University of Edinburgh
School of Informatics
INFR11199 - Advanced Database Systems (Spring 2024)

Tutorial Sheet 2

1. (Files, Pages, Records) Consider the following relation:

```
CREATE TABLE Customer (
  customer_id INTEGER PRIMARY KEY,    -- cannot be NULL!
  age INTEGER NOT NULL,
  name VARCHAR(10) NOT NULL,
  address VARCHAR(20) NOT NULL
);
```

Assume that `INTEGER`s are 4 bytes long and `VARCHAR(n)` can be up to n bytes long.

(a) As the records are variable length, we will need a *record header* in the record. How big is the record header? You may assume pointers are 4 bytes long, and that the record header only contains pointers for variable-length values.

> **Solution:** 8 bytes. In the record header, we need one pointer for each variable length value. In this schema, those are just the two VARCHARs, so we need 2 pointers, each 4 bytes.

(b) Including the record header, what is the smallest possible record size (in bytes) in this schema? Note: `NULL` is treated as a special value by SQL, and an empty string `VARCHAR` is different from `NULL`, just like how a 0 `INTEGER` value is also different from `NULL`.

> **Solution:** 16 bytes (= 8 + 4 + 4 + 0 + 0)
> 8 for the record header, 4 for each of the integers, and 0 for each of the `VARCHAR`s.

(c) Including the record header, what is the largest possible record size (in bytes) in this schema?

Solution: 46 bytes (= 8 + 4 + 4 + 10 + 20)

(d) Suppose we are storing these records using a slotted page layout with variable length records. The page header contains an integer storing the record count and a pointer to free space, as well as a slot directory storing, for each record, a pointer and length. What is the *maximum* number of records that we can fit on a 8KB page?

Solution: 341 records (= (8192 - 4 - 4) / (16 + 4 + 4))
We start out with 8192 bytes of space on the page. We subtract 4 bytes that are used for the record count, and another 4 for the pointer to free space. This leaves us with $8192 - 4 - 4$ bytes that we can use to store records and their slots. A record takes up 16 bytes of space at minimum (from the previous questions), and for each record we also need to store a slot with a pointer (4 bytes) and a length (4 bytes). Thus, we need $16 + 4 + 4$ bytes of space for each record and its slot.

(e) Suppose we stored the maximum number of records on a page, and then deleted one record. Now we want to insert another record. Are we guaranteed to be able to do this? Explain why or why not.

Solution: No, we deleted 16 bytes but the record we want to insert may be up to 46 bytes.

(f) Now suppose we deleted 3 records. Without reorganizing any of the records on the page, we would like to insert another record. Are we guaranteed to be able to do this? Explain why or why not.

Solution: No, there are 48 free bytes but they may be fragmented – there might not be 46 contiguous bytes.

2. (Buffer Management) We are given a buffer pool with 4 pages, which is empty to begin with. Answer the following questions given this access pattern:

A B C D E B A D C A E C

(a) What is the hit rate for MRU?

Solution: 4/12.
A (miss), B (miss), C (miss), D (miss), E (miss, D out), B (hit), A (hit), D (miss, A out), C (hit), A (miss, C out), E (hit), C (miss, E out)

(b) For MRU, which pages are in the buffer pool after this sequence of accesses?

Solution: D, B, A, C.

(c) What is the hit rate for clock? Assume the clock hand initially points at the first frame.

Solution: 5/12.
after ABCD: A(1), B(1), C(1), D(1), hand=A
after E: E(1), B(0), C(0), D(0), hand=B
after B: E(1), B(1), C(0), D(0), hand=B, HIT
after A: E(1), B(0), A(1), D(0), hand=D
after D: E(1), B(0), A(1), D(1), hand=D, HIT
after C: E(0), C(1), A(1), D(0), hand=A
after A: E(0), C(1), A(1), D(0), hand=A, HIT
after E: E(1), C(1), A(1), D(0), hand=A, HIT
after C: E(1), C(1), A(1), D(0), hand=A, HIT

(d) For clock, which pages are in the buffer pool after this sequence of accesses?

Solution: A, C, D, E.

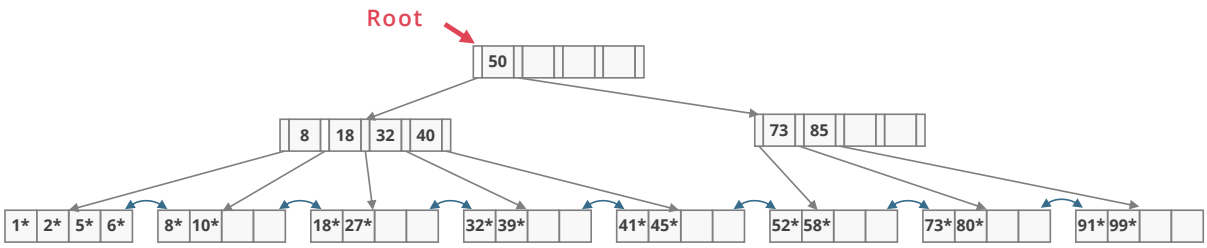
(e) Which pages have their reference bits set?

Solution: A, C, E.

(f) Which page is the hand of the clock pointing to?

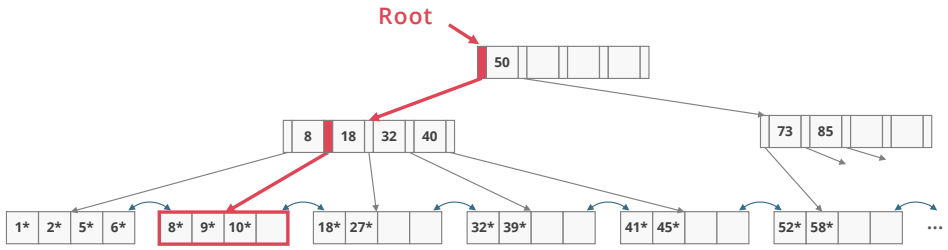
Solution: A. On a page hit, the clock hand does not move.

3. (B+ Tree) Consider the following B+ tree index of order $d = 2$:



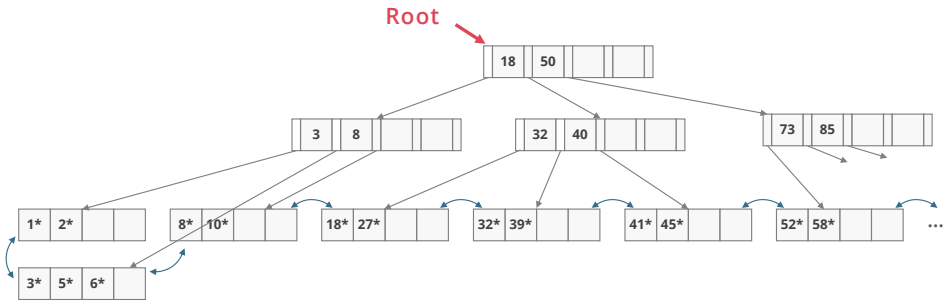
(a) Show the B+ tree that would result from inserting an index entry with key 9 into this tree.

Solution: The index entry with key 9 is inserted on the second leaf page. The resulting tree is shown below:



(b) Show the B+ tree that would result from inserting an index entry with key 3 *into the original tree*. Assume no redistribution between siblings. How many page reads and page writes does the insertion require? Justify your answer.

Solution: The index entry with key 3 goes on the first leaf page. As this page can accommodate at most four index entries ($d = 2$), the page is split. The lowest key of the new leaf is copied to the ancestor, which is also full and needs to be split. The middle key of the ancestor (after adding key 3, that would be key 18) is pushed to the root. The result is shown below:

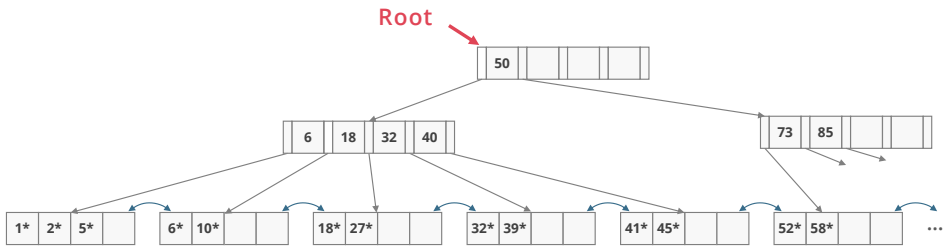


The insertion requires 4 page reads, 6 page writes, and allocation of 2 new pages: read the root, read its left child, read the leftmost leaf, create a new leaf, write the leftmost leaf, write the new leaf, read and write the leaf containing keys 8 and 10 (to update its left pointer), create a new inner page, write the root's left child, write the new inner page, write the root.

(c) Show the B+ tree that would result from deleting the index entry with key 8 *from the original tree*, assuming that the left sibling is checked for possible redistribution.

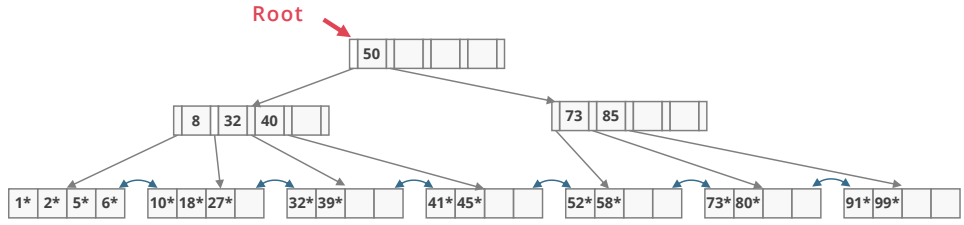
Solution: The index entry with key 8 is deleted, resulting in a leaf page with fewer than two index entries. The left sibling is checked for redistribution. Since the sibling has more than two index entries, the remaining

keys are redistributed between the two leaves. The resulting tree is shown below:

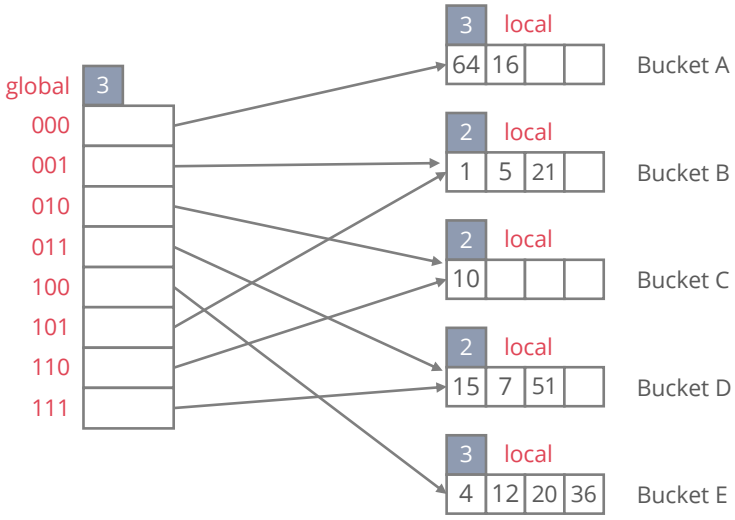


(d) Show the B+ tree that would result from deleting the index entry with key 8 *from the original tree*, assuming that the right sibling is checked for possible redistribution.

Solution: The index entry with key 8 is deleted from the corresponding leaf page. The right sibling has the minimum number of keys, therefore the two siblings merge. The key in the ancestor which distinguishes between the newly merged leaves is deleted. The resulting tree is shown below:

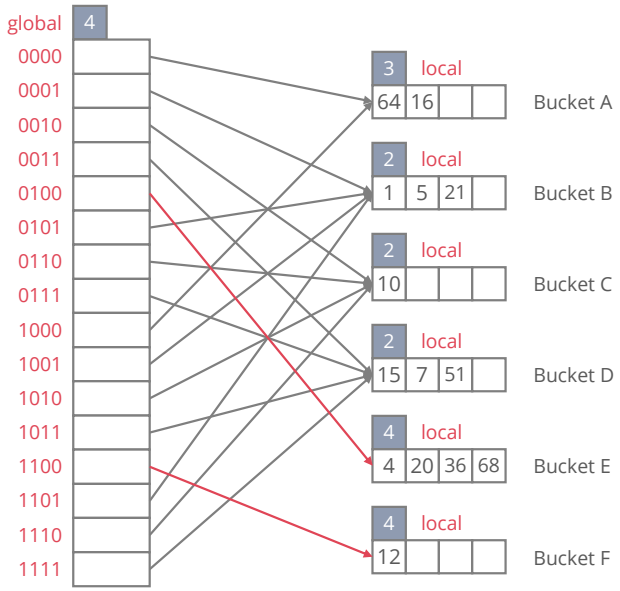


4. (Extendible Hashing) Consider the following Extendible Hashing index:



(a) Draw the index after inserting an entry with hash value 68.

Solution: After inserting an entry with hash value 68:



(b) Draw the index after inserting entries with hash values 17 and 69 into the original index.

Solution: After inserting entries with hash values 17 and 69: