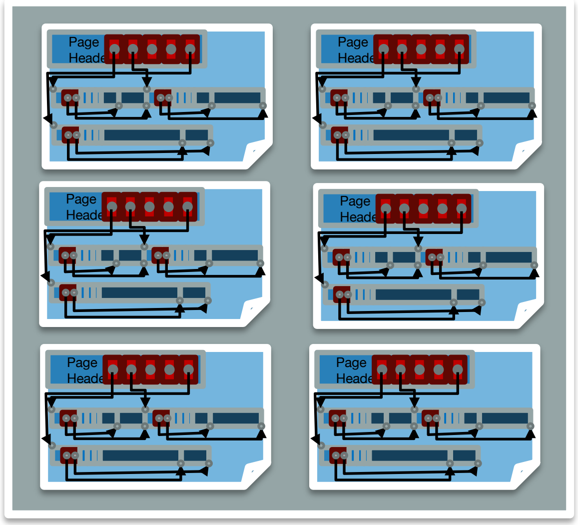
Introduction to Database Systems

2023-Fall

4. Database Management Systems （2/5）

Overview: Files of Pages of Records

- Tables stored as logical files
  - Consist of pages
    - Pages contain a collection of records
- Pages are managed
  - On disk by the disk space manager:
    pages read/written to physical disk/files
  - In memory by the buffer manager:
    higher levels of DBMS only operate in memory



Database Files

4.2 Database Access Management

The access to database is transferred to the operations on files (of OS) eventually. The ***file structure*** and ***access route*** offered on it will affect the speed of data access directly. It is impossible that one kind of file structure will be effective for all kinds of data access.

- Access types
- File organization
- Index technique

Access Types

- Query all or most records of a file (>15%)
- Query some special record
- Query some records (<15%)
- Scope query
- Update

File Organization

- **Heap file:** records stored according to their inserted order, and retrieved sequentially. This is the most basic and general form of file organization.
- **Direct file:** the record address is mapped through hash function according to some attribute's value.
- Dynamic hashing
- Indexed file: index + heap file/cluster
- Grid structure file: suitable for multi attributes queries
- Raw disk (notice the difference between the logical block and physical block of file. You can control physical blocks in OS by using raw disk)

Many DB File Structures

- Unordered Heap Files
  - Records placed arbitrarily across pages
- Clustered Heap Files
  - Records and pages are grouped
- Sorted Files
  - Pages and records are in sorted order
- Index Files
  - B+ Trees, Linear Hashing, …
  - May contain records or point to records in other files

Records

Record Formats

- Relational Model →
  - Each record in table has some fixed type
- Assume System Catalog stores the Schema
  - No need to store type information with records (save space!)
  - Catalog is just another table …
- Goals:
  - Records should be compact in memory & disk format
  - Fast access to fields (why?)

- Easy Case: Fixed Length Fields
- Interesting Case: Variable Length Fields

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*

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

Fixed-Length Records

- Deletion of record *i*: alternatives*:*
  - **move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1**
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

  Record 3 deleted

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

Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - move records *i* + 1, . . ., *n* to *i, . . . , n* – 1
  - **move record *n* to *i***
  - do not move records, but link all free records on a *free list*

  Record 3 deleted and replaced by record 11

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

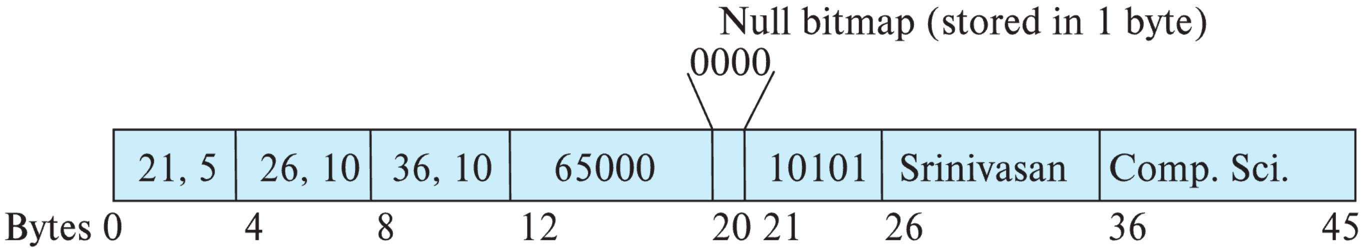
Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record n to i
  - **do not move records, but link all free records on a *free list***

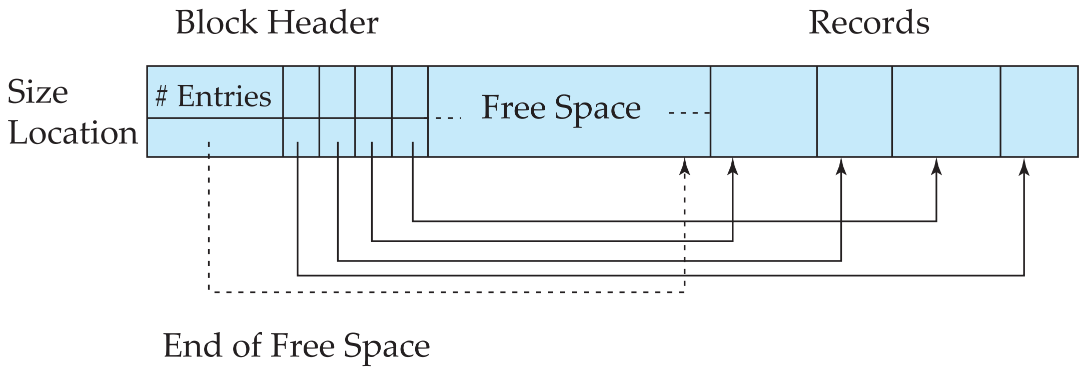
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology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

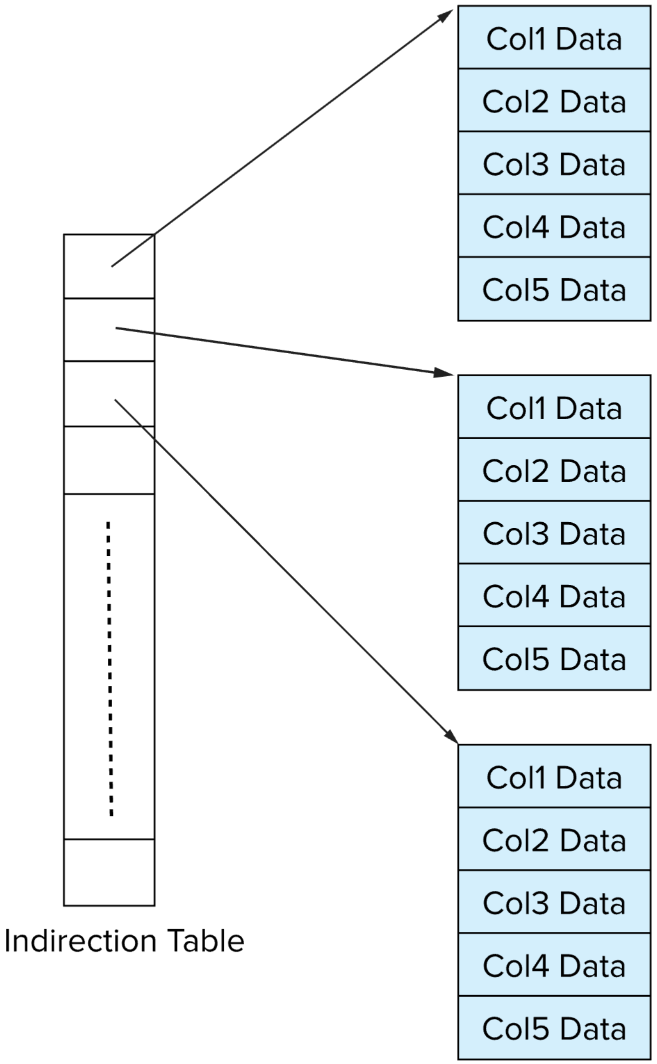
Column-Oriented Storage

- Also known as **columnar representation**
- Store each attribute of a relation separately
- Example

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

Storage Organization in Main-Memory Databases

- Can store records directly in memory without a buffer manager
- Column-oriented storage can be used in-memory for decision support applications
  - Compression reduces memory requirement



Index Technique

- B+ Tree
- Clustering index
- Inverted file
- Dynamic hashing
- Grid structure file and partitioned hash function
- Bitmap index (used in data warehouse)
- Others

Index

An **index** is data structure that enables fast **lookup** and **modification** of **data entries** by **search key**

- **Lookup**: may support many different operations
  - **Equality**, 1-d range, 2-d region, …
- **Search Key:** any subset of columns in the relation
  - Do not need to be unique
    - —e.g. (firstname) or (firstname, lastname)

Index: Basic Concepts

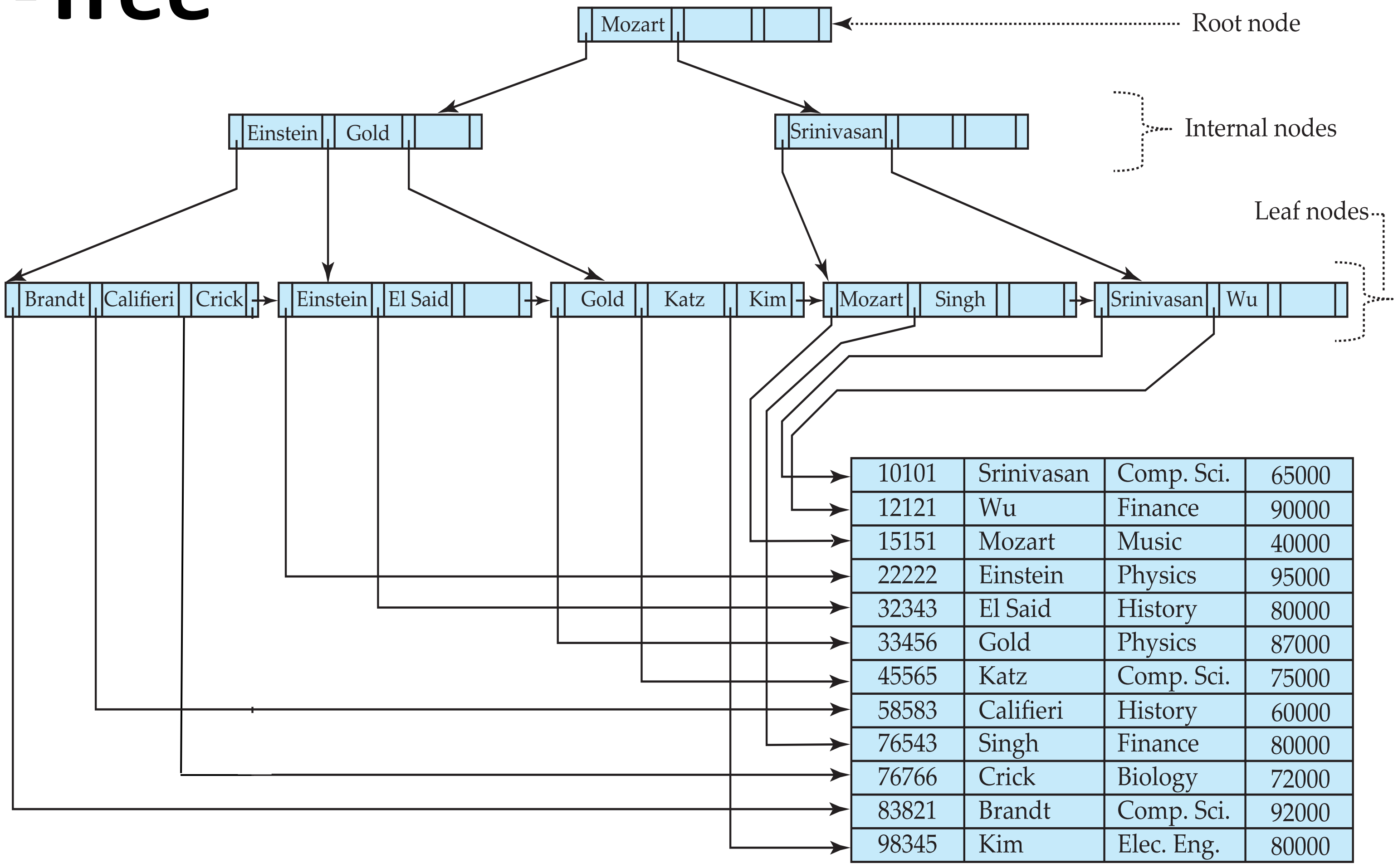
- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer

- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across “buckets” using a “hash function”.

B+-Tree

Example of B$^+$-Tree



- ***Dynamic Tree Index***
  - ***Always Balanced***
  - ***Support efficient insertion & deletion***
  - ***Grows at root not leaves!***

B^+-Tree Index Files

A B^+-tree is a rooted tree satisfying the following properties:

- All paths from root to leaf are of ***the same length***
- Each node that is not a root or a leaf has ***between*** $\lceil n/2 \rceil$ ***and n children***.
- A leaf node has ***between*** $\lceil (n-1)/2 \rceil$ ***and n–1 values***
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values.

B^+-Tree Node Structure

- Typical node

P_1	K_1	P_2	$\dots$	P_{n-1}	K_{n-1}	P_n

  - K_i are the search-key values
  - P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered

$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

Hashing

Example of Hash File Organization

Hash file organization of *instructor* file, using *dept_name* as key.

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Dynamic Hashing

- Periodic rehashing
  - If number of entries in a hash table becomes (say) 1.5 times size of hash table,
    - create new hash table of size (say) 2 times the size of the previous hash table
    - Rehash all entries to new table
- Linear Hashing
  - Do rehashing in an incremental manner
- Extendable Hashing
  - Tailored to disk based hashing, with buckets shared by multiple hash values
  - Doubling of # of entries in hash table, without doubling # of buckets