[1] Storage and Indexing

1. Storage (5 min)

- Data on External Storage (Important points to keep in mind: Disks, tapes and records).
- Physical characteristics of disks and tapes, and how do they affect the design of database systems.
- Levels of Redundancy (RAID levels).

2. Indexes (5 min)

- B+ Tree Indexes.
- Hash-Based Indexes.
- Clustered vs. Unclustered Index.
- Choice of Indexes.

1. Storage

A DBMS stores large quantities of data, and the data must persist across program executions, therefore, data is stored on external storage:

- Data on external storage: Disks, Tapes, Records:

- Physical characteristics of:
  - Disks?: Allow us to retrieve any page at a fixed cost per page. However, if we read several pages in the order that they are stored physically, the cost can be much less than the cost of reading the same pages in a random order.

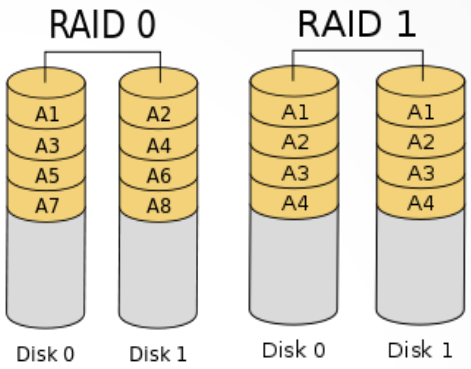
  - Tapes?: Are sequential access devices and force us to read data one page after the other. Mostly used to archive data that is not needed on a regular basis.

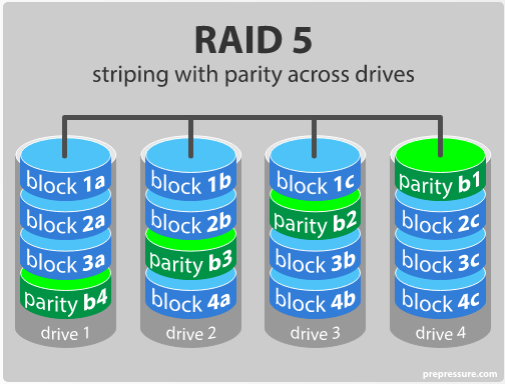
  - Records?: Each record in a file has a unique identifier called a record id (rid) → with property that we can identify the disk address of the page containing the record.

- How do they affect the design of a database system?

1.1 Redundant array of independent disks

- Level of redundancy (RAID)?

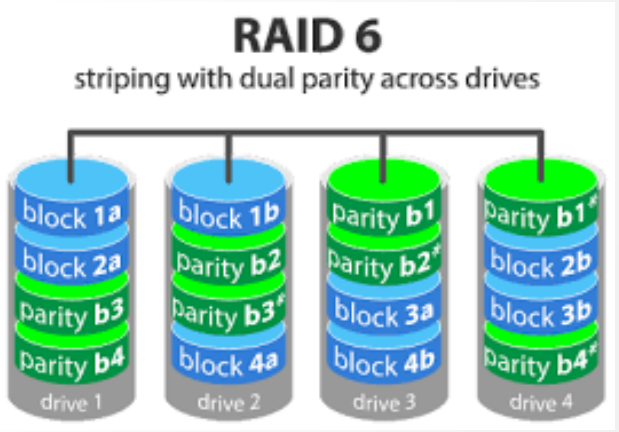




- Level 0 (non-redundant or striping):
if data loss is not an issue. Level 0 improves overall system performance, at the lowest cost.
(example: 1 man says → abcdefghijklmnopqrstuvwxyz
2 men: first → abcdefghijklm,
second → nopqrstuvwxyz)

- Level 1 (mirrored):
same data is written to two drives. If one drive fails, the data still exist on the other drive.

- Level 0+1 (Striping and Mirroring):
Superior to level 1. Systems are small storage subsystems where the cost of mirroring is moderate. Used for application that have a high percentage of writes in their workload.



- Level 5 (Striping with parity / compresses data and stripes):
Just as raid 0 but with a parity bit (ekstra data) across the disk, that allows the disk to be recreated. But if a drive fails the array remains intact, but degraded state → because we have a fail drive, and data is being reconstructed → reduced performance.
No more than one drive must fails before we cant recover.

- Level 6 (P+Q Redundancy):
Just as raid 5 only with double distributed parity. Now to disk can fail and still be reconstructed.

2. Intro + indexes

First a short introduction to DBMS:

- Data in DBMS is a collection of records, or a file, and each file consist of one or more pages.
  A page is: the unit of information read from or written to disk. Typically size 4KB or 8KB

- Files and access methods define how we can access data.
  Files can be created, destroyed, have records inserted into and deleted from. Scan operations allow to step through all the records in the file one at a time.

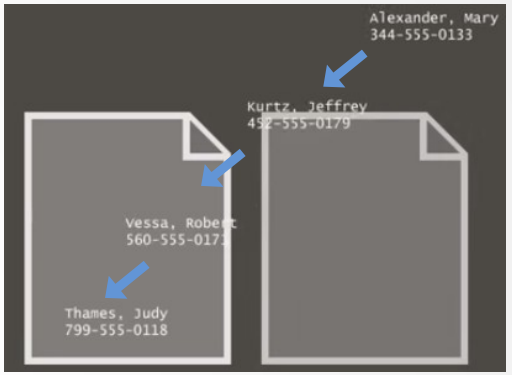
- File organization is a method of arranging records in a file when it is stored on disk.
  - Each file organization makes certain operations efficient but others expensive.
  Keeps track of pages allocated to each file, and as records are inserted into and deleted from the file, it also tracks available space within pages allocated to the file.

- Choosing the right indexes is a powerful tool (the single most powerful tool)
  An index is a data-structure that organizes data records on disk to optimize certain kinds of retrieval operations. An index allows us to efficiently retrieve all records that satisfy search conditions on the search key fields of the index.

- **Data entry** → the data records stored in an index file
  Data entry with search key value k, denoted as k*

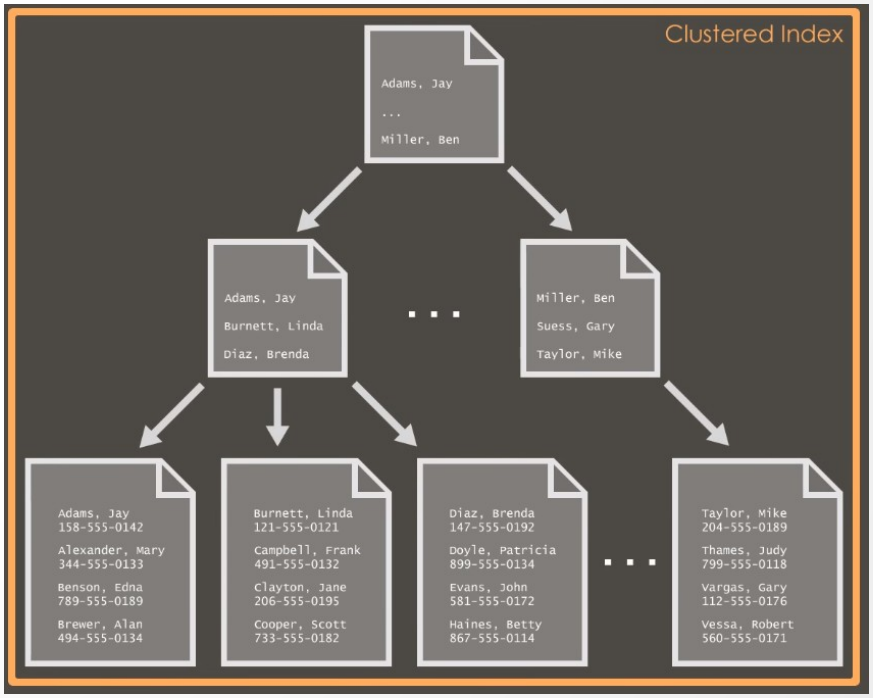
2.1 Indexes

```
Create table phoneBook (
        LastName,
        FirstName,
        PhoneNumber
);
```

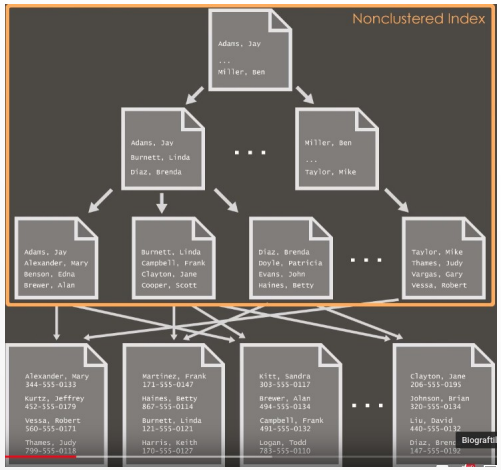


- **Heap-file**
  if a table has no indexes, its a heap-file.
  File of randomly order → inserted where there is free space, and in no particular order. If we looking for a specific persons phone-number, we would have to check every single row in the table. (a scan)

  In a phone book we would sort from last-name A-Z (this is the index key)



- **Clustered vs. unclustered index**
  When a file is organized so ordering of data records is the same (or close to) the ordering of data entries in some index, we say this index is clustered – otherwise the index is unclustered.



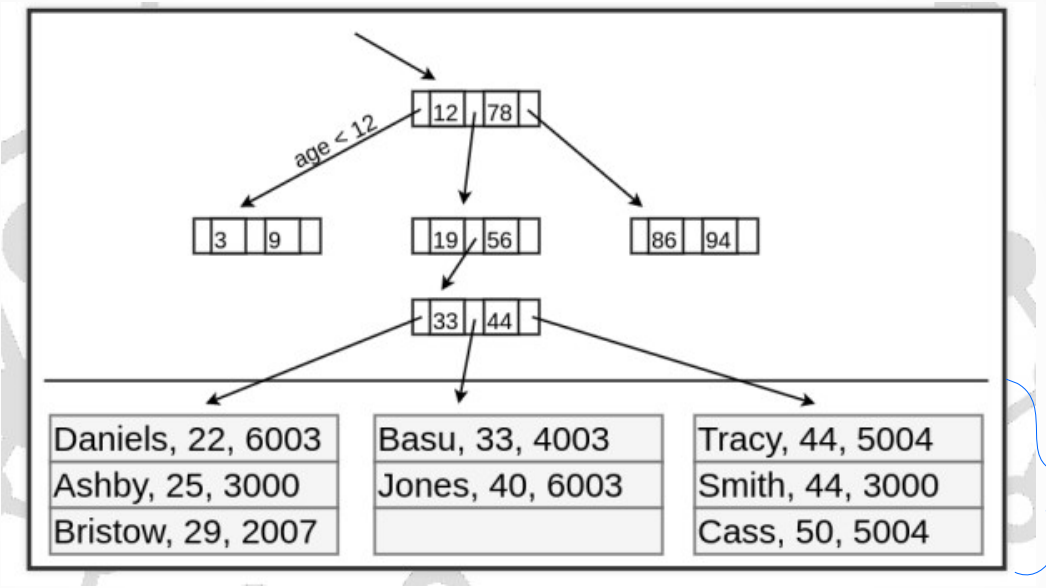
- Bonus info:
  - A primary key = clustered
  - Unique key = unclustered

2.2 Indexes

- B+ tree indexes

Key Issues

- Search
- Insert
- Delete
- Key-compression
- Bulk-loading
- Performance of operations



Root node

Intermediate nodes

Leaf-nodes: Table rows

A B+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B+ tree denote actual data pointers.

B+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B+ tree can support random access as well as sequential access.

- Insert and Delete keep tree balanced
- Root is either leaf or has between roundUp(M/2) and M children
  - Min occupancy of 50% if not root
- All leaves are at the same depth
- Because of high fan-out, height rarely more than 3-4

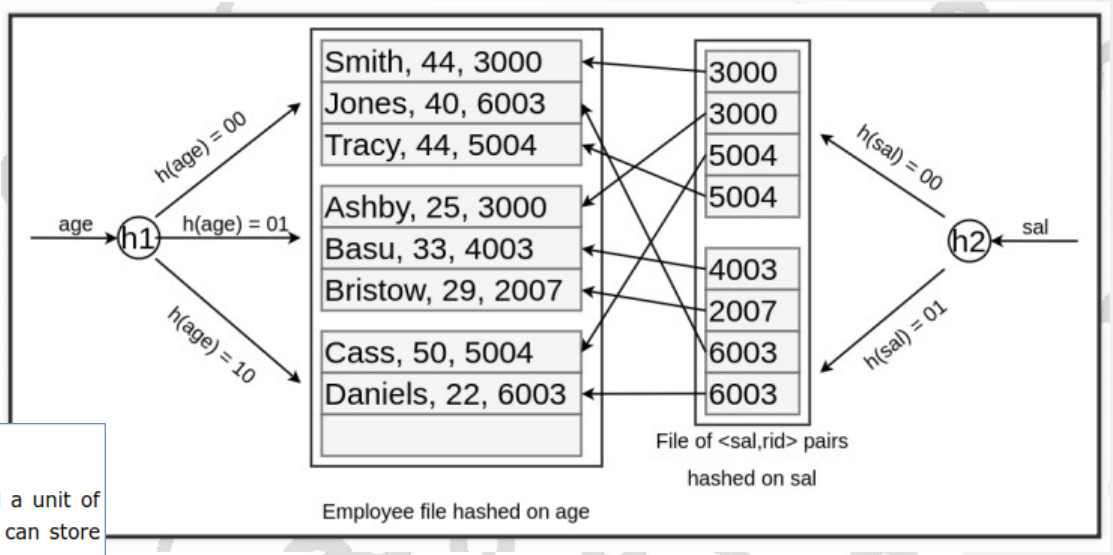
2.3 Indexes

- Hash-based indexes

For a huge database structure, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data.

Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.

Hashing uses hash functions with search keys as parameters to generate the address of a data record.

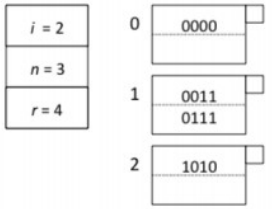


Hash Organization

- **Bucket** – A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
- **Hash Function** – A hash function, **h,** is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

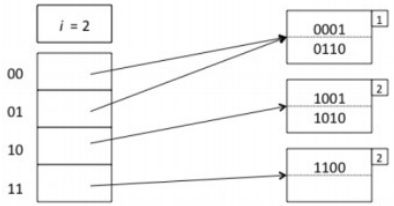
LINEAR HASHING

Linear Hash Table with $k = 4, f = 2, p_{max} = 0.8$



EXTENDIBLE HASHING

Extensible Hash Table with $k = 4, f = 2$



Key Issues

- Static hashing
- Extendible hashing
- Linear Hashing

2.3 Indexes

- Choice of indexes

Questions

- How does a DBMS store and access persistent data?

- Why is I/O cost so important for DB operations?

- How is data organized to minimize I/O cost?

- How are indexes used and what are their properties

- What is the relationship between a file of data records and any indexes on this file of records?

- What are important properties of indexes?

- How does a hash-based index work, and when is it most effective?

- How does a tree-based index work, and when is it most effective?

- How can we use indexes to optimize performance for a given workload?

Key konceps:

external storage, buffer manager, page I/O.
file organization, heap files, sorted files.
indexes, data entries, search keys, clustered index, clustered file, primary index
index organization, hashbased and tree-based indexes
cost comparison, file organizations and common operations
performance tuning, workload, composite search keys, use of clustering