T1)={T2} i setOfCommitedTx(T3)={T2}
- T1 vol fer commit, per tant fem la validació i com que el setOfCommitedTx(T1)={T2} conté T2 hem de comprovar que els lectures de T1 no tinguin cap granul que coincideix amb les escriptures de T2. Com no es dona el cas, no hem de cancel·lar la transacció i per tant realitzem el commit, A més afegim T1 a la llista de T3 setOfCommitedTx(T3)={T2,T1}
- T3 vol fer commit, per tant fem la validació i com que el setOfCommitedTx(T3)={T2,T1} conté T2 i T1 hem de comprovar que els lectures de T3 no tinguin cap granul que coincideix amb les

escriptures de T2 o T1. Com que les escriptures de la T1 conté el granul A i les lectures de T3 també, la validació no es compleix i per tant hem de cancel·lar la transacció.

Després de realitzar totes les accions, el resultat final de les llistes seria el següent:

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

WS(T1)= {A,E}

setOfCommitedTx(T1)={T2}

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

WS(T2)= {C}

setOfCommitedTx(T2)={T1}

RS(T3)= {A,F}

WS(T3)= {A,F}

setOfCommitedTx(T3)={T2,T1}

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?**

- T2 llegeix E, per tant afegim E a la llista de lectures de T2 RS(T2)= {E}
- T1 llegeix A, per tant afegim A a la llista de lectures de T1 RS(T1)= {A}
- T1 escriu A, per tant afegim A a la llista de escriptures de T1 WS(T1)= {A}
- T3 llegeix A, i com que T1 també ha llegit aquest granul, entrem en conflicte i s'inicialitza el TS(T1) = 1 i TS(T3) = 2. Com que es pot realitzar la lectura, actualitzem TSR(A)=1 i afegim A a la llista de lectures de T3 RS(T3)= {A}
- T3 escriu A, hem de comprovar la condició d'escriptura, com compleix la condició del granul A es realitza l'escriptura, llavors actualitzem TSW(A)=2 i afegim A a la llista de escriptures de T3 WS(T3)= {A}
- T1 llegeix F, per tant afegim F a la llista de lectures de T1 RS(T1)= {A,F} i com que es compleix la condició de lectura actualitzem TSR(F)=1
- T1 llegeix D, per tant afegim D a la llista de lectures de T1 RS(T1)= {A,F,D} i com que es compleix la condició de lectura actualitzem TSR(D)=1
- T1 llegeix E, per tant afegim D a la llista de lectures de T1 RS(T1)= {A,F,D,E} i com que es compleix la condició de lectura actualitzem TSR(E)=1
- T1 escriu E, i com que T1 entra en conflicte amb T2, cal inicializar TS(T2) = 3. També hem de comprovar la condició d'escriptura, com compleix la condició del granul E es realitza l'escriptura, llavors actualitzem TSW(E)=1 i afegim E a la llista de escriptures de T1 WS(T1)= {A,E}
- T2 llegeix C, per tant afegim C a la llista de lectures de T2 RS(T2)={E,C} i com que es compleix la condició de lectura actualitzem TSR(C)=3
- T2 escriu C, hem de comprovar la condició d'escriptura, com compleix la condició del granul C es realitza l'escriptura, llavors actualitzem TSW(C)=3 i afegim C a la llista de escriptures de T2 WS(T2)= {C}
- T2 llegeix E, com que la llista de lectures de T2, ja conté el gränul E, no cal tornar-lo a afegir. A més com que es compleix la condició de lectura actualitzem TSR(C)=3
- T3 llegeix F, com que la llista de lectures de T3, ja conté el gränul F, no cal tornar-lo a afegir i com que es compleix la condició de lectura actualitzem TSR(F)= 2
- T3 escriu F, hem de comprovar la condició d'escriptura, com compleix la condició del granul F es realitza l'escriptura, llavors actualitzem TSW(F)=2 i afegim F a la llista de escriptures de T3 WS(T3)= {A,F}
- T2 com no hi ha cap conflicte, realitza el commit
- T1 com no hi ha cap conflicte, realitza el commit
- T3 com no hi ha cap conflicte, realitza el commit

Després de realitzar totes les accions, el resultat final de les llistes seria el següent:

TS(T1) = 1 TS(T2) = 3 TS(T3) = 2

TSR(A)=2 TSR(F)=2 TSR(D)=1 TSR(E)=3 TSR(C)=3

TSW(A)=2 TSW(F)=2 TSW(D)=0 TSW(E)=1 TSW(C)=3

RS(T1)={A,F,D,E} RS(T2)={E,C} RS(T3)={A,F}

WS(T1)={A,E} WS(T2)={C} WS(T3)={A,F}