Module4

PART-2

Hashing –Statuc hashing

- One disadvantage of sequential file organization is that we must access an index structure to locate data.
- File organizations based on the technique of hashing allow us to avoid accessing an index structure.
- In our description of hashing, we shall use the term bucket to denote 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 Ki , we compute h(Ki), which gives the address of the bucket for that record.
- To perform a lookup on a search-key value Ki , we simply compute h(Ki), then search the bucket with that address.

- Suppose that two search keys, K5 and K7, have the same hash value; that is, h(K5) = h(K7).
- If we perform a lookup on K5, the bucket h(K5) contains records with search-key values K5 and records with search-key values K7.
- 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 Ki , we compute h(Ki), then search the corresponding bucket for that record, and delete the record from the bucket.

- Hashing can be used for two different purposes. 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.
- In a **hash index organization** we organize the search keys, with their associated pointers, into a hash file structure.

Hash Functions

- An ideal hash function distributes the stored keys uniformly across all the buckets, so that every bucket has the same number of records.
- Since we do not know at design time precisely which search-key values will be stored in the file, we want to choose a hash function that assigns search-key values to buckets in such a way that the distribution has these qualities:
  - The distribution is uniform.
  - The distribution is random

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Figure 11.23 Hash organization of *instructor* file, with *dept_name* as the key

- Hash functions require careful design. A bad hash function may result in lookup taking time proportional to the number of search keys in the file. A well designed function gives an average-case lookup time that is a (small) constant, independent of the number of search keys in the file

Handling of Bucket Overflows

- If the bucket does not have enough space, a bucket overflow is said to occur. Bucket overflow can occur for several reasons:
- **Insufficient buckets**.
- The number of buckets, which we denote nB , must be chosen such that nB >nr /fr , where nr denotes the total number of records that will be stored and fr denotes the number of records that will fit in a bucket. This designation, of course, assumes that the total number of records is known when the hash function is chosen.

- **Skew**. Some buckets are assigned more records than are others, so a bucket may overflow even when other buckets still have space.
- This situation is called bucket skew.
- Skew can occur for two reasons:
  1. Multiple records may have the same search key.
  2. The chosen hash function may result in nonuniform distribution of search keys

- So that the probability of bucket overflow is reduced, the number of buckets is chosen to be (nr /fr) * (1 + d), where d is a fudge factor, typically around 0.2. Some space is wasted: About 20 percent of the space in the buckets will be empty. But the benefit is that the probability of overflow is reduced.

- Despite allocation of a few more buckets than required, bucket overflow can still occur.
- We handle bucket overflow by using **overflow buckets**. If a record must be inserted into a bucket b, and b is already full, the system provides an overflow bucket for b, and inserts the record into the overflow bucket.
- If the overflow bucket is also full, the system provides another overflow bucket, and so on. All the overflow buckets of a given bucket are chained together in a linked list,

- Overflow handling using such a linked list is called **overflow chaining.**

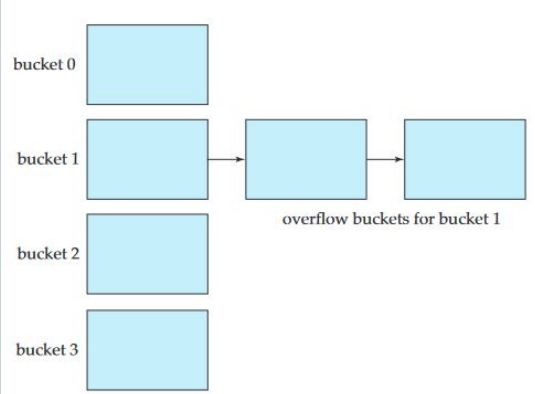

Figure 11.24 Overflow chaining in a hash structure.

- We must change the lookup algorithm slightly to handle overflow chaining.
  As before, the system uses the hash function on the search key to identify a bucket b. The system must examine all the records in bucket b to see whether they match the search key, as before. In addition, if bucket b has overflow buckets, the system must examine the records in all the overflow buckets also.
- The form of hash structure is closed hashing.

- Under an alternative approach, called open hashing, the set of buckets is fixed, and there are no overflow chains. Instead, if a bucket is full, the system inserts records in some other bucket in the initial set of buckets B.
  One policy is to use the next bucket (in cyclic order) that has space; this policy is called **linear probing**.

Dynamic Hashing

- the need to fix the set B of bucket addresses presents a serious problem with the static hashing technique.
- Most databases grow larger over time. If we are to use static hashing for such a database, we have three classes of options:
- 1. Choose a hash function based on the current file size
- 2. Choose a hash function based on the anticipated size of the file at some point in the future
- 3. Periodically reorganize the hash structure in response to file growth.

- Several dynamic hashing techniques allow the hash function to be modified dynamically to accommodate the growth or shrinkage of the database.
- Extendable hashing(dynamic hashing) copes with changes in database size by splitting and combining buckets as the database grows and shrinks. As a result, space efficiency is retained.
- With extendable hashing, we choose a hash function h with the desirable properties of uniformity and randomness. However, this hash function generates values over a relatively large range—namely, b-bit binary integers. A typical value for b is 32.