Fakultas Informatika
School of Computing
Telkom University

CSH2D3 - Database System

# 03 | Indexing

# Goals of the Meeting

01

Students knows the basic concepts of indexing

02

Students understand how to organize B+-Tree Index Files

03

Students can create indexes on the DBMS to speed up searching performance

# Outline

- Basic Concepts
- Ordered Indices
- $B^+$-Tree Index Files
- Creation of Indices

# Basic Concepts

# Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form:

| search-key | pointer |
|---|---|

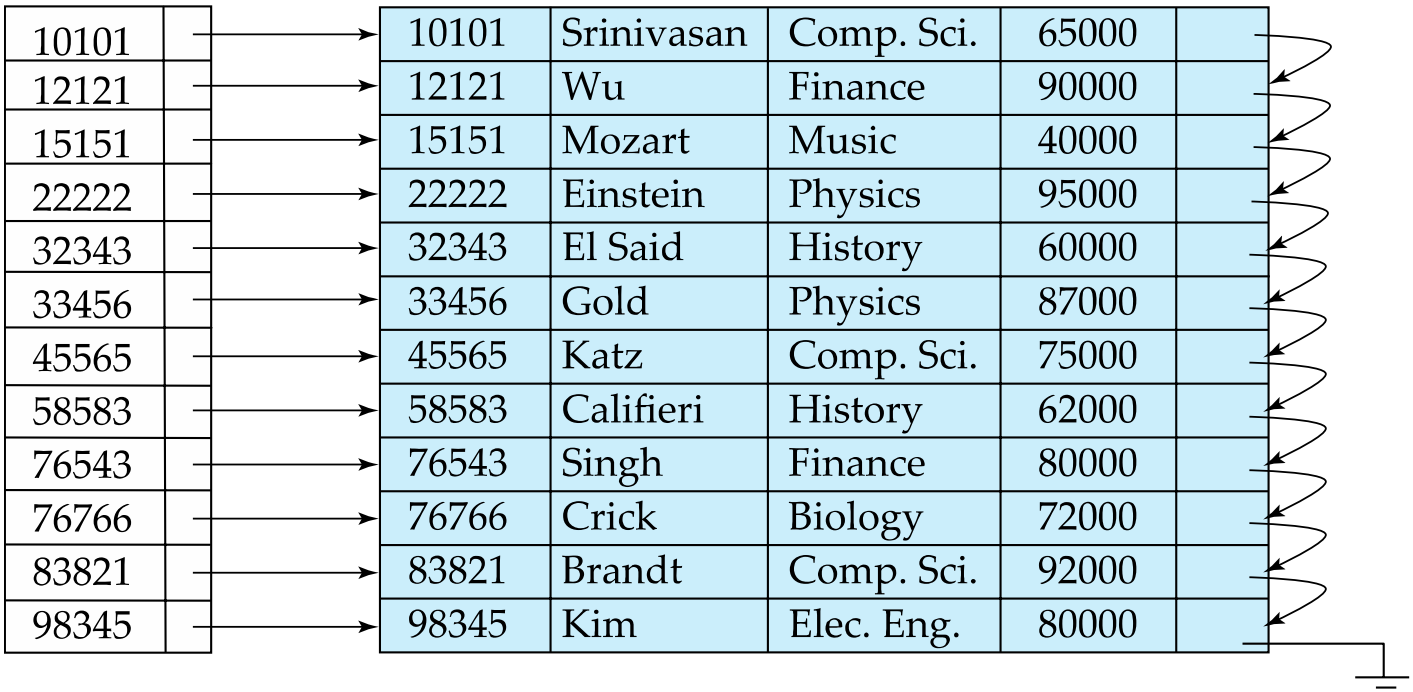
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

# Ordered Indices

- In an **ordered index,** index entries are stored sorted on the search key value.
- **Clustering index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
  - Also called **primary index**
  - The search key of a primary index is usually but not necessarily the primary key.
- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called **nonclustering index.**
- **Index-sequential file:** sequential file ordered on a search key, with a clustering index on the search key.

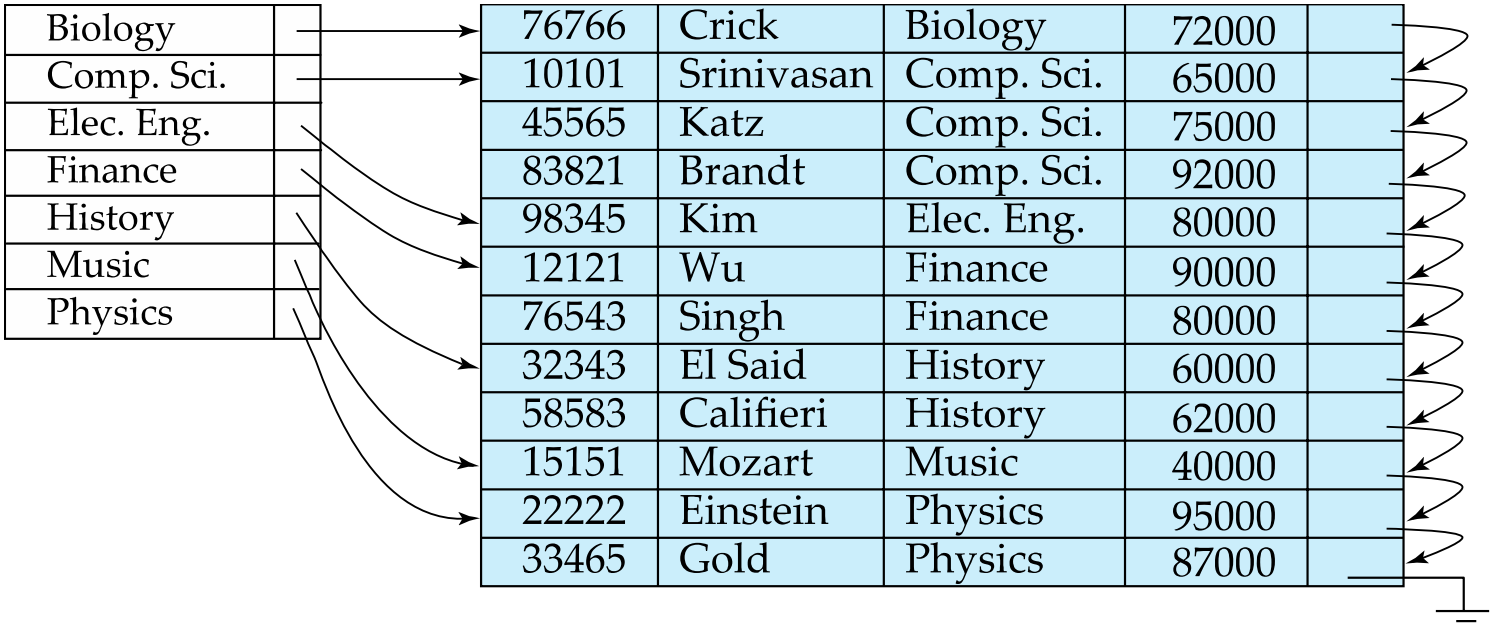
# Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation



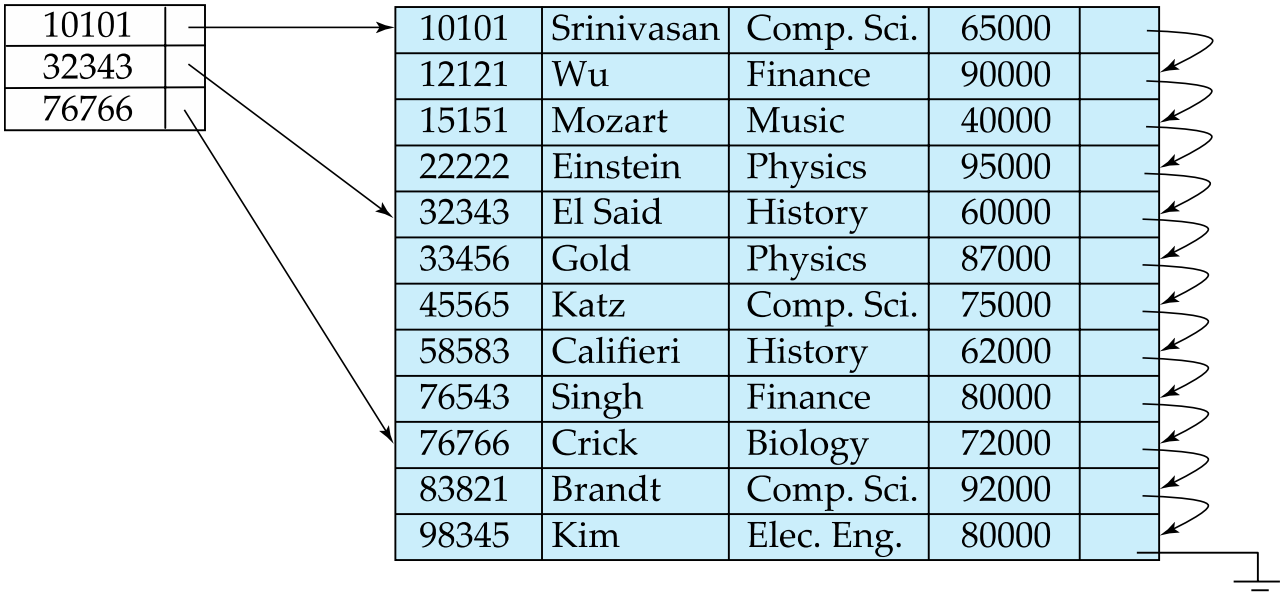
# Dense Index Files (Cont.)

- Dense index on *dept_name*, with *instructor* file sorted on *dept_name*



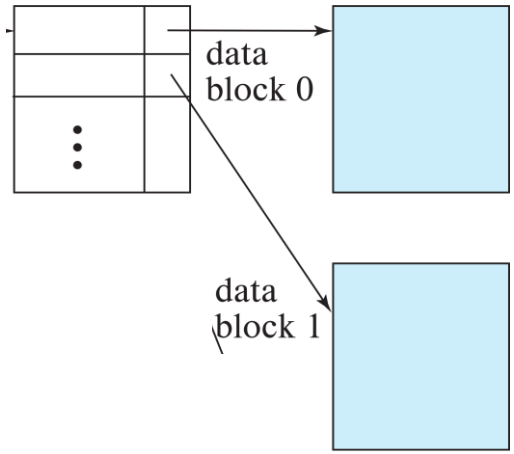
# Sparse Index Files

- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points



# Sparse Index Files (Cont.)

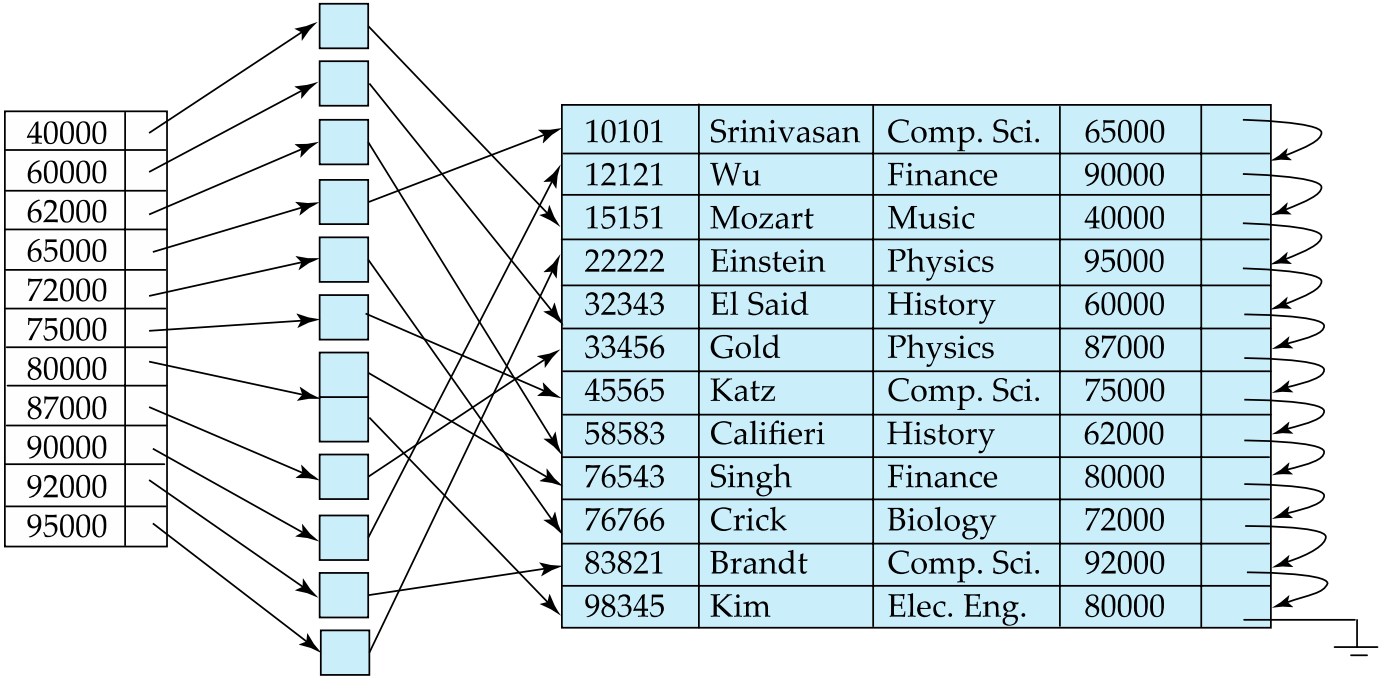
- Compared to dense indices:
  - Less space and less maintenance overhead for insertions and deletions.
  - Generally slower than dense index for locating records.
- **Good tradeoff**:
  - for clustered index: sparse index with an index entry for every block in file, corresponding to least search-key value in the block.




  - For unclustered index: sparse index on top of dense index (multilevel index)

# Secondary Indices Example

- Secondary index on salary field of instructor




- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value.
- Secondary indices have to be dense

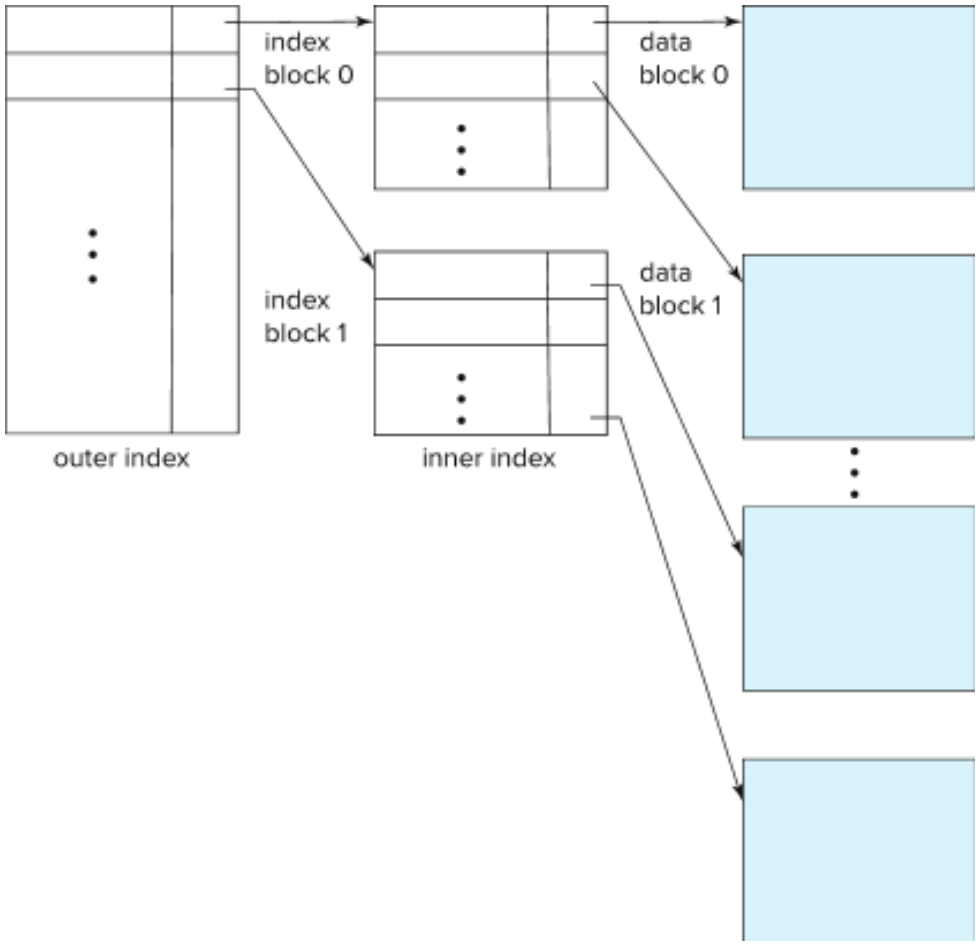
# Clustering vs Nonclustering Indices

- Indices offer substantial benefits when searching for records.
- BUT: indices imposes overhead on database modification
  - when a record is inserted or deleted, every index on the relation must be updated
  - When a record is updated, any index on an updated attribute must be updated
- Sequential scan using clustering index is efficient, but a sequential scan using a secondary (nonclustering) index is expensive on magnetic disk
  - Each record access may fetch a new block from disk
  - Each block fetch on magnetic disk requires about 5 to 10 milliseconds

# Multilevel Index

- If index does not fit in memory, access becomes expensive.
- Solution: treat index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of the basic index
  - inner index – the basic index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

# Multilevel Index (Cont.)

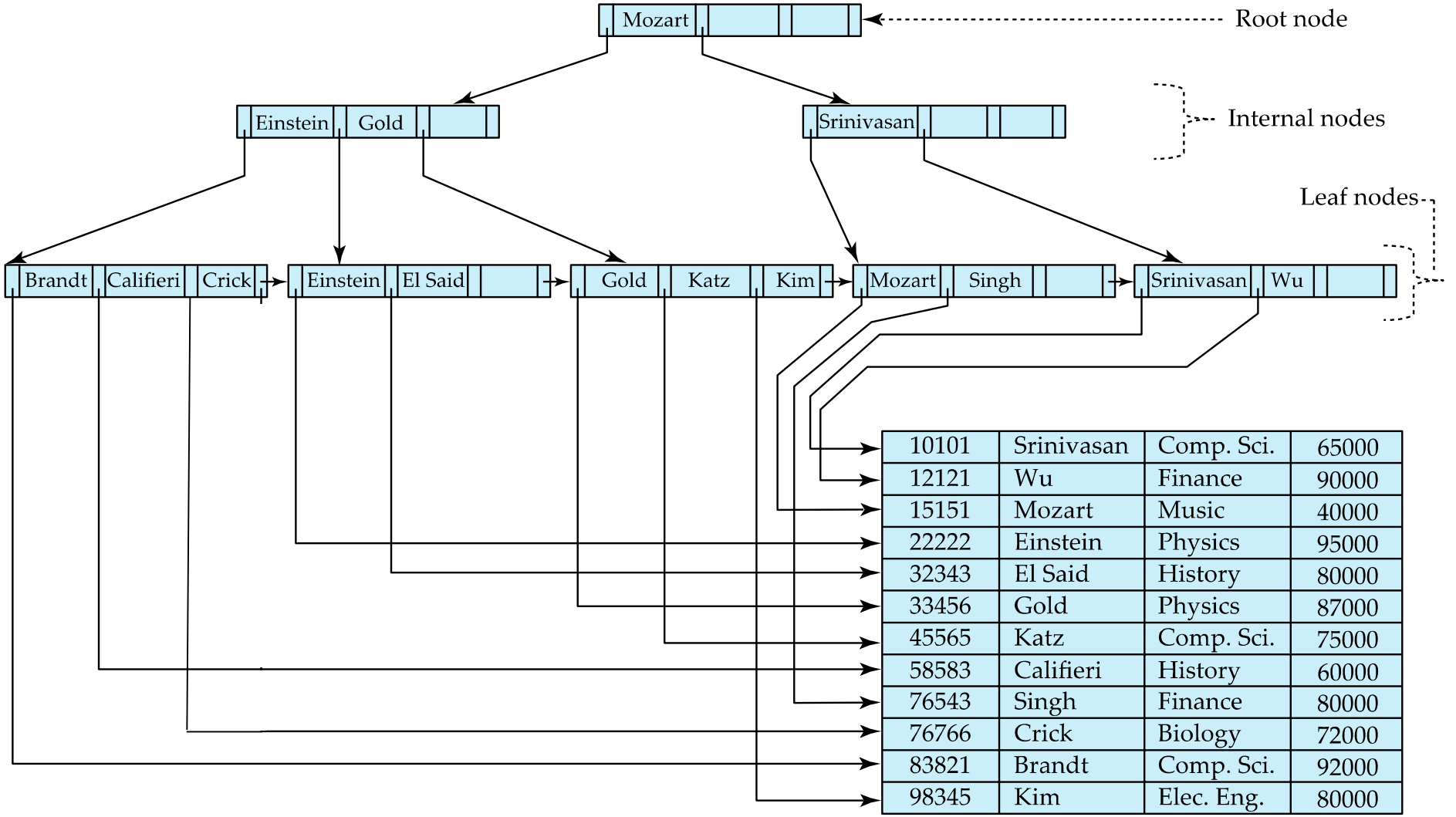


# $B^+$-Tree Index Files

# $B^+$-Tree Index Files

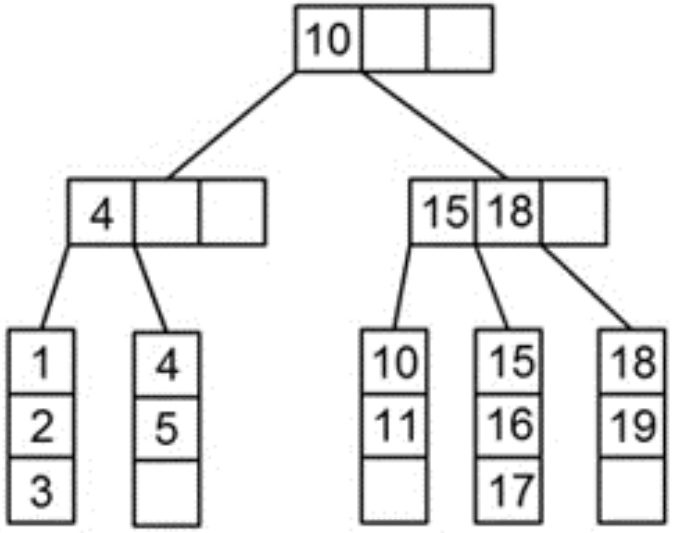
- Disadvantage of indexed-sequential files
  - Performance degrades as file grows, since many overflow blocks get created.
  - Periodic reorganization of entire file is required.
- Advantage of $B^+$-tree index files:
  - Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
  - Reorganization of entire file is not required to maintain performance.
- (Minor) disadvantage of $B^+$-trees:
  - Extra insertion and deletion overhead, space overhead.
- Advantages of $B^+$-trees outweigh disadvantages
  - $B^+$-trees are used extensively

# Example of B+-Tree




| | | | |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 80000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 60000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

# B+Tree Basic




Source : https://youtu.be/49P_GDeMDRo

# B+Tree Insertion

Source : https://youtu.be/h6Mw7_S4ai0

# B+Tree Deletion

Source : https://youtu.be/QrbaQDSuxIM

# Creation of Indices

# Creation of Indices

- Example
  **create index** *takes_pk* **on** *takes* (*ID,course_ID, year, semester, section*)
  **drop index** *takes_pk*
- Most database systems allow specification of type of index, and clustering.
- Indices on primary key created automatically by all databases
  - Why?
- Some database also create indices on foreign key attributes
  - Why might such an index be useful for this query:
    - $takes \bowtie \sigma_{name='Shankar'}(student)$
- Indices can greatly speed up lookups, but impose cost on updates
  - Index tuning assistants/wizards supported on several databases to help choose indices, based on query and update workload

# Index Definition in SQL

- Create an index:

  **create index** <index-name> **on** <relation-name>(<attribute-list>)

  E.g.,: **create index** *b-index* **on** *branch(branch_name)*

- Use **create unique index** to indirectly specify and enforce the condition that the search key is a candidate key is a candidate key.
  - Not really required if SQL **unique** integrity constraint is supported
- To drop an index:

  **drop index** <index-name>

- Most database systems allow specification of type of index, and clustering.

# 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