

Q1. WHAT ARE THE DIFFERENT PHYSICAL STORAGE MEDIA USED FOR DATA STORAGE? COMPARE DIFFERENT PHYSICAL STORAGE MEDIA USED FOR DATA STORAGE.

1) Cache--

- The cache is the fastest and most costly form of storage.
- Cache memory is small;
- Its use is managed by the computer system hardware.

2) Main memory-

- The storage medium used for data that are available to be operated on is main memory.
- RAM

3) Flash memory--

- Also known as electrically erasable programmable read-only memory (EEPROM),
- Flash memory differs from main memory in that data survive power failure

4) Magnetic-disk storage—

- The primary medium for the long-term on-line storage of data is the magnetic disk.
- Optical storage.

- The most popular forms of optical storage are the compact disk (CD),
- Which can hold about 700 megabytes of data,
- And the digital versatile disk (DVD) which can hold 4.7 GB of data

5) Tape storage --

- Tape storage is used primarily for backup and archival data.
- Although magnetic tape is much cheaper than disks, access to data is much slower,
- Because the tape must be accessed sequentially from the beginning.

Q2. EXPLAIN THE FOLLOWING TERMS – SEEK TIME, ROTATIONAL LATENCY, DATA TRANSFER RATE.

1) Seek Time --

- It is the time taken to reposition and settle the arm and the head over the correct track.
- The lower the seek time, the faster the I/O operation.

2) Rotational Latency --

- To access data, the actuator arm moves the R/W head over the platter to a particular track.

- while the platter spins to position the requested sector under the R/W head.
- The time taken by the platter to rotate and position the data under the R/W head is called rotational latency.
- This latency depends on the rotation speed of the spindle and is measured in milliseconds.

3)Data Transfer Rate

- The data transfer rate refers to the average amount of data per unit time that the drive can deliver.

Q3. WHAT IS RAID? EXPLAIN STRIPING, MIRRORING AND PARITY.

RAID levels are defined on the basis of

- 1.Striping
- 2.Mirroring
- 3.Parity

Table Raid Levels

LEVELS	BRIEF DESCRIPTION
RAID 0	Striped array with no fault tolerance
RAID 1	Disk mirroring
RAID 3	Parallel access array with dedicated parity disk
RAID 4	Striped array with independent disks and a dedicated parity disk
RAID 5	Striped array with independent disks and distributed parity
RAID 6	Striped array with independent disks and dual distributed parity
Nested	Combinations of RAID levels. Example: RAID 1 + RAID 0

1) Stripping :

- A RAID set is a group of disks.
- Within each disk, a predefined number of contiguously addressable disk blocks are defined as strips.
- The set of aligned strips that spans across all the disks within the RAID set is called a stripe.

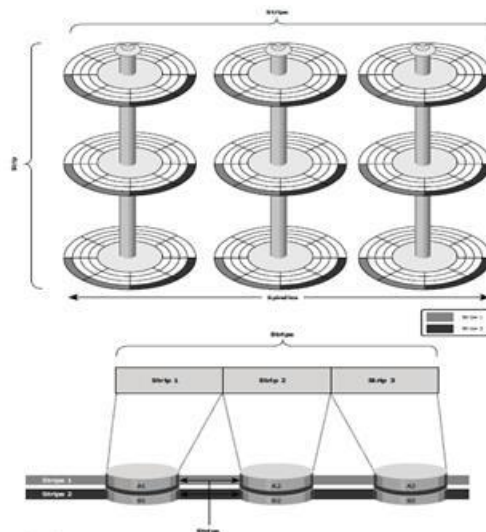
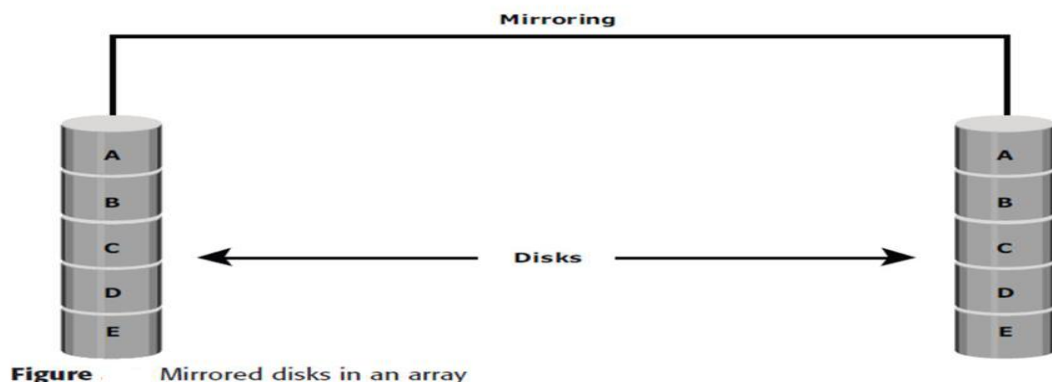


Figure Striped RAID set

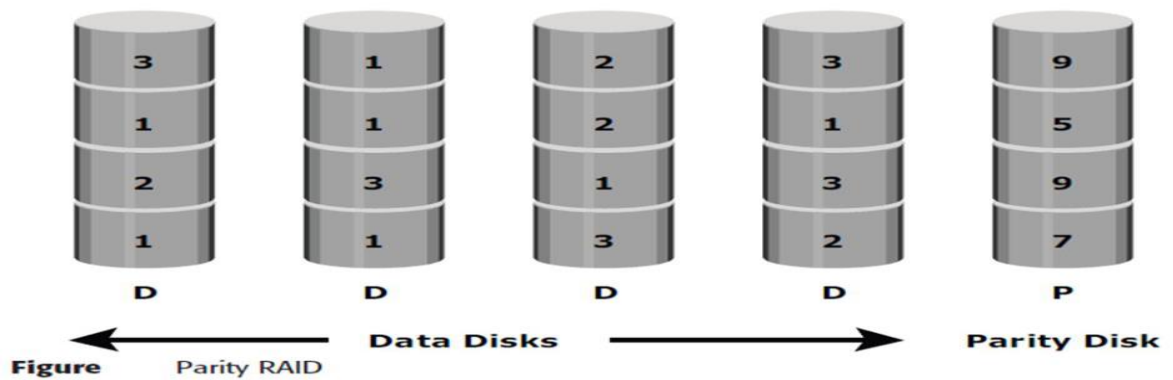
2) Mirroring :

- Data is stored on two different HDDs(hard disk drive), yielding two copies of data.
- In the event of one HDD failure,
- the data is intact on the surviving HDD .



3) Parity --

- Parity is a method of protecting striped data from HDD failure without the cost of mirroring.
- An additional HDD is added to the stripe width to hold parity,
- a mathematical construct that allows re-creation of the missing data.
- Parity is a redundancy check that ensures full protection of data without maintaining a full set of duplicate data.



Q4. HOW RECORDS ARE ORGANIZED IN FILES?

- possible ways of organizing records in files are--

1) Heap file organization.

- Any record can be placed anywhere in the file where there is space for the record.

- ordering of records.

2) Sequential file organization.

- Records are stored in sequential order, according to the value of a “search key” of each record.

- A sequential file is designed for efficient processing of records in sorted order based on some search-key.
- A search key is any attribute or set of attributes.
- It is difficult to maintain physical sequential order as records are inserted and deleted.
- since it is costly to move many record.
- Figure shows a sequential file of account records taken from our banking example.

- In that example, the records are stored in search-key order, using branch name as the search key.

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	




Figure Sequential file for *account* records.

3) Hash file organization.

- A hash function is computed on some attribute of each record.
- The result of the hash function specifies in which block of the file the record should be placed.

Q5. EXPLAIN FIXED LENGTH RECORDS AND VARIABLE LENGTH RECORDS.

One approach to map the database to files is to use several files--

- and to store records of only one fixed length in any given file.
- An alternative is to structure our files so that we can accommodate multiple lengths for records.
- files of fixed length records are easier to implement than are files of variable-length records.

Ex.

type deposit = record

account-number : number (10);

branch-name : varchar(20);

balance : number (10);

End

Total size of one record is 40 bytes.

Problems--

1. It is difficult to delete a record from this structure.

The space occupied by the record to be deleted must be filled with some other record of the file

2. Unless the block size happens to be a multiple of 40 (which is unlikely),

- Some records will cross block boundaries.
- part of the record will be stored in one block and part in another.

Variable-length records arise in database systems in several ways:

- Storage of multiple record types in a file
- Record types that allow variable lengths for one or more fields
- Record types that allow repeating fields.

Ex.

type account-list = record

branch-name : varchar(20);

account-info : array [1 .. ∞] of
record;

account-number : char(10);

balance : real;

End

End

Define account-info as an array with an arbitrary number of elements.

- the type definition does not limit the number of elements in the array,
- There is no limit on how large a record can be

Q6. WHAT IS DATA DICTIONARY? HOW IT CAN BE STORED?

1. A relational-database system needs to maintain data about the relations, such as the schema of the relations.

2. This information is called the data dictionary, or system catalogue.

3. Among the types of information that the system must store are these:

Names of the relations:

- Names of the attributes of each relation

- Domains and lengths of attributes
- Names of views defined on the database, and definitions of those views
- Integrity constraints (for example, key constraints)
- In addition, many systems keep the following data of users of the system:
 - Names of authorized users
 - Accounting information about users
 - Passwords or other information used to authenticate users.

Some database systems store this information by using special-purpose data structures and code.

- It is generally preferable to store the data about the database in the database itself.
- The exact choice of how to represent system data by relations must be made by the system designers.

One possible representation with primary keys underlined is

- Relation-metadata (relation-name, number-of-attributes, storage-organization, location)
- Attribute-metadata (attribute-name, relation-name, domain-type, position, length)
- User-metadata (user-name, encrypted-password, group)

- Index-metadata (index-name, relation-name, index-type, index-attributes)
- View-metadata (view-name, definition)

Q7. EXPLAIN BUFFER REPLACEMENT POLICIES.

- FIFO –First in First Out
- LIFO –Last in First Out
- LRU –Least recently used
- MRU –Most recently used
- Optimal –Assumes access pattern is known
- Random

Q8. WHAT IS USE OF INDEXING? COMPARE PRIMARY INDEX WITH SECONDARY INDEX.

- To allow fast access, design additional structures associate with files.

Q9. EXPLAIN CLUSTERING INDEX WITH NON-CLUSTERING INDEX.

1) Clustered Index--

- Clustered index is a type of index which sorts the data rows in the table on their key values.
- In the Database, there is only one clustered index per table.

- A clustered index defines the order in which data is stored in the table which can be sorted in only one way.
- So, there can be only a single clustered index for every table.

2)Non-clustered index--

- A Non-clustered index stores the data at one location and indices at another location.
- The index contains pointers to the location of that data.
- A single table can have many non-clustered indices, as an index in the non-clustered index is stored in different places.

Q10. COMPARE DENSE AND SPARSE INDEXING TECHNIQUES.

1. It is generally faster to locate a record if we have a dense index rather than a sparse index.
2. However, sparse indices have advantages over dense indices in that they require less space and they impose less maintenance overhead for insertions and deletions.
There is a trade-off that the system designer must make between access time and space overhead.
3. Although the decision regarding this trade-off depends on the specific application,

4. A good compromise is to have a sparse index with one index entry per block.
5. Once we have brought the block in main memory, the time to scan the entire block is negligible.
6. Using this sparse index, we locate the block containing the record that we are seeking.

Q11. EXPLAIN MULTI-LEVEL INDEXING WITH EXAMPLE.

- If we use a sparse index, the index itself may become too large for efficient processing.
- If an index is sufficiently small to be kept in main memory, the search time to find an entry is low.
- If index is so large that it must be kept on disk,
- a search for an entry requires several disk block reads.
- Binary search can be used on the index file to locate an entry, but the search still has a large cost.

To deal with this problem,

- we treat the index just as we would treat any other sequential file,
- Construct a sparse index on the primary index

- To locate a record, we first use binary search on the outer index to find the record for the largest search-key value less than or equal to the one that we desire.
- The pointer points to a block of the inner index.
- We scan this block until we find the record that has the largest search-key value less than or equal to the one that we desire.

The pointer in this record points to the block of the file that contains the record for which we are looking.

- we can repeat this process as many times as necessary.
- Indices with two or more levels are called multilevel indices.

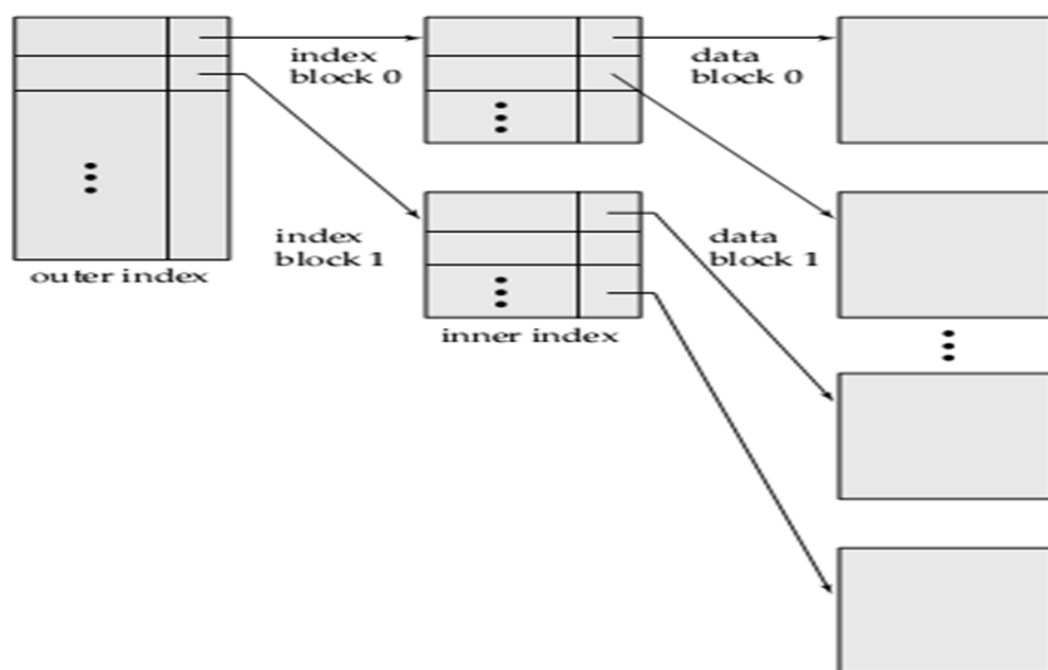


Figure Two-level sparse index.

Q12. WHAT IS HASHING? WHAT IS USE OF HASHING IN DATABASE?

Hashing -one way that we could get $O(1)$ access without wasting a lot of space.

File organizations based on the technique of hashing allow us to avoid accessing an index structure.

Q13. EXPLAIN HASH FILE ORGANIZATION.

- In a hash file organization,
- we obtain the address of the disk block containing a desired record directly by computing a function on the search-key value of the record.
- the term bucket denotes a unit of storage that can store one or more records.
- A bucket is typically a disk block, but could be chosen to be smaller or larger than a disk block.

Let K denote the set of all search-key values, and let B denote the set of all bucket addresses.

- A hash function h is a function from K to B .
- Let h denote a hash function.
- To insert a record with search key K_i ,
- we compute $h(K_i)$,

- which gives the address of the bucket for that record.
- Assume that there is space in the bucket to store the record.
- Then, the record is stored in that bucket.

To perform a lookup on a search-key value K_i , we simply compute $h(K_i)$, then

- search the bucket with that address.
- Suppose that two search keys, K_5 and K_7 , have the same hash value; that is, $h(K_5)=h(K_7)$.
- If we perform a look up on K_5 , the Bucket $h(K_5)$ contains records with search-key values K_5 and records with search-key values K_7 .
- Thus, we have to check the search-key value of every record in the bucket to verify that the record is one that we want.

Deletion is equally straightforward.

- If the search-key value of the record to be deleted is K_i ,
- we compute $h(K_i)$,
- then search the corresponding bucket for that record,
- and delete the record from the bucket.

Q14 .WRITE NOTE ON 'HASH FUNCTION'.

- 1) Worst possible hash function maps all search-key values to the same bucket.
- 2) Such a function is undesirable because all the records have to be kept in the same bucket.
- 3) A lookup has to examine every such record to find the one desired.
- 4) An ideal hash function distributes the stored keys uniformly across all the buckets,
- 5) so that every bucket has the same number of records.
- 6) Choose a hash function that assigns search-key values to buckets in such a way that the distribution has these qualities:
- 7) The distribution is uniform.
- 8) the hash function assigns each bucket the same number of search-key values from the set of all possible search-key values.
- 9) The distribution is random.
- 10) in the average case, each bucket will have nearly the same number of values assigned to it,
- 11) regardless of the actual distribution of search-key values.

Some hash functions:

- **Middle of square**

$H(x) :=$ return middle digits of x^2

- **Division**

$H(x) := \text{return } x \% k$

- **Multiplicative:**

$H(x) := \text{return the first few digits of the fractional part of } x * k$, where k is a fraction

Q15. HOW BUCKET OVERFLOW CAN BE HANDLED IN HASHING?

EXPLAIN TYPES OF HASHING – OPEN HASHING AND CLOSED HASHING.

- 1) If the bucket does not have enough space, a bucket overflow is said to occur.
- 2) Some buckets are assigned more records than others, so a bucket may overflow even when other buckets still have space.
- 3) This situation is called bucket skew.
- 4) The load factor of a hash table is the ratio of the number of keys in the table to the size of the hash table.
- 5) The higher the load factor, the slower the retrieval.
- 6) With open addressing, the load factor cannot exceed 1.
- 7) With chaining, the load factor often exceeds 1.

OPEN HASHING AND CLOSE HASHING:

- **Open Hashing (Separate Chaining):**

1. No bounds of hash table
2. In open hashing, keys are stored in linked lists attached to cells of a hash table.
3. Closed Hashing (Open Addressing):
4. Bound inside hash table

- **In closed hashing,**

all keys are stored in the hash table itself without the use of linked lists.

1. Rehashing
2. Linear Probing
- 3.

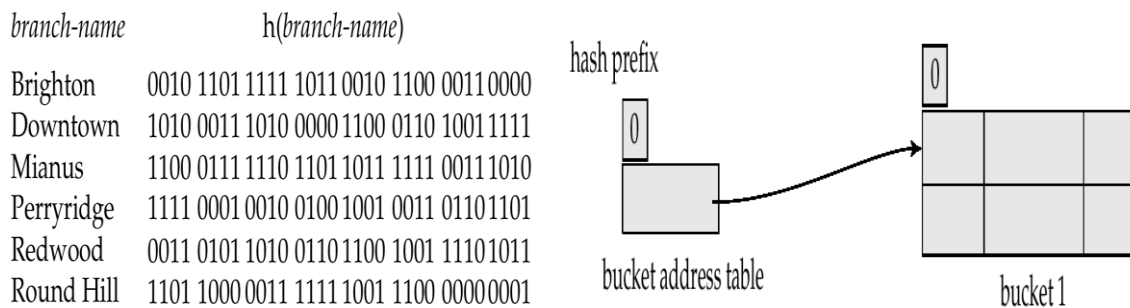
Q16. COMPARE STATIC HASHING AND DYNAMIC HASHING.

Q17. EXPLAIN EXTENDABLE HASHING WITH EXAMPLE.

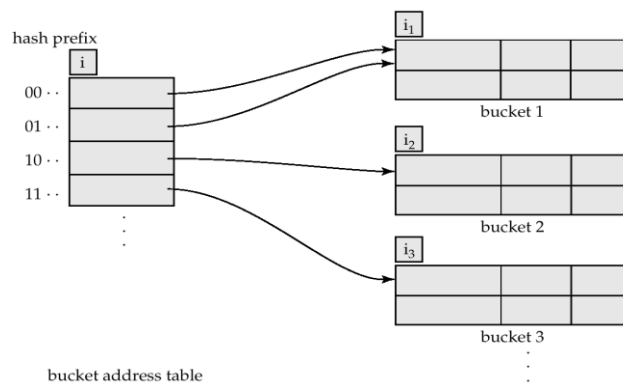
Extendable Hashing

- One form of dynamic hashing
- splits and coalesces buckets appropriately with the database size.

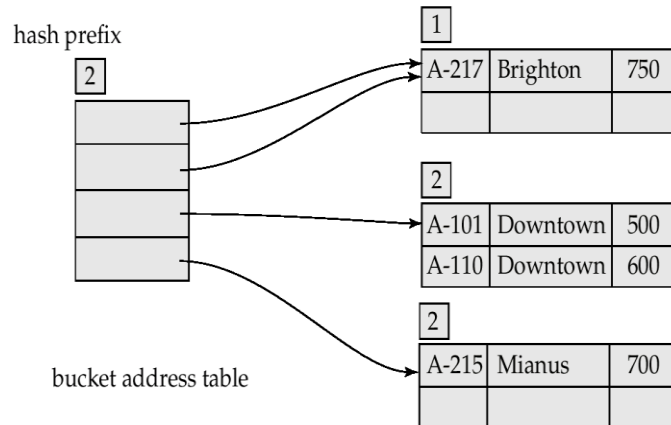
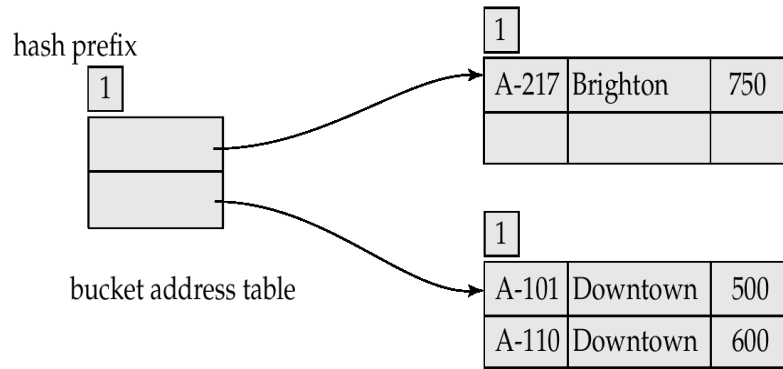
- i.e. buckets are added and deleted on demand
- ### Extendable Hashing
- One form of dynamic hashing
 - splits and coalesces buckets appropriately with the database size.
 - i.e. buckets are added and deleted on demand



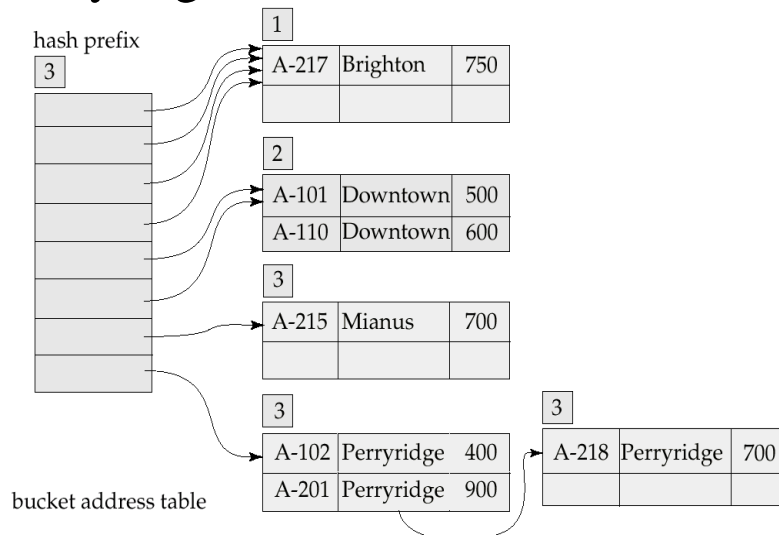
Initial Hash structure, bucket size = 2



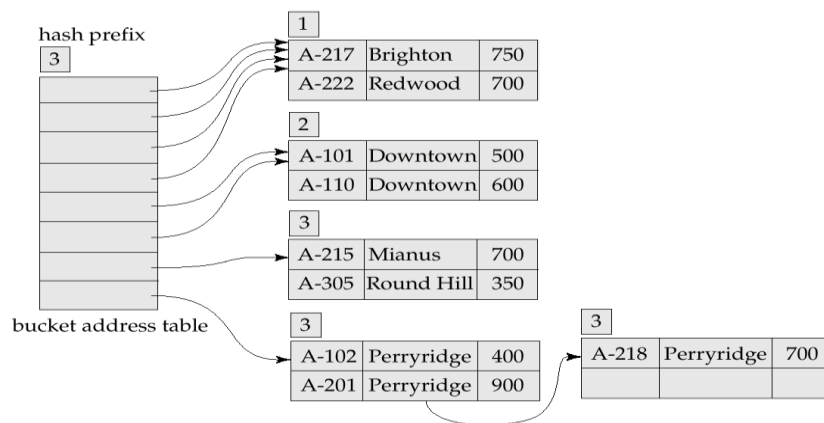
- Hash structure after insertion of one Brighton and two Downtown records:



- Hash structure after insertion of three Perryridge records.:



- Hash structure after insertion of Redwood and Round Hill records:



Q18.COMPARE ORDERED INDEXING WITH HASHING.

•Issues to consider:

- Cost of periodic re-organization
- Relative frequency of insertions and deletions

•Expected type of queries:

- Hashing is generally better at retrieving records having a specified value of the key.
- If range queries are common, ordered indices are to be preferred

19 . EXPLAIN BITMAP INDEX WITH EXAMPLE.

- A bitmap index is a special kind of database index that uses bitmaps.
- Bitmap indices have been considered to work well for low-cardinality columns,

- which have a modest number of distinct values to the number of records that contain the data.
- The extreme case of low cardinality is Boolean data.
- which has two values, True and False.

Bitmap indices use bit arrays(commonly called bitmaps)

- and answer queries by performing bitwise logical operations on these bitmaps.
- Bitmap indices have a significant space and performance advantage over other structures for query of such data.
- Their drawback is, they are less efficient than the traditional B-tree indices for columns whose data is frequently updated:
- consequently, they are more often employed in read-only systems that are specialized for fast query
- e.g., data warehouses, and generally unsuitable for online transaction processing applications.

Bitmap indices are also useful for moderate or even high-cardinality data (e.g., unique-valued data)

- which is accessed in a read-only manner, and queries access multiple bitmap-indexed columns using the AND, OR orXOR operators extensively.

A better representation of a bitmap index

Identifier	Class	Row ID
1	'SY'	R1
2	'TY'	R2
3	'TY'	R3
4	'FY'	R4
5	'SY'	R5

the a bitmap index on the Class column would (conceptually) look like this:

•Class	R1	R2	R3	R4	R5
•FY	0	0	0	1	0
•SY	1	0	0	0	1
•TY	0	1	1	0	0

Bitmap indices are used when the number of distinct values in a column is relatively low

- So with this index in place a query like
- Select * from table1 where class \neq 'TY'.

Q20. EXPLAIN B-TREE INDEXING WITH EXAMPLE.

A B-tree of order m is an m -way (multi-way) tree (i.e. a tree where each node may have up to m children) in which:

1. The number of keys in each non-leaf node is one less than the number of its children and

- These keys partition the keys in the children in the fashion of a search tree

2. All leaves are at the same level

3. All non-leaf nodes except the root have at least $m / 2$ children

4. The root is either a leaf node, or it has from two to m children.

5. A leaf node contains no more than $(m-1)$ keys

- The number m should always be odd.

	Keys		Children	
	Min	Max	Min	Max
Root	0	$m-1$	0	m
Non-root				
Non-leaf	$m/2$	$m-1$	$m/2$	m
Leaf	$m/2$	$m-1$	0	0

EX.:

(PAGE NO 95 FROM NOTES.)

Q21 EXPLAIN B + TREE INDEXING WITH EXAMPLE.

- B+ tree is a balanced tree in which every path from the root of the tree to a leaf is of the same length,
- and each non-leaf node of the tree has between $\lceil m/2 \rceil$ and $\lceil m \rceil$ children,
- where m is fixed for a particular tree.
- It contains index pages and data pages.
- A B+-tree of order m is an m -way (multi-way) tree (i.e. a tree where each node may have up to m children) in which:
 1. The number of keys in each non-leaf node is one less than the number of its children and
- These keys partition the keys in the children in the fashion of a search tree
 2. All leaves are at the same level
 3. All non-leaf nodes except the root have at least $m / 2$ children
 4. The root is either a leaf node, or it has from two to m children
 5. A leaf node contains no more than $(m-1)$ keys
- It contains index pages and data pages.
- Actual Data appears at leaf nodes.
- The number m should always be odd.

	Keys		Children
	Min	Max	Min
Max Root	0	m-1	0
m Non-root			
Non-leaf	m/2	m-1	m/2
m Leaf	m/2	m-1	0
0			

(EXAMPLE ON PAGE NO.118 FROM NOTES.)