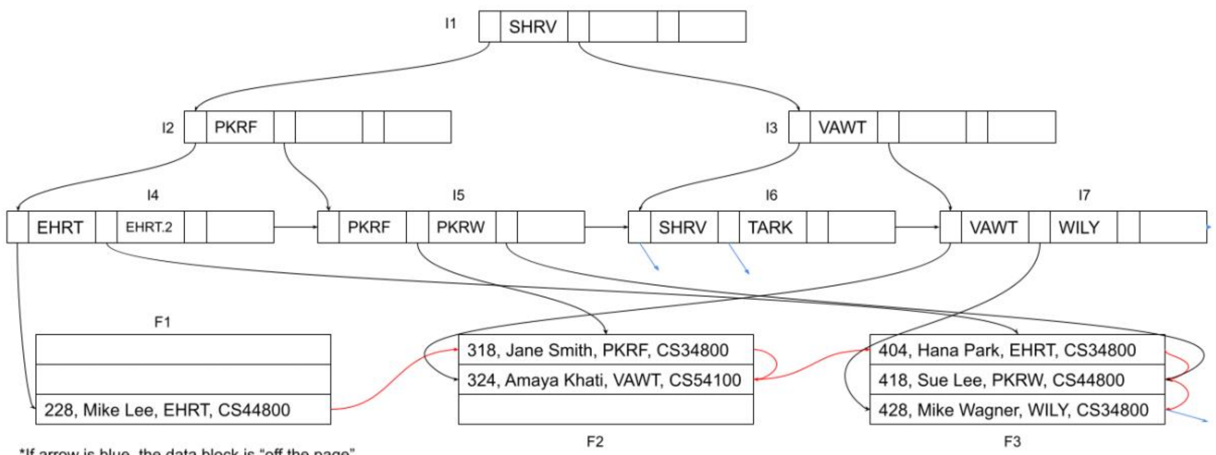
CS44800 Fall 2021 Assignment 3 Solutions

1: B+ Tree

1. l1 -> “SHRV”.
   Since “PKRF” is smaller than “SHRV”, it will go to l2.
   Then compare “MCUT.2” with “PKRF”. “PKRF” > “MCUT.2”, go to l5.
   Then we can find “PKRF” in l5 and that is the end.
   Block I1, I2, and l5 will be read.

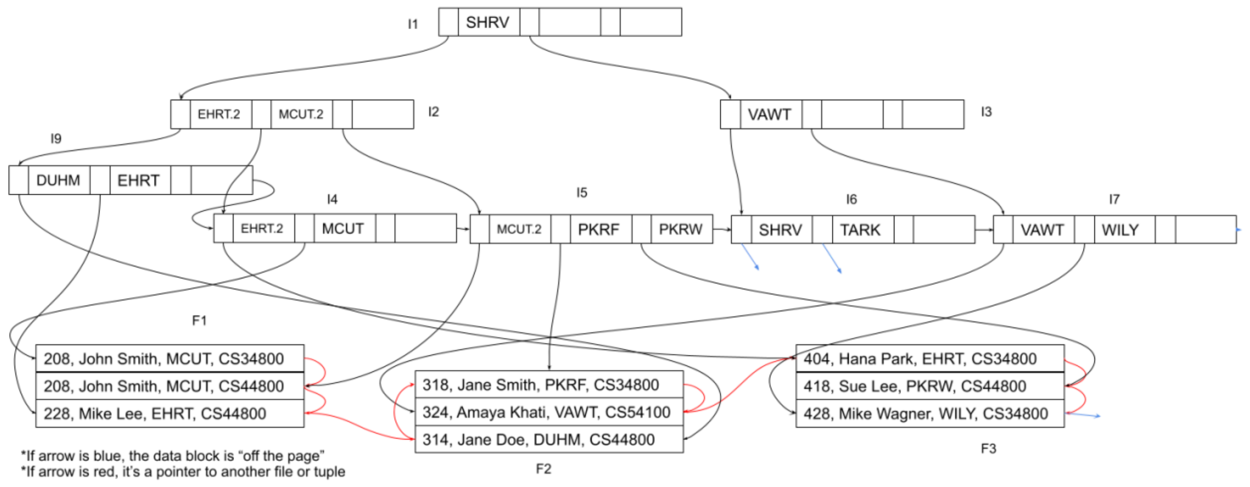
2. We will go through the whole database. If we find the tuple with the name “John Smith”, we will get the key and remove the required key and associated reference from the node. So, we will go to F1 to find whether there is any tuple with name “John Smith” we find that the tuple with the search key “MCUT” is what we want. So, we will find the “MCUT” in the tree and remove it (l1 -> l2 -> l4). After that, we keep finding any tuple with name “John Smith”. We find that another tuple with the search key “MCUT.2” is also what we want. We will find “MCUT.2” in the tree and remove it (l1 -> l2 -> l5). Since “MCUT.2” has been deleted, we need to change l2 into “PKRF”. After that, we cannot find any tuple that fulfills the condition and that is the end. Block l1, l2, l4, and l5 will be read.


I1
SHRV
I2
PKRF
I3
VAWT
I4
I5
I6
I7
EHRT
EHRT.2
PKRF
PKRW
SHRV
TARK
VAWT
WILY
F1
228, Mike Lee, EHRT, CS44800
318, Jane Smith, PKRF, CS34800
324, Amaya Khati, VAWT, CS54100
F2
404, Hana Park, EHRT, CS34800
418, Sue Lee, PKRW, CS44800
428, Mike Wagner, WILY, CS34800
F3

*If arrow is blue, the data block is “off the page”
*If arrow is red, it’s a pointer to another file or tuple

3. Find the place to insert first.
   According to the data, the search key should be “DUHM”.
   First, compare “DUHM” with “SHRV” in l1, which will find “DHUM” is smaller.
   Then go to l2 to compare “DUHM” with “MCUT.2”. Since “DUHM” is smaller, so go to l4.
   Since “DUHM” is smaller than “ERTH”, so add “DUHM” in front of “ERTH” in l4.
   However, since the max degree of this tree is 4, we need to do the split. We can split l4

into 2 different blocks: one with “DUHM” and “EHRT”, and one with “EHRT.2” and “MCUT”. Then change I2 into “EHRT.2” and “MCUT.2”.


I1
SHRV
EHRT.2
MCUT.2
I2
VAWT
I3
I9
DUHM
EHRT
I4
EHRT.2
MCUT
I5
MCUT.2
PKRF
PKRW
I6
SHRV
TARK
I7
VAWT
WILY
F1
208, John Smith, MCUT, CS34800
208, John Smith, MCUT, CS44800
228, Mike Lee, EHRT, CS44800
318, Jane Smith, PKRF, CS34800
324, Amaya Khati, VAWT, CS54100
314, Jane Doe, DUHM, CS44800
F2
404, Hana Park, EHRT, CS34800
418, Sue Lee, PKRW, CS44800
428, Mike Wagner, WILY, CS34800
F3
*If arrow is blue, the data block is “off the page”
*If arrow is red, it’s a pointer to another file or tuple

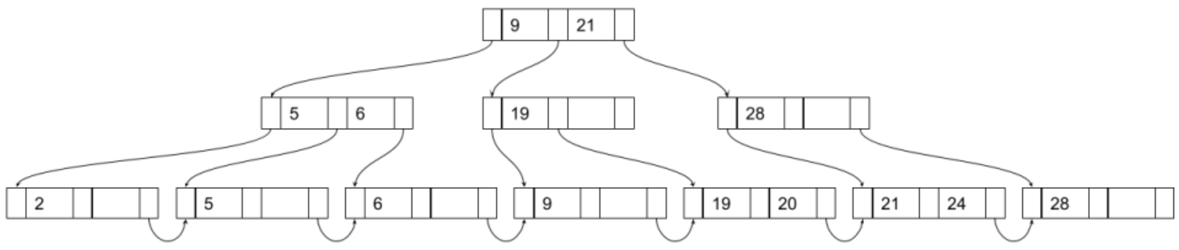
2. Extensible Hashing

1. We first find the hash value of the search key “PKRF” which is 1010. Then I1 shows i=3 so we look at the 3 most significant bits. We get prefix 101. Then reading I1 leads us to I5. We fubd the entry related to PKRF and it leads us to F2 and we read this block. So we read I1, I5 and F2.
2. Since the search key is not the hash key, we know we must do a full file scan to look for the entry. We do a linear scan of files to find the entry containing “John Smith”. We then find 2 entries with the matching values and we find out they both have MCUT for the hash key. We then find the hash value of MCUT, and go to I1 to find out MCUT is stored at I3. We then go to I3 and find all entries with MCUT. We check the entries they point to on the disk which are in F1. If the entries they point to have “John Smith”, we delete them in I3. We also delete them in F1. Finally, we merge I2 and I3.
3. We assume we know F2 has free space. Then we access F2 to insert the entry. Then we compute the hash value of the new entry which is 0001 and we will take the first 3 bits 000. Because 000 points to no bucket, we create a new bucket I7 and let both 000 and 001 point to I7. Finally we access I7 and insert the new entry of DUHM and let that point to F2.

3. Review question: Database Design

No, it is not properly normalized. For example, we can see from the relation that we have a multivalued dependencies ID→→Class and ID→→Name, Resident. Without properly normalizing, we duplicate values, wasting space. An example would be if the students are enrolled in the same class then the students name will be duplicated for each class, they are in. Furthermore, it would be possible to have (for example), a mistake – the same student may have the name “Chris Clifton” in one class and “Clifton Bingham” in another. Decomposing into 4NF would prevent this from occurring, as it would enforce the multivalued dependency (I would either have the same name in all classes, or both names in all classes, depending on if we enforced a functional dependency ID→Name or not.)

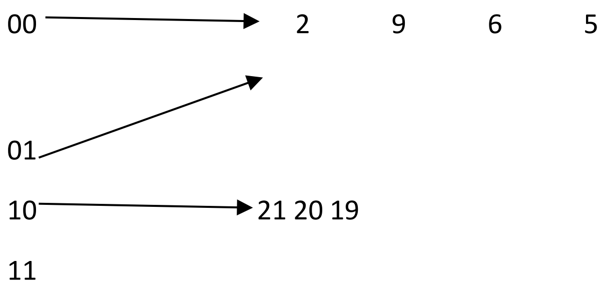
4. B+ Tree



5. Extensible Hashing

Suppose we use the binary of each number as the hash value and we take the most significant bits in our extensible hashing.

Result:



6. Index Choices

Courtesy of Shayne Marques

1. The given query returns a range of data and so the B+ tree index would be most suitable. It would simply have to go through the tree and this would be a logarithmic operation in the worst case. Once the required leaf node is found, in this case the lead node with salary 5000, then using the next pointers of the lead nodes, it would be easy to return all the successive leaf nodes with salaries of greater than 5000. This choice would not run into blocking operations as result of the query does not rely on the complete processing of the input before generating the output.
2. The given query returns data based on a single search rather than a range of a data, therefore using the hash index on id would be most efficient. It would have to find to the required bucket in hash index table which is a constant operation that is dependent on the hash function. Since the bucker is found; we would have to go through all the tuples in the bucket which is also a constant operation since the number of tuples in a bucket is fixed based on the size of the bucket. This choice would not run into blocking operations as the result for the query does not reply on the complete processing of the input before generating the output.

3. We can take two approaches to this query.
    - We could use both indexes to view this query. The first part of the query would use the hash index to find the tuples with id – 10. Then the second part of the query would use the B+ tree index to find the tuples with salary >=5000. Then we would find the intersection of both these sets of relations. However, this would result in a blocking operating in the pipeline evaluation which would affect performance since both input sets would have to be acquired before the final query output us generated.
    - We could use single hash index approach to find all the tuples with id = 10. In this approached we would acquire all the triples with id = 10 and then check each of these tuples to see if the salary is greater than or equal to 5000. This would be a constant operation. This choice would not run into blocking operations as the result of the query not replying on the complete processing of the input before generating the output.

7. Index Scaling

Here, *h* is the height of the B+tree, M=30

Answer:

Query	Heap	B+Tree	Hash Index
1. All Tuples	10^8	h + 10^9/M + 10^9	10^9/M + 10^9
2. s<100	10^8	h + 100/M +100	200
3. s = 100	10^8	h + 1	2
4.s>100 and s<150	10^8	h + 49/M +49	98

For 1→Heap, 2→B+Tree, 3→Hash and 4→B+Tree