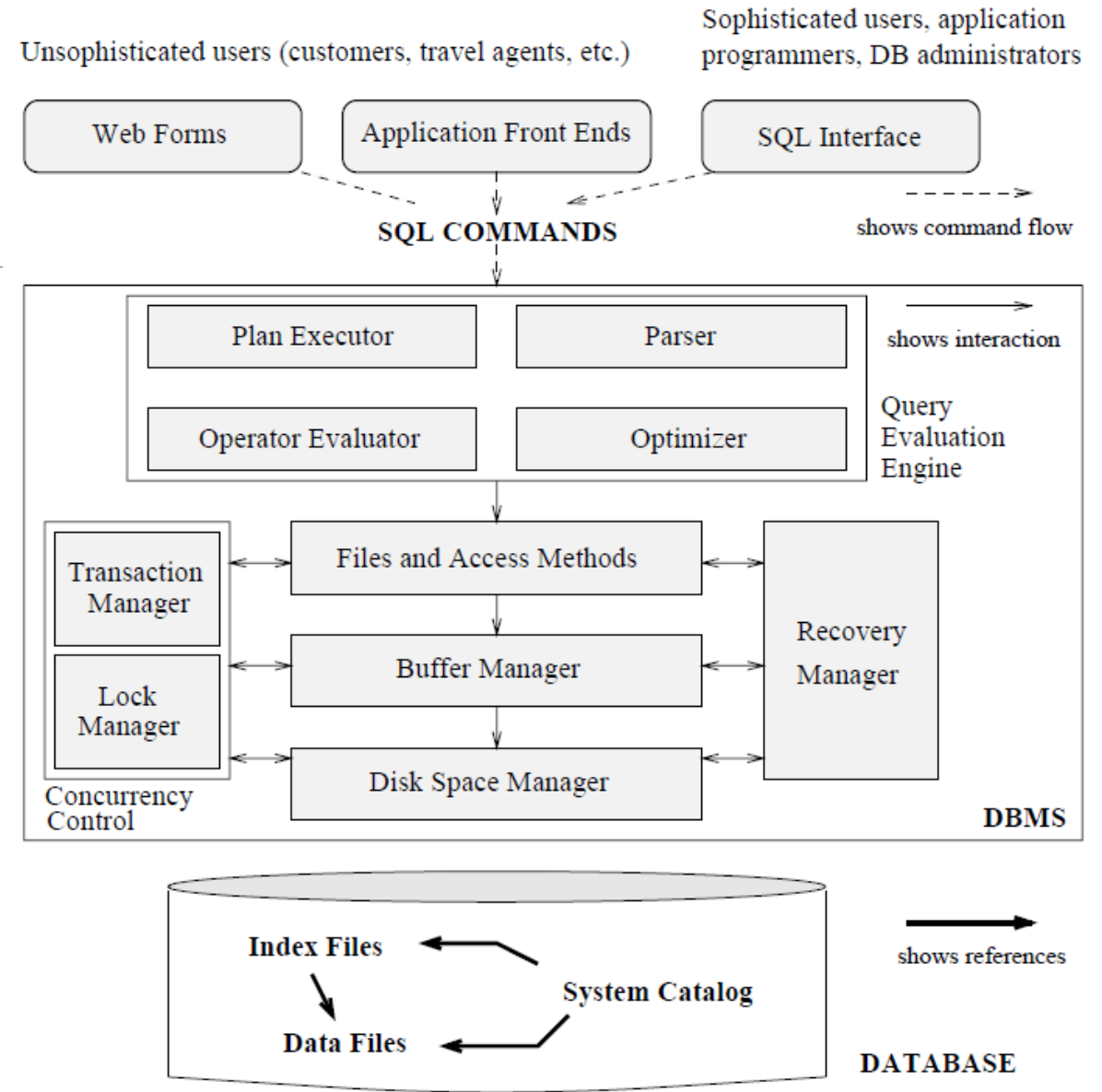


Storage and Indexing

BIRZEIT UNIVERSITY - SAMER ZAIN, PH.D.

Introduction

- ❑ **DBMS** abstracts data as a collection of **records** stored in a **file**.
- ❑ A file is a set of **pages**, each contain certain **set of records**.
- ❑ The **files layer** is responsible for data organization for fast data retrieval.
- ❑ **File organization**: a way of organizing records in a file.
- ❑ Each file organization makes certain operations efficient, but other operations expensive.

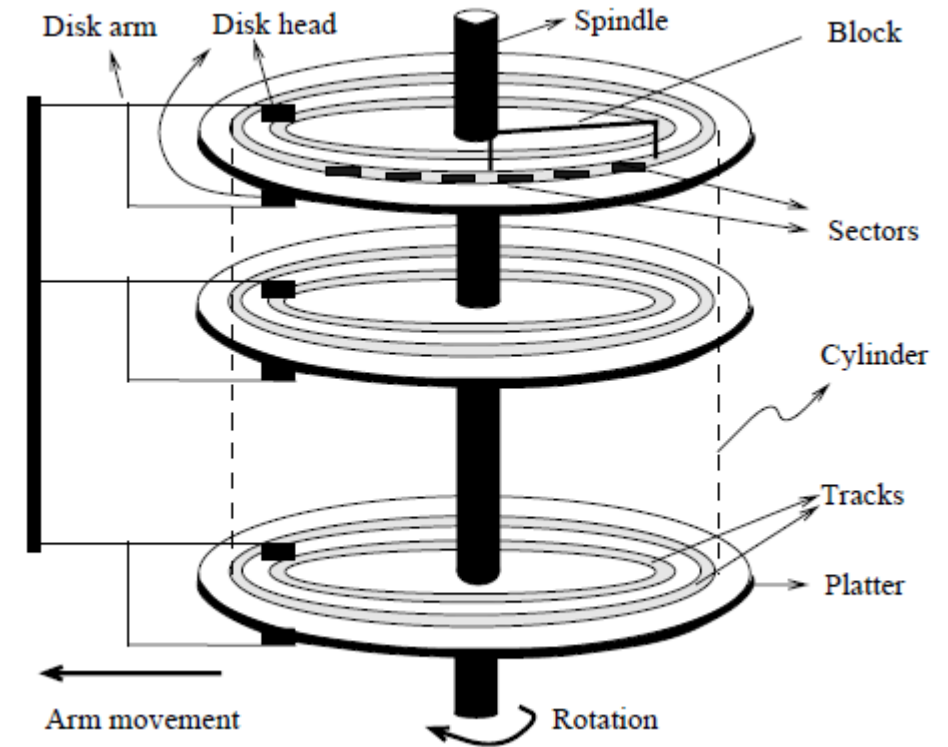


Introduction..2

- ❑ For example, suppose that we have a file of employees (name, age, and salary).
- ❑ If we want to retrieve employees based on **age**, then sorting the records based on age is a good idea.
- ❑ But if we want to retrieve employees based on **salary**, then we need to **scan the entire file**.
- ❑ Further, keeping the file sorted on age can be **expensive** if the file is modified frequently.
- ❑ We use **indexing** to access records of data in additional ways.
- ❑ We can build **more than one** index on the same data file, each with a different **search key**.

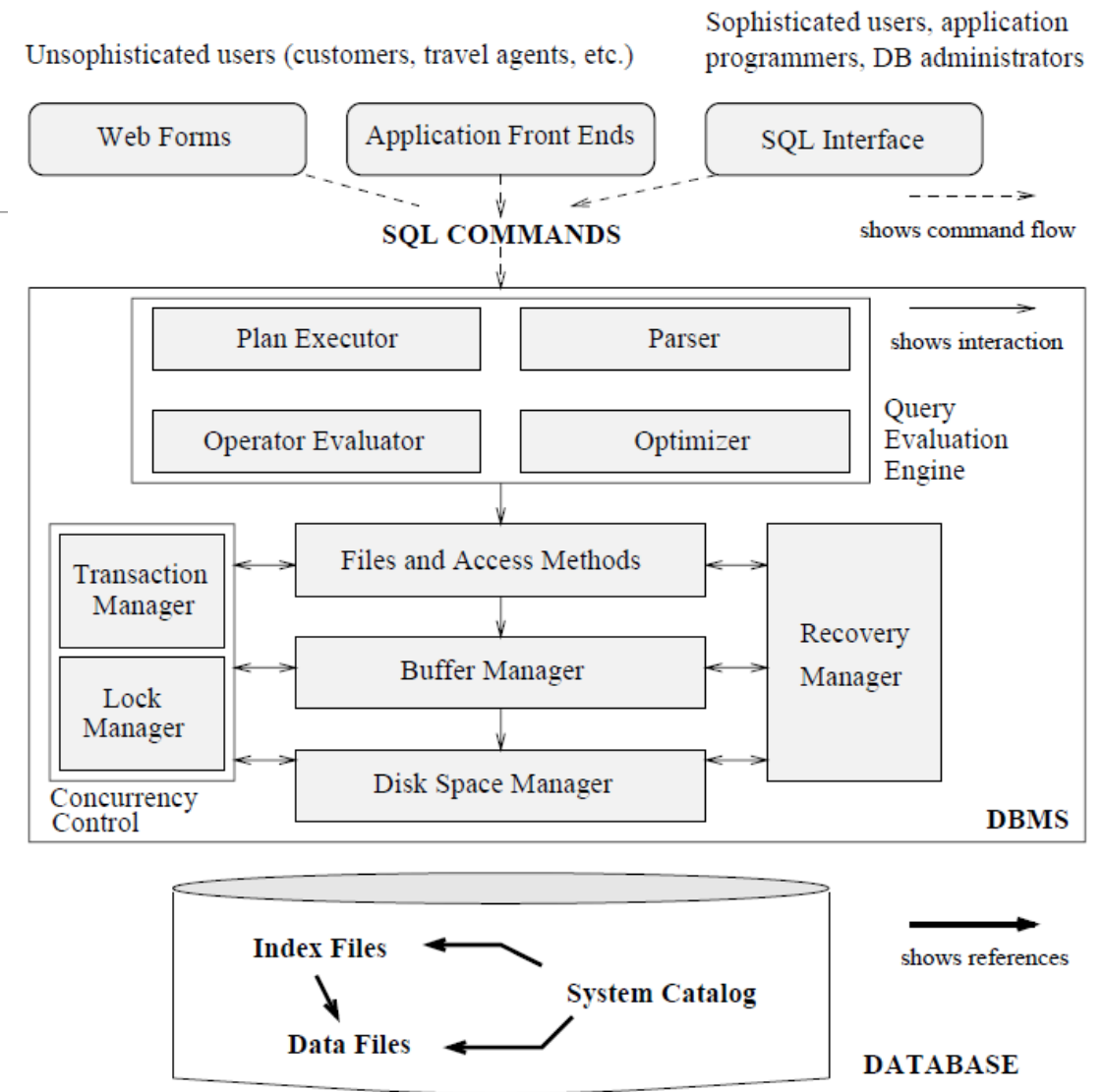
Data on External Storage

- ❑ **Hard disks** are the primary storage devices for DBMS
- ❑ The **taps** are used for archiving.
- ❑ The unit of information read from or written from disk is a **page**.
- ❑ A page is **typically 4KB** or **8KB**
- ❑ The cost of page I/O is the **most expensive** operation.
- ❑ Disks have **fixed cost per page**.
- ❑ Each record in a file has a unique identifier called **rid**.
- ❑ Using the **rid**, we can identify the **page address**



Data on External Storage ..2

- ❑ The **buffer manager** is responsible for loading a page into memory.
- ❑ When the files layer wants to access a certain page, it asks the **buffer manager** to load it into memory (if it is not already there)
- ❑ Space on disk is managed by **disk space manager**.



File Organization and Indexing

- ❑ As noted before, **DBMS (Files Layer)** abstracts data as **files of records**.
- ❑ Files can be created and destroyed, records added and deleted.
- ❑ Files also support scans.
- ❑ A relation (table) is stored as a file of records.
- ❑ the **file layer** stores records in a file in a **set of pages**.
- ❑ The file layer tracks pages and the records inside them.

Heap Files and Indexes

- ❑ A **heap file** is the simplest file organization: records are stored **randomly** across the pages.
- ❑ A heap file supports retrieval of all records or retrieval of particular record using rid.
- ❑ An **index** is a **data structure** that allows **fast retrieval** of data records.
- ❑ An **index** is based on a **search key**.
- ❑ We can create several indexes for same data file, each with different search key.

Indexes ..2

- ❑ Consider as an example our employee records.
- ❑ We can store the records in a file organized as an index on employee **age**.
- ❑ Additionally, we can create a separate index file based on **salary**, to speed up operations that involve retrieving employees based on salaries.
- ❑ The first file contains the actual employee records. While the second file contains **data entries**.
- ❑ A **data entry** is associated with **key** value K, and contains enough information to locate data records.
- ❑ We can **efficiently search** an index to find the desired data entries. We then use the data entries to locate the data records.

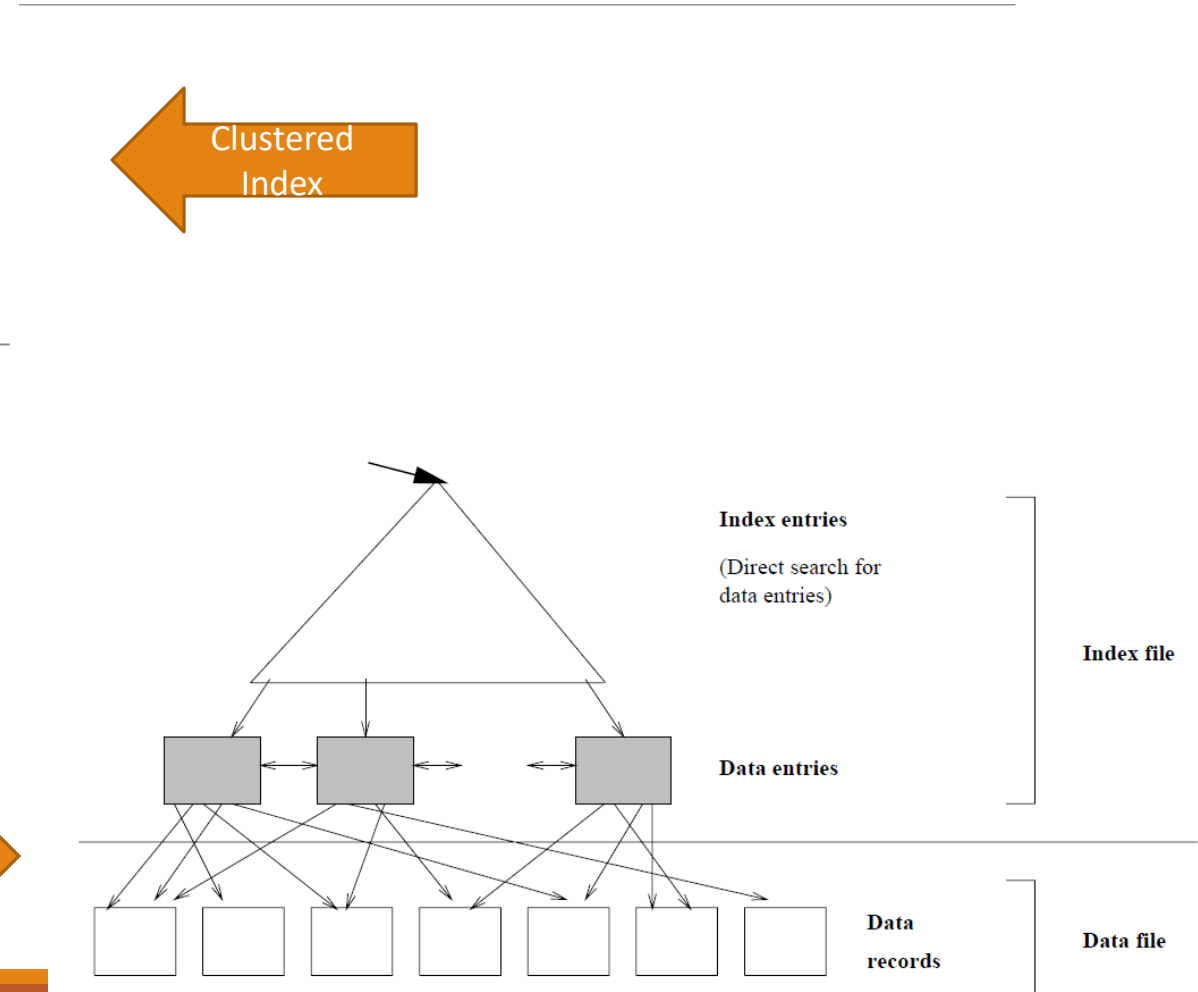
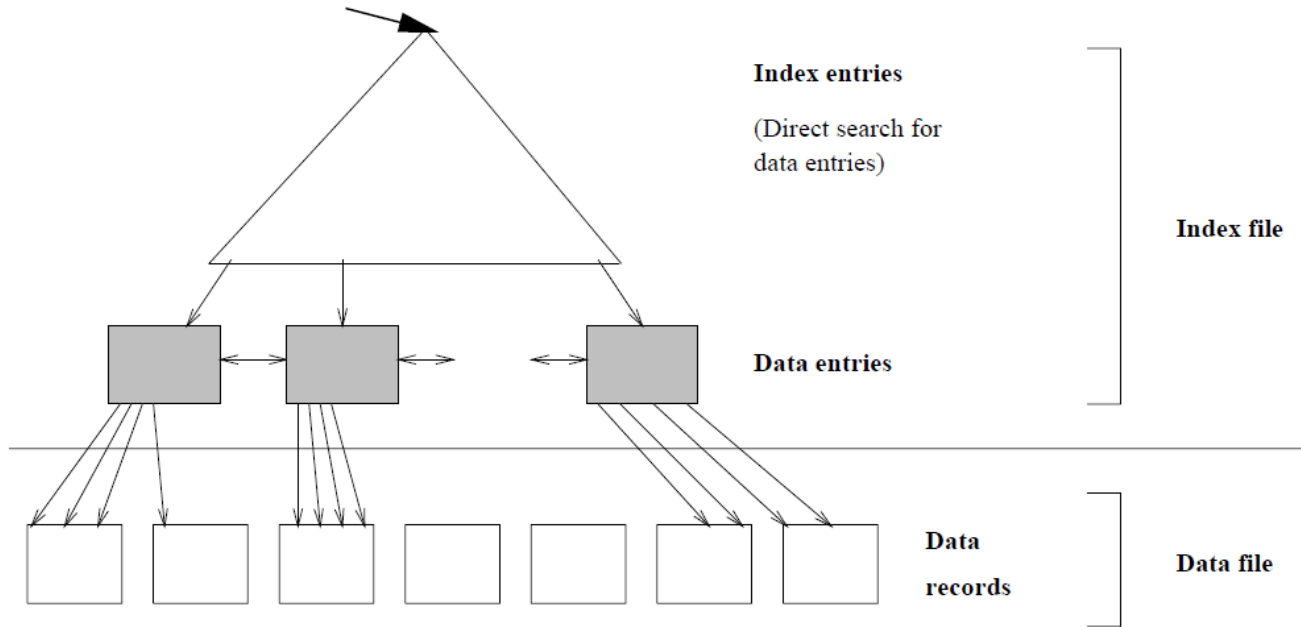
Indexes..3

- ❑ **Data Entries** in an index can be as one of the following:
- ❑ Alternative (1): a data entry is an actual data record associated with search key.
- ❑ Alternative (2): a data entry is a (K, rid) where K is the search key and rid is record id.
- ❑ Alternative (3): a data entry is a $(K, \text{rid list})$
- ❑ If we want to have more than one index on a file of records, at most we can have one Alternative(1), why?

Clustered Indexes

- ❑ **Definition:** when a file is organized so that the data records are ordered same as or close to the ordering of data entries of an index, we say that the index is **clustered**.
- ❑ An index that uses **Alternative (1)** is clustered by definition.
- ❑ An index that uses **Alternative(2) or Alternative(3)** can be clustered only if the data records are sorted on the search key field.
- ❑ In practice, indexes that uses Alternative(1) and Alternative(2) are **un-clustered**, why?
- ❑ The performance can be very efficient if we are answering a range query using clustered index.

Clustered vs Un-Clustered Indexes



Unclustered Index

Clustered Index

Primary and Secondary Indexes

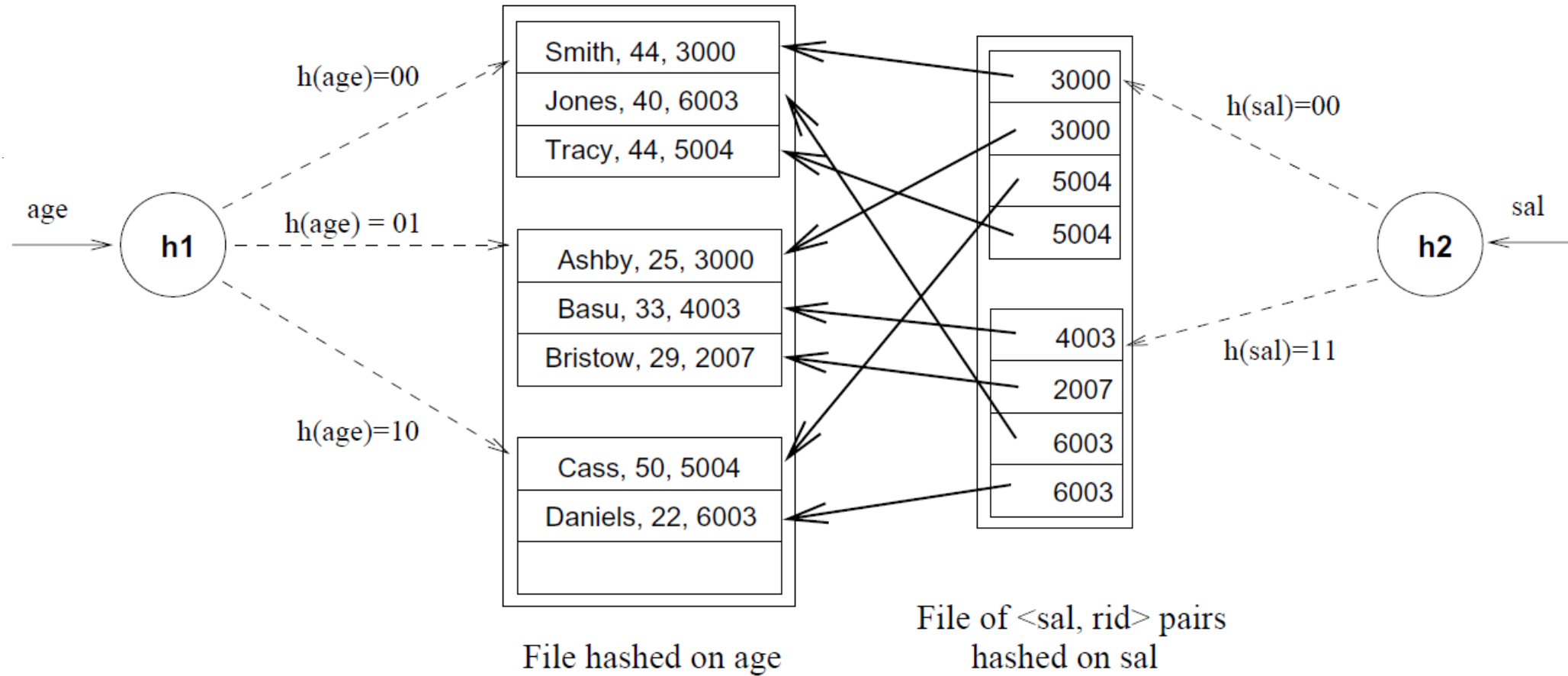
- A **primary index**: is an index on a set of fields that includes the primary key. Other indexes are called secondary indexes.
- Two data entries are said to be duplicate if they contain the same value for search key.
 - A Primary index is guaranteed not to have duplicates.
 - In general, secondary indexes contain duplicates.
 - **Unique index**: an index with no duplicates.

Hash-Based Indexing

- One way to implement indexing is hashing.
- For instance, if the files of employees is hashed on names, we can find all records of employee name John.
- The record are arranged in buckets ,with each bucket has one or more related pages.
- To identify a bucket, we apply hash function to the search key.
- We can retrieve required page in single I/O operation.



Hash-Based Index Example



- Note that the search key can be any set of fields.
- Also note the search key need not uniquely identify record.

Tree-Based Indexing

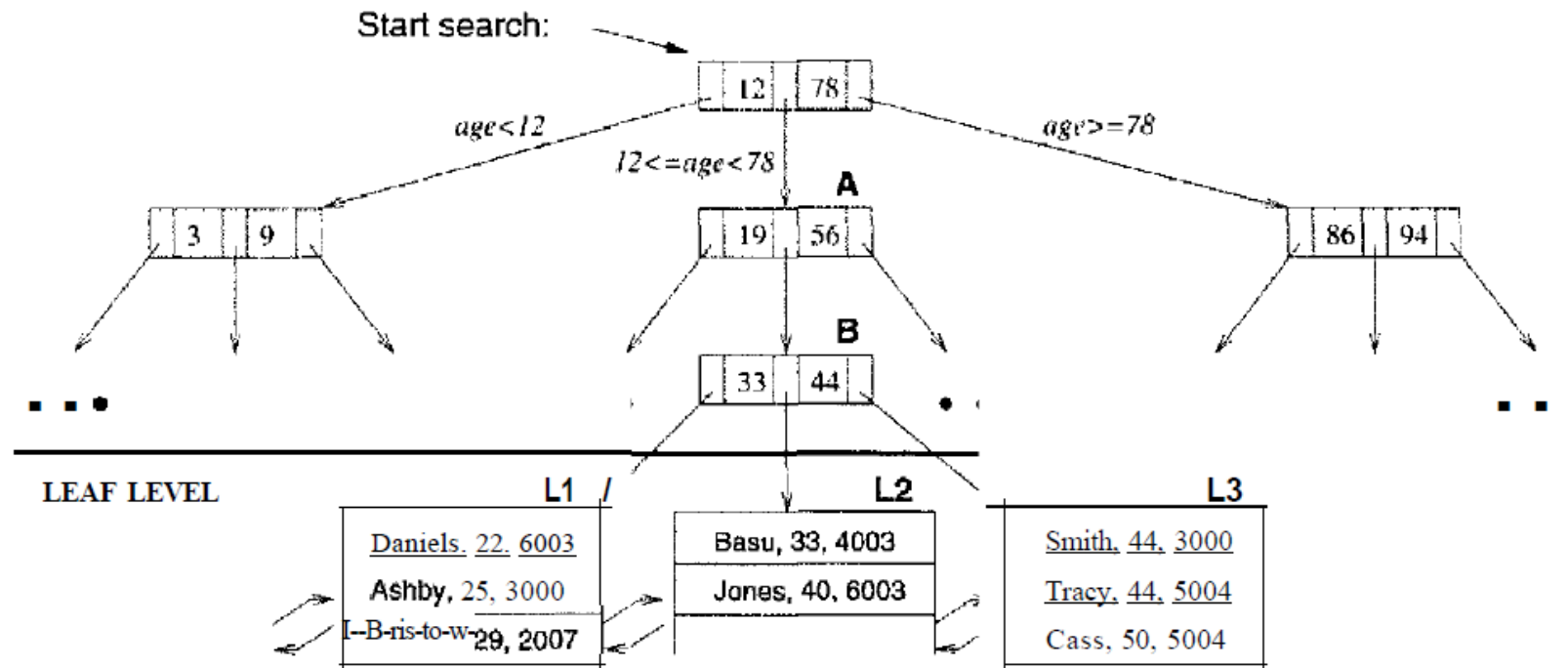


Figure 8.3 Tree-Structured Index

- Another way to arrange data entries in an index is using a **tree data structure**.
- The data entries are arranged in sorted order on search key.
- Note that L1, L2, and L3 are connected via double linked lists.
- Number of I/O = length of path + number of leaf pages.

Tree-Based Indexing..2

- ❑ **B+ tree** is an index structure in which all paths from root node to leaf in a tree is of **same length**.
- ❑ Here, finding the correct leaf is **faster than binary search** of pages in a sorted file.
- ❑ This is because each non-leaf node can have large number of node-pointers, and the **height** of tree is rarely more than **three** or **four** in practice.
- ❑ The average number of children for a non-leaf node is called **fan-out** of the tree.
- ❑ If every non-leaf tree node has N children, and the tree has H height, then the number of leaf pages = N^H
- ❑ In practice the average of N (F) = 100, thus the number of leaf nodes = 100 million.
- ❑ However, to find correct leaf node we need **4** I/Os, while in sorted file we need over **25**.

Comparison of File Organization

- ❑ Now we need to **compare** between different files organizations in terms of simple operations.
- ❑ We assume that the file and indexes are organized based on the composite **search key (sal, age)**
- ❑ We also assume that all selection operations are based on the composite key (sal, age)
- ❑ The operations we consider are:
 - ❑ **Scan**: Fetch all records in the file, thus all pages in the file must be fetched from disk into buffer pool.
 - ❑ **Search with Equality Selection**: such as “find all employees with salary 1200 and age 40”
 - ❑ **Search with Range Selection**: such as “find all employees with age > 35”
 - ❑ **Insert new Record**: identify correct page, load it into memory, change it, and write it back.
 - ❑ **Delete a Record**: using rid, we locate the related page, load it into memory, change it, and write it back.

Cost Model

- ❑ We need to agree on the cost model that we will use to differentiate between files:
 - ❑ **B** = Number of data pages.
 - ❑ **R** = Number of records per page.
 - ❑ **D** = Average time to read a page from disk.
 - ❑ **C** = Time to process a record.
 - ❑ **H** = time to apply the hash function.
 - ❑ **F** = fan-out for a tree based file.

Cost for Heap Files

- **Scan:** the cost = $B(D+RC)$
- **Search with Equality Selection:**
 - ❑ if selection is specified on a candidate key then cost = $0.5B(D + RC)$
 - ❑ If selection is not specified on candidate key then we need to search entire file.
- **Scan with Range Selection:** cost = $B(D+RC)$ coz we need to search all file.
- **Insert:** We assume that inserts are at end of the file, cost = $2D + C$
- **Delete:** cost = cost of search + $C + D$

Cost for Sorted Files

- **Scan:** the cost = $B(D+RC)$, again, all pages will be scanned.
- **Search with Equality Selection:**
 - ❑ if selection is specified on age then cost = $D\log_2 B + C\log_2 R$
 - ❑ If selection is not specified on age, it is same as heap file.
- **Scan with Range Selection:** cost = $B(D+RC)$ coz we need to search all file.
- **Insert:** we must find correct position in the file, add new record, then rewrite all subsequent pages. Cost = search + $B(D + RC)$
- **Delete:** cost = cost of search + $C + D$

Comparison of I/O Costs

<i>File Type</i>	<i>Scan</i>	<i>Equality Search</i>	<i>Range Search</i>	<i>Insert</i>	<i>Delete</i>
Heap	BD	$0.5BD$	BD	$2D$	$Search + D$
Sorted	BD	$D \log_2 B$	$D \log_2 B + \# \text{ matching pages}$	$Search + BD$	$Search + BD$
Clustered	$1.5BD$	$D \log F 1.5B$	$D \log F 1.5B + \# \text{ matching pages}$	$Search + D$	$Search + D$
Unclustered tree index	$BD(R + 0.15)$	$D(1 + \log FO.15B)$	$D(\log FO.15B + \# \text{ matching records})$	$D(3 + \log FO.15B)$	$Search + 2D$
Unclustered hash index	$BD(R + 0.125)$	$2D$	BD	$4D$	$Search + 2D$

Comparison of I/O Cost..2

- ❑ A heap file has good storage efficiency and supports fast scanning and insertion of records. But it is slow for searches and deletions.
- ❑ A sorted file also offers good storage efficiency, but insertion and deletion of records is low. Searches are faster than heap files.
- ❑ A clustered file offers all the advantages of a sorted file plus supporting inserts and deletes efficiently. but requires more space on disk. Searches are even faster than sorted files.
- ❑ Unclustered tree and hash indexes offer fast searches, insertion, and deletion, but scans and range searches with many matches are slow. Hash indexes are a little faster on equality searches, but they do not support range searches