

File Organization

FILE ORGANIZATION

is a method of arranging data on secondary storage devices and addressing them such that it facilitates storage and read/write operations of data or information requested by the user.

A **record** is a collection of logically related fields or data items. Each data item is formed of one or more bytes. Record usually describes entities and their attributes.

File is a collection of related sequence of records.

File Organization

- The physical arrangement of data in a file into records and pages on the disk
- File organization determines the set of access methods for
 - Storing and retrieving records from a file
- Therefore, 'file organization' synonymous with 'access method'
- We study three types of file organization
 - Unordered or Heap files
 - Ordered or sequential files
 - Hash files
- We examine each of them in terms of the operations we perform on the database
 - Insert a new record
 - Search for a record (or update a record)
 - Delete a record

File Indexing

Need for indexing:

- Database - is an organized collection of logically related data.
- The operations (read, modify, update, and delete) access data from the database.
- DBMS must first transfer the data temporarily from secondary memory to a buffer in main memory.

Need for indexing:

- Data is transferred between disk and main memory into units called blocks.
- The transferring of data from secondary memory to main memory is a very slow operation.

Index :

- An index is a data structure, which is maintained to determine the location of records in a file.

The Index Table :

- The index table contains a search key and a pointer.
- Search key - an attribute or set of attributes that is used to look up the records in a file.
- Pointer - contains the address of where the record is stored in the memory.

- Each index entry corresponds to a record on the secondary storage file.

To retrieve a record from the disk-

- First the index is searched for the address of the record in the index table
- Then the record is read from that address.
- It is just like searching for a book in a library catalogue.

- An index a data structure that is added to a file to provide faster access to the data.
- It reduces the number of blocks that the DBMS has to check.

Example:

- If there is a table similar to this:

```
CREATE TABLE test1  
( id integer,  
content varchar2(30) );
```

- The application requires a lot of queries of the form
SELECT content FROM test1 WHERE id =
constant;
- Ordinarily, the system would have to scan the entire test1 table row by row to find all matching entries.

Creating Index :

- SYNTAX:

CREATE INDEX <index-name>

ON <table-name>

(<column-name> [ASC|DESC],


<column-name> [ASC|DESC]...);

Create the index on the id column:

```
CREATE INDEX testid_index ON test1  
(id ASC);
```

INDEX
TABLE

1
2
3
4



ID	CONTENT
1	ABC
2	DEF
3	PQR
4	XYZ

FILE ORGANIZATION

A file can be of 2 formats:

- Fixed length records
- Variable length records

FIXED LENGTH RECORDS

- All records on the page are of the same slot length i.e. every record in the file has exactly the same size(in bytes).
- Record slots are uniform and records are arranged consecutively within a page.

type deposit=**record**

 account_no:char(10);

 branch_name:char(22);

 balance:numeral(12,2);

end

Assume char occupies 1 byte and numeral stores 8 bytes
.So first 40 bytes are stored for first record and next 40
bytes for the second record and so on.

FIXED LENGTH RECORDS

	ACCOUNT_NO	BRANCH_NAME	BALANCE
Record 0->	A-101	AB	100
Record 1->	A-102	CD	400
Record 2->	A-567	BC	200
Record 3->	A-92	AT	900
Record 4->	A-45	TC	300

FIXED LENGTH RECORDS

- **Insertion-** insertion occurs at the first available slot(or empty spaces).
- **Deletion-** after a record is deleted an empty space is created. Now there can be various ways to fill this space:
 - (a) shift all the records just below space by one place upwards. and all the empty slots appear together at the end (requires a large no. of shifting)

FIXED LENGTH RECORDS

- (b) shift the last record in empty slot, instead of disturbing large no. of records.
- (c) here deletion of records is handled by using an array of bits called **file header** at the beginning of the file. When the first record is deleted, the file header stores its address. Now this empty slot of record is used to store the address (also called pointers)of next empty slot.

FIXED LENGTH RECORDS

	ACCOUNT_NO	BRANCH_NAME	BALANCE
Record 0->	A-101	AB	100
Record 2->	A-567	BC	200
Record 3->	A-92	AT	900
Record 4->	A-45	TC	300
	ACCOUNT_NO	BRANCH_NAME	BALANCE
	A-101	AB	100
	A-567	BC	200
	A-92	AT	900
	A-45	TC	300

FIXED LENGTH RECORDS

(b)

	ACCOUNT_NO	BRANCH_NAME	BALANCE
Record 0->	A-101	AB	100
Record 2->	A-567	BC	200
Record 3->	A-92	AT	900
Record 4->	A-45	TC	300

ACCOUNT_NO	BRANCH_NAME	BALANCE
A-101	AB	100
A-45	TC	300
A-567	BC	200
A-92	AT	900

FIXED LENGTH RECORDS

FILE HEADER	ACCOUNT_NO	BRANCH_NAM E	BALANCE
	Stores the address	of first empty slot	
	A-101	AB	100
	A-567	BC	200
	A-45	TC	300



FIXED LENGTH RECORDS

These stored empty slot addresses form a link list called as FREE LIST. Now whenever a new record is to be inserted, it is inserted in the first available empty slot pointed by the header. And value of file header is changed to address of next available empty slot. In case of unavailability of empty slot, the new record is added at the end of the file.

ADVANTAGE OF FIXED LENGTH RECORDS

Because the space made available by a deleted record is exactly the space needed to insert a new record, insertion and deletion for files are simple to implement.

VARIABLE LENGTH RECORDS

- All the records on the page are not of the same length but here different records in the file have different sizes.

```
type account_list=record
```

```
    branch_name:char(22);
```

```
account_info:array[1..infinity]of record
```

```
    account_no:char(10);
```

```
    balance:numeral(12,2);
```

```
end
```

```
end
```

Variable length records

2 ways to implement variable-length records

- Byte string representation
- Fixed length representation

Variable length records:

Byte string representation

- a special symbol called end of record(|_|) is added at the end of each record.
- each record is stored as a string of consecutive bytes.
- Disadvantages:
 - a)it is very difficult to reuse empty slot (occupied formally by deleted record).
 - b)a large no. of small fragments of disk storage are wasted.
- Therefore a modified form of byte-string representation called the **slotted-page structure**, is commonly used for organizing records within a single block.

Byte string representation

Consider an example where a person can have more than one accounts in a bank.

BRANCH_ NAME	ACCOUNT _NO	BALANCE	ACCOUNT _NO	BALANCE	
AB	A-101	500	A-147	200	_I_
CD	A-92	100	_I_		
EF	A-47	900	_I_		
GH	A-52	500	A-66	1000	_I_

Slotted-page structure

- Used for organizing records within a single block .
- There is a header at the beginning of each block containing the following information:
 - (i)The no. of record entries .
 - (ii)The end of free space in the block
 - (iii)An array whose entries contain the loc and size of each record.



Variable length records:

Fixed length representation

- One or more fixed-length records are used to represent variable-length record.
 - 2 methods namely (a) reserved space (b) list representation
 - Reserved Space:
 - (i) a fixed length record of the size equal to that of maximum record length in a file is used.
 - (ii) unused space of the records shorter than the maximum size is filled with a special null or end-of –record symbol.
- Method is useful when most records have length close to the maximum record length otherwise a significant amount of space is wasted.

Fixed length representation: Reserved Space

BRANCH_ AME	ACCOUNT_ NO	BALANCE	ACCOUNT_ NO	BALANCE
AB	A-101	500	A-147	200
CD	A-92	100	_I_	_I_
EF	A-47	900	A-65	400
GH	A-52	500	A-66	1000

Variable length records : Fixed length representation

- List representation(also called link list)
 - (i) here a list of fixed-length records is chained together by pointers. Method is similar to file header address concept except in case of file header method pointers are used to chain together only deleted records, whereas here pointers of all records pertaining to the same branch_name are chained together.

Fixed length representation:

List representation

	BRANCH_N AME	ACCOUNT_ NO	BALANCE
FILE HEADER	AB	A-101	500
	CD	A-92	100
	EF	A-47	900
	GH	A-52	500
		A-147	200
		A-65	400
		A-66	1000



Variable length records:Fixed length representation

Problem with this representation is

when a new record is to be inserted, an empty slot of just right size is needed (neither a smaller size will do nor the big slot would solve the purpose as it would lead to wastage of space).So it becomes imp to allocate a right size of slot for insertion of record and move records to fill the space created by deletion of records to ensure that all the free space in the file is contiguous.

Variable length records: Fixed length representation

To deal with this problem, we allow two kinds of blocks in our file:

1. Anchor block: which contain the first record of a chain
2. Overflow block: This contains records other than those that are the first record of a chain.

Thus all the records *within a block* have the same length, even though not all records in the file have the same length.

Anchor block-Overflow block

■ Anchor block:

BRANCH_NAME	ACCOUNT_NO	BALANCE
AB	A-101	500
CD	A-92	100
EF	A-47	900
GH	A-52	500

OVERFLOW BLOCK

A-147	200
A-65	400
A-66	1000



Creating Index :

- SYNTAX:

CREATE INDEX <index-name>

ON <table-name>

(<column-name> [ASC|DESC],


<column-name> [ASC|DESC]...);

Create the index on the id column:

```
CREATE INDEX testid_index ON test1  
(id ASC);
```

INDEX
TABLE

1
2
3
4



ID	CONTENT
1	ABC
2	DEF
3	PQR
4	XYZ

Types of indexing techniques :

- **Ordered indexing**
- **Hashed indexing**
- Ordered index – which is used to access data sorted by order of values.
Binary search can be used to access records.
- Hash index - used to access data that is distributed uniformly across a range using a hashed function.

Types of Ordered indexing

Dense indexing :

- An index record or entry appears for every search-key value in the table.
- ▶ The index record contains the search key and a pointer to the first data record with that search-key value.

Example:

Taking year as the search key :

		ROLL NO	NAME	YEAR	MARKS	PERCE- NTAGE
1	→	6545	ABC	1	894	76.8
2	→	7645	BCD	2	900	78.4
3		7456	PQR	2	1055	82.9
	↘	8734	XYZ	3	867	75.6

Insertion :

- if the search key value does not appear in the index, the new record is inserted to an appropriate position
- if the index record stores a pointer to only the first record with the same search-key value, the record being inserted is placed right after the other records with the same search-key values

Deletion :

- if the deleted record was the only record with its unique search-key value, then it is simply deleted
- If the index record stores a pointer to only the first record with the same search-key value, and if the deleted record was the first record, update the index record to point to the next record.

Inserting :

Inserting

- a record with year 2 (existing search-key value)
- a record with year 4 (new search-key value)

		ROLL NO	NAME	YEAR	MARKS	PERCE- NTAGE
		6545	ABC	1	894	76.8
1		7645	BCD	2	900	78.4
2		7456	PQR	2	1055	82.9
3		7477	RST	2	1088	88.8
4		8734	XYZ	3	867	75.6
		8899	UVW	4	894	76.8

Dense Index



Sparse Indexing :

- An index record that appears for only some of the values in the file.
- The records are divided into blocks.
- The index is assumed to be storing an entry of each block of the file.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.

Example :



Insertion :

- if no new block is created, no change is made to the index.
- if a new block is created, the first search-key value in the new block is added to the index.

Deletion :

- if the deleted record was the only record with its search key, the corresponding index record is replaced with an index record for the next search-key value
- if the record being deleted is the first record of the block, the next record with the same search-key value is updated as the reference instead

Which one is better?

Dense or sparse?

- Records can be located faster using dense index than sparse index.
 - But, sparse indexes require less space and they impose less maintenance overhead for insertions and deletions.
 - So choosing between dense and sparse indexes , it's a trade off between time and space overhead.
-

Primary Index :

- It is an ordered index with the primary key field as the search-key.
- It can be dense or sparse.
- There can only be one primary index for a file.

Secondary Index :

- It provides a secondary means of accessing a file where a primary access already exists.
- The search-key is a candidate key & has a unique value.
- More than one secondary indices can exist for the same file

Clustered Index :

- If the records are physically ordered on a nonkey field – which does not have a distinct value for each record-that field is called the **clustering field** and the index with the clustering field as the search-key is known as the **clustered index**.
- The clustering index has an entry for each distinct value of the clustering field.
- There can be only one clustered index per table.
- Blocks of fixed size are reserved for each value of the clustering field to avoid physical reordering during insertion & deletion.

Clustered Index(finally !)

A clustered index is a special type of index that reorders the way records in the table are physically stored. Therefore a table can have only one clustered index.

Clustering alters the data block into a certain distinct order to match the index, hence it is also an operation on the data storage blocks as well as on the index.

Example of Clustered Index

An address book ordered by first name resembles a clustered index in its structure and purpose.

A dictionary.

- Clustered indexes are good for range searches.

Non clustered Index :

- The logical order of the index does not match the physical stored order of the rows on disk.
- A file can have multiple non-clustered indexes.
- Non-clustered indexes are good for random searches.

Example of Non-Clustered Index

An index on the backside of the book where you see only the pages on which the words of the index appear.

The order of the words in the book is not according to the order of the words in the index.

Non-Clustered Index

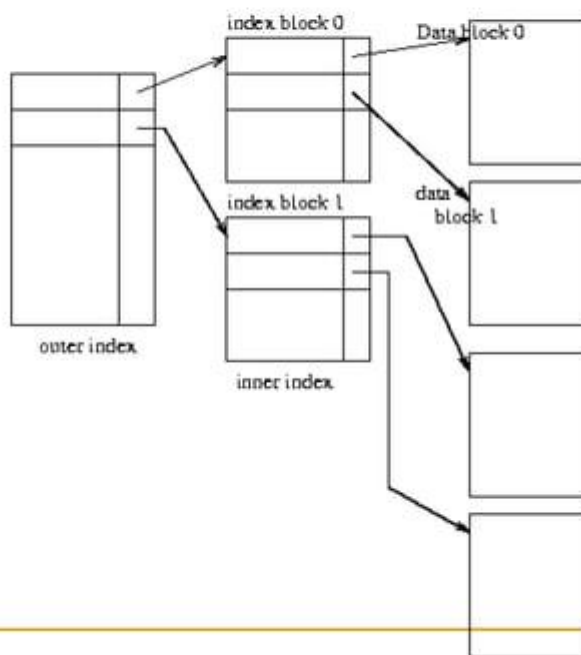
An non-clustered index is structured in a way that doesn't correspond to the order of the actual data records.

For this reason, it will perform worse than clustered indexes on ranged searches where the result set is large, since each result could cost an additional I/O-operation to get the actual data record.

Multilevel Index :

- If an index may be too large for efficient processing we use multi-level indexing.
- The outer index is a sparse index of the primary index whereas the inner index is the primary index.
- In the same way any no. of levels can be made depending on the size of the index.
- Generally all index levels are physically ordered which creates problems in insertion & deletion.
- Therefore , dynamic multilevel indexes (B-trees and B⁺-trees) are used.

Multilevel Index :



HASHING

INTRODUCTION

- Hashing is a method for determining physical location of record.
 - In it the record key is processed mathematically and another no is computed which represents location of record (hashing function).
 - Files which use hashing techniques are called hash files.
-

Example...

empid	ename	salary
4555	abc	5000

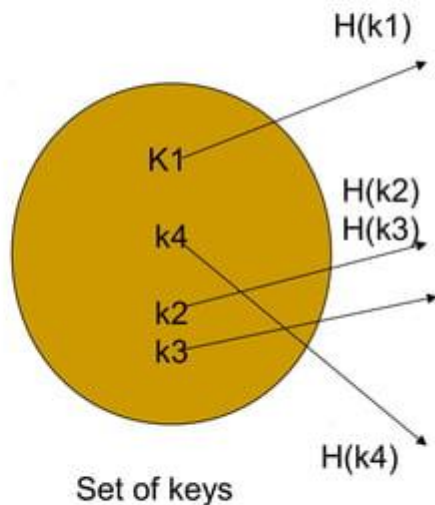
Here empid is primary key. let there be 1000 employees. hash Function is to divide the key by prime number close to 1000 and Use remainder as storage location.

Prime number=997
Remainder =567

Hence the employee record is stored at memory location 567.

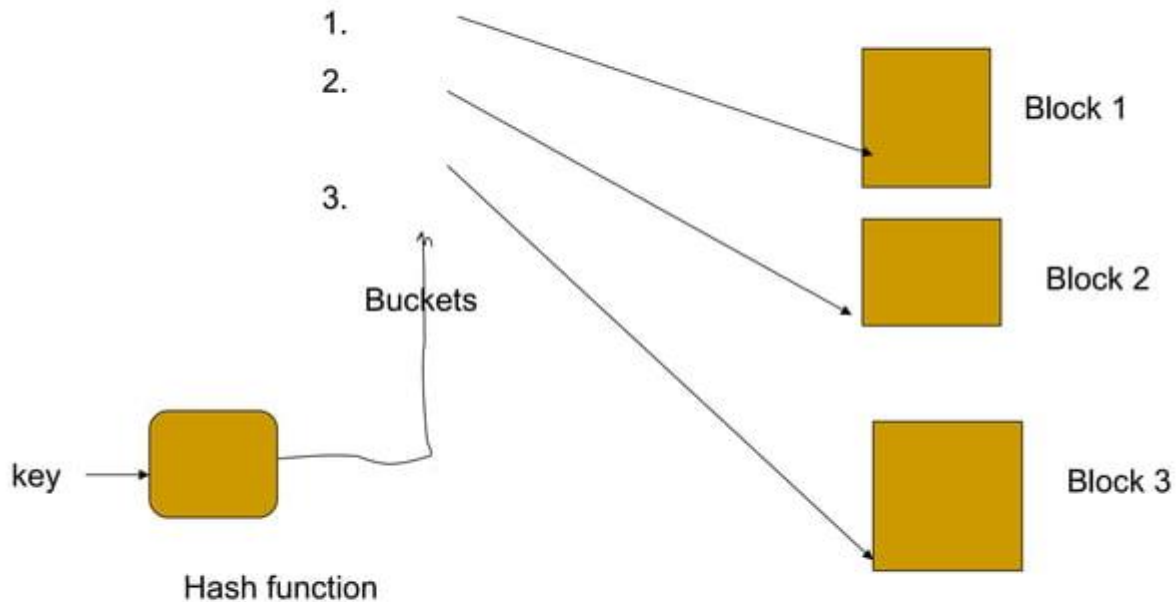
Types of hashing

- **Internal Hashing**-In case of internal files hashing is implemented as a hash table .Hash table is a data structure in which location of data item is determined by hash function. If hashed value of two keys result same then it is called 'collision'.



External hashing

- Hashing for disk files is called external hashing.
- Target address space is made of buckets.
- Each bucket has multiple records (block address).
- Hash functions map record key to bucket number.
- A table maintained in file header converts bucket number to Corresponding block address.
- Hash function is used to locate records for access, insertion as well as deletion.
- This method of buckets prevents collision.
- This technique of hashing is called static hashing as fixed number of buckets are allocated in beginning.

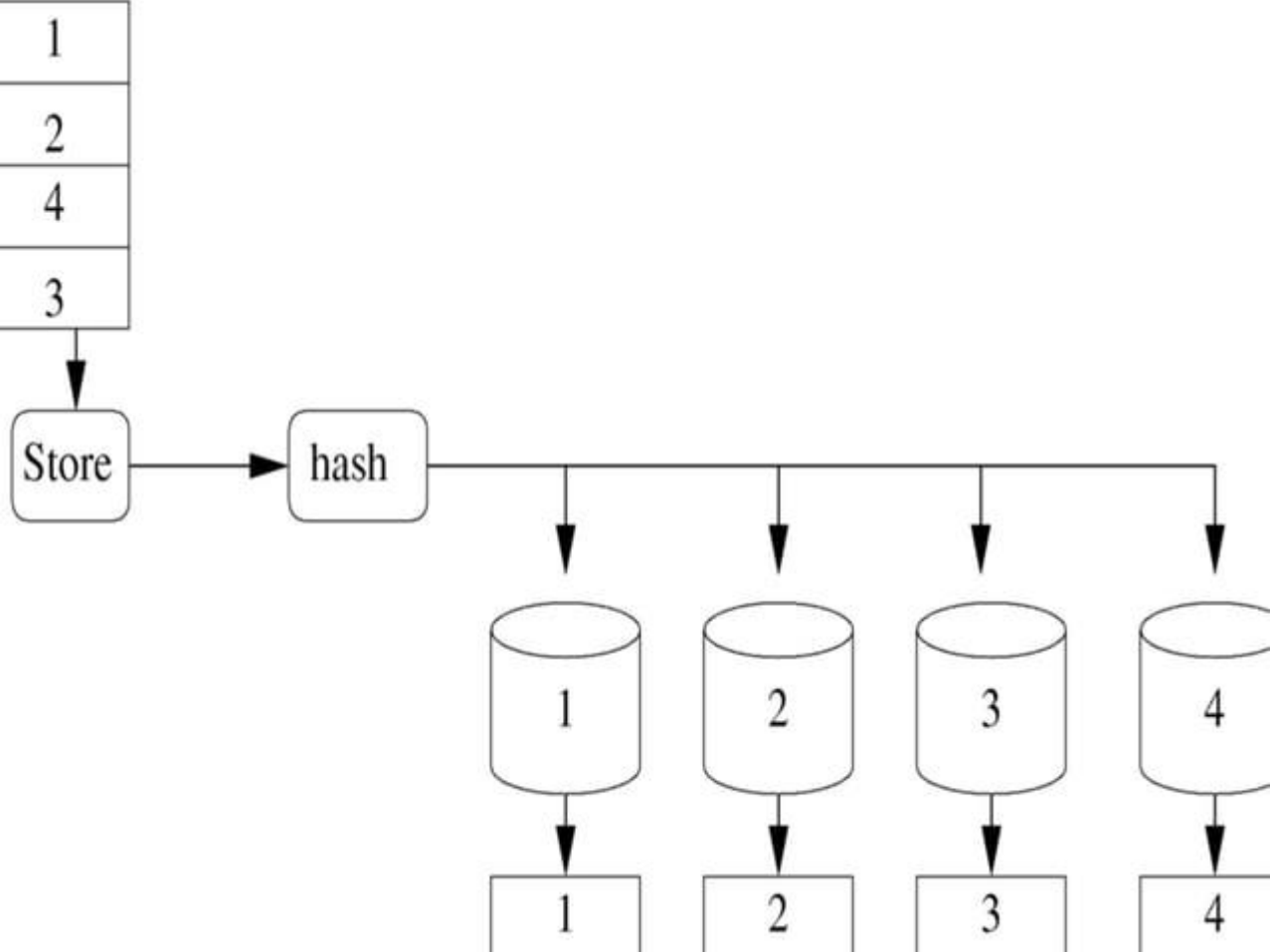


```
CREATE TABLE test ( id INTEGER PRIMARY KEY ,  
                      name varchar(100) );
```

In table test, we have decided to store 4 rows...

```
insert into test values (1,'Gordon'); insert into test values (2,'Jim');  
insert into test values (4,'Andrew'); insert into test values (3,'John');
```

The algorithm splits the places which the rows are to be stored into areas. These areas are called buckets. If a row's primary key matches the requirements to be stored in that bucket, then that is where it will be stored. The algorithm to decide which bucket to use is called the hash function. For our example we will have a nice simple hash function, where the bucket number equals the primary key.



Problem of overflow....

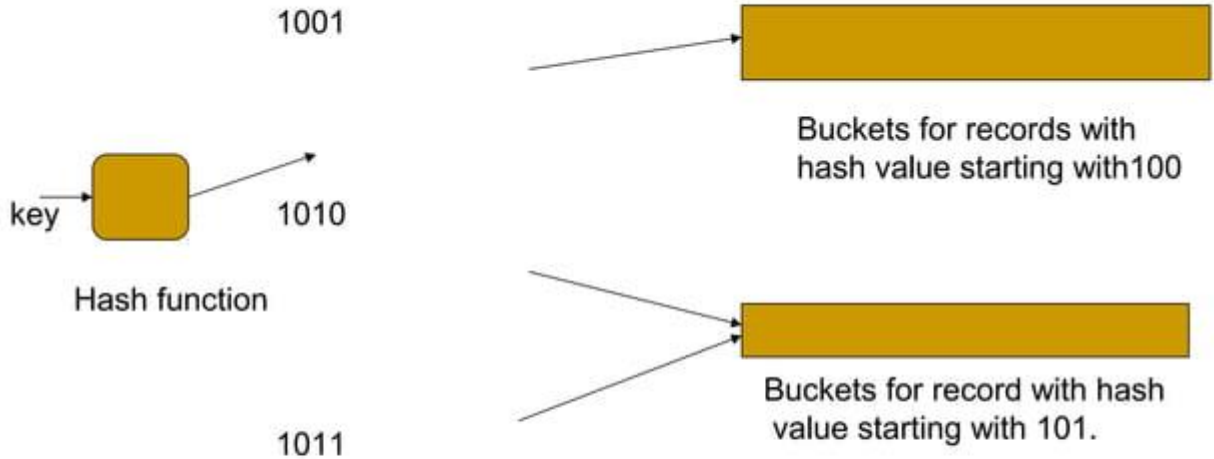
- Overflow occurs when bucket is filled to its capacity and new record has to be inserted.
 - We can have pointer in bucket which points to linked list of overflowed records of buckets.
-

Deficiencies of Static Hashing

- In static hashing, function h maps search-key values to a fixed set of B of bucket addresses.
 - Databases grow with time. If initial number of buckets is too small, performance will degrade due to too much overflows.
 -
 - If database shrinks, again space will be wasted.
 -
 - These problems can be avoided by using techniques that allow the number of buckets to be modified dynamically.
-

Dynamic Hashing

- Good for database that grows and shrinks in size. Allows the hash function to be modified dynamically
- ***Extendable hashing*** – one form of dynamic hashing
- A directory –an array of 2^d bucket address is maintained
- d is called global depth of directory.
- The result of hash function is represented in binary form and the first ' i ' bits of hash value are used to determine directory entry and value at it determines bucket in which record is to be stored.
- We can increase the number of ' d ' to accommodate more buckets.



Here global depth is 4 and let first three bits of hash value represent array Index.

example.....

empid	ename
6	abc
5	xyz

Hash function= $k*2$

Hash value for first record is 1100

Hash value for second record is 1010

1101



Buckets for records having hash value Starting with 110

1011



Bucket for records having hash value Starting with 101

directory