Fakultas Informatika
School of Computing
Telkom University

CSH2D3 - Database System

02 | Storage Management

Goals of the Meeting

01

Students knows various physical storage media.

02

Students understand how to mapping a database to files

03

Students understand how to organize records in files.

Outline

- Physical Storage Media
- File Organization
- Record Organization
- Database Buffer
- Storage Organization in Main-Memory Databases

Physical Storage Media

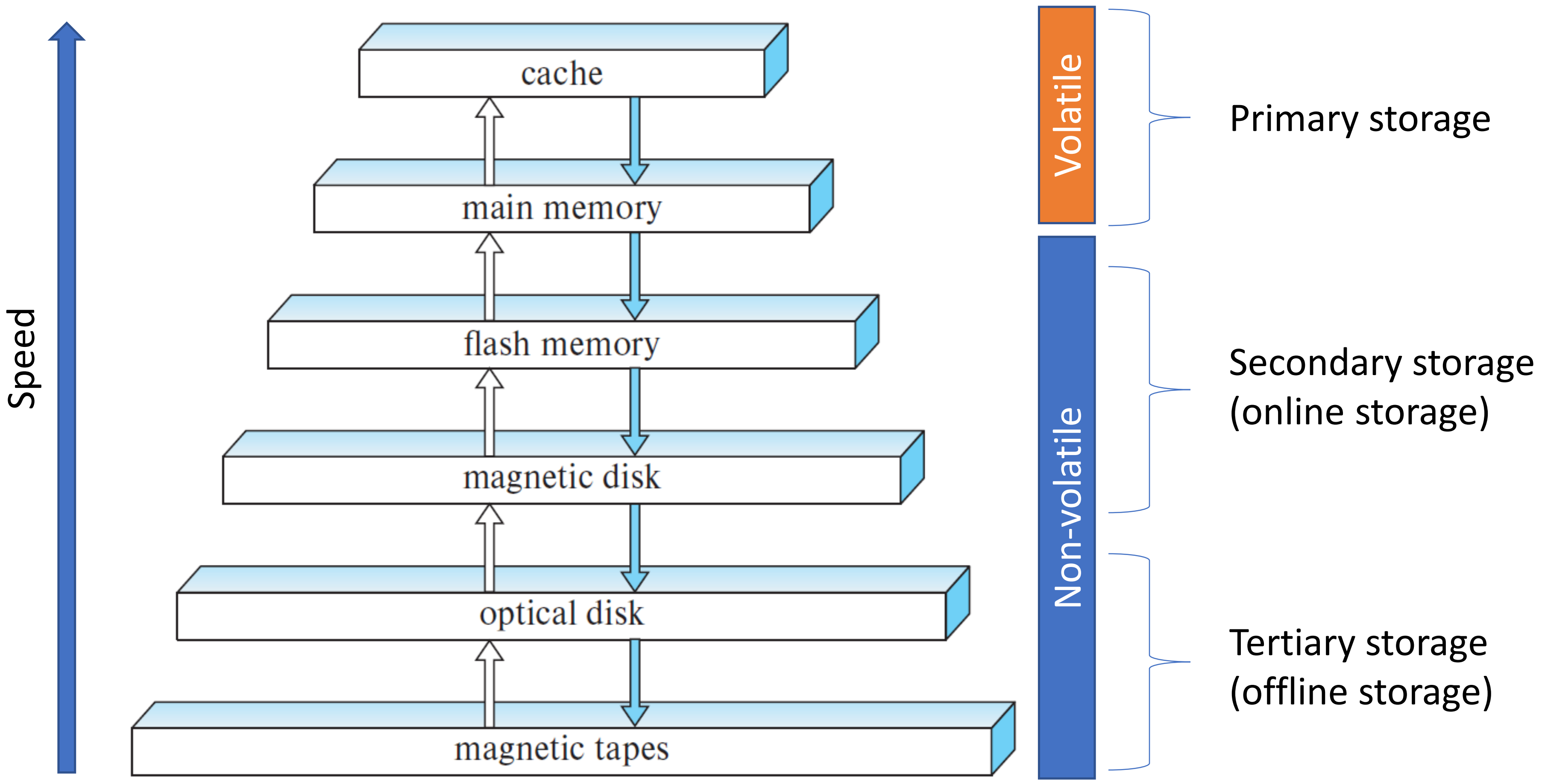
Classification of Physical Storage Media

Storage Hierarchy

Classification of Physical Storage Media

- Physical storage media can differentiate into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage**:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory.
- Factors affecting choice of storage media include
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability

Storage Hierarchy



Storage Hierarchy (Cont.)

- **Primary storage:** Fastest media but volatile (cache, main memory).
- **Secondary storage:** next level in hierarchy, non-volatile, moderately fast access time
  - Also called **on-line storage**
  - E.g., flash memory, magnetic disks
- **Tertiary storage:** lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage** and used for **archival storage**
  - e.g., magnetic tape, optical storage
  - Magnetic tape
    - Sequential access, 1 to 12 TB capacity
    - A few drives with many tapes
    - Juke boxes with petabytes (1000's of TB) of storage

File Organization

Fixed-length Record

Variable-length Record

File Organization

- A database is mapped into a number of different files that are maintained by the underlying operating system. These files reside permanently on disks.
- A file is organized logically as a sequence of records. These records are mapped onto disk blocks.
- In a relational database, tuples of distinct relations are generally of different sizes.
- There are two approaches to mapping the database to files:
  - store records of only one **fixed length** in any given file.
  - structure our files so that we can accommodate **multiple (variable) lengths** for records;
- however, files of fixed-length records are easier to implement than files of variable-length records.

Fixed-Length Records

As an example, let us consider a file of *instructor* records for our university database *instructor* records for university database.

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

- Assume that each character occupies 1 byte and that numeric (8,2) occupies 8 bytes.
- We allocate the maximum number of bytes that each attribute can hold.
- Then, the *instructor* record is 53 bytes long.
- A simple approach is to use the first 53 bytes for the first record, the next 53 bytes for the second record, and so on.

Fixed-Length Records

Problem 1 :

Record access is simple but records may cross blocks (unless the block size happens to be multiple of the record size)

Modification: do not allow records to cross block boundaries

Problem 2:

Difficult to delete a record from this structure.

Alternatives*:*

- move records after the deleted record into the space formerly occupied by the deleted records, an so on.
- move the last record into the space formerly occupied by the deleted records
- do not move records, but link all free records on a *free list*

Fakultas Informatika
School of Computing
Telkom University

Example : File containing instructor records

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**

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

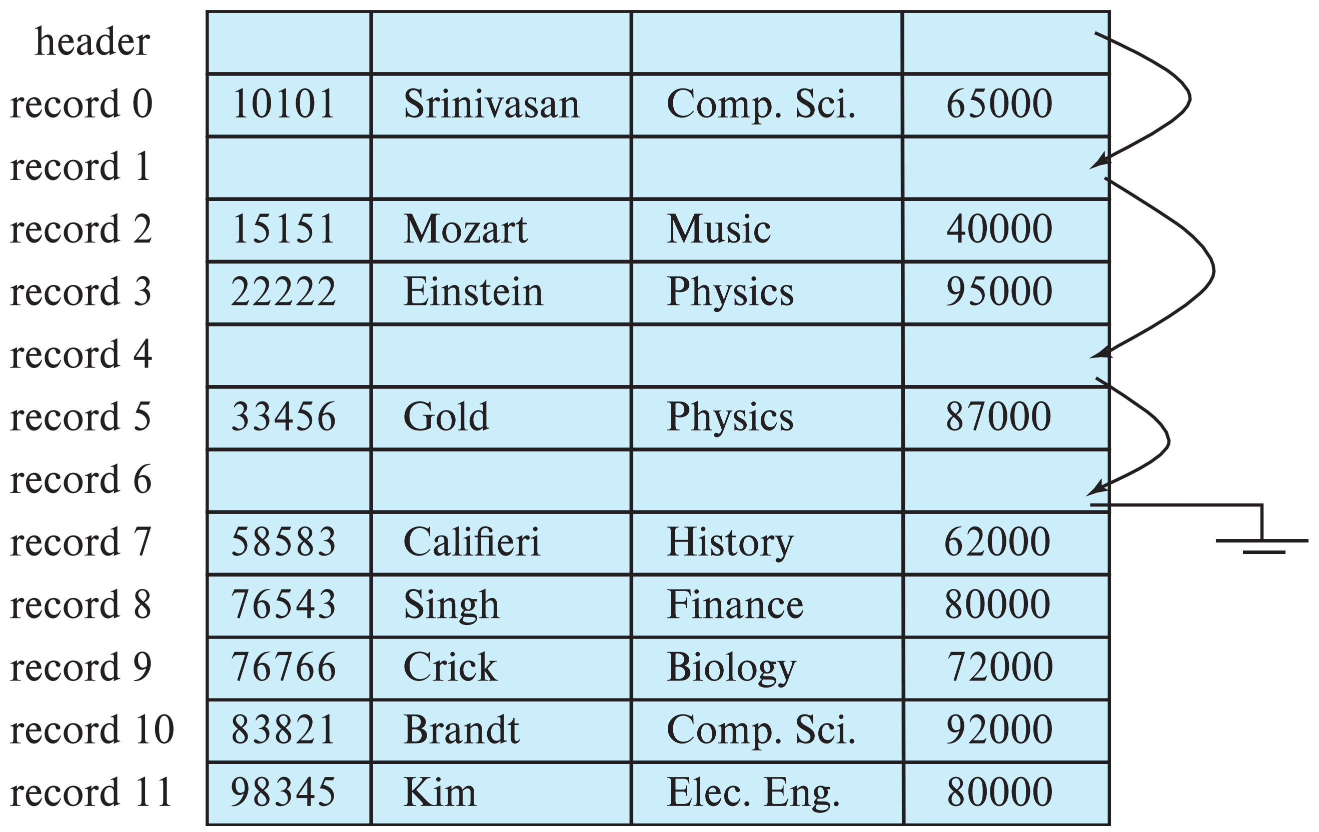
- Deletion of record *i:* alternatives*:*
  - **move record *n* to *i***

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

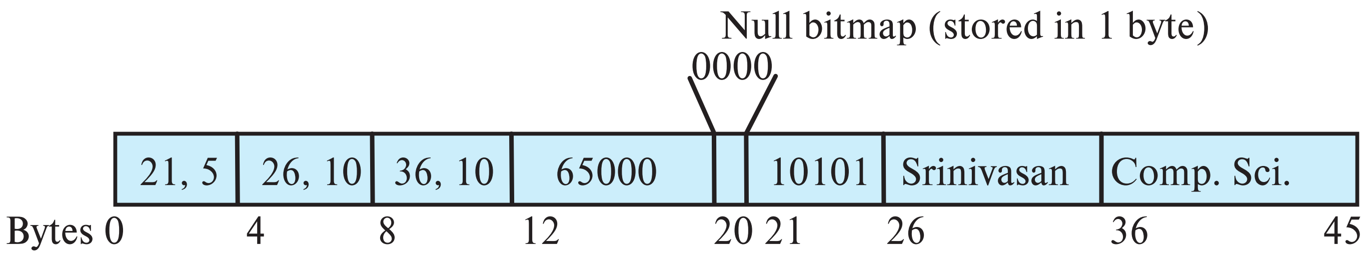
- Deletion of record *i:* alternatives*:*
  - **do not move records, but link all free records on a *free list***



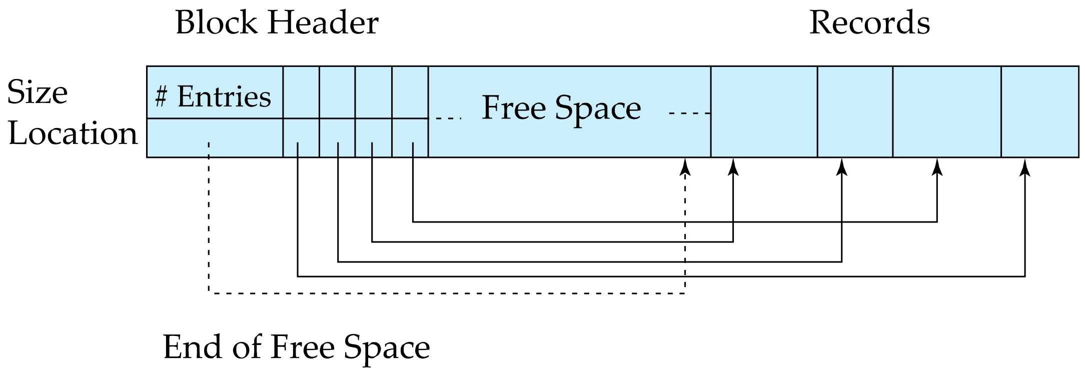
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology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

Blocking

- A Block is a transfer data unit between secondary and primary devices
- Block Size in byte (B)
  - Oversize block size will also transfer unnecessary data in it → needs bigger memory allocation
  - Undersize block size will cause repeatable read
- Records are stored in block.
- Blocking is a method to arrange records in a block.
- Blocking Factor (Bfr) : Number of records in a block

Fixed Blocking

- Number of fixed length records in all blocks remain the same
- Record length <= Block size
- Blocking factor Bfr = $\left\lfloor \frac{B}{R} \right\rfloor$

- Let Block Size B = 100 Byte, R = 30 Byte → Bfr = 3
- Block size – allocated space = wasted space

Variable Blocking

Spanned

- Used in variable length record
- Record can be split across multiple blocks
- $R \leq B \leq R$
- Minimal waste. Difficult to be implemented
- Will cause block chain → slower reading time

Unspanned

- Variable length record size
- A record cannot be spanned across multiple blocks
- Waste space probability is high
- Record length <=block size

Variable Blocking

Spanned

- Let P as block pointer
- Effective block size = B-P
- Record size + marker = R + M
- Bfr = $\left\lfloor \frac{(B-P)}{(R+M)} \right\rfloor$
- If M = P,

  Bfr = $\left\lfloor \frac{(B-P)}{(R+P)} \right\rfloor$

Unspanned

- Average block waste = ½ R
- No block pointer available
- Effective block size = B - ½ R
- Bfr = $\left\lfloor \frac{(B-\frac{1}{2}R)}{(R+M)} \right\rfloor$

Organization of Records in Files

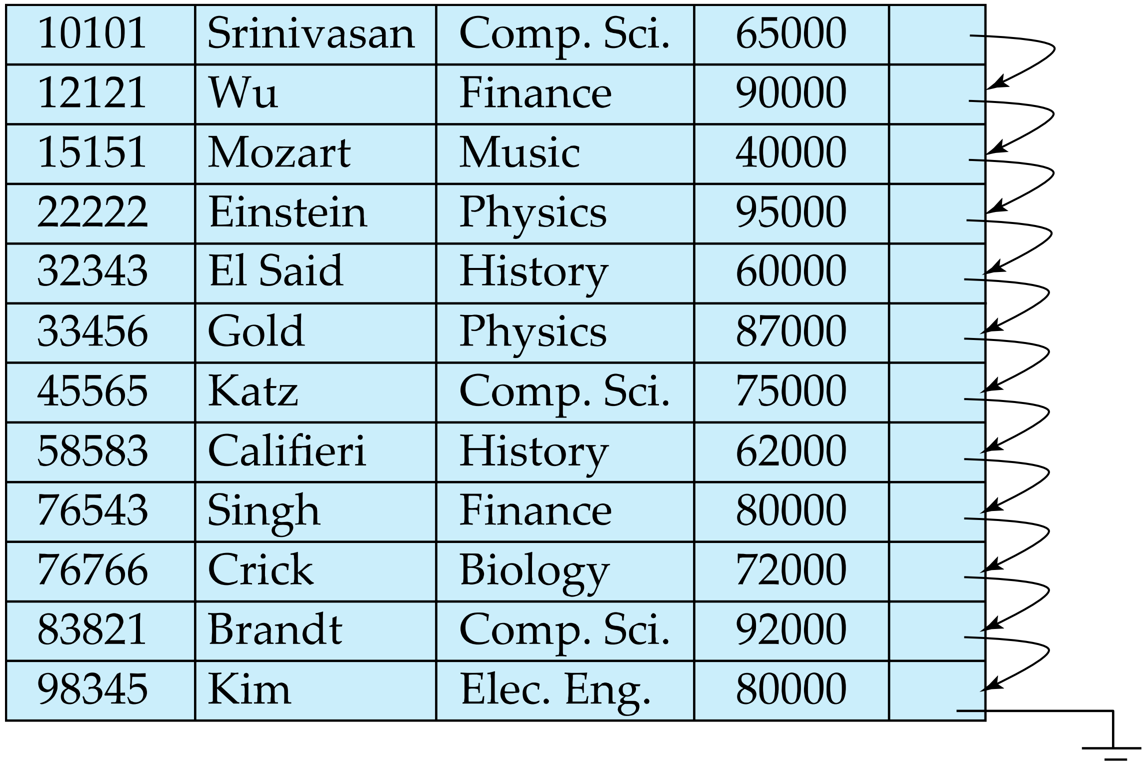
- Heap
- Sequential
- Multitable Clustering
- B+-tree
- Hashing

Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space. Records usually do not move once allocated.
- **Sequential** – store records in sequential order, based on the value of the **search key** of each record
- 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
- **B$^+$-tree file organization**
  - Ordered storage even with inserts/deletes
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed

Sequential File Organization

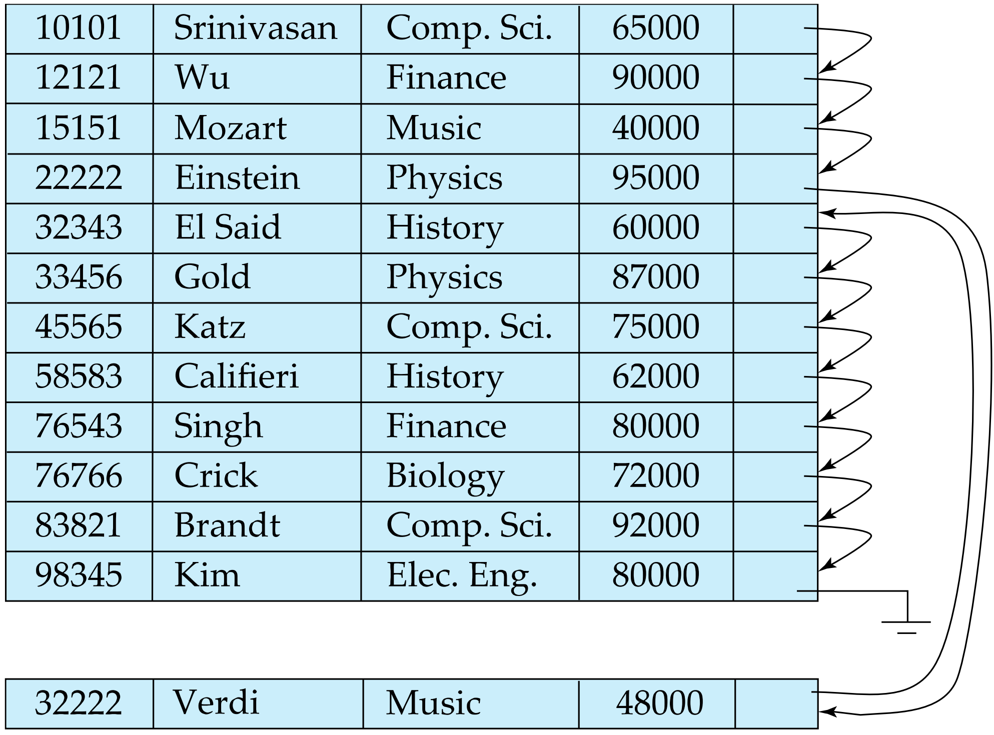
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



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

Data Dictionary Storage

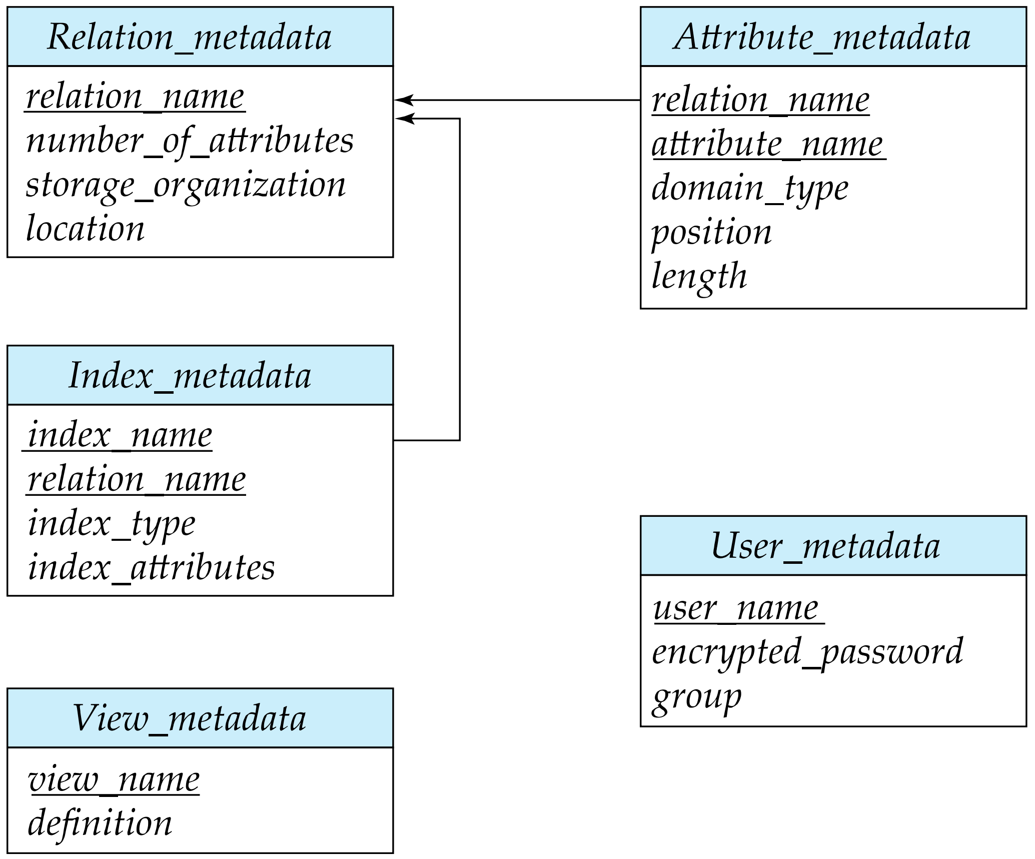
Data Dictionary Storage

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



Database Buffer

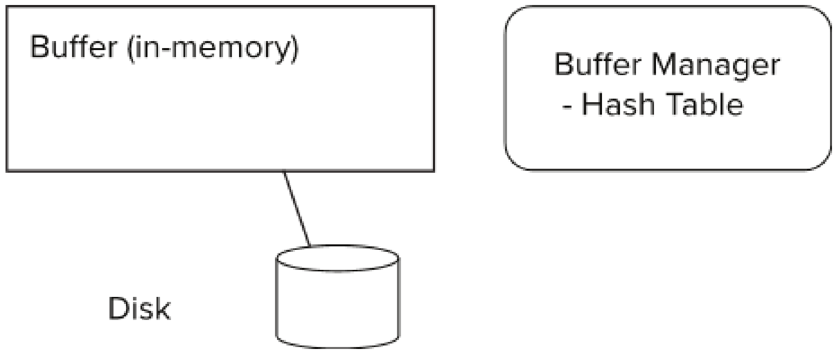
Buffer

Buffer Manager

Fakultas Informatika
School of Computing
Telkom University

Database Buffer

- 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.

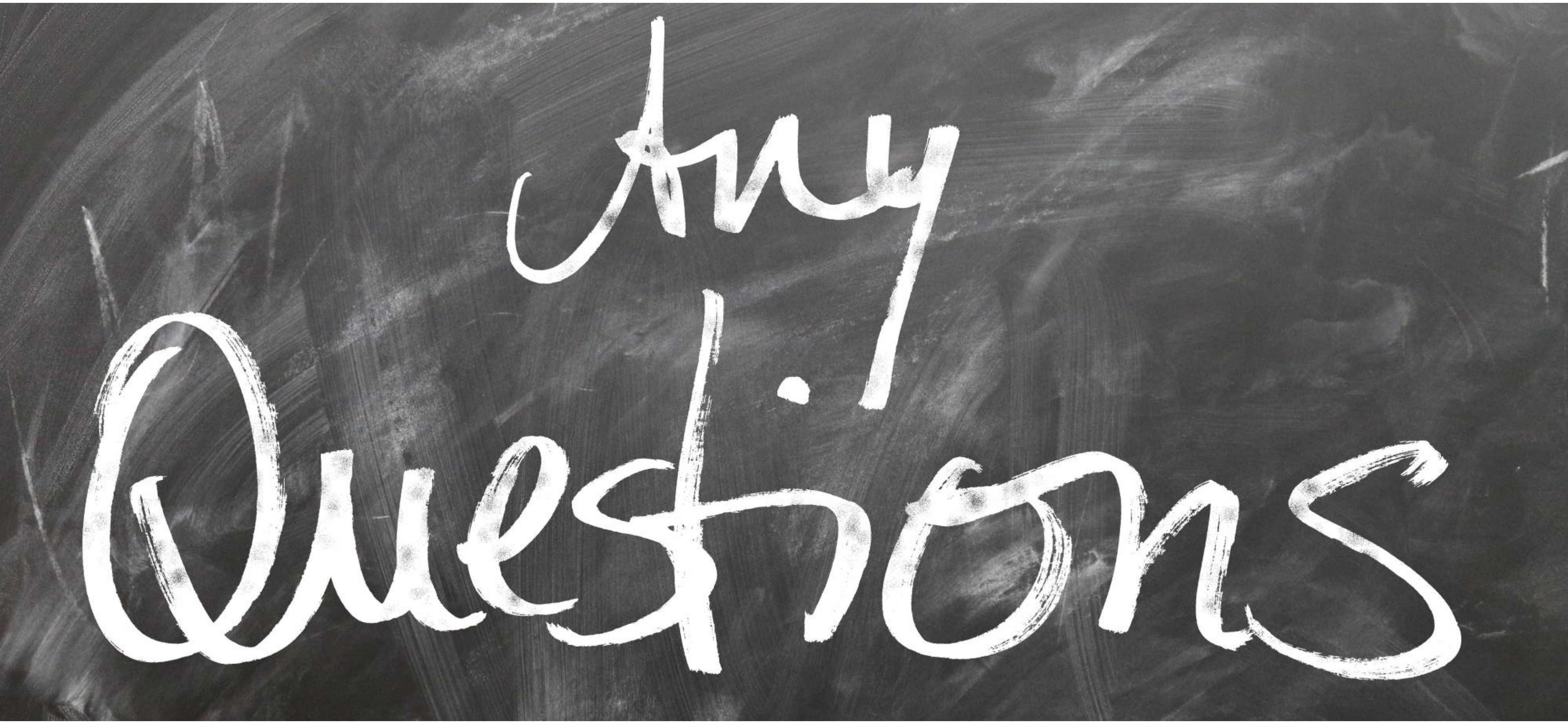
- If the block is already in the buffer, buffer manager returns the address of the block in main memory
- If the block is not in the buffer, the buffer manager:
  - Allocates space in the buffer for the block
    - Replacing (throwing out) some other block, if required, to make space for the new block.
    - 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.

References

Silberschatz, Korth, and Sudarshan. *Database System Concepts* – 7th Edition. McGraw-Hill. 2019.

Slides adapted from Database System Concepts Slide.

Source: https://www.db-book.com/db7/slides-dir/index.html


Any
Questions