Data Storage and Indexing

Lecture Handout

Dr Eugenia Ternovska

Simon Fraser University

Files of Records

Each table is stored on disk in a file of records - an important abstraction in DBMS

Record: memory area (sequence of bits) logically divided into **fields**

Each record in a file

- corresponds to a row of values in the table
- has the same number of fields
  but **not necessarily the same length**
- has a unique identifier: the record id (**rid**)

File of records

Supports the following operations:

- Insertion of records
- Deletion of records
- Modification of records
- Scan of all records, returned one at a time

Files of records and pages

Files of records are logical collections of information. They store tables. Files of records are organized in **pages**.

A page is a unit of information read from or written to disk.

When data is requested for computation
pages must be fetched from disk and loaded in main memory

Page size is a DBMS parameter. Typical size is 4-8 KB

every record in a file has a unique rid
every page in a file is of the same size

We will study how a **file of records** can be organized
as a **collection of pages**

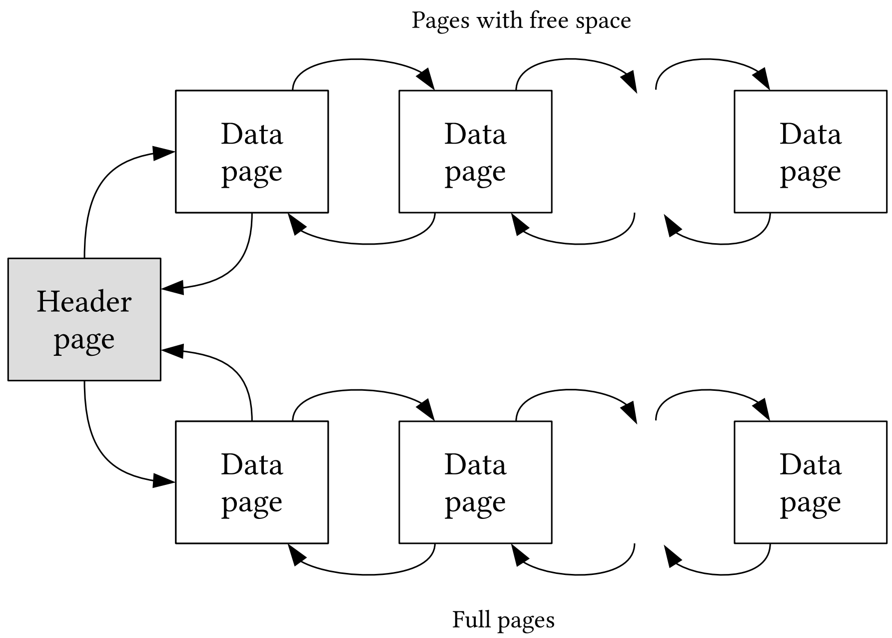
File organization

Simplest structure: unordered file, called **heap file**

- records are stored in random order across the pages
- supports retrieval of a specific record given its rid

Indexed structures allow to efficiently retrieve records
that satisfy a given search condition

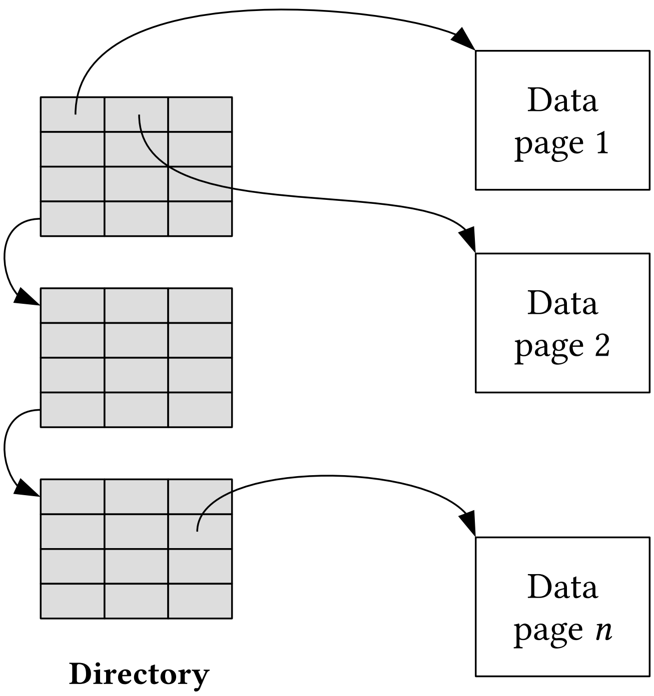
Implementing heap files: Linked list of pages



Disadvantages

- Almost all pages on free list if records are of variable length
- Must scan and examine several pages to insert a record

Implementing heap files: Directory of pages



Free space can be managed by maintaining:

- a bit per entry (free space yes/no)

or

- a count per entry (amount of free space)

Page formats

A page can be thought of as a collection of slots

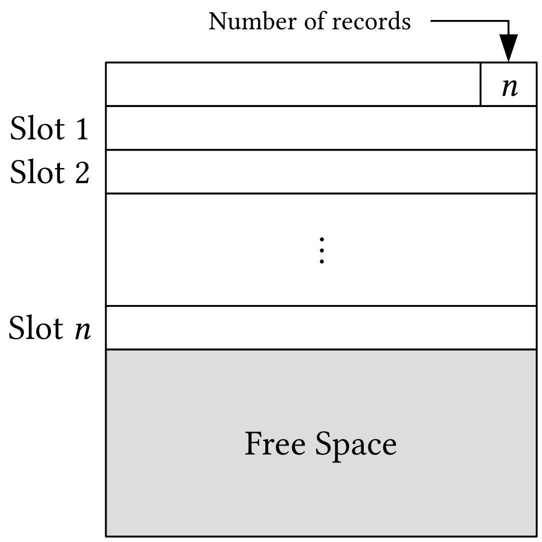
- a record is identified by the **page id** and **slot number** so rid = (page id, slot number)
- alternative: assign unique integer to each record and maintain correspondence between rid and (page, slot)

Format of pages depends on:

- Fixed- vs. variable-length records
- Support for search, insertion, deletion of records

Page formats for fixed-length records

Packed



Records stored in the first n slots

Records located by offset calculation

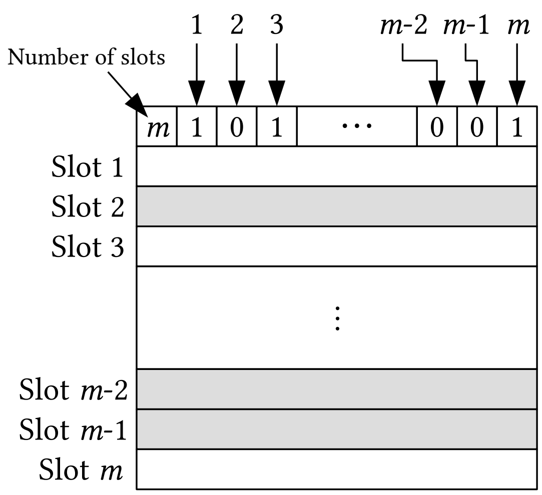
Free space contiguous at the end

When a record is deleted,
the last one is moved to empty slot

Problem if rid contains slot number
– does not work if there are external references to the record that is moved

Page formats for fixed-length records

Unpacked, Bitmap



Bit array tells which slots are free

Records located by offset calculation

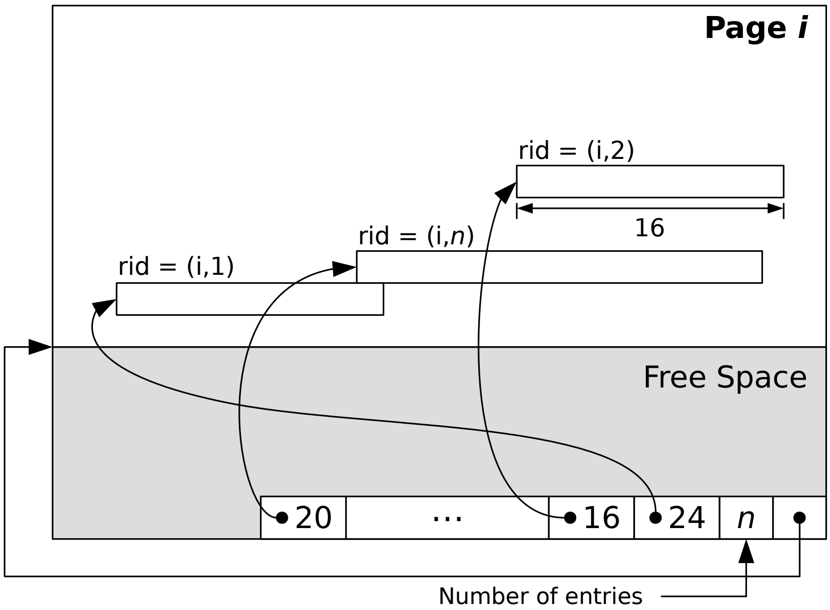
Scanning all records requires
bit array scan + offset calculation

Insertion of record requires
bit array scan + offset calculation

When a record is deleted,
corresponding bit is turned off

A page usually also contains the id of the next page in the file (for both packed and unpacked formats)

Page format for variable-length records



Directory of slots: ⟨*record offset, record length*⟩ per slot
record offset is a pointer, an offset from the start of the data area on the page. Delete record = set the offset to -1

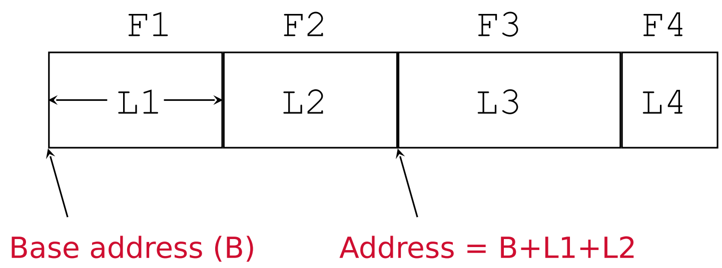
Page format with directory of slots

- Most flexible format
- **Records can be moved without changing rid**
- Can be used also for fixed-length records
- Deletion accomplished by setting slot offset to -1
- Records can be moved around on the page because the rid does not change when the record is moved (only the offset changes)
- Free space must be managed more carefully (the page is not pre-formatted into slots)
- On insertion of record, look for slot entry with offset -1 (if there is none, add new entry to slot directory)

Record Formats: Fixed-length records

Each field has a **fixed** length (available in the system catalog)

Given the base address B of the record,
the address of field i can be calculated as $B + \sum_{k=1}^{i-1} L_k$

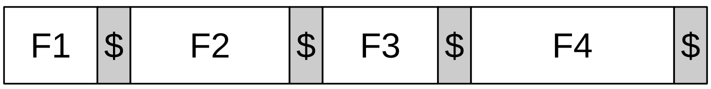


Direct access to fields, but inefficient storage (especially for nulls)

Record Formats: Variable-length records

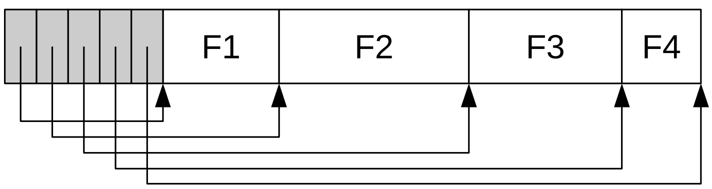
Some of the fields have **variable** length

Fields delimited by special symbol



Access to fields requires a scan of the record

Array of field offsets



Direct access to fields ; efficient storage of nulls

Modifying fields in a record

Potential issues with variable-length records:

- When field grows, shift subsequent fields to make space
- If modified record does not fit in the free space on page, move it to another page leaving a **forwarding address** (we must allocate minimum space for every record)
- If modified record does not fit on any page, split into smaller records across several pages using pointers

Adding fields can cause similar issues in all record formats

Indexing

Index

a data structure that organizes data records on a disk based on a **search key** (any subset of the fields of a relation)

- is used to optimize certain kinds of retrieval operations
- supports efficient retrieval of all data records satisfying a given **condition on the search key**

We can create multiple indexes with different search keys.

Data Entry

Data entry - a record stored in an index file

$k*$ - data entry with search key value k

contains enough information to locate (one or more) *data records* with search key value k.

We can efficiently search an index file to find data entries, then use them to obtain *data records*, if different from data entries

Data Entries: 3 Alternatives

what to store as a data entry in an index:

Alternative 1 A data entry $k*$ is an **actual data record**, with search key value k. It is a special file organization called *indexed*. Can be used **instead** of sorted or unordered file of records.

Alternative 2 A data entry is $\langle k, \textbf{rid} \rangle$.

Alternative 3 A data entry is $\langle k, \textbf{rid-list} \rangle$.

(2) and (3) are independent of the file organization.

(3) has better space utilization, but data entries are variable in length, depending on a number of records for a key.

Indexing Strategies

Two main indexing strategies

- Hashing (good for conditions based on equality)
- Trees (good for conditions based on ordering)

Hash-based indexing

Organize records into **buckets**

based on a **hash function** h applied to the search key value

For record $\bar{r}$ and search key k

$$h\big(\pi_k(\bar{r})\big) = \text{bucket where } \bar{r} \text{ belongs}$$

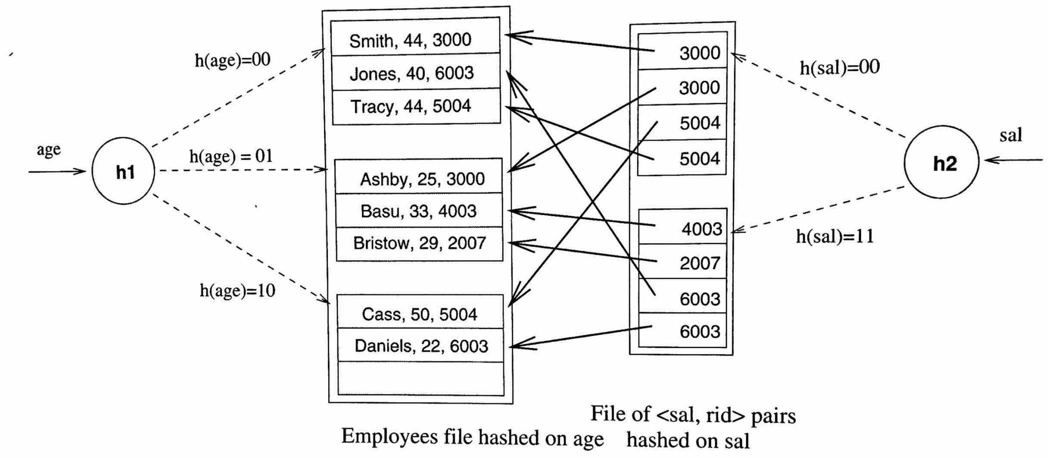
Bucket = primary page + zero or more **overflow** pages in a chain

Given a bucket number, the hush-based index structure allows us to retrieve the primary page from the disc in one or two I/O

Hash-based indexing: Example

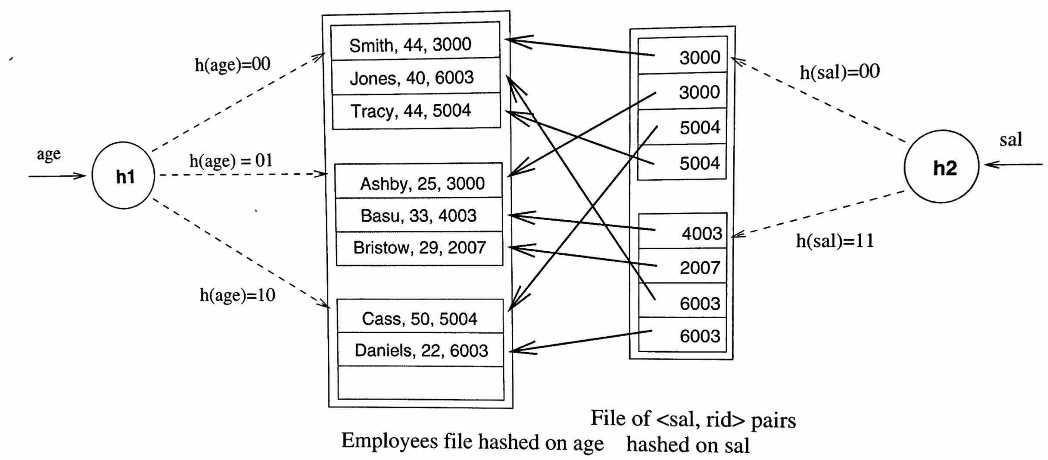
Left: hashed on **age**.
The **data entries are actual data records** (Alternative 1)

h: converts the search key value to its binary representation and uses the two least significant bits as **bucket identifier**



Hash-based indexing: Example (Cont.)

Right: hashed on **sal**.
The **data entries are** $\langle sal, \text{rid}\rangle$ (Alternative 2),
where rid is a pointer to a **record** with search key value sal



Tree-based indexing

a tree-like data structure to direct search for data entries

Allows one to efficiently locate all data entries with search key values in a desired **range**

Records are organized using a **hierarchical tree structure** that directs the search from the root to relevant pages

Non-leaf nodes contain pointers p separated by search key values v

$$p_0, v_1, p_1, v_2, p_2, \ldots, v_n, p_n$$

For each value v_i

- p_{i-1} points to a node with values less than v_i
- p_i points to a node with values greater than or equal to v_i

Leaf nodes are pages of data records

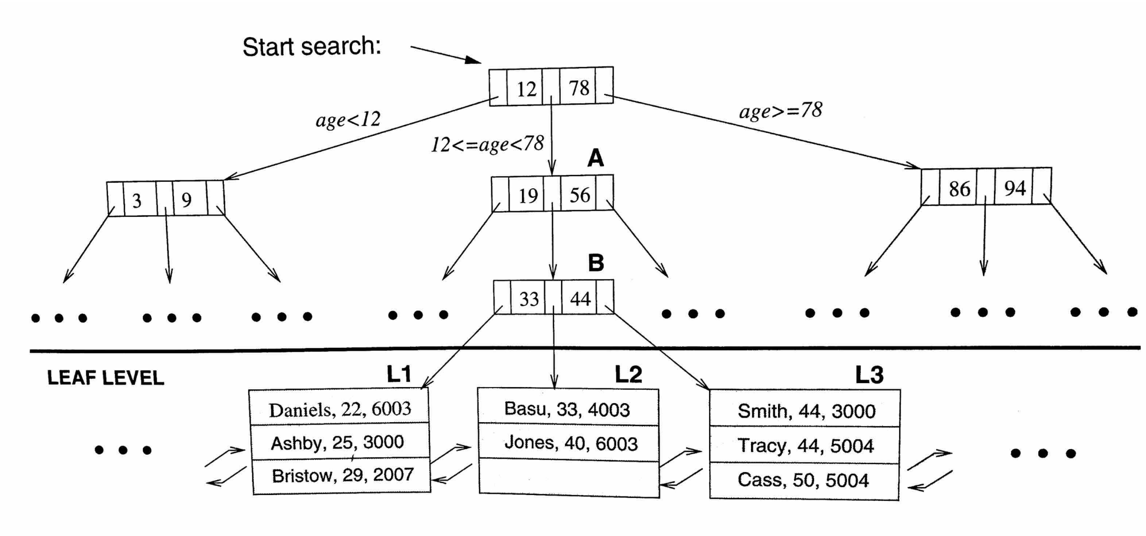
B-tree

Balanced tree: all paths from root to leaves have same length

Tree-based indexing: Example for $24 < age < 50$

Fun-out is the average number of children.

Search key: **age**. Search begins at the root.



number of disk I/O = length of path to a leaf + number of leaf pages with qualifying data entries

Acknowledgements

[1] Database Systems: The Complete Book, 2nd Edition Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer WidomPrentice Hall, 2009

[2] Database System Concepts, Seventh EditionAvi Silberschatz, Henry F. Korth, S. SudarshanMcGraw-Hill, March 2019www.db-book.com

Additional references and resources used in preparation of this course are listed on the course webpage or mentioned in slides.