Introduction to Database

Dr. Faiz Abdulkarim Bazanboor

Concept of Database System

Data redundancy and inconsistency

Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files).

For example:

The address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to inconsistency.

Data inconsistency

That is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

Integrity problems

The data values stored in the database must satisfy certain types of **consistency constraints**. In other word refers to invalid data for a particular data type.

For example:

The balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

For example:

Marks of student can't exceed 150, such kinds of integrity checks can be performed by database system and inform the user in case if he entering invalid data.

Security problems

Not every user of the database system should be able to access all the data.

For example:

In a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts.

But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult.

Database Management System

A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The DBMS is hence a *general-purpose software system* that facilitates the processes of *defining, constructing,* and *manipulating* databases for various applications. **Defining** a database involves specifying the data types, structures, and constraints for the data to be stored in the database. **Constructing** the database is the process of storing the data itself on some storage medium that is controlled by the DBMS. **Manipulating** a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data.

It is not necessary to use general-purpose DBMS software to implement a computerized database. We could write our own set of programs to create and maintain the database, in effect creating our own *special-purpose* DBMS software. In either case—whether we use a general-purpose DBMS or not—we usually have to employ a considerable amount of software to manipulate the database. We will call the database and DBMS software together a *database system***.** Figure illustrates these ideas.

Database System Applications

Databases are widely used. Here are some representative applications:

Banking: For customer information, accounts, and loans, and banking transactions.

Airlines: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner-terminals situated around the world accessed the central database system through phone lines and other data networks.

Universities: For student information, course registrations, and grades.

Credit card transactions: For purchases on credit cards and generation of monthly statements.

Telecommunication: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.

Sales: For customer, product, and purchase information.

View of Data

A database system is a collection of interrelated files and a set of programs that allow users to access and modify these files. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

- **Physical level**. The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.
- **Logical level**. The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
- **View level**. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database. Figure shows the relationship among the three levels of abstraction.

An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Most high-level programming languages support the notion of a record type. For example, in a C-like language, we may

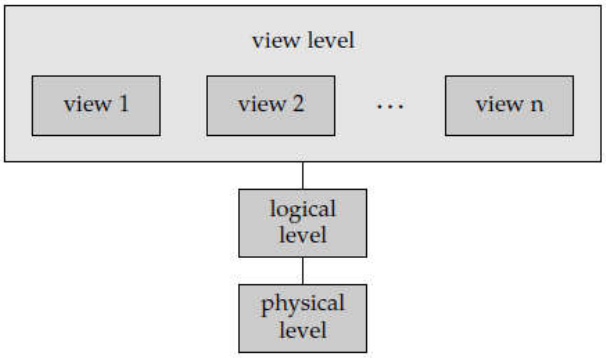
```
struct customer
  {
       int customer-id;
      char customer-name;
      char customer-street ;
      char customer-city;
      struct customer  *next;

  };
```

This code defines a new record type called *customer* with four fields, each field has a name and a type associated with it. A banking enterprise may have several such record types, including

- *Account*, with fields *account-number* and *balance*
- *Employee*, with fields' *employee-name* and *salary*

At the physical level, a *customer*, *account*, or *employee* record can be described as a block of consecutive storage locations (for example, words or bytes). The language



The three levels of data abstraction.

Compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

File Organization Technique

A file can be considered as a sequence of records stored in some local order on a storage device. Record has address. Record can be fixed length or variable length. A record may have a key by which it can be uniquely indentified.

Different operation can be perform on files such as

a) *Add* a record
b) *Delete* record
c) *Modify* a record
d) *Search* record

An application program may not require all the above operation to be performed on a file. Depending on the operation to be perform on the file; various organizations are possible such as

1. Sequential file organization
2. Indexed file organization
3. Binary Tree file Organization
4. Hash file organization

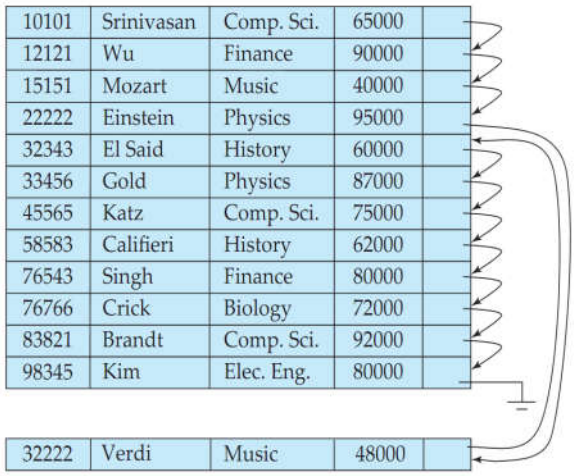
Sequential file Organization

Sequential file can be stored on sequential devices or random devices. Record can be only sequential accessed with sequential organization, random access is not possible. Sequential files can be stored in some order of key value. This organization is useful where a file has to be sequentially processed. Also sequential organization is used for tacking backups. A record from the file can be retrieved by one of the following ways.

Sequential File Organization

A sequential file is designed for efficient processing of records in sorted order based on some search key a search key is any attribute or set of attributes; it need not be the *primary key*, or even a super key. To permit fast retrieval of records in search-key order, we chain together records by pointers. The pointer in each record points to the next record in search-key order. Furthermore, to minimize the number of block accesses in sequential file processing, we store records physically in search-key order, or as close to search-key order as possible.

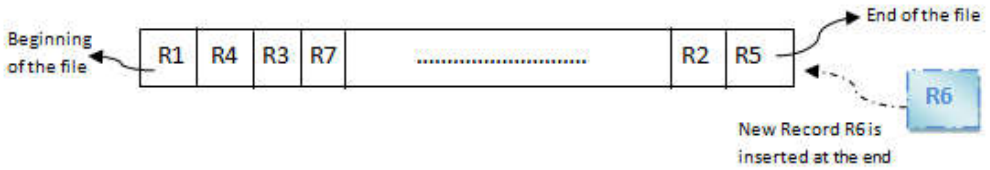
Figure shows a sequential file of *instructor* records. In the example, the records are stored in search-key order, using *ID* as the search key



10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
32222	Verdi	Music	48000	

It is difficult, however, to maintain physical sequential order as records are inserted and deleted, since it is costly to move many records as a result of a single insertion or deletion. We can manage deletion by using pointer chains, as we saw previously. For insertion, we apply the following rules:

1. ***Scanning the file record by record***: this is the slowest method and generally never used.



2. ***Block search***: logical records are packed into physical record. Each such physical record is termed as a block which contains more than one logical record. To search a record a complete block is read and search is done.

Indexed organization

This organization is useful for both sequential and random accessing of data from file. Index file can be stored only on random device. Indexed organization has two files.

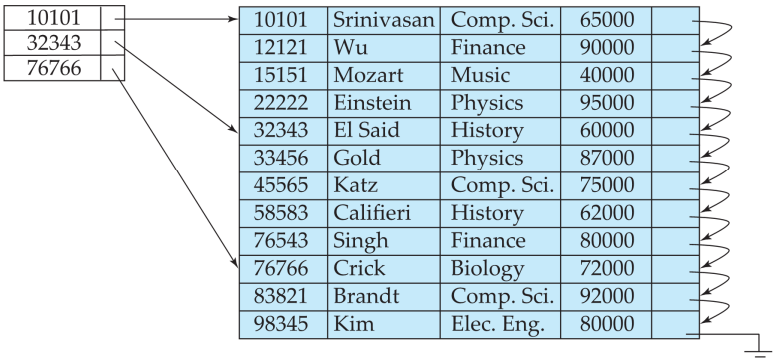
1. Data file
2. Index file

Data file: a *logical records* are stored in data file. To make efficient use of storage devices generally several logical records are blocked in one record(e.g. sector) this is called as blocking, if blocking is not used then space is wasted as *inter record gap* has to be present between records to separate them. If the records are of variable length then they are made of fixed length by padding them with spaces, any read or write operation on this file is done by reading or writing a block.

***Index File*:** this file contains *key value* for record from data file and the relative address of the record in the data file. A record from a file is searched by getting its address in data file by searching in the *index file* and then accessing the *data file*. Two types of *index files* are present

a. Sparse Index
b. Dense Index

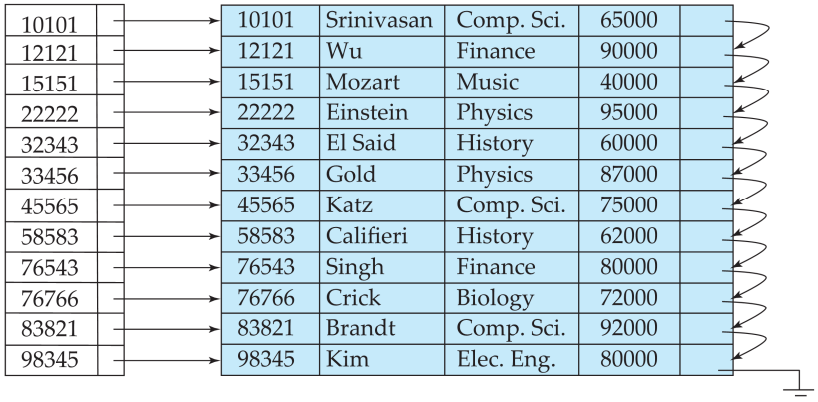
Sparse Index: A sparse index does not contain key value for all the records present in data file, instead a key from each block and the relative address of each block is stored in the index file.



e.g.: ***dictionary***: each block can be consider as page, word in each pages are ordered in alphabetic order, the index contains only the last word (or the first) from each page and the page number, thus to search a word we first search for page number (relative block address) where the word can be found in the index file, then we read the page number and search for the word.

Search inside a block can be either *sequential* or *binary*, sparse indexes are used for primary keys.

Dense Index: a dense index contains key values and relative address of record in data file for every type of record. A dense index is always larger than sparse index, therefore searching is also slower, usually dense indexes are created for secondary indices. Records are stored in file ordered on primary key, therefore the records may not be ordered on other key (secondary index)



E.g. consider a file of student with roll number, passport number and other information. If this file is sorted on roll number, then the passport number need not be ordered, therefore the index for passport number cannot be *sparse* but *dense*.

Generally when a index file is created complete space in a block is stored utilized so that insertion can be made later.

3. ***Binary Search Method***: Many DBMS use a data structure called a *tree* to hold data or index, a tree consist of hierarchy of nodes. Each in the tree, except the *root* node, has one parent node and zero or more child nodes. A *root* node has no parent. A node that does not have any children is called *leaf* node. The depth of a *tree* is the maximum number of levels between the *root* node and a *leaf* node in the *tree*. This method is applicable only if the file is stored on a random device and ordered on the key value. Search is by making use of binary search method.

E.g.

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield