I]File Organization

File

A file is organized logically as a sequence of records.

Block

Each file is also logically partitioned into fixed-length storage units called blocks.
Blocks are the units of both storage allocation and data transfer.
block sizes of 4 to 8 kilobytes by default
assume that no record is larger than a block

A)Fixed-Length Records

Even
varch

type *instruc*

However, there are two problems with this simple approach:

1. Unless the block size happens to be a multiple of 53 (which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.
2. It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file, or we must have a way of marking deleted records so that they can be ignored.

record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Reco
where

Problem 1: Unless the block size is a multiple of ***n***, *the last* record in a block crosses the block boundary

- Requires two block accesses!
- Modification: leave the fractional record at the end of the block unused.

Problem 2: What to do when a record (*i*) is deleted?

Possible solutions:

- shift re
- move
- do not link all *free lis*

Deleting record 3 and shifting

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
				2000
				0000
				2000
				2000
				0000

Deleting record 3 and moving last record

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Free Lists (Linked)

- Store the address of the first deleted record in the **file header**.
- Can think of these stored addresses as **pointers** since they "point" to the location of a record.
- For efficiency, reuse the space for normal attributes in the free records to store pointers. (No pointers stored in in-use records!)

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

B)Variable-Length Records

Variable-length records arise in several ways:

- Storage of multiple record types in a file.
  - E.g. the records represent tuples from different tables
- Record types that allow variable lengths for one or more fields such as strings (**varchar**)
- Record types that allow repeating fields (used in some older data models).

Different techniques for implementing variable-length records exist. Two different problems must be solved by any such technique:

1. How to represent a single record in such a way that individual attributes can be extracted easily, even if they are of variable length
2. How to store variable-length records within a block, such that records in a block can be extracted easily

Problem1;

the representation of a record with variable-length attributes two parts

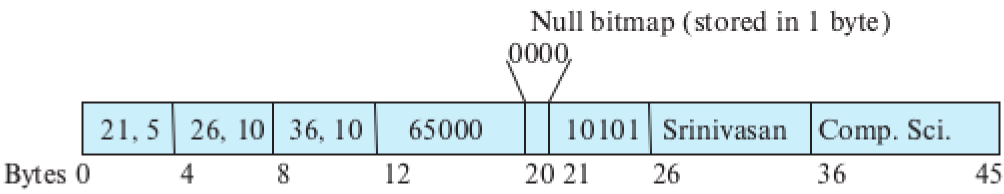
1. **an initial part with fixed-length information**
2. **followed by the contents of variable-length attributes**

type *instructor* = **record**
ID **varchar** (5);
name **varchar**(20);
dept_name **varchar** (20);
salary **numeric** (8,2);
end

- Attributes are stored in order, but
- Variable length attributes represented by a fixed size pair (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap

The figure

shows an instructor record whose first three attributes ID , name, and dept name are variable-length strings,
and whose fourth attribute salary is a fixed-sized number



null bitmap

indicates which attributes of the record have a null value
eg.if the salary were null, the fourth bit of the bitmap would be set to 1
and the salary value stored in bytes 12 through 19 would be ignored
In some representations, the null bitmap is stored at the beginning of the record,
for attributes that are null, no data (value, or offset/length) are stored at all.
Such a representation would save some storage space

Problem2;

slotted-page structure is commonly used for organizing records within a block

- The number of record entries in the header
- The end of free space in the block
- An array whose entries contain the location and size of each record

- Records are allocated contiguously in the page/block, starting from the end.
- Records can be moved around within the page/block to keep them contiguous (no empty space between them)
  - header entry is updated on every move
  - b/c of this, outside pointers should not point directly to record but to the header entry.

If

a

record is inserted, space is allocated for it at the end of free

space, and an entry containing its size and location is added to the header.

If a record is deleted, the space that it occupies is freed, and its entry is set to deleted.

Further, the records in the block before the deleted record are moved, so that the free space created by the deletion gets occupied.

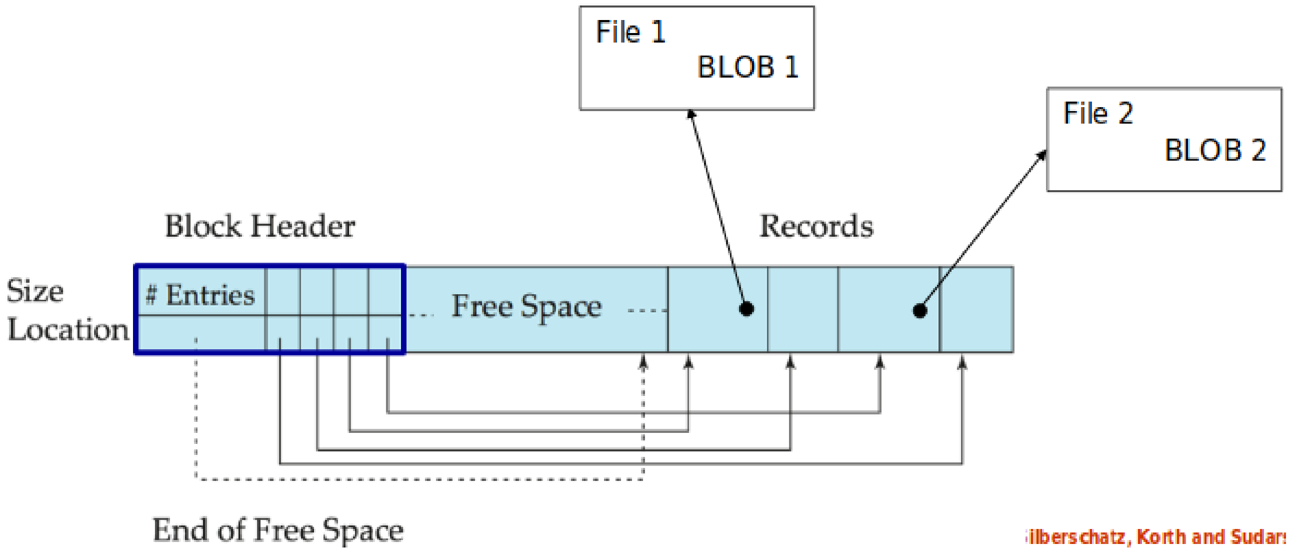
The end-of-free-space pointer in the header is appropriately updated as well.

Storing Large Objects

data that can be much larger than a disk block

e.g. blob and clob, which store binary and character large objects.

Large objects are often stored separately from the other (short) attributes, in special file(s). In this case, the record containing the large object has only a pointer to the object.



II]Organization of Records in Files

1)Heap– a record can be placed anywhere in the file (in any block) where there is space,No ordering whatsoever

2)Sequential – store records in sequential order, based on the value of the search key of each record

3)Multitable clustering file organization: records of several different relations are stored in the same file, and in fact in the same block within a file, to reduce the cost of certain join operations.

4)B+-tree file organization

can provide efficient ordered access to records even if there are a large number of insert, delete, or update operations. Further, it supports very efficient access to specific records, based on the search key.

5)Hashing file organization

a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed.

1 Heap File Organization

a record may be stored anywhere in the file corresponding to a relation.

Once placed in a particular location, the record is not usually moved

When a record is inserted in a file, one option for choosing the location is to always add it at the end of the file.

However, if records get deleted, it makes sense to use the space thus freed up to store new records

space-efficient data structure called a free-space map to track which blocks have free space to store records

an array containing 1 entry for each block in the relation.

a fraction f such that at least a fraction f of the space in the block is free.

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

a free-space map for a file with 16 blocks.

assume that 3 bits are used to store the occupancy
fractioN

a value of 7 indicates that at least 7⁄8th of the space in the block is free

To find a block to store a new record of a given size, the database can scan the free-space map to find a block that has enough free space to store that record. If there
is no such block, a new block is allocated for the relation.

second-level free-space map

with 1 entry for every 4 entries in the main free-space map.

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

4	7	2	6

2 Sequential File Organization

store records in sequential order, based on the value of the search key of each record

Search key need not be PK, or even superkey!

- The records in the file are ordered by a search-key
- Suitable for applications (e.g. queries) that require sequential

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

3Multitable Clustering File Organization

Many large-scale DBMSs do not rely directly on the underlying OS for file management.

Instead, the OS allocates one large file to the DBMS, and the DBMS stores all relations in this one file, and manages the file itself.

Even if multiple relations are stored in this a single large file, the default is to store records of only one relation in a given block.

- This simplifies data management.

However, in some cases it can be useful to store records of more than one relation in a single block. This is called **multitable clustering**. Example:

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

stores related records of two or more relations in each block

The cluster key is the attribute that defines which records are stored together; in our preceding example, the cluster key is dept name.

Multitable Clustering File Organization (cont.)

table

- good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
- bad for queries involving only *department*
- results in variable size records
- Can add pointer chains to link records of a particular relation

Comp. Sci.	Taylor	100000	
45564	Katz	75000	
10101	Srinivasan	65000	
83821	Brandt	92000	
Physics	Watson	70000	
33456	Gold	87000	

partitioning

the records in a relation to be partitioned into smaller relations,that are stored separately

on the basis of an attribute value

transaction relation-->transaction 2018, transaction 2019,

```
select *
from transaction
where year=2019
```

would only access the relation transaction 2019, ignoring the other relations, while a query without the selection condition would read all the relations.