UNIT 3
Lecture 50 & 51
Indexing

Indexing

- An index is a collection of data entries which is used to locate a record in a file.
- Index table records consist of two parts, the first part consists of value of prime or non-prime attributes of file record known as indexing field and, the second part consists of a pointer to the location where the record is physically stored in memory.
- In general, index table is like the index of a book, that consists of the name of topic and the page number. During searching of a file record, index is searched to locate the record memory address instead of searching a record in secondary memory.
- On the basis of properties that affect the efficiency of searching, the indexes can be classified into two categories :
    1. Ordered indexing
    2. Hashed indexing

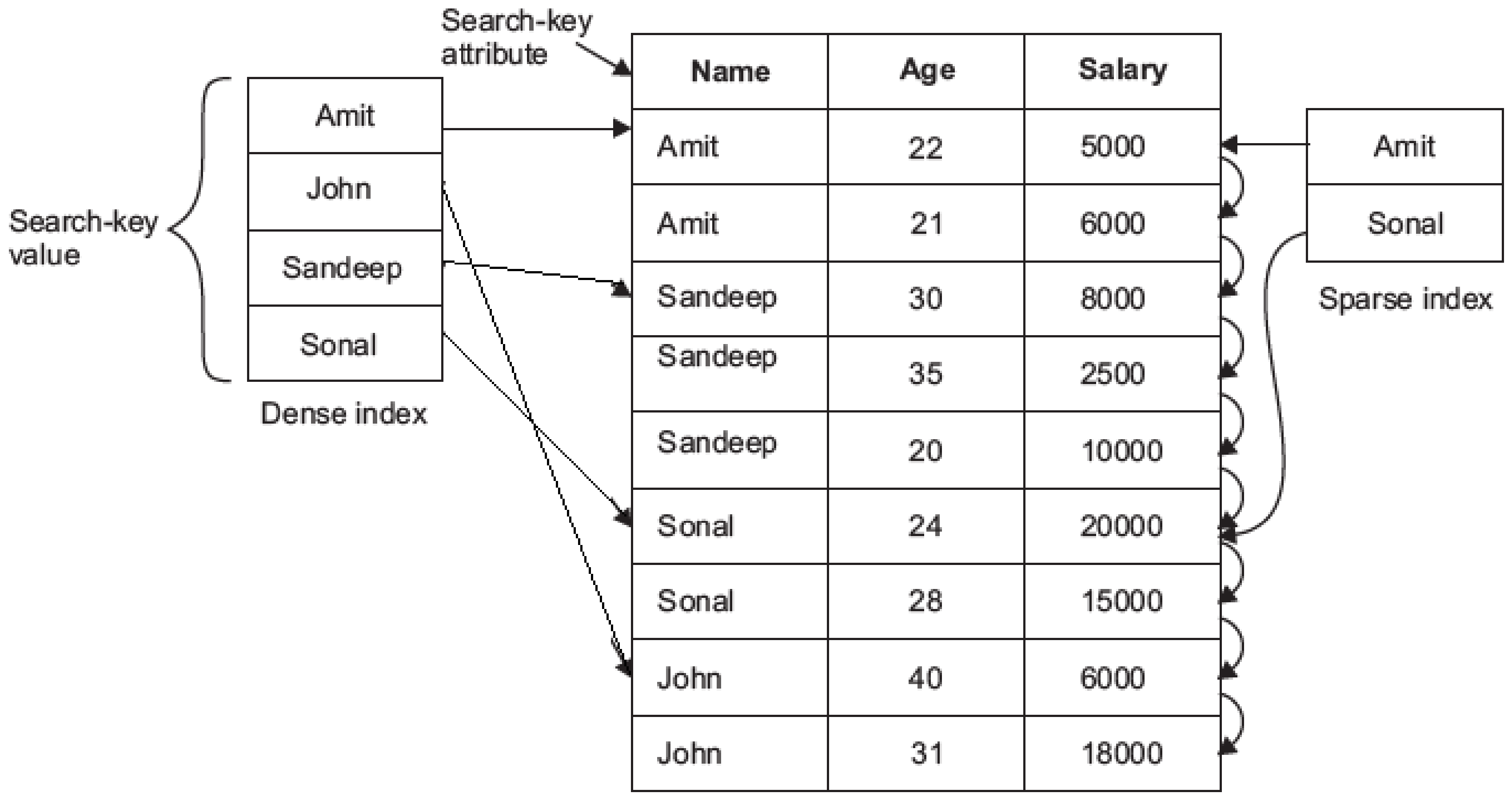
Ordered Indexing

- In ordered indexing, records of file are stored in some sorted order in physical memory.
- The values in the index are ordered (sorted) so that binary search can be performed on the index.
- Ordered indexes can be divided into two categories.
  1. Dense indexing
  2. Sparse indexing

Dense Index

- In dense indexing there is a record in index table for each unique value of the search-key attribute of file and a pointer to the first data record with that value.
- The other records with the same value of search-key attribute are stored sequentially after the first record.
- The order of data entries in the index differs from the order of data records.

Records and Record Types



Name	Age	Salary
Amit	22	5000
Amit	21	6000
Sandeep	30	8000
Sandeep	35	2500
Sandeep	20	10000
Sonal	24	20000
Sonal	28	15000
John	40	6000
John	31	18000

Dense Index

- ***Advantages of Dense index***
  1. It is efficient technique for small and medium sized data files.
  2. Searching is comparatively fast and efficient.
- ***Disadvantages of Dense index***
  1. Index table is large and require more memory space.
  2. Insertion and deletion is comparatively complex.
  3. In-efficient for large data files.

Sparse Index

- On contrary, in sparse indexing there are only some records in index table for unique values of the search-key attribute of file and a pointer to the first data record with that value.
- To search a record in sparse index we search for a value that is less than or equal to value in index for which we are looking.
- After getting the first record, linear search is performed to retrieve the desired record.
- There is at most one sparse index since it is not possible to build a sparse index that is not clustered.

Sparse Index

- ***Advantages of Sparse index*** **:**
  1. Index table is small and hence save memory space (specially in large files).
  2. Insertion and deletion is comparatively easy.
- ***Disadvantages of Sparse index*** **:**
  - Searching is comparatively slower, since index table is searched and then linear search is performed inside secondary memory.

Clustered and Non-Clustered Indexes

- ***Clustered index*** **:**
  1. In clustering, index file records are stored physically in order on a non-prime key attribute that does not have a unique value for each record.
  2. The nonprime key field is known as clustering field and index in known as clustering index.
  3. It is same as dense index.
  4. A file can have at most one clustered index as it can be clustered on at most one search key attribute.
  5. It may be sparse.
- ***Non-clustered index*** **:**
  1. An index that is not clustered is known as non-clustered index.
  2. A data file can have more than one non-clustered index.

Primary index

- A primary index consists of all prime-key attributes of a table and a pointer to physical memory address of the record of data file.
- To retrieve a record on the basis of all primary key attributes, primary index is used for fast searching.
- A binary search is done on index table and then directly retrieve that record from physical memory.
- It may be sparse.

Primary Index

- ***Advantages of Primary index***
  1. Search operation is very fast.
  2. Index table record is usually smaller.
  3. A primary index is guaranteed not to duplicate.
- ***Disadvantages of Primary index***
  1. There is only one primary index of a table. To search a record on less than all prime-key attributes, linear search is performed on index table.
  2. To create a primary index of an existing table, records should be in some sequential order otherwise database is required to be adjusted.

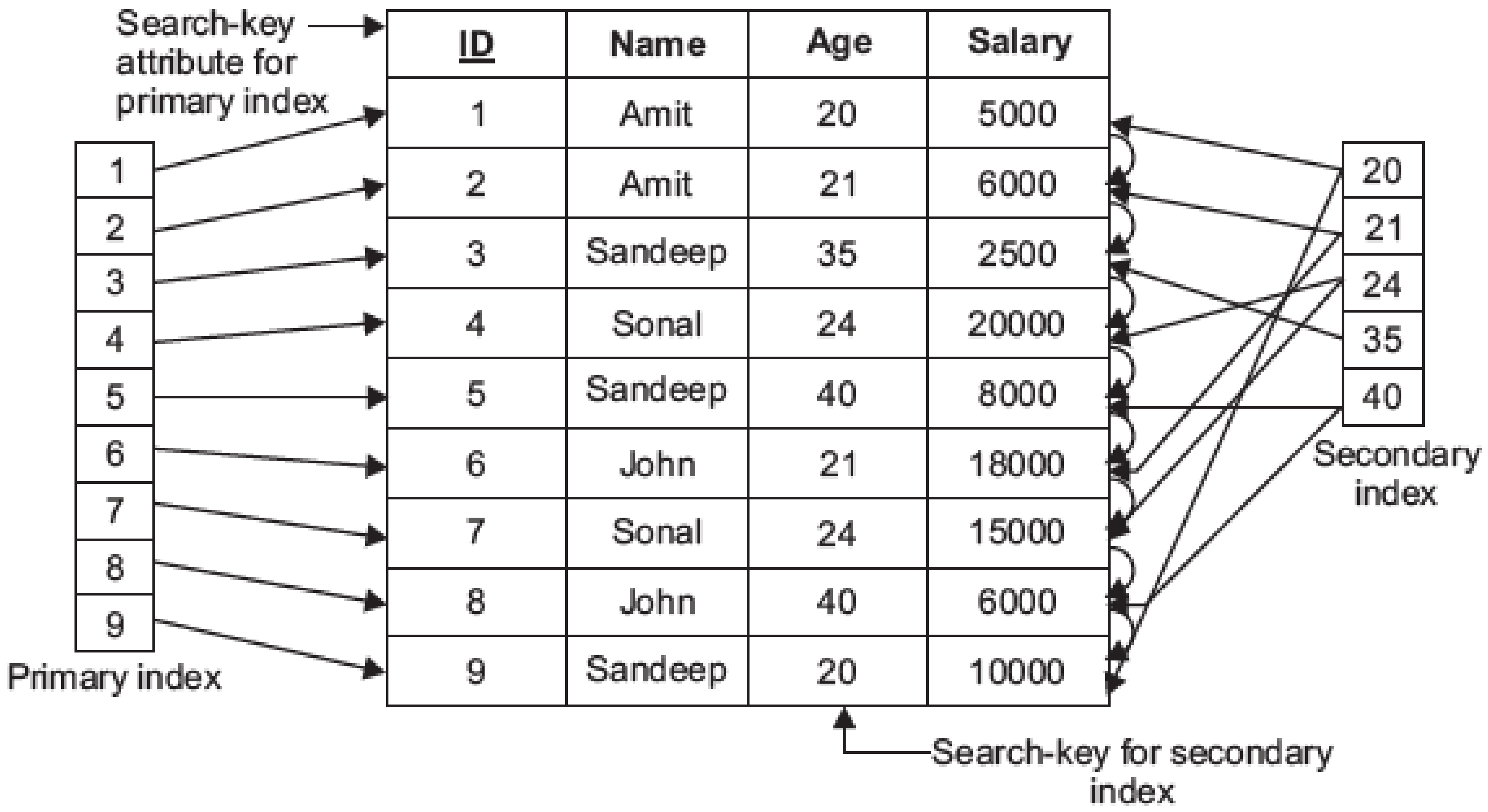
Secondary index

- A secondary index provides a secondary means of accessing a data file.
- A secondary index may be on a candidate key field or on non-prime key attributes of a table.
- To retrieve a record on the basis of non-prime key attributes, secondary index can be used for fast searching.
- Secondary index must be dense with a index entry for every search key value and a pointer to every record in a file.

Secondary index

- ***Advantages of Secondary index***
  1. Improve search time if search on non-prime key attributes.
  2. A data file can have more than one secondary index.
- ***Disadvantages of Secondary index***
  1. A secondary index usually needs more storage space.
  2. Search time is more than primary index.
  3. They impose a significant overhead on the modification of database.

Primary and Secondary Index


Search-key attribute for primary index
Primary index
Secondary index
Search-key for secondary index
1
2
3
4
5
6
7
8
9
20
21
24
35
40

ID	Name	Age	Salary
1	Amit	20	5000
2	Amit	21	6000
3	Sandeep	35	2500
4	Sonal	24	20000
5	Sandeep	40	8000
6	John	21	18000
7	Sonal	24	15000
8	John	40	6000
9	Sandeep	20	10000

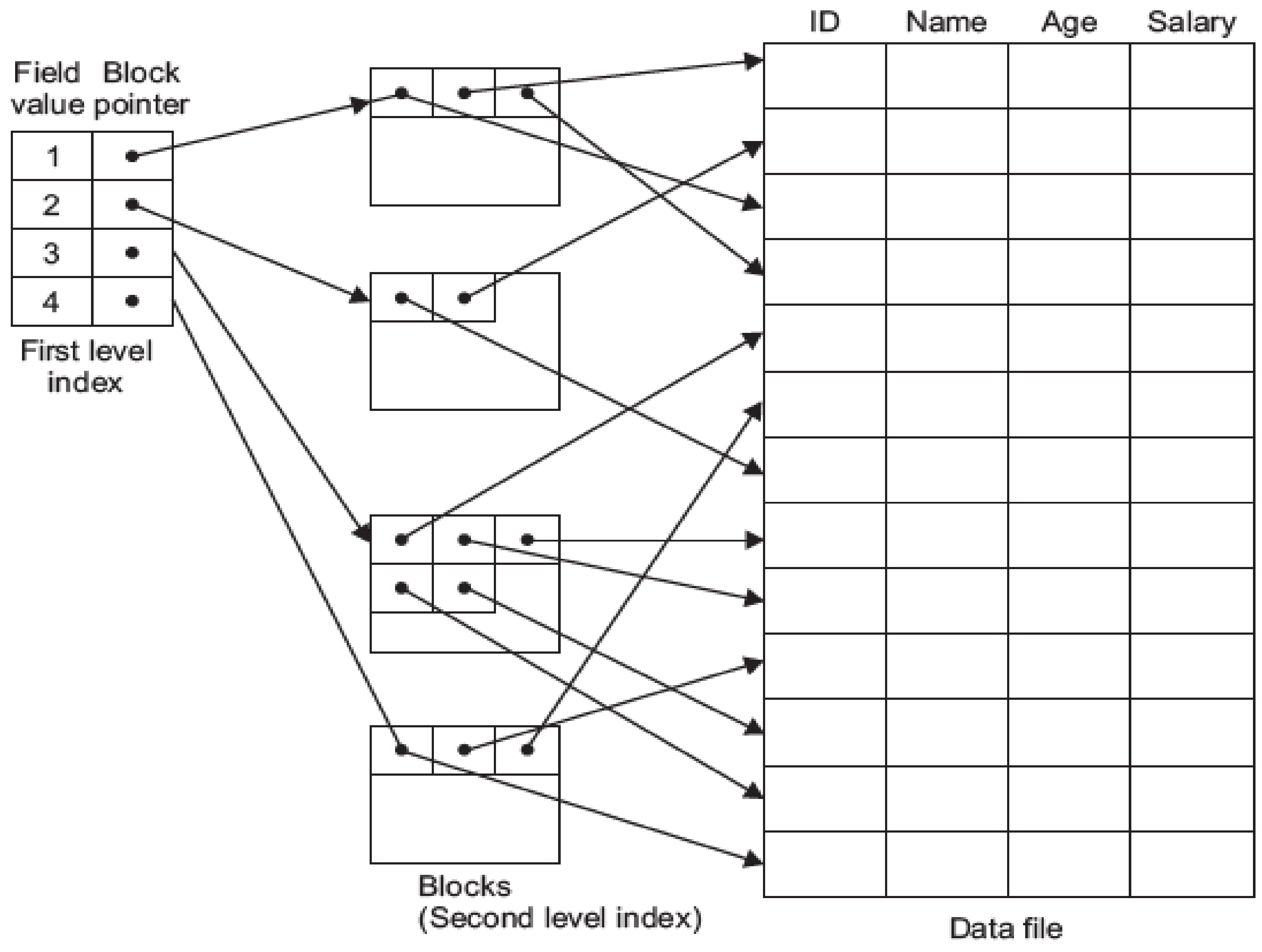
Single and Multilevel Indexes

- ***Single level indexes*** **:**
  1. A single stage index for a data file is known as single level index.
  2. A single level index cannot be divided.
  3. It is useful in small and medium size data files. If the file size is bigger, then single level, indexing is not a efficient method.
  4. Searching is faster than other indexes for small size data files.

Single and Multilevel Indexes

- ***Multilevel indexes*** **:**
  1. A single index for a large size data file increases the size of index table and increases the search time that results in slower searches.
  2. The idea behind multilevel indexes is that, a single level index is divided into multiple levels, which reduces search time.
  3. In multilevel indexes, the first level index consists of two fields, the first field consists of a value of search key attributes and a second field consists of a pointer to the block (or second level index) which consists that values and so on.
  4. To search a record in multilevel index, binary search is used to find the largest of all the small values or equal to the one that needs to be searched. The pointer points to a block of the inner index. After reaching to the desired block, the desired record is searched (in case of two-level indexing) otherwise again the largest of the small values or equal to the one that needs to be searched and so on.

Multilevel Index



Single and Multilevel Indexes

- A multilevel index considers the index file, which is refer to as the **first** (or **base**) **level** of a multilevel index, as an ***ordered file*** with a *distinct value* for each *K*(*i*).
- Therefore, by considering the first-level index file as a sorted data file, we can create a primary index for the first level; this index to the first level is called the **second level** of the multilevel index.
- Because the second level is a primary index, we can use block anchors so that the second level has one entry for *each block* of the first level.
- The blocking factor *bfri* for the second level—and for all subsequent levels—is the same as that for the first-level index because all index entries are the same size; each has one field value and one block address.

Single and Multilevel Indexes

- If the first level has *r*1 entries, and the blocking factor—which is also the fan-out—for the index is *bfri* = *fo*, then the first level needs $\lceil(r1/fo)\rceil$ blocks, which is therefore the number of entries *r*2 needed at the second level of the index.
- We can repeat this process for the second level. The **third level**, which is a primary index for the second level, has an entry for each second-level block, so the number of third-level entries is *r*3 = $\lceil(r2/fo)\rceil$.
- Note that we require a second level only if the first level needs more than one block of disk storage, and, similarly, we require a third level only if the second level needs more than one block. We can repeat the preceding process until all the entries of some index level *t* fit in a single block.

Single and Multilevel Indexes

- This block at the t^{th} level is called the **top** index level.
- Each level reduces the number of entries at the previous level by a factor of *fo*—the index fan-out—so we can use the formula $1 \leq (r1/((fo)^t))$ to calculate t.
- Hence, a multilevel index with $r1$ first-level entries will have approximately t levels, where $t = \lceil (\log_{fo}(r1)) \rceil$.
- When searching the index, a single disk block is retrieved at each level. Hence, t disk blocks are accessed for an index search, where t is the *number of index levels*.

GATE Questions

Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 bytes, and the size of a block pointer is 10 bytes. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively

(A) 8 and 0

(B) 128 and 6

(C) 256 and 4

(D) 512 and 5

[GATE 2008]

GATE Questions

- Content of an index will be <key, block pointer> and so will have size 6 + 10 = 16.
- In the first level, there will be an entry for each record of the file. So,total size of first-level index = 16384 * 16
- No. of blocks in the first-level = Size of first-level index / block size

  = 16384 * 16 / 1024

  = 16 * 16

  = 256
- In the second-level there will be an entry for each block in the first level.
- So, total number of entries = 256 and
- total size of second-level index = No. of entries * size of an entry = 256 * 16
- No. of blocks in second-level index = Size of second-level index / block size

  = 256 * 16 / 102

  = 4

Hashed Indexing

- To overcome the disadvantages of ordered indexing, a hash index can be created for a data file.
- Hashing allow us to avoid accessing an index structure.
- A hashed index consists of two fields, the first field consists of search key attribute values and second field consists of pointer to the hash file structure.
- Hashed indexing is based on values of records being uniformly distributed using a hashed function.

B - tree Index

- In B-Tree index files, tree structure is used.
- A B-tree of order m is an m-way search tree with the following properties.
  - Each node of the tree, except the root and leaves, has at least $\left\lceil \frac{n}{2} \right\rceil$subtrees and no more than *n* subtrees. It ensures that each node of tree is at least half full.
  - The root of the tree has at least two subtrees, unless it is itself a leaf. It forces the tree to branch early.
  - All leaves of the tree are on the same level. It keeps the tree nearly balanced.
- To overcome the performance degradation w.r.t. growth of files in index sequential files, B-Tree index files are used.
- It is a kind of multilevel index file organization.
- In tree structure, search starts from the root node and stops at leaf node.

B - tree Index

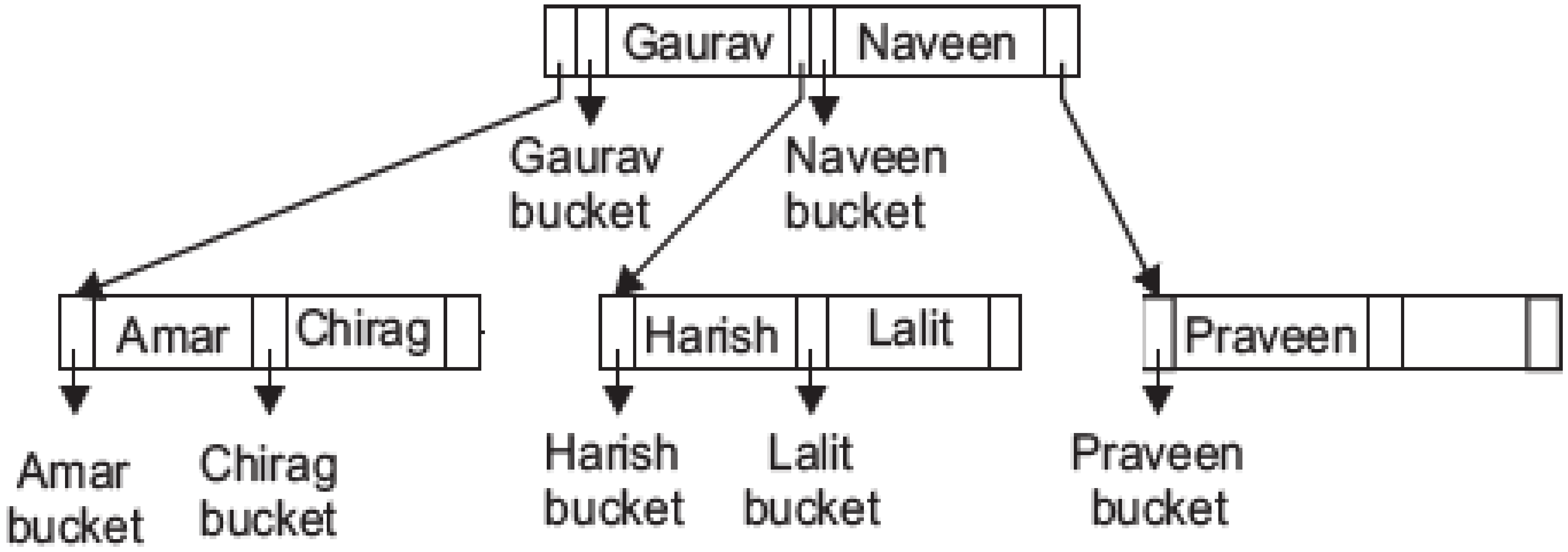
- **Non – Leaf Node**
  - In non-leaf node, there are two pointers. Pointer P_i points to any other node. Pointer R_i points to the block of actual records or storage area of records. K_i represents the key value.

P_1	R_1	K_1	P_2	R_2	K_2	- -	P_{m-1}	R_{m-1}	K_{m-1}	P_m

- **Leaf Node**
  - In leaf node, there is only one pointer P_i which points to block of actual records or storage area of records. K_i represents the key value.

P_1	K_1	P_2	K_2	- - - - - - - - - - - - - -	P_{n-1}	K_{n-1}	P_n

B - tree Example



B – tree Operations

- ***Searching a record* :** Searching a record with its key value starts from root node. It is possible to find desired record without going to leaf node in B-tree.
- ***Deletion of a record* :** Deletion in B-tree is a complicated process. If desired entry is in leaf node, then, simply delete it otherwise find a proper replacement for that entry.
- ***Insertion of a record* :** B-tree is a balanced tree and insertion of a new record in B-tree cause node splits and therefore affects the height of the tree.

B - tree

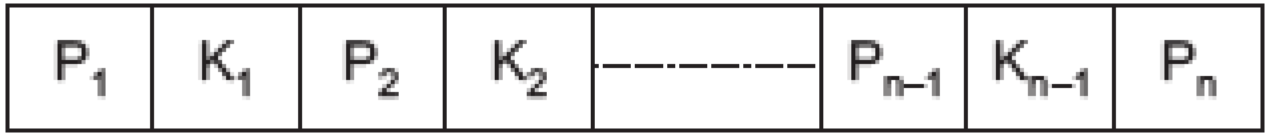
- ***Advantages***
  1. Key value appears only once in the node.
  2. Searching is faster than indexed sequential files.
  3. Performance is maintained w.r.t. growth of file size.
- ***Disadvantages***
  1. Updation of records are more complicated then B^+ trees.
  2. Less efficient than B^+ trees and direct files.
  3. Searching time is still proportional to logarithm of the number of search keys.

B^+ - tree

- A B^+-tree is a kind of balanced tree.
- The length of every path from root of the tree to the leaf of the tree are same.
- The number of children *n* is fixed for a particular tree.
- Each non-leaf node in the tree has between (*n*/2) and *n* children.
- Index used in B^+-tree files is multilevel index but its structure differ from the index structure used in multilevel index-sequential files.

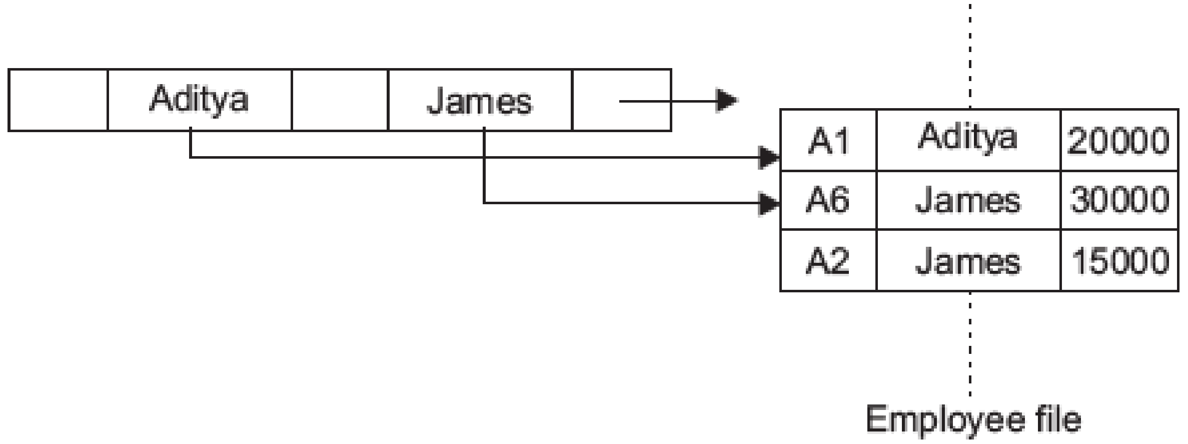
B^+ - tree

- A typical node of B^+-tree contains up to *n* – 1 search key values and n pointers.
- K_i represents search key value and P_i represents pointer to a file record.
- In B^+-trees search key values are in sorted order, thus, if *i* < *j*, then $K_i < K_j$.
- The number of pointers in a node is called ***Fanout*** of the node.



B^+ - tree

- **Leaf nodes :**
  - In a leaf node, a pointer P_i points to either a file record having search key value K_i or to a bucket of pointers where each pointer to a file record with search key value K_i (In case of secondary index in which index is made of nonprime key attributes and file is not sorted in search key value order).



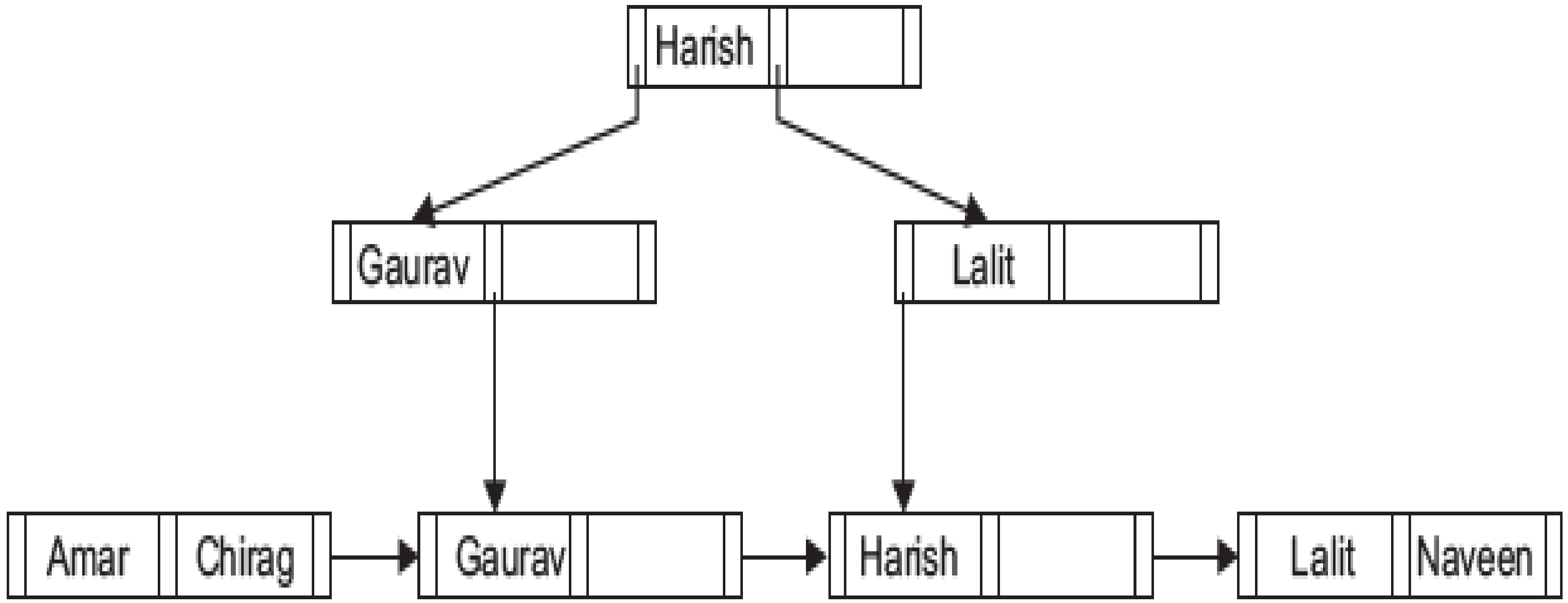
B^+ - tree

- **Non-leaf nodes :**
  1. The structure of non-leaf nodes are same as of leaf node with a single difference that pointer P_i points to the tree nodes.
  2. It contains at least (*n*/2) pointers and a maximum of *n* pointers.
- **Root nodes :**
  1. A root node contains at least 2 pointers and a maximum of less than $\left\lceil \frac{n}{2} \right\rceil$ pointers.
  2. A B^+- tree contains only a single node if root node consists only a single pointer.

B^+ - tree

- A pointer P_i with a search key value K_i in a non-leaf node points to a part of subtree having search key values less than K_i, and greater than or equal to K_{i-1}.
- Pointer P_m points to a part of subtree having search key values greater than or equal to K_{m-1}, Pointer P_1 points to the part of subtree having search key values less than K_1.

B⁺ - tree


Harish
Gaurav
Lalit
Amar
Chirag
Gaurav
Harish
Lalit
Naveen

B^+ - tree

- ***Searching a record*** **:**
  - Searching a record with its key value starts from root node.
  - It is not possible to find desired record without going to leaf node in B^+-tree.
- ***Deletion of a record*** **:**
  - Deletion in B^+-tree is complicated process in some special cases.
  - If desired entry is in leaf node then, simply delete it and if bucket is associated then bucket becomes empty as a result.
  - If deletion causes a node to be less than half full, then it is combined with its neighboring nodes and it is propagated all the way to the root.
- ***Insertion of a record*** **:**
  - To insert a new record in B^+-tree, first search a leaf node with same search key value.
  - Simply add a new record to file or add a pointer in bucket, which points to the record.
  - If search key value does not appear, simply insert the value in leaf node in correct position or create a new bucket with the appropriate pointer if necessary.

B^+ - tree

- ***Advantages of B^+-trees***
  1. It provides a reasonable performance for direct access.
  2. It provides an excellent performance for sequential and range accesses.
  3. Searching is faster.
- ***Disadvantages of B^+-trees***
  1. Insertion is more complex than B-trees.
  2. Deletion is more complex than B-trees.
  3. Search key values are duplicated which results in wastage of memory space.

Order of B^+ - tree

- To calculate the order *p* of a B+-tree,
- Suppose that the search key field is $V = 9$ bytes long, the block size is $B = 512$ bytes, a record pointer is $Pr = 7$ bytes, and a block pointer/tree pointer is $P = 6$ bytes. An internal node of the B+-tree can have up to *p* tree pointers and $p - 1$ search field values; these must fit into a single block.
- Hence, we have:

  $(p * P) + ((p - 1) * V) \leq B$

  $(p * 6) + ((p - 1) * 9) \leq 512$

  $(15 * p) \leq 521$
- We can choose *p* to be the largest value satisfying the above inequality, which gives $p = 34$.

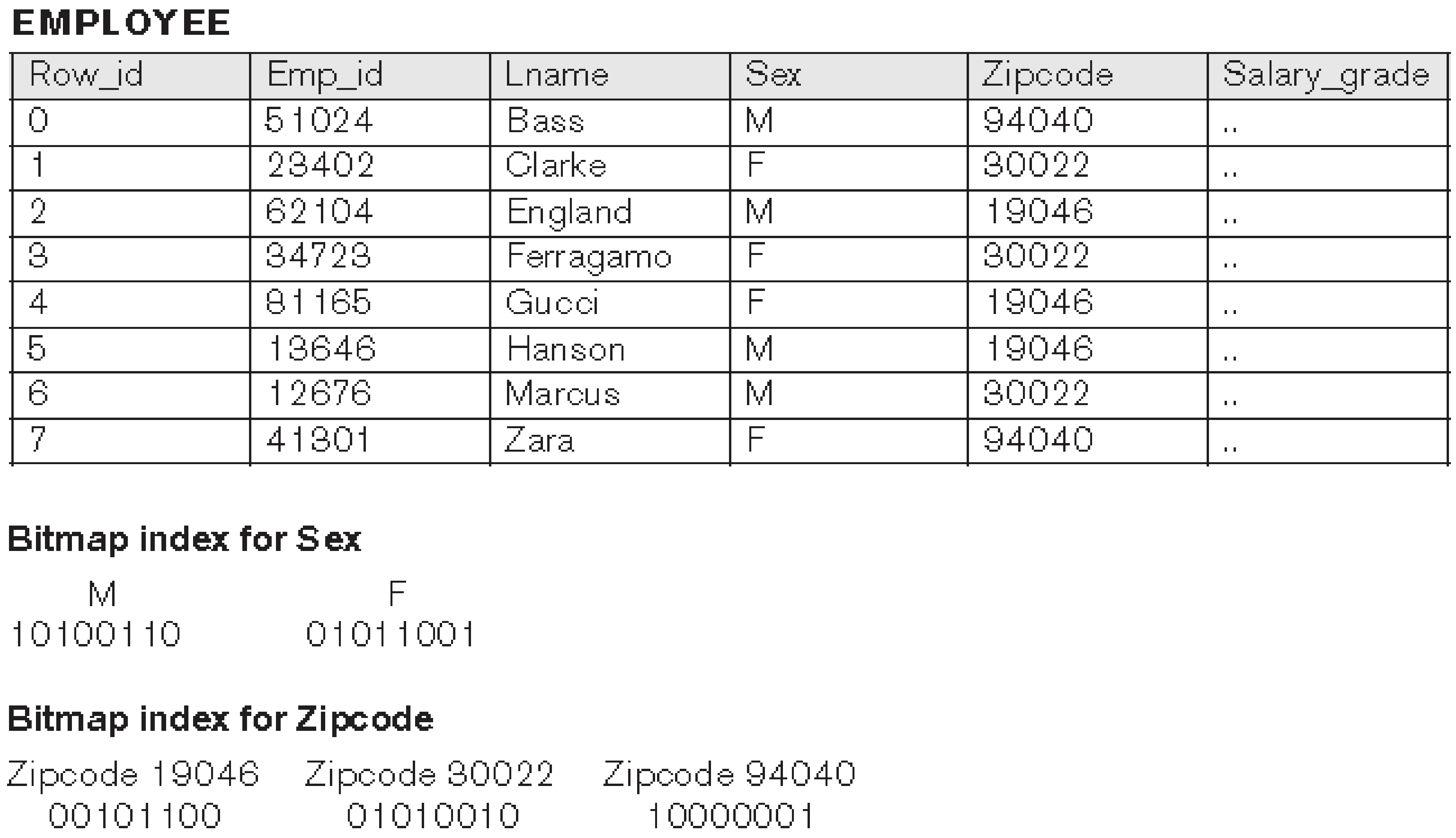
Difference between B-tree and B^+-tree

Sno	B^+ - tree	B - Tree
1	Search keys can be repeated.	Search keys cannot be redundant.
2	Data (Record Pointer) is only saved on the leaf nodes.	Both leaf nodes and internal nodes can store data (Record Pointer).
3	Data stored on the leaf node makes the search more accurate and faster.	Searching is slow due to data stored on Leaf and internal nodes.
4	Deletion is not difficult as an element is only removed from a leaf node.	Deletion of elements is a complicated and time-consuming process.
5	Linked leaf nodes make the search efficient and quick.	You cannot link leaf nodes.

Bitmap Index

- The **bitmap index** is another popular data structure that facilitates querying on multiple keys.
- Bitmap indexing is used for relations that contain a large number of rows.
- It creates an index for one or more columns, and each value or value range in those columns is indexed.
- Typically, a bitmap index is created for those columns that contain a fairly small number of unique values.
- To build a bitmap index on a set of records in a relation, the records must be numbered from 0 to *n* with an id (a record id or a row id) that can be mapped to a physical address made of a block number and a record offset within the block.

Bitmap Index



EMPLOYEE

Row_id	Emp_id	Lname	Sex	Zipcode	Salary_grade
0	51024	Bass	M	94040	..
1	23402	Clarke	F	30022	..
2	62104	England	M	19046	..
3	34723	Ferragamo	F	30022	..
4	81165	Gucci	F	19046	..
5	13646	Hanson	M	19046	..
6	12676	Marcus	M	30022	..
7	41301	Zara	F	94040	..

Bitmap index for Sex

M	F
10100110	01011001

Bitmap index for Zipcode

Zipcode 19046	Zipcode 30022	Zipcode 94040
00101100	01010010	10000001

GATE Questions

In a B^+ tree, if the search – key value is 8 bytes long, the block size is 512 bytes and the block pointer size is 2 bytes, then maximum order of the B^+ tree is _________________.

[GATE 2017]

GATE Questions

B^+ Trees are considered BALANCED because

(A) the lengths of the paths from the root to all leaf nodes are all equal.

(B) the lengths of the paths from the root to all leaf nodes differ from each other by at most 1.

(C) the number of children of any two non-leaf sibling nodes differ by at most 1.

(D) the number of records in any two leaf nodes differ by at most 1.

[GATE 2016]

GATE Questions

A file is organized so that the ordering of data records is the same as or closes to the ordering of data entries in some index. Then that index is called

(A) Dense

(B) Sparse

(C) Clustered

(D) Unclustered

[GATE 2015]

GATE Questions

An index is clustered, if

(A) it is on a set of fields that form a candidate key.

(B) it is on a set of fields that include the primary key.

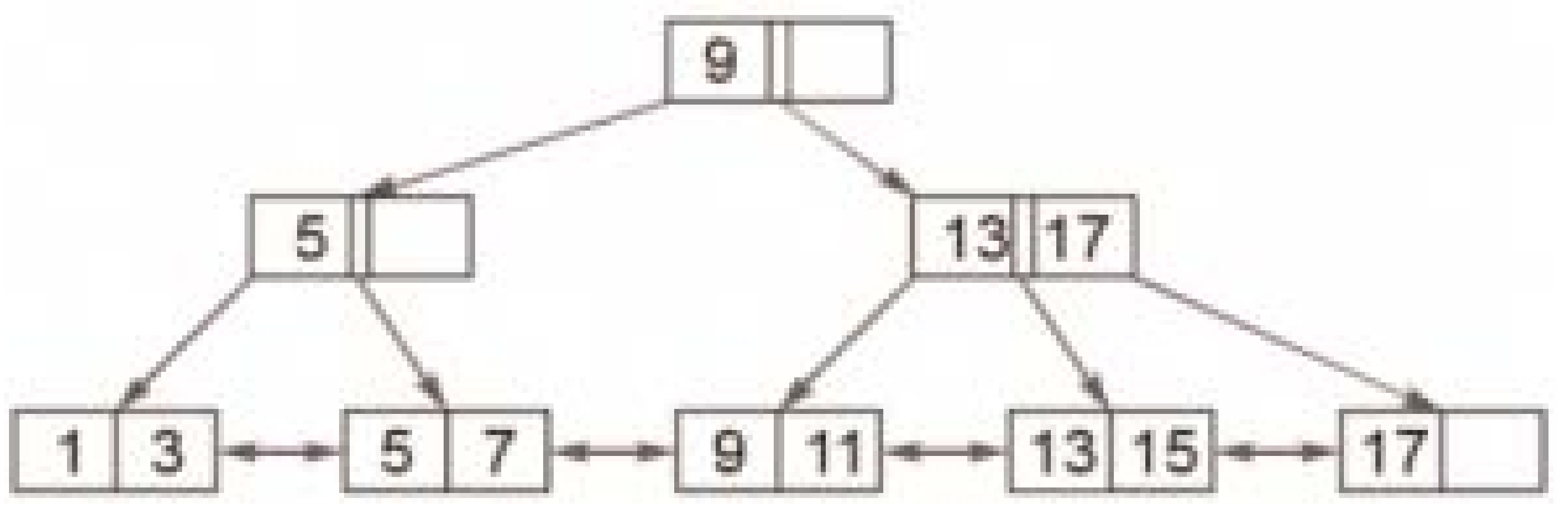
(C) the data records of the file are organized in the same order as the data entries of the index.

(D) the data records of the file are organized not in the same order as the data entries of the index.

[GATE 2013]

GATE Questions

With reference to the B^+ tree index of order 1 shown below, the minimum number of nodes (including the root node) that must be fetched in order to satisfy the following query: "Get all records with a search key greater than or equal to 7 and less than 15" is ________.



[GATE 2015]

GATE Questions

Consider a B^+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is ____ .

[GATE 2015]

GATE Questions

Consider the relational table r with sufficient number of records, having attributes A_1, A_2,… An and let 1≤p≤n. Two queries Q1 and Q2 are given below :

Q1: $\Pi A1,\ldots An\ (\sigma_{Ap\ =\ c}\ (r))$ where c is const

Q2: $\Pi A1,\ldots An\ (\sigma_{c1 \leq Ap \leq c2}\ (r))$ where c1 and c2 are const

The database can be configured to do order indexing on Ap or hashing on Ap. Which of the following statements is TRUE?

(A) Ordered indexing will always outperform hashing for both queries

(B) Hashing will always outperform ordered indexing for both queries

(C) Hashing will outperform ordered indexing on Q1, but not on Q2

(D) Hashing will outperform ordered indexing on Q2, but not on Q1.

[GATE 2011]

GATE Questions

Consider a B$^+$ tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?

(A) 1

(B) 2

(C) 3

(D) 4

[GATE 2010]

GATE Questions

The following key values are inserted into a B+ tree in which order of the internal nodes is 3, and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B^+ tree is initially empty.

10, 3, 6, 8, 4, 2, 1

The maximum number of times leaf nodes would get split up as a result of these insertions is

(A) 2

(B) 3

(C) 4

(D) 5

[GATE 2009]

GATE Questions

A B–tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place?

(A) 3

(B) 4

(C) 5

(D) 6

[GATE 2008]

GATE Questions

A clustering index is defined on the fields which are of type

(A) non – key and ordering

(B) non-key and non – ordering

(C) key and ordering

(D) key and non-ordering

[GATE 2008]

GATE Questions

The order of a leaf node in a B^+ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

(A) 63

(B) 64

(C) 67

(D) 68

[GATE 2007]

GATE Questions

Which one of the following is a key factor for preferring B$^+$ trees to binary search trees for indexing database relations?

(A) Database relations have a large number of records

(B) Database relations are sorted on the primary key

(C) B$^+$ trees require less memory than binary search trees

(D) Data transfer from disks is in blocks

[GATE 2005]

GATE Questions

The order of an internal node in a B^+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

(A) 24

(B) 25

(C) 26

(D) 27

[GATE 2004]

GATE Questions

In the index allocation scheme of blocks to a file, the maximum possible size of the file depends on

(A) the size of the blocks, and the size of the address of the blocks.

(B) the number of blocks used for the index, and the size of the blocks.

(C) the size of the blocks, the number of blocks used for the index, and the size of the address of the blocks.

(D) None of the above

[GATE 2002]

GATE Questions

A B^+ tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 bytes, disk blocks are of size 512 bytes, and index pointers are of size 4 bytes. Given this scenario, what would be the best choice of the degree (i.e. the number of pointers per node) of the B^+ tree?

(A) 16

(B) 42

(C) 43

(D) 44

[GATE 2002]

GATE Questions

B^+ trees are preferred to binary trees in databases because

(A) Disk capacities are greater than memory capacities

(B) Disk access is much slower than memory access

(C) Disk data transfer rates are much less than memory data transfer rates

(D) Disks are more reliable than memory

[GATE 2000]

GATE Questions

Which of the following is correct?

(A) B–trees are for storing data on disk and B^+ trees are for main memory.

(B) Range queries are faster on B^* trees.

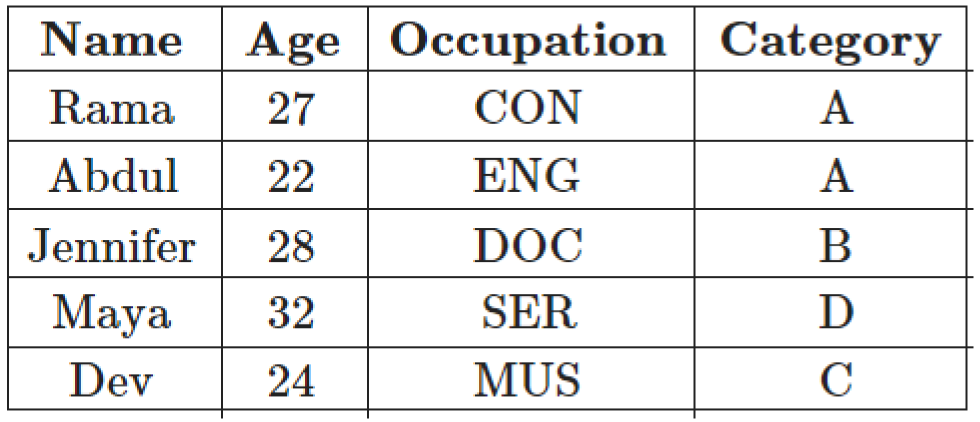
(C) B–trees are for primary indexes and B^+ trees are for secondary indexes.

(D) The height of a B^+ tree is independent of the number of records.

[GATE 1999]

GATE Questions

There are five records in a database.



Name	Age	Occupation	Category
Rama	27	CON	A
Abdul	22	ENG	A
Jennifer	28	DOC	B
Maya	32	SER	D
Dev	24	MUS	C

There is an index file associated with this and it contains the values 1,3,2,5 and 4. Which one of the fields is the index built from?

(A) Age

(B) Name

(C) Occupation

(D) Category

[GATE 1998]

GATE Questions

A data file consisting of 1,50,000 student-records is stored on a hard disk with block size of 4096 bytes. The data file is sorted on the primary key RollNo. The size of a record pointer for this disk is 7 bytes. Each student-record has a candidate key attribute called ANum of size 12 bytes. Suppose an index file with records consisting of two fields, ANum value and the record pointer the corresponding student record, is built and stored on the same disk. Assume that the records of data file and index file are not split across disk blocks. The number of blocks in the index file is _________ .

[GATE 2021]

GATE Questions

An ISAM (indexed sequential) file consists of records of size 64 bytes each, including key field of size 14 bytes. An address of a disk block takes 2 bytes. If the disk block size is 512 bytes and there are 16K records, compute the size of the data and index areas in terms of number blocks. How many levels of tree do you have for the index?

[GATE 1993]

For Video lecture on this topic please subscribe to my youtube channel.

The link for my youtube channel is

https://www.youtube.com/channel/UCRWGtE76JlTp1iim6aOTRuw?sub_confirmation=1