CSc 134 Database Management Systems

8. Disk Storage

Ying Jin
Computer Science Department
California state University, Sacramento

Database files

- Databases are stored physically as files of records
- Storage in disk
- How to organize file storage?
- How to access files efficiently?

Physical Storage

- Primary Storage
  - Storage media accessed by CPU directly
- Secondary storage
  - Data must be copied into primary storage, then the data can be processed by CPU
  - Large capacity
  - Low cost

Database Storage

- Data stored on disk is organized as files of records.
- A record is a collection of data values about entities and relationships.
- Records should be located efficiently
- Primary file organization
  - How to placed files of records on the disk physically.
  - How to access the records.
  - Heap file (unordered files), sorted file, hashed file, B-tree
- Secondary organization
  - Use alternate fields (other than the fields for primary file organization) to locate records. e.g. index. Ch14

Disk block

- Set by OS during disk formatting
- Block size B=512 to 4096 bytes
- Blocks are the units transferred between disk and main memory for processing

Record

- Data is stored in the form of records.
- A record consists of a collection of related data values or items.
- Record

  Multiple fields

  <field name, field type>
- e.g. Employee record type

  employee {SSN char, name char}.
- A file is a sequence of records

Fix length Records

- Fix-length records:
  - Every record in the file has exactly the same size
  - e.g. a record size of 71 bytes

Variable-length records

- Same record type, but
  - variable-length fields
  - repeating field: multiple value field for some records
  - optional fields

Record blocking

- Records are allocated to disk blocks
- One block: many record
- Block size: B, record size R
- blocking factor bfr=$\lfloor B/R \rfloor$
- Floor function: rounds down to an integer
- unused space= B – (bfr * R) bytes

Spanned Record

- To utilize unused space
  - A pointer at the end of the first block points to the block containing the remainder of the record
- Must Use spanned organization when a record is larger than one block
- Figure

Unspanned

- Records are not allow to cross block boundaries
- Used in fix-length record
- Make each record start at a known location
- Variable-length records can be spanned or unspanned.
- Figure

File headers

- Also named file descriptor
- Info about
  - disk addresses of the file block
  - record format descriptions
    - field lengths and order of fields within a record for fixed-length unspanned records
    - field type codes, separator characters, and record type codes for variable-length records.

Operations on file

- Open
  - Prepare the file for reading or writing
  - Retrieve file header
  - Set file pointer to the beginning of the file
- Read (or Get)
  - Copies the current record from the buffer to a program variable in the user program
  - Current record pointer to the next record

Operations on file (cont.)

- Find (or Locate)
  - Searches for the first record that satisfies a search condition
  - Transfers the block into main memory buffer if not there
- findNext
  - Search for the next record satisfies the condition
  - Transfers the block into main memory buffer if not there

Operations on file (cont.)

- Delete
  - Delete the current record
  - Eventually updates the file on disk to reflect the deletion
- Modify
  - Modifies some field values for the current record
  - Eventually update disk to reflect modification

Operations on file (cont.)

- Insert
  - Locate the block to be inserted
  - transfer that block into a main memory buffer
  - insert the record into the buffer
  - eventually writing the buffer back to disk to reflect the insertion
- Close
  - Release the buffers
  - Perform any other needed cleanup operations
- Reset
  - Sets the file pointer of an open file to the beginning of the file.

Operations on file (cont.)

- FindAll
  - Locates all records satisfy a condition
- Find n
  - Search for the first record satisfies a search condition
  - Locate the next n-1 satisfied record
  - Transfer the n record to main memory buffer

Operations of files (Cont.)

- FindOrdered
  - retrieves all the records in the file in some specific order
- Reorganize
  - Start the reorganization process
  - Some file organization require periodic reorganization
  - e.g. reorder on a specified field

Files of unordered records

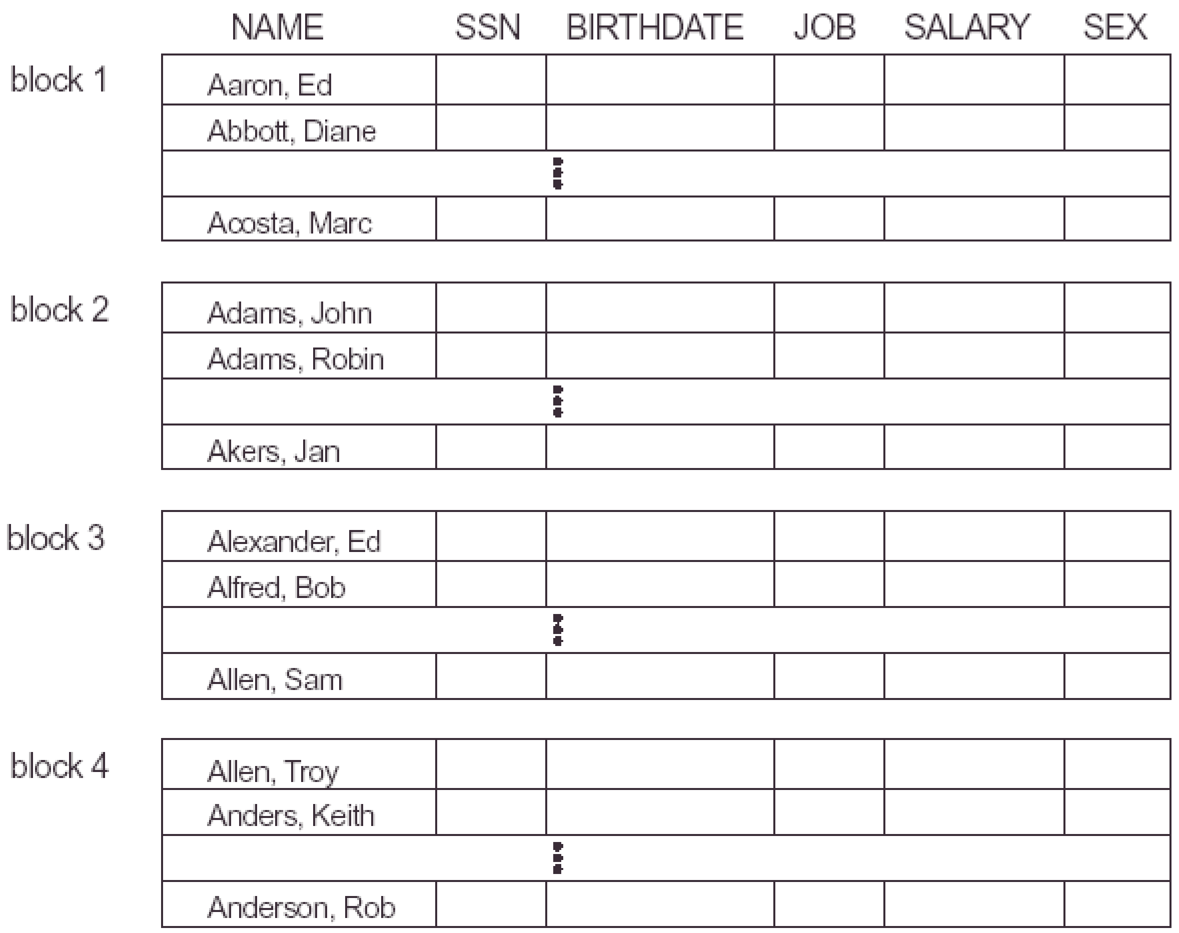
- Also called a *heap* or a *pile* file.
- New records are inserted at the end of the file.
- Record insertion is quite efficient.
- Search for a record
  - a *linear search* through the file records is necessary.
  - requires reading and searching half the file blocks on the average, and is hence quite expensive.

Files of unordered records (cont.)

- Delete
  - Leave deleted space empty
  - Deletion marker with each record
    - Only search/read valid record
  - Waste of storage space → requires periodic reorganization
- Unordered file can be
  - Spanned or unspanned
  - Fixed-length or variable-length

Sorted Files

- Files of ordered records
- File records are kept sorted by the values of an *ordering field*.
- A *binary search* can be used to search for a record on its *ordering field value*.
    - This requires reading and searching $\log_2$ (b) of the file blocks
    - An improvement over linear search (b/2).



	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					

Sorted Files (Cont.)

- Reading the records in order of the ordering field is quite efficient.
- Provide no advantage on searching non-ordering field
- Inserting and deleting is expensive (figure)
- Deleting: deletion markers, periodic reorganization
- Make insertion more efficient (figure)
  - main/master file
  - overflow/transaction file

- Hashing Techniques

Hashing function

- h (hash field) = address of disk block
- Figure
- e.g. h(K) = K mod M
- Noninteger hash field values can be transformed into integers before the mode function is applied

Collision

- Most hash functions do not guarantee that distinct values will hash to distinct addresses
- Collision: Hash to an address that already contains a different record
- Collision resolution
  - Chaining
    - A pointer field is added to each record location
    - Point to a overflow location

External Hashing

- Hashing for disk files is called external hashing
- Bucket: one disk block or a cluster of contiguous blocks
- h(key)=relative buck #
- A Table in file header (relative bucket #, corresponding block address)
- Figure

Overflow of bucket

- Record pointer: a block address and a relative record position
- Figure

Operations on external hashing

- Search for non-hash field is expensive
- Deletion
  - Remove from bucket, move a overflow record to the bucket
  - Remove from a overflow bucket. Maintain a linked list of unused overflow locations
- Modify
  - search to locate the record
  - Modify the record
    - Non-hash filed: change it, rewrite it in the same bucket
    - hash field: move to another bucket: deletion of the old , insert a new.

Problems of Static hashing

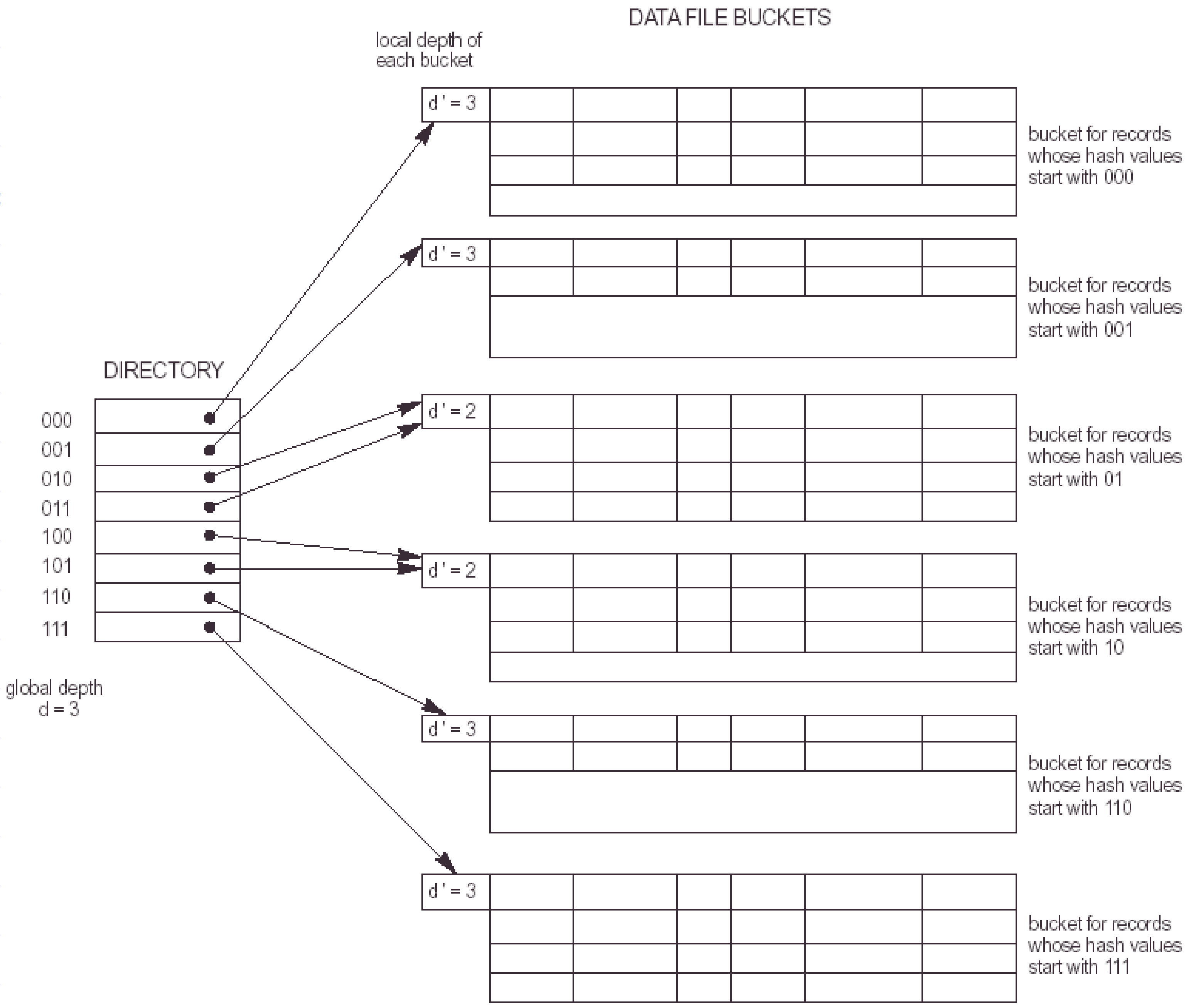
- Fix number of buckets M
- m: maximum # of records per bucket
- At most (m*M) records will fit in the allocated space.
- More record → collision → long lists of overflow records → retrieve slow down →chose new h based on M → reorganize → time consuming

Hashing techniques allow dynamic file expansion

- Allow the dynamic growth and shrinking of the number of file records.
- h(field) =result (e.g. 3)
- binary representation (a string of bit) of the result. (e.g. 011)
- Extendible hashing
- Linear Hashing

Extendible Hashing

- Directory
  - an array of 2^d bucket address
  - d: global depth
- Figure
- Local depth d’
  - stored with each bucket
  - specify the number of bits on which the bucket contents are based


DATA FILE BUCKETS
local depth of each bucket
DIRECTORY
000
001
010
011
100
101
110
111
global depth d = 3
d ' = 3
bucket for records whose hash values start with 000
d ' = 3
bucket for records whose hash values start with 001
d ' = 2
bucket for records whose hash values start with 01
d ' = 2
bucket for records whose hash values start with 10
d ' = 3
bucket for records whose hash values start with 110
d ' = 3
bucket for records whose hash values start with 111

Overflow

- insert a new record → overflow in a bucket
  - d'<d. figure
  - d'=d. figure
    - Double directory
    - Use extra bit to distinguish the two new buckets.

- Two-Phase, Multiway Merge Sort

Two Phases

- Phase 1
  - Read in and sort main-memory-size pieces of the data
  - Write each *sorted sublist* to disk
- Phase 2
  - Merge all the sorted sublists into a single sorted list

Phase 2

Read the first block of each sorted sublist into a main-memory buffer.

1. Find the smallest key among the first remaining elements of all the lists
2. Move the smallest element to the first available position of the output block
3. If the output is full, write it to disk and reinitialize the buffer
4. If the block from which the smallest element was just taken is now exhausted of records, read the next block from the same sorted sublist into the same buffer.

   If no blocks remain, then leave its buffer empty and do not consider elements from that list in any further competition for smallest remaining elements.

These slides are based on the textbooks:

R. Elmaseri and S. Navathe, *Fundamentals of Database Systems*, 7th Edition, Addison-Wesley.

H. Garcia-Molina, J. Ullman, J. Widom, Database System Implementation, 1st Edition, Prentice Hall