Indexed Sequential File Access

- We've seen how physically adjacent records support efficient cosequential file processing.

- But such file structures don't readily support record access, insertion, and deletion. Such operations are supported by indexing (including B-trees).

- What if an application requires both random access and cosequential batch processing?

Maintaining a Sequence Set

- Maintain blocks of physically adjacent records, with a block pointing to the next block in the sequence set.

- insertion and deletion occur within a block, leading to both underflow or overflow.

- in the case of overflow, we need to split the block into two blocks, and distribute the original block's records evenly across the two blocks.

- in the case of underflow, we need to merge with a logically adjacent block or redistribute a logically adjacent block.

How big should the blocks be?

- blocks can accommodate sequential processing, but only within a block, so blocks should be big.

- but, we should be able to hold 2-3 blocks in memory, e.g., so we can perform a 1-2 or 2-3 split.

- and, we should be able to read or write a block quickly, so blocks should not be too large in order to avoid i/o overhead for a single read/write.

- how about making a block the size of a cluster? Remember what a cluster is?

- a cluster is the min number of contiguous sectors allocated at a time, and whose access requires only one seek.

How do we add an index to our sequence set?

- build a fixed-length record index that contains the last key in each block.

- index would be stored in memory, supporting a binary search and supporting quick index changes due to block splitting, merging, and/or redistribution.

- if number of blocks is too large to fit in memory, then we could use a B-tree to store the index to the sequence set.

- the resulting hybrid structure, combining a B-tree index with a sequence of blocks, is called a B^+-tree.

- Note that the keys in the index do not have to be chosen from the file, but rather they can be variable-width strings chosen to optimally separate the blocks, providing space savings.

Localized changes in the sequence set

- if deletion doesn't require merging or redistribution, then changes are completely local to the block containing the record to be deleted.

- if insertion doesn't require splitting, then changes are completely local to the block containing the new record.

- changes involving multiple blocks are handled through the normal B-tree operations.