INF3100: Databasesystemer – Oppgavesett 8

Oppgave 18.2.1: Below are two transactions, described in terms of their effect on two database elements A and B, which we may assume are integers.

$T_1 : READ(A,t); t := t + 2; WRITE(A,t); READ(B,t); t := t * 3; WRITE(B,t);$
$T_2 : READ(B,s); s := s * 2; WRITE(B,s); READ(A,s); s := s + 3; WRITE(A.s);$

We assume that, whatever consistency constraints there are on the database, these transactions preserve them in isolation. Note that $A = B$ is not the consistency constraint.

a) Give examples of a serializable schedule and a nonserializable schedule of the 12 actions above.

d) It turns out that both serial orders have the same effect on the database, that is, (T_1, T_2) and (T_2, T_1) are equivalent. Demonstrate this fact by showing the effect of the two transactions on an arbitrary initial database state.

Oppgave 18.2.5: For each of the following schedules,

a) $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C);$

d) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C);$

answer the following questions:

(i) What is the precedence graph for the schedule?

(ii) Is the schedule conflict-serializable? If so, what are all the equivalent serial schedules?

(iii) Are there any serial schedules that must be equivalent (regardless of what the transactions do to the data), but are not conflict-equivalent?

Oppgave 18.3.2: For each of the schedules of Exercise 18.2.5, assume that each transaction takes a lock on each database element immediately before it reads or writes the element, and that each transaction releases its locks immediately after the last time it accesses an element. Tell what the locking scheduler would do with each of these schedules; i.e., what requests would get delayed, and when would they be allowed to resume?

a) $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C);$

d) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C);$

Oppgave 18.4.1: For each of the schedules of transactions T_1, T_2, and T_3 below,

b) $r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(D); w_1(C); w_2(D); w_3(E);$

c) $r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(A); w_1(A); w_2(B); w_3(C);$

do each of the following:

(i) Insert shared and exclusive locks, and insert unlock actions. Place a shared lock immediately in front of each read action that is not followed by a write action of the same element by the same transaction. Place an exclusive lock in front of every other read or write action. Place the necessary unlocks at the end of every transaction.

(ii) Tell what happens when each schedule is run by a scheduler that supports shared and exclusive locks.

(iii) Insert shared and exclusive locks in a way that allows upgrading. Place a shared lock in front of every read, an exclusive lock in front of every write, and place the necessary unlocks at the ends of the transactions.

(iv) Tell what happens when each schedule from (iii) is run by a scheduler that supports shared locks, exclusive locks, and upgrading.

(v) Insert shared, exclusive, and update locks, along with unlock actions. Place a shared lock in front of every read action that is not going to be upgraded, place an update lock in front of every read action that will be upgraded, and place an exclusive lock in front of every write action. Place unlocks at the ends of transactions, as usual.

(vi) Tell what happens when each schedule from (v) is run by a scheduler that supports shared, exclusive, and update locks.

Oppgave 18.7.1: Suppose we perform the following actions on the B-tree of Fig. 14.13 below. If we use the tree protocol, when can we release a write-lock on each of the nodes searched?

(a) Insert 4 (b) Insert 30 (c) Delete 37 (d) Delete 7.

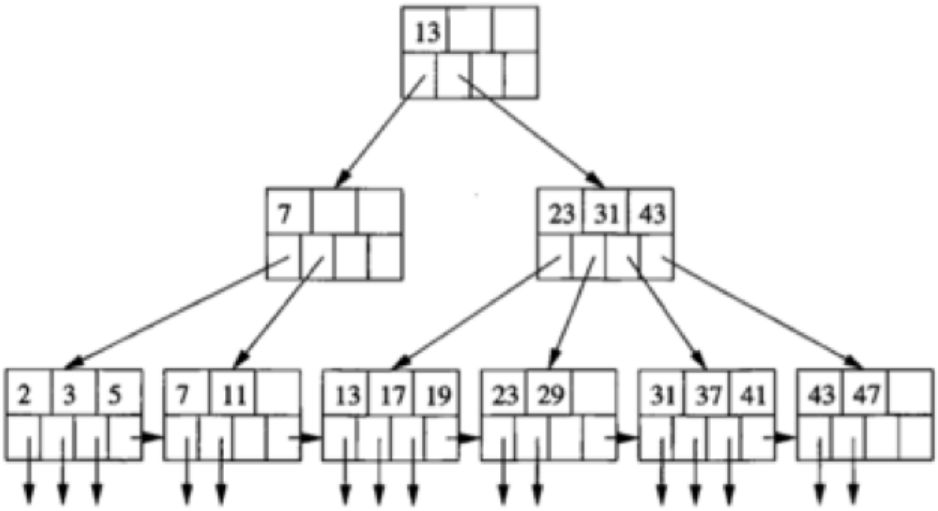

Figure 14.13: A B-tree

Figure 14.13 shows an entire three-level B-tree, with $n = 3$. We have assumed that the data file consists of records whose keys are all the primes from 2 to 47. Notice that at the leaves, each of these keys appears once, in order. All leaf blocks have two or three key-pointer pairs, plus a pointer to the next leaf in sequence. The keys are in sorted order as we look across the leaves from left to right.

The root has only two pointers, the minimum possible number, althought it could have up to four. The only key at the root separates those keys reachable via the first pointer from those reachable via the second. That is, keys up to 12 could be found in the first subtree of the root, and keys 13 and up are in the second subtree.