File Organization

* A database is stored as a collection of files. Each file is a sequence of records, and a record is a sequence of fields.
* Approaches to organizing records in files:
  + Assume the record size is fixed
  + Each file has records of one particular type only
  + Different files are (typically) used for different relations

Fixed-Length Records

* Simple approach:
  + Store record *i* starting at byte *n*∗(*i*−1), where *n* is the size of each record.
  + Record access is simple but records may span blocks.
* Deletion of record *i* (alternatives to avoid *fragmentation*)
  + move records *i* +1*,...,n* to *i,...,n* −1
  + move record *n* to *i*
  + Link all free records in a *free list*

Free Lists

* Maintained in the block(s) that contains deleted records.
* Store the address of the first record whose content is deleted in the file header.
* Use this first record to store the address of the second available record, and so on.

One can think of these stored addresses as *pointers*, because they “point” to the location of a record. (linked list)

* More space efficient representation: reuse space for normal attributes of free records to store pointers (i.e., no pointers are stored in in-use records).
* Requires careful programming: *Dangling pointers* occur if a record is moved or deleted and another record contains a pointer to this record; that pointer then doesn’t point any longer to the desired record.

Variable-Length Records

* Variable-length records arise in database systems in several ways:
  + Storage of multiple record types in a file.
  + Record types that allow variable length for one or more fields (e.g., varchar)
* Approaches to store variable length records:

1. *End-of-Record* marker

* + - difficult to reuse space of deleted records (fragmentation)
    - no space for a record to grow (e.g., due to an update)

*;* requires moving

2. *Field delimiters* (e.g., a $)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Field1 | $ | Field2 | $ | Field3 | $ | Field4 | $ |

* + - requires scan of record to get to *n*-th field value
    - requires a field for a *null* value

3. Each record has an *array of field offsets*

Field1

Field2

Field3

Field4

+ For the overhead of the offset, we get direct access to any field

+ Null values: begin/end pointer of a field point to the same address

Variable length records typically cause problems in case of (record) attribute modifications:

* Growth of a field requires shifting all other fields
* A modified record may no longer fit into the block (leave forwarding address at the old location)
* A record can span multiple blocks

Block formats for variable length records

Each record is identified by a *record identifier (rid)* (or *tuple identifier (tid)*). The rid/tid contains number of block and position in block

*;* simplifies shifting a record from one position/block to another position/block

Allocation of records in a block is based on *slotted block structure*:

......

Free Space

......

R1

R2

R3

R4

Size

Location

#Entries

Block Header

* block header contains number of record entries, end of the free space in the block, and location and size of each record
* records are inserted from the end of the block
* records can be moved around in block to keep them contiguous

RID-Concept:

RID Block 4711 Block 4711

4711/2

3

2

1

record

record

5013/1

3

2

1

record

record

record

Update of

the record

Organization of Records in Files

Requirements: A file must (efficiently) support

* insert/delete/update of a record
* access to a record (typically using rid)
* scan of all records

Ways to organize blocks (pages) in a file:

Heap File (unsorted file) simplest file structure; contains records in no particular order; record can be placed anywhere in the file where there is space

Sequential File records are stored in sequential order, based on the value of the search key of each record

Clustered Index related to sequential files; we’ll talk about this later in Section 7 (Indexes)

Heap File Organization

* At DB run-time, pages/blocks are allocated and deallocated
* Information to maintain for a heap file includes pages, free space on pages, records on a page
* A typical implementation is based on a two doubly linked list of pages; starting with header block.
* Two lists can be associated with header block: (1) full page list, and (2) list of pages having free space

Sequential File Organization

* Suitable for applications that require sequential processing of the entire file
* The records in the file are ordered by a *search-key*.
* Deletions of records are managed using pointer chains.
* Insertions – must locate the position in the file where the record is to be inserted
  + if there is free space, insert the record there
  + if no free space, insert the record in an *overflow block*
  + in either case, pointer chain must be updated
* If many record modifications (in particular insertions and deletions), correspondence between search key order and physical order can be totally lost =⇒ file reorganization

Extract from: <https://web.cs.ucdavis.edu/~green/courses/ecs165a-w11/6-storage.pdf>