



## **Storage and File Structure**

# File Organization

- A database is:
  - A collection of *files*, A file is
    - A sequence of *records*, A record is
      - A sequence of *fields*
- One approach:
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations
- This case is easiest to implement; will consider variable length records later

# Fixed-Length Records

- Simple approach:
  - Store record  $i$  starting from byte  $n * (i - 1)$ , where  $n$  is the size of each record
  - Record access is simple but records may cross blocks
    - Modification: Do not allow records to cross block boundaries

- Deletion of record  $i$ : Alternatives:
  - Move records  $i + 1, \dots, n$  to  $i, \dots, n - 1$
  - move record  $n$  to  $i$
  - Do not move records, but link all free records on a *free list*

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
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
record 11	98345	Kim	Elec. Eng.	80000

# Deleting Record 3 and Compacting

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
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

# 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

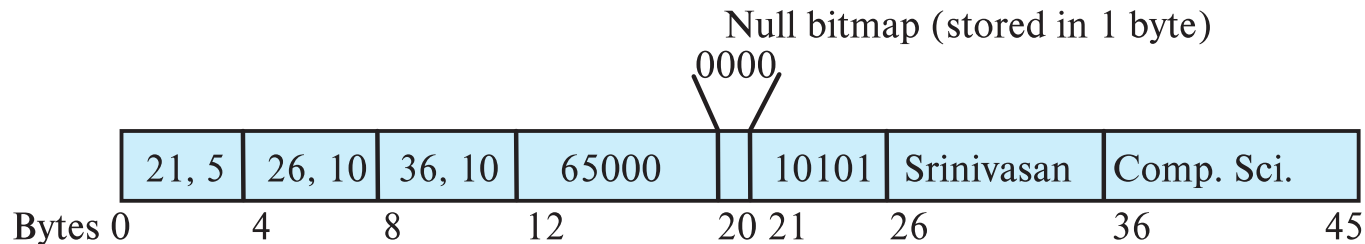
# Free Lists

- Store the address of the first deleted record in the file header
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they “point” to the location of a record
- More space efficient representation: Reuse space for normal attributes of free records to store
- Pointers (no pointers stored in in-use records)

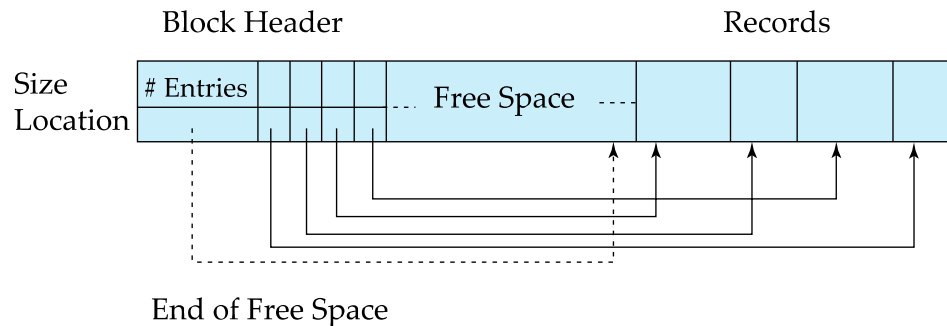
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology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

# 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 lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



# Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - Number of record entries
  - End of free space in the block
  - Location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated
- Pointers should not point directly to record, instead they should point to the entry for the record in header



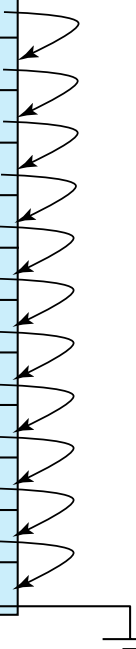
# Organization of Records in Files

- **Heap:** Record can be placed anywhere in the file where there is space
- **Sequential:** Store records in sequential order, based on the value of the search key of each record
- **Hashing:** A hash function computed on search key; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file
- In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: Store related records on the same block to minimize I/O

# Sequential File Organization

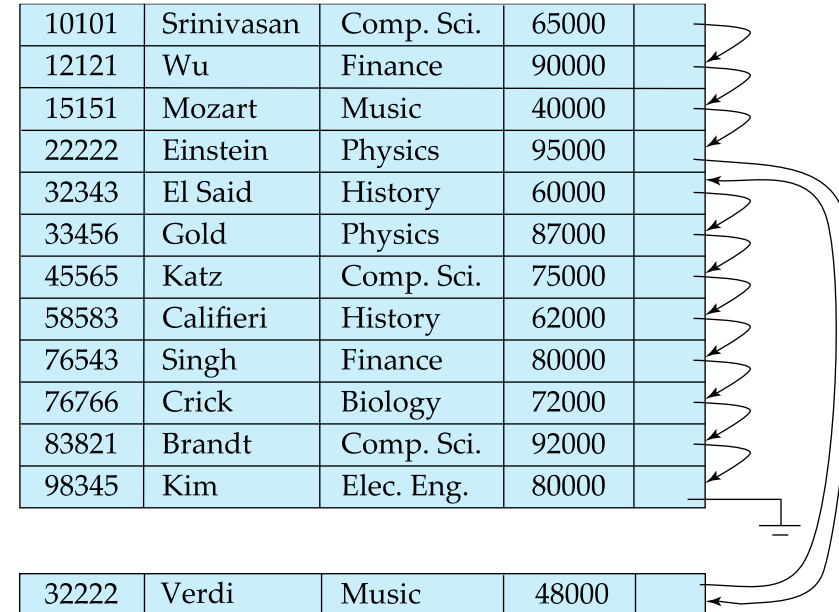
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

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	



# Sequential File Organization

- 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



# Multitable Clustering File Organization

- Store several relations in one file using a **multitable clustering** file organization

*department*

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Physics	Watson	70000

*instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Comp. Sci.	Taylor	100000
10101	Srinivasan	65000
45565	Katz	75000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

# Multitable Clustering File Organization

- 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	

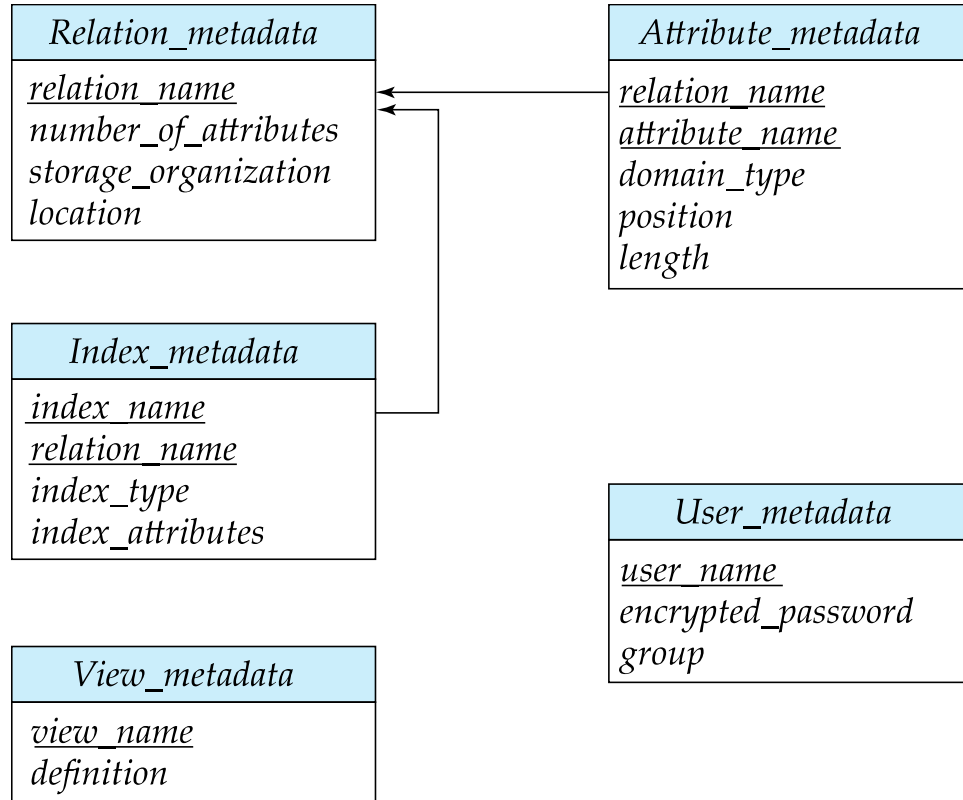


# Data Dictionary Storage

- The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as,
  - Information about relations
    - Names of relations
    - Names, types and lengths of attributes of each relation
    - Names and definitions of views
    - Integrity constraints
  - User and accounting information, including passwords
  - Statistical and descriptive data
    - Number of tuples in each relation
  - Physical file organization information
    - How relation is stored (sequential/hash/...)
    - Physical location of relation
  - Information about indices

# Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



# Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**
  - Blocks are units of both storage allocation and data transfer
- Database system seeks to minimize the number of block transfers between the disk and memory
  - We can reduce the number of disk accesses by keeping as many blocks as possible in main memory
- **Buffer:** Portion of main memory available to store copies of disk blocks
- **Buffer manager:** Subsystem responsible for allocating buffer space in main memory



# Buffer Manager

- Programs call on the buffer manager when they need a block from disk
  - If the block is already in the buffer, buffer manager returns the address of the block in main memory
  - If the block is not in the buffer, the buffer manager
    - Allocates space in the buffer for the block
      - Replacing (throwing out) some other block, if required, to make space for the new block
      - Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk
    - Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester

# Buffer-Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
- Idea behind LRU: Use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data
    - For example: When computing the join of 2 relations  $r$  and  $s$  by a nested loops
      - for each tuple  $tr$  of  $r$  do
        - for each tuple  $ts$  of  $s$  do
          - if the tuples  $tr$  and  $ts$  match ...
    - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

# Buffer-Replacement Policies

- **Pinned block:** Memory block that is not allowed to be written back to disk
- **Toss-immediate** strategy: Frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy:** System must pin the block currently being processed
  - After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed
  - Heuristic: Keep data-dictionary blocks in main memory buffer
- Buffer managers also support **forced output** of blocks for the purpose of recovery

# Next Lecture

## **Basic Concepts of Indexing**

# Thank you for your attention...

Any question?

**Contact:**

Department of Information Technology, NITK Surathkal, India  
6<sup>th</sup> Floor, Room: 13

**Phone:** +91-9477678768

**E-mail:** [shrutilipi@nitk.edu.in](mailto:shrutilipi@nitk.edu.in)