Module 11: Data Storage Structure

Database System Concepts, 7th Ed.

Database Storage Architecture

- Persistent data is stored on nonvolatile storage, which, is typically magnetic disk or SSD.
- Most databases use operating system files as an intermediate layer for storing records.
- Given a set of records, there are different ways on how to organize them in the file structure.
- Databases must be fetched from nonvolatile storage and saved back if it is updated.
- Main-memory database system are used for applications that need very fast access to data and have small enough data sizes that the entire database can fit into the main memory of a database server machine.

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records.* A record is a sequence of fields. These records are mapped onto disk blocks.
- Each file is logically partitioned into fixed-length storage units blocks, which are the units of both storage allocation and data transfer Most databases use block sizes of 4 to 8 kilobytes.
- A block may contain several records.
- We assume that no record is larger than a block.
- We require that each record is entirely contained in a single block.
- In a relational database, tuples of distinct relations are generally of different sizes. They can be stored as:
  - Fixed-length records
  - Variable-length records

Fixed-Length Records

- 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 $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - Move record *n* to *i*
  - Do not move records, but link all free records on a *free list*

Deletion of a Record 3 and Compacting

Before deletion

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

After deletion

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 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

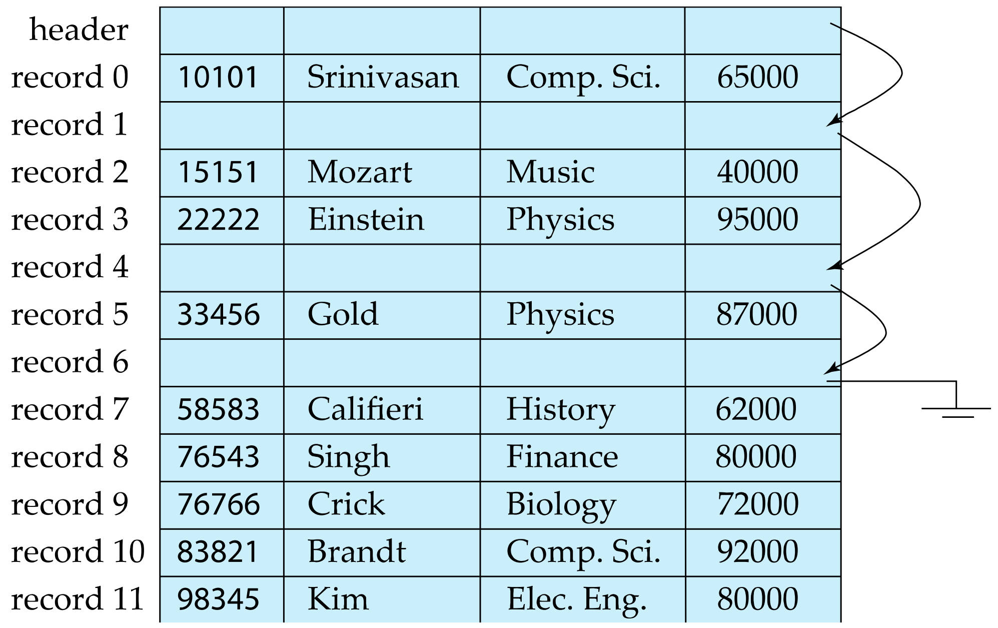
Before deletion

After deletion

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.)



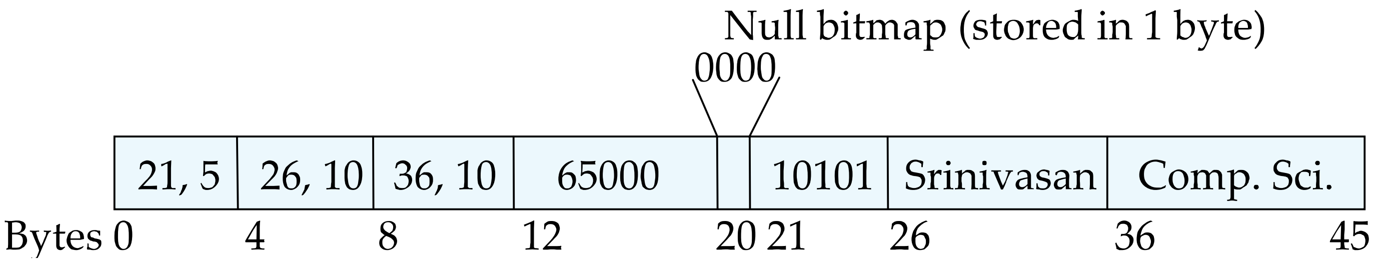
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	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).
- Different techniques for implementing variable-length records exist. Two different problems must be solved:
  - How to represent a single record in such a way that individual attributes can be extracted easily, even if they are of variable length
  - How to store variable-length records within a block, such that records in a block can be extracted easily

Variable-Length Records (Cont.)

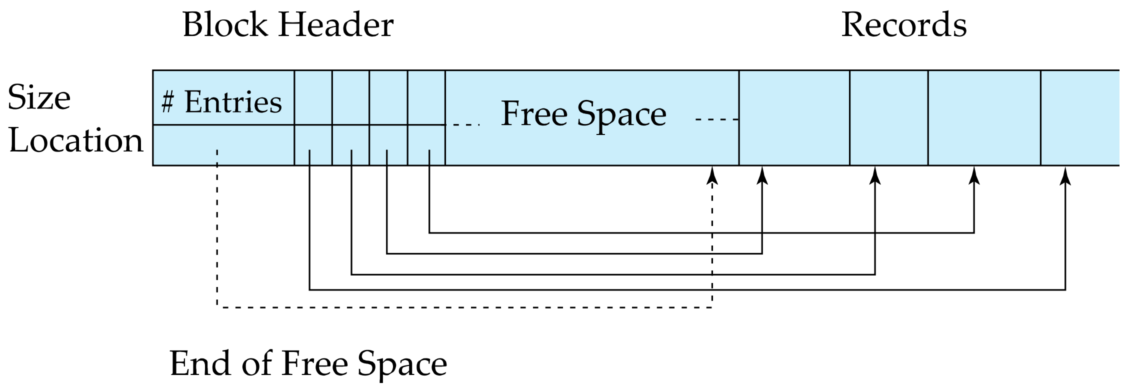
- The representation of a record with variable-length attributes typically has two parts: an initial part with fixed-length information, whose structure is the same for all records of the same relation, followed by the contents of variable-length attributes.
  - Attributes are stored in order
  - Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
  - Null values represented by Null bitmap stored in 1 byte
- An example



Storing variable-length records in a block

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.



Storing Large Objects

- Databases store data that can be much larger than a disk block.
  - An image or an audio recording may be multiple megabytes in size,
  - A video object may be multiple gigabytes in size.
- SQL supports the types **blob** (binary) and **clob** (character large objects.
- Many databases internally restrict the size of a record to be no larger than the size of a block.
- These databases allow records to logically contain large objects,
  - They store the large objects separate from the other (short) attributes of records in which they occur. A (logical) pointer to the object is then stored in the record containing the large object.

Storing Large Objects (Cont.)

- Large objects may be stored as:
  - Files in a file system area managed by the database, or
  - File structures stored in and managed by the database.
- There are some performance issues with storing very large objects in databases.
  - The efficiency of accessing large objects via database interfaces is one concern.
  - The size of database backups.
- Many applications therefore choose to store very large objects, such as video data, outside of the database, in a file system.

Organization of Records in Files

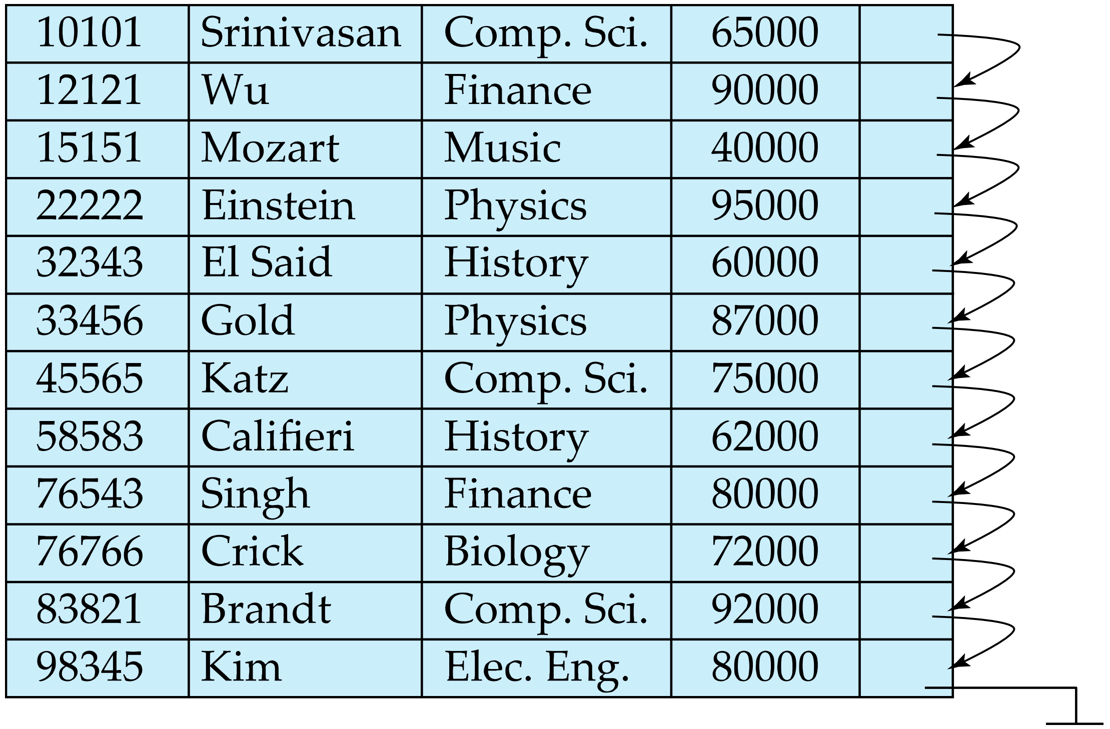
- Generally, a separate file or set of files is used to store the records of each relation.
- **Heap file organization**– a record can be placed anywhere in the file where there is space
- **Sequential file organization**– store records in sequential order, based on the value of the search key of each record
- **Multitable clustering file organization** --records of several different relations can be stored in the same file (block)
- **B+ tree file organization** -- supports ordered access to records even if there are insert, delete, and update operations, which may change the ordering of records.
- **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

Heap File Organization

- A record may be stored anywhere in he file corresponding to a relation.
- Once placed in a particular location, the record is not usually moved
- The database must keep track of all the blocks that are not used (have free space)
- **Free-space map** –a data structure to keep track on which blocks have free space.
- 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.

Sequential File Organization

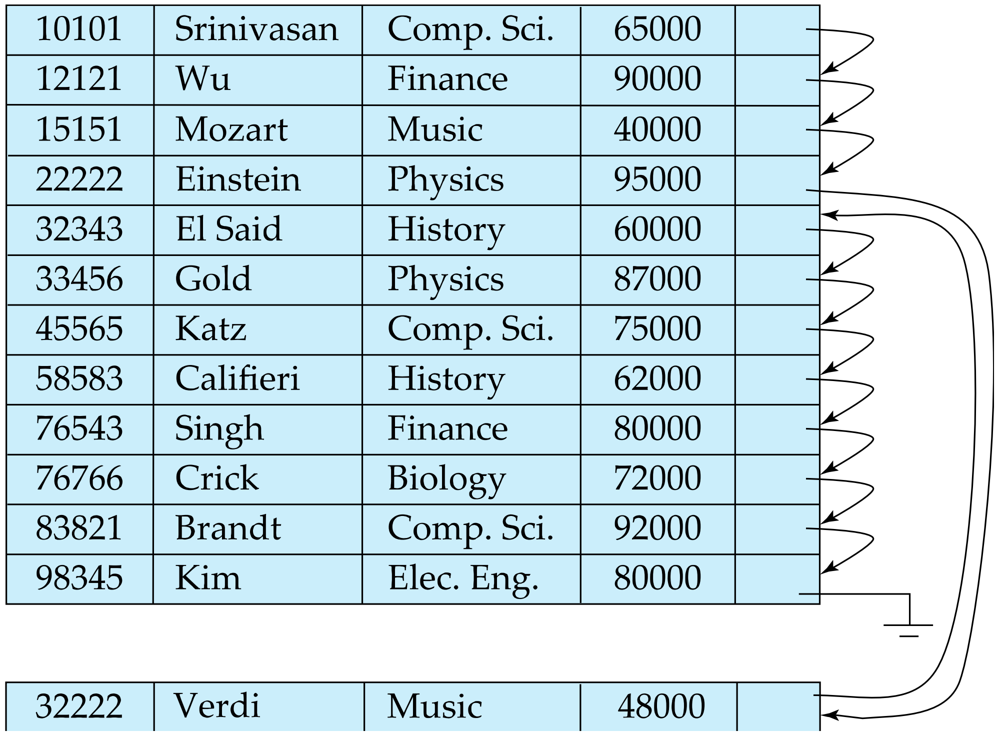
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



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 (Cont.)

- 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	

Multitable Clustering File Organization

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

department

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

instructor

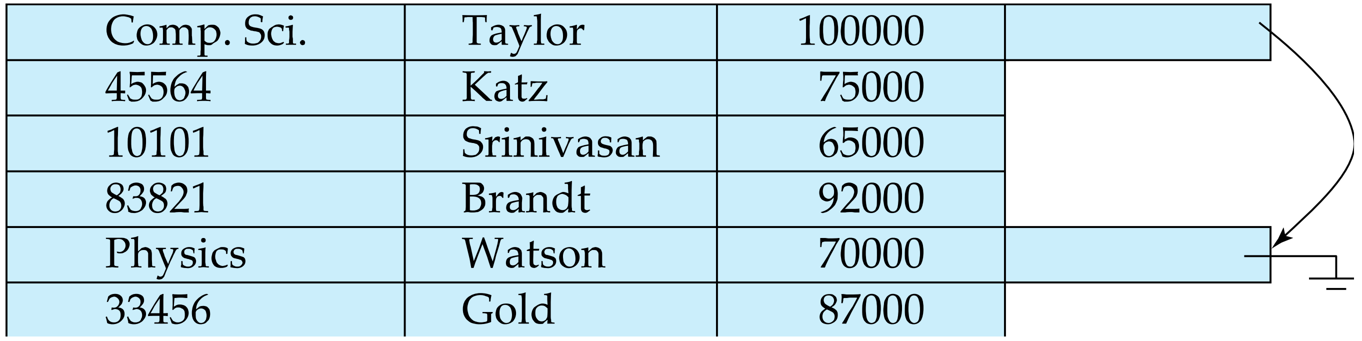
ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

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

Multitable Clustering File Organization (cont.)

- Good for queries involving *department* $\bowtie$ *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

- Many databases allow the records in a relation to be partitioned into smaller relations that are stored separately.
- **Table partitioning** is typically done on the basis of an attribute value.

DATA DICTIONARY STORAGE

Data Dictionary Storage

- **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data.
  - Information about relations
    - Names of relations
    - Names of attributes of each relation
    - Domains and lengths of attributes
    - Names and definitions of views
    - Integrity constraints
  - Information about user
    - Names of users and password
    - Information about authorizations for each user

Data Dictionary Storage (cont.)

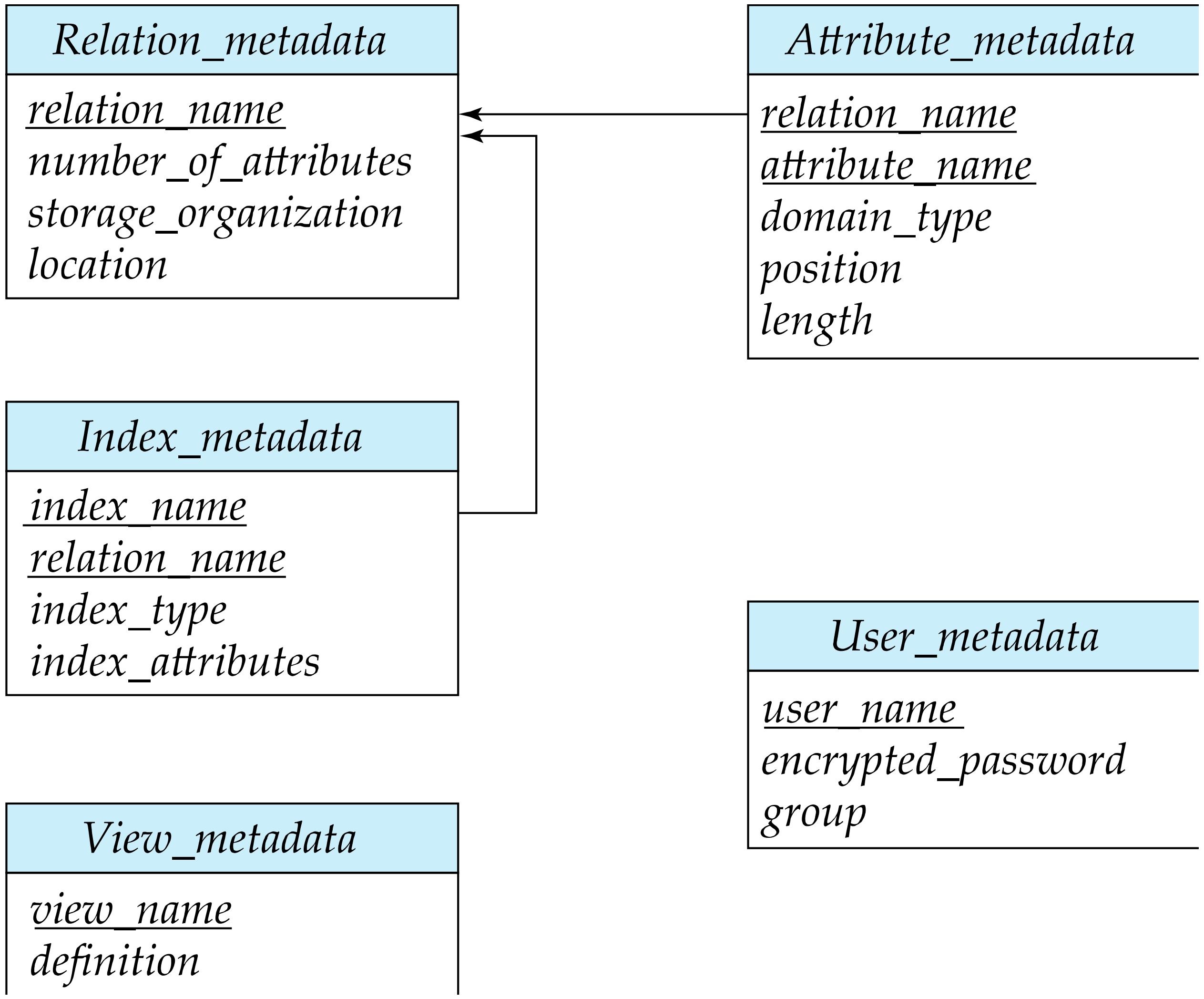
- Data dictionary (Cont)
  - Statistical and descriptive data
    - Number of tuples in each relation
  - Physical file organization information
    - How relation is stored (sequential/hash/…)
    - Physical location of relation
  - Information about indices
    - Name of the index
    - Name of the relation being indexed
    - Attributes on which the index is defined
    - Type of index formed

Metadata information Storage

- Metadata information constitutes a miniature database.
- The information about this miniature database must be stored using:
  - Special-purpose data structures and code.
  - The database as relations in the database itself.
- The benefits of using database relations to store the system metadata,
  - Simplify the overall structure of the system
  - Harness the full power of the database for fast access to system data.
  - See next slide

Relational Representation of System Metadata

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



DATABASE BUFFER

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-Replacement Policies

- Most operating systems replace the block in the buffer using the **least recently used** (LRU) strategy.
  - Idea behind LRU – use past pattern of block references as a predictor of future references
  - LRU can be bad for some queries
- 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
- Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

Buffer-Replacement Policies (Cont.)

- **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 (more in Chapter 16)

Optimization of Disk Block Access

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer before writing them to disk.
  - Non-volatile RAM: battery backed up RAM or flash memory
    - Even if power fails, the data is safe and will be written to disk when power returns
  - Disk controller writes to disk whenever the disk has no other requests or request has been pending for some time
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - *Writes to disk can be reordered to minimize disk arm movement*

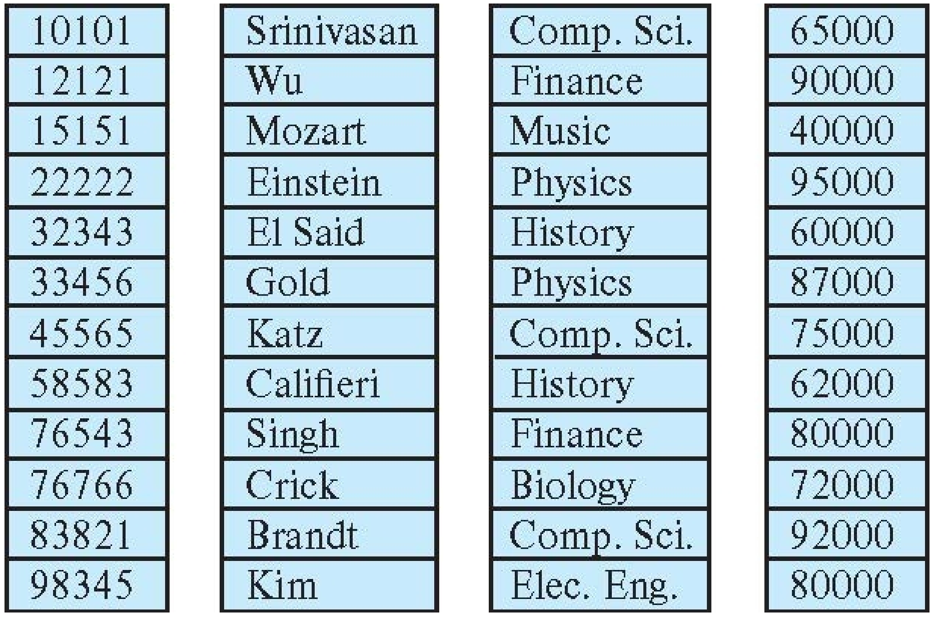
Optimization of Disk Block Access (Cont.)

- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (nonvolatile RAM)
- File systems typically reorder writes to disk to improve performance
  - **Journaling file systems** write data in safe order to nonvolatile RAM or log disk
  - Reordering without journaling: risk of corruption of file system data

COLUMN-ORIENTED STORAGE

Column-Oriented Storage

- **Row-oriented storage** - all attributes of a tuple are stored together in a record and all records are stored in a file.
- **Column-oriented storage** -- each attribute of a relation is stored separately, with values of the attribute from successive tuples stored at successive positions in the file.

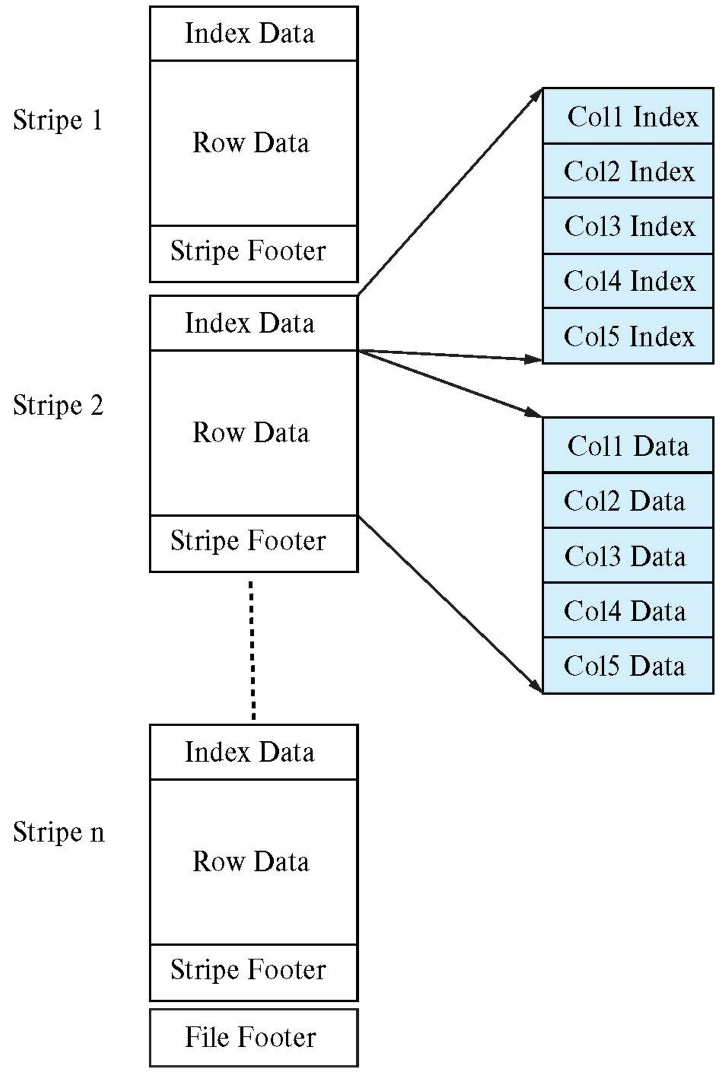


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000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

Column-Oriented Storage (Cont.)

- Column-oriented storage has the major drawback that fetching multiple attributes of a single tuple requires multiple I/O operations.
- Column-oriented storage is well suited for data analysis queries, which process many rows of a relation, but often only access some of the attributes.
  - Reduced I/O
  - Improved CPU cache performance
  - Improved compression
  - Vector processing
- Column-oriented storage is increasingly used in data warehousing applications, where queries are primarily data analysis queries.

Columnar data representation in the Orc file format



Main-Memory Databases

- Today, main-memory sizes are large enough and cheap enough, to accommodate many organizational databases that fit entirely in memory.
- Allocating a large amount of memory to the database buffer, allows the entire database to be loaded into buffer, avoiding disk I/O operations for reading data;
- Updated blocks still have to be written back to disk for persistence.
- Such a setup would provide much better performance than one where only part of the database can fit in the buffer.

End of Module 11