**Vitamin 4 Solutions**

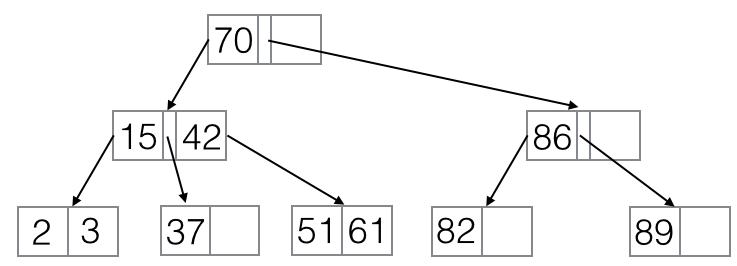
Q1: Suppose that all nodes in our B+ tree have an order of 1605. What's the MAXIMUM number of records we can index with a B+ tree of height 4?

(3210 pointers/page + 1 pointer) ^**4** \* (3210 leaf pointers) = **341244966965525610** records (a lot)

Q2: We want to bulk-load a B+ tree, and we reduce the fill factor of this bulk load. Which of the following applies, in general?

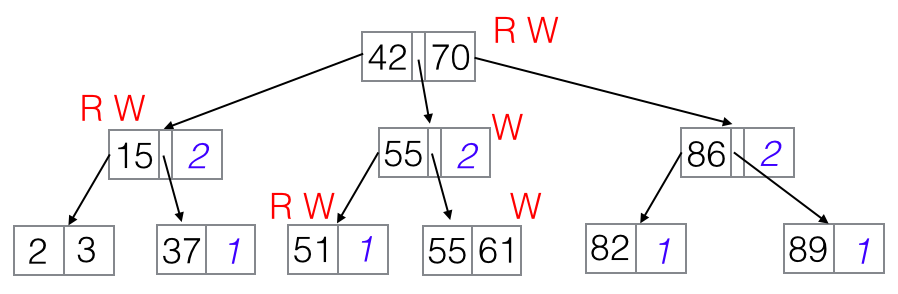
The key fact is that a smaller fill factor means each page will hold fewer records on the initial bulk load. This means we’ll have to write more pages to disk, which means (**1**) our bulk loads take more time to complete, (**2**) pages consume more space on disk, and (**3**) the tree is (in general) deeper because we’ll have more leaf pages, requiring us to perform more I/Os to get to any one page. The upshot is that since we’ve “pre-allocated” pages for additional records, insertions should trigger **fewer disk writes**.

Original tree:



Q3: We insert 55 into the B+ tree in figure A. How many I/Os (page reads and writes) does this operation take?

After inserting 55:

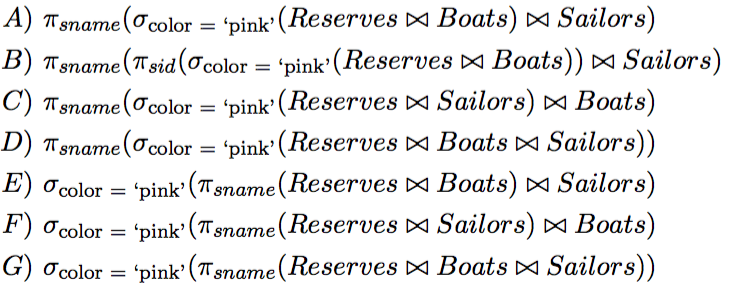


tl;dr **8** read and write ops (3 read, 5 write)

1. **Read** the root [70]. 55 < 70, so follow the leftmost pointer and
2. **read** the page containing [15, 42]. 55 > 42, so follow the rightmost pointer and
3. **read** the page containing [51, 61]. We’re at the leaf, so we can perform an insertion here. However, our leaf page is full! We therefore have to split it on the key 55, so
4. we create a page containing [55, 61], and we **write** it to disk. (Note that we’re at a leaf page, so we must copy 55 to its parent.) We also
5. delete 61 from the page containing [51, 61] and **write** that to disk ([51]). Now we recursively insert 55 into our parent [15, 24], which is full, so we need to split it on the key 42, and
6. we create a page containing [55], setting its leftmost pointer to be the page containing [51] and its rightmost pointer to be the page containing [55, 61]. (Note that since this is an internal node, we don’t need to keep 42, as we’re recursively pushing it up to the root!) We **write** this to disk, and now
7. we delete 42 and the rightmost pointer from the page containing [15, 42], and we **write** that to disk ([15]). Finally,
8. we insert the separator 42 to the root node, setting its left pointer and the pointer between 42 and 70. We **write** this node ([42, 70]) to disk as the new root.

Q4: After performing the insert in Q3, what's the maximum number of keys we can insert into the B+ tree in figure A without splitting the ROOT?

Each leaf node which isn’t full yet can support an additional key, and we have 4 of these empty slots. In addition, each node with spare space on level 1 of the tree can accommodate an additional reference to a whole index leaf page, so each of these empty slots increases our capacity by 2. (Exercises for the reader: 1. What if we had extra space in the root? 2. How many extra keys can an empty slot on level (k) reference? Make sure to account for the extra leftmost pointer!)



Q5: Which of the relational algebra(s) above describe(s) the name of all sailors who have reserved pink boats?

**ABD**

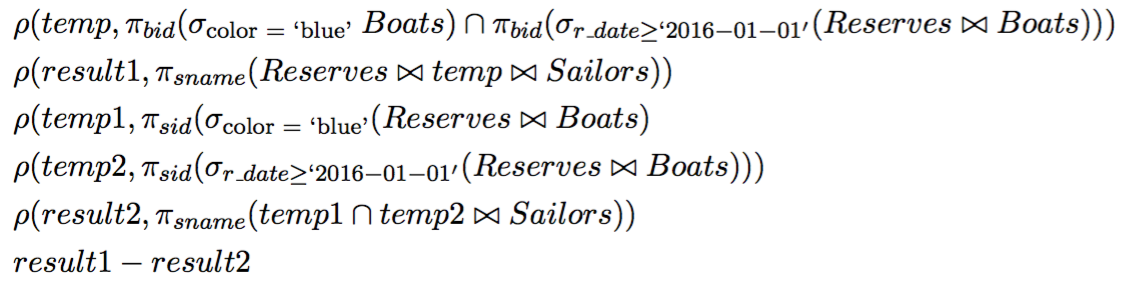
C’s first join wont work because there is no column color

EFG materializes too early and do not have the color column by the time of selection (also in general queries need to end in projections)

Q6: Which one of the above expressions that is correct, if executed as a query plan, is the most performant?

**B**

Since B has selected only the ids out for the join (as opposed to carrying around extra information not needed) AND it doesn’t do the three way join but only joins once the filter is applied.



Q7: Which of the following \*could\* be in the output?

sailors who reserved blue boats but no other boats

sailors who reserved boats in 2016 but only blue boats

sailors who have only reserved boats after 2016

sailors who have reserved blue boats that were reserved by others in 2016 but not the sailor him/herself in 2016

**1,4**

result1 first finds all the blue boats reserved after 2016, then finds the sailors who have ever reserved those boats, even in 2015.

result2 finds sailors who have reserved blue boats anytime in the past, and reserved boats in 2016. So if a sailer has only reserved pink boats, then the sailor will not be included.

A student made a great private post that explains each choice in detail and chose to stay anonymous; here is the answer (with slight modification):

*Since we want to determine if a record \*could\* be in the result. Then the answer should be true iff exists a specific record satisfying certain conditions that is in result1 but not in result2.*

1. sailors who reserved blue boats but no other boats

**YES**: Consider a sailor S who only made one reservation ever, in which he reserved a blue boat B in 2015. The boat B is reserved again by another sailor in 2016.

B∈temp⟹S∈result1. S∉temp2⟹S∉result2

1. sailors who reserved boats in 2016 but only blue boats

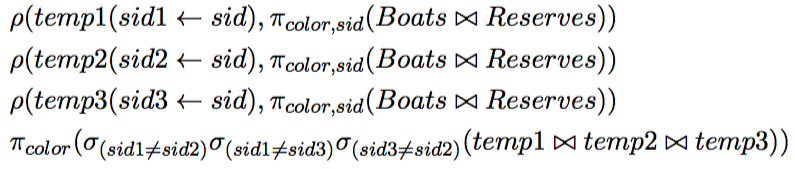
**NO**: Obviously, any such sailor will have his/her record in result2, which prevents it from being in the result.

1. sailors who have only reserved boats after 2016

**NO**: If a record of a such sailor S actually is in result, S can't be in result2. Then S must have not reserved any blue boat, which prevents his/her record from being in result1. Contradiction.

1. sailors who have reserved blue boats that were reserved by others in 2016 but not the sailor him/herself in 2016

**YES**: Again, consider a sailor S who only made one reservation ever, in which she reserved a blue boat B in 2015. The boat B is reserved again by another sailor in 2016.



Q8: What does the algebra above yield?

colors that have been chosen by at least three different sailors in their reservations

colors that have been chosen by at least three sailors in their reservations

the three most common boat colors among boats

**1**

This is similar to the self join we did for the 6 degrees of separation in worksheet for SQL where we make sure the paths length is 6 --- here there must be 3 different sailors.