## 1 Question 1

Please provide short answers to the following questions:

1. Why might we want to store relations in separate files?
2. Why might we want to store all relations, or even an entire database, in the same file?
3. When might the leaf nodes of a B+-tree file organization lose sequentiality? Suggest how the file organization may be reorganized to restore sequentiality.
4. What are the three parts of a slotted page structure header?
5. Why does the data and keys in a slotted page block grow from either end?
6. How does columnstore make file compression easier? What feature of column stores makes interacting with single tuples expensive?

### 1.1 Solution

1. Explain why we might want to store relations in separate files.

Any of the following are valid: when an index grows, don't have to worry about shifting other data over; don't have to mix indexing types.

2. Explain why we might want to store all relations, or even an entire database, in the same file.

Any of the following are valid: fewer file descriptors open; faster joins, can optimize indexing to account for joins.

3. Why might the leaf nodes of a B+Tree file organization lose sequentiality? Suggest how the file organization may be reorganized to restore sequentiality.

As new records are inserted, splits are made and new pages are allocated for new nodes at the end of the database file. Then, new pages won't be in sequence. If you take the database and insert all of its records into a new B+Tree in order, you'll get a new B+Tree whose underlying data is sequenced correctly. However, over time, it will lose sequentiality again.

4. What are the three types of data held in a slotted page structure header?

A slotted page header stores: number of record entries, end of free space in the block, location and size of each record.

5. Why does the data and keys in a slotted page block grow from either end?

Because records may be of variable length, we don't know how many entries we will need. To minimize the amount of reshuffling and moving to accomodate new keys or records, both the keys and the records grow into free space, and the block is full when no new key and record can fit.

6. How does columnstore make file compression easier? What feature of column stores makes deletes and updates of single tuples especially expensive and why?

In a columnstore, all values for an attribute are stored sequentially, separately from the rest of the record it is in. Since all of these values are of the same type, we can apply type-specific compression to them sequentially, reducing compression time and complexity. However, because records are sharded across multiple parts of disk, updating a single tuple can require up to n block reads, assuming n attributes in the record.

## 2 Question 2

### 2.1 2.1 Storage

Consider the following schema for an online game distributor:

```
CREATE TABLE games (
    gameId INT PRIMARY KEY,
    gameTitle VARCHAR(48),
    distributorId INT,
    price FLOAT
);
```

You can assume the following:

- The table contains 1024 tuples in total.
- `INT`s are 8 bytes long, `VARCHAR(48)`s are 48 bytes long, and `FLOAT`s are 4 bytes long.
- All attributes of a tuple are stored in contiguous space within the same disk block.
- A disk block size is 512 bytes.
- The disk on average performs the sequential read at 1ms per disk block and the random read at 10ms per disk block.

1. What is the maximum number of tuples that can be stored in a disk block?
2. What is the minimum number of disk blocks that need to be allocated to store all tuples in the table?
3. What is the minimum time to read all tuples (in no particular order), assuming that the minimum number of disk blocks are allocated?

### 2.2 2.2 Indexing

Suppose that a secondary index of B+Tree is created on the `distributorId` column. Assume the following:

- The tree has an order of 5 (up to 4 keys, 5 children), each leaf node is 60% loaded, and internal nodes are all fully loaded.
- You can assume distributorId is a candidate key with unique values for simplicity.

1. How many leaf nodes does this secondary index require?
2. How many disk reads (including index search and tuple retrieval) in the worst case are required to find a tuple by its distributorId? Hint: find the height of the tree.

### 2.3 Solution

#### 2.3.1 2.1 Storage

1. What is the maximum number of tuples that can be stored in a disk block?

Each tuple if 2*8 + 48 + 4 = 68 bytes long. 512 // 68 = 7 tuples per disk block.

2. What is the minimum number of disk blocks that need to be allocated to store all tuples in the table?

1024 // 7 + 1 = 147 disk blocks.

3. What is the minimum time to read all tuples (in no particular order), assuming that the minimum number of disk blocks are allocated?

147ms, assuming sequential read.

#### 2.3.2 2.2 Indexing

1. How many leaf nodes does this secondary index require?

Our tree's nodes have up to 5 children, so each leaf node has up to 5 pointers to data. We're at 60% load, so each leaf node has 3 pointers to data. It's a secondary index, meaning every tuple needs a pointer, so we have 1024 / 3 + 1 = 342 leaf nodes.

2. How many disk reads (including index search and tuple retrieval) in the worst case are required to find a tuple by its distributorId? Hint: find the height of the tree.

Using log(342)/log(5) = 3.8, we deduce that our tree height is 4. So, we have to read 4 node blocks, and then the block containing our tuple, leading to 5 total disk reads.

Another interpretation of this problem has that each internal node also has 3 children. In this case, we have log(342)/log(3) = 5.3, meaning our tree height is 6. We have to read one block containing our tuple, leading to 7 total disk reads.

## 3 Question 3

### 3.1 3.1 Insertion

Starting with the given B tree (degree = 3), insert all of the following keys in the given order and show the resulting tree at the end. In the case of equal keys, insert to the left.

[11, 15, 3, 18, 20, 4, 6, 18, 8, 3, 11]

### 3.2 3.2 Deletion

Starting with the given B tree (degree = 4), remove all of the following keys in the given order and show the resulting tree at the end. (11pts)

[15, 11, 18, 3, 8, 5, 3, 7]

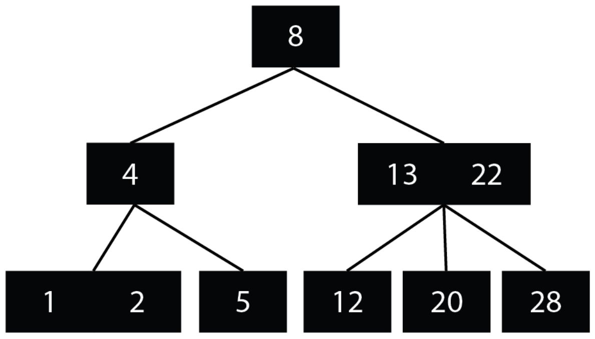

Figure 1: Q3.1

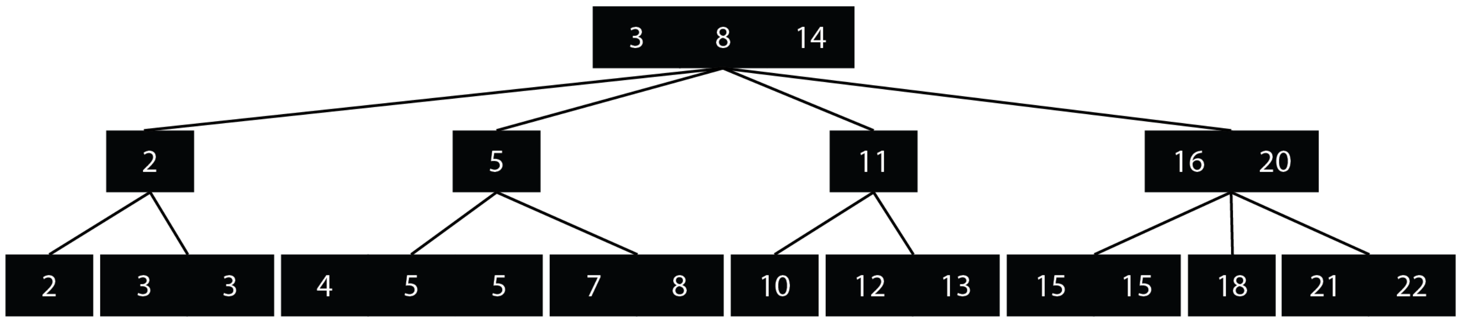

Figure 2: Q3.2

### 3.3 Solution

Link

## 4 Question 4

### 4.1 4.1

Consider the B-tree in 3.2 before we removed the entries. Suppose we want to retrieve all values between 7 and 19 that reside in a leaf node (assume that values in internal nodes only exist for organization purposes; they don't represent real tuples). Assuming our B-tree edges only point downward (i.e. we cannot traverse back up an edge), what is the least number of tree searches it would take to perform this search?

### 4.2 4.2

We've decided we need a faster way to search for a RANGE of records! Suppose we connected each leaf node in our B-Tree with a pointer to the leaf node directly to the right of it. How many tree searches will our query now take and in which order will we visit all nodes?

### 4.3 4.3

Alternatively, instead of using the solution we proposed in 4.2, we could make all edges bidirectional. This allows us to visit neighbouring nodes by traversing up to an ancestor node, then traversing back down. Is it better to make every branch a doubly-linked pointer, or is our linked list solution better? Why?

### 4.4 Solution

#### 4.4.1 4.1

It would take 5 tree searches. One to find the lowest value, 7. One to find the highest value, 18. There are 3 leaf nodes "in between" 7 and 18, so they have to be found with 3 individual searches. In total, that's 5.

#### 4.4.2 4.2

It now only takes one tree search. {3, 8, 14} -> {5} -> {7, 8} -> {10} -> {12, 13} -> {15, 15} -> {18}

#### 4.4.3 4.3

Linked list is better. As the tree grows, there will be many more branches then there will be leaf nodes. There will be more total branches than leaf nodes because there will be more total nodes than leaf nodes.

## Feedback

As this is a new course, we appreciate any feedback you have for us! If you enjoyed this assignment, hated this assignment, or have other thoughts to share, please do so here!