Maintenance of Data Stored on Disk

Data Organization in a File

Individual records can be organized in a number of different ways in the database file.

Heap File Organization. Heap File denotes a record organization method where records are stored on disk pages on first-come — first-serve basis, in no particular order. That is, any record can be placed anywhere in the file, subject to space availability.

Sequential File Organization. A *search key* is defined for the relation, and records are stored ordered according to the search key. Note that the *search key* need not be a primary key, or even a superkey of the relation being stored. New records must be inserted according to their search key value.

Hashing File Organization. Records are hashed on some attribute value. Each hash value is associated with a block (sequence of blocks is overflow buckets are needed) where the record is to be stored.

Record Modifications

We need to discuss three basic types of modification:

- Record Insertion
- Record Deletion
- Record Update

Insertion

Record insertion procedures are different for different file organizations.

Insertion into a Heap File

To insert records into a heap file, it is convenient to keep a (double-)linked list of all heap file pages with available space. The pointer to the first page of the list can be stored on the *header page*. The pointers to next/previous page on the list can be stored in the *block headers*.

Assuming existance of such support, insertion of a record into a heap file can be done as follows:

```
Algorithm InsertRecordHeap(File F, Record R)

begin
   HeaderPage = ReadBlock(F,1);   // Retrieve the header page of the file
   FreeSpacePageId = HeaderPage.FreeSpaceList;

   Block = ReadBlock(F,FreeSpacePageId);  // Retrieve the block
   RecordNum = FindFreeSlot(Block);       // find a free slot
   Put(R, Block, RecordNum);              // write the contents of R into the
                                          // available slot
   if (No more free space left in Block)
         begin
           HeaderPage.FreeSpaceList = Block.FreeSpaceListNext;
           NextBlock = ReadBlock(F,Block.FreeSpaceListNext);
   NextBlock.FreeSpaceListPrevious = 1;
   WriteBlock(NextBlock, F, Block.FreeSpaceListNext);
 end

    WriteBlock(Block, F, FreeSpacePageId); // write the data back to disk
    WriteBlock(HeaderPage, F, 1);
 end
```

Notice that this algorithm assumes that `FreeSpacePageId` is not NULL. If it is NULL, then a new page needs to be created, and the records needs to be put on it.

Finding a free slot on the disk page

There is a number of ways by which a free slot can be found on the disk page. All the computations occur in main memory and do not have I/O costs associated with them.

1. Direct Scan. Each record contains a *tombstone* - a byte indicating whether it is active or deleted. The disk page is scanned until the first record with the *tombstone* set to "*deleted*" is found.

2. Bitmap. Record header contains a bitmap, specifying which of the records on the page are available. Instead of the entire disk page, the bitmap is

scanned until the first available slot is discovered. After the record is inserted, the bitmap must be updated.

Insertion into a Sequential File.

When data is inserted into a sequential file, the following must be observed:

- The record needs to be inserted according to the value of its key.
- The file needs to be scanned (or an outside index needs to be used) to find the disk block where there record must be inserted.
- If there is no empty slot at the location where the record needs to be inserted, *record sliding* must be used to shift all subsequent records on the page (assuming there is space available). This operation is as simple *byte copy*, so it can be performed fast.
- If there is not space on the page, two possibilities can be considered:
  - Record sliding to next page. If the next disk block has enough space, we can do record sliding and put some overflow records on the next page, using record sliding there as well.
  - Overflow page. Alternatively, a new disk block can be allocated. The disk block is inserted between the current block and the next block in the *search key* order. The contents of the current block are split roughly evenly between the old block and the new block, after which the new record is inserted in its designated location (which can be either on the old page, or on the new one).

Insertion into a Hashed File.

Insertion into a *hashed file* proceeds in two steps.

Step 1: The hash key of the record being inserted is computed, and the disk page(s) for the hash bucket is/are identified.

Step 2: An empty slot is found on one of the pages of the hash bucket (if not - a new page is added to the hash bucket), and the record is inserted into that slot. The procedure is similar to the one used for a *heap file*.

Deletion

Unlike insertion, methodology of record deletion does not depend on the type of file used to store the relational table.

Some traditional approaches to deletion are:

Deletion with slide: The deleted record creates a "gap" on a disk page. To cover this gap, the records following the gap on the disk page are shifted left, covering the gap, and moving the empty slot to the end of the page.

Advantages: Free space is always located only at the end of the disk page. One pointer in the block header (or simply a count of occupied slots for fixed-length records) is sufficient to keep track of free space within the block.

Disadvantages: The deleted record gets immediately overwritten. This removes the possibility of an easy "undo" action.Additionally, this complicates handling dangling pointers to the deleted record (the pointers will now point at a completely different "live" record).

Deletion without slide: The slot of the deleted record is declared available, and the record itself is marked as deleted (by activating the tombstone flag, for example), but the actual space occupied by the record stays as-is. The record can be later overwrritten, when a new record is inserted in the same space.

Advantages: Simplicity. This method requires little extra actions, save for changing the status of the slot to "available".

Disadvantages: Gaps in the disk page. This means that more space or more complicated organization is needed to keep track of free space on the page. Possible solutions include a bitmap in the disk header, or a linked list of all available slots: the pointer to the first available slot is stored in the disk header, while all subsequent pointers are stored in the records themsleves.

Deletion without giving up space: The record is marked as deleted, but the space is not freed. No new record can be stored in this space until a major DB reconstruction/reconciliation occurs.

Advantages: Ensures proper handling of all "dangling pointers".

Disadvantages: The relation grows in size with every insertion. Deletions do not decrease the space alotted for the insertion.

Additionally, deletions may lead to removal of disk pages from the file. There is a number of approaches here.

- "Lazy evaluation." The disk page is removed only when it contains no live records on it. Disk pages are never merged.
- "Eager evaluation." Each time a deletion occurs, the system checks whether current disk page can be merged with one of its neighbors. If the answer is yes, such merger is performed. The minimal criterion for merger is the ability to fit all records into the free space available in the two neighboring pages. Other, stricter criteria (e.g., requiring at least a few empty slots available in the resulting blocks) are also possible.

Update

For fixed-length record, *record updates* do not change the storage. The basic algorithm is the same: the block with the desired record is located on disk (via scan, index structure or other means), the disk page is retrieved, the record is found, update it performed, and the block is flushed back to disk.