

Unit - 6

Trends in Database Technology

Physical Storage Media

- ◆ Although a database system provides a high-level view of data, ultimately data have to be stored as bits on one or more storage devices.
- ◆ A vast majority of databases today store data on magnetic disk and fetch data into main memory for processing, or copy data onto tapes and other backup devices for archival storage.
- ◆ The physical characteristics of storage devices play a major role in the way data are stored, in particular because access to a random piece of data on disk is much slower than memory access: Disk access takes tens of milliseconds, whereas memory access takes a tenth of a microsecond.

Physical Storage Media

- ◆ Several types of data storage exist in most computer systems. These storage media are classified by the speed with which data can be accessed, by the cost per unit of data to buy the medium, and by the medium's reliability. Among the media typically available are these are cache, main memory, magnetic disk, optical disk, tape.

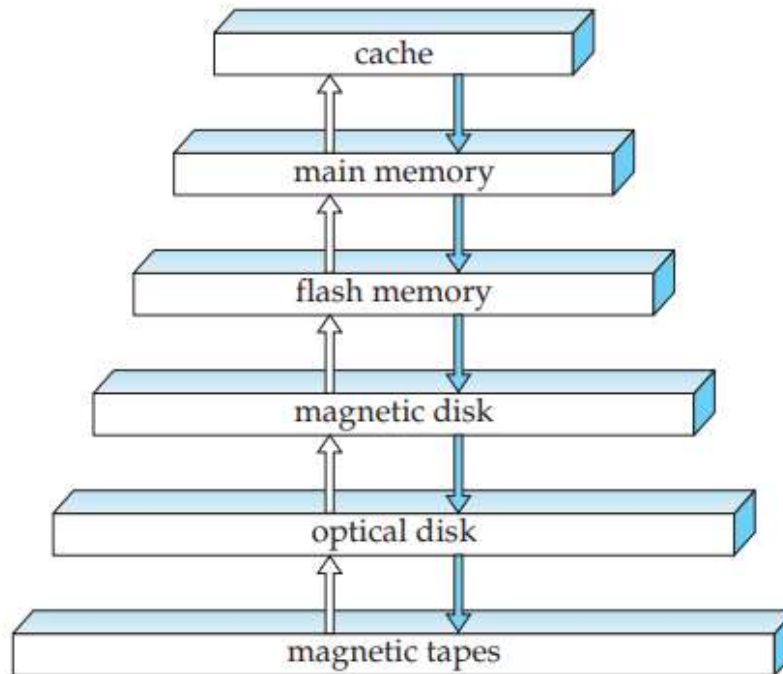
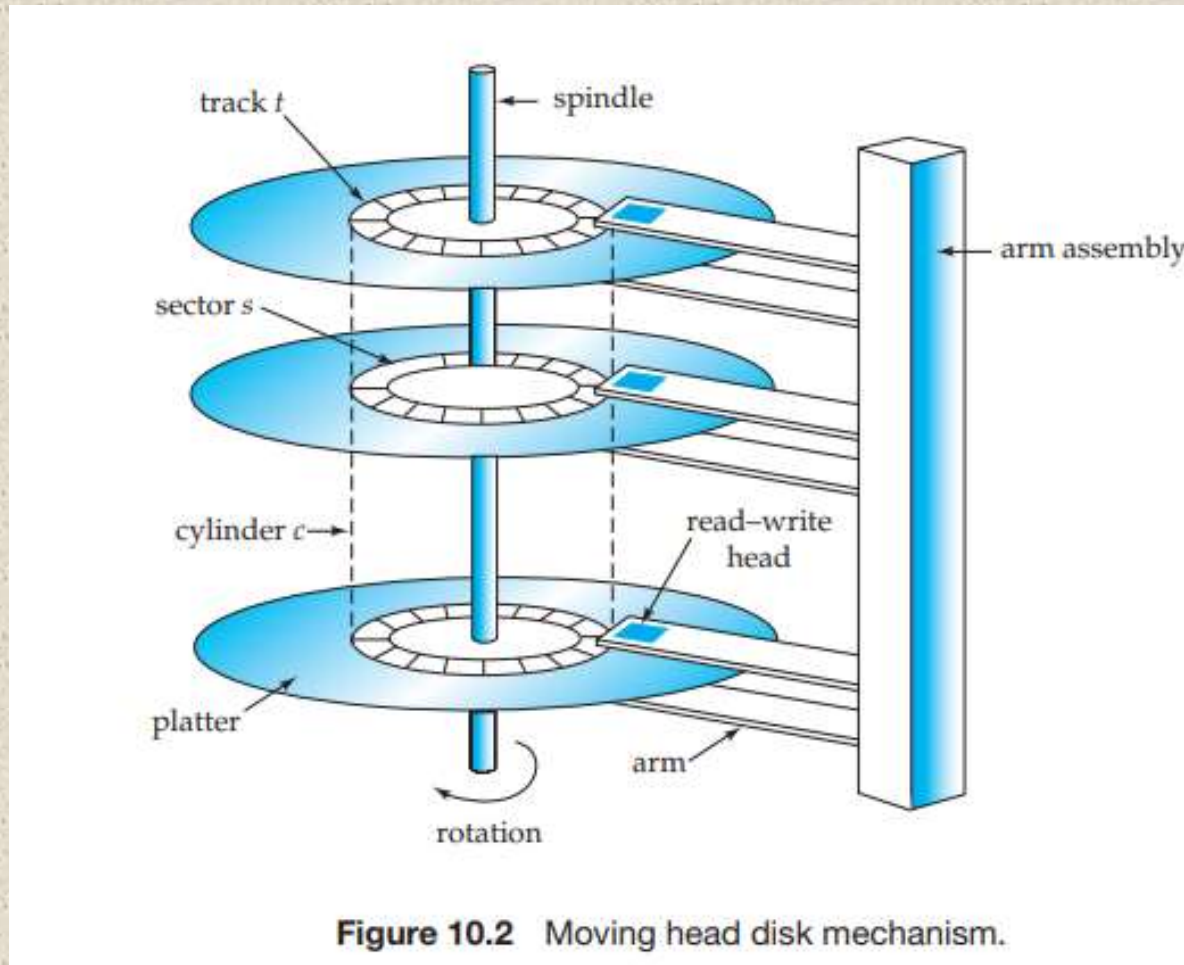


Figure 10.1 Storage device hierarchy.

Physical Storage Media

- ◆ Several types of data storage exist in most computer systems. These storage media are classified by the speed with which data can be accessed, by the cost per unit of data to buy the medium, and by the medium's reliability.
- ◆ Among the media typically available are these are cache, main memory, magnetic disk, optical disk, tape.

Physical Storage Media



Physical Storage Media

- ◆ **Performance Measures of Disk**
- ◆ **Access time** is the time from when a read or write request is issued to when data transfer begins. To access (that is, to read or write) data on a given sector of a disk, the arm first must move so that it is positioned over the correct track, and then must wait for the sector to appear under it as the disk rotates. The time for repositioning the arm is called the seek time, and it increases with the distance that the arm must move. Typical seek times range from 2 to 30 milliseconds, depending on how far the track is from the initial arm position. Smaller disks tend to have lower seek times since the head has to travel a smaller distance.
- ◆ The **average seek time** is the average of the seek times, measured over a sequence of (uniformly distributed) random requests. I

Physical Storage Media

- ◆ **Performance Measures of Disk**
- ◆ Once the head has reached the desired track, the time spent waiting for the sector to be accessed to appear under the head is called the **rotational latency time**.
- ◆ The access time is then the sum of the seek time and the latency, and ranges from 8 to 20 milliseconds. Once the first sector of the data to be accessed has come under the head, data transfer begins.
- ◆ The data-transfer rate is the rate at which data can be retrieved from or stored to the disk.
- ◆ The final commonly used measure of a disk is the **mean time to failure (MTTF)**, which is a measure of the reliability of the disk. The mean time to failure of a disk (or of any other system) is the amount of time that, on average, we can expect the system to run continuously without any failure

Physical Storage Media

- ◆ **Optimization of Disk-Block Access**
- ◆ **Buffering.** Blocks that are read from disk are stored temporarily in an in-memory buffer, to satisfy future requests. Buffering is done by both the operating system and the database system.
- ◆ **Read-ahead.** When a disk block is accessed, consecutive blocks from the same track are read into an in-memory buffer even if there is no pending request for the blocks. In the case of sequential access, such read-ahead ensures that many blocks are already in memory when they are requested, and minimizes the time wasted in disk seeks and rotational latency per block read. Operating systems also routinely perform read-ahead for consecutive blocks of an operating system file.

Physical Storage Media

- ◆ **Optimization of Disk-Block Access**
- ◆ **Scheduling.** If several blocks from a cylinder need to be transferred from disk to main memory, we may be able to save access time by requesting the blocks in the order in which they will pass under the heads. If the desired blocks are on different cylinders, it is advantageous to request the blocks in an order that minimizes disk-arm movement. Disk-arm-scheduling algorithms attempt to order accesses to tracks in a fashion that increases the number of accesses that can be processed.
- ◆ **File organization.** To reduce block-access time, we can organize blocks on disk in a way that corresponds closely to the way we expect data to be accessed. For example, if we expect a file to be accessed sequentially, then we should ideally keep all the blocks of the file sequentially on adjacent cylinders.

RAID

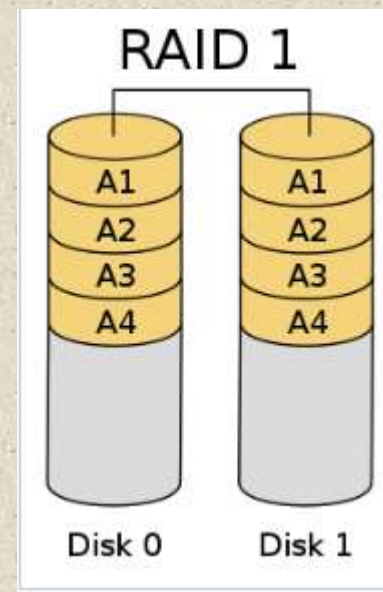
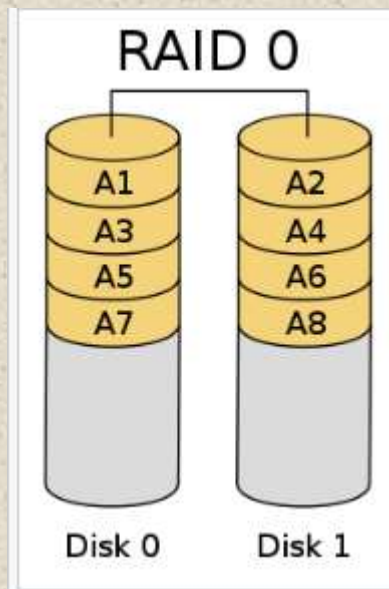
- ◆ The data-storage requirements of some applications (in particular Web, database, and multimedia applications) have been growing so fast that a large number of disks are needed to store their data, even though disk-drive capacities have been growing very fast.
- ◆ Having a large number of disks in a system presents opportunities for improving the rate at which data can be read or written, if the disks are operated in parallel. Several independent reads or writes can also be performed in parallel. Furthermore, this setup offers the potential for improving the reliability of data storage, because redundant information can be stored on multiple disks. Thus, failure of one disk does not lead to loss of data.
- ◆ A variety of disk-organization techniques, collectively called redundant arrays of independent disks (RAID), have been proposed to achieve improved performance and reliability.

RAID

- ◆ The simplest (but most expensive) approach to introducing redundancy is to duplicate every disk. This technique is called **mirroring** (or, sometimes, shadowing). A logical disk then consists of two physical disks, and every write is carried out on both disks. If one of the disks fails, the data can be read from the other. Data will be lost only if the second disk fails before the first failed disk is repaired.
- ◆ With multiple disks, we can improve the transfer rate as well (or instead) by **striping data** across multiple disks. In its simplest form, data striping consists of splitting the bits of each byte across multiple disks; such striping is called **bit-level striping**. For example, if we have an array of eight disks, we write bit i of each byte to disk i .
- ◆ **Block-level striping** stripes blocks across multiple disks.

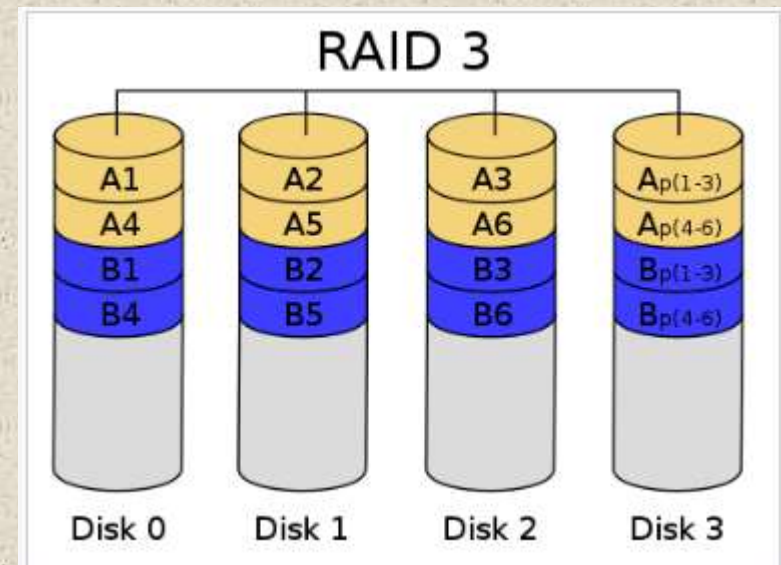
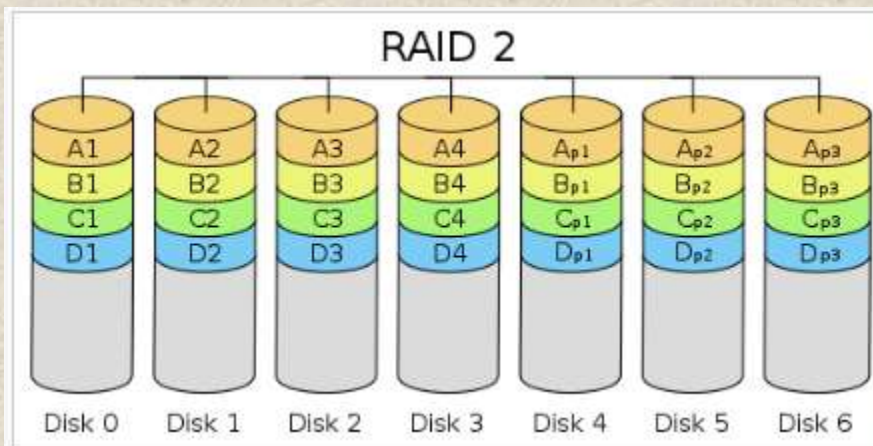
RAID Levels

- ◆ **RAID 0** (also known as a *stripe set* or *striped volume*) splits ("stripes") data evenly across two or more disks, without parity information, redundancy.
- ◆ **RAID 1** consists of an exact copy (or *mirror*) of a set of data on two or more disks.



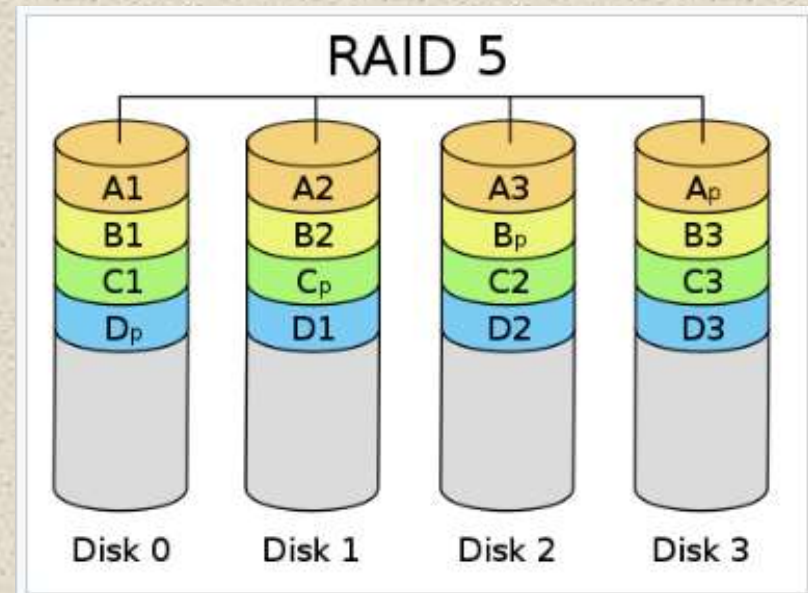
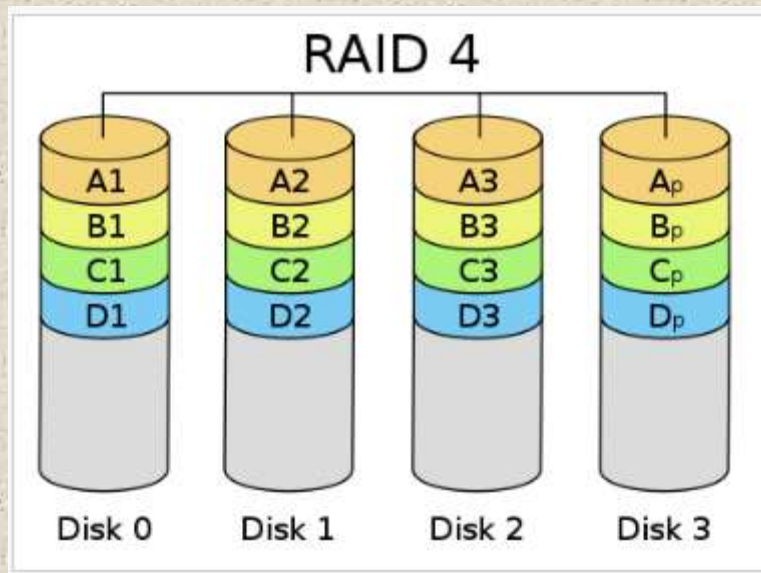
RAID Levels

- ◆ **RAID 2**, stripes data at the bit (rather than block) level, and uses a Hamming code for error correction.
- ◆ **RAID 3**, consists of byte-level (block-level) striping with a dedicated parity disk.



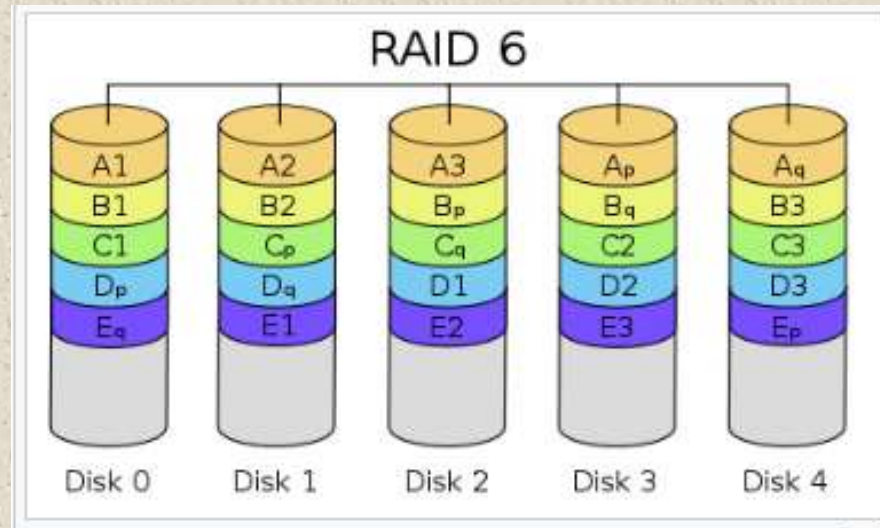
RAID Levels

- ◆ **RAID 4** consists of block-level striping with a dedicated parity disk.
- ◆ **RAID 5** consists of block-level striping with distributed parity. Unlike in RAID 4, parity information is distributed among the drives.



RAID Levels

- ◆ **RAID 6** extends RAID 5 by adding another parity block; thus, it uses block-level striping with two parity blocks distributed across all member disks



Tertiary Storage

- ◆ In a large database system, some of the data may have to reside on tertiary storage. The two most common tertiary storage media are **optical disks and magnetic tapes**.
- ◆ Tapes are used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another. Tapes are also used for storing large volumes of data, such as video or image data, that either do not need to be accessible quickly or are so voluminous that magnetic-disk storage would be too expensive

File Organization

- ◆ A database is mapped into a number of different files that are maintained by the underlying operating system. These files reside permanently on disks. A file is organized logically as a sequence of records. These records are mapped onto disk blocks. Files are provided as a basic construct in operating systems, so we shall assume the existence of an underlying file system. We need to consider ways of representing logical data models in terms of files.
- ◆ Each file is also logically partitioned into fixed-length storage units called blocks, which are the units of both storage allocation and data transfer.

File Organization

- ◆ A database is mapped into a number of different files that are maintained by the underlying operating system. These files reside permanently on disks. A file is organized logically as a sequence of records. These records are mapped onto disk blocks. Files are provided as a basic construct in operating systems, so we shall assume the existence of an underlying file system. We need to consider ways of representing logical data models in terms of files.
- ◆ Each file is also logically partitioned into fixed-length storage units called blocks, which are the units of both storage allocation and data transfer.

File Organization

- ◆ A block may contain several records; the exact set of records that a block contains is determined by the form of physical data organization being used. It's an assumption that no record is larger than a block.
- ◆ we shall require that each record is entirely contained in a single block; that is, no record is contained partly in one block, and partly in another. This restriction simplifies and speeds up access to data items.

File Organization

- ◆ A block may contain several records; the exact set of records that a block contains is determined by the form of physical data organization being used. It's an assumption that no record is larger than a block.
- ◆ we shall require that each record is entirely contained in a single block; that is, no record is contained partly in one block, and partly in another. This restriction simplifies and speeds up access to data items.
- ◆ In a relational database, tuples of distinct relations are generally of different sizes. One approach to mapping 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; however, files of fixed-length records are easier to implement than are files of variable-length records.

File Organization

- ◆ let us consider a file of instructor records for a university database. Each record of this file is defined (in pseudocode) as:

```
type instructor = record
    ID varchar (5);
    name varchar(20);
    dept name varchar (20);
    salary numeric (8,2);
end
```

File Organization

- ◆ Fixed Length Records:
- ◆ Assume that each character occupies 1 byte and that numeric (8,2) occupies 8 bytes.
- ◆ Suppose that instead of allocating a variable amount of bytes for the attributes ID, name, and dept name, we allocate the maximum number of bytes that each attribute can hold. Then, the instructor record is 53 bytes long.
- ◆ A simple approach is to use the first 53 bytes for the first record, the next 53 bytes for the second record, and so on.

File Organization

◆ Fixed Length Records:

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Figure 10.4 File containing *instructor* records.

File Organization

- ◆ Fixed Length Records:
- ◆ Unless the block size happens to be a multiple of 53 (which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.
- ◆ 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, or we must have a way of marking deleted records so that they can be ignored.

File Organization

- ◆ Fixed Length Records:
- ◆ To avoid the first problem, we allocate only as many records to a block as would fit entirely in the block (this number can be computed easily by dividing the block size by the record size, and discarding the fractional part). Any remaining bytes of each block are left unused.
- ◆ When a record is deleted, we could move the record that came after it into the space formerly occupied by the deleted record, and so on, until every record following the deleted record has been moved ahead.
- ◆ Such an approach requires moving a large number of records. It might be easier simply to move the final record of the file into the space occupied by the deleted record. It is undesirable to move records to occupy the space freed by a deleted record, since doing so requires additional block accesses.

File Organization

◆ Fixed Length Records:

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Figure 10.5 File of Figure 10.4, with record 3 deleted and all records moved.

File Organization

◆ Fixed Length Records:

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Figure 10.6 File of Figure 10.4, with record 3 deleted and final record moved.

File Organization

- ◆ Fixed Length Records:
- ◆ Since insertions tend to be more frequent than deletions, it is acceptable to leave open the space occupied by the deleted record, and to wait for a subsequent insertion before reusing the space.
- ◆ A simple marker on a deleted record is not sufficient, since it is hard to find this available space when an insertion is being done. Thus, we need to introduce an additional structure.
- ◆ At the beginning of the file, we allocate a certain number of bytes as a file header. The header will contain a variety of information about the file. For now, all we need to store there is the address of the first record whose contents are deleted. We use this first record to store the address of the second available record, and so on. Intuitively, we can think of these stored addresses as pointers, since they point to the location of a record. The deleted records thus form a linked list, which is often referred to as a free list.

File Organization

◆ Fixed Length Records:

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000




Figure 10.7 File of Figure 10.4, with free list after deletion of records 1, 4, and 6.

File Organization

- ◆ Fixed Length Records:
- ◆ On insertion of a new record, we use the record pointed to by the header. We change the header pointer to point to the next available record. If no space is available, we add the new record to the end of the file.
- ◆ Insertion and deletion for files of fixed-length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record. If we allow records of variable length in a file, this match no longer holds. An inserted record may not fit in the space left free by a deleted record, or it may fill only part of that space.

File Organization

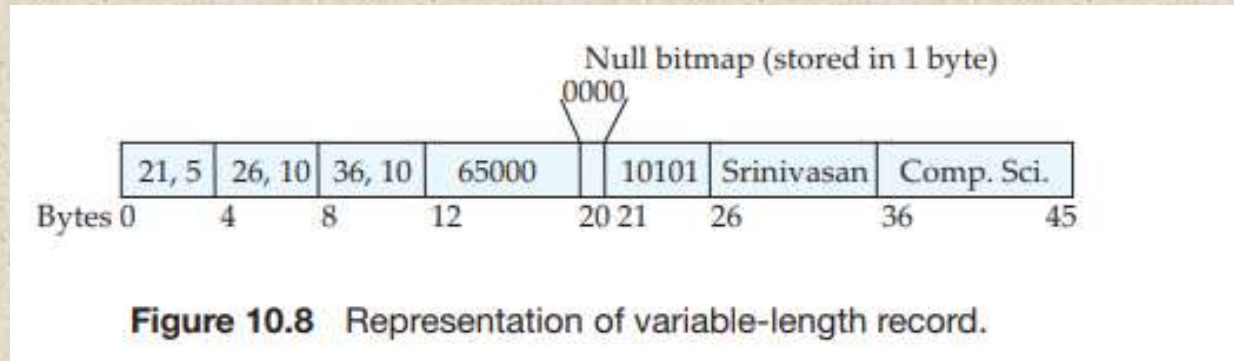
- ◆ Variable Length Records:
- ◆ Variable-length records arise in database systems in several ways:
 - Storage of multiple record types in a file.
 - Record types that allow variable lengths for one or more fields.
 - Record types that allow repeating fields, such as arrays or multisets.
- ◆ Different techniques for implementing variable-length records exist. Two different problems must be solved by any such technique:
 - How to represent a single record in such a way that individual attributes can be extracted easily.
 - How to store variable-length records within a block, such that records in a block can be extracted easily.

File Organization

- ◆ Variable Length Records:
- ◆ The representation of a record with variable-length attributes typically has two parts: an initial part with fixed length attributes, followed by data for variable length attributes.
- ◆ Fixed-length attributes, such as numeric values, dates, or fixed length character strings are allocated as many bytes as required to store their value.
- ◆ Variable-length attributes, such as varchar types, are represented in the initial part of the record by a pair (offset, length), where offset denotes where the data for that attribute begins within the record, and length is the length in bytes of the variable-sized attribute. The values for these attributes are stored consecutively, after the initial fixed-length part of the record.
- ◆ Thus, the initial part of the record stores a fixed size of information about each attribute, whether it is fixed-length or variable-length.

File Organization

◆ Variable Length Records:



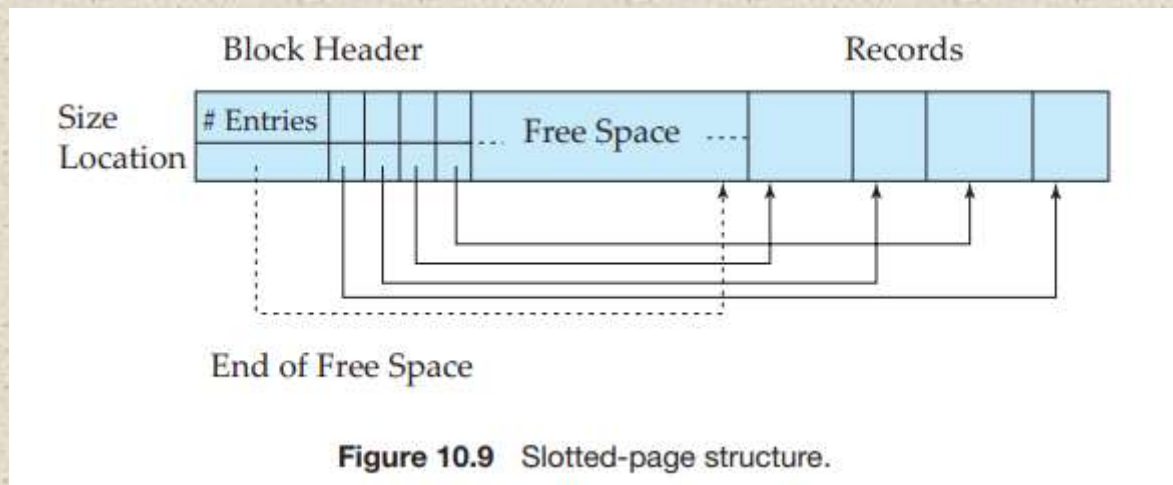
- ◆ An example of such a record representation is shown in Figure 10.8. The figure shows an instructor record, whose first three attributes ID, name, and dept name are variable-length strings, and whose fourth attribute salary is a fixed-sized number. We assume that the offset and length values are stored in two bytes each, for a total of 4 bytes per attribute. The salary attribute is assumed to be stored in 8 bytes, and each string takes as many bytes as it has characters.

File Organization

- ◆ Variable Length Records:
- ◆ The Figure 10.8 also illustrates the use of a null bitmap, which indicates which attributes of the record have a null value. In this particular record, if the salary were null, the fourth bit of the bitmap would be set to 1, and the salary value stored in bytes 12 through 19 would be ignored.
- ◆ In some representations, the null bitmap is stored at the beginning of the record, and for attributes that are null, no data (value, or offset/length) are stored at all. Such a representation would save some storage space.

File Organization

- ◆ Variable Length Records:
- ◆ The slotted-page structure is commonly used for organizing records within a block, and is shown in Figure 10.9.
- ◆ There is a header at the beginning of each block, containing the following information:
 - The number of record entries in the header.
 - The end of free space in the block.
 - An array whose entries contain the location and size of each record



File Organization

- ◆ Variable Length Records:
- ◆ The actual records are allocated contiguously in the block, starting from the end of the block.
- ◆ The free space in the block is contiguous, between the final entry in the header array, and the first record.
- ◆ If a record is inserted, space is allocated for it at the end of free space, and an entry containing its size and location is added to the header.

Organization of Records in Files

- ◆ **Heap file organization:** Any record can be placed anywhere in the file where there is space for the record. There is no ordering of records. Typically, there is a single file for each relation
- ◆ **Sequential file organization:** Records are stored in sequential order, according to the value of a “search key” of each record.
- ◆ **Hashing 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

Data-Dictionary Storage

- ◆ A relational database system needs to maintain data about the relations, such as the schema of the relations. In general, such “data about data” is referred to as metadata.
- ◆ Relational schemas and other metadata about relations are stored in a structure called the **data dictionary or system catalog**.
- ◆ 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)

Data-Dictionary Storage

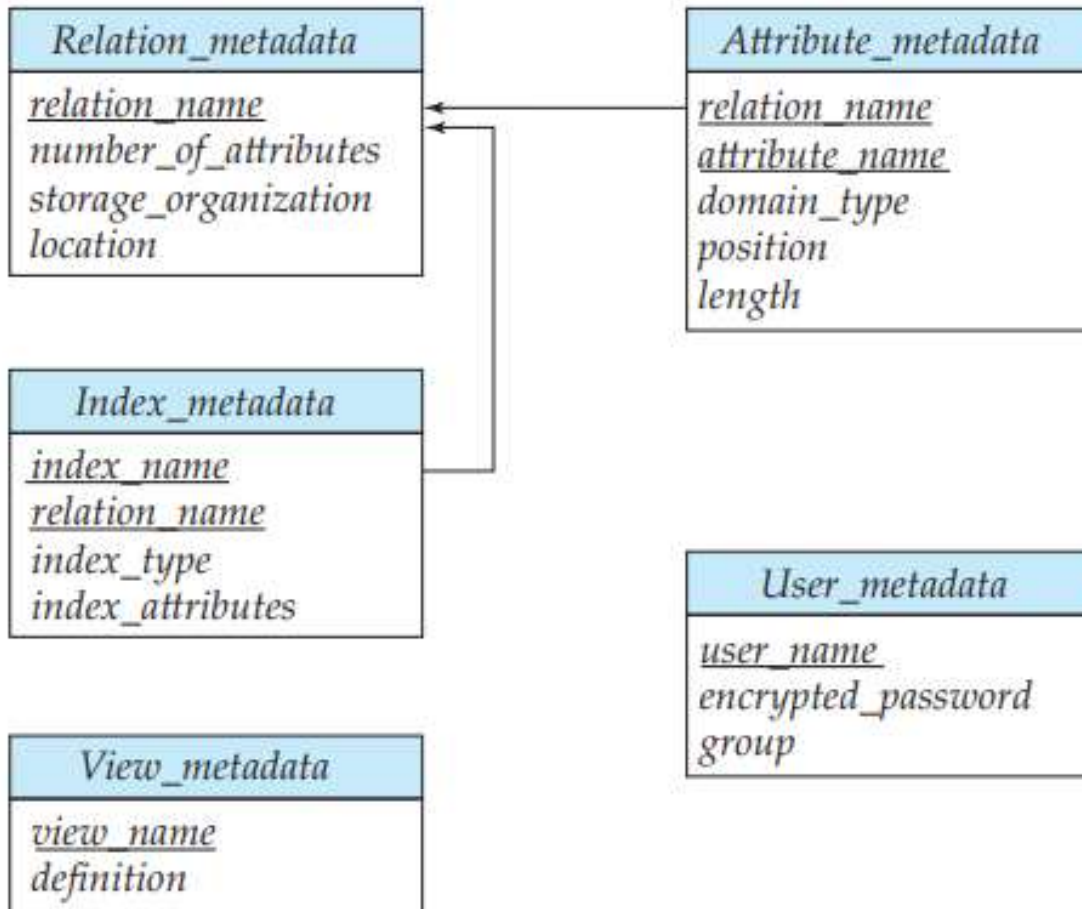


Figure 10.16 Relational schema representing system metadata.

Basic Concepts of Indexing

- ◆ An index for a file in a database system works in much the same way as the index in this textbook. If we want to learn about a particular topic (specified by a word or a phrase) in this textbook, we can search for the topic in the index at the back of the book, find the pages where it occurs, and then read the pages to find the information for which we are looking. The words in the index are in sorted order, making it easy to find the word we want. Moreover, the index is much smaller than the book, further reducing the effort needed.
- ◆ Database-system indices play the same role as book indices in libraries. For example, to retrieve a student record given an ID, the database system would look up an index to find on which disk block the corresponding record resides, and then fetch the disk block, to get the appropriate student record.

Basic Concepts of Indexing

- ◆ Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.
- ◆ An **index** is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations. An **index allows** us to efficiently retrieve all records that satisfy search conditions on the search key fields of the index.
- ◆ We can also create additional indexes on a given collection of data records, each with a different search key, to speed up search operations that are not efficiently supported by the file organization used to store the data records.

Basic Concepts of Indexing

- ◆ There are two basic kinds of indices:
 - **Ordered indices: Based on a sorted ordering of the values.**
 - **Hash indices. Based on a uniform distribution of values across a range of buckets.** The bucket to which a value is assigned is determined by a function, called a *hash function*.

Basic Concepts of Indexing

- ◆ Each technique must be evaluated on the basis of these factors:
 - **Access types:** The types of access that are supported efficiently. Access types can include finding records with a specified attribute value and finding records whose attribute values fall in a specified range.
 - **Access time:** The time it takes to find a particular data item, or set of items, using the technique in question.
 - **Insertion time:** The time it takes to insert a new data item. This value includes the time it takes to find the correct place to insert the new data item, as well as the time it takes to update the index structure.
 - **Deletion time:** The time it takes to delete a data item. This value includes the time it takes to find the item to be deleted, as well as the time it takes to update the index structure.
 - **Space overhead:** The additional space occupied by an index structure. Provided that the amount of additional space is moderate, it is usually worthwhile to sacrifice the space to achieve improved performance.

Ordered Indices

- ◆ An attribute or set of attributes used to look up records in a file is called a search key.
- ◆ To gain fast random access to records in a file, we can use an index structure. Each index structure is associated with a particular search key. Just like the index of a book or a library catalog, **an ordered index stores** the values of the search keys in sorted order, and associates with each search key the records that contain it.
- ◆ A file may have several indices, on different search keys. If the file containing the records is sequentially ordered, a **clustering index is an index** whose search key also defines the sequential order of the file.

Ordered Indices

- ◆ **Clustering indices** are also called **primary indices**; the term **primary index** may appear to denote an index on a primary key, but such indices can in fact be built on any search key.
- ◆ The search key of a clustering index is often the primary key, although that is not necessarily so.
- ◆ Indices whose search key specifies an order different from the sequential order of the file are called **nonclustering indices**, or **secondary indices**.
- ◆ An **index entry**, or **index record**, consists of a **search-key value** and **pointers** to one or more records with that value as their search-key value. The pointer to a record consists of the identifier of a disk block and an offset within the disk block to identify the record within the block.

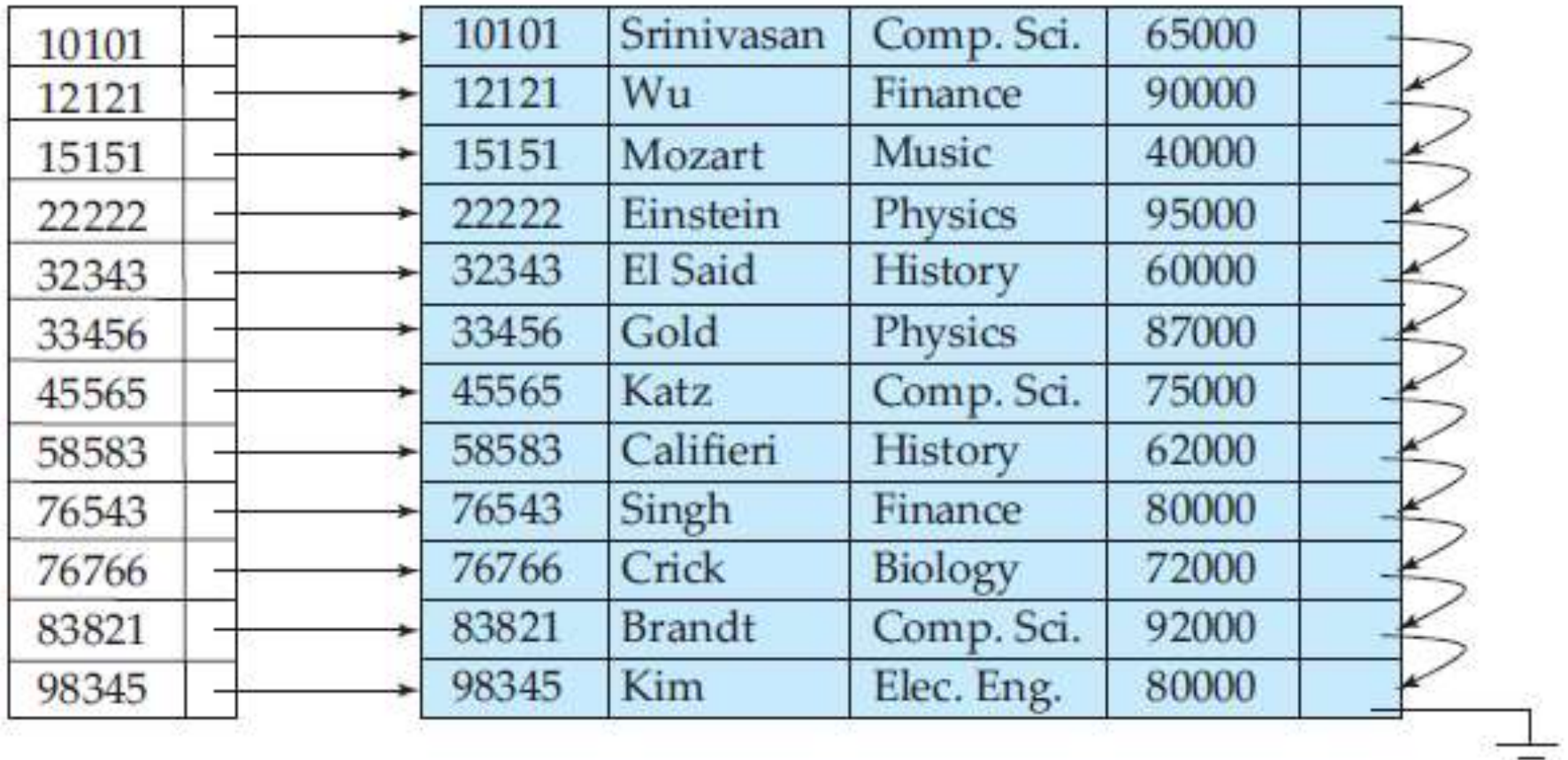
Ordered Indices

- ◆ There are two types of ordered indices; **Dense and Sparse.**
- ◆ **Dense index:** In a dense index, an index entry appears for every **search-key** value in the file.
- ◆ In a dense clustering index, the index record contains the search-key value and a pointer to the first data record with that search-key value.
- ◆ The rest of the records with the same search-key value would be stored sequentially after the first record, since, because the index is a clustering one, records are sorted on the same search key.

Ordered Indices

◆ Dense Index:

10101	→	10101	Srinivasan	Comp. Sci.	65000	↙
12121	→	12121	Wu	Finance	90000	↘
15151	→	15151	Mozart	Music	40000	↙
22222	→	22222	Einstein	Physics	95000	↘
32343	→	32343	El Said	History	60000	↙
33456	→	33456	Gold	Physics	87000	↘
45565	→	45565	Katz	Comp. Sci.	75000	↙
58583	→	58583	Califieri	History	62000	↘
76543	→	76543	Singh	Finance	80000	↙
76766	→	76766	Crick	Biology	72000	↘
83821	→	83821	Brandt	Comp. Sci.	92000	↙
98345	→	98345	Kim	Elec. Eng.	80000	↘

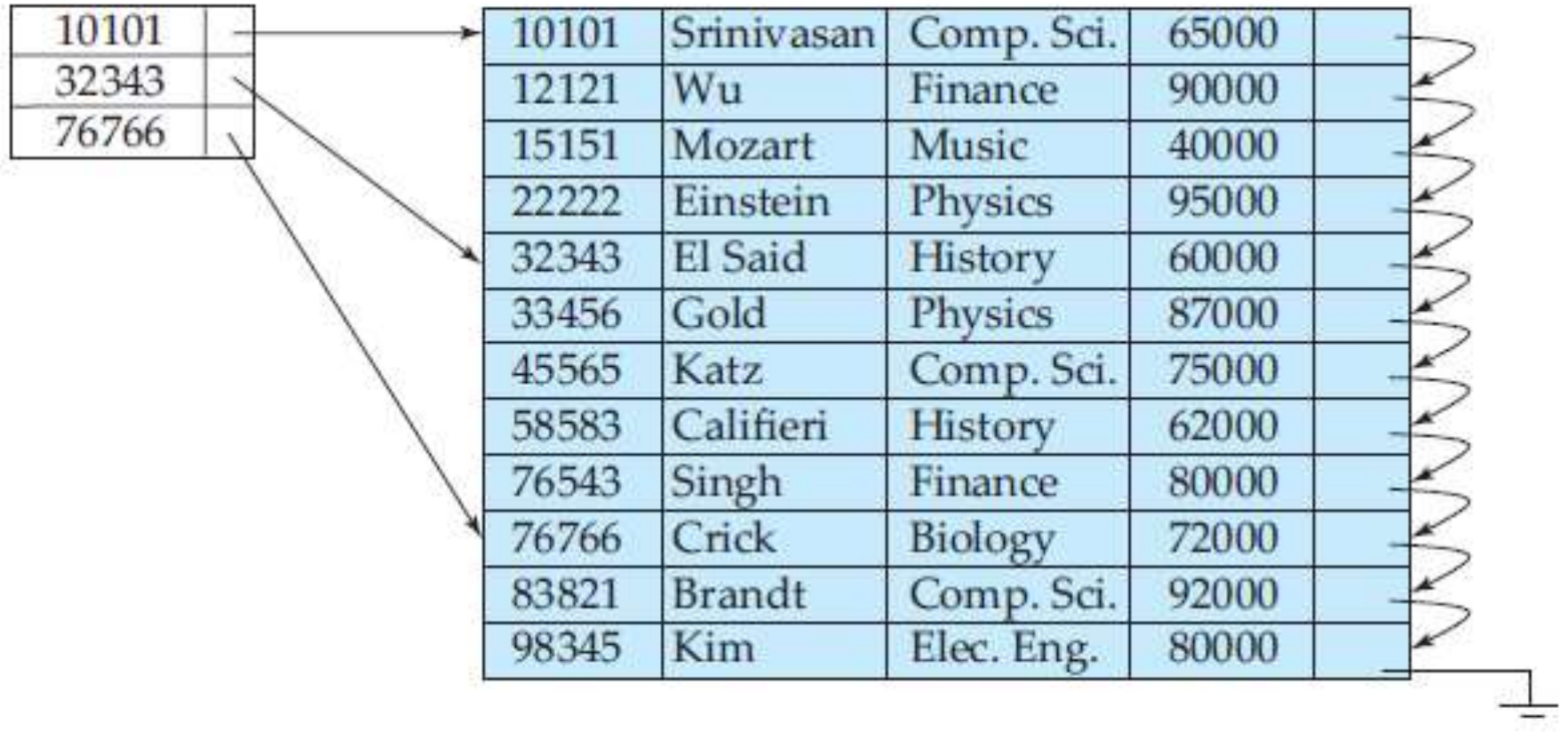


Ordered Indices

- ◆ **Sparse index:**
- ◆ In a sparse index, an index entry appears for only some of the search-key values. Sparse indices can be used only if the relation is stored in sorted order of the search key, that is, if the index is a clustering index. As is true in dense indices, each index entry contains a search-key value and a pointer to the first data record with that search-key value.
- ◆ To locate a record, we find the index entry with the **largest search-key value** that is less than or equal to the search-key value for which we are looking. We start at the record pointed to by that index entry, and follow the pointers in the file until we find the desired record.

Ordered Indices

- ◆ **Sparse index:**



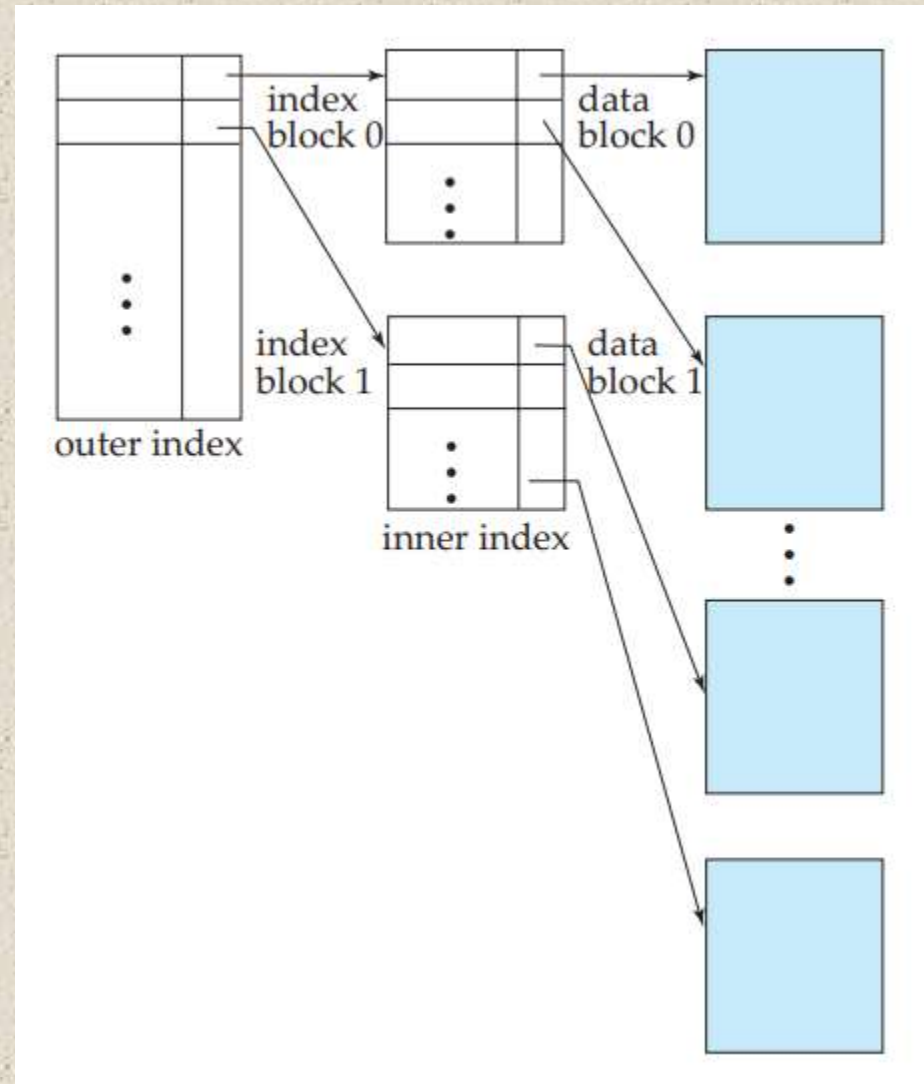
Ordered Indices

- ◆ It is generally faster to locate a record if we have a dense index rather than a sparse index. However, sparse indices have advantages over dense indices in that they require less space and they impose less maintenance overhead for insertions and deletions.

Multilevel Indices

- ◆ Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.
- ◆ Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.
- ◆ Multilevel indices are closely related to tree structures, such as the binary trees used for in-memory indexing.

Multilevel Indices



B+-Tree Index Files

- ◆ The main disadvantage of the index-sequential file organization is that performance degrades as the file grows, both for index lookups and for sequential scans through the data. Although this degradation can be remedied by reorganization of the file, frequent reorganizations are undesirable.
- ◆ The B+-tree index structure is the most widely used of several index structures that maintain their efficiency despite insertion and deletion of data. A B+-tree index takes the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length. Each nonleaf node in the tree has between $n/2$ and n children, where n is fixed for a particular tree.

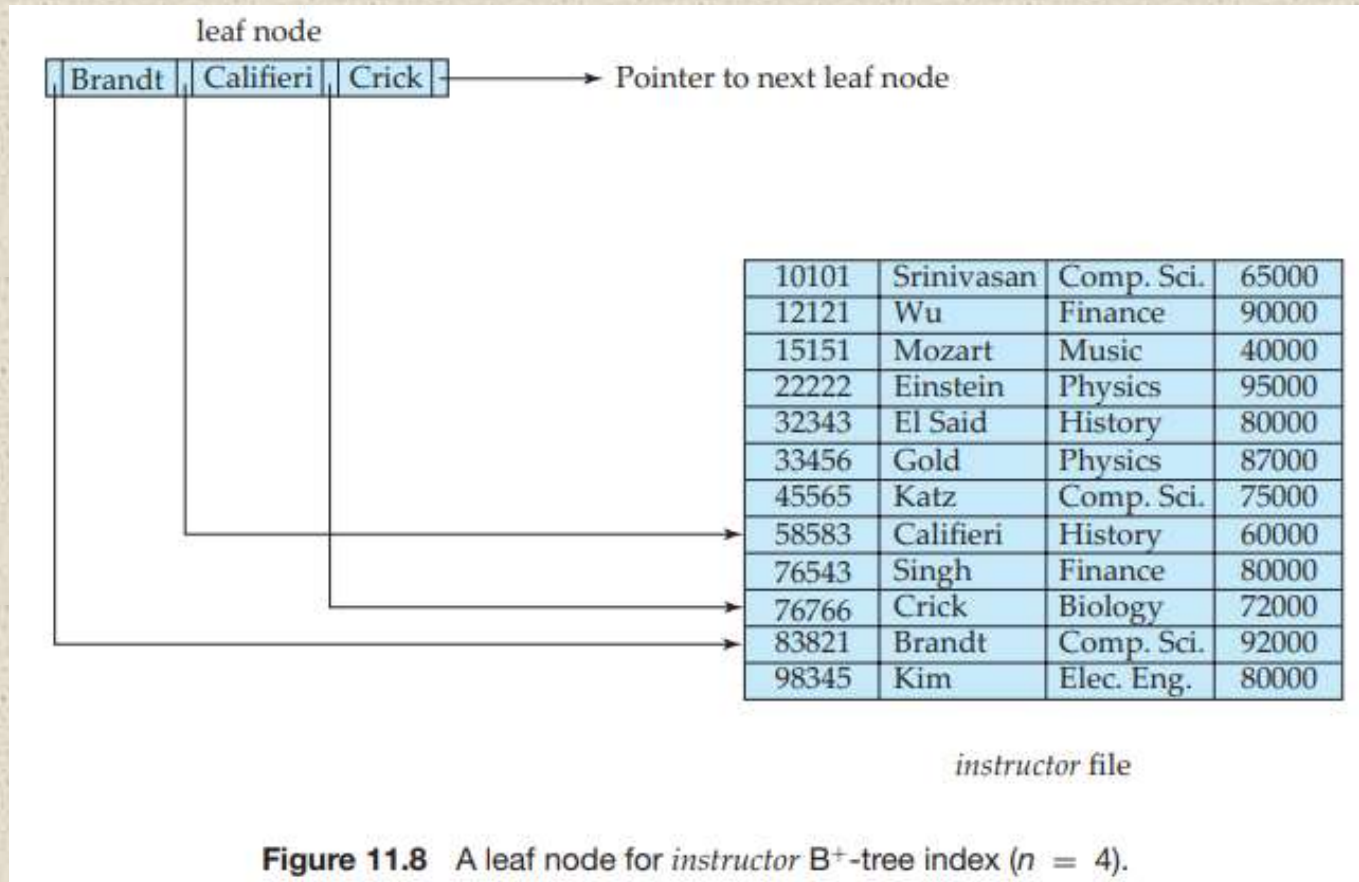
B+-Tree Index Files

- ◆ A B+-tree index is a multilevel index, but it has a structure that differs from that of the multilevel index-sequential file. A node of a B+- tree contains up to $n - 1$ search-key values K_1, K_2, \dots, K_{n-1} , and n pointers P_1, P_2, \dots, P_n .
- ◆ The search-key values within a node are kept in sorted order; thus, if $i < j$, then $K_i < K_j$. We consider first the structure of the leaf nodes. For $i = 1, 2, \dots, n-1$, pointer P_i points to a file record with search-key value K_i .



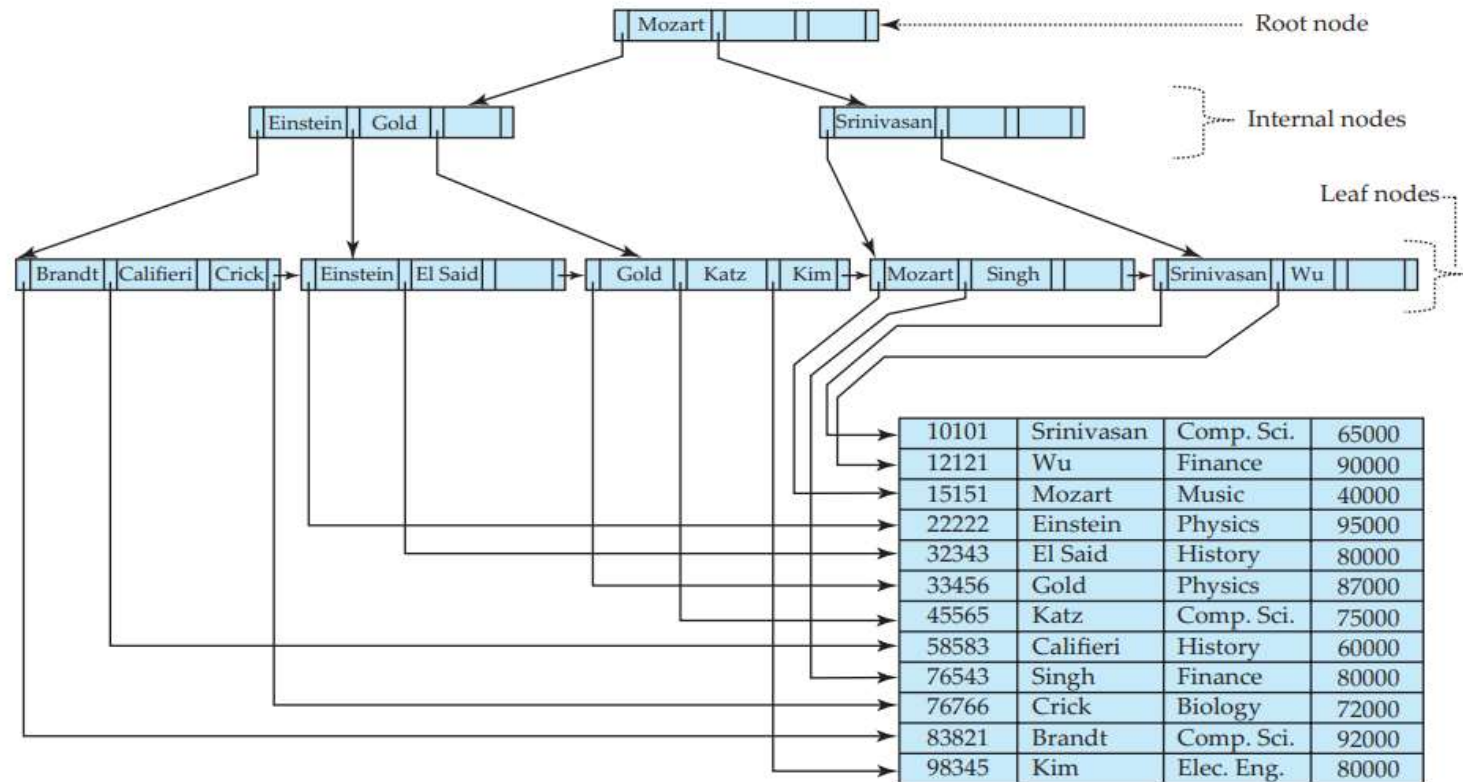
Figure 11.7 Typical node of a B⁺-tree.

B+-Tree Index Files



B+-Tree Index Files

- ◆ The nonleaf or internal nodes of the B+-tree form a multilevel (sparse) index on the leaf nodes. The structure of nonleaf nodes is the same as that for leaf nodes, except that all pointers are pointers to tree nodes.



B+-Tree Index Files

- ◆ The nonleaf nodes of the B+-tree form a multilevel (sparse) index on the leaf nodes. The structure of nonleaf nodes is the same as that for leaf nodes, except that all pointers are pointers to tree nodes.

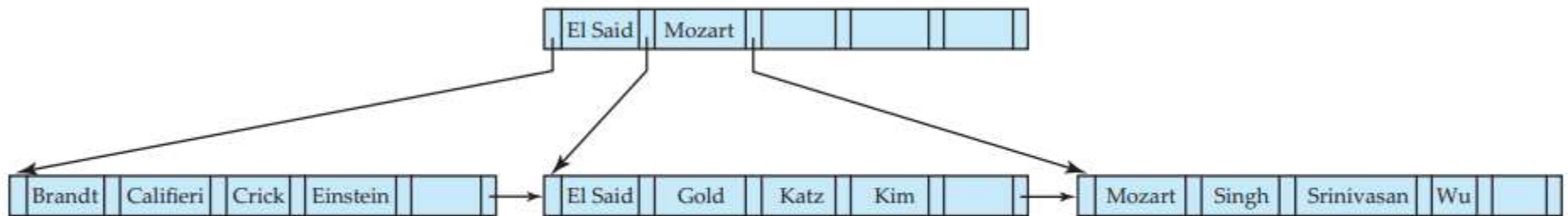


Figure 11.10 B⁺-tree for *instructor* file with $n = 6$.

B+-Tree Index Files

- ◆ We now consider an example of insertion in which a node must be split. Assume that a record is inserted on the instructor relation, with the name value being Adams. We then need to insert an entry for “Adams” into the B+-tree of Figure 11.9.
- ◆ Using the algorithm for lookup, we find that “Adams” should appear in the leaf node containing “Brandt”, “Califieri”, and “Crick.” There is no room in this leaf to insert the search-key value “Adams.” Therefore, the node is split into two nodes. Figure 11.12 shows the two leaf nodes that result from the split of the leaf node on inserting “Adams”. The search-key values “Adams” and “Brandt” are in one leaf, and “Califieri” and “Crick” are in the other.

B+-Tree Index Files

- ◆ Having split a leaf node, we must insert the new leaf node into the B+-tree structure. In our example, the new node has “Califieri” as its smallest search-key value. We need to insert an entry with this search-key value, and a pointer to the new node, into the parent of the leaf node that was split.
- ◆ The B+-tree of Figure 11.13 shows the result of the insertion. It was possible to perform this insertion with no further node split, because there was room in the parent node for the new entry. If there were no room, the parent would have had to be split, requiring an entry to be added to its parent. In the worst case, all nodes along the path to the root must be split. If the root itself is split, the entire tree becomes deeper.

B+-Tree Index Files

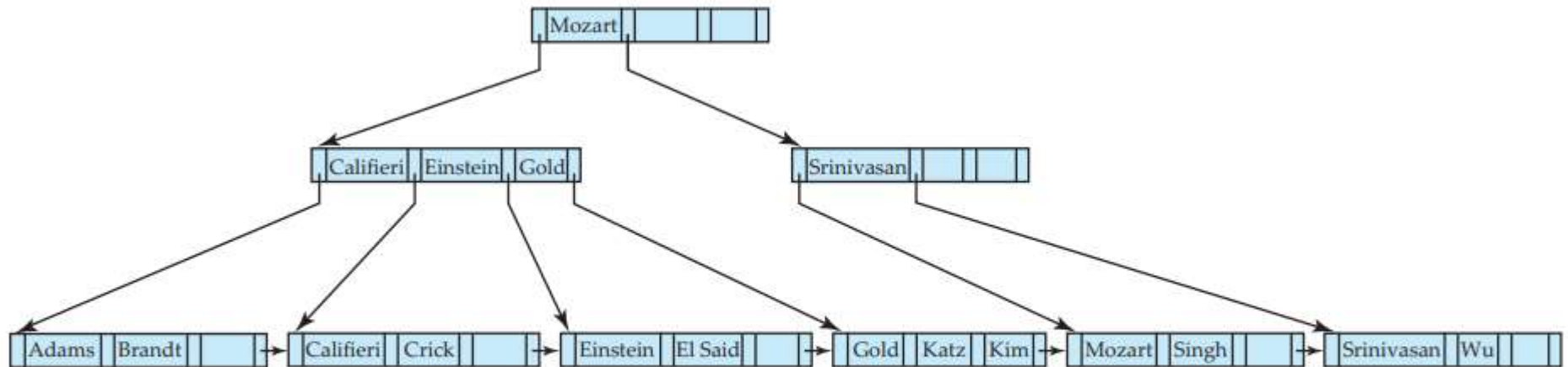


Figure 11.13 Insertion of “Adams” into the B⁺-tree of Figure 11.9.

B+-Tree Index Files

- ◆ Having split a leaf node, we must insert the new leaf node into the B+-tree structure. In our example, the new node has “Califieri” as its smallest search-key value. We need to insert an entry with this search-key value, and a pointer to the new node, into the parent of the leaf node that was split.
- ◆ The B+-tree of Figure 11.13 shows the result of the insertion. It was possible to perform this insertion with no further node split, because there was room in the parent node for the new entry. If there were no room, the parent would have had to be split, requiring an entry to be added to its parent. In the worst case, all nodes along the path to the root must be split. If the root itself is split, the entire tree becomes deeper.

B-Tree Index Files

- ◆ B-tree indices are similar to B+-tree indices. The primary distinction between the two approaches is that a B-tree eliminates the redundant storage of search-key values. In the B+-tree of Figure 11.13, the search keys “Califieri”, “Einstein”, “Gold”, “Mozart”, and “Srinivasan” appear in nonleaf nodes, in addition to appearing in the leaf nodes. Every search-key value appears in some leaf node; several are repeated in nonleaf nodes.
- ◆ A B-tree allows search-key values to appear only once (if they are unique), unlike a B+-tree, where a value may appear in a nonleaf node, in addition to appearing in a leaf node.

B-Tree Index Files

- Figure 11.21 shows a B-tree that represents the same search keys as the B+-tree of Figure 11.13. Since search keys are not repeated in the B-tree, we may be able to store the index in fewer tree nodes than in the corresponding B+-tree index.

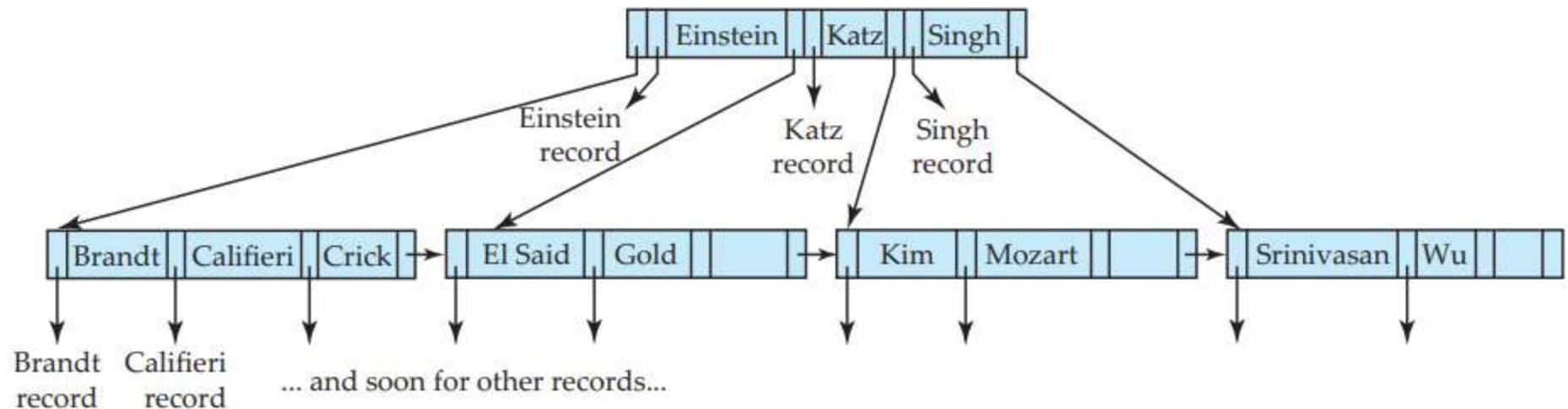


Figure 11.21 B-tree equivalent of B⁺-tree in Figure 11.13.

Multiple-Key Access

- ◆ We can use multiple indices if they exist, or to use an index built on a multiattribute search key.
- ◆ **Using Multiple Single-Key Indices**
- ◆ Assume that the *instructor* file has two indices: one for dept name and one for salary. Consider the following query: “Find all instructors in the Finance department with salary equal to \$80,000.”
We write

```
select ID
from instructor
where dept name = 'Finance' and salary= 80000;
```
- ◆ There are three strategies possible for processing this query:

Multiple-Key Access

- ◆ **Using Multiple Single-Key Indices**
- ◆ There are three strategies possible for processing the above query:
- ◆ **Use the index on *dept name* to find all records pertaining to the *Finance* department.** Examine each such record to see whether *salary*= 80000.
- ◆ **Use the index on *salary* to find all records pertaining to instructors *with* salary of \$80,000.** Examine each such record to see whether the department name is “Finance”.
- ◆ **Use the index on *dept name* to find pointers to all records pertaining to the Finance department.** Also, use the index on *salary* to find pointers to all records pertaining to instructors with a salary of \$80,000. Take the intersection of these two sets of pointers. Those pointers that are in the intersection point to records pertaining to instructors of the Finance department and with salary of \$80,000.

Multiple-Key Access

- ◆ **Indices on Multiple Keys**
- ◆ An alternative strategy for this case is to create and use an index on a composite search key (*dept name, salary*)—that is, the search key consisting of the department name concatenated with the instructor salary.
- ◆ We can use an ordered (B+-tree) index on the above composite search key to answer efficiently queries of the form
select *ID*
from *instructor*
where *dept name* = '*Finance*' and *salary*= 80000

Creating Index

- ◆ **We can create index in sql like;**
- ◆ `CREATE INDEX index_name
ON table_name (column1, column2, ...);`
- ◆ **For Example;**
- ◆ The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:
`CREATE INDEX idx_lastname
ON Persons (LastName);`
- ◆ If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:
`CREATE INDEX idx_pname
ON Persons (LastName, FirstName);`

Dropping Index

- ◆ The DROP INDEX statement is used to delete an index in a table. There are various way to drop index as;

```
ALTER TABLE table_name  
DROP INDEX index_name;
```

Hashing

- ◆ One disadvantage of sequential file organization is that we must access an index structure to locate data, or must use binary search, and that results in more I/O operations. File organizations based on the technique of hashing allow us to avoid accessing an index structure. Hashing also provides a way of constructing indices.
- ◆ In hashing, the term bucket is used 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.
- ◆ Formally, 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.

Hashing

- ◆ 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 for now 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.

Hashing

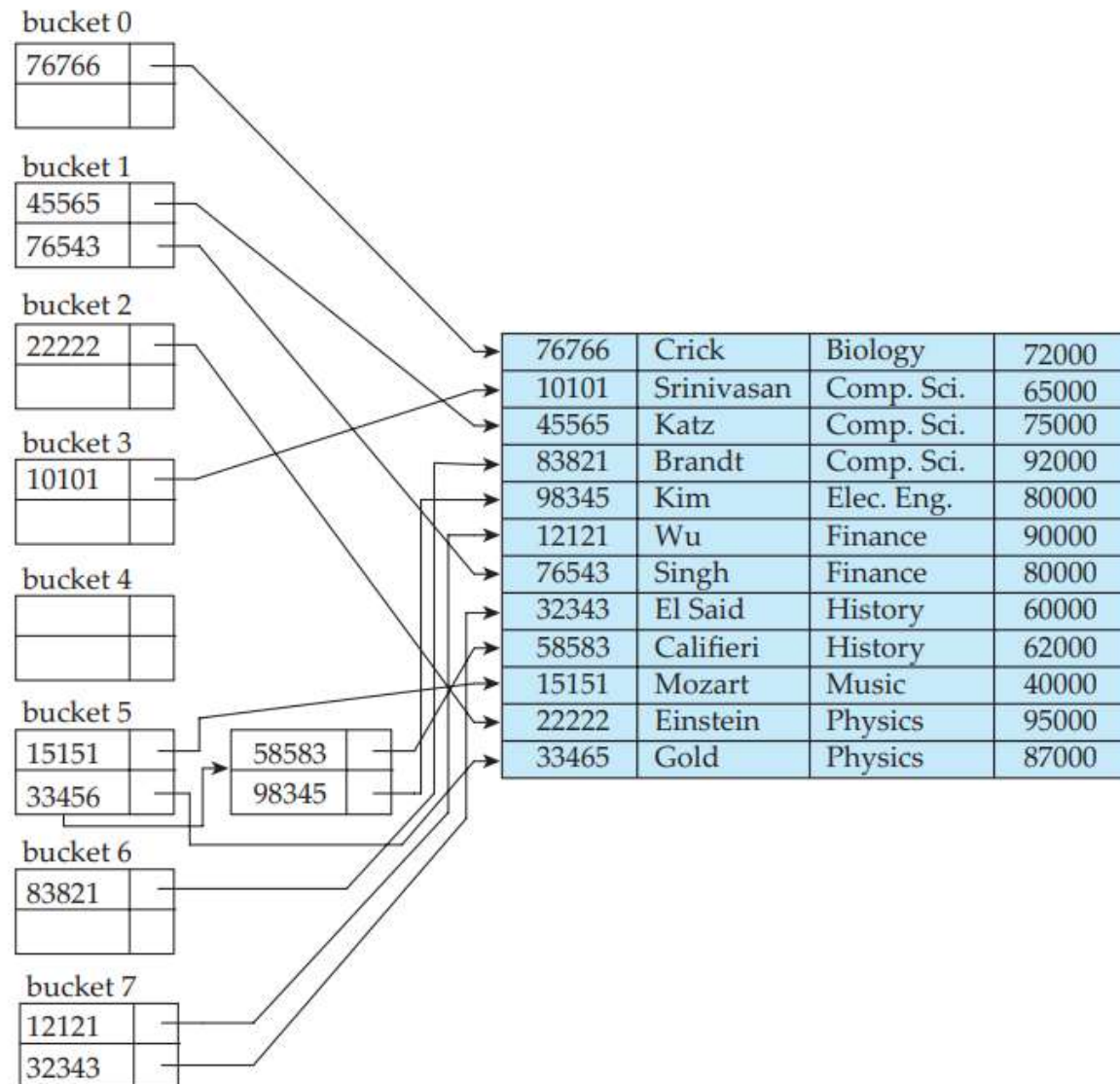


Figure 11.25 Hash index on search key ID of instructor file.

Static Hashing Vs Dynamic Hashing

- ◆ The **main difference** between static and dynamic hashing is that, **in static hashing, the resultant data bucket address is always the same while, in dynamic hashing, the data buckets grow or shrink according to the increase and decrease of records.**
- ◆ In static hashing, the resultant data bucket address is always the same. In other words, the bucket address does not change. Thus, in this method, the number of data buckets in memory remains constant throughout.
- ◆ An issue in static hashing is bucket overflow. Dynamic hashing helps to overcome this issue. It is also called **Extendable hashing method**. In this method, the data buckets increase and decrease depending on the number of records.

Static Hashing Vs Dynamic Hashing

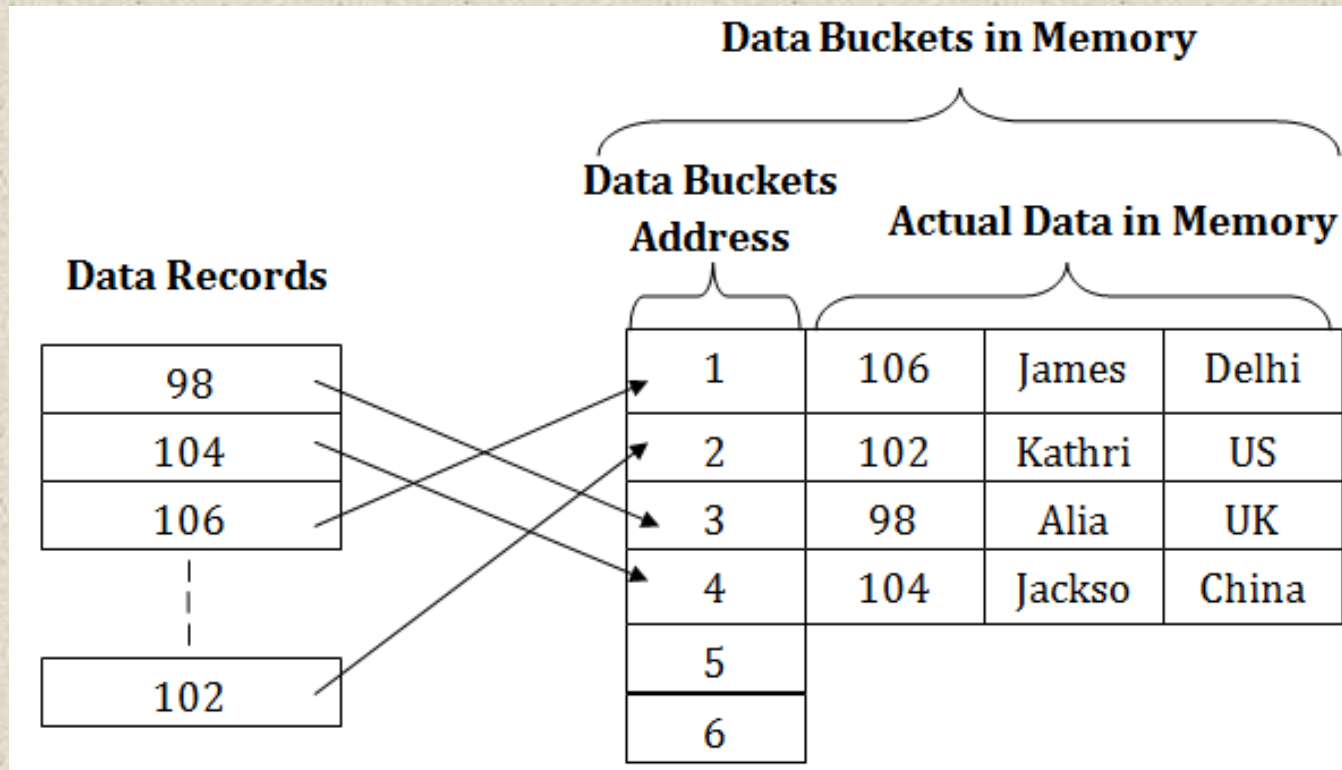
- ◆ Static hashing is suitable for applications where quick access to data is required, and the amount of data to be stored is known in advance. Dynamic hashing is suitable for applications where the amount of data to be stored is not known in advance, and scalability is important.

Static Hashing Vs Dynamic Hashing

- ◆ In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the hash function $\text{mod } (5)$ then it will always result in same bucket address 3. Here, there will be no change in the bucket address.
- ◆ Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.
- ◆ If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**.

Static Hashing Vs Dynamic Hashing

◆ Static Hashing



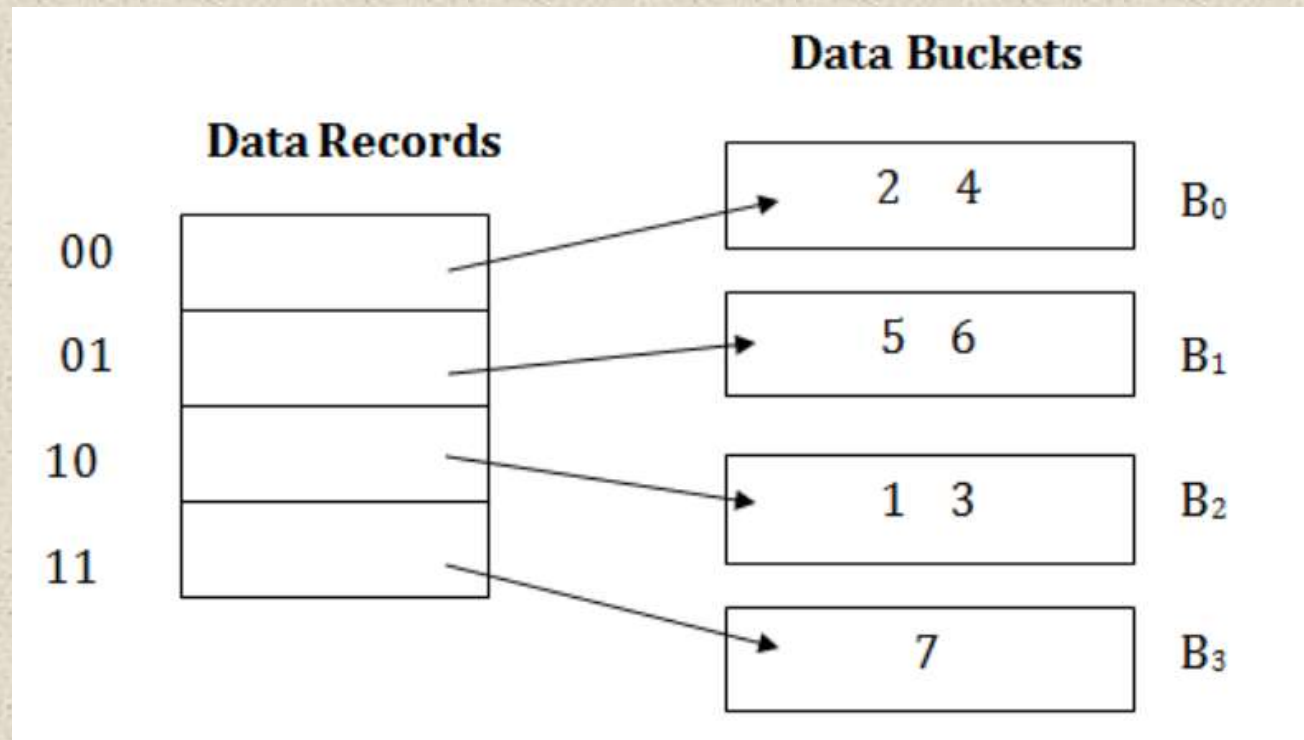
Static Hashing Vs Dynamic Hashing

- ◆ Dynamic Hashing
- ◆ Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

Static Hashing Vs Dynamic Hashing

- ◆ Dynamic Hashing
- ◆ The last two bits of 2 and 4 are 00. So it will go into bucket B₀. The last two bits of 5 and 6 are 01, so it will go into bucket B₁. The last two bits of 1 and 3 are 10, so it will go into bucket B₂. The last two bits of 7 are 11, so it will go into B₃.

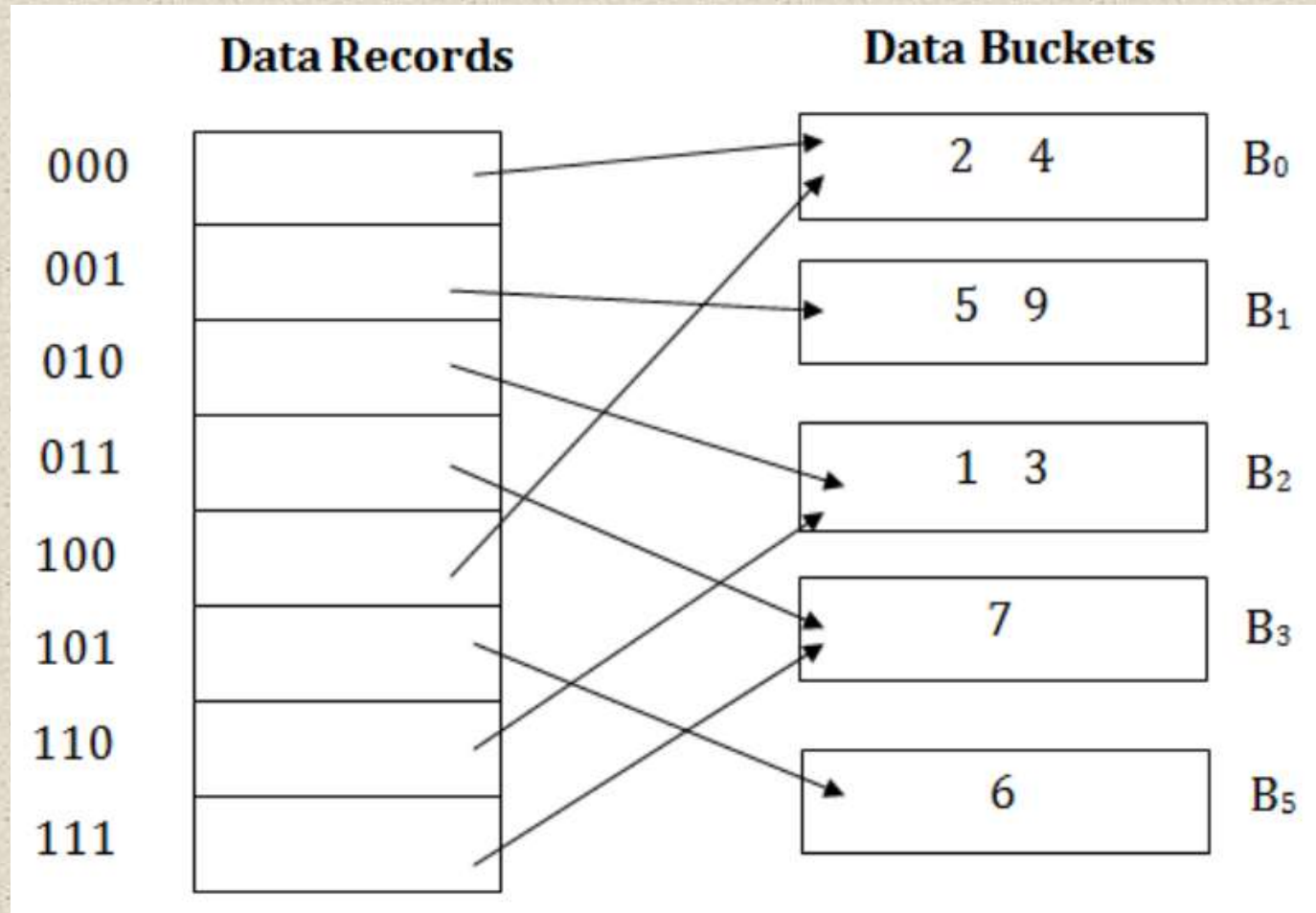


Static Hashing Vs Dynamic Hashing

- ◆ Dynamic Hashing
- ◆ To insert key 9 with hash address 10001 into the above structure:
- ◆ Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- ◆ The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- ◆ Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- ◆ Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- ◆ Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.

Static Hashing Vs Dynamic Hashing

- ◆ Dynamic Hashing

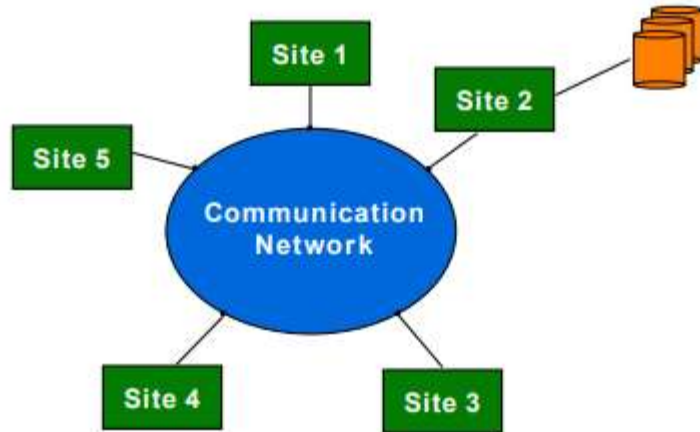


Distributed Database

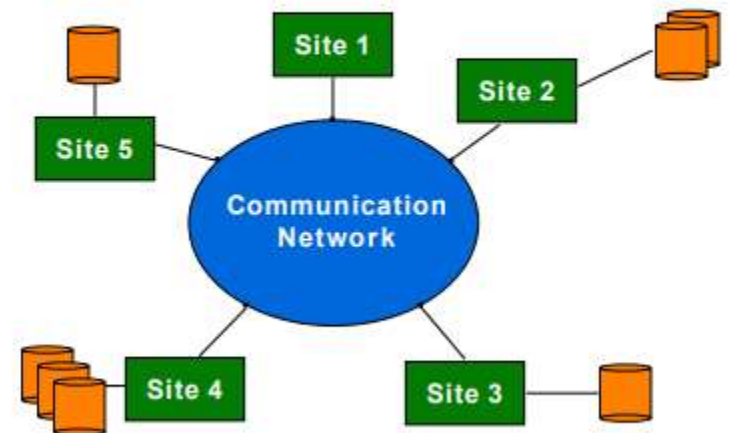
- ◆ Distributed database system is a type of database management system that stores data across multiple computers or sites that are connected by a network.
- ◆ A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network.
- ◆ A distributed database management system (D-DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

Distributed Database

Centralized DBMS on a Network



Distributed DBMS Environment



Distributed Database

- ◆ In a **homogeneous distributed database** system, all sites have identical database management system software, are aware of one another, and agree to cooperate in processing users' requests. In such a system, local sites surrender a portion of their autonomy in terms of their right to change schemas or database-management system software. That software must also cooperate with other sites in exchanging information about transactions, to make transaction processing possible across multiple sites.
- ◆ In contrast, in a **heterogeneous distributed database**, different sites may use different schemas, and different database-management system software. The sites may not be aware of one another, and they may provide only limited facilities for cooperation in transaction processing. The differences in schemas are often a major problem for query processing, while the divergence in software becomes a hindrance for processing transactions that access multiple sites.

Distributed Database

- ◆ Consider a relation r that is to be stored in the database. There are two approaches to storing this relation in the distributed database:
 - Replication. The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The alternative to replication is to store only one copy of relation r .
 - Fragmentation. The system partitions the relation into several fragments, and stores each fragment at a different site.
- ◆ Fragmentation and replication can be combined: A relation can be partitioned into several fragments and there may be several replicas of each fragment.

Distributed Database

◆ Replication

- ◆ If relation r is replicated, a copy of relation r is stored in two or more sites. In the most extreme case, we have **full replication**, in which a copy is stored in every site in the system
- ◆ In general, replication enhances the performance of read operations and increases the availability of data to read-only transactions. However, update transactions incur greater overhead.
- ◆ Controlling concurrent updates by several transactions to replicated data is more complex than in centralized systems.

Distributed Database

- ◆ **Fragmentation**

- ◆ If relation r is fragmented, r is divided into a number of fragments r_1, r_2, \dots, r_n . These fragments contain sufficient information to allow reconstruction of the original relation r . There are two different schemes for fragmenting a relation: **horizontal fragmentation** and **vertical fragmentation**.
- ◆ **Horizontal fragmentation** splits the relation by assigning each tuple of r to one or more fragments.
- ◆ **Vertical fragmentation** splits the relation by decomposing the scheme R of relation r .

Distributed Database

- ◆ **Horizontal Fragmentation**

- ◆ **In horizontal fragmentation**, a relation r is partitioned into a number of subsets, r_1, r_2, \dots, r_n . Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.
- ◆ In general, a **horizontal fragment** can be defined as a selection on the global relation r . That is, we use a predicate P_i to construct fragment r_i :

$$r_i = P_i (r)$$

- ◆ We reconstruct the relation r by taking the union of all fragments; that is:

$$r = r_1 \cup r_2 \cup \dots \cup r_n$$

Distributed Database

- ◆ **Horizontal Fragmentation**
- ◆ As an illustration, the account relation can be divided into several different fragments, each of which consists of tuples of accounts belonging to a particular branch.
- ◆ If the banking system has only two branches—Hillside and Valleyview — then there are two different fragments:
 - account 1 = branch name = “Hillside” (account)
 - account 2 = branch name = “Valleyview” (account)

Distributed Database

- ◆ **Vertical Fragmentation**

- ◆ **Vertical fragmentation** of $r(R)$ involves the definition of several subsets of attributes R_1, R_2, \dots, R_n of the schema R so that:

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

- ◆ Each fragment r_i of r is defined by: $r_i = R_i(r)$.
- ◆ The fragmentation should be done in such a way that we can reconstruct relation r from the fragments by taking the natural join:

$$r = r_1 \star r_2 \star r_3 \star \dots \star r_n$$

Distributed Database

◆ Vertical Fragmentation

- ◆ One way of ensuring that the relation r can be reconstructed is to include the primary-key attributes of R in each R_i . More generally, any superkey can be used. It is often convenient to add a special attribute, called a tuple-id, to the schema R . The tuple-id value of a tuple is a unique value that distinguishes the tuple from all other tuples. The tuple-id attribute thus serves as a candidate key for the augmented schema, and is included in each R_i .
- ◆ To illustrate vertical fragmentation, consider a university database with a relation `employee info` that stores, for each employee, employee id, name, designation, and salary. For privacy reasons, this relation may be fragmented into a relation `employee private info` containing employee id and salary, and another relation `employee public info` containing attributes employee id, name, and designation. These may be stored at different sites, again, possibly for security reasons.

Distributed Database

◆ Transparency

- ◆ The user of a distributed database system should not be required to know where the data are physically located nor how the data can be accessed at the specific local site.
- ◆ This characteristic, called **data transparency**, can take several forms:
 - **Fragmentation transparency.** Users are not required to know how a relation has been fragmented.
 - **Replication transparency.** Users view each data object as logically unique. The distributed system may replicate an object to increase either system performance or data availability. Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed.
 - **Location transparency.** Users are not required to know the physical location of the data. The distributed database system should be able to find any data as long as the data identifier is supplied by the user transaction.

Client Server Technology

- ◆ Refer Handouts from Unit 1.2 Data Models Slide No. 47 to 56.

Multidimensional Databases

- ◆ Multidimensional databases are used mostly for OLAP (online analytical processing) and data warehousing. They can be used to show multiple dimensions of data to users .
- ◆ A multidimensional database is created from multiple relational databases. While relational databases allow users to access data in the form of queries, the multidimensional databases allow users to ask analytical questions related to business or market trends.
- ◆ The multidimensional databases uses MOLAP (multidimensional online analytical processing) to access its data. They allow the users to quickly get answers to their requests by generating and analyzing the data rather quickly.
- ◆ The data in multidimensional databases is stored in a data cube format. This means that data can be seen and understood from many dimensions and perspectives.

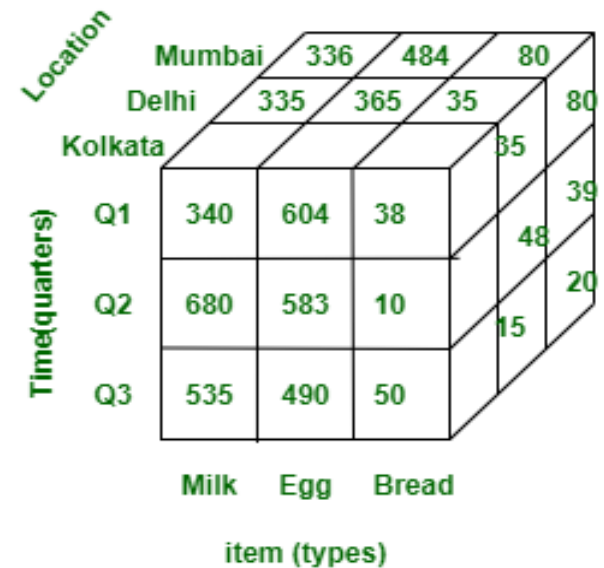
Multidimensional Databases

- ◆ It represents data in the form of data cubes. Data cubes allow to model and view the data from many dimensions and perspectives. It is defined by dimensions and facts and is represented by a fact table. Facts are numerical measures and fact tables contain measures of the related dimensional tables or names of the facts.
- ◆ Multidimensional databases, also known as OLAP (Online Analytical Processing) databases, organize data into a multidimensional structure called a cube. A cube is a data structure that has multiple dimensions, each representing a different aspect of the data, such as time, location, or product. Each dimension is made up of a hierarchy of members, such as days, months, and years for the time dimension, or countries, states, and cities for the location dimension.
- ◆ The process of building an MDDB summarizes the raw data; the data stored in the MDDB is thus said to be presummarized. The MDDB enables users to quickly retrieve multiple levels of presummarized data through a multidimensional view.

Multidimensional Databases

Time	Location="Kolkata"			Location="Delhi"			Location="Mumbai"		
	item			item			item		
	Milk	Egg	Bread	Milk	Egg	Bread	Milk	Egg	Bread
Q1	340	604	38	335	365	35	336	484	80
Q2	680	583	10	684	490	48	595	594	39
Q3	535	490	50	389	385	15	366	385	20

3D data representation as 2D



3D data representation

Parallel Databases

- ◆ Nowadays organizations need to handle a huge amount of data with a high transfer rate. For such requirements, the client-server or centralized system is not efficient. With the need to improve the efficiency of the system, the concept of the parallel database comes in picture. A parallel database system seeks to improve the performance of the system through parallelizing concept.
- ◆ **A parallel database** is one which involves multiple processors and working in parallel on the database used to provide the services.
- ◆ A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries

Parallel Databases

- ◆ Parallel DBMS is a Database Management System that runs through multiple processors and disks. They combine two or more processors also disk storage that helps make operations and executions easier and faster.
- ◆ **Parallel Databases** are designed to execute concurrent operations. They exist, happen, or done at the same time even if the data processed are not from one source or one processing unit.
- ◆ Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel.
- ◆ In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

Parallel Databases

- ♦ **Intraquery parallelism**

- ♦ A single query that is executed in parallel using multiple processors or disks.

- ♦ **Independent parallelism**

- ♦ Execution of each operation individually in different processors only if they can be executed independent of each other. For example, if we need to join four tables, then two can be joined at one processor and the other two can be joined at another processor. Final join can be done later.

Parallel Databases

- ♦ **Pipe-lined parallelism**

- ♦ Execution of different operations in pipe-lined fashion. For example, if we need to join three tables, one processor may join two tables and send the result set records as and when they are produced to the other processor. In the other processor the third table can be joined with the incoming records and the final result can be produced.

- ♦ **Intraoperation parallelism**

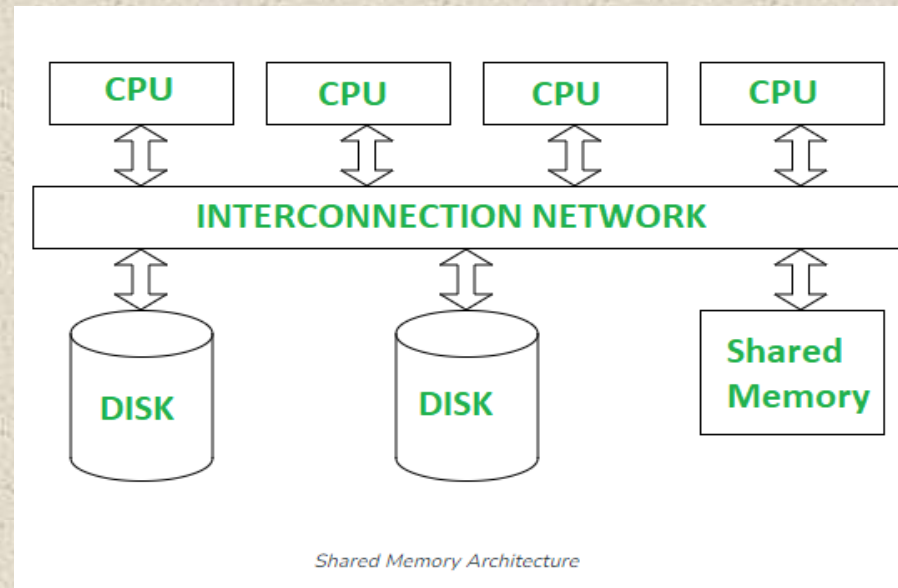
- ♦ Execution of single complex or large operations in parallel in multiple processors. For example, ORDER BY clause of a query that tries to execute on millions of records can be parallelized on multiple processors.

Parallel Databases

- ◆ Multiple resources like CPUs and Disks are used in parallel. The operations are performed simultaneously, as opposed to serial processing. A parallel server can allow access to a single database by users on multiple machines. It also performs many parallelization operations like data loading, query processing, building indexes, and evaluating queries. Parallel databases may have following architectures;
 - Shared Memory Architecture
 - Shared Disk Architecture
 - Shared Nothing Architecture
 - Hierarchical Architecture

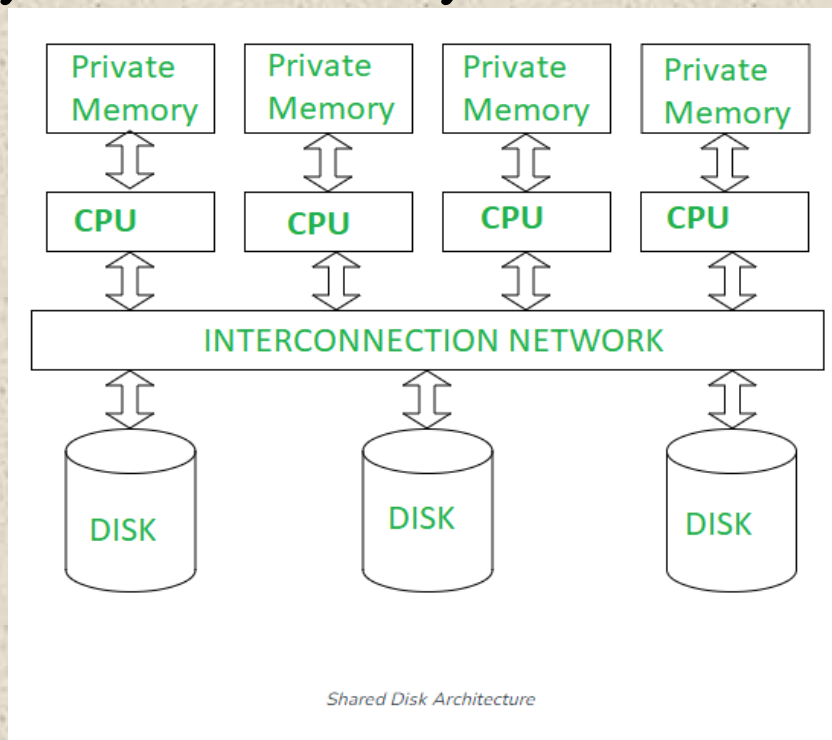
Parallel Databases

- ◆ **Shared Memory Architecture-** In Shared Memory Architecture, there are multiple CPUs that are attached to an interconnection network. They are able to share a single or global main memory and common disk arrays. It is to be noted that, In this architecture, a single copy of a multi-threaded operating system and multithreaded DBMS can support these multiple CPUs. Also, the shared memory is a solid coupled architecture in which multiple CPUs share their memory. It is also known as Symmetric multiprocessing (SMP).



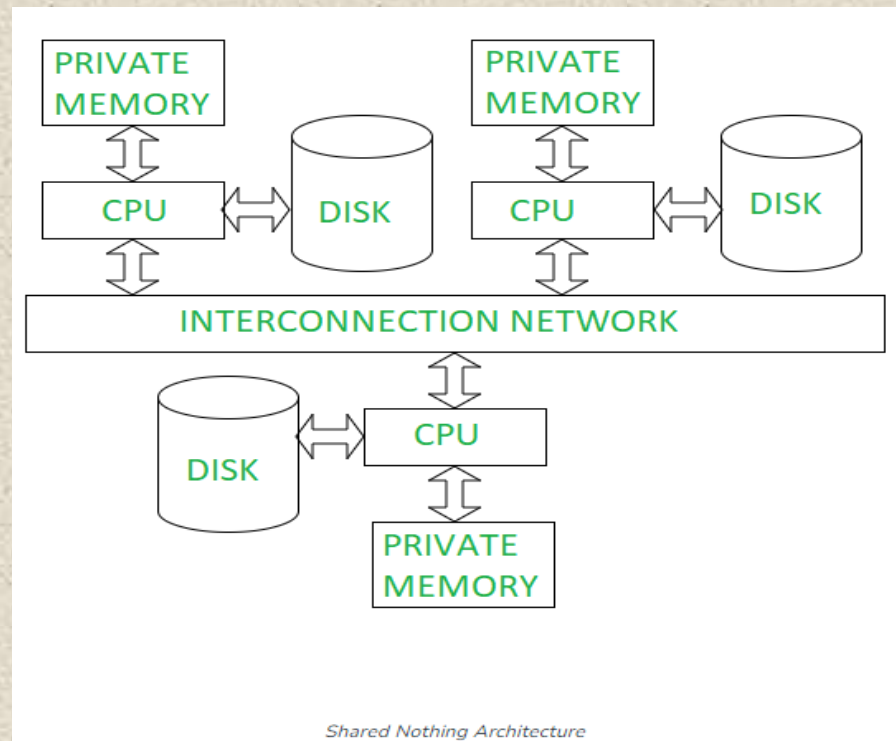
Parallel Databases

- ♦ **Shared Disk Architectures** : In Shared Disk Architecture, various CPUs are attached to an interconnection network. In this, each CPU has its own memory and all of them have access to the same disk. Also, note that here the memory is not shared among CPUs therefore each node has its own copy of the operating system and DBMS. Shared disk architecture is a loosely coupled architecture optimized for applications that are inherently centralized. They are also known as **clusters**.



Parallel Databases

- ♦ **Shared Nothing Architecture** : Shared Nothing Architecture is multiple processor architecture in which each processor has its own memory and disk storage. In this, multiple CPUs are attached to an interconnection network through a node. Also, note that no two CPUs can access the same disk area. In this architecture, no sharing of memory or disk resources is done. It is also known as Massively parallel processing (MPP).



Parallel Databases

- ♦ **Hierarchical Architecture** : This architecture is a combination of shared disk, shared memory and shared nothing architectures. This architecture is scalable due to availability of more memory and many processor. But is costly to other architecture.

Multimedia Databases

- ◆ Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images (such as pictures and drawings), video clips (such as movies, newsreels, and home videos), audio clips (such as songs, phone messages, and speeches), and documents (such as books and articles).
- ◆ A Multimedia Database (MMDB) hosts one or more multimedia data types (i.e. text, images, graphic objects, audio, video, animation sequences). These data types are broadly categorized into three classes:
 - Static media (time-independent: image and graphic object).
 - Dynamic media (time-dependent: audio, video and animation).
 - Dimensional media(3D game and computer aided drafting programs).

Multimedia Databases

- ◆ The contents of the multimedia database has additional information related to the primary multimedia data. To effectively manage and query a vast collection of multimedia data. These contents are –
- ◆ Media data
 - ◆ It is actual multimedia data or primary data stored in the multimedia database. It represents a multimedia object. It can be an image, audio, video, animation, graphic object, or text.
- ◆ Media format data
 - ◆ It is information related to the format of the multimedia data such as frame rates and encoding schemes.
- ◆ Media keyword data
 - ◆ It is also known as content descriptive data. This information pertains to the generation of multimedia data, such as date and time in the case of images and videos.
- ◆ Media feature data
 - ◆ This data is used to describe the characteristics of multimedia data, such as the color distribution.

Multimedia Databases

- ◆ The main types of database queries that are needed involve locating multimedia sources that contain certain objects of interest.
- ◆ For example, one may want to locate all video clips in a video database that include a certain person, say Michael Jackson. One may also want to retrieve video clips based on certain activities included in them, such as video clips where a soccer goal is scored by a certain player or team.
- ◆ The above types of queries are referred to as content-based retrieval, because the multimedia source is being retrieved based on its containing certain objects or activities.

Multimedia Databases

- ◆ A multimedia database must use some model to organize and index the multimedia sources based on their contents. Identifying the contents of multimedia sources is a difficult and time-consuming task. There are two main approaches. The **content based** is based on automatic analysis of the multimedia sources to identify certain mathematical characteristics of their contents. This approach uses different techniques depending on the type of multimedia source (image, video, audio, or text).
- ◆ The second approach depends on **manual identification** of the objects and activities of interest in each multimedia source and on using this information to index the sources. This approach can be applied to all multimedia sources, but it requires a manual preprocessing phase in which a person must scan each multimedia source to identify and catalog the objects and activities it contains so that they can be used to index the sources.

Multimedia Databases

- ◆ A typical **image database query** would be to find images in the database that are similar to a given image. The given image could be an isolated segment that contains, say, a pattern of interest, and the query is to locate other images that contain that same pattern.
- ◆ There are two main techniques for this type of search. The first approach uses a **distance function** to compare the given image with the stored images and their segments. If the distance value returned is small, the probability of a match is high. Indexes can be created to group stored images that are close in the distance metric so as to limit the search space.
- ◆ The second approach, called the **transformation approach**, measures image similarity by having a small number of transformations that can change one image's cells to match the other image. Transformations include rotations, translations, and scaling. Although the transformation approach is more general, it is also more time-consuming and difficult.

Multimedia Databases

- ◆ The multimedia database must support large objects, since multimedia data such as videos can occupy up to a few gigabytes of storage.
- ◆ Many database systems do not support objects larger than a few gigabytes.
- ◆ Larger objects could be split into smaller pieces and stored in the database.
- ◆ Alternatively, the multimedia object may be stored in a file system, but the database may contain a pointer to the object; the pointer would typically be a file name.
- ◆ The SQL/MED standard (Management of External Data) allows external data, such as files, to be treated as if they are part of the database. With SQL/MED, the object would appear to be part of the database, but can be stored externally.

Temporal Databases

- ◆ Temporal databases permit the database system to store a history of changes and allow users to query both current and past states of the database. Some temporal database models also allow users to store future expected information, such as planned schedules.
- ◆ Databases that store information about states of the real world across time are called temporal databases.

Temporal Databases

- ♦ A temporal relation is one where each tuple has an associated time when it is true; the time may be either valid time or transaction time.

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>from</i>	<i>to</i>
10101	Srinivasan	Comp. Sci.	61000	2007/1/1	2007/12/31
10101	Srinivasan	Comp. Sci.	65000	2008/1/1	2008/12/31
12121	Wu	Finance	82000	2005/1/1	2006/12/31
12121	Wu	Finance	87000	2007/1/1	2007/12/31
12121	Wu	Finance	90000	2008/1/1	2008/12/31
98345	Kim	Elec. Eng.	80000	2005/1/1	2008/12/31

Figure 25.1 A temporal *instructor* relation.

Spatial Databases

- ◆ Spatial databases incorporate functionality that provides support for databases that keep track of objects in a multidimensional space.
- ◆ For example, cartographic databases that store maps include two-dimensional spatial descriptions of their objects— from countries and states to rivers, cities, roads, seas, and so on.
- ◆ The systems that manage geographic data and related applications are known as Geographic Information Systems (GISs).

Spatial Databases

- ◆ In general, a spatial database stores objects that have spatial characteristics that describe them and that have spatial relationships among them.
- ◆ A spatial database is optimized to store and query data related to objects in space, including points, lines and polygons.
- ◆ Queries posed on these spatial data, where predicates for selection deal with spatial parameters, are called spatial queries. For example, “What are the names of all bookstores within five miles of the College of Computing building at Georgia Tech?” is a spatial query

Analytical Operations on Spatial Databases

- ◆ Measurement operations are used to measure some global properties of single objects (such as the area, the relative size of an object's parts, compactness, or symmetry) and to measure the relative position of different objects in terms of distance and direction.
- ◆ **Spatial analysis operations**, which often use statistical techniques, are used to uncover spatial relationships within and among mapped data layers. An example would be to create a map—known as a prediction map—that identifies the locations of likely customers for particular products based on the historical sales and demographic information.
- ◆ **Flow analysis operations** help in determining the shortest path between two points and also the connectivity among nodes or regions in a graph.
- ◆ **Location analysis** aims to find if the given set of points and lines lie within a given polygon (location).

Mobile Databases

- ◆ A Mobile database is a database that can be connected to a mobile computing device over a mobile network (or wireless network). Here the client and the server have wireless connections. In today's world, mobile computing is growing very rapidly, and it is huge potential in the field of the database. It will be applicable on different-different devices like android based mobile databases, iOS based mobile databases, etc. Common examples of databases are *SQLite*, *Couch base Lite*, *Object Box*, etc.
- ◆ Mobile Database typically involves three parties :
 - ◆ **Fixed Hosts** – It performs the transactions and data management functions with the help of database servers.
 - ◆ **Mobiles Units** – These are portable computers that move around a geographical region that includes the cellular network that these units use to communicate to base stations.
 - ◆ **Base Stations** – These are two-way radios installation in fixed locations, that pass communication with the mobile units to and from the fixed hosts.

Mobile Databases

- ♦ **Mobile Database Issues**
 - ♦ Data Management
 - ♦ Data Caching
 - ♦ Data Broadcast
 - ♦ Data Classification (Location Dependent/ Independent)
 - ♦ Transaction Management
 - ♦ Query processing
 - ♦ Transaction processing
 - ♦ Concurrency control
 - ♦ Database recovery

Web Databases

- ♦ **A Web database** is a database application designed to be managed and accessed through the Internet. Website operators can manage this collection of data and present analytical results based on the data in the Web database application.
- ♦ A web database is a system for storing and displaying information that is accessible from the Internet / web. The database might be used for any of a wide range of functions, such as a membership database, client list, or inventory database.
- ♦ Content management systems commonly use web databases to store information such as posts, usernames, and comments. Using a database allows the website to be updated easily and without the need to edit the HTML code for each individual page. Not only is this a much more efficient way of creating and updating a website, but it also makes the process more accessible to people who aren't fluent in the programming languages of the Internet.

Web Databases

- ♦ A web database is ideal for situations when the information should be shared, or when it must be accessed from various locations. It is especially beneficial when the system is to be shared between locations or different devices.
- ♦ Businesses both large and small can use Web databases to create website polls, feedback forms, client or customer and inventory lists. Personal Web database use can range from storing personal email accounts to a home inventory to personal website analytics. The Web database is entirely customizable to an individual's or business's needs.

Data Warehouse, Data Mining and Data Mart

- ◆ Will Cover in Next Unit 7.