CS1555/CS2055 Recitation 14

Objective: Practice B+ Tree, Concurrency Control, Linear Hashing

---

Part 1: B+ Tree

1. Build the B+ Tree maintaining the index on the name of students (n=3), with the following items: Alex, Christine, Bella, Mary, Peter, Dave.

2. Then do the followings:
   a) Add "Nancy" to the tree
   b) Delete "Mary" from the tree.
   c) Delete "Peter" from the tree.

3. Build a B+ tree for n=4 for the following keys (2, 3, 4, 5, 14, 10, 6, 25, 13, 30, 1)

Part 2: Concurrency Control

1. Consider the following two transactions:

T1:	r1(A) ;	T2:	r2(C);
	r1(B);		r2(B);
	If A=0 then B:= B+1;		r2(A);
	w1(B);		if B>C then {A:= A+1; C:=C+1;}
			w2(C)
			w2(A)

- For each of the following histories/schedules:
  a) Is it a valid history?
  b) Use *serializability graphs* to check whether it is serializable or not, and if it is, what is the equivalent serial history/schedule

  H1: r1(A) r1(B) r2(C) w1(B) r2(B) r2(A)w2(C) w2(A)
  H2: r1(A) r1(B) r2(C) r2(B) w1(B) r2(A)w2(C) w2(A)

2. Consider the following two transactions:

T1:		T2:	
	r1(A) ;		r2(A);
	A:=A+100;		A:=A*2;
	w1(A);		w2(A);
	r1(B);		r2(B);
	B:=B+100;		B:=B*2;
	w1(B)		w2(B)

- For each of the following histories/schedules check:
  a) Is it a serializable history?
  b) What histories are conflict equivalent?

  H1: r1(A), w1(A), r1(B), w1(B), r2(A), w2(A), r2(B), w2(B)
  H2: r1(A), w1(A), r2(A), w2(A), r1(B), w1(B), r2(B), w2(B)
  H3: r1(A), w1(A), r2(A), w2(A), r2(B), w2(B), r1(B), w1(B)
  H4: r2(A), w2(A), r1(A), r2(B), w1(A), w2(B), r1(B), w1(B)
  H5: r1(A), w1(A), r1(B), w1(B), r2(A), w2(A), w2(B), r2(B)

3. Consider the following history, with lock and unlock statements added for each transaction:

T1	T2
rl1(A)	
r1(A)	
rl1(B)	
r1(B)	
	rl2(C)
	r2(C)
	rl2(B)
	r2(B)
wl1(B)	
w1(B)	
	rl2(A)
	r2(A)
	wl2(C)
	w2(C)
	wl2(A)
	w2(A)
unlock1(A), unlock1(B)	
commit	
	unlock2(C), unlock2(A) unlock2(B)
	commit

a) Does the history follow 2PL protocol?
b) Did deadlock happen?

Part 3: Linear Hashing

1) **<u>Dynamic hashing:</u>**
   a) **<u>Linear hashing:</u>**
      - Allows a hash file to expend and shrink the number of buckets dynamically *without* the use of a directory
      - **Initial hash function:** $h_0(K)$ =K mod M, where K is the key and M is the number of buckets available
      - **s** is the number of buckets (initially s = M)
      - **BLast** and **BSplit** pointers are recorded. BLast is a pointer to the current last bucket (initially BLast=M-1). BSplit is a pointer to the bucket that should be split next (initially BSplit =0).
      - **bfr** is the blocking factor (number of records per block).
      - When inserting a new tuple, if there is a collision, push tuple to an overflow bucket.
      - If there are no overflow buckets available, proceed as follows until one becomes available:
         - A new bucket is appended at the end of the hash table and BLast" = BLast+1
         - Records in BSplit bucket are hashed again using h1(key) = key mod (2*M)
            - these will either remain in BSplit or stored in BLast"
            - this action may free an overflow bucket.!
         - BSplit becomes BSplit +1.
      - if (BLast = 2s-1) then
         - set s = 2s
         - BLast = s-1
         - BSplit =0,
         - h0(key)=h1(key)

1. Consider the following record keys: (3, 2, 1, 8, 6, 4, 14, 5, 9). Create the linear hashing structure for the above records after each split assuming $h_0(k)$ = k mod 4 , bfr=2 and:

a) no overflow buckets.

b) 1 overflow bucket (add two additional values: 10, 17).