DBMS Week 9 TA Session

Indexing

- Indexing mechanisms used to speed up access to desired data.
- An index file consists of records (called index entries) of the form

search-key	pointer

Types 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.

Primary index

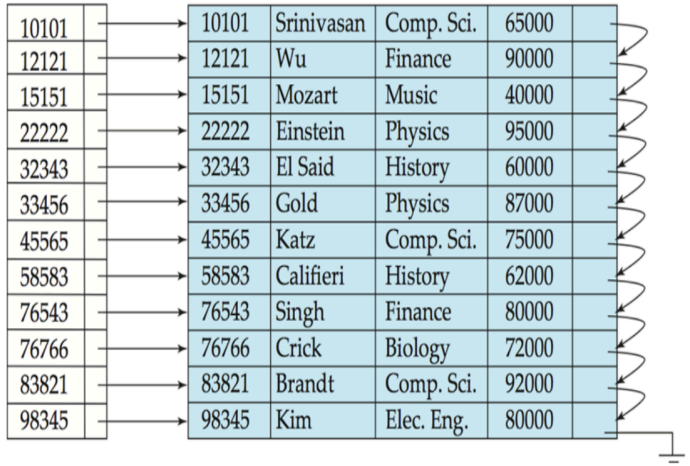
- In a sequentially ordered file, the index whose search key specifies the sequential order of the file
- Also called **clustering index**

Secondary index

- An index whose search key specifies an order different from the sequential order of the file
- Also called **non-clustering index**

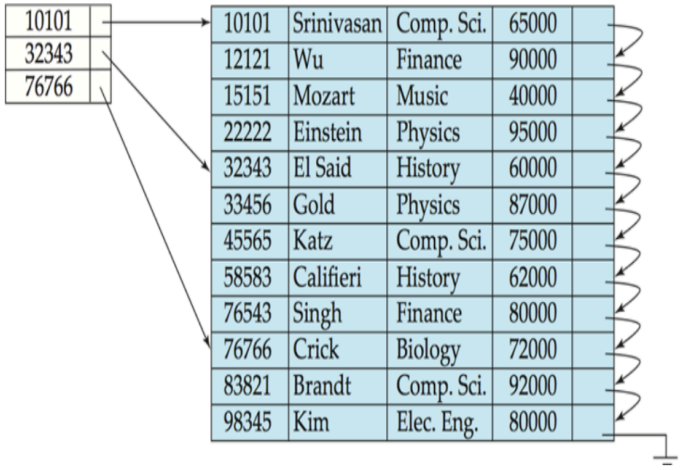
Dense Index

- Index record appears for every search-key value in the file.



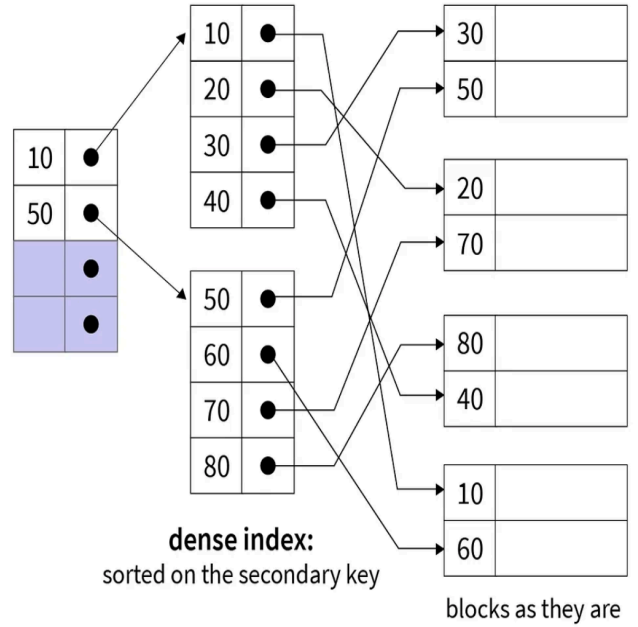
Sparse Index

- It contains index records only some search-key values.
- Applicable when records are sequentially ordered on search-key.



Multilevel Indexing

- If primary index does not fit in memory, access becomes expensive.
- Treat primary index kept on disk as a sequential file and construct a sparse index on it.
  - **outer index** – a sparse index of primary index
  - **inner index** – the primary index file



Some Formulas for solving the problem

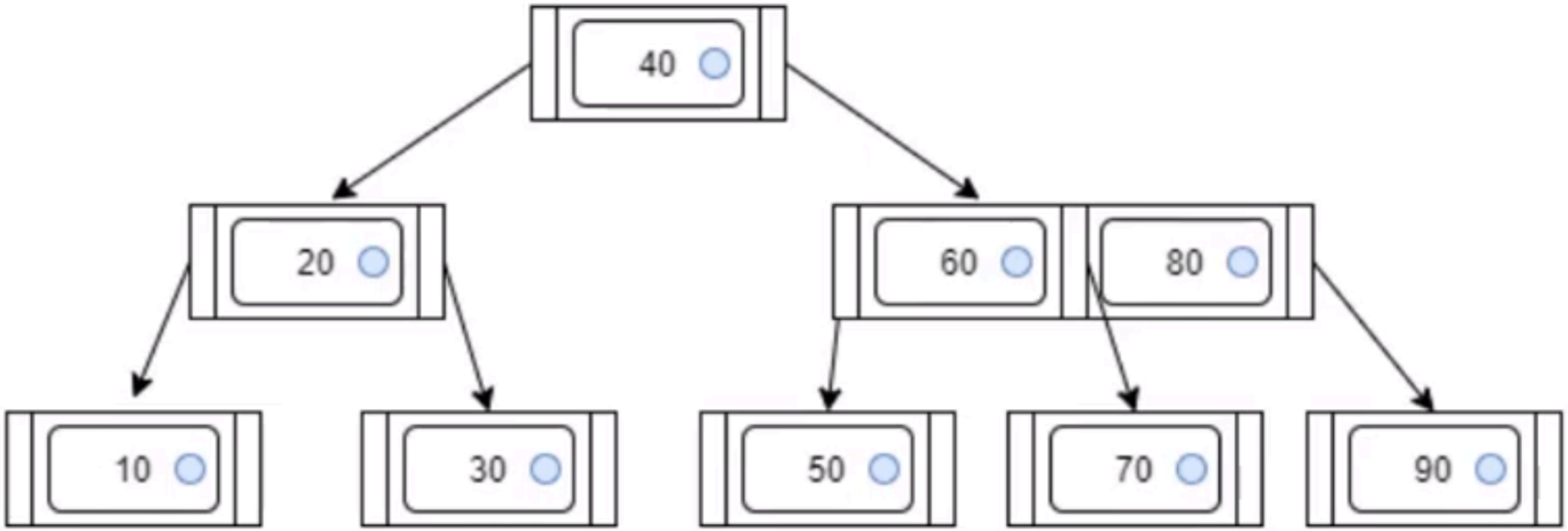
- $Space\ of\ index = (size\ of\ key\ field + Pointer\ field)$
- $Required\ space\ to\ store\ all\ the\ index = space\ of\ index \times no.\ of records$
- $One\ Block\ size = \frac{size\ of\ file}{no.\ of\ block\ used\ stored\ the\ file}$
- $No.\ of\ blocks\ requiredto\ store\ the\ index = \frac{space\ occupied\ by\ the\ indicies}{one\ block\ size}$

B Tree

Construct a 3 order B Tree from the key values inserted sequentially

20, 50, 40, 80, 90, 10, 30, 70, 60

Solution

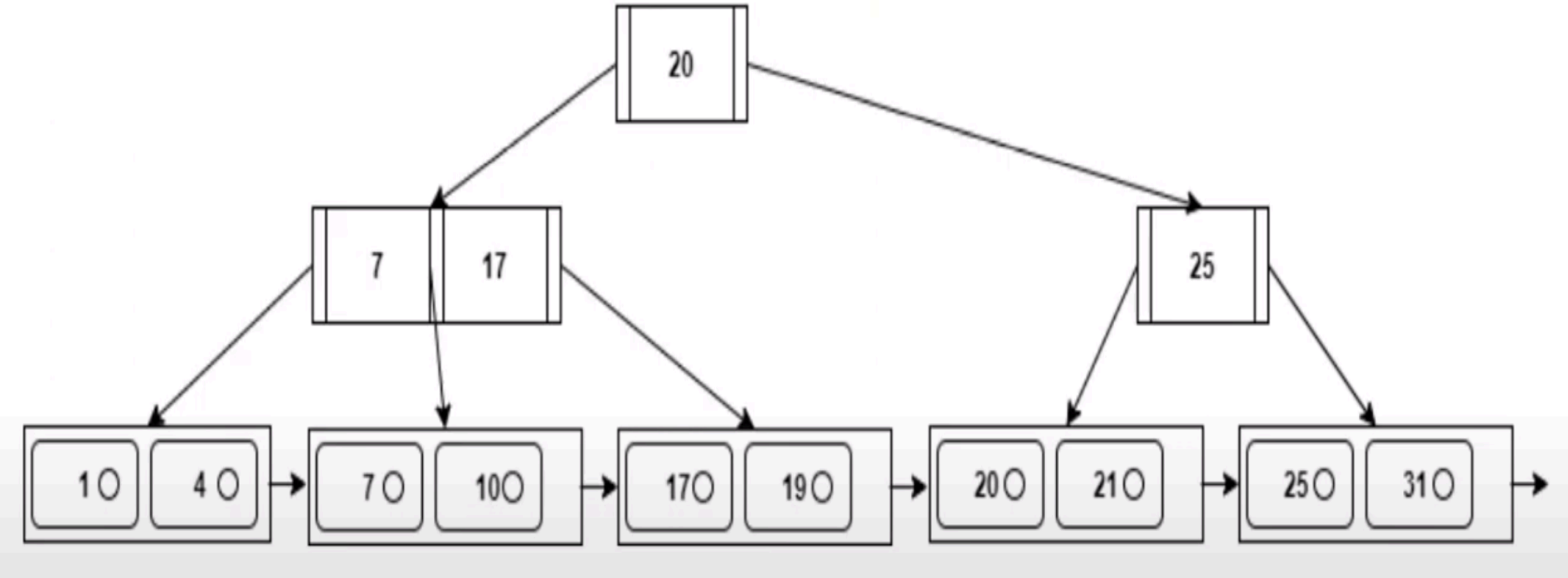


B+ Tree

Construct a 4 order B^+ Tree from the key values inserted sequentially

1, 4, 7, 10, 17, 21, 31, 25, 19, 20

Solution



Difference between B Tree and B+ Tree

Basis of Comparison	B tree	B+ tree
Pointers	All internal and leaf nodes have data pointers	Only leaf nodes have data pointers
Redundant Keys	No duplicate of keys is maintained in the tree.	Duplicate of keys are maintained and all nodes are present at the leaf.
Leaf Nodes	Leaf nodes are not stored as structural linked list.	Leaf nodes are stored as structural linked list.
Tree	B Tree may or may not be balanced.	B^+ Tree is always balanced.

Facts about B and B^+ Tree

- **2 Order Tree** - 1 node & 2 child
- **3 Order Tree** - 2 node & 3 child
- **4 Order Tree** - 3 node & 4 child

A non-root node of a B tree has

- min. number of child-node pointers = $\lceil \frac{p}{2} \rceil$,
- min. number of keys = $\lceil \frac{p-1}{2} \rceil$.

A Internal node of a B^+ tree has

- atleast $\lceil \frac{p}{2} \rceil$ child pointers and at most p pointers

Bitmap Indices

ID	Gender	Income Level
76766	m	L1
22222	f	L2
12121	f	L1
15151	m	L4
58583	f	L3

Bitmaps for Gender

Gender	Bits
m	10010
f	01101

Index in SQL

Consider a Relation: **Student (Student_ID, Name, Address, Age, Gender, Semester)**

Create Index

```
CREATE INDEX idx_stud ON Student (name, address)
```

Drop Index

```
DROP INDEX idx_stud
```

Bitmap Index

```
CREATE BITMAP INDEX idx_Gender ON Student (Gender)
```

Rules for Indexing

- Indexes lead to Access – Update Tradeoff
- Index the Correct Tables
- Index the Correct Columns
- Limit the Number of Indexes for Each Table
- Choose the Order of Columns in Composite Indexes
- Gather Statistics to Make Index Usage More Accurate
- Drop Indexes That Are No Longer Required