Uttar Pradesh
Rajarshi Tandon Open University

BCA-1.5
Introduction to Database Management System

Block

1

INTRODUCTORY CONCEPTS OF DATA BASE MANAGEMENT SYSTEMS

Expert Advisors

Prof. P.S. Grover
Professor of Computer Sciences
University of Delhi
Delhi

Brig. V.M. Sundaram
Co-ordinator
DoE-ACC Centre
New Delhi

Prof. Karmeshu
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Prof. L.M. Patnaik
Indian Institute of Science
Bangalore

Prof. M.M. Pant
Director
School of Computer and
Information Sciences
IGNOU
New Delhi

Dr. S.C. Mehta
Sr. Director
Manpower Development Division
Department of Electronics
Govt. of India
New Delhi

Dr. G. Haider
Director
Information Technology Centre
TCIL, Delhi

Prof. H.M. Gupta
Department of Electrical Engineering
Indian Institute of Technology
Delhi

Prof. S. Sadagopan
Department of Industrial Engineering
Indian Institute of Technology
Kanpur

Prof. R.G. Gupta
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Prof. S.K. Wason
Professor of Computer
Science
Jamia Millia
Delhi

Dr. Sugata Mitra
Principal Scientist
National Institute of
Information Technology
New Delhi

Prof. Sudhir Kaicker
Director
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Faculty of the School

Prof. M.M. Pant
Director

Mr. Akshay Kumar
Lecturer

Mr. Shashi Bhushan
Lecturer

Course Preparation Team

Prof. M.M. Pant
Director SOCIS
IGNOU

Mr. Millind Mahajani
Manager
Information Services
Time of India Group
New Delhi

Dr. N. Parimala
Birla Institute of Technology
and Science, Pilani

Utpal Bhattacharya
NIIT
New Delhi

Mr. Shashi Bhushan
Lecturer, IGNOU

Block Writer
Mr. Shashi Bhushan
Lecturer, IGNOU

Course Coordinator
Mr. Shashi Bhushan
Lecturer, IGNOU

Print Production : Sh. Jitender Sethi, APO, MPDD

March, 2003 (Reprint)

ISBN-81-7263-861-2

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi. New Delhi- 110 068.

Reproduced and reprinted with the permission of Indira Gandhi National Open University by Sri D.P.Tripathi, Registrar, U.P.R.T.Open University, Allahabad (December, 2015)
Reprinted by : Nitin Printers, 1 Old Katra, Manmohan Park, Allahabad.

BCA-1.5/2

COURSE INTRODUCTION

Database management systems have become an essential part of a computer science education. This course provides a clear description of the concepts underlying different database models. It introduces issues related to implementation of conventional database models. It also describes emerging trends in DBMS.

The important topics covered in this course are as follows:

- Entity-relationship Model
- Hierarchical
- Network Model
- Relational Model
- File organization of Conventional DBMS
- Evaluation and Administration of DBMS
- Normalisations
- SQL
- Distributed DBMS
- Object-oriented DBMS
- Client/Server DBMS
- Knowledge DBMS

This course contains 3 blocks.

BLOCK INTRODUCTION

Being an introductory block of this course, it describes basic concepts related to all conventional DBMS models. It introduces E-R model which is used for logical database design. It also takes up the implementation of all the conventional database models. The focal point of this block is the file organisation for conventional DBMS. It also considers evaluation and administration of DBMS. Finally it is presented with a case Study.

UNIT 1 BASIC CONCEPTS

Structure

1.0 INTRODUCTION

A database is a collection of related information stored so that it is available to many users for different purposes. The content of a database is obtained by combining data from all the different sources in an organisation. So that data are available to all users and redundant data can be eliminated or atleast minimised. A computer database gives us some electronic filing system which has a large number of ways of cross-referencing and this allows the user many different ways in which to reorganise and retrieve data. A database can handle business inventory, accounting and filing and use the information in its files to prepare summaries, estimates and other reports. There can be a database which stores new paper articles, magazines, books and comics. There is already a well-defined market for specific information for highly selected group of users on almost all subjects. MEDLINE is a well-known database service providing medical information for doctors and similarly WESTLAW is a computer based information service catering to the requirements of lawyers. The key to making all this possible is the manner in which the information in the database is managed. The management of data in a database system is done by means of a general purpose software package called a database management system. The database management system is the major software component of a database system. Some commercially available DBMS are INGRES, ORACLE, Sybase. **A database management system, therefore, is a combination of hardware and software that can be used to set up and monitor a database, and can manage the updating and retrieval of database that has been stored in it.** Most database management systems have the following facilities/capabilities:

(a) Creating of a file, addition to data, deletion of data, modification of data; creation, addition and deletion of entire files.

(b) Retrieving data collectively or selectively.

(c) The data stored can be sorted or indexed at the user's discretion and direction.

(d) Various reports can be produced from the system. These may be either standardised report or that may be specifically generated according to specific user definition.

(e) Mathematical functions can be performed and the data stored in the database can be manipulated with these functions to perform the desired calculations.

(f) To maintain data integrity and database use.

The DBMS interprets and processes users' requests to retrieve information from a database. The following figure shows that a DBMS serves as an interface in several forms. They may be keyed directly from a terminal, or coded as high-level language programs to be submitted for interactive or batch processing. In most cases, a query request will have to penetrate several layers of software in the DBMS and operating system before the physical database can be accessed.

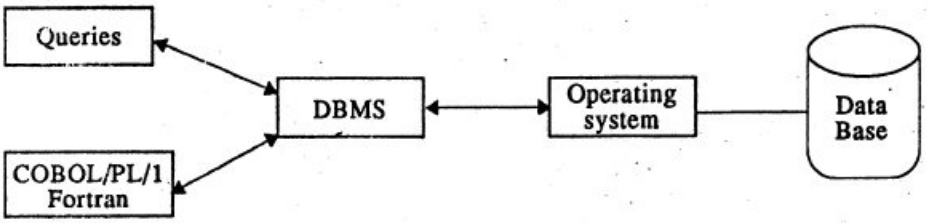


Figure 1 : The DBMS as an interface between physical Database and Users' requests

The DBMS responds to a query by invoking the appropriate subprograms, each of which performs its special function to interpret the query, or to locate the desired data in the database and present it in the desired order. Thus, the DBMS shields database users from the tedious programming they would have to do to organise data for storage, or to gain access to it once it was stored.

As already mentioned, a database consists of a group of related files of different record types, and the DBMS allows users to access data anywhere in the database without the knowledge of how data are actually organised on the storage device.

The role of the DBMS as an intermediary between the users and the database is very much like the function of a salesperson in a consumers' distributor system. A consumer specifies desired items by filling out an order form which is submitted to a salesperson at the counter. The salesperson presents the specified items to consumer after they have been retrieved from the storage room. Consumers who place orders have no idea of where and how the items are stored; they simply select the desired items from an alphabetical list in a catalogue. However, the logical order of goods in the catalogue bears no relationship to the actual physical arrangement of the inventory in the storage room. Similarly, the database user needs to know only what data he or she requires; the DBMS will take care of retrieving it.

In this unit we will introduce basic concepts of DBMS.

1.1 OBJECTIVES

After going through this unit, you should be able to

- appreciate the limitations of the traditional approacn to application system development;
- give reasons why the database approach is now being increasingly adopted;
- discuss different views of data;
- list the components of a database management system;
- enumerate the feature/capabilities of a database management system; and
- list several advantages and disadvantages of DBMS.

1.2 TRADITIONAL FILE ORIENTED APPROACH

The traditional file-oriented approach to information processing has for each application a separate master file and its own set of personal files. You have seen examples of these in the earlier course on COBOL where various examples such as of payroll, inventory and financial accounting have been described at various level. An organisation needs flow of information across these applications also and this requires sharing of data, which is significantly lacking in the traditional approach. One major limitations of such a file-based approach is that the programs become dependent on the files and the files become dependent upon the programs.

Although such file-based approaches which came into being with the first commercial applications of computers did provide an increased efficiency in the data processing compared to earlier manual paper record-based systems as the demand for efficiency and speed increased, the computer-based simple file-oriented approach to information processing started suffering from the following significant disadvantages :

(1) **Data Redundancy** : The same piece of information may be stored in two or more files. For example, the particulars of an individual who may be a customer or an employee may be stored in two or more files. Some of these information may be changing, such as the address, the pay drawn, etc. It is therefore quite possible that while the address in the master file for one application has been updated the address in the master file for another application may have not been. It may not also be easy for the computer-based system to even find out as to in how many files the repeating items such as the address is occurring. The solution therefore is to avoid this data redundancy and the keeping of multiple copies of the same information and replace it by a system where the address is stored at just one place physically, and is accessible to all applications from this itself.

(2) **Program/Data Dependency** : In the traditional approach if a data field is to be added to a master file, all such programs that access the master file would have to be changed to allow for this new field which would have been added to the master record.

(3) **Lack of Flexibility** : In view of the strong coupling between the program and the data, most information retrieval possibilities would be limited to well-anticipated and pre-determined requests for data, the system would normally be capable of producing scheduled records and queries which it has been programmed to create. In the fast moving and competent business environment of today, apart from such regularly scheduled records, there is a need for responding to un-anticipatory queries and some kind of investigative analysis which could not have been envisaged professionally. These disadvantages of file based system motivates a database approach, which will be taken at the next section.

1.3 MOTIVATION FOR DATABASE APPROACH

Having pointed out some difficulties that arise in a straight forward file-oriented approach towards information system development, it is useful to see how the problems stated above can be mitigated by using the database approach.

The preceding discussion may have led you to believe that the traditional file oriented approach to data processing was entirely wrong and that all new and the correct modern approach to data processing should only be through databases. This is not entirely true. With the large scale availability of personal computers and greater with power being available on the desktops, simple file management systems such as the kind briefly referred to in section 1.2 may be quite appropriate. In fact only large scale organisations involved in manufacturing and business or public utility services such as hospitals, hotels, government departments, etc. would be in a position to rely into the database approach. Some of the reasons why every organisation may not be able to successfully adopt the database approach are :

(1) The work in the organisation may not require significant sharing of data or complex access. In other words the data and the way it is used in the functioning of the organisation is not appropriate to database processing.

(2) Apart from needing a more powerful hardware platform, the software for database management systems are also quite expensive. This means that a significant extra cost has to be incurred by an organisation if it wants to adopt this approach.

(3) The advantages gained by the possibility of sharing of the data with others, also carries with it the risk of the data being unauthorisedly accessed. This may range from violation of office procedures to violation of privacy rights of information to down right thefts. The organisations, therefore, have to be ready to cope with additional managerial problems.

(4) A database management processing system is complex and it could lead to a more inefficient system than the equivalent file-based one.

(5) The staff available for the organisation may not be experienced enough to cope with. The training of personnel in the management in use of a database takes time, is expensive and requires special attention.

(6) The use of the database and its possibility of being shared will, therefore affect many departments within the organisation. If the integrity of the data is not maintained, it is possible that one relevant piece of data could have been used by many programs in different applications by different users without they are being aware of it. The impact of this, therefore may be very widespread. Since data can be input from a variety sources, the control over the quality of data become very difficult to implement.

However, for most large organisations, the difficulties in moving over to a database approach are still worth getting over in view of the advantages that are gained, namely, avoidance of data duplication, sharing of data by different programs, greater flexibility and data independence. The advantages and disadvantages of DBMS will be discussed in detail in section 1.9.

1.4 DATABASE BASICS

You have seen in the previous section the purposes for which a DBMS approach is preferred over the conventional approach. Since the DBMS of an organisation will in some sense reflect the nature of activities in the organisation, some familiarity with the basic concepts, principles and terms used in the field are important.

The previous courses on Computer fundamentals, software and programming languages have already given you an awareness of the essential ingredients of computer-based information systems. This section concentrates on those matters which are relevant in the context of a database approach.

Data-items: The term data item is the word for what has traditionally been called the field in data processing and is the smallest unit of data that has meaning to its users. The phrase data element or elementary item is also sometimes used. Although the data item may be treated as a molecule of the database, data items are grouped together to form aggregates described by various names. For example, the data recured is used to refer to a group of data items and a program usually reads or writes the whole records. The data items could occasionally be further broken down into what may be called an automatic level for processing purposes. For example, a data item such as a date would be a composite value comprising the day, date and year. But for doing date arithmetic these may have to be first separated before the calculations are performed. Similarly an identification number may be a data item but it may contain further information embedded in it. For example, the IGNOU uses a 9 digit enrollment number. The first 2 digits of these number reflect the year of admission, the next 2 digits refer to the Regional Centre where the student has first opted for admission, the next 4 digits are simple sequence numbers and the last digit is a check digit. For purposes of processing, it may sometimes be necessary to split the data item.

Standardisation of data items can become a fairly serious problem in a large corporate with several divisions or plans. Each such unit tends to have its own ways of referring to the data items related to personal accounting, engineering, sales, production, purchase activities, etc. It would be extremely desirable if at the stage of adopting the database approach a commitment from the top management is acquired for prospective standardisation across the enterprise for schemas of the data items.

Entities and Attributes: The real world which is being attempted to market on to the database would consist of occasionally a tangible object such as an employee, a component in an inventory or a space or it may be intangible such as a event, a job description, identification numbers or a abstract construct. All such items about which relevant information is stored in the database are called **Entities.** The qualities of the entity which we

store as information are called the **attributes**. An attribute may be expressed as a number or as a text. It may even be a scanned picture, a sound sequence, a moving picture which is now possible in some visual and multi-media databases.

Data processing normally concerns itself with a collection of similar entities and records information about the same attributes of each of them. In the traditional approach, a programmer usually maintains a record about each entity and a data item in each record relates to each attribute. Similar records are grouped into files and such a 2-dimensional array is sometimes referred to as a flat file.

Logical and Physical Data : One of the key features of the database approach is to bring about a distinction between the logical and the physical structures of the data. The term logical structure refers to the way the programmers see it and the physical structure refers to the way data are actually recorded on the storage medium. Even in the early stages of records stored on tape, the length of the inter-record tape requires that many logical records be grouped into one physical record to several storage places on tape. It was the software which separated them when used in an application program and combined them again before writing back on tape. In today's system the complexities are even greater and as will be seen when one is referring to distributed databases that some records may physically be located at significantly remote places.

Schema and Subschema: Having seen that the database does not focus on the logical organisation and decouples it from the physical representation of data, it is useful to have a term to describe the logical database description. A schema is a logical database description and is drawn as a chart of the types of data that are used. It gives the names of the entities and attributes and specify the relationships between them. It is a framework into which the values of the data item can be fitted. Like an information display system such as that giving arrival and departure time at airports and railway stations, the schema will remain the same though the values displayed in the system will change from time to time. The relationships that has specified between the different entities occurring in the schema may be a one to one, one to many, many to many or conditional.

The term schema is used to mean an overall chart of all the data item types and record-types stored in a database. The term subschema refers to the same view but for the data-item types and record types which are used in a particular application or by a particular user. Therefore, many different subschemas can be derived from one schema. A simple analysis to distinguish between the schema and the sub schema may be that if the schema represented a road map of Delhi showing major historical sites, educational institutions, railway stations, roadway stations and airports, a subschema could be a similar map showing one route each from the railway station or the airport to the IGNOU campus at Maidan Garhi.

Data Dictionary : It holds detailed information about the different structures and data types : the details of the logical structure that are mapped into the different structure, details of relationship between data items, details of all users privileges and access rights, performance of resource with details.

The last two items discussed in this section will be further elaborated in the subsequent sections.

1.5 THREE VIEWS OF DATA

DBMS is a collection of interrelated files and a set of programs that allow several 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 is stored and maintained. However, in order for the system to be usable, data must be retrieved efficiently.

The concern for efficiently leads to the design of complex data structure for the representation of data in the database. However since database systems are often used by non computer professionals, this complexity must be hidden from database system users. This is done by defining levels of abstract as which the database may be viewed, there are logical view or external, conceptual view and internal view or physical view.

External view : This is the highest level of abstraction as seen by a user. This level of abstraction describes only the part of entire database.

Conceptual view : This is the next higher level of abstraction which is the sum total of user's views. This level describes what data are actually stored in the database. This level contains information about entire database in terms of a small number of relatively simple structure.

Internal level : This is the lowest level of abstraction at which one describes how the data are physically stored. The interrelationship of any three levels of abstraction is illustrated in figure 2.

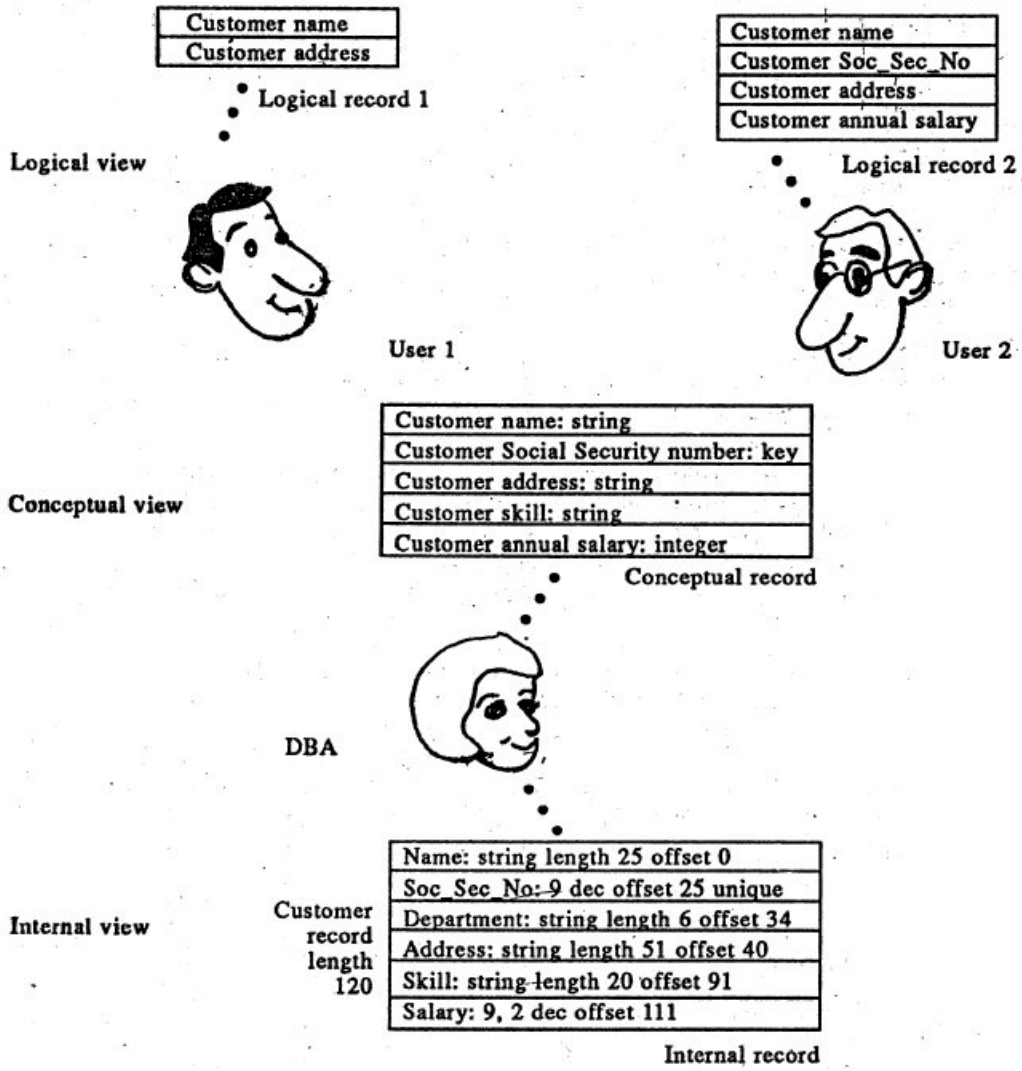


Figure 2: The three views of data

To illustrate the distinction among different views of data, it can be compared with the concept of data types in programming languages. Most high level programming language such as C, Pascal, COBOL, etc. support the notion of a record or structure type. For example in the 'C' language we declare structure (record) as follows:

```
struct Customer {
char name [15];
char address [30];
        }
```

This defines a new record called customer with 2 fields. Each field has a name and data type associated with it.

In a banking organisation, we may have several such record types, including among others :

- **account** with fields number and balance
- **employee** with fields name and salary

At the internal level, a customer, account or employee can be described as a sequence of consecutive bytes. At the conceptual level each such record is described by a type definition, illustrated above and also the interrelation among these record types is defined. Finally at the external level, we define several views of the database. For example, for preparing the payroll checks of bank employees only information about them is required, one does not need to access information about customer accounts. Similarly, tellers can access only account information. They cannot access information concerning salaries of employees.

1.6 THE THREE LEVEL ARCHITECTURE OF DBMS

In the previous section we defined three levels of abstraction at which the database may be viewed. A database management system that provides these three levels of data is said to follow three-level architecture as shown in figure 3. These three levels are the external level, the conceptual level and the internal level.

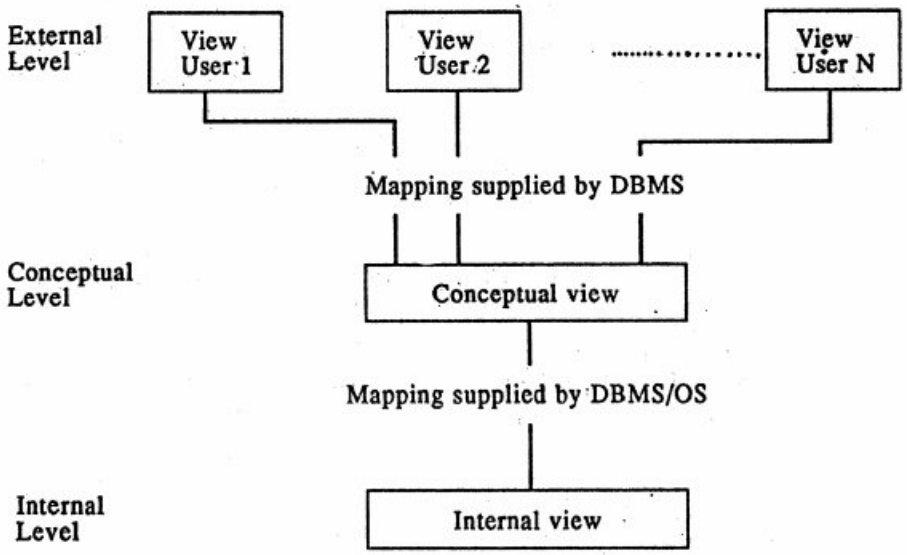


Figure 3: The three level architecture for a DBMS

The view at each of these levels is described by a schema. A schema as mentioned earlier is an outline or a plan that describes the records and relationships existing in the view. The schema also describes the way in which entities at one level of abstraction can be mapped to the next level. The overall design of the database is called the database schema. A database schema includes such information as:

- characteristics of data items such as entities and attributes
- logical structure and relationship among those data items
- format for storage representation
- integrity parameters such as physically authorisation and backup politics.

The concept of a database schema corresponds to programming language notion of **type definition.** A variable of a given type has a particular value at a given instant in time. The concept of the value of a variable in Programming languages corresponds to the concept of an instance of a database schema.

Since each view is defined by a schema, there exists several schema in the database and these exists several schema in the database and these schema are partitioned following three levels of data abstraction or views. At the lower level we have the physical schema, at the intermediate level we have the conceptual schema, while at the higher level we have a subschema. In general, database system supports one physical schema, one conceptual schema and several subschema.

1.6.1 External Level or Subschema

The external level is at the highest level of database abstraction where only those portions of the database of concern to a user or application program are included. Any number of user views (some of which may be identical) may exist for a given global or conceptual view.

Each external view is described by means of a schema called an external schema or subschema. The external schema consists of the definition of the logical records and the relationships in the external view. The external schema also contains the method of deriving the objects in the external view from the objects in the conceptual view. The objects includes entities, attributes, and relationships.

1.6.2 Conceptual Level or Conceptual Schema

At this level of database abstraction all the database entities and the relationships among them are included. One conceptual view represents the entire database. This conceptual view is defined by the conceptual schema. It describes all the records and relationships included in the conceptual view and, therefore, in the database. There is only one conceptual schema per database. This schema also contains the method of deriving the objects in the conceptual view from the objects in the internal view.

The description of data at this level is in a format independent of its physical representation. It also includes features that specify the checks to retain data consistency and integrity.

1.6.3 Internal Level or Physical Schema

We find this view at the lowest level of abstraction, closest to the physical storage method used. It indicates how the data will be stored and describes the data structures and access methods to be used by the database. The internal view is expressed by the internal schema, which contains the definition of the stored record, the method of representing the data fields, and the access aids used.

1.6.4 Mapping Between different Levels

Two mappings are required in a database system with three different views as shown in figure 3. A mapping between the external and conceptual level gives the correspondence among the records and the relationships of the external and conceptual levels.

Similarly, there is a mapping from a conceptual record to an internal one. An internal record is a record at the internal level, not necessarily a stored record on a physical storage device. The internal record of figure 3 may be split up into two or more physical records. The physical database is the data that is stored on secondary storage devices. It is made up of records with certain data structures and organised in files. Consequently, there is an additional mapping from the internal record to one or more stored records on secondary storage devices.

1.7 DATABASE MANAGEMENT SYSTEM FACILITIES

Two main types of facilities are supported by the DBMS:

- the data definition facility or data definition language (DDL)
- the data manipulation facility or data manipulation language (DML)

1.7.1 Data Definition Language

Database management systems provide a facility known as the **data definition language (DDL)**, which can be used to define the conceptual schema and also give some details about how to implement this schema in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes as well as the relationships among he entity sets. The definition also includes any constraints that have to be maintained, ncluding the constraints on the value that can be assigned to a given attribute and the constraints on the values assigned to different attributes in the same or different records. These definitions, which can be described as metadata about the data in the database, are expressed in the DDL of the DBMS and maintained in a compiled form (usually as a set of tables). The compiled form of the definitions is known as a **data dictionary, directory,** or

system catalogue The data dictionary contains information on the data stored in the database and is consulted by the DBMS before any data manipulation operation.

The database management system maintains the information on the file structure, the method used to efficiently access the relevant data (i.e., the access method). It also provides a method whereby the application programs indicate their data requirements. The application program could use a subset of the conceptual data definition language or a separate language. The database system also contains mapping functions that allow it to interpret the stored data for the application program. (Thus, the stored data is transformed into a form compatible with the application program.)

The internal schema is specified in a somewhat similar data definition language called data storage definition language. The definition of the internal view is compiled and maintained by the DBMS. The compiled internal schema specifies the implementation details of the internal database, including the access methods employed. This information is handled by the DBMS; the user need not be aware of these details.

1.7.2 Data Manipulation Language

DML is a language that enables users to access or manipulate as organised by the appropriate data model. Data manipulation involves retrieval of data from the database, insertion of new data into the database, and deletion or modification of existing data. The first of these data manipulation operations is called a **query**. A query is a statement in the DML that requests the retrieval of data from the database. The subset of the DML used to pose a query is known as a **query language**; however, we use the terms DML and query language synonymously.

The DML provides commands to select and retrieve data from the database. Commands are also provided to insert, update, and delete records. They could be used in an interactive mode or embedded in conventional programming languages such as Assembler, COBOL, FORTRAN, Pascal, or PL/I. The data manipulation functions provided by the DBMS can be invoked in application programs directly by procedure calls or by preprocessor statements. The latter would be replaced by appropriate procedure calls by either a preprocessor or the compiler.

There are basically two types of DML:

- Procedural : which requires a user to specify what data is needed and how to get it
- Nonprocedural : which requires a user to specify what data is needed without specifying how to get it

Data definition of the external view in most current DBMSs is done outside the application program or interactive session. Data manipulation is done by procedure calls to subroutines provided by a DBMS or via preprocessor statements. In an integrated environment, data definition and manipulation are achieved using a uniform set of constructs that forms part of the user's programming environment.

1.8 ELEMENTS OF A DATABASE MANAGEMENT SYSTEM

The major components of a DBMS are explained below :

1.8.1 DML Precompiler

It converts DML statement embedded in an application program to normal procedure calls in the host language. The precompiler must interact with the query processor in order to generate the appropriate code.

1.8.2 DDL Compiler

The DDL compiler converts the data definition statements into a set of tables. These tables contain information concerning the database and are in a form that can be used by other components of the DBMS.

1.8.3 File Manager

File manager manages the allocation of space on disk storage and the data structure used to represent information stored on disk. The file manager can be implemented using an interface to the existing file subsystem provided by the operating system of the host computer or it can include a file subsystem written especially for the DBMS.

1.8.4 Database Manager

Databases typically require a large amount of storage space. Corporate databases are usually measured in terms of gigabytes of data. Since the main memory of computers cannot store this information, it is stored on disks. Data is moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of control processing unit of computers, it is imperative that database system structure data so as to minimise the need to move data between disk and main memory. A database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. It is responsible for interfacing with file system. One of the function of database manager is to convert user's queries coming directly via the query processor or indirectly via an application program from the user's logical view to the physical file system. In addition, the tasks of enforcing constraints to maintain the consistency and integrity of the data as well as its security are also performed by database manager. Synchronising the simultaneous operations performed by concurrent users is under the control of the data manager. It also performs backup and recovery operations. Let us summarise now the important responsibilities of Database manager:

- **Interaction with file manager** : The raw data is stored on the disk using the file system which is usually provided by a conventional operating system. The database manager translates the various DML statements into low-level file system commands. Thus the database manager is responsible for the actual storing, retrieving and updating of data in the database.
- **Integrity enforcement** : The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (for example Rs. 200). Similarly the number of holidays per year an employee may be having should not exceed 25 days. These constraints must specified explicitly by the DBA. If such constraints are specified, then the database manager can check whether updates to the database result in the violation of any of these constraints and if so appropriate action may be imposed.
- **Security enforcement** : As discussed above, not every user of the database needs to have access to the entire content of the database. It is the job of the database manager to enforce these security requirements.
- **Backup and recovery** : A computer system like any other mechanical or electrical device, is subject to failure. There are a variety of causes of such failure, including disk crash, power failure and s/w errors. In each of these cases, information concerning the database is lost. It is the responsibility of database manager to detect such failures and restore the database to a state that existed prior to the occurrence of the failure. This is usually accomplished through the backup and recovery procedures.
- **Concurrency control** : When several users update the database concurrently, the consistency of data may no longer be preserved. It is necessary for the system to control the interaction among the concurrent users, and achieving such a control is one of the responsibilities of database manager.

1.8.5 Query Processor

The database user retrieves data by formulating a query in the data manipulation language provided with the database. The query processor is used to interpret the online user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution. The query processor uses the data dictionary to find the structure of the relevant portion of the database and uses this information in modifying the query and preparing an optimal plan to access the database.

1.8.6 Database Administrator

One of the main reasons for having database management system is to have control of both data an programs accessing that data. The person having such control over the system is called the database administrator (DBA). The DBA administers the three levels of the database and, in consultation with the overall user community, sets up the definition of the global view or conceptual level of the database. The DBA further specifies the external view of the various users and applications and is responsible for the definition and implementation of the internal level, including the storage structure and access methods to be used for the optimum performance of the DBMS. Changes to any of the three levels necessitated by changes or growth in the organisation and/or emerging technology are under the control of the DBA. Mappings between the internal and the conceptual levels, as well as between the internal and the conceptual levels, as well as between the conceptual and external levels, are also defined by the DBA. Ensuring that appropriate measures are in place to maintain the integrity of the database and that the database is not accessible to unauthorised users is another responsibility. The DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database. This profile describes the permissible activities of a user on that portion of the database accessible to the user via one or more user views. The user profile can be used by the database system to verify that a particular user can perform a given operation on the database.

The DBA is also responsible for defining procedures to recover the database from failures due to human, natural, or hardware causes with minimal loss of data. This recovery procedure should enable the organisation to continue to function and the intact portion of the database should continue to be available.

Let us summarise the functions of DBA are :

- **Schema definition** : The creation of the original database schema. This is accomplished by writing a set of definition which are translated by the DDL compiler to a set of tables that are permanently stored in the data dictionary.
- **Storage Structure and access method definition** : The creation of appropriate storage structure and access method. This is accomplished by writing a set of definitions which are translated by the data storage and definition language compiler.
- **Schema and Physical organisation modification** : Either the modification of the database schema or the description of the physical storage organisation. These changes, although relatively rare, are accomplished by writing a set of definition which are used by either the DDL compiler or the data storage and definition language compiler to generate modification to the appropriate internal system tables (for example the data dictionary).
- **Granting of authorisation for data access** : The granting of different types of authorisation for data access to the various users of the database.
- **Integrity constraint specification** : These constraints are kept in a special system structure that is consulted by the database manager whenever one of the valuable tools that the DBA uses to carry out data administration in data dictionary.

1.8.7 Data Dictionary

It is seen that when a program become somewhat large in size, keeping a track of all the available names that are used and the purpose for which they were used becomes more and more difficult. Of course it is possible for a programmer who has coined the available names to bear them in mind, but should the same author come back to his program after a significant time or should another programmer have to modify the program, it would be found that it is extremely difficult to make a reliable account of for what purpose the data files were used.

The problem becomes even more difficult when the number of data types that an organisation has in its database increased. It has also now perceived that the data of an organisation is a valuable corporate resource and therefore some kind of an inventory and catalogue of it must be maintained so as to assist in both the utilisation and management of the resource.

It is for this purpose that a data dictionary or dictionary/directory is emerging as a major tool. An inventory provides definitions of things. A directory tells you where to find them. A data dictionary/directory contains information (or data) about the data.

A comprehensive data dictionary would provide the definition of data item, how they fit into the data structure and how they relate to other entities in the database. With the comprehensive base of information the data dictionary can serve several useful purposes connecting across the whole spectrum of planning, determining information requirement, designing and implementation operation and revision. There is now a greater emphasis on having an integrated system in which the data dictionary is part of the DBMS. In such a case the data dictionary would store the information concerning the external, conceptual and internal levels of the databases. It would combine the source of each data field value that is from where the authenticate value is obtained. The frequency of its use and audit trail regarding the updates including user identification with the time of each update.

The greater acceptance and proliferation of relational databases have encouraged the evolution of data dictionary to "information resource dictionary system" (IRDS) for such facilities, as is the suggestion from ANSI (American National Standards Institute).

The DBA uses the data dictionary in every phase of a database life cycle, starting from the embryonic data gathering phase to the design, implementation and maintenance phases. Documentation provided by a data dictionary is as valuable to end users and managers as it provided by a data dictionary is as valuable to end users and managers as it is essential to the programmers. Users can plan their applications with the database only if they know exactly what is stored in it. For example, the description of a data item in a data dictionary may include its origin and other text description in plain English, in addition to its data format. Thus users and managers will be able to see exactly what is available in the database. You could consider a data dictionary to be a road map which guides users to access information within a large database.

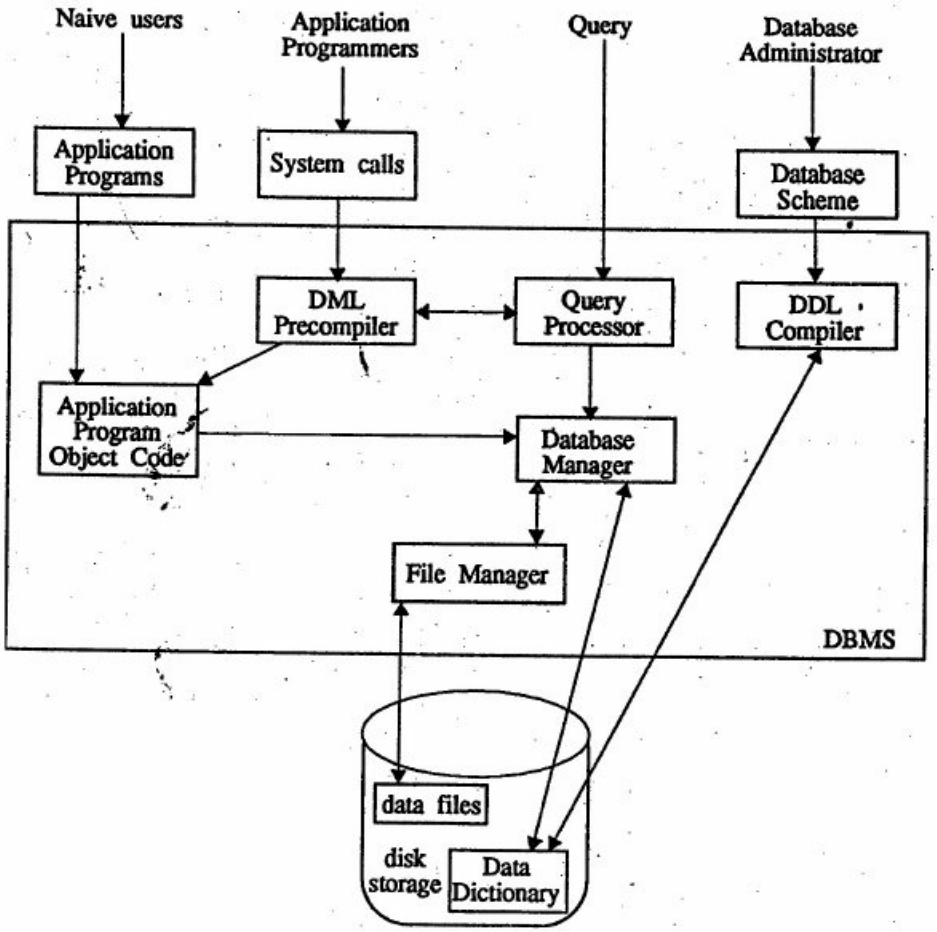


Figure 4: DBMS Structure

An ideal data dictionary should include everything a DBA wants to know about the database.

(1) external, conceptual and internal database descriptions

(2) descriptions of entities (record types), attributes (fields), as well as cross-references, origin and meaning of data elements

(3) synonyms, authorisation and security codes

(4) which external schemas are used by which programs, who the users are, and what their authorisations are.

A data dictionary is implemented as a database so that users can query its content by either interactive or batch processing. Whether or not the cost of acquiring a data dictionary system is justifiable depends on the size and complexity of the information system. The cost effectiveness of a data dictionary increases as the complexity of an information system increases. A data dictionary can be a great asset not only to the DBA for database design, implementation and maintenance, but also to managers or end users in their project planning. Figure 4 shows these components and the connection among them.

1.9 ADVANTAGES AND DISADVANTAGES OF DATABASE MANAGEMENT SYSTEM

One of the main advantages of using a database system is that the organisation can exert, via the DBA, centralised management and control over the data. The database administrator is the focus of the centralised control. Any application requiring a change in the structure of a data record requires an arrangement with the DBA, who makes the necessary modifications. such modifications do not affect other applications or users of the record in question. Therefore, these changes meet another requirement of the DBMS: data independence. The following are the important advantages of DBMS :

1.9.1 Advantages

Reduction of Redundancies

Centralised control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large mass of data. Another advantage of avoiding duplication is the elimination of the inconsistencies that tend to be present in redundant data files. Any redundancies that exist in the DBMS are controlled and the system ensures that these multiple copies are consistent.

Sharing Data

A database allows the sharing of data under its control by any number of application programs or users.

Data Integrity

Centralised control can also ensure that adequate checks are incorporated in the DBMS to provide data integrity. Data integrity means that the data contained in the database is both accurate and consistent. Therefore, data values being entered for storage could be checked to ensure that they fall within a specified range and are of the correct format. For example, the value for the age of an employee may be in the range of 16 and 75. Another integrity check that should be incorporated in the database is to ensure that if there is a reference to certain object, that object must exist. In the case of an automatic teller machine, for example, a user is not allowed to transfer funds from a nonexistent saving account to a checking account.

Data Security

Data is of vital importance to an organisation and may be confidential. Such confidential data must not be accessed by unauthorised persons. The DBA who has the ultimate responsibility for the data in the DBMS can ensure that proper access procedures are followed, including proper authentication schemas for access to the DBMS and additional checks before permitting access to sensitive data. Different levels of security could be implemented for various types of data and operations. The enforcement of security could be datavalue dependent (e.g., a manager has access to the salary details of employees in his or

her department only), as well as data-type dependent (but the manager cannot access the medical history of any employees, including those in his or her department).

Conflict Resolution

Since the database is under the control of the DBA, she or he should resolve the conflicting requirements of various users and applications. In essence, the DBA chooses the best file structure and access method to get optimal performance for the response-critical applications, while permitting less critical applications to continue to use the database, albeit with a relatively slower response.

Data Independence

Data independence, is usually considered from two points of view: **physical data independence and logical data independence.** Physical data independence allows changes in the physical storage devices or organisation of the files to be made without requiring changes in the conceptual view or any of the external views and hence in the application programs using the database. Thus, the files may migrate from one type of physical media to another or the file structure may change without any need for changes in the application programs. Logical data independence implies that application programs need not be changed if fields are added to an existing record; nor do they have to be changed if fields not used by application programs are deleted. Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. Data independence is advantageous in the database environment since it allows for changes at one level of the database without affecting other levels. These changes are absorbed by the mappings between the levels.

Logical data independence is more difficult to achieve than physical independence. Since application programs are heavily dependent on the logical structure of the data they access.

The concept of data independence is similar in many respects to the concept of abstract data type in modern programming languages like C++. Both hide implementation details from the users. This allow users to concentrate on the general structure rather than low-level implementation details.

1.9.2 Disadvantages

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing or developing the software, the hardware has to be upgraded to allow for the extensive programs and the work spaces required for their execution and storage. The processing overhead introduced by the DBMS to implement security, integrity, and sharing of the data causes a degradation of the response and through-put times. An additional cost is that of migration from a traditionally separate application environment to an integrated one.

While centralisation reduces duplication, the lack of duplication requires that the database be adequately backedup so that in the case of failure the data can be recovered. Backup and recovery operations are fairly complex in a DBMS environment, and this is exacerbated in a concurrent multiuser database system. Furthermore, a database system requires a certain amount of controlled redundancies and duplication to enable access to related data items.

Centralisation also means that the data is accessible from a single source namely the database. This increases the potential severity of security breaches and disruption of the operation of the organisation because of downtimes and failures. The replacement of a monolithic centralised database by a federation of independent and cooperating distributed databases resolves some of the problems resulting from failures and downtimes.

Check Your Progress

1. What are the important tasks of Database manager?

   .....

   .....

   .....

2. What are the main functions of database administrator?

   .....

   .....

What are the drawbacks of file processing system?

.....

.....

1.10 SUMMARY

A database system is an integrated collection of related files along with the details about their definition, interpretation, manipulation and maintenance. A DBMS is a major software component of database system. It consists of collection of interrelated data and programs to access that data. The primary goal of a DBMS is to provide an environment which is both convenient and efficient to use in retrieving information from and storing information into the database.

The DBMS not only makes the integrated collection of reliable and accurate data available to multiple applications and users but also controls from unauthorised users to access the data.

A DBMS is a major software system consisting of a number of elements. It provides users DDL for defining the external and conceptual view of the data and DML for manipulating the data stored in the database. The database manager is the component of DBMS that provide the interface between the user and the file system. The database administration defines and maintains the three levels of the database as well as the mapping between levels to insulate the higher levels from changes that take place in the lower levels. The DBA is responsible for implementing measures for ensuring the security, integrity and recovery of the database.

1.11 MODEL ANSWERS

1. The database manager is responsible for the following tasks :
   - interaction with the file manager
   - integrity enforcement
   - security enforcement
   - backup and recovery
   - concurrent control

Some database system, designed for use on small personal computers are missing several of the features listed above. This allows for a smaller data manager. A small data manager has less requirement for physical resources, specially main memory and costs less to implement.

2. The function of database administrator include :
   - Schema definition
   - Storage structure and access method definition
   - Granting of authorisation for data access
   - Integrity constraint specification

3. The drawbacks of the file processing system are :
   - Data redundancy and inconsistency
   - Data isolation
   - Security problems
   - Integrity problems

These difficulties among others, have prompted the development of DBMS.

1.12 FURTHER READINGS

1. Bipin C. Desai, *An Introduction to Database Systems*, Golgotia Publication Pvt. Ltd. 1994.
2. Henery F. Korth, Abraham Silberschatz, *Database System Concepts*, McGraw Hill International Editions.

UNIT 2 DATABASE MODELS AND ITS IMPLEMENTATION

Structure

2.0 INTRODUCTION

In the previous unit, you have seen the limitations of the traditional approach to information processing and the advantages and limitations of data base approach. However, there are still many ways in which a data base approach can be implemented. You are possibly now familiar with dBASE III kind of approach to data base management which is branded as a relational data base type approach.But other varieties also exist, though relational data bases becoming more and more popular.

One can say that a **DBMS is a mechanism for coordinating the storage and retrieval of data in such a manner that its integrity, consistency and availability is ensured.**Some of the earlier approaches adopted a tree like hierarchical structure, while another approach more commonly known as the CODASYL data base adopted the network architecture.This unit will briefly describe the features of all these three approaches,including, **entity relationship** which is usually used for application development.With each model we will also discuss some of its implementation strategies.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- identify the structures in the different models of DBMS;
- convert any given data base situation to a hierarchical or relational model;
- discuss entity-relationship model
- state the essential features of the relational model; and
- discuss implementation issues of all the three models.

2.2 FILE MANAGEMENT SYSTEM

The precursor to the present day database management systems was File Management Systems (FMSs). In the early days of data processing, all files were flat files.A flat file is one where each record contains the same types of data items.One or more of these data items are designated as the key and is used for sequencing the file and for locating and grouping records by sorting and indexing.You will see that in the real world, which is to be translated into a database approach, many file structures are not flat.They are described with words or phrases like hierarchical files, codes, sets, tree structures and networks.All these types of structures can be closed as either trees or clause structures.However, it may be borne in mind that all these complicated file structure can be broken down into groups of flat files with redundant data item.

It is in this context that the FMS resembles the DBMS and it allows applications to be developed without having to write high level language programs. They came into being in the 1960s as a more productive approach to information access than a traditional route of programming via a high level language.

An FMS consists of a number of application programs. Because productivity enhancement in using an FMS compared to a conventional high level language is about 10 to 1, programmers use it. But the ease of use of an FMS also encourages end users with no previous programming experience to perform queries with special FMS language. One of the more well known in this regard is **RPG (Report Program Generator)** which was very popular for generating routine business reports. In order to use the RPG the user would define the input fields required by filling out an input specification format. Similarly output formats can be specified by filling out an output specification forms. The possibility of giving a certain structure to the output and the availability of default options made the package relatively easy to learn and use. Some well-known examples of such FMS packages are Mark-4, Data tree, easy tree and power ~use. The structure of an FMS is diagramatically illustrated below:

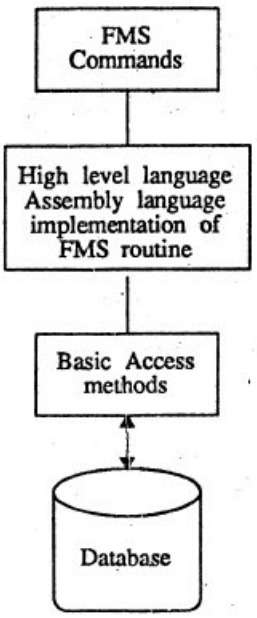


Figure 1: File management system

The FMS relies upon the basic access methods of the host operating system for data management. But it may have its own special language to be used in performing the retrievals. This language in some ways is more powerful than the standard high level programming languages in the way they define the data and development applications. Therefore, the file management system may be considered to be a level higher than mere high level languages. If we think of conventional high level programming languages such as FORTRAN, PASCAL, etc., as third generation languages and the non-procedural SQL as fourth generation languages, then the FMS language would fit somewhere in between.

The FMS program may actually take less time to execute than an equivalent program written in a high level language because of the built-in algorithm for sort, merge, and report generation in the FMS software. Therefore some of the advantages of FMS compared to standard high level language are:

- less software development cost – Even by experienced programmers it takes months or years in developing a good software system in high level language.
- Support of efficient query facility – On line queries for multiple-key retrievals are tedious to program.

Of course one could bear in mind the limitations of an FMS in the sense FMS cannot handle complicated mathematical operations and array manipulations. In order to remedy the situation some FMSs provide an interface to call other programs written in a high level language or an assembly language.

Another limitation of FMS is that for data management and access it is restricted to basic access methods. The physical and logical links required between different files to be able to cope with complex multiple key queries on multiple files is not possible. Even though FMS is a simple, powerful tool it cannot replace the high level language, nor can it perform complex information retrieval like DBMS. It is in this context that reliance on a good database management system become essential.

2.3 ENTITY-RELATIONSHIP (E-R) MODEL

E-R model grew out of the excercise of using commercially available DBMS to model application database. Earlier DBMS were based on hierarchical and network approach. E-R is a generalisation of these models. Although it has some means of describing the physical database model, it is basically useful in the design of logical database model. This analysis is then used to organise data as a relation, normalising relations and finally obtaining a relational database model.

The entity-relationship model for data uses three features to describe data. These are:

1. **Entities** which specify distinct real-world items in an application.
2. **Relationships** which connect entities and represent meaningful dependencies between them.
3. **Attributes** which specify properties of entities and relationships.

We illustrate these terms with an example. A **vendor** supplying items to a company, for example, is an **entity.** The **item** he supplies is another entity. A vendor and an item are related in the sense that a vendor supplies an item. The act of supplying defines a relationship between a vendor and an item. An **entity set** is a collection of similar entities. We can thus define a **vendor set** and an **item set.** Each member of an entity set is described by some attributes. For example, a vendor may be described by the attributes

(vendor code, vendor name, address)

An item may be described by the attributes

(item code, item name)

Relationship also can be characterised by a number of attributes. We can think of the relationship as **supply** between vendor and item entities. The relationship supply can be described by the attributes: (order_no, date of supply).

2.3.1 Relationship Between Entity Sets

The relationship between entity sets may be many-to-many (M:N), one-to-many (1:N), many-to-one (M:1) or one-to-one (1:1). The 1:1 relationship between entity sets E_1 and E_2 indicates that for each entity in either set there is at most one entity in the second set that is associated with it. The 1:M relationship from entity set E_1 to E_2 indicates that for an occurrence of the entity from the set E_1, there could be zero, one, or more entities from the entity set E_2 associated with it. Each entity in E_2 is associated with at most one entity in the entity set E_1. In the M:N relationship between entity sets E_1 and E_2, there is no restriction as

to the number of entities in one set associated with an entity in the other set. The database structure, employing the E-R model is usually shown pictorially using entity-relationship (E-R) diagram.

To illustrate these different types of relationships, consider the following entity sets: DEPARTMENT, MANAGER, EMPLOYEE, and PROJECT.

The relationship between a DEPARTMENT and a MANAGER is usually one-to-one; there is only one manager per department and a manager manages only one department. This relationship between entities is shown in figure 2 . Each entity is represented by a

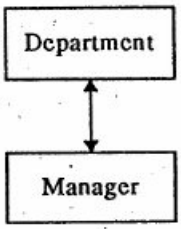


Figure 2 : One-to-one relationship

rectangle and the relationship between them is indicated by a direct line. The relationship from MANAGER to DEPARTMENT and from DEPARTMENT to MANAGER is both 1:1. Note that a one-to-one relationship between two entity sets does not imply that for an occurrence of an entity from one set at any time there must be an occurrence of an entity in the other set. In the case of an organisation, there could be times when a department is without a manager or when an employee who is classified as a manager may be without a department to manage. Figure 3 shows some instances of one-to-one relationships between the entities DEPARTMENT and MANAGER.

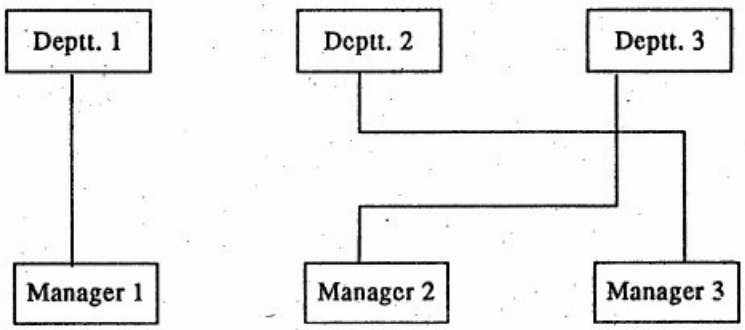


Figure 3 : Some instances of One-to-one relationship

A one-to-one relationship exists from the entity MANAGER to the entity EMPLOYEE because there are several employees reporting to the manager. As we just pointed out, there could be an occurrence of the entity type MANAGER having zero occurrences of the entity type EMPLOYEE reporting to him or her. A reverse relationship, from EMPLOYEE to MANAGER, would be many to one, since many employees may be supervised by a single manager. However, given an instance of the entity set EMPLOYEE, there could be only one instance of the entity set MANAGER to whom that employee reports (assuming that no employee reports to more than one manager). These relationships between entities are illustrated in figure 4. Figure 5 shows some instances of these relationships.

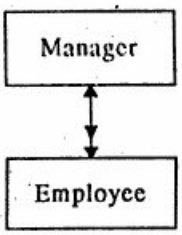


Figure 4 : Relationship 1:M

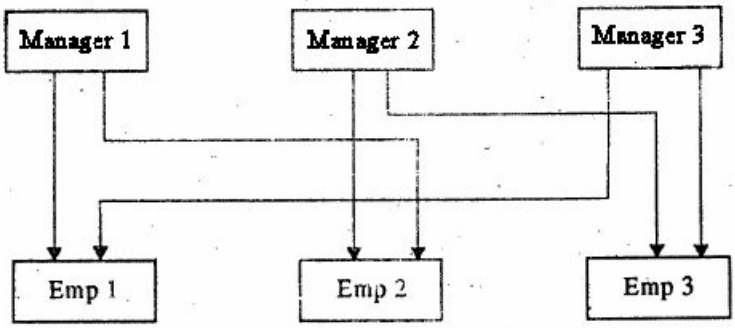


Figure 5 : Instances of 1:M relationship

The relationship between the entity EMPLOYEE and the entity PROJECT can be derived as follows: Each employee could be involved in a number of different projects, and a number of employees could be working on a given project. This relationship between EMPLOYEE and PROJECT is many-to-many. It is illustrated in figure 6. Figure 7 shows some instances of such a relationship.

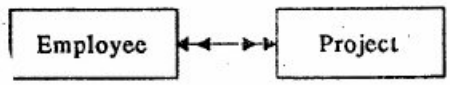


Figure 6 : M:N relationship

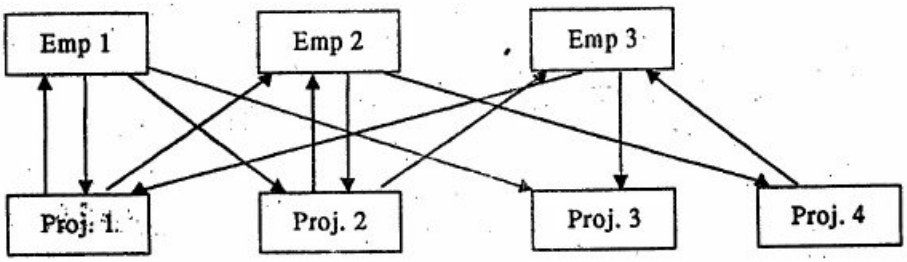


Figure 7: Instances of M:N relationship

In the entity-relationship (E-R) diagram, entities are represented by rectangles and relationships by a diamond-shaped box and attributes by ellipses or ovals.The following E-R diagram for vendor, item and their relationship is illustrated in figure 8(a).

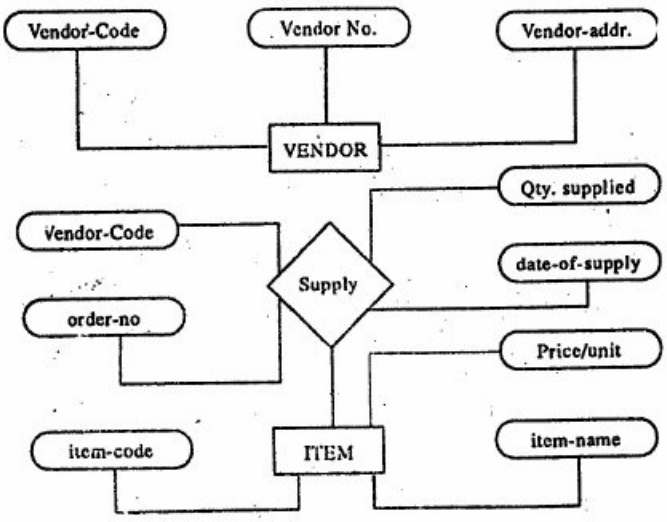


Figure 8(a) : E-R diagram for Vendors; items and their relationship

2.3.2 Representation of entity sets in the form of relations

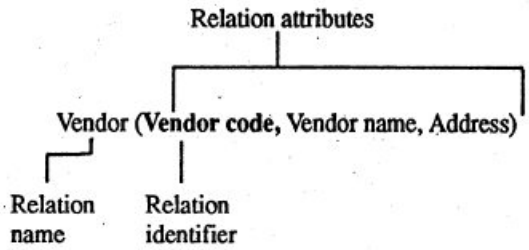
The entity relationship diagrams are useful in representing the relationship among entities they show the logical model of the database. E-R diagrams allow us to have an overview of the important entities for developing an information system and other relationship. Having obtained E-R diagrams, the next step is to replace each entity set and relationship set by a table or a relation. Each table has a name. The name used is the entity name. Each table has a number of rows and columns. Each row contains a member of the entity set. Each column corresponds to an attribute. Thus in the E-R diagram, the vendor entity is replaced by table 1.

Table 1 : Table for the Entity Vendor

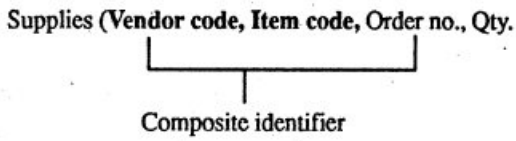
Vendor code	Vendor name	Address
1456	Ram and Co.	112 First Cross, Bangalore - 12
1685	Gopal and Sons	452 Fourth Main, Delhi - 8
1284	Sivraj Bros.	368 M.G. Road, Pune - 8
1694	Gita Ltd.	495 N.S.C. Road, Calicut - 9

The above table is also known as a relation. Vendor is the relation name. Each row of a relation is called a tuple. The titles used for the columns of a relation are known as relation attributes. Each tuple in the above example describes one vendor. Each element of a tuple gives a specific property of that vendor. Each property is identified by the title used for an attribute column. In a relation the rows may be in any order. The columns may also be depicted in any order. No two rows can be identical.

Since it is inconvenient to show the whole table corresponding to a relation, a more concise notation is used to depict a relation. It consists of the relation name and its attributes. The identifier of the relation is shown in bold face. The relation of table 1 is depicted as:



The other relations corresponding to figure are:



Supplied, Date of supply, price/unit)

A specified value of a relation identifier uniquely identifies the row of a relation.

If a relationship is M:N, then the identifier of the relationship entity is a composite identifier which includes the identifiers of the entity sets which are related. On the other hand, if the relationship is 1:N, then the identifier of the relationship entity is the identifier of one of the entity sets in the relationship. For example, the relations and identifiers corresponding to the E-R diagram of figure 8(b) are:

Teacher (**Teacher-id**, name, department, address)

Advises (**Teacher-id**, Student-id)

Students (**Student-id**, name, department, address)

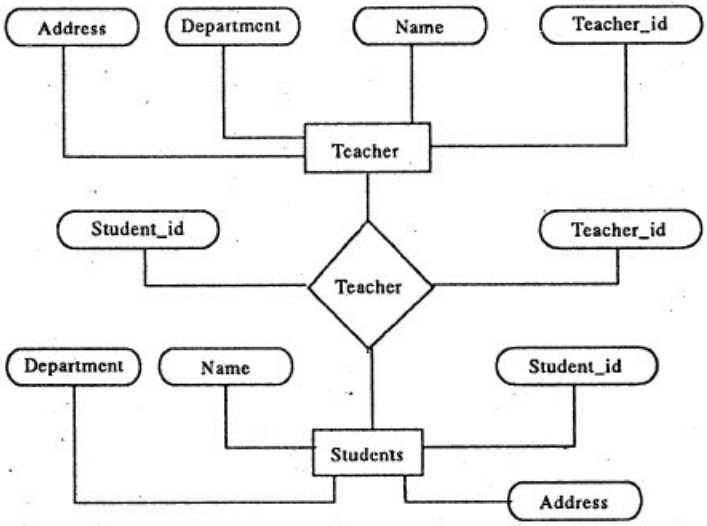


Figure 8(b) : E-R Diagram for Teacher, Students and their relationship

One may ask why an entity set is being represented as a relation. The main reasons are ease of storing relations as flat files in a computer and, more importantly, the existence of a sound theory on relations, which ensures good database design. The raw relations obtained as a first step in the above examples are transformed into normal relations. The rules for transformations called normalisation are based on sound theoretical principles and ensure that the final normalised relations obtained reduce duplication of data, ensure that no mistakes occur when data are added or deleted and simplify retrieval of required data. We will not discuss in detail the theory of normalisation. We will, however, describe what these normalisation steps are, why they are needed, and how they are done. The end product of all these steps in a good database design for an application.

2.3.3 Generalisation and Specification

Abstraction is the simplification mechanism used to hide superfluous details of a set of objects, it allows one to concentrate on the properties that are of interest to the application. As such, car is an abstraction of a personal transportation vehicle but does not reveal details about mode, year, colour, and so on. Vehicle itself is an abstraction that includes the types car, truck, and bus.

There are two main abstraction mechanisms used to model information: Generalisation and aggregation. **Generalisation** is the abstracting process of viewing set of objects as a single general class by concentrating on the general characteristics of the constituent sets while suppressing or ignoring their differences. It is the union of a number of lower-level entity types for the purpose of producing a higher-level entity type. For instance, student is a generalisation of graduate or undergraduate, full-time or part-time students. Similarly, employee is a generalisation of the classes of objects cook, waiter, cashier, maltre d'. Generalisation is an IS_A relationship; therefore, manager IS_An employee, cook IS_An employee, waiter IS_An employee, and so forth. **Specialisation** is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. This involves taking a higher-level entity and, using additional characteristics, generating lower-level entities. The lower-level entities also inherit the characteristics of the higher-level entity. In applying the characteristic size to car we can create a full-size, mid-size, compact, or subcompact car. Specialisation may be seen as the reverse process of generalisation: additional specific properties are introduced at a lower level in a hierarchy of objects. Both processes are illustrated in the following figure 9 wherein the lower levels of the hierarchy are disjoint.

The entity set EMPLOYEE is a generalisation of the entity sets FULL__TIME_EMPLOYEE and PART_TIME_EMPLOYEE. The former is a generalisation of the entity sets faculty and staff; the latter, that of the entity sets TEACHING and CASUAL. FACULTY and STAFF inherit the attribute Salary of the entity set FULL_TIME_EMPLOYEE and the latter, in turn, inherits the attributes of EMPLOYEE. FULL_TIME_EMPLOYEE is a specialisation of the entity set EMPLOYEE and is differentiated by the additional attribute Salary. Similarly, PART_TIME _EMPLOYEE is a specialisation differentiated by the presence of the attribute Type.

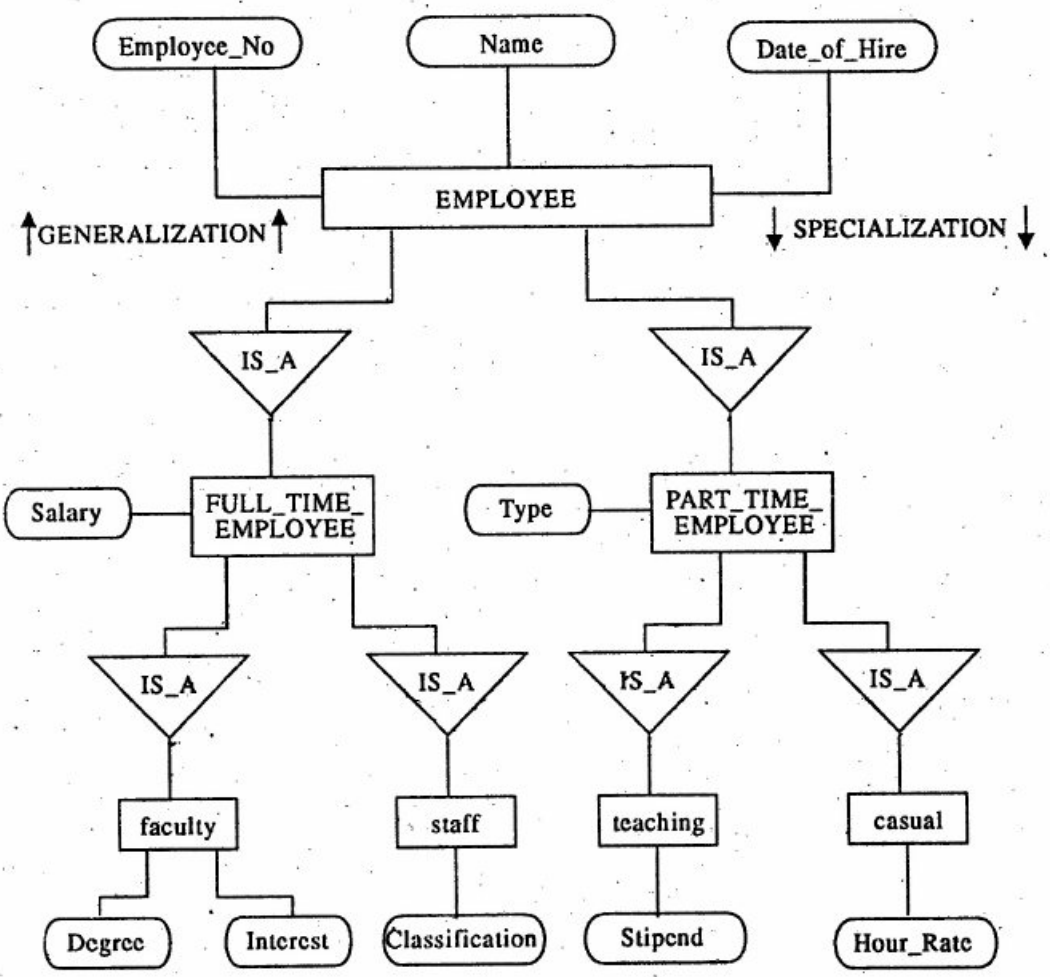


Figure 9 : Generalisation and Specialisation

In designing a database to model a segment of the real world, the data modelling scheme must be able to represent generalisation. It allows the model to represent generic entities and treat a class of objects and specifying relationships in which the generic objects participate.

Generalisation forms a hierarchy of entities and can be represented by a hierarchy of tables which can also be shown through following relations for conveniences.

EMPLOYEE(**Empl_no**, name, Date_of_birth)

FULL_TIME(**Empl_no**, salary)

PART_TIME(**Empl_no**, type)

FACULTY(**Empl_no**, Degree, Interest)

STAFF(**Empl_no**, Hour_rate)

TEACHING(**Empl_no**, Stipend)

Here the primary key of each entity corresponds to entries in different tables and directs one to the appropriate row of related tables.

Another method of representing a generalisation hierarchy is to have the lowest-level entities inherit the attributes of the entities of higher levels. The top and intermediate-level entities are not included as only those of the lowest level are represented in tabular form. For instance, the attributes of the entity set FACULTY would be {**Empl_No**, Name, Date_of_Hire, Salary, Degree, Interest}. A separate table would be required for each lowest-level entity in the hierarchy. The number of different tables required to represent these entities would be equal to the number of entities at the lowest level of the generalisation hierarchy.

2.3.4 Aggregation

Aggregation is the process of compiling information on an object, thereby abstracting a higher-level object. In this manner, the entity person is derived by aggregating the characteristics name, address, and Social Security number. Another form of the aggregation is abstracting a relationship between objects and viewing the relationship as an object. As such, the ENROLLMENT relationship between entities student and course could be viewed as entity REGISTRATION. Examples of aggregation are shown in figure 10.

Consider the relationship COMPUTING of figure 11. Here we have a relationship among the entities STUDENT, COURSE, and COMPUTING SYSTEM. A student registered in a given

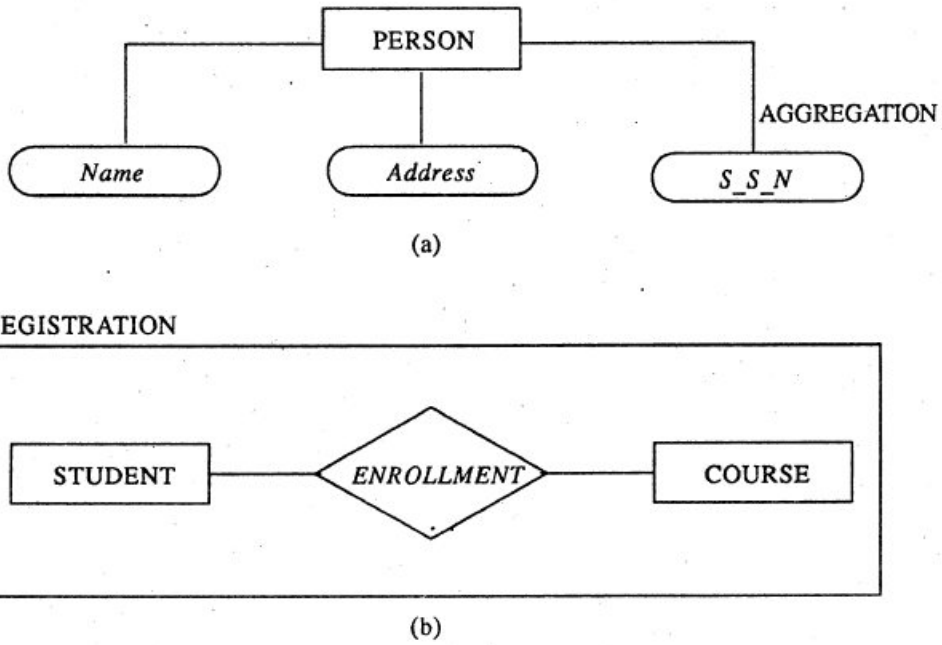


Figure 10 : Examples of aggregation

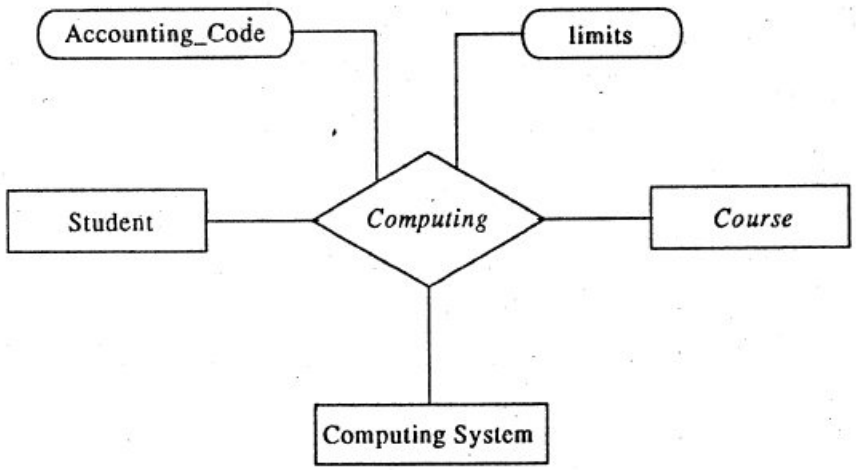


Figure 11 : A relationship among three entities

course uses one of several computing systems to complete assignments and projects. The relationship between the entities STUDENT and COURSE could be the aggregated entity REGISTRATION (figure 10b), as discussed above. In this case, we could view the ternary relationship of figure 11 as one between registration and the entity computing system. Another method of aggregating is to consider a relationship consisting of the entity COMPUTING SYSTEMs being assigned to COURSEs. This relationship can be aggregated as a new entity and a relationship established between it and STUDENT. Note that the difference between a relationship involving an aggregation and one with the three entities lies in the number of relationships. In the former case we have two relationships; in the latter,

only one exists. The approach to be taken depends on what we want to express. We would use the ternary relationship related to a COMPUTING SYSTEM.

2.4 THE HIERARCHICAL MODEL

A DBMS belonging to the hierarchical data model uses tree structures to represent relationship among records. Tree structures occur naturally in many data organisations because some entities have an intrinsic hierarchical order. For example, an institute has a number of programmes to offer. Each program has a number of courses. Each course has a number of students registered in it. The following figure depicts the four entity types **Institute, Program, Course** and **Student** make up the four different levels of hierarchical structure. The figure 12 shows an example of database occurrence for an institute. A database is a collection of database occurrence.

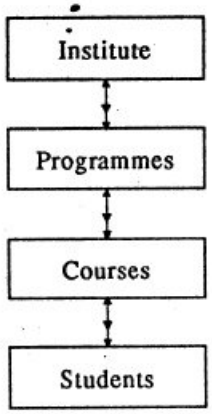


Figure 12 : A simple Hierarchy

A hierarchical database therefore consists of a collection of records which are connected with each other through links. Each record is a collection of fields (attributes), each of which contains one data value. A link is an association between precisely two records.

A tree structure diagram serves the same purpose as an entity-relationship diagram; namely it specifies the overall logical structure of the database.

The following figure shows typical database occurrence of a hierarchical structure (tree structure)

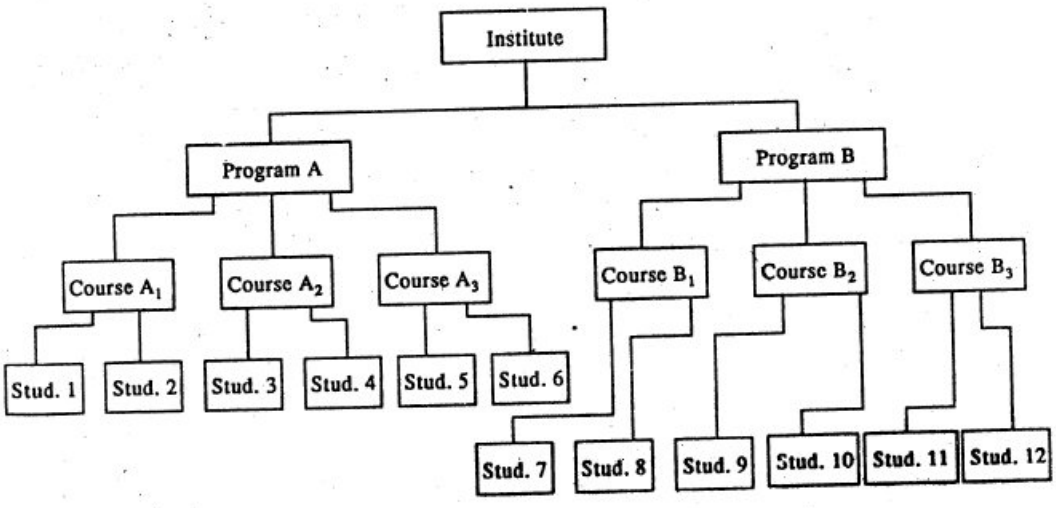


Figure 13 : Database occurrence of a hierarchical structure

The hierarchical data model has the following features:

- Each hierarchical tree can have only one root record type and this record type does not have a parent record type.
- The root can have any number of child record types and each of which can itself be a root of a hierarchical subtree.
- Each child record type can have only one parent record type; thus a M:N relationship cannot be directly expressed between two record types.
- Data in a parent record applies to all its children records
- A child record occurrence must have a parent record occurrence; deleting a parent record occurrence requires deleting all its children record occurrence.

2.4.1 Replication Vs Virtual Record

The hierarchical model, like the network model (discussed in the next section) cannot support a many-to many relationship directly. In the network model the many-to-many relationship is implemented by introducing an intermediate record and two one-to- many relationships. In the hierarchical model, the many-to-many relationship can be expressed using one of the following methods: **replication** or virtual record. When more than one employee works in a given department, then for the hierarchical tree with **EMPLOYEE** as the root node we have to replicate the record for the department and have this replicated record attached as a child to the corresponding occurrence of the **EMPLOYEE** record type.

Replication of data would mean a waste of storage space and could lead to data inconsistencies when some copies of replicated data are not updated. The other method of representing the many- to many relationship in the hierarchical data model is to use an indirect scheme similar to the network approach. In the hierarchical model the solution is to use the so-called **virtual record**. A virtual record is essentially a record containing a pointer to an occurrence of an actual physical record type. This physical record type is called the **logical parent** and the virtual record is the **logical child**. When a record is to be replicated in several database trees, we keep a single copy of that record in one of the trees and replace, each other record with a virtual containing a pointer to that physical record. To be more specific, let R be a record type that is replicated in several hierarchical diagrams say H_1, H_2, ... H_n. To eliminate replication we create a new virtual record type virtual - R, and replace R in each of the n-1 trees with a record of type virtual - R. Virtual - R will contain no data.

2.4.2 The Accessing of Data Records in Hierarchical Data Structure

The tree type data structure is used to represent hierarchical data model shows the relationships among the parents, children, cousins, uncles, aunts, and siblings. A tree is thus a collection of nodes. One node is designated as the root node; the remaining nodes form trees or subtrees.

An **ordered tree** is a tree in which the relative order of the subtrees is significant. This relative order not only signifies the vertical placement or level of the subtrees but also the left to right ordering. Figures 14 (a) and (b) give two examples of ordered trees with **A** as the root node and **B, C,** and **D** as its children nodes. Each of the nodes **B, C,** and **D**, in turn, are

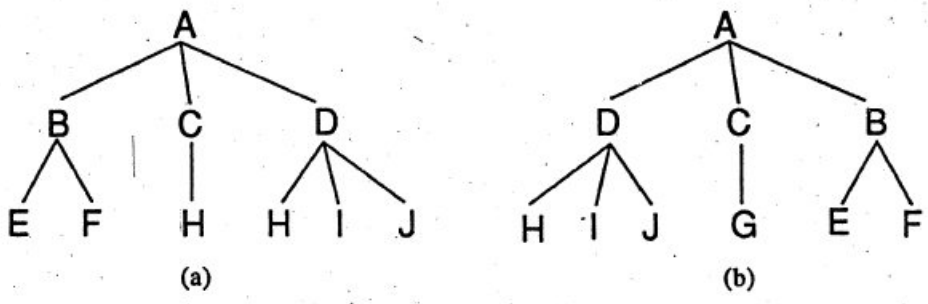


Figure 14 : Example of two trees

root nodes or subtrees with children nodes (E, F), (G), and (H, I, J), respectively. The significance in the ordering of the subtrees in these diagrams is discussed below.

Traversing an ordered tree can be done in a number of ways. The order of processing the nodes of the tree depends on whether or not one processes the node before the node's subtree and the order of processing the subtrees (left to right or right to left). The usual practice is the so-called **preorder traversal** in which the node is processed first, followed by the leftmost subtree not yet processed.

The preorder processing of the ordered tree of figure 14(a) will process the nodes in the sequence **A, B, E, F, C, G, D, H, I, J.**

The significance of the ordered tree becomes evident when we consider the sequence in which the nodes could be reached when using a given tree traversing strategy. For instance, the order in which the nodes of the hierarchical tree of figure are processed using the preorder processing strategy is not the same as the order for figure 14(a), even though the tree of part b contains the same nodes as the tree of part a.

Two distinct methods can be used to implement the preorder sequence in the ordered tree. The first method, shown in figure 15 uses hierarchical pointers to implement the ordered tree of part 14(a). Here the pointer in each record points to the

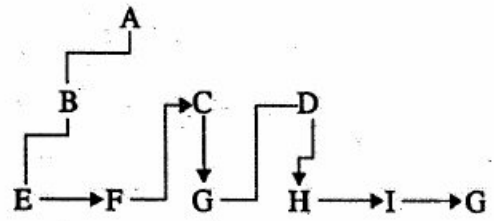


Figure 15: Preorder Traversal of figure 14(a)

next record in the preorder sequence. The second method, shown in figure 16 uses two types of pointers, the **child** and the **sibling pointers**. The child pointer is used to point to the

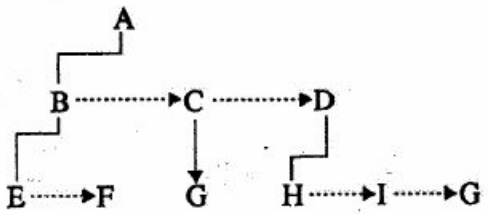


Figure 16 : Child/Sibling Pointers

leftmost child and the sibling pointer is used to point to the right sibling. The siblings are nodes that have the same parent. Thus, the binary tree corresponding to tree in the figure (a) is obtained by connecting together all siblings of a node and deleting all links from a node to its children except for the link to its leftmost child. Using this transformation, we obtain the tree representation as shown in figure 16.

2.4.3 Implementation of the Hierarchical Data Model

Each occurrence of a hierarchical tree can be stored as a variable length physical record, the nodes of the hierarchy being stored in preorder. In addition the stored record contains a prefix field. This field contains control information including pointers, flags, locks and counters, which are used by DBMS to allow concurrent usage and enforce data integrity.

A number of methods could be used to store the hierarchical trees in the physical medium affects not only the performance of the system but also the operations that can be performed on the database. For example, if each occurrence of the hierarchical tree is stored as a variable length record on a magnetic tape like device, the DBMS will allow only sequential

retrieval and insertion or modification may be disallowed or performed only by recreating the entire database with the insertion and modification storage of the hierarchical database on a direct access device allows an index structure to be supported for the root nodes and allows direct access to an occurrence of a hierarchical tree. The storage of one occurrence of the hierarchical definition tree of figure 14 (a) using the variable length record approach is given in the following figure.

$$\left\{ \overline{a_1\ b_1\ e_{11}\ l_{12}\ f_1\ b_2\ e_{21}\ e_{22}\ f_2\ c_1\ g_1\ d_1\ h_1\ I_1\ J_1\ d_2\ h_{21}\ h_{22}\ I_2\ J_2} \right\}$$

Figure 17 : Sequential storage of hierarchical database

The hierarchy can also be represented using pointer of either preorder hierarchical type or child/sibling type. In the hierarchical type of pointer, each record occurrence has a pointer that points to the next record in the preorder sequence. In the child/sibling scheme, each record has two types of pointers. The **child** pointer points to its leftmost child record occurrence. The **sibling** pointer points to its right sibling (or twin). A record has one sibling pointer and as many child pointers as the number of child types associated with the node corresponding to the record. The following two figures, figure 18 and figure 19 illustrates preorder hierarchical pointer and child sibling pointers respectively of hierarchical tree shown in figure 14.

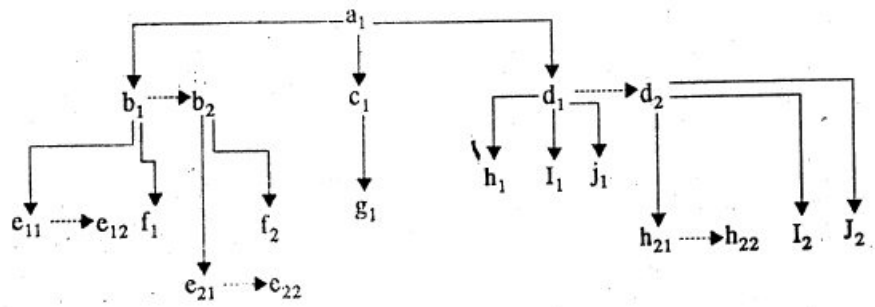


Figure 18 : Preorder Hierarchical Pointer

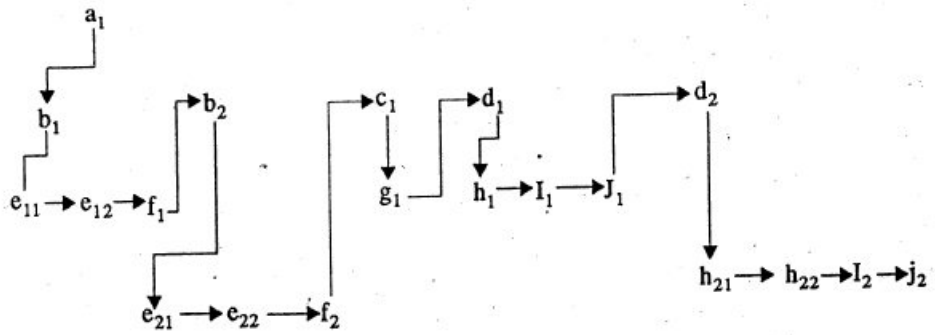


Figure 19 : Child Sibling Pointers

2.5 THE NETWORK MODEL

The network data model was formalised in the late 1960s by the Database Task Group of the Conference on Data System Language (DBTG/CODASYL). Their first report which has been revised a number of times, contained detailed specifications for the network data mode (a model conforming to these specifications is also known as the DBTG data model). The specifications contained in the report and its subsequent revisions have been subjected to much debate and criticism. Many of the current database applications have been built on commercial DBMS systems using the DBTG model.

2.5.1 DBTG Set

The DBTG model uses two different data structures to represent the database entities and relationships between the entities, namely record type and set type. A **record type** is used to represent an entity type. It is made up of a number of data items that represent the attributes of the entity.

A **set type** is used to represent a directed relationship between two record types, the so-called **owner record type**, and the **member record type**. The set type, like the record type, is named and specifies that there is a one-to-many relationship (1:M) between the owner and member record types. The set type can have more than one record type as its member, but only one record type is allowed to be the owner in a given set type. A database could have one or more occurrences of each of its record and set types. An occurrence of a set type consists of an occurrence of each of its record and set types. An occurrence of a set type consists of an occurrence of the owner record type and any number of occurrences of each of its member record types. A record type cannot be a member of two distinct occurrences of the same set type.

Bachman introduced a graphical means called a data structure diagram to denote the logical relationship implied by the set. Here a labelled rectangle represents the corresponding entity or record type. An arrow that connects two labelled rectangles represents a set type. The arrow direction is from the owner record type to the member record type. Figure shows two record types (DEPARTMENT and EMPLOYEE) and the set type DEPT_EMP, with DEPARTMENT as the owner record type and EMPLOYEE as the member record type.

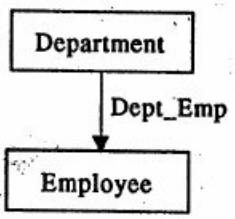


Figure 20 : A DBTG set

The data structure diagrams have been extended to include field names in the record type rectangle, and the arrow is used to clearly identify the data fields involved in the set association. A one-to-many (1:M) relationship is shown by a set type arrow that starts from the owner field in the owner record type. The arrow points to the member field within the member record type. The fields that support the relationship are clearly identified.

Each entity type in an E-R diagram is represented by a logical record type with the same name. The attributes of the entity are represented by data fields of the record. We use the term **logical record** to indicate that the actual implementation may be quite different.

The conversion of the E-R diagram into a network database consists of converting each 1:M binary relationship into a set (a 1:1 binary relationship being a special case of a 1:M relationship). If there is a 1:M binary relationship R_1 from entity type E_1 to entity type E_2,

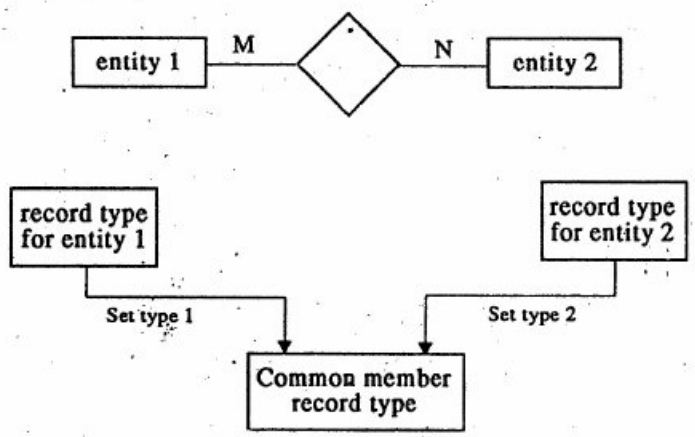


Figure 21 : Conversion of an M:N relationship into two 1:M DBTG sets

then the binary relationship is represented by a set. An instance of this would be S_1 with an instance of the record type corresponding to entity E_1 as the owner and one or more instances of the record type corresponding to entity E_2 as the member. If a relationship has attributes, unless the attributes can be assigned to the member record type, they have to be maintained in a separate logical record type created for this purpose. The introduction of this additional record type requires that the original set be converted into two symmetrical sets, with the record corresponding to the attributes of the relationship as the member in both the sets and the records corresponding to the entities as the owners.

Each many-to-many relationship is handled by introducing a new record type to represent the relationship wherein the attributes, if any, of the relationship are stored. We then create two symmetrical 1:M sets with the member in each of the sets being the newly introduced record type. The conversion of a many-to-many relationship into two one-to-many sets using a common member record type is shown in figure 21.

In the network model, the relationships as well as the navigation through the database are predefined at database creation time.

2.5.2 Implementation of the Network Data Model

The record is a basic unit to represent data in the DBTG network database model. The implementation of the one-to-many relationships of a set is represented by linking the members of a given occurrence of a set to the owner record occurrence. The actual method of linking the member record occurrence to the owner is immaterial to the user of the database; however, for our discussion, we can assume that the set is implemented using a linked list. The list starts at the owner record occurrence and links all the member record occurrences with the pointer in the last member record occurrence leading back to the owner record. Figure 22 shows the implementation of the set occurrence DEPT-EMP where the owner record is Comp.sc. and the member records are the instances Jancy and Santosh. Note that for simplicity we have shown only one of the record fields of each record. This method of implementation assigns one pointer (link) in each record for each set type in which the record participates and, therefore, allows a record occurrence to participate in only one occurrence of a given set type. Any other method of implementing the set construct in a database management system based on the DBTG proposal is, in effect, equivalent to the linked list method.

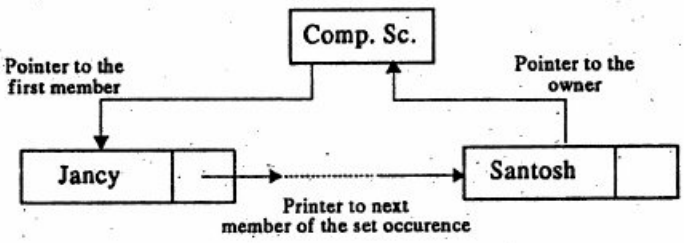


Figure 22 : Implementation of the DBTG SET in network model

A second form of network implementation, especially useful for M:N relationships, is a **bit map**, which is depicted in figure. A **bit map** is a matrix created for each relationship. Each row corresponds to the relative record number of a target record of a relationship. A 1 bit in a cell for row X and column Y means that the records corresponding to row X and column Y are associated in this relationship; a zero means no association. For example, figure 23 indicates that PRODUCT with relative record number X is related to VENDOR with relative record numbers 1 and Y (and possibly others not shown). Bit maps are powerful data structures for the following reasons:

1. Anu record type(s) can be included in rows or columns.
2. 1:1, 1:M, and M:1 relationships can all be represented.
3. Rows and columns can be logically manipulated by Boolean operators ("and," "or," "not") to determine records that satisfy complex associations (e.g., any record that has both parent S and parent T).

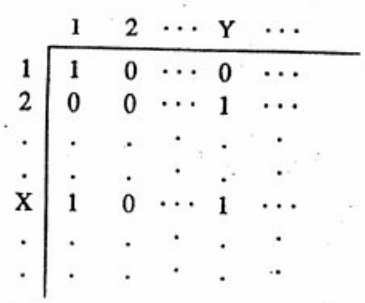


Figure 23 : Example of a bit map implementation for Product and vendor relationship in a network

4. A bit map can be manipulated equally as well in either a row or column access (all the row records for a common column or all the column records for a common row) and can be easily extended for n-ary relationships).

2.6 THE RELATIONAL MODEL

The relational data base approach is relatively recent and begun with a theoretical paper of Codd which proposed that by using a technique called normalisation the entanglement observation in the tree and network structure can be replaced by a relatively neater structure. Codd, principles relate to the logical description of the data and it is important to bear in mind that this is quite independent and feasible way in which the data is stored. It is only some years back that these concepts have emerged from the research development test and trial stages and are being seen as commercial projects. The attractiveness of the relational approach arouse from the simplicity in the data organisation and the availability of reasonably simple to very powerful query languages. The size of the relational database approach is that all the data is expressed in terms of tables and nothing but tables. Therefore, all entities and attributes have to be expressed in rows and columns. In the PC world also the availability of dBASE III and its later versions have encouraged the greater use of relational databases. The immense popularity of spreadsheets also arouse because of the inherent simplicity of expressing information in terms of rows and columns.

The differences that arise in the relational approach is in setting up relationships between different tables. This actually makes use of certain mathematical operations on the relation such as projection, union, joins, etc. These operations from relational algebra and relational calculus are discussion in some more details in the second Block of this course. Similarly in order to achieve the organisation of the data in terms of tables in a satisfactory manner, a technique called normalisation is used.

A unit in the second block of this course describes in detail the processing of normalisation and various stages including the first normal forms, second normal forms and the third normal forms. At the moment it is sufficient to say that normalisation is a technique which helps in determining the most appropriate grouping of data items into records, segments or tuples. This is necessary because in the relational model the data items are arranged in tables which indicate the structure, relationship and integrity in the following manner:

(1) In any given column of a table, all items are of the same kind

(2) Each item is a simple number or a character string

(3) All rows of a table are distinct. In other words, no 2 rows which are identical in every column.

(4) Ordering of rows within a table is immaterial

(5) The columns of a table are assigned distinct names and the ordering of these columns is immaterial

(6) If a table has N columns, it is said to be of degree N. This is sometimes also referred to as the cardinality of the table. From a few base tables it is possible by setting up relations, create views which provide the necessary information to the different users of the same database.

2.6.1 Advantages and Disadvantages of Relational Approach

Advantages of Relational approach

The popularity of the relational database approach has been apart from access of availability of a large variety of products also because it has certain inherent advantages.

(1) **Ease of use**: The revision of any information as tables consisting of rows and columns is quite natural and therefore even first time users find it attractive.

(2) **Flexibility**: Different tables from which information has to be linked and extracted can be easily manipulated by operators such as project and join to give information in the form in which it is desired.

(3) **Precision**: The usage of relational algebra and relational calculus in the manipulation of the relations between the tables ensures that there is no ambiguity which may otherwise arise in establishing the linkages in a complicated network type database.

(4) **Security**: Security control and authorisation can also be implemented more easily by moving sensitive attributes in a given table into a separate relation with its own authorisation controls. If authorisation requirement permits, a particular attribute could be joined back with others to enable full information retrieval.

(5) **Data Independence**: Data independence is achieved more easily with normalisation structure used in a relational database than in the more complicated tree or network structure.

(6) **Data Manipulation Language**: The possibility of responding to ad-hoc query by means of a language based on relational algebra and relational calculus is easy in the relational database approach. For data organised in other structure the query language either becomes complex or extremely limited in its capabilities.

Disadvantages of Relational Approach

One should not get carried way into believing that there can be no alternative to the RDBMS. This is not so. A major constraint and therefore disadvantage in the use of relational database system is machine performance. If the number of tables between which relationships to be established are large and the tables themselves are voluminous, the performance in responding to queries is definitely degraded. It must be appreciated that the simplicity in the relational database approach arise in the logical view. With an interactive system, for example an operation like **join** would depend upon the physical storage also. It is, therefore common in relational databases to tune the databases and in such a case the physical data layout would be chosen so as to give good performance in the most frequently run operations. It therefore would naturally result in the fact that the lays frequently run operations would tend to become even more shared.

While the relational database approach is a logically attractive, commercially feasible approach, but if the data is for example naturally organised in a hierarchical manner and stored as such, the hierarchical approach may give better results. It is helpful to have a summary view of the differences between the relational and the non-relational approach in the following section.

2.6.2 Difference between Relational and Other models

1. **Implementation independence** : The relational model logically represents all relationships implicitly, and hence, one does not know what associations are or are not physically represented by an efficient access path (without looking at the internal data model).

2. **Logical key pointers** : The relational data model uses primary (and secondary) keys in records to represent the association between two records. Because of this model's implementation independence, however, it is conceivable that the physical database (totally masked from the user of a relational database) could use address pointers or one of many other methods.

3. **Normalisation theory** : Properties of a database that make it free of certain maintenance problems have been developed within the context of the relational model (although these properties can also be designed into a network data model database).

4. **High-level programming languages** : Programming languages have been developed

specifically to access databases defined via the relational data model; these languages permit data to be manipulated as groups or files rather than procedurally: one record at a time.

2.6.3 An Example of a Relational Model

Let us see important features of a RDBMS through some examples as shown in figure 24.

A relation has the following properties:

1. Each column contains values about the same attribute, and each table cell value must be simple (a single value).
2. Each column has a distinct name (attribute name), and the order of columns is immaterial.
3. Each row is distinct; that is, one row cannot duplicate another row for selected key attribute columns.
4. The sequence of the rows is immaterial.

PRODUCT relation

Attributes

	PRODUCT #	DESCRIPTION	PRICE	QUANTITY-ON-HAND	Relative record#
Tuples	0100	TABLE	500.00	42	1
	0975	WALL UNIT	750.00	0	2
	1250	CHAIR	400.00	13	3
	1775	DRESSER	500.00	8	4

Primary Key

VENDOR relation

VENDOR#	VENDOR-NAME	VENDOR-CITY
26	MAPLE HILL	DENVER
13	CEDAR CREST	BOULDER
16	OAK PEAK	FRANKLIN
12	CHERRY MTN	LONDON

SUPPLIES relation

VENDOR#	PRODUCT#	VENDOR-PRICE
13	1775	250.00
16	0100	150.00
16	1250	200.00
26	1250	200.00
26	1775	275.00

Figure 24 : Example of a relational data model

As shown in figure 24, a **tuple** is the collection of values that compose one row of a relation. A tuple is equivalent to a record instance. An n-tuple is a tuple composed of n attribute values, where n is called the **degree** of the relation. PRODUCT is an example of a 4-tuple. the number of tuples in a relation is its **cardinality**.

A domain is the set of possible values for an attribute. For example, the domain for QUANTITY-ON-HAND in the PRODUCT relation is all integers greater than or equal to zero. The domain for CITY in the VENDOR relation is a set of alphabetic characters strings restricted to the names of U.S. cities.

We can use a shorthand notation to abstractly represent relations (or tables). The three relations in figure 24 can be written in this notation as

PRODUCT (PRODUCT#, DESCRIPTION, PRICE,

QUANTITY-ON-HAND)

VENDOR(VENDOR#, VENDOR-NAME, VENDOR-CITY)

SUPPLIES (VENDOR#, PRODUCT#, VENDOR-PRICE)

In this form, the attribute (or attributes in combination) for which no more than one tuple may have the same (combined) value is called the **primary key.** (The primary key attributes are underlined for clarity.) The relational data model requires that a primary key of a tuple (or any component attribute if a combined key) may not contain a null value. Although several different attributes (called **candidate keys**) might serve as the primary key, only one (or one combination) is chosen. These other keys are then called **alternate keys.**

The SUPPLIES relation in figure 24 requires two attributes in combination in order to identify uniquely each tuple. A composite or **concatenated key** is a key that consists of two or more attributes appended together. Concatenated keys appear frequently in a relational data base, since intersection data, like VENDOR-PRICE, may be uniquely identified by a combination of the primary keys of the related entities. Each component of a concatenated key can be used to identify tuples in another relation. In fact, values for all component keys of a concatenated key must be present, although monkey attribute values may be missing. Further, the relational model has been enhanced to indicate that a tuple (e.g., for PRODUCT) logically should exist with its key value (e.g., PRODUCT#) if that value appears in a SUPPLIES tuple; this deals with existence dependencies.

We can relate tuples in the relational model only when there are common attributes in the relations involved. We will expand on this idea in the next section. The SUPPLIES relation also suggests that an M:N relationship requires the definition of a third relation, much like a link or intersection record in the simple network model.

Codd (1970) popularised the use of relations and tables as a way to model data. At first glance, this view of data may seem only to be a different perspective on the network data model (all we have done is replace address pointers with logical pointers and eliminate lines from the database diagram). Several debates have essentially argued this point. Codd and many others have shown that relations are actually formal operations on mathematical sets. Further, most data processing operations (e.g., printing of selected records and finding related records) can also be represented by mathematical operators on relations. The result of mathematical operations can be proved to have certain properties. A collection of operations, called normalisation, has been shown to result in databases with desirable maintenance and logical properties. This mathematical elegance and visual simplicity have made the relational data model one of the driving forces in the information systems field.

2.6.4 Conversion of Hierarchical and Network Structure into Relation

The relational data model is as rich as the complex network model in its ability to represent directly, without much redundancy, a wide variety of relationship types. However, unlike the network model, relationships are implicit; that is, there is no diagrammatic convention (arcs, or links) used to explicitly show a relationship between two relations (i.e., relationship between entities).

Hierarchical and network structure can be decomposed into relations when appropriate connection fields are inserted into relevant child record types. The figure 25 shows an example of converting a 4-level hierarchical structure into a relational data model.

To establish a data path from the root to a child node, the primary keys of their respective parents are inserted into all child nodes. For example, the primary key of COLLEGE is inserted into the PROGRAM relation, while the primary key of PROGRAM (PROGRAM(NAME) is in turn added to the STUDENT relation.

Hierarchical Schema

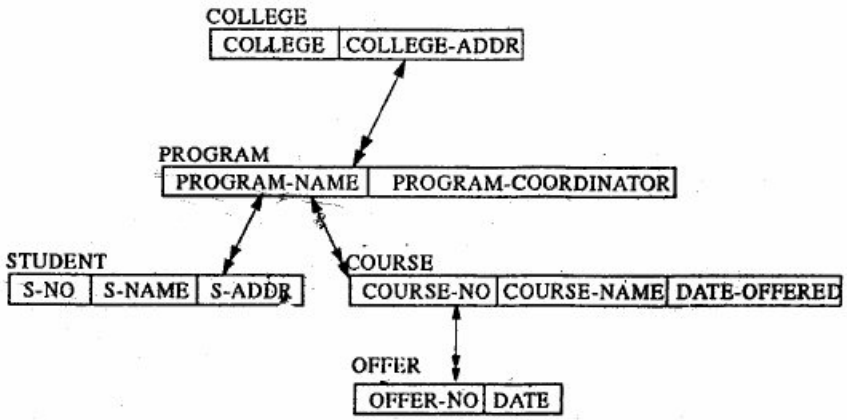


Figure 25 : Conversion of an N_Level Hierarchical Structure into a Relational Data Model

Let us consider the following set of relations that define a relational database for the complex network of figure 26:

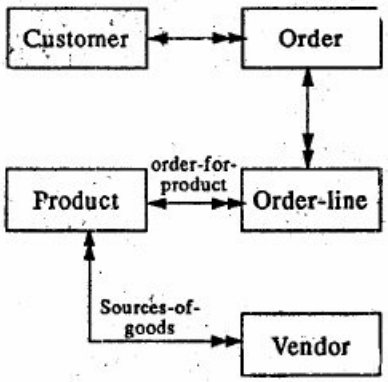


Figure 26 : Complex Network Data Model

CUSTOMER(CUSTOMER#, CUSTOMER-ADDRESS),
CUSTOMER-DETAILS)

ORDER(ORDER#,CUSTOMER#,ORDER-DATE,
DELIVERY-DATE, TOTAL-AMOUNT)

PRODUCT(PRODUCT#,DESCRIPTION, PRICE,
QUANTITY-ON-HAND)

ORDER-LINE(ORDER#,PRODUCT#,
QUANTITY-ORDERED, EXTENDED-PRICE)

VENDOR(VENDOR#,VENDOR-NAME,VENDOR-CITY)

SUPPLIES(VENDOR#, PRODUCT#)

In this example, CUSTOMER, PRODUCT, and VENDOR are basic relations that exist independently of all other data. The ORDER relation, too, can exist independently, but one of its attributes, CUSTOMER#, called a **cross-reference key**, implements the Orders-for-Customer relationship from figure 26. The attribute CUSTOMER# in the ORDER relation could have any name (say, ACCOUNT#). As long as the domain of values and the meaning of CUSTOMER# and ACCOUNT# are the same, then proper linking of related tuples can occur. We will use a dashed underline to denote a cross-reference key. The problem with using different names in different relations for the same attribute is that a "reader" of a relational database definition may not readily understand that these two attributes can be used to link related data. In most cases, use of a cross-reference key in the relational data model means that, for example, any value of CUSTOMER# found in an ORDER tuple logically should exist a CUSTOMER# in some unique existing CUSTOMER tuple.

The ORDER relation has its own unique key; ORDER#. An alternate key might be the combination of CUSTOMER# and ORDER- DATE (if customers do no submit two or more orders in a day). If ORDER# was not an essential piece of data for applications of this database, then the following SALE relation would be sufficient:

SALE(CUSTOMER#,ORDER-DATE,DELIVERY-DATE, TOTAL-AMOUNT)

Here the CUSTOMER# key appears as (part of) the primary key in each related record (tuple). In this case, CUSTOMER# is referred to as a **foreign key**. The term **referential integrity** applies to both cross-reference and foreign keys, and means that the key value must exist in the associated relation for database integrity. Thus, a SALE cannot be created unless a CUSTOMER row exists for the referenced customer and a CUSTOMER row may not be deleted if this will leave any SALE row without a referenced CUSTOMER. Foreign keys are common in relational data bases due to the way they are designed, as will be seen later.

The ORDER-LINE and SUPPLIES relations exist because of M:N relationships. ORDER-LINE is like the intersection record of a network database where QUANTITY-ORDERED and EXTENDED-PRICE are the intersection data. The concatenated key is composed of the keys of the related relations. The SUPPLIES relation is like the link record of a simple network database. In this database, we do not care to know anything about this M:N relationship other than the PRODUCT and VENDOR associations themselves.

In general, a hierarchical or network structure can be decomposed into a relational data model as follows:

(1) Each node in a hierarchical or network structure is isolated into a separate relation.

(2) The primary key of a parent node is incorporated into its child relation to establish the one to many relationship between the parent and its child.

2.6.5 Implementation of the Relational Data Model

As stated earlier, the relational data model is a purely logical view of data. Unlike the hierarchical and network models, whose structure and diagrammatic conventions imply specify physical linkages, in the relational model, we do not know how relationships have been implemented.

We might conclude that, in practice, a wide variety of data structures would be used. Surprisingly this is not the case.

By far, the most common data structure for implementing a relational database is the use of tree-structured indexes (often B-trees) on primary and selected secondary keys. Any attribute that is used to select tuples in a PROJECT or WHERE clause is a possible candidate for indexing. Attributes used to JOIN relations can be indexed; frequently, this greatly reduces the cost to perform a JOIN. To JOIN relations VENDOR and SUPPLIES from figure 24 without an index (or without sorting both relations into order by values for the common attribute), we would have to follow this procedure.

1. Do Until end of VENDOR table.
2. Read next VENDOR tuple.
3. Scan the whole SUPPLIES relation, and if a tuple has the same VENDOR# as the current VENDOR tuple, then create a new RESULT tuple.
4. End Do.
5. Eliminate redundant tuples from relation RESULT.

With an index, step (3) is made much more efficient, since only the SUPPLIES tuples with the same VENDOR#, if any exist, need be retrieved (which is probably a very small percentage for each value of VENDOR#).The DBA cannot optimise the database for all possible query formulations. Thus, for every relation the anticipated volume of different types of queries, updates, and so on is estimated to come up with an anticipated usage pattern. Based on these statistics, decisions on physical organisation are made. For example, it would be inappropriate to provide an access structure (say a B^+ -tree) for every attribute of every relation; these secondary access structures have storage and search overheads.

The DBMS can make use of all the features of the file management system. As most DBMSs have versions that run on different machines and under different operating system environments, the DBMS may support file systems not available under the host machine environment. Thus, every DBMS defines the file and index structures it supports. The DBA chooses the most appropriate file organisation. In the event of changes to usage patterns or to expedite the processing of certain queries, a reorganisation can take place.

A large number of queries requires the joining of two relations. It may be appropriate to keep the joining tuples of the two relations either as linked records or physically grouped into a single record.

We may consider a relation to be implemented in terms of a single (or multiple) file(s) and a tuple of the relation to be a record (or collection of records). For the file, we may define a storage strategy, for example, sequential, indexed, or random, and for each attribute we can define additional access structures.

The most powerful DBMSs allow a great deal of implementation detail to be defined for the relations. The more common but less powerful DBMSs (mostly on microcomputers) allow very simple definitions, for example, indexing on certain attributes (this is usually a B^+ -tree index). Some systems require the index to be regenerated after any modification to the indexing attribute values. Additional commands for sorting and other such operations are also supported. The typical file organisation is plain sequential. (In fact, many micro-based DBMSs confuse a relation or table with a flat sequential file.)

A single relation may be stored in more than one file, i.e., some attributes in one, the rest in others. This is known as **fragmentation**. This may be done to improve the retrieval of certain attribute values; by reducing the size of the tuple in a given file, more tuples can be fetched in a single physical access. The system associates the same internally generated identifier, called the **tuple identifier**, to the different fragments of each tuple. Based on these tuple identifiers a complete tuple is easy to reconstruct.

In addition to making use of the file system, the DBMS must keep track of the details of each relation and its attribute defined in the database. All such information is kept in the directory. The directory can be implemented using a number of system-defined and maintained relations. For each relation, the system may contain a tuple in some system relation, recording the relation name, creator, date, size, storage, structure, and so on. For each attribute of the relation, the system may maintain a tuple recording the relation identifier, attribute name, type, size, and so forth. Different DBMSs keep different amounts of information in the directory relations. However, because the implementation is usually as relations, the same data manipulation language that the DBMS supports can be used to query these relations.

Relational database management systems are often used for highly interactive on-line information systems, which may have many adhoc queries. Fast response, at the expense of extra index space, seems to be the popular choice.

Check Your Progress

1 Define a bit map and explain how it can be used to implement M:N relationship.

.....

.....

.....

.....

2. Define the following terms:

- Inverted list
- Referential Integrity
- Foreign key
- Candidate key
- B-Tree

2.7 SUMMARY

In this unit we reviewed three major traditional data models used in current DBMSs. These three models are hierarchical, network and relational.

The hierarchical model evolved from the file based system. It uses tree type data structure to represent relationship among records. The hierarchical data model restricts each record type to only parent record type. Each parent record type can have any number of children record types.

In a network model, one child record may have more than one parent nodes. A network can be converted into one or more trees by introducing redundant nodes.

The relational model is based on a collection of tables. A table is also called **relation**. A tree or network structure can be converted into a relational structure by separating each node in the data structure into a relation.

The entity-relationship diagrams are useful in representing the relationship among entities. They help in logical database design. We have also presented implementation schemes of each of the traditional database models. You should refer to the next unit for understanding data structure concept for implementation schemes.

2.8 MODEL ANSWERS

1. A bit-map is a matrix created for each relationship. Each row corresponds to the relative record number of a source record and each column corresponds to the relative record number of a target record of the relationship. A 1 bit in a cell for row x and column y means that the records corresponding to row x and column y are associated in this relationship; a zero means no association.

2. - Inverted list — It is a table or list that is organised by secondary key values.
   - Referential integrity — It is an integrity constraint that specifies that the value of an attribute in one relation depends on the value of the same attribute in another relation.
   - Foreign key — If a non-key attribute in one relation appears as the primary key (or part of the primary key) in another relation, it is called foreign key.
   - Candidate key — One or more attributes in a relation that uniquely identify instance of an entity, and therefore, may serve as a primary key in that relation.
   - B-Tree — It is a tree data structure in which all leaves are at the same distance from the root (B stands for **balanced**).

2.9 FURTHER READINGS

1. Bipin C.Desai, *An Introduction to Database Systems*, Galgotia Publication Pvt. Ltd. 1994.

2. Henery F. Korth Abraham Silberschatz, *Database System Concepts*, McGraw Hill International Editions.

UNIT 3 FILE ORGANISATION FOR CONVENTIONAL DBMS

Structure

3.0 INTRODUCTION

Just as arrays, lists, trees and other data structures are used to implement data organisation in main memory, a number of strategies are used to support the organisation of data in

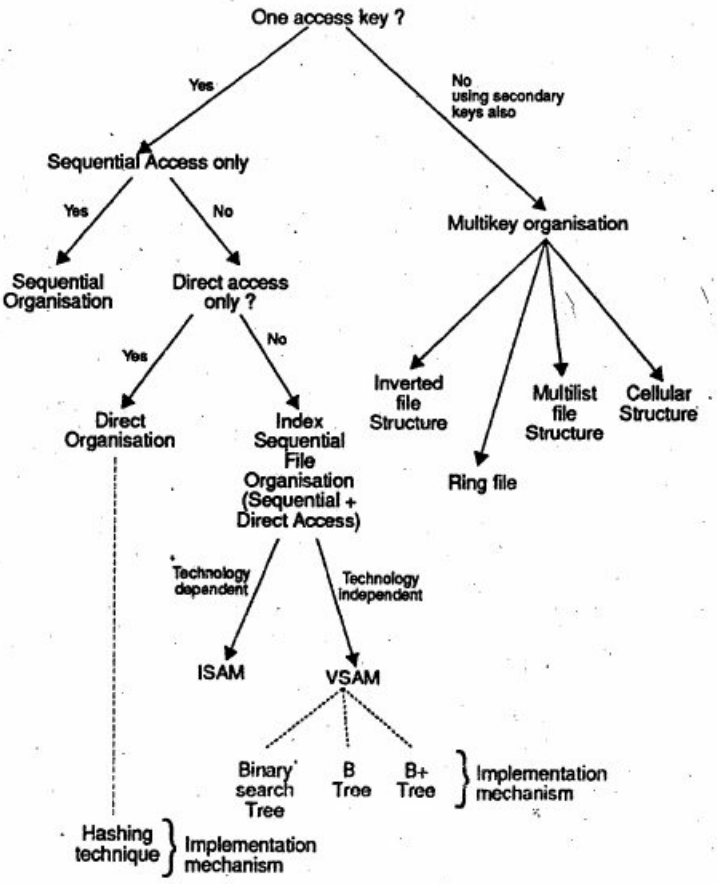


Figure 1 : File organisation techniques

secondary memory. In this unit we will look at different strategies of organizing data in the secondary memory. In this unit, we are concerned with obtaining data representation for files on external storage devices so that required functions (e.g. retrieval, update) may be carried out efficiently. The particular organisation most suitable for any application will depend upon such factors as the kind of external storage available, types of queries allowed, number of keys, mode of retrieval and mode of update. The figure 1 illustrates different file organisations based on an access key.

3.1 OBJECTIVES

After going through this unit you will be able to:

- define what is a file organisation
- list file organisation techniques
- discuss implementation techniques of indexing through tree-structure
- discuss implementation of direct file organisation
- discuss implementation of multikey file organisation
- discuss trade-off and comparison among file organisation techniques

3.2 FILE ORGANISATION

Precise definition of each of these technique will be presented later in this unit.

The technique used to represent and store the records on a file is called the file organisation. The four fundamental file organisation techniques that we will discuss are the following:

1. Sequential
2. Relative
3. Indexed sequential
4. Multi-key

There are two basic ways, that the file organisation techniques differ. First, the organisation determines the **file's record sequencing,** which is the physical ordering of the records in storage.

Second, the file organisation determines the set of operation necessary to find particular records. Individual records are typically identified by having particular values in search-key fields. This data field may or may not have duplicate values in file, the field can be a group or elementary item. Some file organisation techniques provide rapid accessibility on a variety of search key; other techniques support direct access only on the value of a single one.

The organisation most appropriate for a particular file is determined by the operational characteristics of the storage medium used an the nature of the operations to be performed on the data. The most important characteristic of a storage device that influences selection of a storage device once the appropriate file organisation technique has been determined) is whether the device allows **direct access** to particular record occurrences without accessing all physically prior record occurrences that are stored on the device, or allows only **sequential access** to record occurrences. Magnetic disks are examples of direct access storage devices (abbreviated DASD's); magnetic tapes are examples of sequential storage devices.

3.3 SEQUENTIAL FILE ORGANISATION

The most basic way to organise the collection of records that from a file is to use sequential organisation. In a sequentially organised file records are written consecutively when the file is created and must be accessed consecutively when the file is later used for input (figure 2).

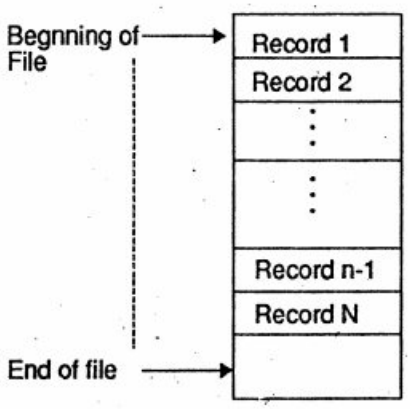


Figure 2 : Structure of sequential file

In a sequential file, records are maintained in the logical sequence of their primary key values. The processing of a sequential file is conceptually simple but inefficient for random access. However, if access to the file is strictly sequential, a sequential file is suitable. A sequential file could be stored on a sequential storage device such as a magnetic tape.

Search for a given record in a sequential file requires, on average, access to half the records in the file. Consider a system where the file is stored on a direct access device such as a disk. Suppose the key value is separated from the rest of the record and a pointer is used to indicate the location of the record. In such a system, the device may scan over the key values at rotation speeds and only read in the desired record. A binary or logarithmic search technique may also be used to search for a record. In this method, the cylinder on which the required record is stored is located by a series of decreasing head movements. The search, having been localised to a cylinder, may require the reading of half the tracks, on average, in the case where keys are embedded in the physical records, or require only a scan over the tracks in the case where keys are also stored separately.

Updating usually requires the creation of a new file. To maintain file sequence, records are copied to the point where amendment is required. The changes are then made and copied into the new file. Following this, the remaining records in the original file are copied to the new file. This method of updating a sequential file creates an automatic backup copy. It permits updates of the type U_1 through U_4.

Addition can be handled in a manner similar to updating. Adding a record necessitates the shifting of all records from the appropriate point to the end of file to create space for the new record. Inversely, deletion of a record requires a compression of the file space, achieved by the shifting of records. Changes to an existing record may also require shifting if the record size expands or shrinks.

The basic advantage offered by a sequential file is the ease of access to the next record, the simplicity of organisation and the absence of auxiliary data structures. However, replies to simple queries are time consuming for large files. Updates, as seen above, usually require the creation of a new file. A single update is an expensive proposition if a new file must be created. To reduce the cost per update, all such requests are batched, sorted in the order of the sequential file, and then used to update the sequential file in a single pass. Such a file, containing the updates to be made to a sequential file, is sometimes referred to a **transaction file.**

In the batched mode of updating, a transaction file of update records is made and then sorted in the sequence of the sequential file. The update process requires the examination of each individual record in the original sequential file (the **old master file**). Records requiring no changes are copied directly to a new file (the **new master file**); records requiring one or more changes are written into the new master file only after all necessary changes have been made. Insertions of new records are made in the proper sequence. They are written into the new master file at the appropriate place. Records to be deleted are not copied to the new master file. A big advantage of this method of update is the creation of an automatic backup copy. The new master file can always be recreated by processing the old master file and the transaction file.

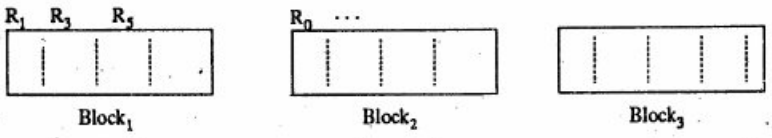


Figure 3 : A file with empty spaces for record insertions

A possible method of reducing the creation of a new file at each update run is to create the original file with "holes" (space left for the addition of new records, as shown in the last figure). As such, if a block could hold K records, then at initial creation it is made to contain only L * K records, where $0 < L \leq 1$ is known as the loading factor. Additional space may also be earmarked for records that may "overflow" their blocks, e.g., if the record ri logically belongs to block Bi but the physical block Bi does not contain the requisite free space. This additional free space is known as the overflow area. A similar technique is employed in index-sequential files.

3.4 INDEX-SEQUENTIAL FILE ORGANISATION

The retrieval of a record from a sequential file, on average, requires access to half the records in the file, making such enquiries not only inefficient but very time consuming for large files. To improve the query response time of a sequential file, a type of indexing technique can be added.

An index is a set of y, address pairs. Indexing associates a set of objects to a set of orderable quantities, which are usually smaller in number or their properties provide a mechanism for faster search. The purpose of indexing is to expedite the search process. Indexes created from a sequential (or sorted) set of primary keys are referred to as index sequential. Although the indices and the data blocks are held together physically, we distinguish between them logically. We shall use the term **index file** to describe the indexes and **data file** to refer to the data records. The index is usually small enough to be read into the processor memory.

A sequential (or sorted on primary keys) file that is indexed is called an **index sequential file.** The index provides for random access to records, while the sequential nature of the file provides easy access to the subsequent records as well as sequential processing. An additional feature of this file system is the **overflow area.** This feature provides additional space for record addition without necessitating the creation of a new file. Before starting discussion on index sequential file structure, let us, discuss types of indexes.

3.4.1 Types of Indexes

The idea behind an index access structure is similar to that behind the indexes used commonly in textbooks. A textbook index lists important terms at the end of the book in alphabetic order. Along with each term, a list of page numbers where the term appears is given. We can search the index to find a list of **addresses** - page numbers in this case - and use these addresses to locate the term in the textbook by searching the specified pages. The alternative, if no other guidance is given, is to sift slowly through the whole textbooks word by word to find the term we are interested in, which corresponds to doing a linear search on a file. Of course, most books do have additional information, such as chapter and section titles, which can help us find a term without having to search through the whole book. However, the index is the only exact indication of where each term occurs in the book.

An index is usually defined on a single field of a file, called an **indexing field.** The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain a record with that field value. The values in the index are ordered so that we can do a binary search on the index. The index file is much smaller than the data file, so searching the index using binary search is reasonably efficient. Multilevel indexing does away with the need for binary search at the expense of creating indexes to the index itself!

There are several types of indexes. **A primary index** is an index specified on the ordering key field of an ordered file of records. Recall that an ordering key field is used to physically order the file records on disk, and every record has a unique value for that field. If the

ordering field is not a key field that is, several records in te file can have the same value for the ordering field another type of index, called a **clustering index,** can be used. Notice that a file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, but not both. A third type of index, called a **secondary index,** can be specified on any nonordering field of a file. A file can have several secondary indexes in addition to its primary access method. In the next three subsections we discuss these three types of indexes.

Primary Indexes

A **primary index** is an ordered file whose records are of fixed length with two fields. The first field is of the same data types as the ordering key field of the data file, and the second field is a pointer to a disk blokc - a block address. The ordering key field is called the **primary key** of the data file.There is one **index entry** (or **index record**) in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to other block as its two field values. We will refer to the two field values of index entry i as K(i), P(i).

To create a primary index on the ordered file shown in fiugre 4, we use the Name field as the primary key, because that is the ordering key field fo the file (assuming that each value of

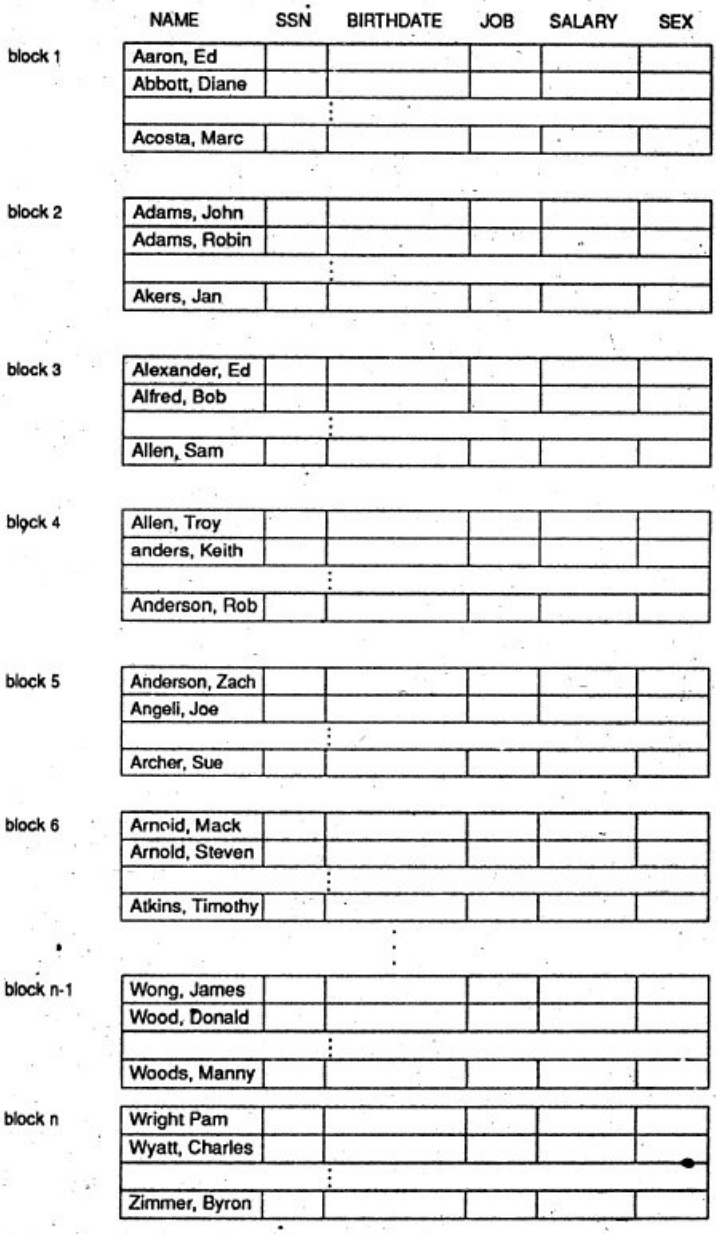


Figure 4 : Some blocks on an ordered (sequential) file of EMPLOYEE records with NAME as the ordering field

NAME is unique). Each entry in the index will have a NAME value and a pointer. The first three index entries would be:

<K(1) = (Aaron, Ed), P(1)= address of block 1>

<K(2) = (Adams, John), P(1) = address of block 2 >

<K(3) = (Alexander,Ed), P(3) = address of block 3>

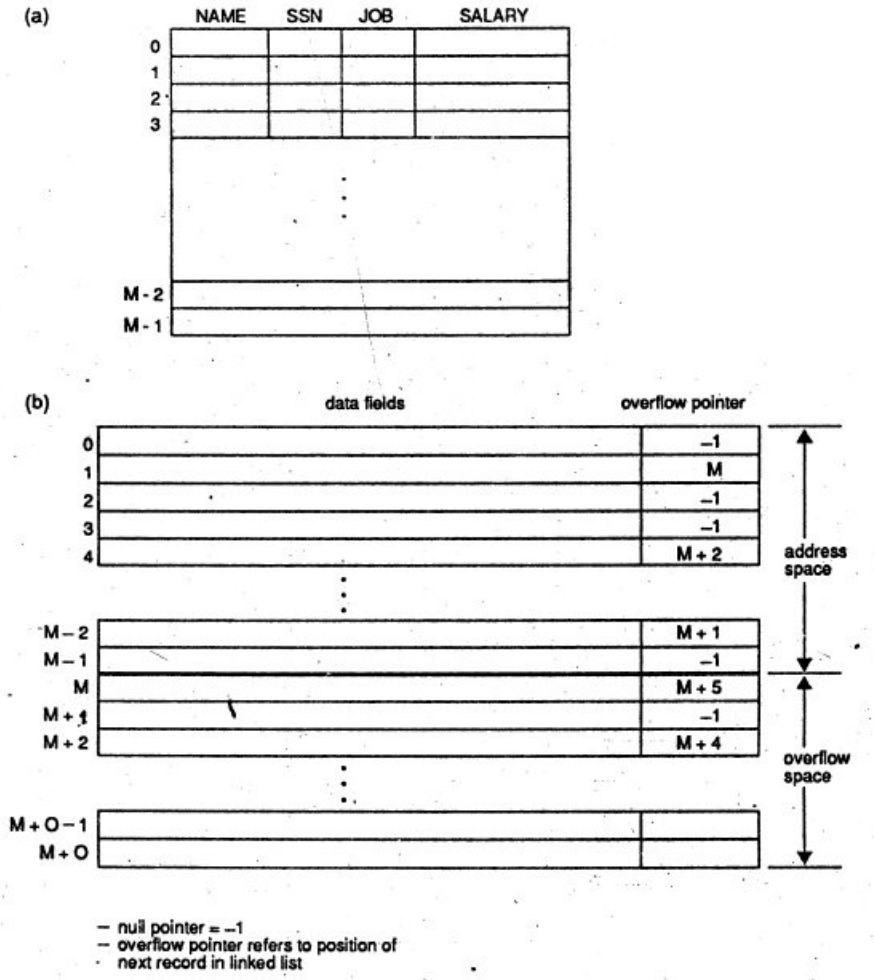


Figure 5 : Illustrating internal hashing data structures.
(a) Array of M positions for use in hashing. (b) Collision resolution by chaining of records.

Figure 6 illustrates this primary index. The total number of entries in the index will be the same as the number of disk blocks in the ordered data file. The first record in each block of the data file is called the **anchor record** of the block, or simply the **block anchor** (a scheme similar to the one described here can be used, with the last record in each block, rather than the first, as the block anchor. A primary index is an example of what is called a **nondense index** because it includes an entry for each disk block of the data file rather than for every record in the data file. A **dense index**, on the other hand, contains an entry for every record in the file.

The index file for a primary index needs substantially fewer blocks than the data file for two reasons. First, there are fewer index entries than there are records in the data file because an entry exists for each whole block of the data file rather than for each record. Second, each index entry is typically smaller in size than a data record because it has only two fields, so

more index entries than data records will fit in one block. A binary search on the index file will hence require fewer block accesses than a binary search on the data file.

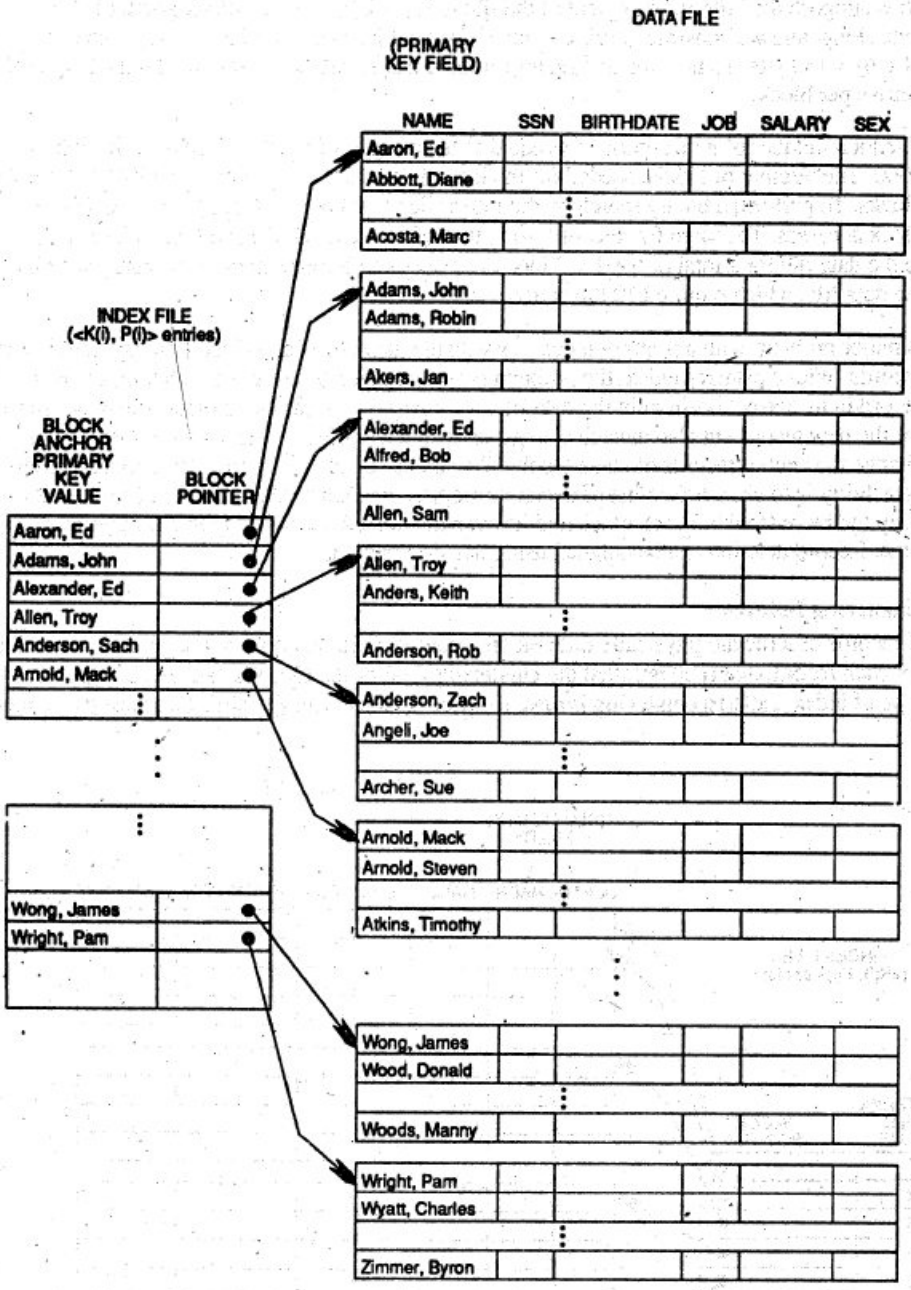


Figure 6 : Primary index on the ordering key field of the file shown in figure 5

A record whose primary key value is K will be in the block whose address is P(i), where $Ki \leq K \leq (i + 1)$. The ith block in the data file contains all such records because of the physical ordering of the file records on the primary key field, we do a binary search on the index file to find the appropriate index entry i, then retrieve the data file block whose address is P(i). Notice that the above formula would not be correct if the data file was ordered on a nonkey field that allows multiple records to have the same ordering field value. In that case the same index value as that in the block anchor could be repeated in the last records of the previous block. Example 1 illustrates the saving in block accesses when using an index to search for a record.

Example 1 : Suppose we have an ordered file with r = 30,000 records stored on a disk with block size B = 1024 bytes. File records are of fixed size and unspanned with record length R = 100 bytes. The blocking factor for the file would be bfr $\lfloor (B/R) \rfloor = \lfloor (1024/100) \rfloor = 10$ records per block. The number of blocks needed for the file is $b = \lceil (r/bfr) \rceil = \lceil (30,000/10) \rceil =$

3000 blocks. A binary search on the data file would need approximately $\lceil (\log_2 b) \rceil = \lceil (\log_2 3000) \rceil = 12$ block accesses.

Now suppose the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we construct a primary index for the file. The size of each index entry is $R_i = (9 + 6) = 15$ bytes, so the blocking factor for the index is $bfr_i = \lfloor (B/R) \rfloor = \lfloor (1024/15) \rfloor = 68$ entries per block.

The total number of index entries r_i is equal to the number of blocks in the data file, which is 3000. The number of blocks needed for the index is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (3000/68) \rceil = 45$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 45) \rceil = 6$ block accesses. To search for a record using the index, we need one additional block access to the data file for a total of 6 + 1 = 7 block accesses - an improvement over binary search on the data file, which required 12 block accesses.

A major problem with a primary index - as with any ordered file - is insertion and deletion of records. With a primary index, the problem is compounded because if we attempt to insert a record in its correct position in the data file, we not only have to move records to make space for the new record but also have to change some index entries because moving records will change the anchor records of some blocks. We can use an unordered overflow file. Another possibility is to use a linked list of overflow records for each block in the data file. We can keep the records within each block and its overflow linked list sorted to improve retrieval time. Record deletion can be handled using deletion markers.

Clustering Indexes

If records of a file are physically ordered on a nonkey field that does not have a distinct value for each record, that field is called the **clustering field** of the file. We can create a different type of index, called a **clustering index,** to speed up retrieval of records that have the same

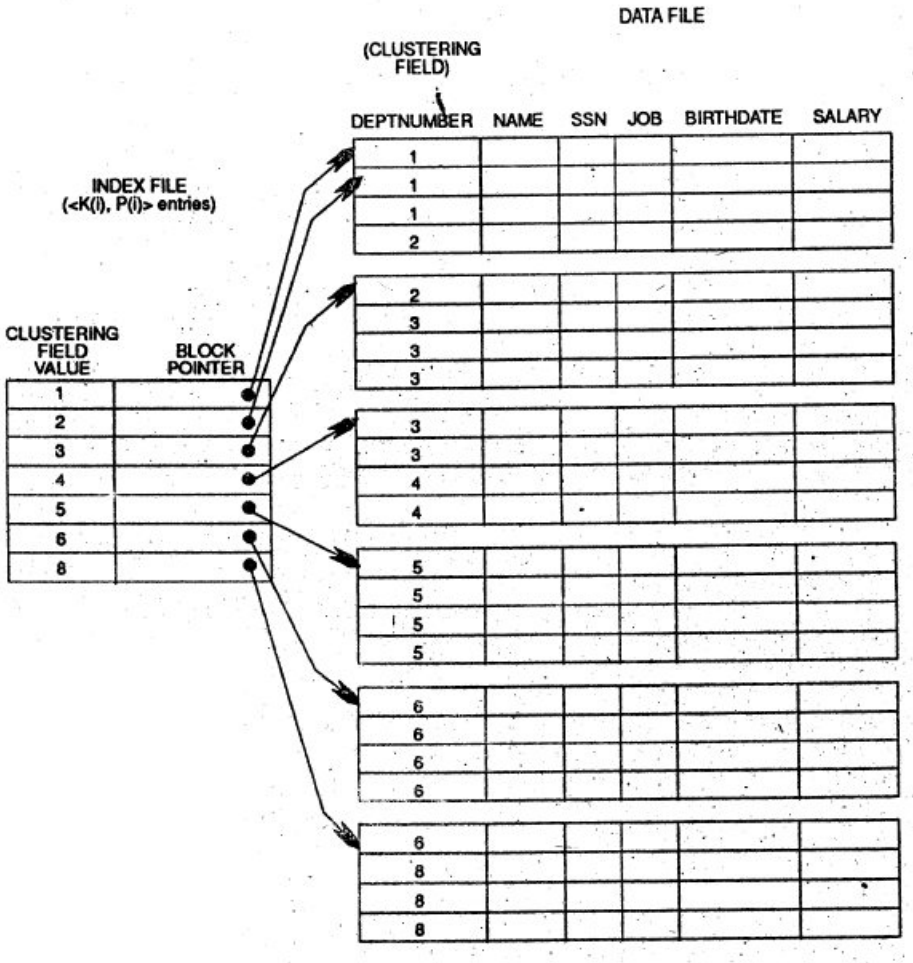


Figure 7 : A clustering index on the DEPTNUMBER ordering field of an EMPLOYEE file

value for the clustering field. This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.

A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file and the second field is a block pointer. There is one entry in the clustering index for each distinct value of the clustering field, containing that value and a pointer to the first block in the data file that has a record with that value for its

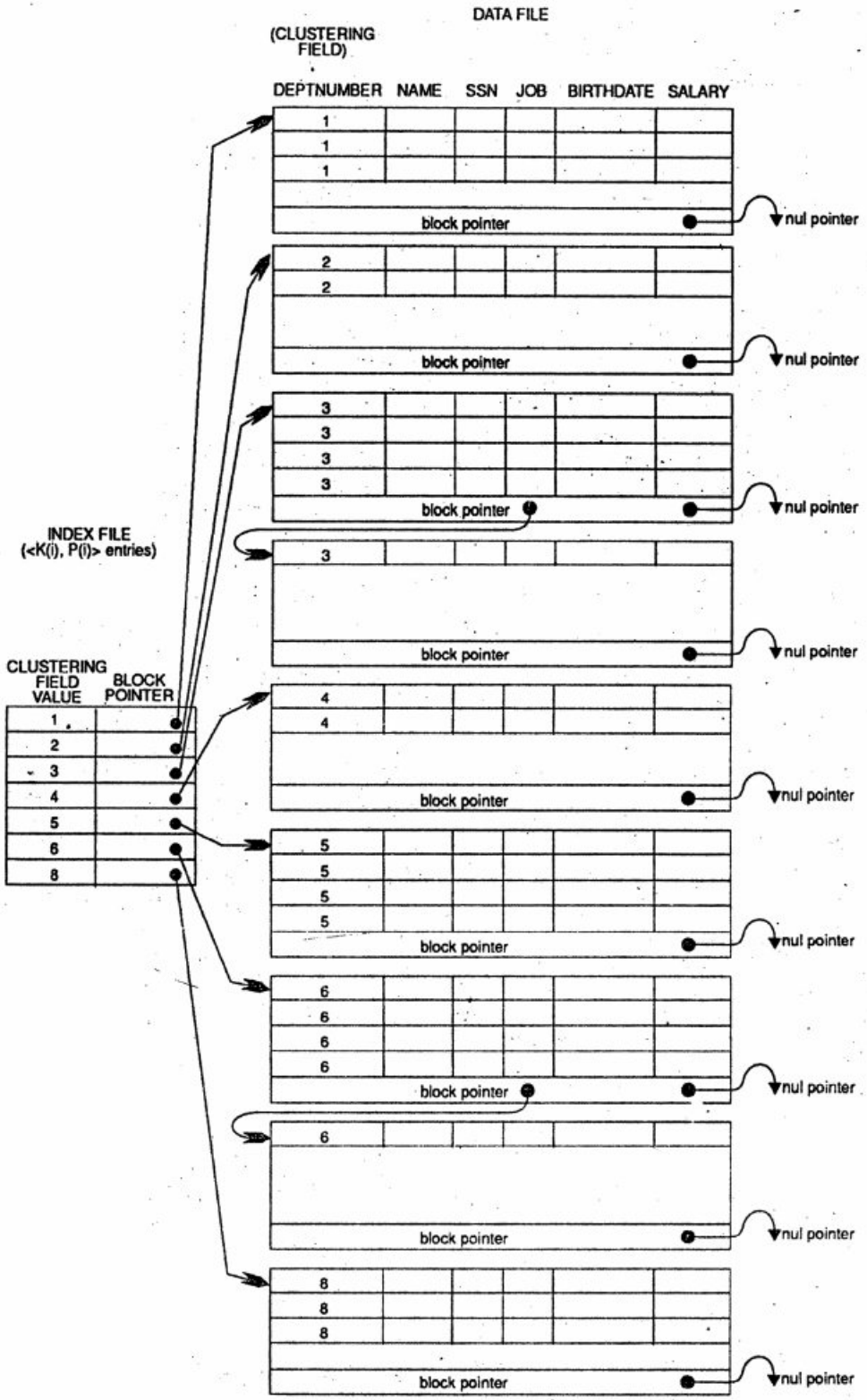


Figure 8 : Clustering index with separate blocks for each group of records with the same value for the clustering field

clustering field. Figure 7 shows an example of a data file with a clustering index. Note the record insertion and record deletion still cause considerable problems because the data records are physically ordered. To alleviate the problem of insertion, it is common to reserve a whole block for each value of the clustering field; all records with that value are placed in the block. If more than one block is needed to store the records for a particular value, additional blocks are allocated and linked together. This makes insertion and deletion relatively straightforward. Figure 8 shows this scheme.

A clustering index is another example of a nondense index because it has an entry for every distinct value of the indexing field rather than for evrey record in the file.

Secondary Indexes

A **secondary index** also is an ordered file with two fields, and, as in the other indexes, the second field is a pointer to a disk block. The first field is of the same data type as some nonordering field of the data file. The field on which the secondary index is constructed is called an **indexing field** of the file, whether its values are distinct for every record or not. There can be many secondary indexes, and hence indexing fields, for the same file.

We first consider a secondary index on a key field - a field having a distinct value for every record in the data file. Such a field is sometimes called a **secondary key** for the file. In this case there is one index entry for each record in the data file, which has the value of the secondary key for the record and a pointer to the block in which the record is stored. A secondary index on a key field is a **dense** index because it contains one entry for each record in the data file.

We again refer to the two field values of index entry i as K(i), P(i). The entries are **ordered** by value of K(i), so we can use binary search on the index. Because the records of the data file are not physically ordered by values of the secondary key field, we cannot use block

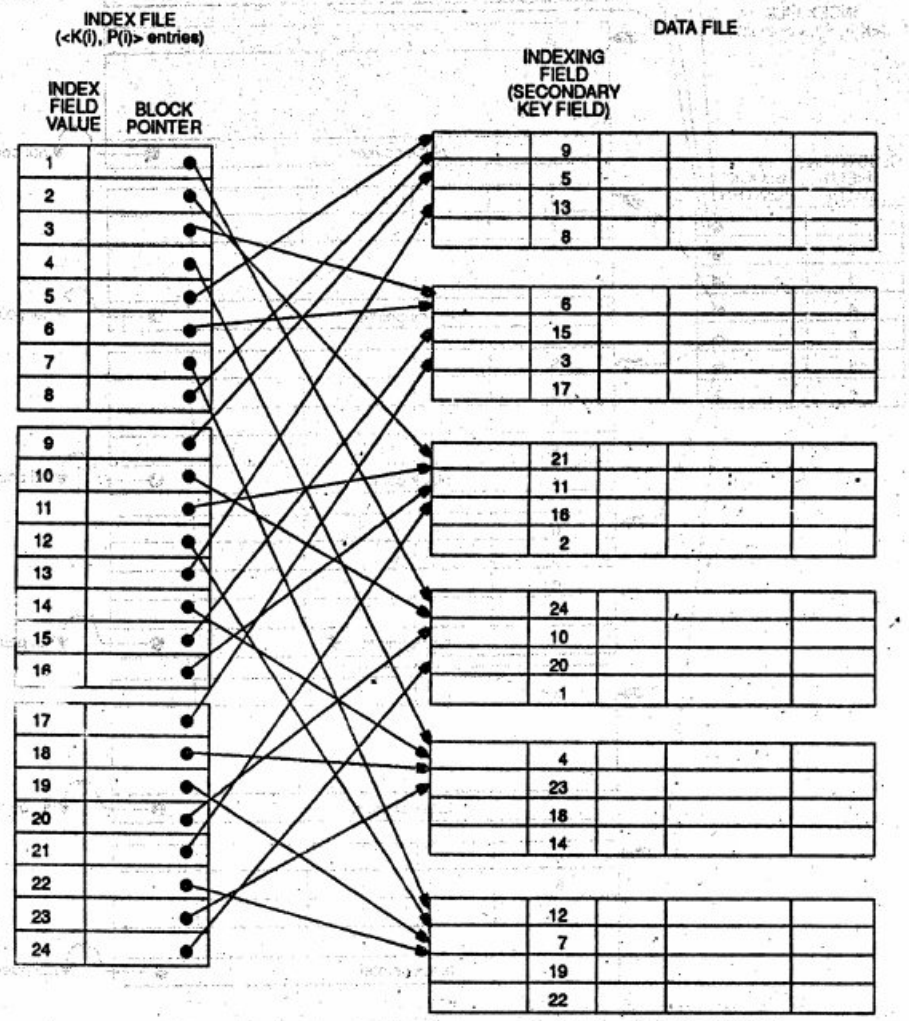


Figure 9 : A dense secondary index on a nonordering key field of a file

anchors. That is why an index entry is created for each record in the data file rather than for each block as in the case of a primary index. Figure 9 illustrates a secondary index on a key attribute of a data file. Notice that in figure 9 the pointers P(i) in the index entries are block pointers, not record pointers. Once the appropriate block is transferred to main memory, a search for the desired record within the block can be carried out.

A secondary index will usualy need substantially more storage space than a primary index becaue of its larger number of entries. However, the improvement in search time for an arbitrary record is much greater for a secondary index than it is for a primary index, because we would have to do a linear search on the data file if the secondary index did not exist. For a primary index, we could still use binary search on the main file even if the index did not exist because the records are physically ordered by the primary key field. Example 2 illustrates the improvement in number of blocks accessed when using a secondary index to search for a record.

Example 2 : Consider the file of Example 1 with r = 30,000 fixed- length records of size R = 100 bytes stored on a disk with block size B = 1024 bytes. The file has b = 3000 blocks as calculated in Example 1. To do a linear search on the file, we would require b/2 = 3000/2 = 1500 block accesses on the average.

Suppose we construct a secondary index on a nonordering key field of the file that is V = 9 bytes long. As in Example 1, a block pointer is P = 6 bytes long, so each index entry is R_i = (9 + 6) = 15 bytes, and the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (1024/15) \rfloor$ = 68 entries per block. In a dense secondary index such as this, the total number of index entries is r_i is equal to the number of records in the data file, which is 30,000. The number of blocks needed for the index is hence $b_i = (r_i/bfr_i) = (30,000/68) = 442$ blocks.

Compare this to the 45 blocks needed by the nondense primary index in Example 1.

A binary search on this secondary index needs $(\log_2 b_i) = (\log_2 442) = 9$ block accesses. To search for a record using the index, we need an additional block access to the data file for a total of 9 + 1 = 10 block accesses - a vast improvement over the 1500 block accesses needed on the average for a linear search.

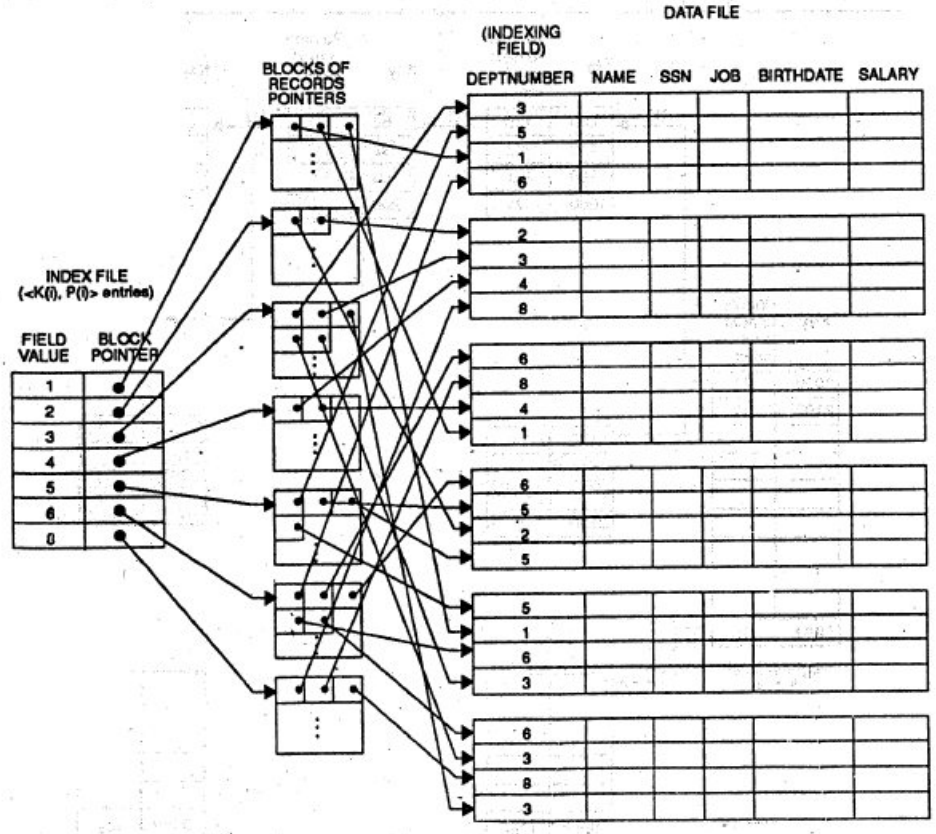


Figure 10 : A secondary index on a nonkey field implemented using one level of indirection so that index entries are fixed length and have unique field values .

We can also create a secondary index on a nonkey field of a file. In this case numerous records in the data file can have the same value for the indexing field. There are several options for implementing such an index:

- Option 1 is to include several index entries with the same K(i) value - one for each record. This would be a dense index.
- Option 2 is to have variable-length records for the index entries, with a repeating field for the pointer. We keep a list of pointers (i,1),.....P(i,k) in the indexentry for K(i) - one pointer to each block that contains a record whose indexing field value equals K(i). In either option 1 or option 2, the binary search algorithm on the index must be modified appropriately.
- Option 3, which is used commonly, is to keep the index entries themselves at a fixed length and have a single entry for each index field value, but create an extra level of indirection to handle the multiple pointers. In this scheme, which is nondense, the ponter P(i) in index entry (i), P(i) points to a block of record pointers; each record pointer in that block points to one of the data file reords with a value K(i) for the indexing field. If some value K(i) has too many records, so that their record pointers cannot fit in a single disk block, a linked list of blocks can be used. This technique is illustrated in figure 10. Retrieval via the index requires an additional block access because of the extra level, but the algorithms for searching the index and, more important, for insertion of new records in the data file are straightforward. In addition, retrievals on complex selection conditions may be handled by referring to the pointers without having to retrieve many unnecessary file records.

Notice that a secondary index provides a **logical ordering** on the records by the indexing field. If we access the records in order of the entries in the secondary index, we get them in order of the indexing field.

Multilevel Indexing Schemes : Basic Technique

In a full indexing scheme, the address of every record is maintained in the index. For a small file, this index would be small and can be processed very efficiently in main memory. For a

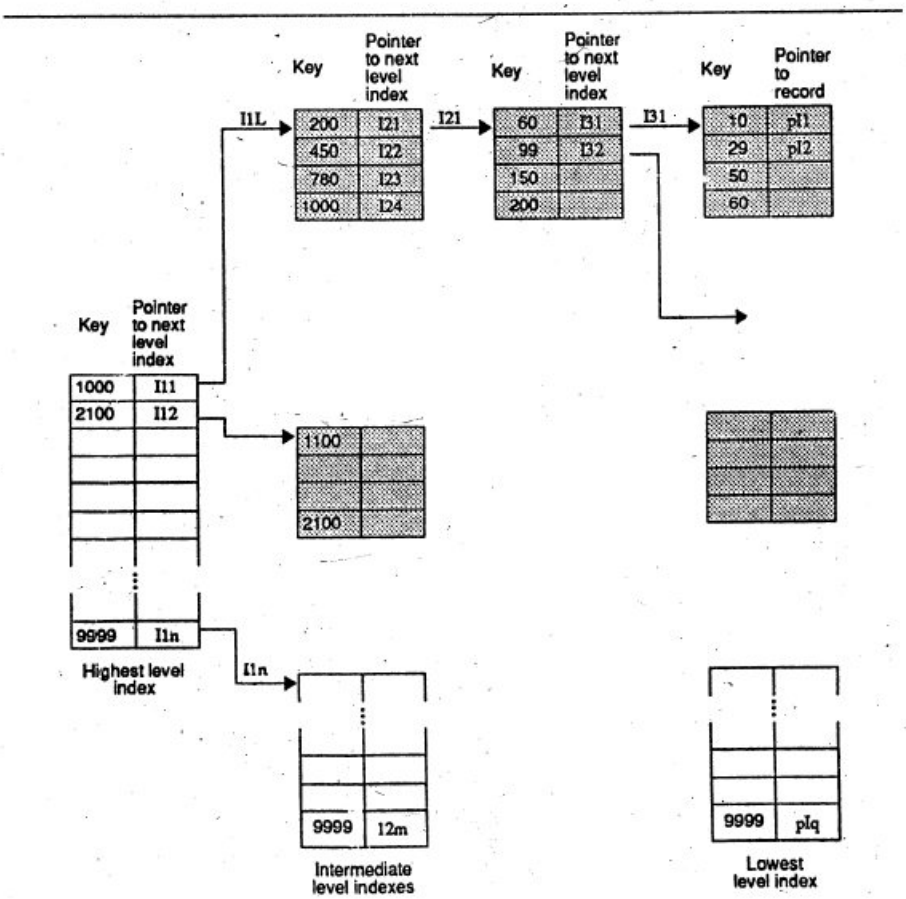


Figure 11 : Hierarchy of indexes

large file, the index's size would pose problems. It is possible to create a hierarchy of indexes with the lowest level index pointing to the records, while the higher level indexes point to the indexes below them (figure 11). The higher level indices are small and can be moved to main memory, allowing the search to be localised to one of the larger lower level indices.

The lowest level index consists of the <key, address> pair for each record in the file; this is costly in terms of space. Updates of records require changes to the index file as well as the data file. Insertion of a record requires that its <key, address> pair be inserted in the index at the correct point, while deletion of a record requires that the <key, address> pair be removed from the index. Therefore, maintenance of the index is also expensive. In the simplest case, updates of variable length records require that changes be made to the address field of the record entry. In a variation of this scheme, the address value in the lowest level index entry points to a block of records and the key value represents the highest key value of recods in this block. Another variation of this scheme is described in the next section.

3.4.2 Structure of Index Sequential Files

An index-sequential file consists of the data plus one or more levels of indexes. When inserting a record, we have to maintain the sequence of records and this may necessitate shifting subsequent records. For a large file this is a costly and inefficient process. Instead, the records that overflow their logical area are shifted into a designated overflow area and a pointer is provided in the logical area or associated index entry points to the overflow location. This is illustrated below (figure 12). Record 165 is inserted in the original logical block causing a record to be moved to an overflow block.

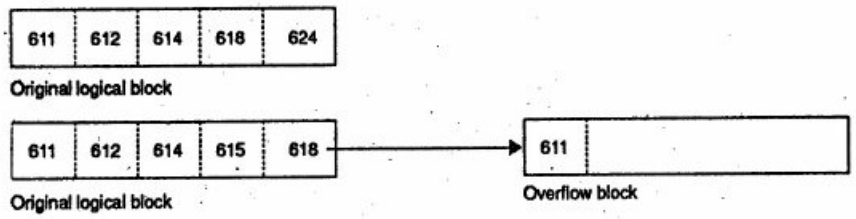


Figure 12 : Overflow of record

Multiple records belonging to the same logical area may be chained to maintained logical sequencing. When records are forced into the overflow areas as a result of insertion, the insertion process is simplified, but the search time is increased. Deletion of records from index-sequential files creates logical gaps; the records are not physically removed but only flagged as having been deleted. If there were a number of deletions, we may have a great amount of unused space.

An index-sequential file is therefore made up of the following components:

1. A primary data storage area. In certain systems this area may have unused spaces embedded within it to permit addition of records. It may also include records that have been marked as having been deleted.
2. Overflow area(s). This permits the addition of records to the files. A number of schemes exist for the incorporation of records in these areas into the expected logical sequence.
3. A hierarcy of indices. In a random enquiry or update, the physical location of the desired record is obtained by accessing these indices.

The primary data area contains the records written by the users' programs. The records are written in data blocks in ascending key sequence. These data blocks are in turn stored in ascending sequence in the primary data area. The data blocks are sequenced by the highest key of the logical records contained in them.

There are several approaches to structuring both the index and sequential data portion of an indexed sequential file. The most common approach is to build the index as a tree of key values. The tree is typicaly a variation on B-tree which we will discuss later. The other common approach is to build the index based on the physical layout of the data in storage.

The important technique for building index based on the physical layout of the data in storage is ISAM (Index sequential access method) which we will discuss.

Physical Data Organisation Under ISAM

When a record is stored by ISAM, its record key must be one of the fields in the record. The records themselves are first sorted by record key into ascending order before they are stored on one or more disk drives. ISAM will always maintain the records in this sorted order. Each record is stored on one of the tracks of a disk. Those records that follow it in sorted sequence are placed directly after it on the same track or, if room does not permit, are spilled over onto the next track in the same cylinder. In other words, they are dropped down to the next platter surface. The arm does not move; looking downwared, the next head is selected eletronically. Since the tracks on a cylinder are labelled 0, 1, 2, ... the records that follow those on track 1 are placed on track 2. Track 0 is the next file cylinder. The cylinders are also labelled 0, 1, 2, ...

Figure 13 shows two cylinders of records, but only their keys are shown. Note that the keys are in ascending sequence throughout their storage on both cylinders. We have not shown record 0 on either cylinder, as this is used by ISAM for control. Of course, the number of tracks on each cylinder is a function of the size of the disk pack.

When ISAM retrieves a record, it needs to know the cylinder, the track address, and the record key. These are the components that must make up the directory entries for the ISAM file. In ISAM a directory is called an index. For example, if a directory entry for record 1500 gave cylinder 9 and track 3, then ISAM would select cylinder 9. The read head associated with track 3 would then be activated. The bottom side of the top platter is usually track 0 because the top being exposed is subject to damage; therefore, the read head selected would be that for the top side of the third platter, as shown in figure 14. Of course, the required record might be one of the many records stored on track 3. Rotation of the drive would eventually bring the required record under the read head. The desired record is identified by its record key.

Because the records in an ISAM file are kept in sorted order by record key, it is not necessary to have a directory entry for every single record. It is sufficient to know the largest record key on every track of te file. For example, suppose the largest key on the track 3 is 100 and the largest on track 4 is 200. A record with key 175, if it exists in the file at all, must be on track 4. It cannot be on track 3 as the largest key on that track is 100.

The most obvious place to keep the directory for each cylinder on the file is, of course, on the cylinder itself, and it is on track 0 of each cylinder that ISAM keeps its directory. This directory is known as a track index; it contains the largest key on every track and the hardware address of that track. Figure 15 shows a typical track index for one cylinder of a file. In this cylinder, for example, 400 is shown to be the largest key on track 3 and 700 the largest key on the cylinder. Later we will see that this directory is slightly more complicated. This will be clarified when we discuss how ISAM keeps track of records that are added to

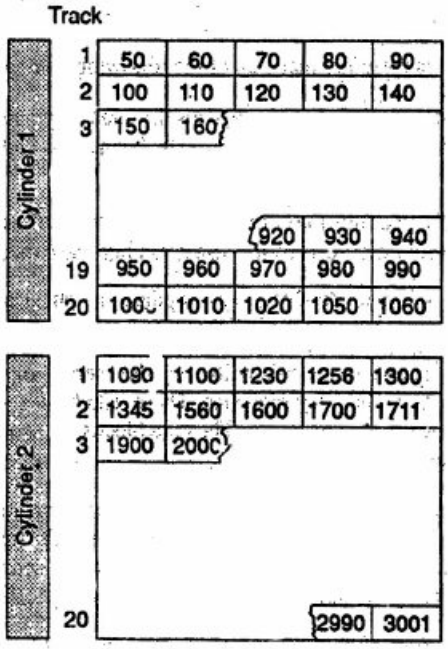


Figure 13 : Record Storage

the file after its original creation. For the moment, this simplificd version of the directory is more than sufficient.

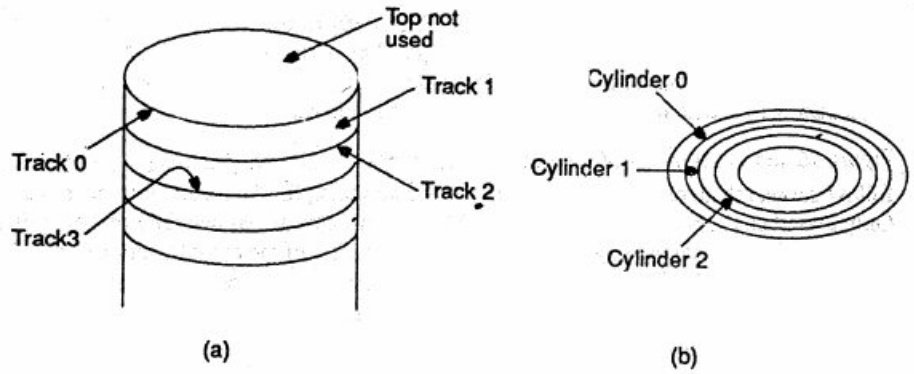


Figure 14 : Disk organisation (a) disk; (b) top view of platter

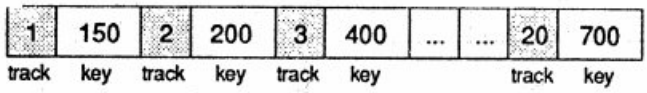
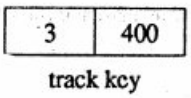
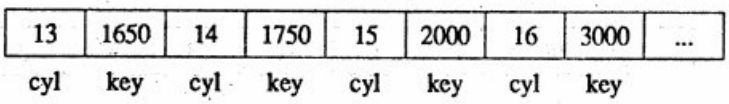


Figure 15 : Track Index

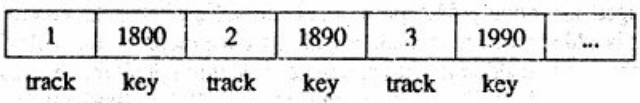
How does the ISAM use this directory to find a record on the file? First it positions the read/write mechanism over the appropriate cylinder and selects track 0. Let us suppose that the index on track 0 has the entries shown below in figure 15 and the system seeks the key 350. The entry indicates that the record, if it is to be found, will be on track 3. The read head for track 3 is selected and the rotation of the drive will eventually bring the record with key 350, if it exists, under this read head. The fact that the index for this cylinder is on the cylinder itself means that no additional movement of the read/write mechanism is necessary.



When an ISAM file is spread over several cylinders, there is more than one track index. A track index is placed on track 0 of each cylinder used by the file. There remains then in this case a further problem. When a record is being sought, which track index should be examined? Not surprisingly, ISAM keeps a cylinder index with an entry for each of its track indexes. Each entry in this index specifies the address of every track index and the largest entry in each track index. In other words, the cylinder index has an entry for each cylinder of the file and the largest entry on that cylinder. The following is a typical cylinder index:



This cylinder index shows that on cylinder 15 the largest key that will be found is 2000. If ISAM is seeking record 1886, an examination of this cylinder index reveals that the record, if it exists, can be found on cylinder 15. The read/write mechanism moves to cylinder 15, selects track 0, and consults the track index. If that track index is then track 2 is selected.



The cylinder index is not associated with any particular cylinder of the file and is stored in a separate area or on another disk altogether. This are is referred to as the cylinder area. The file itself along with the track indexes is called the prime area.

Sometimes a file may be very large, even extending across several disk drives. In this case hundreds of cylinders may be involved causing the cylinder index itself to be several tracks or cylinders in size. In this eventuality, ISAM may even create an index of the cylinder index. Such an index is called a master index. Each entry of the master index then points to a track of the cylinder index and specifies the largest key given on this track of the cylinder index. Even another master index might be made of this index. Perhaps now the reader can appreciate why the name "Indexed" is the first word in ISAM.

Figure 16 shows segments of each of the three types of indexes. The reader should try to follow the search algorithm, beginning at the master index, for the location of record 45. Only two tracks of both the master index and cylinder index are shown. The first entry in the master index says that the largest key mentioned in track 1 of the cylinder index is 211. The first entry on track 1 of the cylinder index shows that 95 is the largest key on the track index on cylinder 6. Checking the track index of cylinder 6 shows that record 45 is located on track 2. Record 45 also happens to be the largest record key on track 2.

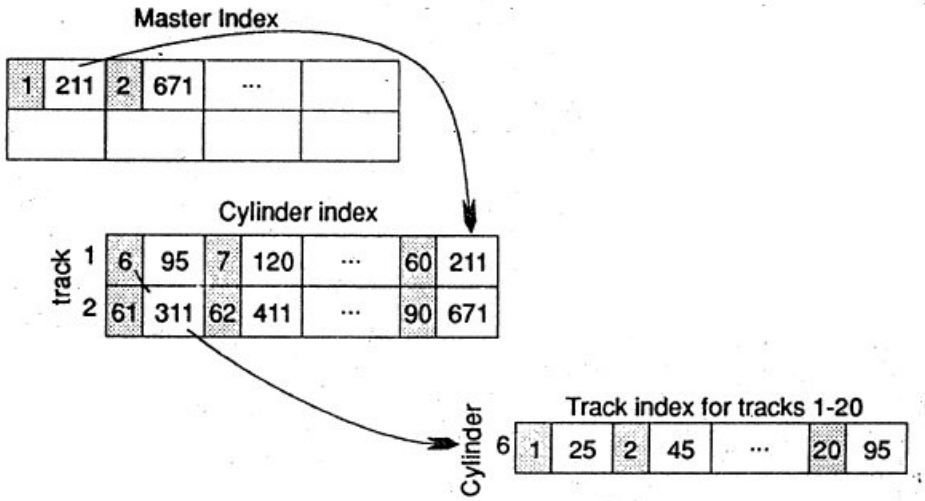


Figure 16 : Search algorithm through three indexes Overflow Records in ISAM

Overflow Records in ISAM

Unlike Relative I-O, which does not permit the addition of a new record to a file unless an empty slot is available. ISAM allows any number of new records to be added to an existing file. The number is limited by the availability of sufficient storage space. As mentioned earlier, ISAM also maintains the original ordering of the sorted file. Any record to be added is inserted into the file at an appopriate place. To accomplish this insertion, room must be made available for the record on its track by shifting each record that logically follows the one to be added forward on the track and dropping the last record on the track off the end. For example, if the records at the end of a track are

...	26	28	30	31	33	35	37

and record 34 is to be added, then the track will be changed to

...	26	28	30	31	33	34	35

and record 37 will be dropped off the end. The track's highest key is now 35 and the track index is changed accordingly. The question, of course, is what to do with record 37 that was dropped. If it is added to the next track, it will cause the record at the end of that track to be dropped off the end and a domino effect will cascade through all the records on the file. In each case, the track index will need to be changed as will the cylinder index when the last record on the cylinder is forced off. To avoid this problem, the record dropped off the original track is removed from the file and placed in an overflow area. This overflow area may be on another disk unit, elsewhere on the same disk unit, or even perhaps on the same cylinder if several tracks on each cylinder are set aside and designated for overflow use. The exact placement of the overflow area is determined when space for the file is requested from the operating system.

Earlier, it was noted that a simplified version of the track index had been presented. This version can now be upgraded. In actual fact, there are two entries for each track on a given cylinder. We shall designate them as "N" and "O" entries, where "N" denotes a normal entry and "O" an overflow entry. Before overflow records are added to the file, both entries are the same. For example, the track index for cylinder 6 of a file might appear as

N		O		N		O		N		
1	120	1	120	2	200	2	200	3	250	...

In this case, both the N and O entries for track 2 designate that 200 is the largest key on this track. Suppose, in fact, that track 2 contains the following records (only the keys shown):

130	145	150	...	180	190	200

As indicated by the track index, the largest key to be found on track 2 is 200. Now suppose record 185 is to be added to this track forcing record 200 off the end into the overflow area. Track 2 now becomes

130	145	150	...	180	185	190

As the largest key on track 2 is now 190, the N entry for this track in the index must be changed to 190 as follows:

N		O		N		O		N		
1	120	1	120	2	190	2	200	3	250	...

Suppose further that record 200 is placed in an overflow area on track 10 and is the first record on this overflow track. If this is designated as 10:1, the overflow area should be changed as follows:

N		O		N		O		N		
1	120	1	120	2	190	10:1	200	3	250	...

In effect, then, record 200 has become the first of many possible records in the overflow area.

If record 186 is added to track 2, forcing 190 off the end into the overflow area leaving track 2 as

130	145	150	...	180	185	186

then record 190 will be added as the second record in the overflow area, namely 10:2, and the overflow entry on the track index will be replaced by 10:2 so that the track index becomes

N		O		N		O		N		
1	120	1	120	2	186	10:2	200	3	250	...

Note that in the O entry the 200 is not changed as it still represents the largest record key in the overflow area. In fact the previous entry 10:1 is added to the latest record to be added to the overflow area, record 190, so that it is not lost. The overflow area now looks like

#	200	10:1	190	...

with record 190 pointing to record 200. The symbol "#" indicates that record 200 does not point to another record. The overflow entry always contains two values :

One represents the largest key value in the overflow area that has been moved there from an individual track (200 in the above example) and the other contains a pointer to the smallest key in the overflow area (190 in the above example). If record 194 is now spilled to the overflow area, the O entry will not be changed as 190 is still the smallest record key value in

the overflow area. As the record with key 194 comes after 190, record 190 is adjusted to point to record 194 and 194 to 200. The sorted order is maintained in the overlow area, which now appears as

#	200	10:3	190	10:1	194	...

It is not necesaary that the records stored on a track in the overflow area be associated with only one track of the prime area. This is only the case here because we have assumed that all the overflow records on track 10 come from track 2. This is not always so. It is quite possible to have overflow records from many other tracks so that we could well imagine an overflow area as follows:

N		O		N		O		N		
#	200	10:4	190	#	216	10:1	194	10:3	214	...

where records 214 and 216 have arrived from track 4. Record 214 is the second overflow record from track 4 and points to the first from track 4, namely 216, at location 10:3.

The algorithm to be employed in adding a record to the overflow area can now be stated:

ALGORITHM : OVERFLOW ADDITIONS

1. Find the first available position in the overflow area.
2. Move the record to this position
3. If this record is the record of lowest key in the overflow area, place the pointer to this record in the overflow entry of the track index and move the old value in the track index to the pointer field of the newly added record. If this is not the case, move the address of the new record to the pointer field of the record in the overflow area that precedes it in sorted sequence and place the old value of the pointer into the pointer field of the new record.

Overflow Considerations

If a record is in the prime area of a file, its retrieval is straightforward. The master, cylidner, and track indexes are examined; the appropriate track selected; and finally, after rotational delay of the drive, the record is retrieved. This is not true if the record is in the overflow area.

If the record is in the overflow area, its retrieval can take a long time. Suppose, for example, that record 16,000 is the first record moved to an overflow area and later followed by 60 more such records. As these 60 later records are chained together in key sequence order by pointer, all 60 records will have to read before record 16,000 can be located. As each read is a time-consuming process, this can take a lon time. the efficiency of ISAM is defeated by allowing large numbers of records to overflow from a single track.

This problem can be overcome by writing a "clean-up" program which reads all the records on the file, including those in the overflow areas and creates a larger file. This can be done whenever the time taken to retrieve records has become unacceptable. **The time taken for retrieval is the dominant criterion here as it is acceptable to have a large overflow area if the records in it are seldom retrieved.**

The other possibility is, as with relative files to create dummy records. In ISAM a dummy record is a record that contains HIGH-VALUES in the first character position but must, as with all records in ISAM, also contain a unique key. During file creation these records are sorted along with all the other file records and scattered whereever desired within the file.

Dummy records prevent growth in the overflow area in two ways. First, if a record to be added to an ISAM file has the same key as a dummy record, it merely replaces the dummy record. This is the ideal situation as no records are shifted along a given track and none of the indexes are changed. The second feature of dummy records is that they are not moved to the overflow area if they are forced off the end of the track by the insertion of a new record. They are simply ignored. The N entry in the track index is changed to reflect the fact that a different record now holds the last position on the track. If both the O and N keys are the same before the addition takes place, then they are both changed. Suppose, for example, that the N and O entreis for track 7 of a certain cylinder are given as

N		O	
7	100	7	100

and that track 7 has the keys

50	60	70	...	90	100

with record 100 being a dummy record. The addition of record 55 would change track 7 as follows·

50	55	60	70	...	90

and since record 100 is a dummy record (and therefore not transferred to the overflow area), the N and O entries become

N		O	
7	90	7	90

Creating an ISAM File

An ISAM file must be loaded sequentially in sorted order by record key. ISAM will detect a record out of order. Any dummy records to be added to the file should be placed in the input data stream in sequence. These records are best added where record additions are expected to take place. For instance, a credit card company may expect in the near future to add records whose keys range between 416-250-000 and 416-275-000 as a new district of credit card holders is opened up. In this case, dummy records with these keys can be created and added to the file during file creation. Another possibility is simply to scatter a certain percentage of dummy records throughout the file. This is not nearly as effective nor alway possible (there may be no unused keys in the file). Recall that a dummy record is only ignored if it is at the end of a track. It will stay on a track until it is replaced by a valid record with the same key or pushed off the end by a new insertion.

Once the file is in use, any record whose deletion is desired can be turned into a dummy record by writing HIGH-VALUES in its first character position. This is a useful feature, especially in a credit card situation or phone number list where inactive customers can be replaced.

So what is the significance of the ISAM organisation ?

Advantages of ISAM indexes :

1) Because the whole sructure is ordered to a large extent, partial (LIKE ty%) and range (BETWEEN 12 and 18) based retrievals can often benefit from the use of this type of index.
2) ISAM is good for static tables becasue there are usually fewer index levels than B-tree.
3) Because the Index is never updated, there are never any locking contention problems within the index itself—this can occur in B-tree indexes, especially when they get to the point of 'splitting' to create another level.
4) In general there are fewer disk I/Os required to access data, provided there is no overflow.
5) Again if little overflow is evident, then data tends to be clustered. This means that a single block retrieval often brings back rows with similar key values and of relevance to the initiating query.

Disadvantages of ISAM indexes :

1) ISAM is still not as quick as some (hash organisation, dealt with later, is quicker).
2) Overflow can be a real problem in highly volatile tables.
3) The sparse index means that the lowest level of index has only the highest key for a specific data page, and therefore the block (or more usually a block header), must be searched to locate specific rows in a block.

In a nutshell therefore, these are the two types of indexing generally available. As I have already said, indexes can be either created on one single, or several groups, of columns within single tables, and generally the ability to create them should be a privilege under the control of the DBA. Indexes usually take up significant disk space, and although generally of significant benefit in the case of data retrieval, they can slow down insert/update and delete operation because of overhead in maintaining the index, and in ISAM of ensuring the logical organisation of the data rows. The presence of an index does not mean that the RDBMS will always use it, a reality of life discussed later; and it is also true that it is not generally possible to pick and choose which index will be used under which conditions. If follows, therefore, that the administration of indexes should be done centrally, with great case, and should be a major consideration in the physical design stage of a project due to its application dependence.

Before leaving these types of access mechanisms and moving on to an explanation of hashing, it is worth listing some of the other functions that may be required within the context of manipulating indexes.

3.4.3 VSAM

The major disadvantage of the index-sequential organisation is that as the file grows, performance deteriorates rapidly because of overflows and consequently there arises the need for periodic reorganisation. Reorganisation is an expensive process and the file becomes unavailable during reorganisation. The **virtual storage access method (VSAM)** is IBM's advanced version of the index-sequential organisation that avoids these disadvantages. The system is immune to the characteristics of the storage medium, which could be considered as a pool of blocks. The VSAM files are made up of two components: the index and data. However, unlike index-sequential organisation, overflows are handled in a different manner. The VSAM index and data are assigned to distinct blocks of virtual storage called a **control interval**. To allow for growth, each time a data block overflows it is divided into two blocks and appropriate changes are made to the indexes to reflect this division.

3.4.4 Implementation of Indexing through Tree-Structure

Indexes support applications that selectively access individual records, rather than searching through the entire collection in sequence. One field (or a group of fields) is used as the index field. For example in a banking application, there might be a file of records describing Branch Offices. It might be appropriate to index the file on Branch Name, providing access to Branch Office information to support interactive inquiries.

We shall start with a relatively simple tree-structured index and then progress to more complex structures.

Binary Search Trees As Indexes

Let us first reconsider the binary search tree. Recall that the nodes of a binary search tree are arranged in such a way that a search for a particular key value proceeds down one branch of the tree. The sought key value is compared with the key value of the tree's root: if it is less than the root value, the search proceeds down the left subtree; if it is greater than the root value, the search proceeds down the right subtree. The same logic is applied at each node encountered until the search is satisfied or it is determined that the sought key is not included in the tree.

Typically, a key value does not stand alone. Rather, the key value is associated with information fields to form a record. In general, storing these information fields in the binary search tree would make for a very large tree. In order to speed searches and to reduce the tree size, the information fields of records commonly are stored separate from the key values. These records are organised into files and are sotred on secondary storage devices such as rotating disks. The connection between a key value in the binary search tee and the corresponding record in the file is made by housing a pointer to the record with the key value.

Figure 17 shows the binary search tree with pointers included to the data records.

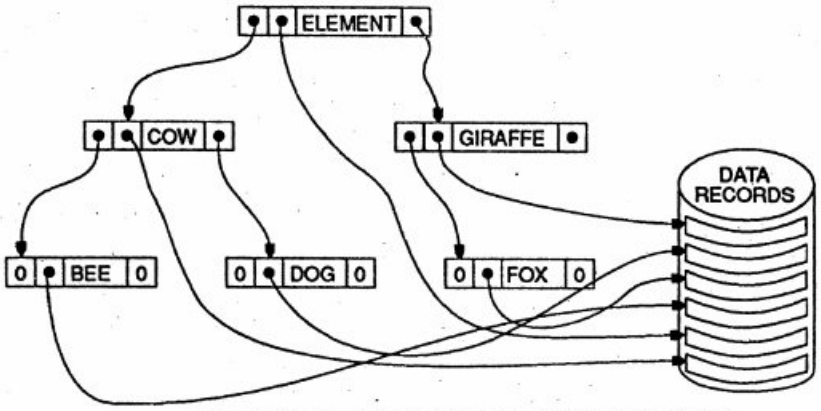


Figure 17 : Binary search tree from figure 8-23 used as an index

This augmentation of the binary search tree to include pointers (i.e. addresses) to data records outside the structure qualifies the tree to be an index. An index is a structured collection of key value and address pairs; the primary purpose of an index is to facilitate access to a collection of records. An index is said to be a dense index if it contains a key value-address pair for each record in the collection. An index that is not dense is sometimes called a sparse index. There are many ways to organise an index; the augmented binary search tree is one approach.

At this time we will not concern ourselves with the actual location of those records on primary or secondary storage. Suffice it to say that in contrast to relative files where the physical locations of records are determined by a hash algorithm applied to key values, there is significantly more freedom in the physical placement of records in an indexed file.

M-Way Search Trees

The performance of an index can be enhanced significantly by increasing the branching factor of the tree. Rather than binary branching, m-way (m2) branching can be used. For expository purposes, we ignore for the moment pointers out of the structure to data records and consider only the internal structure of the tree. An m-way search tree is a tree in which each node has out- degree <= m. When an m-way search tree is not empty, it has the following properties.

1. Each node of the tree has the structure shown in figure 18.

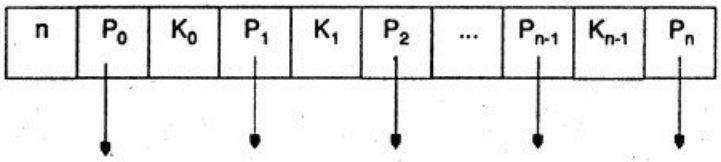


Figure 18 : Fundamental structure of a node in an m-way search tree

The $P_0, P_1, \ldots\ldots, P_n$ are pointers to the node's subtrees and the $K_0, \ldots\ldots, K_{n-1}$ are key values. The requirement that each node have out-degree <= m forces n = m-1.

2. The key values in a node are in ascending order:

   $K_i < K_{i+1}$

   for i = 0, ... n-2.

3. All key values in nodes of the subtree pointed to by P_i are less than the key value K_i for i = 0, ..., n-1.

4. All key values in nodes of the subtree pointed to by P_n are greater than the key value Kn-1.

5. The subtrees pointed to by the P_i, i = 0, ..., n are also m-way search trees.

   Note that the arrangement of key values in nodes is analogous to their arrangement in binary search trees. A five-way search tree has a maximum of five pointers out of each node, i.e., n 4. Some nodes of the tree may contain fewer than five pointers.

Example

Figure 19 illustrates a three-way search tree. Only the key values and the non-null subtree pointers are shown. In the figure, leaf nodes have been depicted as containing just key values. Each internal node of the search tree has the structure depicted in figure 18, here with a maximum of three subtree pointers.

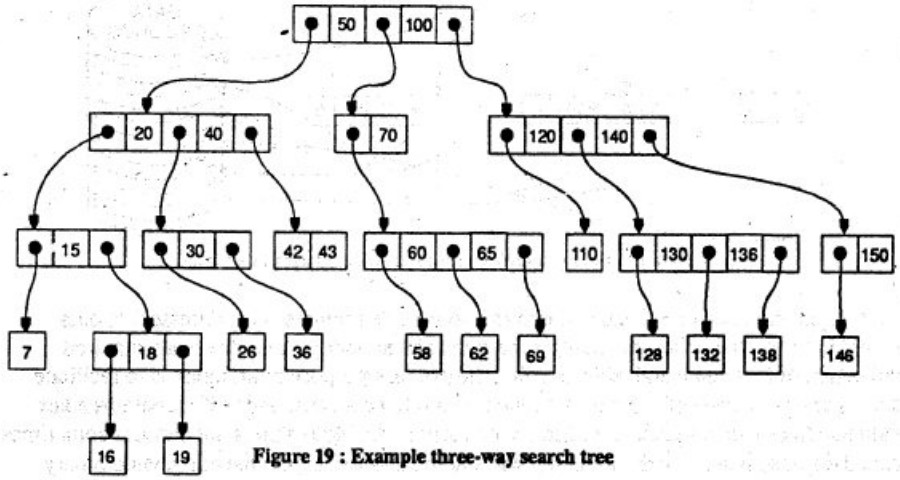


Figure 19 : Example three-way search tree

M-Way Search Trees as Indexes

When an m-way search tree is used as an index, each key- pointer pair (K_i, P_i) becomes a triplet P_i, K_i, A_i where A_i is the address of the data record associated with key value K_i. Thus, each tree node not only points to its child nodes in the tree, but also points into the collection of data records. If the index is dense, every record in the collection will be pointed to by some node in the index.

We can define the node type for an m-way search tree index as follows in Pascal.

```
type nodeptr = ↑ nodetype;
     recptr = ↑ rectype;
     n1, keytype = integer;
     nodetype = record
                  n:integer
                  keyptrs:array [0..n1] of record
                                             ptr : nodeptr;
                                             key : keytype;
                                             addr : recptr
                                           end;
                  keyptrn:nodeptr
                end;
```

Again key's type need not be integer, $n \leq m - 1$ and $n1 = n - 1$.

Searching an M-Way Search Tree

The process of searching for a key value in an m-way search tree is a relatively straightforward extension of the process of searching for a key value in a binary search tree. A recursive version of the search algorithm follows. The variable keys contains the sought key value; r initially points to the root of the tree. The search tree is assumed to be global, with var node: nodetype

```
procedure search(skey;keytrype;var r:nodeptr, foundrec:recptr);
var i;0..n;
begin if (r=nil)
        then foundrec: = nil
        else begin i: = 0;
                while (i < n and skey  node.key[i])
                do i : = i + 1;
                if (i < n and skey = node.key[i])
                then foundrec : = node.addr[i]
                else if i < n
                     then search (skey,node.ptr[i],foundrec)
                     else search(skey,node.keyptrn,foundrec)
             end;
      end;
```

Compare this logic with direct searches of a binary search tree. The primary difference is that the array of keys in each node of the m-way tree must be scanned to find the appropriate pointer to follow either down to a child node or directly to the data record.

B-TREES

The basic B-tree structure was discovered by R.Bayer and E. McCreight (1970) of Boeing Scientific Research Labs and has grown to become one of the most popular techniques for organising an index structure. Many variations on the basic B-tree have been developed; we cover the basic structure first and then introduce some of the variations.

The B-tree is known as the balanced sort tree, which is useful for external sorting. There are strong uses of B-trees in a database system as pointed out by D.Comer (1979) : "While no single scheme can be optimum for all applications, the technique of organising a file and its index called the B-tree is, de facto, the standard organisation for indexes in a database system."

The file is a collection of records. The index refers to a unique key, associated with each record. One application of B- trees is found in IBM's Virtual Storage Access Method (VSAM) file organisation. Many data manipulation tasks require data storage only in main memory. For applications with a large database running on a system with limited company, the data must be stored as records on secondary memory (disks) and be accessed in pieces. The size of a record can be quite large, as shown below:

```
struct DATA
    {
        int ssn;
        char name [80];
        char address [80];
        char schoold [76];
        struct DATA * left;          /* main memory addresses */
        struct DATA * right;         /* main memory addresses */
        d_block d_left;              /* disk block address */
        d_block d_right;             /* disk block address */
    }
```

There are two sets of pointers in the struct DATA. The main memory pointers, left and right, are used when the children of the node are in memory and the disk addresses, d_left and d_right, are used to reference the children on the disk. The size of DATA is 256 bytes.

Data is moved by block transfer into main memory for manipulation; however, the disk access is slow (tens of milliseconds) compared with a main memory move (tens of microseconds), so we want to minimise the disk accesses. One method is to make the data nodes even fractions (1/2, 1/4, etc. or multiples (1,2, etc.) of the disk sector size.

Another method is to expand the capacity of the node for storage of data. A binary tree node contains one key or data element and has pointers to two children. We can construct a tree with nodes that have more than two possible children and more than one key. The tree has a property called order, the maximum number of children for any given node. If the maximum is N children, then the order of the tree is N.

The ADT B-Tree

To reduce disk accesses, several conditions of the tree must be true; the height of the tree must be kept to a minimum; there must be no empty subtrees above the leaves of the tree; the leaves of the tree must all be on the same level; and all nodes except the leaves must have at least some minimum number of children (perhaps half of the maximum). The toot alone may have no children, if the tree has only one node. Otherwise it may have as few as two and as many as the maximum number of children. The keys in the tree should have some defined ordering (numerical, lexical, or some other relationship). A tree that has these properties is called a balanced sort tree (a B-tree). The B-tree properties are listed below.

An ADT B-Tree of order N is a tree in which:

1. Each node has a maximum of N children and a minimum of N/2 children. The root may have no children or any number from 2 to the maximum.
2. Each node has one fewer keys than children with a maximum of N-1 keys.
3. the keys are arranged in a defined order within the node. All keys in the subtree to the left of a key are predecessors of the key, and all keys in the subtree to the right of a key are successors of the key.

4. When a new key is to be inserted into a full node, the node is split into two nodes, and the key with the median value is inserted in the parent node. In case the parent node is the root, a new root node is created.
5. All leaves are on the same level. There is no empty subtree above the level of the leaves.

Insertion in the B-Tree

The insertion of a new key into a B-tree begins with the search for a match. If a match is found for the key, then the insertion operation takes some action (an error message is sent or a count is incremented, etc.) and returns an error indication to the caller. If no match is found, then the key is simply added to a leaf, unless the left is ful. If the leaf is full then it is split into two nodes (requiring creation of a new node and the copying of half of the old nodes keys and pointers to the new node) and the median value key is inserted into the parent of the node. If the parent was also full, the split and key push up ripples upward. The ordering of the keys is maintained, and the ordering of keys among subtrees is maintained.

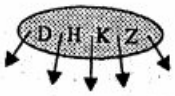
The tree trends to become more balanced with subsequent insertions. The B-tree is like a crystal in its growth: upward and outward from its base, the leaf level. To illustrate the insertion process, we will build a B-tree of order 5 by inserting the following data:

{D, H, K, Z, B, P, Q, E, A, S, W, T, C, L, N, Y, M}

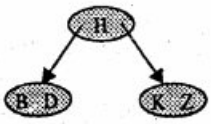
The key ordering is ascending lexical. We keep track of the pointers by numbering them.

D. First the root node is created (+), then key D is inserted into it.

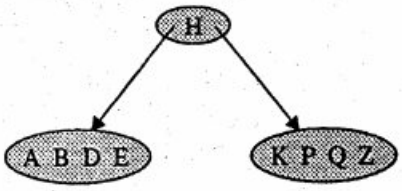
H, K, Z: the keys are searched for match; if match not found, they are simply inserted.



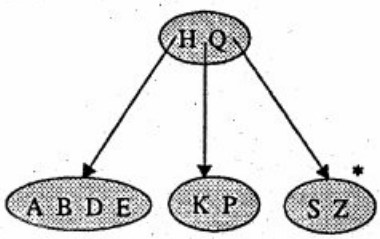
B: The node is full, so it must be split; of the keys, B, D, H, K, and Z, key H is the median value key, so it promotes to the parent, but this is the root node so we must create a new node (+).



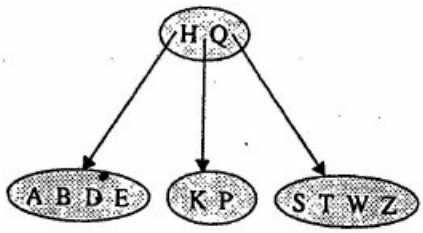
P, Q, E, A: Each key is simply inserted.



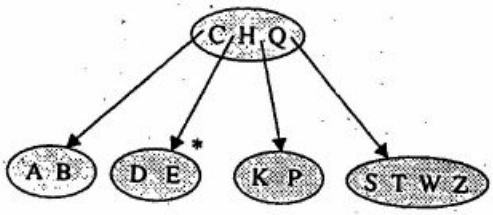
S: The node is full, so it must be split. A new node at the leaf level is created (*). Of the keys, K, P, Q, S, and Z, key Q is the median value.



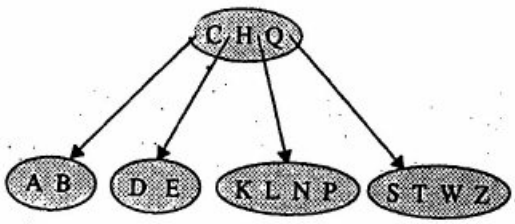
W, T: The keys are simply inserted.



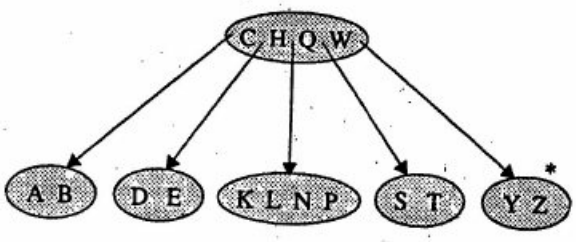
C: The node is full, so it must be split. A new node at the leaf level is created (*). Of the keys, A, B, C, D, and E, key C is the median value.



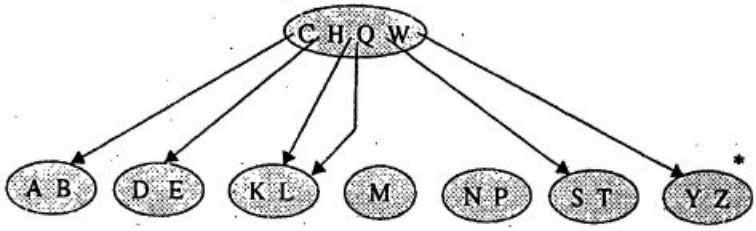
L, N: The keys are simply inserted.



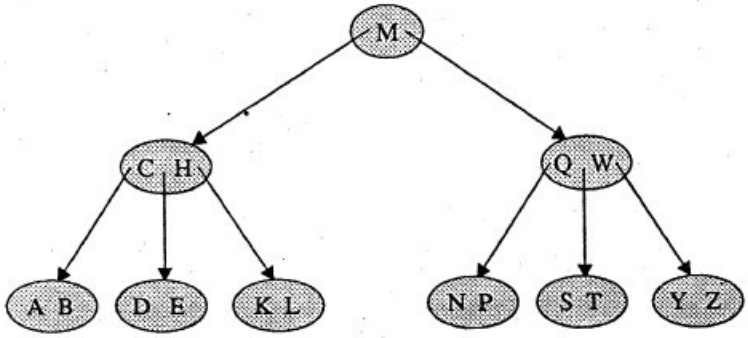
Y: The node is full, so it must be split. A new node at the leaf level is created (*). Of the keys, S, T, W, Y, and Z, key W is the median value.



M: The node is full, so it must be split. A new node at the leaf level is created (*). Of the keys, K, L, M, N, and P, key M is the median value.



The root node is also full, so it is split. A new root node is created (+). Of the keys, C, H, M, Q, and W, the key M is the median key. It is inserted into the new root.



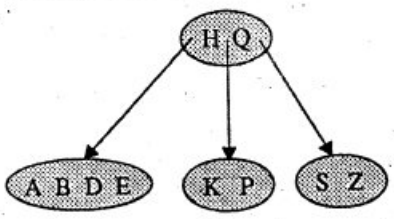
Note how the balance has been maintained throughout the insertions. A node split prepares the way for a number of simple insertions. If the number of keys in a node is large, the time before the next split (involving node creation and data transfer into the new node) will be fairly long.

Deletion of a Node in the B-Tree

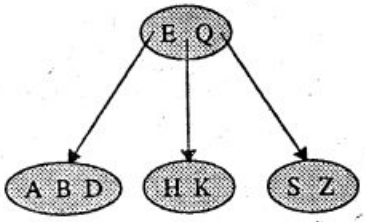
Deletion of a key is somewhhat the reverse of insertion. If the key is found in a leaf, the deletion is fairly straightforward. If the key is not in a leaf, then the key's successor or predecessor must be in a leaf, because the insertion is done starting at a leaf. If the leaf is full, the median value key is pushed upward, so the key closest in value to a key in a nonleaf node must be in the root. The requirement that there be a minimum number of keys in a node also plays a part, as we shall see.

If the key is found in a leaf node and the node has more than the minimum number of nodes, then the key is simply deleted and the other keys in the node adjusted in position. If the node has only the minimum number of keys, then we must look at the immediately adjacent leaf nodes. If one of nodes has more than the minimum number of keys, the mdeian key in the parent node is moved down to replace the deleted key and one of the keys from the adjacent leaf node is moved into the parent node in place of the median key.

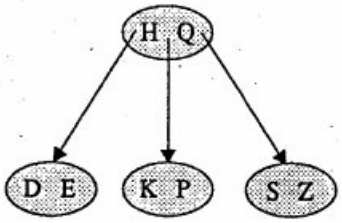
To demonstrate this deletion, we use the B-tree of order 5 shown in the section on insertion. We will delete the key P from the tree after the insertion of the key S.



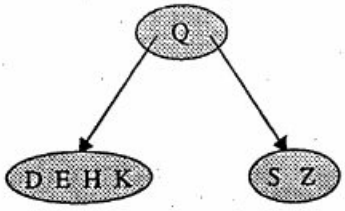
P:Its leaf node is at minimum key count, and the adjacent node with keys A, B, D, and E has more than the minimum key count. We can replace P with the median key H in the parent node. We then replace H with the key closest to it in value, key E.



If both of the adjacent nodes have only the minimum number of keys, one of the adjacent nodes can be combined with the node that held the deleted key, and the median key from the parent node that partitioned the two leaf nodes is pushed down into the new leaf node. We will use the tree from the last example with keys A and B deleted.



P:Its leaf node is at minimum key count, and the adjacent nodes also have minimum key count. We push H down into the combined node.



We summarise the key features of a B-tree as follows:

1. There is no redundant storage of search key values. That is, B-tree stores each search key value in only one node, which may contain other search key values.
2. The B-tree is inherently balanced, and is ordered by only one type of search key.
3. The insertion and deletion operations are complex with the time complexity $O(\log_2 n)$.
4. The search time is $O(\log_2 n)$.
5. The number of keys in the nodes is not always the same. The storage management is only complicated if you choose to create more space for pointers and keys, otherwise the size of a node is fixed.
6. The B-tree grows at the node as opposed to the binary tree, BST, and AVL trees.
7. For a B-tree of order N with n nodes, the height is log n. The height of a B-tree increases only because of a split at the root node.

There are several variations of B-tree, including B^+ -tree and B^* -tree. The B^+ -tree indices are similar to B-tree indices. The main difference between the B^+ -tree and B-tree are:

1. In a B^+ - tree, the search keys are stored twice; each of the search keys is found in some leaf nodes.
2. In a B-tree, there is no redundancy in storing search-key values; the search key is found only one in the tree. Since search-key values that are found in nonleaf nodes are not found anywhere else in the B-tree, an additional pointer field for each search key is kept in a nonleaf node.
3. In a B^+ -tree, the insertion and deletion operations are complex and $O(\log_2 n)$, but the search operation is simple, efficient, and $O(\log_2 n)$.

Because this B-tree structure is so common place, it is worth simply listing some of the more importatn advantages and disadvantages.

Advantages of Btree indexes :

1) Because there is no overflow problem inherent with this type of organisation it is good for dynamic table – those that suffer a great deal of insert / update / delete activity.
2) Because it is to a large extent self-maintaining, it is good in supporting 24-hour operation.
3) As data is retrieved via the index it is always presented in order.
4) 'Get next' queries are efficient because of the inherent ordering of rows within the index blocks.
5) Btree indexes are good for very large tables because they will need minimal reorganisation.

6) There is predictable access time for any retrieval (of the same number of rows of course) because the Btree structure keeps itself balanced, so that there is always the same number of index levels for every retrieval. Bear in mind of course, that the number of index levels does increase both with the number of records and the length of the key value.

Because the rows are in order, this type of index can service range type enquiries, of the type below, efficiently.

SELECT ... WHERE COL BETWEEN X AND Y.

Disadvantages of Btree indexes :

1) For static tables, there are better organisations that require fewer I/Os. ISAM indexes are preferable to Btree in this type of environment.

2) Btree is not really appropriate for very small tables because index look-up becomes a significant part of the overall access time.

3) The index can use considerable disk space, especially in products which allow different users to create separate indexes on the same table/column combinations.

4) Because the indexes themselves are subject to modification when rows are updated, deleted or inserted, they are also subject to locking which can inhibit concurrency.

So to conclude this section on Btree indexes, it is worth stressing that this structure is by far and away the most popular, and perhaps versatile, of index structures supported in the world of the RDBMS today. Whilst not fully optimised for certain activity, it is seen as the best single compromise in satisfying all the different access methods likely to be required in normal day-to-day operation.

3.5 DIRECT FILE ORGANISATION

In the index-sequential file organisation considered in the previous sections, the mapping from the search-key value to the storage location is via index entries. In direct file

Figure 20 : Mapping from a key value to an address value

organisation, the key value is mapped directly to the storage location. The usual method of direct mapping is by performing some arithmetic manipulation of the key value. This process is called **hashing**. Let us consider a hash function h that maps the key value k to the value h(k). The value h(k) is used as an address and for our application we require that this value be in some range. If our address area for the records lies between S_1 and S_2, the requirement for the hash function h(k) is that for all values of k it should generate values between S_1 and S_2.

It is obvious that a hash function that maps many different key values to a single address or one that does not map the key values uniformly is a bad hash function. A **collision** is said to occur when two distinct key values are mapped to the same storage location. Collision is handled in a number of ways. The colliding records may be assigned to the next available space, or they may be assigned to an overflow area. We can immediately see that with hashing schemes there are no indexes to traverse. With well-designed hashing functions where collisions are few, this is a great advantage.

Another problem that we have to resolve is to decide what address is represented by h(k). Let the addresses generated by the hash function the addresses of buckets in which the y, address pair values of records are stored. Figure shows the buckets containing the y, address pairs that allow a reorganisation of the actual data file and actual record address without affecting the hash functions. A limited number of collisions could be handled automatically by the use of a bucket of sufficient capacity. Obviously the space required for the buckets will be, in general, much smaller than the actual data file. Consequently, its reorganisation will not be that expensive. Once the bucket address is generated from the key by the hash function, a

search in the bucket is also required to locate the address of the required record. However, since the bucket size is small, this overhead is small.

The use of the bucket reduces the problem associated with collisions. In spite of this, a bucket may become full and the resulting overflow could be handled by providing overflow buckets and using a pointer from the normal bucket to an entry in the overflow bucket. All such overflow entries are linked. Multiple overflow from the same bucket results in a long list and slows down the retrieval of these records. In an alternate scheme, the address generated by the hash function is a bucket address and the bucket is used to store the records directly instead of using a pointer to the block containing the record.

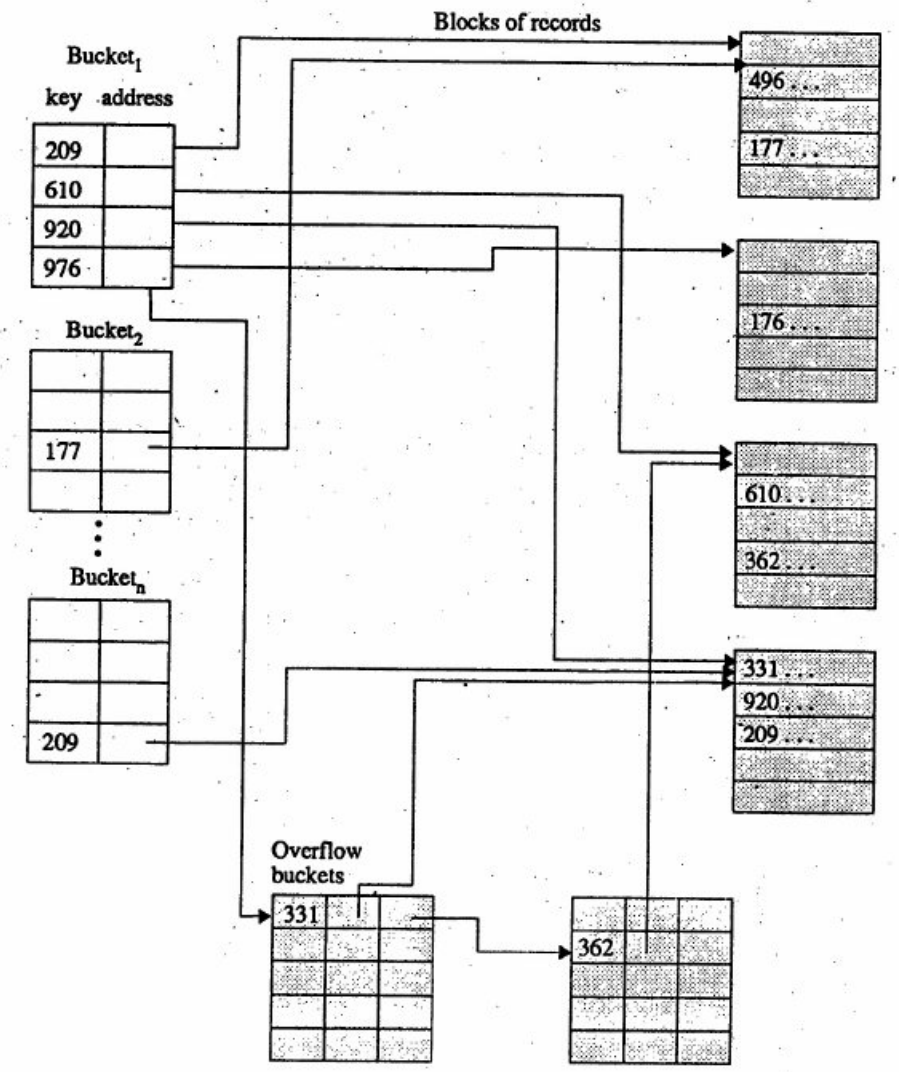


Figure 21: Bucket and block organisation for hashing

Let s represent the value:

s = upper bucket address value - lower bucket address value + 1

Here, s gives the number of buckets. Assume that we have some mechanism to convert key values to numeric ones. Then a simple hashing function is:

h(k) = k mod s

where k is the numeric representation of the key and h(k) produces a bucket address. A moment's thought tells us that this method would perform well in some cases and not in others.

It has been shown, however, that the choice of a prime number for s is usually satisfactory. A combination of multiplicative and divisive methods can be used to advantage in many practical situations.

There are innumerable ways of converting a key to a numeric value. Most keys are numeric, others may be either alphabetic or alphanumeric. In the latter two cases, we can use the bit representation of the alphabet to generate the numeric equivalent key. A number of simple hashing methods are given below. Many hashing functions can be devised from these and other ways.

1. Use the low order part of the key. For keys that are consecutive integers with few gaps, this method can be used to map the keys to the available range.

2. End folding. For long keys, we identify start, middle, and end regions, such that the sum of the lengths of the start and end regions equals the length of the middle region. The start and end digits are concatenated and the concatenated string of difits is added to the middle region digits. This new number, mod s where s is the upper limit of the hash function, gives the bucket address:

   123456 123456789012 654321

   For the above key (converted to integer value if required) the end folding gives the two values to be added as: 123456654321 and 123456789012.

3. Square all or part of the key and take a part from the result. The whole or some defined part of the key is squared and a number of digits are selected from the square as being part of the hash result. A variation is the multiplicative scheme where one part of the key is multiplied by the remaining part and a numebr of digits are selected from the result.

4. Division. As stated in the beginning of this section, the key can be divided by a number, usually a prime, and the remainder is taken as the bucket address. A simple check with, for instance, a divisor of 100 tells us that the last two digits of any key will remain unchanged. In applications where keys may be in some multiples, this would produce a poor result. Therefore, division by a prime number is recommended. For many applications, division by odd numbers that have no divisors less than about 19 gives satisfactory results.

We can conclude from the above discussion that a number of possible methods for generating a hash function exist. In general it has been found that hash functions using division or multiplication performs quite well under most conditions.

To summarise the advantages and disadvantages of this approach :

Advantages of hashing :

1) Exact key matches are extremely quick.

2) Hashing is very good for long keys, or those with multiple columns, provided the complete key value is provided for the query.

3) This organisation usually allows for the allocation of disk space so a good deal of disk management is possible.

4) No disk space is used by this indexing method.

Disadvantages of hashing.:

1) It becomes difficult to predict overflow because the workings of the hashing algorithm will not be visible to the DBA.

2) No sorting of data occurs either physically or logically so sequential access is poor.

3) This organisation usually takes a lot of disk space to ensure that no overflow occurs — there is a plus side to this though: no space is wasted on index structures because they simply don't exist.

To sum up hashing it's true to say that not many products support this type of strucutre, and it is likely, I feel, to become entirely redundant in most software RDBMSs. In a hashing organisation, the key that is hashed should be the one that is used most to retrieve the data (or join it to other tables) and this will often not be the primary key that I have previously defined within the scope of logical data design.

3.6 MULTIKEY FILE ORGANISATION

In this section, we have introduced a family of file organisation schemes that allow records to be accessed by more than one key field. Until this point, we have considered only single-key file organisation. Sequential by a given key, direct access by a particular key and indexed sequential giving both direct and sequential access by a single key. Now we enlarge our base to include those file organisation that enable a single data file to support multiple access paths, each by a different key. These file organisation techniques are at the heart of database implementation.

There are numerous techniques that have been used to implement multikey file organisation. Most of the approaches are based on building indexes to provide direct access by key value. The fundamental indexing techniques were already introduced in the section 3.4. In this section we discuss two approaches for providing additional access paths into a file of data records.

- Multilist file organisation
- Inverted file organisation

3.6.1 Need for the Multiple Access Path

Many interactive information systems require the support of multi-key files. Consider a banking system in which there are several types of users: teller, loan officers, branch manager, bank officers, account holders, and so forth. All have the need to access the same data, say records of the format shown in figure 22. Various types of users need to access these records in different ways.A teller might identify an account record by its ID value. A loan officer might need to access all account records with a given value for OVERDRAW-LIMIT, or all account records for a given value of SOCNO. A branch manager might access records by the BRANCH and TYPE group code. A bank officer might want periodic reports of all accounts data, sorted by ID. An account holder (customer) might be able to access his or her own record by giving the appropriate ID value or a combination of NAME, SOCNO, and TYPE code.

ACCOUNT

ID	NAME		GROUP-CODE		BALANCE	OVERDRAW LIMIT
	LAST	FIRST	BRANCH	TYPE		

Figure 22 : Example record format

Support by Replicating Data

One approach to being able to support all these types of access is to have several different files, each organised to serve one type of request. For this banking example, there might be one indexed sequential account file with key ID (to server tellers, bank officers, and account holders), one sequential account file with records ordered by OVER-DRAW-LIMIT (to serve loan officer), one account file with relative organisation and user-key SOCNO (to serve loan officers), one sequential account file with records ordered by GROUP-CODE (to serve branch managers), and one relative account file with user-key NAME, SOCNO, and TYPE code (to serve account holders). We have just identified five files, all containing the same data records! The five files differ only in their organisations, and thus in the access paths they provide.

Difficulties Caused by Replication

Replicating data across files is not a desirable solution to the problem of providing multiple access paths through that data. One obvious difficulty with this approach is the resultant storage space requirements. However, a more serious difficulty with this approach is keeping updates to the replicated data records coordinated. The multi-key file is a classical and often successful solution to the multiple-path retrieval problem; it uses indexes rather than data replication.

Whenever multiple copies of data exist, there is the potential for discrepancies. Assume that you have three calendars. You keep one at home by the telephone, one in your briefcase, and one at your office. What is the probability that those three calendars show the same record of appointments and obligations? The likely situation is that you will post some updates to one copy but forget to enter them in other copies. Even if you are quite conscientious about updating all three copies, there must be some lag between the times that the updates actually appear in the three locations. The problem becomes more complex if somebody in addition to you, for example your secretary, also updates one or more of your calendars. The same difficulties arise in updating data that appear in multiple files.

The result of incomplete and asynchronous updates is loss of data integrity. If a loan officer queries one file and finds that account #123456 has an overdraw limit of $250, then queries another file and finds that the same account has an overdraw limit of $1000, he or she should question the validity of the data.

Support by Adding Indexes

Another approach to being able to support several different kinds of access to a collection of data records is to have one data file with multiple access paths. Now there is only one copy of any data record to be updated, and the update synchronization problem caused by record duplication is avoided. This approach is called multi-key file organisation.

The concept of multiple-key access generally is implemented by building multiple indexes to provide different access paths to the data records. There may also be multiple linked lists through the data records. We have seen already that an index can be structured in several ways, for example as a table, a binary search tree, a B-tree, or a B^+ - tree. The most appropriate method of implementing a particular multi-key file is dependent upon the actual uses to be made of the data and the kinds of multi-key file support available.

3.6.2 Multilist file organisation

Before defining multilist file organisation, let us understand the difference between linked organisation and sequential file organisation. Linked organisations differ from sequential organisations essentially in that the logical sequence of records is generally from the physical sequence. In a sequential organisation, if the i'th record of the file is at location l_i then the i + 1'st record is in the next physical position $l_i + c$ where c may be the length of the i'th record or some constant that determines the inter-record spacing. In a linked organisation the next logical record is obtained by following a link value from the present record. Linking records together in order of increasing primary key value facilitates easy insertion and deletion once the place at which the insertion or deletion to be made is known. Searching for a record with a given primary key value is difficult when no index is available, since the only search possible is a sequential search. To facilitate searching on the primary key as well as on secondary keys it is customary to maintain several indexes, one for each key. An employee number index, for instance, may contain entries corresponding to ranges of employee numbers. One possibility for the example of figure 23 would be to have an entry for each of the ranges 501-700, 701-900 and 901-1100. All records having E# in the same range will be linked together as in figure 24. Using an index in this way reduces the length of the lists and thus the search time. This idea is very easily generalised to allow for easy secondary key retrieval. We just set up indexes for each key and allow records to be in more than one list. This leads to the **multilist structure** for file representation. Figure 25 shows the indexes and lists corresponding to multilist representation of the data of figure 24. It is assumed that the only fields designated as keys are: E#, Occupation, Sex and Salary. Each record in the file, in addition to all the relevant information fields, has 1 link field for each key field.

Record	E#	Name	Occupation	Degree	Sex	Locaion	MS	Salary
A	800	HAWKINS	programmer	B.S.	M	Los Angeles	S	10,000
B	510	WILLIAMS	analyst	B..S	F	Los Angeles	M	15,000
C	950	FRAWLEY	analyst	M.S.	F	Minneapolis	S	12,000
D	750	AUSTIN	programmer	B.S.	F	Los Angeles	S	12,000
E	620	MESSER	programmer	B.S.	M	Minneapolis	M	9,000

Figure 23 : Sample data for Employee File

The logical order of records in any particular list may or may not be important depending upon the application. In the example file, lists corresponding to E#, Occupation and Sex have been set up in order of increasing E#. The salary lists have been set up in order of increasing salary within each range (record A precedes D and C even though E#(C) and E#(D) are less than E#(A)).

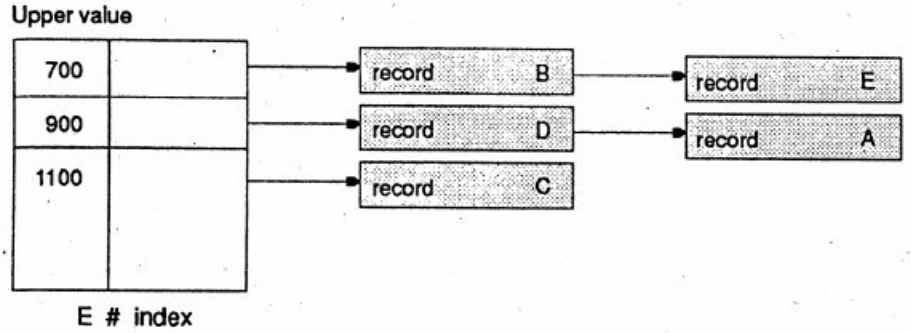


Figure 24 : Linking together all seconds in the same type.

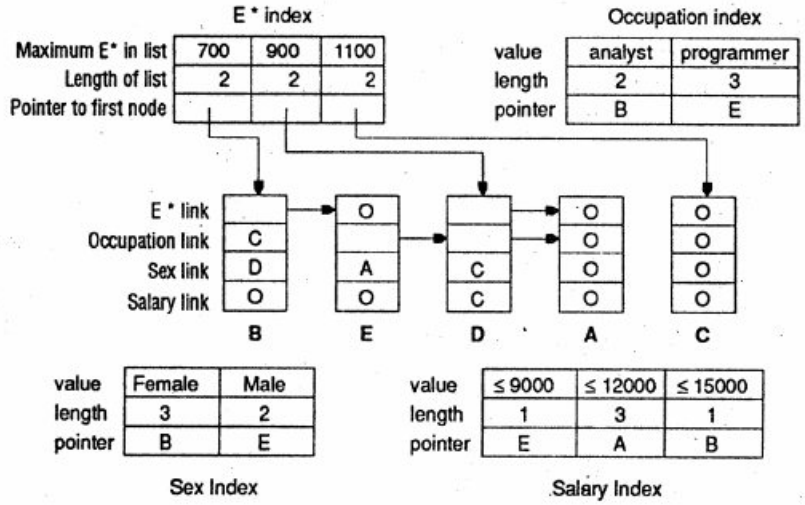


Figure 25 : Multilist representation for figure 23

Notice that in addition to key values and pointers to lists, each index entry also contains the length of the corresponding list. This information is useful when retrieval on boolean queries is required. In order to meet a query of the type, retrieve all records with Sex = female and Occupation = analyst, we search the Sex and Occupation indexes for female and analyst respectively. This gives us the pointers B and B. The length of the list of analysts is less than that of the list of females, so the analyst list starting at B is searched. The records in this list are retrieved and the Sex key examined to determine if the record truly satisfies the query. Retaining list lengths enables us to reduce search time by allowing us to search the smaller list. Multilist structures provide a seemingly satisfactory solution for simple and range queries. When boolean queries are involved, the search time may bear no relation to the number of records satisfying the query. The query K1 = XX and K2 = XY may lead to a XX list of length n and a K2 list of length m. Then, min{n,m} records will be retrieved and tested against the query. It is quite possible that none or only a very small number of these min{n,m} records have both K1 = XX and K2 = XY. This situation can be remedied to some extent by the use of compound keys. A compound key is obtained by combining two or more keys together. We would combine the Sex and Occupation keys to get a new key Sex-Occupation. The values for this key would be: female analyst, female programmer, male analyst and male programmer. With this compound key replacing the two keys Sex and Occupation, we can satisfy queries of the type, all male programmers or all programmers, by retrieving only as many records as actually satisfy the query. The index size, however, grows rapidly with key compounding. If we have ten keys $K_1, \ldots K_{10}$, the index for K_i having n_i entries, then the index for the compound key $K_1 K_2 \ldots K_{10}$ will have $\pi_{i=1}^{10} n_i$ entries while the original indexes, had a total of $\Sigma_{i=1}^{10} n_i$ entries. Also, handling simple queries becomes more complex if the individual key indexes are no longer retained.

Inserting a new record into a multilist structure is easy so long as the individual lists do not have to be maintained in some order. In this case the record may be inserted at the front of the appropriate lists. Deletion of a record is difficult since there are no back pointers. Deletion may be simplified at the expense of doubling the number of link fields and maintaining each list as a doubly linked list. When space is at a premium, this expense may not be acceptable. An alternative is the coral ring structure described below.

Coral Rings

The coral ring structure is an adaptation of the doubly linked multilist structure discussed above. Each list is structured as a circular list with a headnode. The headnode for the list for key value $K_i = X$ will have an information field with value X. The field for key K_i is replaced by a link field. Thus, associated with each record, Y, and key, K, in a coral ring there are two link fields: ALINK(Y,i) and BLINK (Y,i). The ALINK field is used to link together all records with the same value for key K_i. The ALINKS form a circular list with a headnode whose information field retains the value of K_i for the records in this ring. The BLINK field for some records is a back pointer and for others it is a pointer to the head node. To distinguish between these two cases another field FLAG(Y,i) is used. FLAG(Y,i) = 1 if BLINK(Y,i) is a back pointer and FLAG(Y,i) = 0 otherwise. In practice the FLAG and BLINK fields may be combined with BLINK(Y,i) 0 when it is a back pointer and 0 when it is a pointer to the head node. When the BLINK field of a record BLINK(Y,i) is used as a back pointer, it points to the nearest record, Z, preceding it in its circular list for K, having BLINK(Z,i) also a back pointer. In any given circular list, all records with back pointers form another circular list in the reverse direction (See figure 27). The presence of these back pointers makes it possible to carry out a deletion without having to start at the front of each list containing the record being deleted in order to determine the preceding records in these lists. Since these BLINK fields will usually be smaller than the original key fields they replace, an overall saving in space will ensue. This is, however, obtained at the expense of increased retrieval time. Indexes are maintained as for multilists. Index entries now point to head nodes. As in the case of multilists, an individual node may be a member of several rings on different keys.

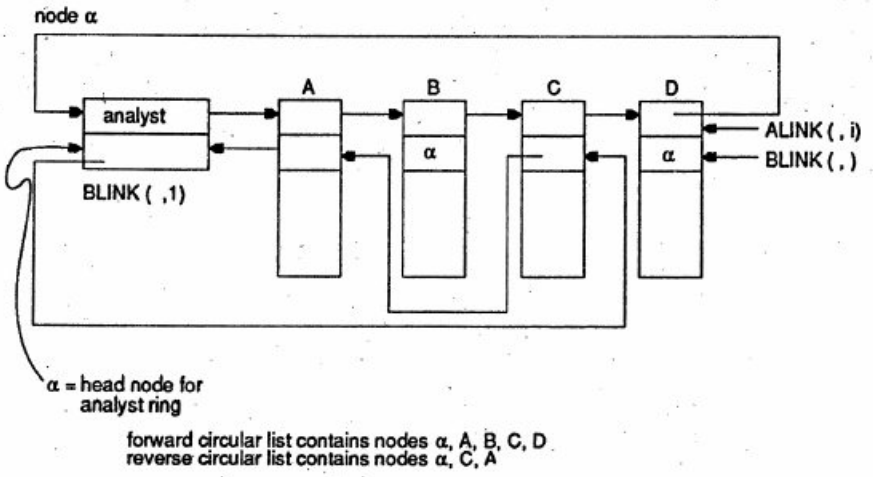


Figure 26 : Coral rings for analysts in a hypothetical file

3.6.3 Inverted File Organisation

Conceptually, inverted files are similar to multilists. The difference is that while in multilists records with the same key value are linked together with link information being kept in individual records, in the case of inverted files this link information is kept in the index itself. Figure 27 shows the indexes for the file of figure 24. A slightly different strategy has been used in the E# and salary indexes than was used in figure 26, though the same strategy could have been used here too. To simplify further discussion, we shall assume that the index for every key is dense and contains a value entry for each distinct value in the file. Since the index entries are variable length (the number of records with the same key value is variable), index maintenance becomes more complex than for multilists. However, several benefits accrue from this scheme. Boolean queries require only one access per record satisfying the query (plus some accesses to process the indexes). Queries of the type K1 = XX and K2 = XY. These two lists are then merged to obtain a list of all records satisfying the query. K1 =

XX and K2 = XY can be handled similarly by intersecting the two lists. K1 = .not. XX can be handled by maintaining a universal list, U, with the addresses of all records. Then, K1 = .not. XX is just the difference between U and the list for K1 = XX. Any complex boolean query may be handled in this way. The retrieval works in two steps. In the first step, the indexes are processed to obtain a list of records satisfying the query and in the second, these records are retrieved using this list. The number of disk accesses needed is equal to the number of records being retrieved plus the number to process the indexes.

Inverted files represent one extreme of file organisation in which only the index structures are important. The records themselves may be stored in any way (sequentially ordered by primary key, random, linked ordered by primary key etc.).

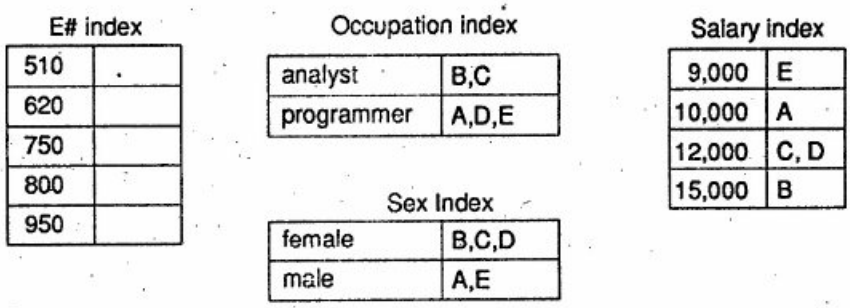


Figure 27 : Indexes for fully inverted file

Inverted files may also result in space saving compared with other file structures when record retrieval does not require retrieval of key fields. In this case, the key fields may be deleted from the records. In the case of multilist structures, this deletion of key fields is possible only with significant loss in system retrieval performance. Insertion and deletion of records requires only the ability to insert and delete within indexes.

3.6.4 Cellular Partitions

In order to reduce file search times, the storage media may be divided into cells. A cell may be an entire disk pack or it may simply be a cylinder. Lists are localised to lie within a cell. Thus if we had a multilist organisation in which the list for KEY1 = PROG list included records on several different cylinders then we could break this list into several smaller lists where each PROG list included only those records in the same cylinder. The index entry for PROG will now contain several entries of the type. (addr, length), where addr is a pointer to the start of a list of records with KEY1 = PROG and length is the number of records on this list. By doing this, all records in the same cell (i.e. records on this list. By doing this, all records in the same cell (i.e. cylinder) may be accessed without moving the read/write heads. In case a cell is a disk pack then using cellular partitions it is possible to search different cells in parallel (provided the system hardware permits simultaneous reading/writing from several disk drives).

It should be noted that in any real situation a judicious combination of the techniques of this section would be called for. I.e., the file may be inverted on certain keys, ringed on others, and a simple multilist on yet other keys.

3.6.5 Comparison and Tradeoff in the Design of Multikey File

Both inverted files and multi-list files have

- An index for each secondary key.
- An index entry for each distinct value of the secondary key.

In either file organisation

- The index may be tabular or tree-structured.
- The entries in an index may or may not be sorted.
- The pointers to data records may be direct or indirect.

The indexes differ in that

- An entry in an inversion index has a pointer to each data record with that value.
- An entry in a multi-list index has a pointer to the first data record with that value.

Thus an inversion index may have variable-length entries whereas a multi-list index has fixed-length entries. In either organisation

- The data record pointers for a key value may or may not appear in some sorted order.
- Keeping entries in sorted order introduces overhead.

The data record file

- Is not affected by having an inversion index built on top of it.
- Must contain the linked lists of records with identical secondary key values in the multi-list structure.

Some of the implications of these differences are the following:

- Index management is easier in the multi-list approach because entries are fixed in length.
- The inverted file approach tends to exhibit better inquiry performance. Many types of queries can be answered by accessing inversion indexes without necessitating access to data records, thereby reducing I/O-access requirements.
- Inversion of a file can be transparent to a programmer who accesses that file but does not use the inversion indexes, while a multi-list structure affects the file's record layout. The multi-list pointers can be made transparent to a programmer if the data manager does not make them available for programmr use and stores them at the end of each record.

Additionally, the multi-list structure has proven useful in linking together occurrences of different record types, thereby providing access paths based upon logical relationships. It is also possible to provide multiple sort orders through a single data collection, by linking the records together in order by various keys.

Check Your Progress

1. What is the difference between B-Tree and B^+ tree.

.....

.....

.....

2. Why a B^+ tree is a better structure than a B-tree for implementation of an indexed sequential file?

.....

.....

3.7 SUMMARY

In this unit, we discussed four fundamental file organisation techniques. These are sequential, indexed sequential, direct and multi-key file organisation. The selection of the appropriate organisation for a file in an information system is important to the performance of that system. The fundamental factors that influence the selection process include the following:

1. Nature of operation to be performed.
2. Characteristics of storage media to be used.

3. Volume and frequency of transaction to be processed.
4. Response time requirement.

We also discussed trade-offs between them.

3.8 MODEL ANSWERS

1. In a B^+ tree the leaves are linked together to form a sequence set; interior nodes exist only for the purposes of indexing the sequence set (not to index into data/records). The insertion and deletion algorithm differ slightly.
2. Sequential access to the keys of a B-tree is much slower than sequential access to the keys of a B^+ tree, since the latter are linked in sequential order by definition.

3.9 FURTHER READINGS

1. Bipin C. Desai, *An Introduction to Database Systems*, Galgotia Publication Pvt. Ltd. New Delhi, 1994.
2. Mary E.S. Loomis, *Data Management and File Structures* (Second Edition) PHI.
3. Horowtz & Sahni, *Fundamentals of Data Structure*, New Delhi.

UNIT 4 MANAGEMENT CONSIDERATIONS

Structure

4.0 INTRODUCTION

Unlike the previous two units where we discussed mainly technical issues related to DBMS i e. file organisation of conventional DBMS, different models of DBMS, in this unit we will focus on administrative aspects of managing data. This unit will comprise issues of organisational resistance, the methodology for conversion from an old system to a new system, the importance of adopting a de-centralised distributed approach and evaluation and administration of such systems. The material stated in this unit would get further strengthen by a specific example of enterprise-wide different management that is being discussed in the following unit.

4.1 OBJECTIVES

After going through this unit, you should be able to :

- identify the factors causing resistance to the induction of new DBMS tools;
- determine the path that must be chosen in converting from an old existing system to a new system;
- list the various factors that are important in evaluating a DBMS system;
- formulate a simple evaluation methodology for DBMS selection and acquisition;
- enumerate the functions of the database administrator; and
- list the check-points and principles which must be adhered to in order that information quality is assured.

4.2 ORGANISATIONAL RESISTANCE TO DBMS TOOLS

Organisations who theoretically and ideally be rationale and their decision making not to be guided by purely an objective approach of its own good. In practice, this does not happen and organisations react to information systems by offering resistance. This is a part of an inherent opposition to change. There are some aspects of change related to information system that arose great passion. This arose because of some of the following factors:

- **Political observation:** The officers and managers at different levels of an organisation feel threatened with the nice long standing political equations and relationships which have enjoyed their otherwise upward movement within the organisation, and may be threatened by a new intervention into their styles of working.
- **Information transparency:** In the absence of an electronic computer-based efficient information system, many functionaries in an organisation have access to information which they control and pass on giving it the colour that would suit them. The availability of information through computer-based systems to almost all who would have an interest in it makes this authority disappear. It is therefore naturally resented.

- **Fear of future potential:** The very fact that computers can store information in a very compact manner and it can be collated and analysed very speedily gives rise to apprehensions of future adverse use of this information against an individual. Mistakes in decision making can now be highlighted and analysed in detail after learning spells of time. It would not have been possible in manual file-based systems or any system where the data does not flow so readily.

Inter-departmental rivalry, fear of personal inadequacy, in comprehensive of the new regime the loss of ones own power and the greater freedom to others and difference in work styles - all add up to produce resistance to the induction of new information processing tools. Apart from these general considerations, there are reasons to resist installation of a new DBMS.

There are several points of resistance to new DBMS tools:

- Resistance to acquiring a new tool
- Resistance to choosing to use a new tool
- Resistance to learning how to use a new tool
- Resistance to using a new tool.

The selection and acquisition of a DBMS and related tools is one of the most important computer-related decisions made in an organisation. It is also one of the most difficult. There are many systems from which to choose and it is very difficult to obtain the necessary information to make a good decision. Vendors always have great things to say, convincing argument for their systems, and often many satisfied customers. Published literature and software listing services are too cursory to provide sufficient information on which to base a decision. The mere difficulty in gathering information and making the selection is one point of resistance to acquiring the new DBMS tools.

The initial cost may also be a barrier to acquisition. However, the subsequent investment in training people, developing applications, and entering and maintaining data will be many time more. Selection of an inadequate system can greatly increase these subsequent costs to the point where the initial acquisition cost becomes irrelevant.

In spite of the apparent resistance to acquisition, the projections for the database industry are forecasting a multi-billion dollar industry in the 1990's. Even though an organisation may acquire a DBMS, there are still several additional points of resistance to overcome.

Simply having a DBMS does not mean that it will be used. Several factors may contribute to the lack of use of new DBMS tools.

- lack of familiarity with the tools and what it can do
- system developers used to writing COBOL (or other language) programs prefer to build systems using the tools they already know
- the pressure to get new application development projects completed dictates using established tools and techniques
- systems development personnel have not been thoroughly trained in the use of the new tool
- the organisation has not set up a program to train users of new DBMS tools
- users are reluctant to use a new tool because there is no one in the organisation to provide advice in its use and to help when problems arise
- tool is only known to a few specialists in the data processing department
- no one in the organisation is compelling even encouraging the use of new DBMS tools
- DP management is afraid of run away demand on the computing facilities if they allow users to directly access the data on the host computer using an easy to use, high-level retrieval facility
- organisational policies which do not demand appropriate justification for the tools chosen (or not chosen) for each system development project.

Having pointed out the transactions from which utility can arise to the intervention of a new DBMS, it may be useful to have a summary of a few pointers which would possibly lead to a greater success in such an endeavour.

Reasons for success	Reasons for failure
Appreciation for information is a valuable corporate reasons and its management must be given special importance.	Perception by the barrens in the organisation that the MIS design is a menace conflicting interest to prevent the success.
Focusing on most beneficial usage of database, which relate to the bottom level.	Over sailing MIS to top management and chosen applications for their challenge to the programmer members.
An incremental approach to building applications with each new step being reasonably small and relatively easy to implement.	A grant design for creation of an impressive system that can be a pinata for all information problems.
Cooperate-wide planning by a high level, empowered, competent data administrator.	Fragmented plans by non-communicating and not eventually response groups.
Conversion planning which permits all the systems to co-exit with the new.	A situation which may put into the new system and attempts to re-write to many old programs.
Awareness education and involvement of all persons at a level appropriate to their functions.	Apathy by most people to implementation of the new system.
Good understanding of the technical issues and tight technical control by the database administrators.	Inadequate computing power, incorrect assignment of throughput and assigned time and failure to monitor usage and performance.
Recognition of the importance of a data dictionary and standards for naming, update control and version synchronisation.	Casual approach to data standards and documentation.
Simplicity.	Confused thinking.
A proper mix of centralised guidance and de-centralised implementation.	Indifference of the central system and proliferation of incompatible systems.
Proven work-free software.	The latest software wonder.

If the above factors leading to success or failure of the project are borne in mind, the chances of a successful implementation and the possibility of organisational benefit from this is greater.

4.3 CONVERSION FROM AN OLD SYSTEM TO A NEW SYSTEM

Management is also concerned with long-term corporate strategy. The database selected has to be consistent with the commitments of that corporate strategy. But if the organisation does not have a corporate database, then one has to be developed before conversion is to take place. Selecting a database has to be from the top down: data flow diagrams, representing the organisation's business functions, processes and activities, should be drawn up first;

followed by entity-relation charts detailing the relationships between different business information; and then finally by data modelling. If the entity- relationship chart has a tree-like structure, then a hierarchical data structure should be adopted; if the chart shows a network structure, a network data structure should be chosen. Otherwise, a universal structure, such as that of a relational database, should be chosen.

Corporate Strategic Decisions: The database approach to information systems is a long-term investment. It requires a large-scale commitment of an organisation's resources in compatible hardware and software, skilled personnel and management support. Accompanying costs are the education and training of the personnel, conversion and documentation. It is essential for an organisation to fully appreciate, if not understand, the problems of converting from an existing, file- based system to a database system, and to accept the implications of its operation before the conversion.

Before anything else, the management has to decide whether or not the project is a feasible one or that it matches the users' requirements. Costs, timetables, performance considerations and the availability of expertise are major concerns too. A pilot project to act as a benchmark is always necessary. A successful data resource management environment must have this management commitment, along with adequate resources in budget, people, equipment and material, a data dictionary, and integrated organisation of people and data in the data-administration section.

Hardware Requirements and Processing Time: The database approach should be in a position to delegate to the database management system some of the functions that was previously performed by the application programmer. As a result of this delegation, a computer with a large internal memory and greater processing power is needed. Powerful computer systems were once the luxury enjoyed by those database users who could afford such systems but fortunately, this trend is now changing. Recent developments in hardware technology has made it possible to acquire powerful, yet affordable system.

Depending on the structure of the data and the access methods to them, the use of a database management system may result in longer processing times. For some database applications the run time can be just as quick - if not quicker than the conventional environment. But if the run times for a majority of cases in the existing environment is so much slower, than the database approach is an unwise decision.

For some applications, the need for high-volume transaction processing may force a company to engineer one or even several systems designed to satisfy this need. This sacrifices a certain flexibility for the system to respond to ad-hoc requests.

And it is also argued that because of the easier access to data in the database, the frequency of access will become higher. Such overuse of computing resources will cause slips in performance, resulting in an increased demand for computing capacity. It is sometimes difficult to determine if the increased access to the database is rally necessary.

The database approach offers a number of important and practical advantages to an organisation. Reducing data redundancy improves consistency of the data as well as making savings in storage space. Sharing data often enables new applications to be developed without having to create new data files. Less redundancy and greater sharing also result in less confusion between organisational units and less tie spent by people resolving inconsistencies in reports. Centralised control over data standards, security restrictions, and so on, facilitates the evolution of information systems and organisations in response to changing business needs and strategies. Now-a-days, users with little or no previous programming experience can, with the aid of powerful user- friendly query languages, manipulate data to satisfy ad-hoc queries. Data independence helps ease program development and maintenance, raising programmer productivity. All the benefits of the database approach contribute to reduced costs of application development and improved quality of managerial decisions.

A principal component of the changeover from a conventional system to a database system is the conversion of data files and applications programs to a form needed by the database management system. The accuracy with which this is done is vital to the success of the database system. Once the programs and files have been converted, the new procedures may be introduced by either parallel running or pilot running. This must be properly planned and controlled, and the necessary instructions must be issued to both users and data-processing staff. When the users are satisfied, the new systems can be handed over, and the database administrator will stay on as part of the maintenance group.

Amid the volatile data processing world where technology advances so rapidly, the data processing manager must satisfy user demands while maintaining an economical operation. Management must set specific goals for developers and users alike to learn the new tools of the database management system. The investments will pay off when users and developers become proficient in accessing data and building systems with the new tools. They reap the economic regards and benefits of the hardware/software capabilities of the database system, the database manner and the database administrator both need a variety of database conversion tools. A new software technology called data translation is being developed at many research institutions, but the research is still in its infancy. Many more years of research is needed to dispel the doubts and fears faced by many processing installations on the decision to go for database.

Database conversion is not an easy task. Depending on each situation, management has to decide which approach is the best one - coexistence, or having two databases, redevelopment, conversion, transparency, or DML substitution and packages. The management must bear in mind the importance of user-learning curve in accepting a new database for the organisation. Also, conversion to a hierarchical or a network database is more difficult than to a relational database because relational databases have simpler data structures. Users need only to define keys in each table file; very often keys are defined with the table files.

In general, converting to a database involves the following:

1. Inventorise current systems such as data volume, user satisfaction, present condition and the cost to maintain or redevelop.
2. Determining conversion priority in strategic information system plans, building block systems and critical needs to replace system.
3. Obtain commitment from senior/top management.
4. Appoint qualified database-administration staff.
5. Education management information systems staff.
6. Select suitable and appropriate software.
7. Install data dictionary first.
8. Involve and educate users.
9. Redesign and implement new data structures.
10. Write software utility tools to convert the files or database in the old system to the new database.
11. Modify all application programs to make use of the new data structures.
12. Design a simple database first for pilot testing.
13. Implement all software.
14. Update policies and procedures.
15. Install the new database on production.

In the recent trend of database development, a common front- end to the various database management systems will often be constructed in such a way that the original systems and the programs on them are not modified, but their databases can be mapped to each other through a single uniform language.

Another approach is to unify various database structures by applying the database standards laid down by the International Standards Organisation for data definition and data manipulation. Public acceptance of these standard database structures will ensure a more rapid development of additional conversion tools, such as automatic functions for loading and unloading databases into standard forms for model-to-model database mapping.

If an organisation after weighing all the relevant factors decides to make an investment in a good database management system, it has to develop a product planned for doing so. Many of the steps required are more or less along the lines that are required when an organisation

first moves in towards the use of computer-based information system. One would immediately note the similarity to the steps referred to in the course on "System Analysis and Design". In the interest of briefing therefore the reference would be only to those factors which are of greater consequences for the problem at hand. It may, however, be useful to bear in mind that a detailed implementation plan would be more or less along the lines of creation of a computer information system for the first time.

4.4 EVALUATION OF A DBMS

The evaluation, is not simply a matter of comparison or description of one system against another independent system, and surveying sometimes available through publication do describe and compare the features of available systems, but a value of an organisation depends upon its own problem environment. An organisation must therefore look at this own needs to evaluation of the available systems.

It is worthwhile putting some attention to who should do this. In a small organisation it is possible that a single individual would be able to do the job, but larger organisations need to formally establish an evaluation team. Even this team's composition would somewhat change as the evaluation process moves on. A good role in the initial stage would be played by users and management focus on the organisational needs. Computers and Information technology professionals then evaluate the technical gaps of several candidate system and finally financial and accounting personnel examine the cost estimates, payment alternatives, tax consequences, personnel requirements and contract negotiations.

The reasons which inspire the organisation to acquire a DBMS should be clearly documented and used to determine the properties and help in making trade offs between conflicting objectives and in the selection of various features that the candidate DBMS may have, depending upon the end-user requirements. The evaluation team should also be aware of technical and administrative issues. These technical criteria could be the following:

(a) SQL implementation
(b) Transaction management
(c) Programming interface
(d) Database server environment
(e) Data storage features
(f) Database administration
(g) Connectivity
(h) DBMS integrity

Similarly there could be administrative criteria such as:

(1) Required hardware platform
(2) Documentation
(3) Vendor's financial stability
(4) Vendor support
(5) Initial cost
(6) Recurring cost

Each of these, especially the technical criteria could be further broken into sub-criteria. For example the data storage features can be further sub-classified into :

(a) lost database segments
(b) clustered indexes
(c) clustered tables

Once this level of detailing is done, the list of features become quite large and may even run into hundreds. If a dozen products are to be evaluated, we are talking of a fairly large matrix.

At this point, it is important for the evaluation teams and especially its technical members to segregate these features into those which are mandatory. Mandatory features would be those which if not present in the candidate system, the system need not be considered further. For example, does DBMS provide facilities for programming and non-programming users? Can be considered as one among several mandatory conditions. Mandatory requirement may also

flow from a desire to preserve the previous investment in information systems made by an organisation. The presence of the mandatory condition means that the system is a candidate for the rating procedure.

Having done the first stage of creating a feature list, one of the simplest ways could be to develop a table where the features and its related information for each candidate system is listed to in a tabular form against the desired feature. Such forms can be chosen to compare the various systems and although this can not be enough to conclude an evaluation, it is a useful method for at least broadly ranking and short-listing the systems. A quantitative flavour can be given to the above approach by awarding points for features which are in simple Yes and No type. If all the features are not equally important to the organisation, then the summing up of the points awarded for each of the features for any of the system is not quite appropriate. In such a case a rating factor can be assigned to each feature to reflect the relevant level of importance of that feature to the organisation. Of course such rating or scoring should be done after the first condition of mandatory requirements have been met by the proposed system. Sometimes the mandatory characteristics may be expressed in the negative as something which the system must not have.

The points of the rates is a contentious issue and must be decided looking only to the needs of the organisation and with reference to the characteristics of any specific candidate system one of the approaches used towards arriving at a suitable set of rating factors is to follow the **Delphi method.** In brief, the Delphi approach requires key people who may be expected to be knowledgeable to make suggestions as to what would be the appropriate rating factor. These are collected, compiled, averages taken and deviation from averages pointed out. This data is then re-circulated to the same set of people for wanting to change their opinions where their own views were varying largely from the average. The details can then be carried out and it has been found that in about as few as 3 to 4 iterations in good consensus emerges.

One of the weakness of the methodologies discussed so far is that they are focusing on the systems but not on the **cost benefit aspects.** A good evaluation methodology should be possibly suggest the most cost effective solution to the problem. For example, if a system is twice as good as another system, but costs only 40% more than it ought to be a preferred solution.

In order to carry a cost after analysis one has to use a **rating function** with each feature to normalize the sequence. Rather than having an approach where a feature is characterised as a Yes/No, the attribute corresponding to its presence or absence which in marks term could be 0 or 1, a mark can be given on a scale which is appropriate to the feature. This can arise in issues such as the number of terminals that are supported or the amount of main memory required. Rating functions can be of several types of which 4 are illustrated in Figure 1.

(a) **Linear:** In a linear rating function the rating increases in proportion to higher marks starting from 0.

(b) **Broken linear:** There are situations where the minimum threshold is essential and similarly there is a saturated value above which no additional value is given. Typically in general concurrent access, few or 3 would be includable value and more than 9 is of no additional value.

(c) **Binary:** This is of course an Yes/No type where a system either has or does not have the feature or some minimum value for the feature.

(d) **Inverse:** There are some attributes where a higher mark actually implies a lower rating. For example in accessing the time to process a standard query, the mark may be simply the time scale in an appropriate manner. Therefore, a shorter time actually has a higher rating.

For each feature, the rating function uses an appropriate and convenient scale of measurement for determining a system's feature mark. The rating function transforms a system's feature mark into a normalised rating indicating its value relative to a nominal mark for that feature. The nominal mark for each feature has a nominal rating of one.

The use of rating function is more sophisticated and costly to apply than the simplified methodologies. The greater objectivity and precision obtained must be weighted against the overall benefits of DBMS acquisition and use. Some features will have no appropriate objective scale on which to mark the feature. The analyst could use a five point scale with a linear rating function as follows:

Feature evaluation	Rating point
Excellent (A)	5
Good (B)	4
Average (C)	3
Fair (D)	2
Poor (E)	1

Variations can expand or contract the rating scale, using a nonlinear rating function, or expand the points in the feature evaluation scale to achieve greater resolution. In extreme cases, the analyst could simply use subjective judgement to arrive at a rating directly, remembering that a feature rating of one applies to a nominal or average system.

Having converted all the marks to ratings, the system score is the product of the rating and the weight summed across all features, just as before. The overall score for a nominal system would be one (since all weights sum to one and all nominal ratings are one). This is important for determining cost effectiveness, the ratio between the value of a system and its cost. The organisation first determines the value of a system which earns a nominal mark for all features. This is called the nominal value. Then the actual value of a given system is the

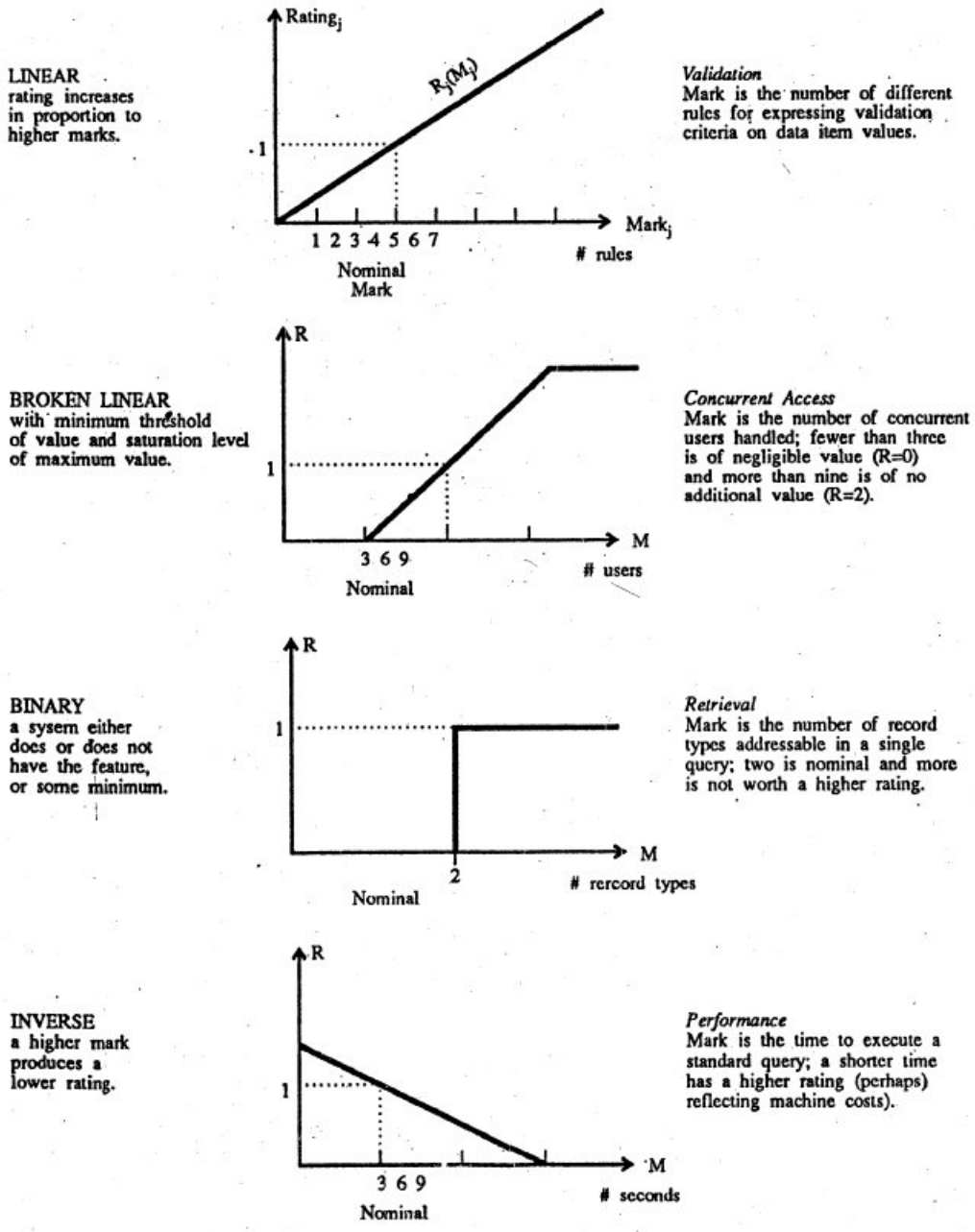


Figure : Sample Feature Rating Functions

product of the overall system score and the nominal value. The cost effectiveness of a system is the actual value divided by the cost of the system. System cost is the present value cost of acquisition, operation and maintenance over the estimated life of the system.

With a cost-effectiveness measure for several candidate systems, the organisation would tentatively select the system with the highest cost-effectiveness ratio.

Of course there may be intangible factors other than the technical and administrative criteria referred to earlier which may influence the final selection based upon political judgements of the management or some other considerations. It would of course be possible to even to build these up of that can be explicitly so illustrated into the evaluation process.

4.5 ADMINISTRATION OF A DATABASE MANAGEMENT SYSTEM

Acquiring a DBMS is not sufficient for successful data management. The role of database administrator provide the human focus of responsibility to make it all happen. The DBA role may be filled by one person or several persons.

Whenever people share the use of a common resource such as data, the potential for conflict exists. The database administrator role is fundamentally a people-oriented function to mediate the conflicts and seek compromise for the global good for the organisation.

Within an organisation, database administration generally begins as a support function within the systems development unit. Sometimes it is in a technical support unit associated with operations. Eventually, it should be separate from both development and operations, residing in a collection of support functions reporting directly to the director of information systems. Such a position has some stature, some independence, and can work directly with users to capture their data requirements. Database administration works with development, operations, and users to coordinate the response to data needs. The database administrator is the key link in establishing and maintaining management and user confidence in the database and in the system facilities which make it available and control its integrity.

While the 'doing' of database system design and development can be decentralised to several development projects in the Data Processing Department or the user organisations, planning and control of database development should be centralised. In this way an organisation can provide more consistent and coherent information to successively higher levels of management.

The functions associated with the role of database administration include:

- Definition, creation, revision, and retirement of data formally collected and stored within a shared corporate database.
- Making the database available to the using environment through tools such as a DBMS and related query languages and report writers.
- Informing and advising users on the data resources currently available, the proper interpretation of the data, and the use of the availability tools. This includes educational materials, training sessions, participation on projects, and special assistance.
- Maintaining database integrity including existence control (backup and recovery), definition control, quality control, update control, concurrency control, and access control.
- Monitor and improve operations and performance, and maintain an audity trail of database activities.

The data dictionary is one of the more important tools for the database administrator. It is used to maintain information relating to the various resources used in information systems (hence sometimes called an information resource dictionary)—data, input transactions, output reports, programs, application systems, and users. It can :

- Assist the process of system analysis and design.
- Provide a more complete definition of the data stored in the database (than is maintained by the DBMS).

- Enable an organisation to assess the impact of a suggested change within the information system or the database.
- Help in establishing and maintaining standards, for example, of data names.
- Facilitate human communication through more complete and accurate documentation.

Several data dictionary software packages are commercially available.

The DBA should also have tools to monitor the performance of the database system to indicate the need for reorganisation or revision of the database.

Check Your Progress

1. List factors which motivate the move to acquire the DBMS approach.

.....

.....

.....

2. What are some of the purposes of a data dictionary?

.....

.....

.....

.....

4.6 SUMMARY

The process of selecting, evaluating and finally acquiring a DBMS package takes a substantial time and efforts. The tasks begin when the need and requirement of an organisation and user is strongly felt. Designing such package in-house is not a realistic alternative with more and more reasonably good commercial system available in market.

The important criteria for selection of DBMS are technical and administrative criteria. The key technical criteria relate to the type of system required, balancing the competing objectives of efficiency and functionality. Administrative criteria include vendor charactcristic maintenance support, documentation, training and ease of learning and use, cost etc.

An organisation will live with chosen DBMS for several years. If the initial study and selection is done with a broad view or organisational needs now and into the future, the choice can enhance data processing is responsiveness to user needs, managerial difference effectiveness and organisation profitability.

4.7 MODEL ANSWERS

1.
   - Faster response to queries
   - Faster application development
   - Data sharability
   - Reduced program maintenance
   - Adaptability to changing requirement
   - Increased security
   - Transferability across hardware
2. The foundation of the data dictionary is information about data items with a comprehensive base of information, the data dictionary can serve several useful

purposes. These purposes span the whole spectrum of planning, determining information requirement, design and implementations, operations and revision.

- Data availability : A data map for end users to discover what data exists in the organisation, what it means, where it is stored and how to access it. May be provided using a facility for browsing through a data dictionary.
- Documentation : Providing reports of data about data. The data dictionary can be used to generate a graphical representation of database structure similar to automatic program flowcharting. In a general sense, the data dictionary is a vehicle for managing size and complexity in a database environment. In a typical single-function organisation (not a mixed conglomerate) the individual data items will number in the several thousand. The data items appear in hundreds of files (record types) which are interrelated and in hundred of input transactions or data capture screens and output reports.

4.8 FURTHER READING

Everest, Gorden C., *Database Management Objectives System Functions & Administration*, McGraw Hill International Editions, 1986.

UNIT 5 ENTERPRISE WIDE INFORMATION SYSTEM OF THE TIMES OF INDIA GROUP (A CASE STUDY)

Structure

5.0 INTRODUCTION

The Times of India is a leading Publishing House of India. The Group is implementing an Enterprise wide Information System to help it realise its strategic goals. One component of the System is RESPNET. Respnet is discussed at some length. The problems faced in implementing it are mentioned. The role of Ingres in the Information System is touched upon.

5.1 OBJECTIVES

After going through this unit, you should be able to:

- understand the need for an Enterprise wide Information System in a large Publishing House;
- understand the meaning and complexity of an Enterprise wide Information System in a large organisation with offices in several places throughout the country;
- understand the components of an Enterprise wide Information System;
- understand the details of the implementation of an Enterprise wide Information System;
- understand the problems encountered during the implementation of an Enterprise wide Information System;
- understand the benefits of an Enterprise wide Information System;
- understand the role of a commercially available Database Management System (Ingres) in implementing an Enterprise wide Information System; and
- appreciate the need for continual enhancement and technological upgradation in maintaining an Enterprise wide Information System.

5.2 ORGANISATION AND THE OPERATING ENVIRONMENT

The Times of India Group is the leading publishing house of the country. The group publishes three national newspapers, two regional ones and one evening paper besides a few magazines. For over 155 years now the Times of India has consistently maintained its position as the flagship among Indian dailies. Its six editions can boast of a combined circulation of over 500,000 and a readership of over 2,000,000. Independent surveys have

shown that about 70% of Indian decision makers read the Times of India. The Economic Times, the national financial paper, enjoys a pre-eminent position in its category.

The combined annual turnover of the Group is Rs. 3 billion (about US $ 100 million). It has 12 Major branch offices of which 9 are publishing centers. In addition there are over 40 smaller marketing offices.

In addition to its publishing activities, the Group produces software for Television and is a major purchaser of Radio (FM) broadcasting time. It also has a company offering financial services. Other expansion plans are afoot.

The Group has to operate in a fiercely competitive business environment. The rivals include other newspapers and magazines and other media, mainly television. With the expansion in the reach of national and satellite television, there has been a perceptible shift in the preference of people away from newspapers and towards television and video. This decline in the reading habit has affected all newspapers and publishers and has led to greater competition among them.

This has led to a situation not unlike that in any other business activity. Timely and accurate commercial information has become indispensable not only to grow and to thrive, but for mere survival as well. The information required can be classified broadly into that gleaned from external entities and the portion which originates in the organisation itself.

While computers are of immense help in both these pursuits, this paper discusses mainly the experience of the Information Services Division (ISD) of the Times of India Group in making available internal commercial information to various levels of management. The task would be a mammoth one anywhere, but in a developing country like India it poses an additional set of problems to those carrying it out. The process is still on, though good progress has been made already.

5.3 UNIQUE NATURE OF THE BUSINESS

Before going on to sharing ISD's experience in more detail, it would not be out of place to elucidate the dynamics of the newspaper business. It has certain features that make it quite different from the typical manufacturing or trading corporation. These features have had their impact on both the approach to the design of the information system as well as on the development of its specific components.

There are three main functions in this industry. These are the editorial, which gathers news, articles, syndicated columns and the rest of the contents of the newspaper for which the majority of the readers read and buy it. The talent in this area consists of the journalists who lend character to the contents of the paper. Excellence in editorial and news content helps boost circulation, which is the life blood of the newspaper. As is known even to lay persons, the income from circulation forms only a small part of the earnings of a newspaper. A large circulation helps increase the columnage of advertisements carried. Advertisements are the major source of revenue for a newspaper and determine its financial fortunes.

Of course, success in selling advertisement space means enough resources for the newspaper to plough back into improving its editorial content, in improving facilities for staff functions and generally all over the organisation. This is the circle which drives this line of business.

The absence of any reference to the actual production process will not have gone unnoticed by discerning readers. While its importance cannot be overstated, the potential benefits of excellence in manufacturing are cost saving due to reduction in wastage and the like. The potential as regards increasing revenue is only peripheral, by using good paper, producing legible copy and so on. Hence the printing process has not been mentioned while describing the main business cycle of a newspaper.

Since advertising is the major revenue earner, it assumes the greatest importance in any information system built for a newspaper. The peculiar characteristics of its functioning therefore need to be elaborated upon.

In a Newspaper Group like the Times of India, advertisements may be booked from any branch office for publication in any edition of any newspaper published by the Group, which is often called a product. Different products may be published from different sets of branches, and each of them could have a different value as far as advertising worth to clients

goes. Hence the situation is different from a multi-location manufacturing organisation. In such an organisation, although the same product could be manufactured from different facilities located at different geographical sites, there would not be any difference between them. A difference would exist only if they were different products.

For example, consider a company which manufactures washing machines and refrigerators, the former from locations A, B, C and D, and the latter from B, C and E. If an order is received for refrigerators, they can be supplied from B, C or E. Similarly washing machines can be supplied from A, B, C or D and there is no difference between appliances supplied from different locations.

This is not the case with advertising space. If a newspaper 'x' is published from A, B, C and D, then the value of advertising in 'x' at A is different from the value of an advertisement in the same newspaper 'x' at B, and so on. This is because of the different circulation figures and readership profiles at different locations. And yet the situation is not like that of two different products, since the editorial content is largely the same.

5.4 SHIFT IN STRATEGY

Upto 1985, the Group was content with the way things were going for it. It had steady business, there were no dangerous rivals and there were adequate profits to be had. The Group was conservative and there was no attempt at innovation in the area of marketing.

Around that time, there was a shift in the thinking of the owners and it was decided to gain a leadership position and expand aggressively. It was also decided to change the focus of activity from mere publication of a newspaper to becoming an information and marketing company. This meant expanding the activities of the Group to other media, syndicating news and other such extensions. Thus the Times of India Group pioneered the colour newspaper in the country. Such innovation required a change in the way things were done and above all it meant that a strategic plan would have to be conceived of and executed.

The exact nature of the plan will be of interest to students at a business school more than to the readers of this paper, and so it will not be dwelt upon. But what would certainly be of interest is the fact that even the development, let alone the execution of such a strategic plan was handicapped by the absence of information. This was a pity because the data already was available to the organisation, either within or from external sources.

The tactical aspects of the plan required that changes be made in the way the activities of the various functions were carried out. It was not possible to do this using the primitive software available and working at that time.

The above difficulties meant that the changed philosophy of the Group could not really show up in its working. However, whatever little was possible was done, and in particular, the importance of timely and accurate information was brought to the notice of the management.

5.5 INFORMATION SYSTEM GOALS AND HOW TO ACHIEVE THE GOAL

With this background, it will now be possible to elaborate on the Information System goals of the Times of India Group and then on the experience of ISD in achieving them. The central role played in this by commercially available database management systems will also be discussed.

Over three years ago, ISD conceived of and embarked on the task of designing, developing, implementing and maintaining an Enterprise wide Information System for the Times of India Group. The system was to embrace all the internal commercial information needs of the Group. The various functions at major locations are Response (as advertising is called internally), Circulation, Inventory, Finance, Transport, Newsprint, Personnel and others.

At each location these systems would be put together into an Integrated Information System for the location. Similarly most functions were carried out at different locations. So the functions were to be integrated across geographical locations as well. The result would be an Organisation Wide Information System.

The goal of this system was to provide any authorised user access to any piece of information required, even if this meant that the data had to come from all over the country. The operation of the system was to be as user friendly as possible. The user was to be transparent to any task except posing his query and obtaining the result.

The system was to operate such that the activities of the various functional areas were all performed with the help of the computer. This would ensure that all data entered the system at source. No subsequent transcribing process would be necessary. All users would do their own data entry, pose their own queries to the system and satisfy all their information needs.

Given this operational information system, it was decided to build a decision support system around the data available. This would help all levels of users to make decisions based on all relevant data.

The above goal was recognised as an ambitious one and one that would take a long time to be attained. The task would be a mammoth one in any part of the world, but was recognised as being especially difficult in a country like India, with weak power and telecommunications infrastructure.

There are various components involved in this kind of endeavour. Broadly, these are the hardware, the operating system, the database management system, the application, the physical network and the networking software. In addition there would be the man-machine interface.

Here the main emphasis will be on narrating the experience of ISD in developing the applications and the role of commercially available database management systems in this. The network choices and solutions will also be discussed, as they show the special problems ISD has had to solve, and as they are related to database issues.

It would not be out of place here to mention that when ISD embarked on this task, it was not as if the organisation was devoid of computers or commercial applications. However, the applications then running were typically batch operations written in third generation languages. This helped in that users were somewhat familiar with computers and did not harbour any dread of the machines. However, it also meant organising a smooth transition to new applications with as little disruption in operations as possible.

Given the importance of the Response function to the organisation, this was the natural choice to begin the changeover to the enterprise wide information system. First the task of rewriting the whole application was started. As each usable module was ready, the implementation and transition operation was performed. Since ISD is based at Delhi, this was the first office to have the new system. The Response system is discussed in more detail later.

5.6 IMPLEMENTATION PLANS AND PROBLEMS DURING THE IMPLEMENTATION

The broad plan of implementation was to install the Response system in standalone mode at all offices, while networking these together subsequently. This involved changes to the application to make it a true network application and use the myriad possibilities this opened up. The work on the other applications was to continue simultaneously. All such applications like the Finance System were to be implemented at Delhi first as this allowed ISD to observe and support them easily.

The decision support system for the Response function at the operational level was gradually developed together with the implementation of RESPNET, as the Response software system was christened. This system was to be refined as and when more possibilities were suggested by the users. By this time ISD is working on decision support facilities for top management. This includes support for pricing decisions and tariff structures.

The task of integration was to be taken up in various phases depending on the situation. Thus, in the case of Response, it was felt convenient to integrate vertically across branches rather than first wait for other applications to come up at a branch. So integration was begun in both directions at the same time. For example, the Finance System under development and implementation at Delhi is linked to the Response and Circulation applications. It will soon be connected to the Inventory and Personnel systems. The exact order in which the

integration of various software systems was to be done was felt to be not important. It was therefore decided not to work out the detailed plan in advance and to integrate in the most convenient order.

Problems during the Implementation

These were the decisions taken during the early stages when the Response software was yet to be developed. As already mentioned, this was the first package taken up for development and implementation. The experience gained during this task will now be narrated, together with the problems faced and how these were overcome.

The difficulties were broadly of two kinds—those concerning the physical system of the Response Department and those with the network. There were issues with the reliability of the hardware and the UNIX ports available in India, but these have been solved by now with the availability of international brand names which offer reliable UNIX boxes, capable of providing the uninterrupted service required from the hardware and with well tested system software. This problem might be peculiar to the country, but it underscored the need for good hardware and reliable software. The ISD lost a large amount of time in dealing with hardware crashes and unreliable operating systems during all stages of the development and implementation of the Information System. Otherwise the progress made so far would have been much greater.

Another twist to the issue of reliable software was that there was no certified port of Ingres on the hardware available to ISD. This resulted in various problems with Ingres which perhaps have not been faced by others elsewhere. However these need not be dwelt on now that they are behind us.

There were two major difficulties encountered, with data lines and with the physical operations of the Response department. The leased lines available to ISD were all 4800 baud lines with a fall back to 2400 baud. There was a standby line offered with each main line. However the uptime of the lines was very low and even when working, the lines were noisy, resulting in a very slow speed of operation. This problem was found to be mainly with the local leads at the various centres, as the long distance lines available, even in India, were of quite good quality. The long distance network is rapidly being converted to use fibre optic cable, ensuring first class transmission. However, the same could not be said for the local network. To get around this difficulty, dedicated cables were proposed at all centres from the organisation's premises to the telephone exchange concerned. This helped reduce the problem somewhat. However, the reliability of lines is an issue which still has not been resolved and one for which no solution is in sight yet. It is hoped that at some time good quality lines will become available to ISD.

Irrespective of the issues concerning the network, it was decided to have local databases for each main branch so that local operations and autonomy were not compromised in the event of failure of any segment of the network. This meant that the full Response system consisted of a system of loosely coupled, co-operative databases. The coupling is weak because although in the normal situation the local databases interact extensively, it is possible for a centre to be cut off from the network and still function normally for operations related to that branch. As soon as the network is re-established, the interaction with other databases starts again. This is a less than desirable situation, but the only feasible solution given the compulsions of the reliability of the network.

Because of this situation, certain other problems came to the fore. Thus it was found impossible to implement a fully on-line network application because of the line conditions. The way out was to write software which would try and work on-line and fall back to batch updates as and when the line was available. After some experimentation the idea of having a transaction based on-line application had to be postponed indefinitely as it was found almost impossible to be on-line for any reasonable length of time. The Response system is thus currently one where updates take place across the network at the earliest point in time possible, depending on the availability of the data lines. However, it must be emphasised again that this has not been done as a matter of choice, but after failing on more desirable options. Whenever the line quality improves, the application will be made fully transaction based.

This brings one to another problem concerning commercially available databases. While Ingres was very useful in quick application development, robust local operation, satisfactory speed and so on, the fact was that the developers of Ingres who had conceived of and implemented its network capabilities came from a different cultural background. The Ingres

network related products were all built for a situation where reliability of telecommunication lines was taken for granted. Hence there were no features to allow for recovery after a network breakdown, much less to cater to a situation where a network was partially down almost all the time.

It was therefore not found possible to use Ingres network products for the Response system application, although it must be again emphasised that this is no reflection on the quality of those products. The problem was environmental and cultural. However as far as ISD was concerned, it was necessary to write routines for taking care of the resulting problems. This has been done and the necessary subroutines are available to the application developers for their use.

In this connection, it must be mentioned here that late last year, Ingres did realise this possibility, and released commercially a product which takes care of a situation where communication links are down. The services provided by this product are very similar to what ISD had had to develop to get around the difficulties faced with the network. If only this product had been available earlier, ISD could have saved a lot of time in moving towards its information service goals.

An entirely different set of difficulties was encountered because of the fact that the earlier operations were not networked. Since each branch functioned independently, the physical operations everywhere were slightly different. This meant that implementing a single package all over the organisation required changes in operation all over. Every branch had to change somewhat in order to make operations uniform. Apart from this, certain master information like agency codes, rate codes and the like had to be made uniform all over the country. This required several rounds of discussions among the branches and a lot of organisation wide data processing exercises. At the end of all these, the operations at branches were made uniform.

The above process could not take place at one stroke. There were several changes made piecemeal which affected the software development process as well. The software had to be changed repeatedly to accommodate changes found to be required in the physical operations of Response. In fact, the implementation of the software resulted in several changes in the Response Department.

5.7 THE RESPONSE SYSTEM AND RESPNET CHOICES

The Response system can now be described at some length. As already mentioned, this is the main revenue earner for the organisation, and hence commands the greatest attention from the management. The organisation publishes three national, two regional and one evening newspaper from 9 branches, with two more branches being sales offices. There are also over 40 minor sales offices, each being attached to some branch or the other. There are also some nationally popular magazines published.

The tariff structure for publishing advertisements in these publications is somewhat complex. The rate chart of the Group has been felt worthy of a name, Mastermind. There are four main categories of rates—single, multiple, super and slam. A single insertion is carried at the single rate. Various combinations of insertions merit better rates, of which the best for the client is the slam. The actual rate in any category depends on the publication and the publishing centre. The rate can also vary depending on the category of the advertisement. There are four major categories—display, tenders, appointments and financial. Classified advertisements have a different rate structure altogether and there is a completely different module in the software which deals with them.

In any category, publication and centre, the rate can attract discounts or premiums. Discounts are offered on certain kinds of advertisements, for example, those promoting books, or on volumes like full page advertisements. Premiums are charged for special positions or pages and for solus advertisements. The rates for colour and black and white advertisements are different.

The real complexity of the rate structure arises because of linkages between publications. There are various advertisement package deals available to clients, where the publications and the total rate are fixed. In some packages, a few publications are fixed and the client has the privilege of choosing the others. There are linkages among newspapers and magazines as well.

This is further complicated by the possibility of cancellations. Suppose an agency books an advertisement to be carried in the Times of India at four centres. This would attract a super rate. Now if the agency cancels two of these insertions, it will be entitled to only a multiple rate, which is higher. So the cancellation process involves not only marking the two insertions which were actually cancelled, but must also affect the two other insertions which were booked with it. And it should be remembered that all these could be located at different centres so that the information has to travel over the network, transparent to the operator at the booking counter. There is thus strong interaction among different centres.

In addition to all this is the fact that all the publications of the Group are not part of the same company. So the software must handle a multi-company scenario. For advertisements booked by foreign entities, payments have to be collected in a currency different from the domestic rupee. The software therefore needs to take care of different currencies.

The advertising rates and the structure itself is subject to frequent change, usually at least twice an year. For the purposes of management information, an year's report could require data coming from at least two different rate cycles. Therefore a history of past rates and structures has to be maintained.

This was a brief description of the system on the advertisement booking side. The collection and follow up of outstanding amounts forms a major sub-system of the software. It is part of the accounting module. The situation is that the organisation interacts directly with several kinds of parties who book advertisements—accredited agencies, non accredited agencies, government agencies and direct clients. All of these but the last are entitled to deferred payment. This period can go upto 60 days.

A major effort in the Response Department is that of producing bills correctly and on time. Although an agency might have booked an advertisement for a particular size, exigencies of the situation might result in the advertisement being published in a somewhat different size. Such a change could also occur by mistake. When billing a client, he cannot be charged a higher amount but must be given the benefit of a lower amount if a smaller size has been published. This requires that the actual size of an advertisement be fed into the system after publication. Similarly advertisements that are held over due to any compulsions on the part of the publishers cannot be billed. Thus booking information is only a rough guide to billing.

The financial health of the company depends to a large extent on proper follow up on these receivables and their collection. Most of the major agencies have offices and clients at more than one place, and they book advertisements for various branches. When collections are made, they need to be matched against the bills which are being discharged. If this process is not gone through, it results in what are called unmatched credits. These are a problem as far as accounting is concerned.

Apart from all this, there is the question of research on competition. The columnage of advertisements in various categories is captured by the software. The revenue of rivals is then estimated based on their published rate charts. Not only this, the gist of the actual advertisements is stored and compared with the organisation's publications to determine advertisements and campaigns which went to rivals but were denied to the Group. This helps the sales staff in planning and follow up.

The complexities of accounting will now be touched upon. Earlier, the Response operations were on what is internally called the "A" system. This meant that the publishing branch was responsible for billing and collection, irrespective of where the actual booking or payment was done by the client. Credit for revenue was given to the publishing branch. This resulted in difficulties in collection because usually it was the booking branch which was in a better position to collect the receivables. Since the booking branch got no credit for collecting the money, there was lack of vigorous effort on its part in this direction, resulting in large receivables organisation wide. This system was followed by what was called the "B" system. This gave credit for revenue to the booking branch. For a long period, billing was done on "A" basis by the publishing centre and collection was done by the booking centre.

This system was switched to the "B" system meaning responsibility and credit on booking basis for all activities. This helped billing to be more accurate as the booking branch is best equipped to bill. The next step would be consolidated billing for a client, possibly from a centralised facility. That would really eliminate almost all problems that currently occur in billing, as the client would be presented with a single bill for the whole billing period.

Given all these complexities, the Response software was expected to help in client servicing

and help in pointing out the loopholes in the operations of the department. It was necessary to have a good system of obtaining operational information, help in making decisions and tighten controls to help plug revenue leakage.

There were various choices to be made to implement just this one component of the Information System. These were on all components of the System, comprising of the hardware, the system software, the database management system, the application, the networking protocols and the choice of the backbone network. The options available and the selections made will now be explained.

The organisation had a fairly large amount of hardware available, but this was mostly of Personal Computers being used for office automation. The hardware used by the applications then running was a heterogeneous mixture of machines of varying power from different vendors. It was necessary to decide whether to try and use the existing hardware or procure new machines without reference to those existing then. It was felt that the best course would be to try and use whatever hardware was available to the extent possible. Later more hardware could be procured after some success had been attained and the management was alive to the possibilities from the Information System. Accordingly the existing hardware was used for implementing the new Response software package and only later were new machines procured to run it. The old hardware was shunted off to other applications and other, smaller, centres. Some of it has now been discarded as it had outlived its useful life.

As far as the operating system was concerned, it was felt that it was imperative to have a standard operating system which was open and not tied to any particular vendor or hardware. The natural choice was AT&T Unix (SVR 3.2) at that time since this was the operating system in use in the organisation for various applications.

Fortunately at that time various relational database management packages had become available commercially. This enabled a much quicker and easier path to application development, as otherwise writing the Information System in a third generation language would have meant a large lead time in developing the various tools to be used by the application team — forms packages, code generation routines and the like. Also database management packages provided various features like security, power fail backup and so on. After studying various options available and evaluating the leading RDBMSs for current features and the roadmap for the future, Ingres was chosen as the RDBMS for all application development work. All 3GL work would be done in ANSI 'C' or in C++.

For the networking it was decided to use TCP/IP running over X.25. This would enable reliable communication with facilities for automatic routing transparent to the end user or even the application developer. It was felt necessary to have at least two routes between any two pairs of nodes given the unreliability of the backbone network.

For the physical network, there was not much choice as the availability was restricted to using dial up telephone lines or leasing 4-wire, full duplex, data circuits offered by the relevant authority for public use. The choice was for leased lines between major branches given the potentially large volume of traffic and dial up facilities for minor offices.

The topology of the network was also dictated by the kind of application envisaged. For a file transfer kind of application based on central processing and control, a star network might have been appropriate. However it was desired to have a transaction based true on-line network application.

The modalities of application development also had to be decided upon. The possibility of using external assistance for developing the system was considered. However, knowing that adequate technical expertise was available in ISD, it was decided to develop the Information System completely in-house.

Wherever any special purpose utilities or tools were found necessary they would be developed in-house unless the effort involved was disproportionately large.

5.8 BENEFITS

The benefits from RESPNET are already becoming apparent. One of the things to be noted about the software is that it has evolved over the years rather than being designed at some point of time. It now covers the entire gamut of Response operations and now covers all its activities, from booking, billing, accounting and credit control to market research. The

foremost advantage is that client servicing has improved tremendously. For example, client queries can be now answered speedily. Earlier if a client put in a query about rates or the total amount to be paid for a package, it used to take several minutes to work out the figures and explain the various combinations possible. This can now be done in seconds. Again, questions like whether an advertisement for another centre actually got carried can now be answered almost immediately. Billing has become more accurate and complaints on that score have reduced quite a bit. Revenue leaks on account of cancellations not resulting in rate changes have also vanished.

The natural spin-off from this has been more revenue. It has also been possible to implement the complex rate structure because of the Response software. Without this, the rate chart would have been simpler. While the advantage might not be apparent easily, a more customised rate structure results in greater revenue by enabling rates to be pegged closest to what the market will bear. Strong publications can now charge higher rates than those at low revenue centres. It may be mentioned here that advertisement revenue has increased by 98% over the three year period from 1990 to 1993. This averages to a growth rate of 25% per annum.

What is more, it is now possible to evaluate the impact of rate changes on revenue. This was an exercise which could not have been even attempted before. This helps in making decisions on rate changes. More data is available to the decision makers in this respect, whereas such decisions were done more on the basis of management perceptions and less on any facts available.

Another benefit is in credit control. Earlier, this was possible at a local level only. It was a difficult exercise getting the account of an agency across the country and when such a statement was compiled, it would be out of date. Such information can now be quickly compiled and the appropriate action taken.

It is now possible to know who the top customers of the organisation are. Since the actual clients do not normally interact directly with the organisation, information about the biggest end clients was earlier not available easily. This is now known and is an important input to the marketing team.

The different publications of the Group at the various locations are not all part of the same company. The same publication can be part of a different company in a different location. Now while advertisement rates and the like need to be uniform and the agencies need not take this into consideration, the accounting has to be done separately for each company. However, the Management needs information on Response without reference to companies. This situation is easily handled by the Response software by taking different views of the data. This was not easy to do earlier.

Apart from decision support, the software now makes available to the users an effective operational information system. This takes care of many day to day problems faced earlier. For example information on banned agencies now is flashed to all offices within minutes, and actually an agency can be banned at all offices from any office. This obviates the need for notifications going to various offices and then getting entered into the system, which arrangement could result in advertisements from the agency concerned getting published in spite of a ban on the agency.

Such an operational information system has the desirable side effect of plugging revenue leaks. Some of the leaks were due to operational problems like the matter of banned agencies. Others are concerned with malfeasance on the part of some employees. Thus earlier the availability of certain premium positions in a publication at a location was known only to the persons concerned there. People at other offices could not quickly get to know about the availability of such space. This left room for various malpractices at that location. Now persons at any office can query the availability of space at any other office. This precludes any person from taking advantage of the lack of this information.

Another useful feature now available to response users is the ability to send e-mail, converse with colleagues at any office and even have a conference. Many small points can now be discussed like this over the network without having to make an expensive phone call or having to travel. Conferencing facilities over the network have reduced the amount of travelling done by executives to sort out difficult issues. The savings on these heads alone more than recover the costs of setting the network up.

The result of all these benefits from the Response system is a great competitive advantage to the organisation. None of the rivals have anything close to offer as yet, and it is no surprise if clients find ours a pleasant organisation to deal with. The effort at improvement is to continue so as to provide clients and internal users the best environment possible.

5.9 FUTURE

Some of the enhancements to be made to the Response system as it now exists are now touched upon. These are plans and could be changed if found necessary. However they follow as logical steps to build upon the foundation already laid.

The first enhancement would be to extend the system to all minor offices as well as the major ones. This will entail setting up the system at over 40 additional locations, as well as expanding the network to include all those cities. An extension of this phase of implementation would be to allow major agencies to dial into the network and have limited operational access. Thus an agency would be allowed to only make reservations for space and not to actually book an advertisement. It could be allowed to look at its accounts only and not of any other party. It could query the availability of space at any location and could check up on whether an advertisement has been published or not. This sort of extension to the system will be a big change for the agencies.

The Group rate chart being so complex, another feature planned is to provide media planning services to all potential clients. This would enable an analysis to be done of how to maximise reach in a given readership profile at minimum cost. The software would have to be sophisticated enough to be of real use. At present there are quite a few combinations of publications which are not even examined because there are so many of them.

The decision support facilities will be enhanced to help all levels of users, from the order entry operators to top management.

A feature which would greatly improve productivity apart from any other benefits is that of transmitting the advertisement matter, whether text or photographic, over the network to other offices for direct incorporation into the paper. This would necessitate good quality, fast data lines as well as a good page layout software module. At present this is done manually.

To summarise, RESPNET will be continuously improved so as to be of even more benefit to the organisation.

5.10 SUMMARY

The somewhat detailed treatment of Response was only indicative of the magnitude of the task before the organisation. ISD has to put in a comparable amount of effort in each functional area and then integrate the various software systems vertically and horizontally before the goal of having an enterprise wide information system and a decision support system is anywhere near realisation.

The individual applications continue to grow in complexity as has been found while developing the Response system. So the task of improving and refining the software as well as the user interface will pose a continuous challenge.

When the implementation of only one component of the Enterprise wide Information System, an implementation which is only partially complete, has resulted in such great benefits visible to the clients, the management and those who operate the system, it is certain that when the complete Information System is in place, it will mean changes nothing short of revolutionary for all concerned.

Before closing the paper, a few words on the database management system used for this task, Ingres, will be in order. Ingres has been very useful in this whole endeavour because of the very good productivity it has been possible to achieve. Otherwise the task of writing the Response software alone would have been a herculean one. There has been no loss of data so far in spite of hardware and power failures. Never, so far, has it been necessary to use backups to retrieve data. The robustness of Ingres has been a great comforting factor during the long years of intensive effort at developing and implementing a package which already has over 200,000 lines of 4GL code, not counting any comment lines. Repeated changes which had to be made to the software were possible because of the ease of coding in Ingres.

It must, however, be mentioned that this endorsement of Ingres is based on the actual experience of the organisation and does not in any way insinuate that other database management systems available commercially are inferior or could not have been used to develop and implement the enterprise wide information system discussed in this paper. But at no time did the organisation have occasion to regret having chosen Ingres.

The task of having the information system in place is far from complete. There will be various problems to be solved and experiences to be gone through, which can perhaps be described in papers and discussed at conferences in the years to come.

Uttar Pradesh
Rajarshi Tandon Open University

BCA-1.5

Introduction to Database Management System

Block

2

RDBMS AND DDBMS

UNIT 1

Relational Model

UNIT 2

Normalization

UNIT 3

Structured Query Language

UNIT 4

Distributed Databases

Expert Advisors

Prof. P.S. Grover
Professor of Computer Sciences
University of Delhi
Delhi

Brig. V.M. Sundaram
Co-ordinstor
DoE-ACC Centre
New Delhi

Prof. Kamesha
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Prof. L.M. Patnaik
Indian Institute of Science
Bangalore

Prof. M.M. Pant
Director
School of Computer and
Information Sciences
IGNOU
New Delhi

Dr. S.C. Mehta
Sr. Director
Manpower Development Division
Department of Electronics
Govt. of India
New Delhi

Dr. G. Haider
Director
Information Technology Centre
TCIL, Delhi

Prof. H.M. Gupta
Department of Electrical Engineering
Indian Institute of Technology
Delhi

Prof. S. Sadagopan
Department of Industrial Engineering
Indian Institute of Technology
Kanpur

Prof. R.G. Gupta
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Prof. S.K. Wason
Professor of Computer
Science
Jamia Millia
Delhi

Dr. Sugata Mitra
Principal Scientist
National Institute of
Information Technology
New Delhi

Prof. Sudhir Kaicker
Director
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Faculty of the School

Prof. M.M. Pant
Director

Mr. Akshay Kumar
Lecturer

Mr. Shashi Bhushan
Lecturer

Course Preparation Team

Prof. M.M. Pant
Director SOCIS
IGNOU

Mr. Millind Mahajani
Manager
Infornation Services
Time of India Group
New Delhi

Dr. N. Parimala
Birla Institute of Technology
and Science, Pilani

Utpal Bhattacharya
NIIT
New Delhi

Mr. Shashi Bhushan
Lecturer, IGNOU

Block Writer
Mr. Shashi Bhushan
Lecturer. IGNOU

Course Coordinator
Mr. Shashi Bhushan
Lecturer, IGNOU

Print Production : Sh. Jitender Sethi, APO, MPDD

March, 2003 (Reprint)

ISBN-81-7263-866-3

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi - 110068.

BLOCK INTRODUCTION

This block describes topics related to RDBMS and Distributed DBMS. One of the main advantages of the relational model that it is conceptually simple and more importantly based on mathematical theory of relation. It also frees the users from details of storage structure and access methods. In a distributed database system, the database is stored in several computers. The computers in a distributed system communicate with each other through various communication media, such as high speed buses or telephone lines. They do not share main memory nor do they share a clock. There are 4 units in this block. Three units are related to RDBMS and one unit is on DDBMS.

Important issues discussed in this block are :

- Relational Algebra
- Normalization
- SQL
- Data Replication and Data fragmentation.

UNIT 1 RELATIONAL MODEL

Structure

1.0 INTRODUCTION

One of the main advantage of the relational model is that it is conceptually simple and more importantly based on mathematical theory of relation. It also frees the users from details of storage structure and access methods.

The relational model like all other models consists of three basic components:

- a set of domains and a set of relations
- operation on relations
- integrity rules

In this unit, we first provide the formal definition of a relational data model. Then we define basic operations of relational algebra and finally discuss the integrity rules.

1.1 OBJECTIVES

After completing this unit, you will be able to:

- define the concepts of relational model
- discuss the basic operations of the relational algebra
- state the integrity rules

1.2 CONCEPTS OF A RELATIONAL MODEL

The relational model was propounded by E.F. Codd of the IBM in 1972. The basic concept in the relational model is that of a relation

A relation can be viewed as a table which has the following properties :

Property 1: it is column homogeneous. In other words, in any given column of a table, all items are of the same kind.

Property 2: each item is a simple number or a character string. That is, a table must be in 1NF. (First Normal Form) which will be introduced in the second unit.

Property 3: all rows of a table are distinct.

Property 4: the ordering of rows within a table is immaterial.

Property 5: the columns of a table are assigned distinct names and the ordering of these columns is immaterial.

Example of a valid relation

S#	P#	SCITY
10	1	BANGALORE
10	2	BANGALORE
11	1	BANGALORE
11	2	BANGALORE

1.3 FORMAL DEFINITION OF A RELATION

Formally, a relation is defined as the subset of the expanded cartesian product of domains. In order to do so, first we define the cartesian product of two sets and then the expanded cartesian product.

The cartesian product of two sets A and B, denoted by $A \times B$ is

$$A \times B = \{(a,b) : a \in A \text{ and } b \in B\}$$

The expanded cartesian product of n sets $A_1, A_2, \ldots, A_n$ is defined by

$$X(A_1, A_2, \ldots, A_n) = \{(a_1, a_2, \ldots a_n) : a_j \in Aj \quad 1 <= j < = n)\}$$

The element $(a_1, a_2, \ldots, a_n)$ is called an n-tuple.

Given domains $D_1, D_2, \ldots D_n$ we define a relation, R, as a subset of the expanded cartesian product of these domains as follows:

$$R(D_1, D_2, \ldots D_n) \leq X(D_1, D_2, \ldots D_n)$$

In general we say that a relation defined over n domains has a degree n or is n-ary. The elements of this set are n-tuples.

We shall distinguish between the definition of a relation and the relation itself. We shall say that the definition of a relation gives a name to the relation and specifies the components over which it is defined. These components are referred to as relation attributes or attributes for short. An attribute has a domain associated with it from which it takes on values. The relation itself, on the other hand, is the set of tuples which constitute it at a given instance of time. For example, a statement which says that a relation Supplier is built over attributes S#, P#, SCITY having domains integer, character string respectively is the definition of the relation Supplier. The relation itself is shown below. It must be noted that at the time the definition of a relation is just given, a relation with no tuples in it, i.e. a null relation, is created.

Supplier

S#	P#	SCITY
10	1	BANGALORE
10	2	BANGALORE
10	3	BANGALORE
11	1	BOMBAY
11	2	BOMBAY

A **relational schema** is defined to be a collection of relation definitions.

We can now define the notion of a relational database or database for short. A database is a collection of relations of assorted degrees such that these relations are in accordance with their definitions in the relational schema. Since a relation is time varying, by this definition we can infer that a database is also time varying.

1.4 THE CODD COMMANDMENTS

In the most basic of definitions a DBMS can be regarded as relational only if it obeys the following three rules: -

- All information must be held in tables
- Retrieval of the data must be possible using the following types of operations:

  SELECT, JOIN and PROJECT
- All relationships between data must be represented explicitly in that data itself.

This really is the minimum requirement, but it is surprising to see just how some well-known database products fail according to these simple rules to be in fact, relational, no matter what their vendors claim.

To define the requirements more rigorously, compliance with the 12 rules stated below must be demonstrable, within a single product, for it to be termed relational. In reality it's true to say that they don't all carry the same degree of importance, and indeed some very good products exist today supporting major large-scale production systems that cannot, hand on heart, claim to obey any more than eigl.. or so of these rules. It's likely however, that it is only when all 12 rules can be satisfied, by facilities that coexist together, that the full benefits of the relational database can be realised.

The Twelve Rules

Just as in the 12 rules that define the distributed product, there is a single overall rule which in some ways covers all others and is commonly called Rule 0. It states that:

Any truly relational database must be manageable entirely through its own relational capabilities

Having stated this rule, we will not delve deeper except to say that its meaning can be interpreted by stating that a relational database must be **relational, wholly relational and nothing but relational.** If a DBMS depends on record-by-record data manipulation tools, it is not truly relational.

Rule 1: The information rule

All information is explicitly and logically represented in exactly one way — by data values in tables.

In simple terms this means that if an item of data doesn't reside somewhere in a table in the database then it doesn't exist and this should be extended to the point where even such information as table, view and column names to mention just a few, should be contained somewhere in table form. This necessitates the provision of an active data dictionary, that is itself relational, and it is the provision of such facilities that allow the relatively easy additions to RDBMS's of programming and CASE tools for example. This rule serves on its own to invalidate the claims of several databases to be relational simply because of their lack of ability to store dictionary items (or indeed **metadata**) in an integrated, relational form. Commonly such products implement their dictionary information systems in some native file structure, and thus set themselves up for failing at the first hurdle.

Rule 2 : The rule of guaranteed access

Every item of data must be logically addressable by resorting to a combination of table name, primary key value and column name.

Whilst it is possible to retrieve individual items of data in many different ways, especially in a relational/SQL environment, it must be true that any item can be retrieved by supplying the table name, the primary key value of the row holding the item and the column name in which it is to be found. If you think back to the table like storage structure, this rule is saying that at the intersection of a column and a row you will necessarily find one value of a data item (or null).

Rule 3 : The systematic treatment of null values

It may surprise you to see this subject on the list of properties, but it is fundamental to the DBMS that null values are supported in the representation of missing and inapplicable information. This support for null values must be consistent throughout the DBMS, and independent of data type (a null value in a CHAR field must mean the same as ull in an INTEGER field for example).

It has often been the case in other porduct types, that a character to represent missing or inapplicable data has been allocated from the domain of characters pertinent to a particular

item. We may for example define four permissible values for a column SEX as:

M	Male
F	Female
X	No data available
Y	Not applicable

Such a solution requires careful design, and must decrease productivity at the very least. This situation is particularly undesirable when very high-level languages such as SQL are used to manipulate such data, and if such a solution is used for numeric columns all sorts of problems can arise during aggregate functions such as SUM and AVERAGE etc.

Rule 4 : The database description rule

A description of the database is held and maintained using the same logical structures used to define the data, thus allowing users with appropriate authority to query such information in the same ways and using the same languages as they would any other data in the database.

Put into easy terms, Rule 4 means that there must be a data dictionary within the RDBMS that is constructed of tables and/or views that can be examined using SQL. This rule states therefore that a dictionary is mandatory, and if taken in conjunction with Rule 1, there can be no doubt that the dictionary must also consist of combinations of tables and views.

Rule 5 : The comprehensive sub-language rule

There must be at least one language whose statements can be expressed as character strings conforming to some well defined syntax, that is comprehensive in supporting the following :

- Data definition
- View definition
- Data manipulation
- Integrity constraints
- Authorisation
- Transaction boundaries

Again in real terms, this means that the RDBMS must be completely manageable through its own dialect of SQL, although some products still support SQL-like languages (Ingress support of Quel for example). This rule also sets out to scope the functionality of SQL – you will detect an implicit requirement to support access control, integrity constraints and transaction management facilities for example.

Rule 6 : The view updating rule

All views that can be updated in theory, can also be updated by the system. This is quite a difficult rule to interpret, and so a word of explanation is required whilst it is possible to create views in all sorts of illogical ways, and with all sorts of aggregates and virtual columns, it is obviously not possible to update through some of them. As a very simple example, if you define a **virtual column** in a view as A*B where A and B are columns in a base table, then how can you perform an update on that virtual column directly? The database cannot possible break down any number supplied, into its two component parts, without more information being supplied. To delve a little deeper, we should consider that the possible complexity of a view is almost infinite in logical terms, simply because a view can be defined in terms of both tables and other views. Particular vendors restrict the complexity of their own implementations, in some cases quite drastically.

Even in logical terms it is often incredibly difficult to tell whether a view is theoretically updatable, let alone delve into the practicalities of actually doing so. In fact there exists another set of rules that, when applied to a view, can be used to determine its level of logical complexity, and it is only realistic to apply Rule 6 to those views that are defined as **simple** by such criteria.

Rule 7 : The insert and update rule

An RDBMS must do more than just be able to retrieve relational data sets. It has to be

capable of inserting, updating and deleting data as a relational set. Many RDBMSes that fail the grade fall back to a single-record-at-time procedural technique when it comes time to manipulate data.

Rule 8 : The physical independence rule

User access to the database, via monitors or application programs, must remain logically consistent whenever changes to the storage representation, or access methods to the data, are changed.

Therefore, and by way of an example, if an index is built or destroyed by the DBA on a table, any user should still retrieve the same data from that table, albeit a little more slowly. It is largely this rule that demands the clear distinction between the logical and physical layers of the database. Applications must be limited to interfacing with the logical layer to enable the enforcement of this rule, and it is this rule that sorts out the men from the boys in the relational market place. Looking at other architectures already discussed, one can imagine the consequences of changing the physical structure of a network or hierarchical system.

However there are plenty of traps awaiting even in the relational world. Consider the application designer who depends on the presence of a B-tree type index to ensure retrieval of data is in a predefined order, only to find that the DBA dynamically drops the index! What about the programmer who doesn't check for prime key uniqueness in his application, because he knows it is enforced by a unique index. The removal of such an index might be catastrophic. I point out these two issues because although they are serious factors, I am not convinced that they constitute the breaking of this rule; it is for the individual to make up his own mind.

Rule 9 : The logical data independence rule

Application programs must be independent of changes made to the base tables.

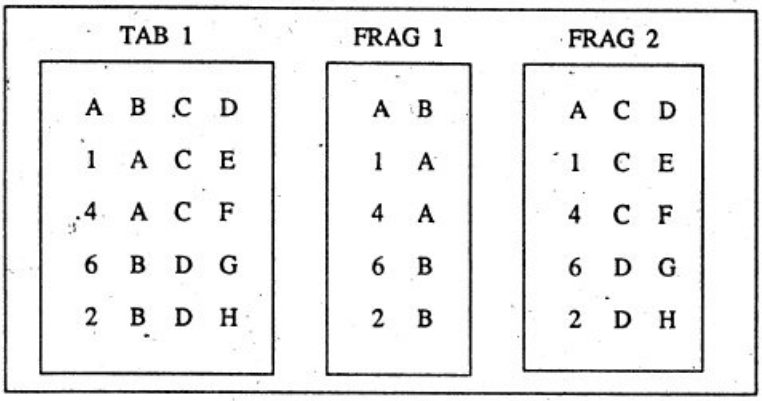


Figure 1 : TAB 1 Split into two fragments

This rule allows many types of database design change to be made dynamically, without users being aware of them. To illustrate the meaning of the rule the examples on the next page show two types of activity, described in more detail later, that should be possible if this rule is enforced.

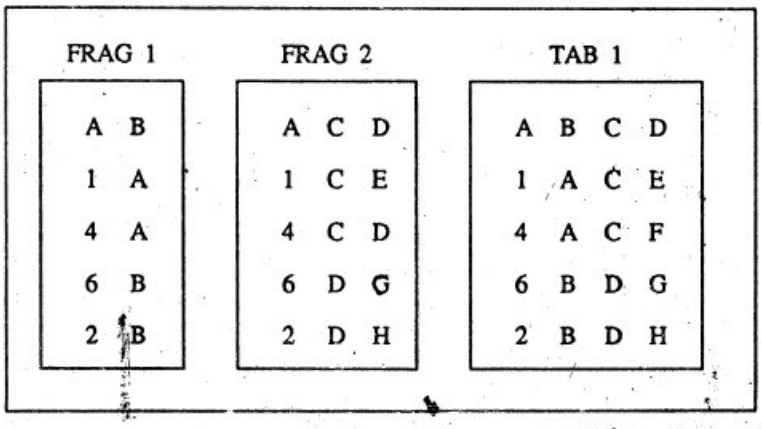


Figure 2 : Two fragments Combined into One Tab

Firstly, it should be possible to split a table vertically into more than one **fragment**, as long as such splitting preserves all the original data (is **non-loss**), and maintain the primary key in each and every fragment. This means in simple terms that a single table should be divisible into one or more other tables.

Secondly it should be possible to combine base tables into one by way of a **non-loss join.**

Note that if such changes are made, then views will be required so that users and applications are unaffected by them.

Rule 10 : Integrity rules

The relational model includes two general integrity rules. These integrity rules implicitly or explicitly define the set of consistent database states, or changes of state, or both. Other integrity constraints can be specified, for example, in terms of dependencies during database design. In this section we define the integrity rules formulated by Codd.

Integrity Rule 1

Integrity rule 1 is concerned with primary key values. Before we formally state the rule, let us look at the effect of null values in prime attributes. A null value for an attribute is a value that is either not known at the time or does not apply to a given instance of the object. It may also be possible that a particular tuple does not have a value for an attribute; this fact could be represented by a null value.

If any attribute of a primary key (prime attribute) were permitted to have null values, then, because the attributes in the key must be nonredundant, the key cannot be used for unique identification of tuples. This contradicts the requirements for a primary key. Consider the relation P in figure 3. The attribute Id is the primary key for P. If null values (represented as @ were permitted, as in figure 3, then the two tuples @, Smith are indistinguishable, even though they may represent two different instances of the entity type employee. Similarly, the tuples < @, Lalonde > and 10^4, Lalonde >, for all intents and purposes, are also indistinguishable and may be referring to the same person. As instances of entities are distinguishable, so must be their surrogates in the model.

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
107	Evan
110	Drew
112	Smith

(a)

P:

Id	Name
101	Jones
@	Smith
104	Lalonde
107	Evan
110	Drew
@	Lalonde
@	Smith

(b)

Figure 3 : (a) Relation without null values and (b) relation with null values

Integrity rule 1 specifies that instances of the entities are distinguishable and thus no prime attribute (component of a primary key) value may be null. This rule is also referred to as the entity rule. We could state this rule formally as:

Definition: Integrity Rule 1 (Entity Integrity):

If the attribute A of relation R is a prime attribute of R, then A cannot accept null values.

Integrity Rule 2 (Referential Integrity) :

Integrity rule 2 is concerned with foreign keys, i.e., with attributes of a relation having domains that are those of the primary key of another relation.

Relation (R), may contain references to another relation (S). Relations R and S need not be distinct. Suppose the reference in r is via a set of attributes that forms a primary key of the relation S. This set of attributes in R is a foreign key. A valid relationship between a tuple in R to one in S requires that the values of the attributes in the foreign key of R correspond to the primary key of a tuple in S. This ensures that the reference from a tuple of the relation R

is made unambiguously to an existing tuple in the S relation. The referencing attribute(s) in the R relation can have null value(s); in this case, it is not referencing any tuple in the S relation. However, if the value is not null, it must exist as the primary attribute of a tuple of the S relation. If the referencing attribute in R has a value that is nonexistent in S, R is attempting to refer a nonexistent tuple and hence a nonexistent instance of the corresponding entity. This cannot be allowed. We illustrate this point in the following example:

Example

Consider the example of employees and their has a manager and as managers are also employees, we may represent managers by their employee numbers, if the employee number is a key of the relation employee. Figure 4 illustrates an example of such an employee relation. The Manager attribute represents the employee number of the manager. Manager is a foreign key; note that it is referring to the primary key of the same relation. An employee can only have a manager who is also an employee. The chief executive officer (CEO) of the company can have himself or herself as the manager or may take null values. Some employees may also be temporarily without manager, and this can be represented by the Manager taking null values.

Emp#	Name	Manager
101	Jones	@
103	Smith	110
104	Lalonde	107
107	Evan	110
110	Drew	112
112	Smith	112

Figure 4 : Foreign Keys

Definition : Integrity Rule 2 (Referential Integrity)

Given two relations R and S, suppose R refers to the relation S via a set of attributes that forms the primary key of S and this set of attributes forms a foreign key in R. Then the value of the foreign key in a tuple in R must either be equal to the primary key of a tuple of S or be entirely null.

If we have the attribute A of relation R defined on domain D and the primary key of relation S also defined on domain D, then the values of A in tuples of R must be either null or equal to the value, let us say v, where v is the primary key value for a tuple in S. Note that R and S may be the same relation. The tuple in S is called the target of the foreign key. The primary key of the referenced relation and the attributes in the foreign key of the referencing relation could be composite.

Referential integrity is very important. Because the foreign key is used as a surrogate for another entity, the rule enforces the existence of a tuple for the relation corresponding to the instance of the referred entity. In example, we do not want a nonexisting employee to be manager. The integrity rule also implicitly defines the possible actions that could be taken whenever updates, insertions, and deletions are made.

If we delete a tuple that is a target of a foreign key reference, then three explicit possibilities exist to maintain database integrity:

- All tuples that contain references to the deleted tuple should also be deleted. This may cause, in turn, the deletion of other tuples. This option is referred to as a **domino or cascading deletion**, since one deletion leads to another.
- Only tuples that are not referenced by any other tuple can be deleted. A tuple referred by other tuples in the database cannot be deleted.
- The tuple is deleted. However, to avoid the domino effect, the pertinent foreign key attributes of all referencing tuples are set to null.

Similar actions are required when the primary key of a referenced relation is updated. An update of a primary key can be considered as a deletion followed by an insertion.

The choice of the option to use during a tuple deletion depends on the application. For example, in most cases it would be inappropriate to delete all employees under a given manager on the manager's departure; it would be more appropriate to replace it by null

Another example is when a department is closed. If employees were assigned to departments, then the employee tuples would contain the department key too. Deletion of a department tuples should be disallowed until the employees have either been reassigned or their appropriate attribute values have been set to null. The insertion of a tuple with a foreign key reference or the update of the foreign key attributes of a relation require a check that the referenced relation exists.

Although the definition of the relational model specifies the two integrity rules, it is unfortunate that these concepts are not fully implemented in all commercial relational DBMSs. The concept of referential integrity enforcement would require an explicit statement as to what should be done when the primary key of a target tuple is updated or the target tuple is deleted.

Rule 11 : Distribution rule :

A RDBMS must have distribution independence.

This is one of the more attractive aspects of RDBMSes.Database system built on the relational framework are well suited to today's client/server database design.

Rule 12 : No subversion rule :

If an RDBMS supports a lower level language that permits for example, row-at-a-time processing, then this language must not be able to bypass any integrity rules or constraints of the relational language.

Thus, not only must a RDBMS be governed by relational rules, but those rules must be its primary laws.

The practical importance of these rules is difficult to estimate, and depends largely on the RDBMS in question, its proposed use and individual view points, but the theoretical importance is undeniable. It is interesting to see how some of the rules relate to others, and to some of the more important advantages of the relational model. It is unlikely at the present time that any RDBMS can claim full logical data independence because of their generally poor ability to handle updating through views. Even token adherence to this rule however, when combined with facilities enabling physical data independence, potentially yield advantages to applications developers, unheard of with any other type of database system. Coupling these two rules with the data independence and distribution independence rules can take the protection of customer investment to new heights.

The beauty of the relational database is that the concepts that define it are few, easy to understand and explicit. The 12 rules explained can be used as the basic relational design criteria, and as such are clear indications of the purity of the relational concept. Whilst you do not find these rules being quoted so often these days as in the recent past, it does not mean that they are any less important. Rather it can be interpreted as reflecting a reduced importance as propaganda. Other factors, of which performance is the most obvious, have now taken precedence.

1.5 RELATIONAL ALGEBRA

Relational algebra is a procedural language. It specifies the operations to be performed on existing relations to derive result relations. Furthermore, it defines the complete scheme for each of the result relations. The relational algebraic operations can be divided into basic set-oriented operations and relational- oriented operations. The former are the traditional set operations, the latter, those for performing joins, selection, projection, and division.

Basic Operations

Basic operations are the traditional set operations: union, difference, intersection and cartesian product. Three of these four basic operations – union, intersection, and difference – require that operand relations be **union compatible.** Two relations are union compatible if they have the same arity and one-to-one correspondence of the attributes with the corresponding attributes defined over the same domain. The cartesian product can be defined on any two relations. Two relations P and Q are said to be union compatible if both P and Q are of the same degree n and the domain of the corresponding n attributes are identical, i.e. if $P = P\{P_1, ... _n\}$ and $Q = \{Q_1, ... Q_n\}$ then

$$Dom(P_i) = Dom(Q_i) \text{ for } i = \{1,2, ...n\}$$

where $Dom(P_i)$ represents the domain of the attribute P_i.

Example 1

In the examples to follow, we utilise two relations P and Q given in Figure 5. R is a computed result relation. We assume that the relations P and Q in Figure 5 represent employees working on the development of software application packages J_1 and J_2 respectively.

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
107	Even
110	Drew
112	Smith

Q:

Id	Name
103	Smith
104	Lalonde
106	Byron
110	Drew

Figure 5 : Union compatible relations

If we assume that P and Q are two union-compatible relations, then the union of P and Q is the set-theoretic union of P and Q. The resultant relation, $R = P \cup Q$, has tuples drawn from P and Q such that

$$R = \{t \mid \in P \vee t \in Q\}$$

The result relation R contains tuples that are in either P or Q or in both of them. The duplicate tuples are eliminated.

Remember that from our definition of union compatibility the degree of the relations P and R is the same. The cardinality of the resultant relation depends on the duplication of tuples in P and Q. From the above expression, we can see that if all the tuples in Q were contained in P, then $|R| = |P|$ and $R = P$, while if the tuples in P and Q were disjoint, then $|R| = |P| + |Q|$.

Example 2

R, the union of P and Q given in Figure 5 in the above example 1 is shown in Figure 6(a). R represents employees working on the packages J_1 or J_2, or both of these packages. Since a relation does not have duplicate tuples, an employee working on both J_1 and J_2 will appear in the relation R only once.

R:

Id	Name
101	Jones
103	Smith
104	Lalonde
107	Even
110	Drew
112	Smith

(a) $P \cup Q$

R:

Id	Name
101	Jones
107	Evan
112	Smith

(b) $P - Q$

R:

Id	Name
103	Smith
104	Lalonde
110	Draw

(c) $P \cap Q$

Figure 6 : Results of (a) union (b) difference and (c) intersection operations

Difference (–)

The difference operation removes common tuples from the first relation.

$R = P - Q$ such that

$R = \{t \mid t \in P \wedge t \notin Q\}$

Example 3

R, the result of $P - Q$, gives employees working only on package J_1. (figure 6(b) in example 2). Employees working on both packages J_1 and J_2 have been removed.

Intersection ($\cap$)

The intersection operation selects the common tuples from the two relations.

$$R = P \cap Q \text{ where}$$

$$R = \{t \mid t \in P \wedge t \in Q\}$$

Example 4

The resultant relation of $P \cap Q$ is the set of all employees working on both the packages. (figure 5(c) of example 2).

The intersection operation is really unnecessary. It can be very simply expressed as:

$$P \cap Q = P - (P - Q)$$

It is, however, more convenient to write an expression with a single intersection operation than one involving a pair of difference operations.

Note that in these examples the operand and the result relation schemes, including the attribute names, are identical i.e. $P = Q = R$. If the attribute names of compatible relations are not identical, the naming of the attributes of the result relation will have to be resolved.

Cartesian Product ($\times$)

The extended cartesian or simply the cartesian product of two relations is the concatenation of tuples belonging to the two relations. A new resultant relation scheme is created consisting of all possible combinations of the tuples.

$$R = P \times Q$$

where a tuple $r \in R$ is given by $\{t_1 \,||\, t_2 \mid t_1 \in P \wedge t_2 \in Q\}$, i.e. the result relation is obtained by conatenating each tuple in relation P with each tuple in relation Q. Here, represents the concation operation.

The scheme of the result relation is given by.

$$R = P \,||\, Q$$

The degree of the result relation is given by:

$$|R| = |P| + |Q|$$

The cardinality of the result relation is given by:

$$|R| = |P| * |Q|$$

Example 5

The cartesian product of the PERSONNEL relation and SOFTWARE_PACKAGE relations of figure 7(a) is shown in figure 7(b). Note that the relations P and Q from figure 5 of example 1 are a subset of the PERSONNEL relation.

PERSONNEL :

Id	Name
101	Jones
103	Smith
104	Lalonde
106	Byron
107	Evan
110	Drew
112	Smith

Software Packages :

S
J_1
J_2

7(a)

PERSONNEL :

Id	P.Name	S
101	Jones	J_1
101	Jones	J_2
103	Smith	J_1
103	Smith	J_2
104	Lalonde	J_1
104	Lalonde	J_2
106	Byron	J_1
106	Byron	J_2
107	Evan	J_1
107	Evan	J_2
110	Drew	J_1
110	Drew	J_2
112	Smith	J_1
112	Smith	J_2

(b)

Figure 7 : (a) PERSONNEL (Emp#, Name) and SOFTWARE_PACKAGE(S) represent employees and software packages respectively; (b) the Cartesian product of PERSONNEL and SOFTWARE_PACKAGES

The union and intersection operations are associative and commutative; therefore, given relations R, S, T:

$$R \cup (S \cup T) = (R \cup S) \cup T = (S \cup R) \cup T = T \cup (S \cup R) = \ldots R \cap (S \cap T) = (R \cap S) \cap T = \ldots$$

The difference operation, in general, is noncommutative and nonassociative.

$$R - S \neq S - R \qquad \text{noncommutative}$$

$$R - (S - T) \neq (R - S) - T \qquad \text{nonassociative}$$

Additional Relational Algebraic Operations

The basic set operations, which provide a very limited data manipulation facility, have been supplemented by the definition of the following operations: projection, selection, join, and division. These operations are represented by symbols π, σ, ∞ and + respectively. Projection and selection are unary operations; join and division are binary.

Projection (π)

The projection of a relation is defined as a projection of all its tuples over some set of attributes, i.e., it yields a **vertical subset** of the relation. The projection operation is used to either reduce the number of attributes in the resultant relation or to reorder attributes. In the first case, the arity (or degree) of the relation is reduced. The projection operation is shown graphically in figure 8. Figure 8 shows the projection of the relation PERSONNEL on the attribute Name. The cardinality of the result relation is also reduced due to the deletion of duplicate tuples.

We defined the projection of a tuple t_i over the attribute A, denoted $t_i[A]$ or $\pi_A(t_i)$, as (a), where a is the value

PERSONAL :

Id	Name
101	Jones
103	Smith
104	Lalonde
106	Byron
107	Evan
110	Drew
112	Smith

→

Name
Jones
Smith
Lalonde
Byron
Evan
Drew

Figure 8 : Projection of relation PERSONNEL over attribute Name

of tuple t_i over the attribute A. Similarly, we define the projection of a relation T, denoted by T[A] or $\pi_A(T)$, on the attribute A. This is defined in terms of the projection for each tuple in t_i belonging to T on the attribute A as:

$$T[A] = \{a_i \mid t_i[A] = a_i \wedge t_i \in T\}$$

where T[A] is a single attribute relation and $|T[A]| \leq T$. The cardinality T[A] may be less than the cardinality |T| because of the deletion of any duplicates in the result. A case in point is illustrated in figure 8.

Similarly, we can define the projection of a relation on a set of attribute names, X, as a concatenation of the projections for each attribute A in X for every tuple in the relation.

$$T[X] = \{ t_i[A] \mid t_i \in T\}$$

A belongs to X

where $t_i[A]$ represents the concatenation of all $t_i[A]$ for all $A \in X$.

A belongs to X

Simply stated, the projection of a relation P on the set of attribute names Y belong to P is the projection of each tuple of the relation P on the set of attribute names Y.

Note that the projection operation reduces the arity if the number of attributes in X is less than the arity of the relation. The projection operation may also reduce the cardinality of the result relation since duplicate tuples are removed. (Note that the projection operation produces a relation as the result. By definition, a relation cannot have duplicate tuples. In most commercial implementations of the relational model, however, the duplicates would still be present in the result).

Selection (σ)

Suppose we want to find those employees in the relation PERSONNEL of figure 7(a) of example 5 with an Id less than 105. This is an operation that selects only some of the tuples the relation. Such an operation is known as a **selection operation.** The projection operation yields a vertical subset of a relation. The action is defined over a subset of the attribute names but over all the tuples in the relation. The selection operation, however, yields a horizontal subset of a given relation, i.e., the action is defined over the complete set of attribute names but only a subset of the tuples are included in the result. To have a tuple included in the result relation, the specified selection conditions or predicates must be satisfied by it. The selection operation, is sometimes known as the **restriction operation.**

PERSONNEL :

Id	Name
101	Jones
103	Smith
104	Lalonde
106	Byron
107	Evan
110	Drew
112	Smith

Results of Selection

Id	Name
101	Jones
103	Smith
114	Lalonde

Figure 9 : Result of Selection over PERSONNEL for Id < 105.

Any finite number of predicates connected by Boolean operators may be specified in the selection operation. The predicates may define a comparison between two domain-compatible attributes or between an attribute and a constant value; if the comparison is between attribute A_1 and constant c_1, then c_1 belong to $Dom(A_1)$.

Given a relation P and a predicate expression B, the selections of those tuples of relation P that satisfy the predicate B is a relation R written as:

$$R = \sigma_B(P)$$

The above expression could be read as "select those tuples t from P in which the predicate B(t) is true." The set of tuples in relation R are in this case defined as follows:

$R = \{t \mid t \in to P \wedge B(t)\}$

JOIN (⋈)

The join operator, as the name suggests, allows the combining of two relations to form a single new relation. The tuples from the operand relations that participate in the operation and contribute to the result are related. The join operation allows the processing of relationships existing between the operand relations.

Example 6

In figure10we encounter the following relations: ASSIGNMENT (**Emp#, Prod#, Job#**)

JOB_FUNCTION (**Job#,** Title)

EMPLOYEE :

Emp#	Name	Profession
101	Jones	Analyst
103	Smith	Programmer
104	Lalonde	Receptionist
106	Byron	Receptionist
107	Evan	VPR & D
110	Drew	VP operations
112	Smith	Manager

PRODUCT :

Prod#	Prod-Name	Prod-Details
HEAP1	HEAP-SORT	ISS Module
BINS9	BINARY-SEARCH	ISS/R Module
FM6	FILE-MANAGER	ISS/R-PC Subsys
B++1	B++_TREE	ISS/R Turbo Sys
B++2	B++_TREE	ISS/R-PC Turbo

(a)

JOB-FUNCTION

Job#	Title
1000	CEO
900	President
800	Manager
700	Chief Programmer
600	Analyst

ASSIGNMENT

Emp#	Prod#	Job#
107	HEAP 1	800
101	HEAP 1	600
110	BINS9	800
103	HEAP1	700
101	BINS9	700
110	FM6	800
107	B++1	800

(b)

Figure 10 (a) Relation schemes for employee role in development teams (b) Sample relations

Suppose we want to respond to the query **Get product number of assignments whose development teams have a chief programmer.** This requires first computing the cartesian product of the ASSIGNMENT and JOB_FUNCTION relations. Let us name this product relation TEMP. This is followed by selecting those tuples of TEMP where the attribute Title has the value chief programmer and the value of the attribute Job# in ASSIGNMENT and JOB_FUNCTION are the same. The required result, shown below is obtained by projecting these tuples on the attribute Prod#. The operations are specified below.

TEMP = (ASSIGNMENT X JOB_FUNCTION)

$\pi_{Prod\#}$ (σ Title = 'chief programmer' $\wedge$ ASSIGNMENT.Job# (TEMP))

Prod #
HEAP 1
BINS 9

In another method of responding to this query, we can first select those tuples from the JOB FUNCTION relation so that the value of the attribute Title is chief programmer. Let us call this set of tuples the relation TEMP1. We then compute the cartesian product of TEMP1 and ASSIGNMENT, calling the product TEMP2. This is followed by a projection on Prod# over TEMP2 to give us the required response. These operations are specified below:

TEMP1 = (σ Title = 'chief programmer'(JOB_FUNCTION))

TEMP2 = (σ ASSIGNMENT.Job# = JOB_FUNCTION.Job# (ASSIGNMENT X TEMP1))

$\pi_{Prod\#}$ (TEMP2) gives the required result.

Notice that in the selection operation that follows the cartesian product we take only those tuples where the value of the attributes ASSIGNMENT.Job# and JOB_FUNCTION.Job# are the same. These combined operations of cartesian product followed by selection are the join operation. Note that we have qualified the identically named attributes by the name of the corresponding relation to distinguish them.

In case of the join of a relation with itself, we would need to rename either the attributes of one of the copies of the relation or the relation name itself. We illustrate this in example 7.

In general the join condition may have more than one term, necessitating the use of the subscript in the comparison operator. Now we shall define the different types of join operations.

In these discussions we use P, Q, R and so on to represent both the relation scheme and the collection or bag of underlying domains of the attributes. We call it a bag of domains because more than one attribute may be defined on the same domain.

Typically, $P \cap Q$ may be null and this guarantees the uniqueness of attribute names in the result relation. When the same attribute name occurs in the two schemes we use qualified names.

Two common and very useful variants of the join are the **equi-join** and the **natural join**. In the equi-join the comparison operator $theta_i (i = 1,2,....., n)$ is always the equality operator (=). Similarly, in the natural join the comparison operator is always the equality operator. However, only one of the two sets of domain compatible attributes involved in the natural join are A_i from P and B_i from Q, for i = 1, ..., n, the natural join predicate is a conjunction of terms of the following form:

$$(t_1[A_i] = t_2[B_i]) \text{ for } i = 1,2, ..., n$$

Domain compatibility requires that the domains of A_i and B_i be compatible, and for this reason relation schemes P and Q have attributes defined on common domains, i.e., $P \cap Q \neq \phi$. Therefore, join attributes have common domains in the relation schemes P and Q. Consequently, only one set of the join attributes on these common domains needs to be preserved in the result relation. This is achieved by taking a projection after the join operation, thereby eliminating the duplicate attributes. If the relation P and Q have attributes with the same domains but different attribute names, then renaming or projection may be specified.

Example 7

Given the EMPLOYEE and SALARY relations of figure 11(i), if we have to find the salary of employees by name, we join the tuples in the relation EMPLOYEE with those in SALARY such that the value of the attribute Id in EMPLOYEE is the same as that in SALARY. The natural join takes the predicate expression to be EMPLOYEE.Id = SALARY.Id. The result of the natural join is shown in figure 11(ii). When using the natural join, we do not need to specify this predicate. The expression to specify the operation of finding the salary of employees by name is given as follows. Here we project the result of the natural join operation on the attributes Name and Salary:

π (Name.Salary) (EMPLOYEE ⋈ SALARY)

EMPLOYEE :

Id	Name
101	Jones
103	Smith
104	Lalonde
107	Evan

SALARY :

Id	Salary
101	67
103	55
104	75
107	80

EMPLOYEE ⋈ SALARY

Id	Name	Salary
01	Jones	67
03	Smith	55
04	Lalonde	75
07	Evan	80

ASSIGNMENT.Emp#	COASSIGN.Emp #
107	107
107	101
107	103
101	107
101	101
101	103
110	110
110	101
103	107
103	101
103	103
101	110

Figure 11 : (i) The natural join of EMPLOYEE and SALARY relations; (ii) The joint of ASSIGNMENT with the renamed copy

Division (÷)

Before we define the division operation, let us consider an example.

Example 8

Given the relations P and Q as shown in figure 12 (a), the result of dividing P by Q is the relation R and it has two tuples. For each tuple in R, its product with the tuples of Q must be in P. In our example (a_1,b_1) and (a_1,b_2) must both be tuples in P; the same is true for (a_5, b_1) and (a_5,b_2).

Simply stated, the cartesian product of Q and R is a subset of P. In figure 12 (b), the result relation R has four tuples; the cartesian product of R and Q gives a resulting relation which is

P :

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

Q :

B
b_1
b_2

R (result) :

A
a_1
a_5

(a)

Q :

B
b_1

then R is :

A
a_1
a_2
a_3
a_5

(b)

Q :

B
b_1
b_2
b_3

then R is

A

Q :

B

then R is :

A
a_1
a_2
a_3
a_4
a_5

(d)

Figure 12 : Examples of the division operation. (a) R = P ÷ Q; (b) R = P ÷ Q (P is the same as in part i); (c) R = P ÷ Q (P is the same as in part i); (d) R = P ÷ Q (P is the same as in part i)

again a subset of P. In figure12 (c), since there are no tuples in P with a value b_3 for the attribute B (i.e., $selection_{B = b3}(P) = 0$), we have an empty relation R, which has a cardinality of zero.

In figure 12(d), the relation Q is empty. The result relation can be defined as the projection of P on the attributes in P – Q. However, it is usual to disallow division by a empty relation.

Finally, if relation P is an empty relation, then relation R is also an empty relation.

Let us treat the Q as representing one set of properties (the properties are defined on the Q, each tuple in Q representing an instance of these properties) and the relation r as representing entities with these properties (entities are defined on P – Q, and the properties are, as before, defined on Q); note that $P \cup Q$ must be equal to P. Each tuple in P represents an object with some given property. The resultant relation R, then, is the set of entities that possesses all the properties specified in Q. The two entities a_1 and a_5 possess all the properties, i.e., b_1 and b_2. The other entities in P, a_2, a_3, and a_4, only possess one, not both, of the properties. The division operation is useful when a query involves the phrase "for all objects having all the specified properties." Note that both P – Q and Q in general represent a set of attributes. It should be clear that Q not a subset of P.

1.6 RELATIONAL COMPLETENESS

The notion of relational completeness was propounded by Codd in 1972 as a basis for evaluating the power of different query languages.

A language is relationally complete if the basic relational algebra operations can be performed. The basic relational algebra operations are

- Union
- Difference
- Gross product
- Projection
- Selection

Query languages that are actually used in practice provide features in addition to the one mentioned above. For example, they provide facilities for

1. modification, storage and deletion of information
2. printing relations
3. assigning relations to some relation names
4. computing aggregate functions like SUM and MAX
5. performing arithmetic, for example, like retrieving the Salary + commission.

Check Your Progress

1. Define the following terms:
   (a) Intention of a set
   (b) Extension of a set
2. What is union compatible?

.....

.....

.....

.....

1.7 SUMMARY

The relational model has evoked a wide amount of interest in the database community. This model has a very sound mathematical basis to it. It exhibits a high degree of data independence.

However, it has its share of difficulties. These are:

- The relational model does not deal with issues like **semantic integrity, concurrency** and **database security.** These issues are left to be solved by the implementors of database management systems based on the relational model. The most serious consequence of the foregoing was the absence of the concept of semantic integrity in relational systems.
- Traditionally, implementations of the relational model have suffered from the drawback that they are relatively poor on response time. The biggest problem is in the realization of the **join operator.** Whereas, a DBMS based on the relational model can handle small databases, as the sizes of databases reach the region of billions of bytes the performance of these systems falls rather drastically. Consequently, these systems are able to support databases of relatively small sizes.

1.8 MODEL ANSWERS

1. (a) The **intention** of a set defines the permissible occurrences by specifying a membership condition.

   (b) The **extension** of the set specifies one of numerous possible occurrences by explicitly listing the set members. These two methods of defining a set are illustrated by the following example:

   Intention of set G = {g | g is an odd positive integer less than 10}

   Extension of set G = {1,3,5,7,9}

2. Two relations are union compatible if they have some arity and one to one correspondence of the attributes with the corresponding attributes defined over the same domain.

1.9 FURTHER READING

Bipin C.Desai, *An Introduction to Database System*, Galgotia Publication, New Delhi.

UNIT 2 NORMALIZATION

Structure

2.0 INTRODUCTION

The basic objectives of normalization is to reduce redundancy which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Relations are normalized so that when relations in database are to be altered during the life time of the atabase, we do not lose information or introduce inconsistencies. The type of alterations normally needed for relations are

1. **Insertion** of new data values to a relation. This should be possible without being forced to leave blank fields for some attributes.
2. **Deletion** of a tuple, namely, a row of a relation. **This should be possible without losing** vital information unknowingly.
3. **exhaustively searching all the tuples in the relation**

In this unit, in the beginning we discuss the importance of having a consistent database **Without repetition of data and points out the anomalies that could be introduced in a database with an undesirable scheme. Then we discuss the different forms of normalization**

2.1 OBJECTIVES

After going through this unit, you may be able to:

- discuss the different types of anomalies in a database
- state what is functional dependency
- list the different forms of normalization
- differentiate among different types of normalization.

2.2 FUNCTIONAL DEPENDENCY

As the concept of dependency is very important, it is essential that we first understand it well and then proceed to the idea of normalization. There is no fool-proof algorithmic method of identifying dependency. We have to use our commonsense and judgement of specify dependencies.

Let X and Y be two attributes of a relation. Given the value of X, if there is only one value of Y corresponding to it, then Y is said to be functionally dependent on X. This is indicated by the notation:

$$X \rightarrow Y$$

For example, given the value of item code, there is only one value of item name for it. Thus item name is functionally dependent on item code. This is shows as:

Item code → item name

Similarly in table 1, given an order number, the date of the order is known. Thus : order no. Order date

Functional dependency may also be based on a composite attribute. For example, if we write

$$X,Z \rightarrow Y$$

it means that there is only one value of Y corresponding to given values of X, Z. In other words, Y is functionally dependent on the composite X, Z. In other words, Y is functionally dependent on the composite X, Z. In table 1 mentioned below, for example, Order no., and Item code together determine Qty. and Price. Thus :

Order no., Item code → Qty., Price

As another example, consider the relation

Student (Roll no., Name, Address, Dept., Year of study)

Order no.	*Order date*	*Item code*	Quantity	*Price/unit*
1456	260289	3687	52	50.40
1456	260289	4627	38	60.20
1456	260289	3214	20	17.50
1886	040389	4629	45	20.25
1886	040389	4627	30	60.20
1788	040489	4627	40	60.20

Table 1: Normalized Form of the Relation

In this relation, Name is functionally dependent on Roll no. In fact, given the value of Roll no., the values of all the other attributes can be uniquely determined. Name and Department are not functionally dependent because given the name of a student, one cannot find his department uniquely. This is due to the fact that there may be more than one student with the ' same name. Name in this case is not a key. Department and Year of study are not functionally dependent as Year of study pertains to a student whereas Department is an independent attribute. The functional dependency in this relation is shown in the following figure as a dependency diagram. Such dependency diagrams shown in figure 1 are very useful in normal-ization

Relation Key: Consider the relation of table 1. Given the Vendor code, the Vendor name and Address are uniquely determined. Thus Vendor code is the relation key. Given a relation, if the value of an attribute X uniquely determines the values of all other attributes in

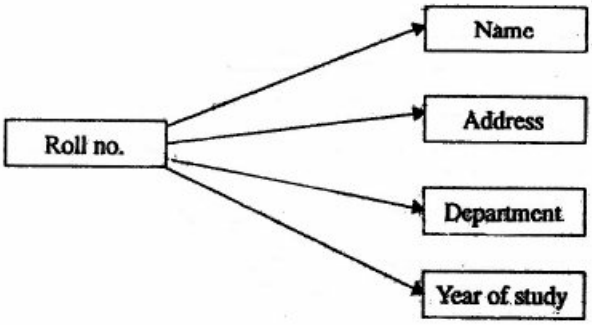


Figure 1: Dependency diagram for the relation "Student

a row, then X is said to be the key of that relation. Sometimes more than one attribute is needed to uniquely determine otherattributes in a relation row. In that case-stteh a set of

' *attributes is thqikgyl: In table I, Order no. and Item code together foAyiit key. In t|e

relation "Supplies" (Vendor code, Item code, Qty. supplied, Date of supply, Price/unit), Vendor code and Item code together form the key. This dependency is shown in the following diagram (figure 2).

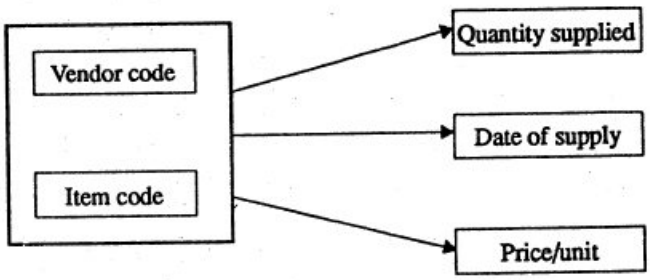


Figure 2: Dependency diagram for the relation "Supplies"

Observe that in the figure the fact that Vendor code and Item code together form a composite key is clearly shown by enclosing them together in a rectangle.

2.3 ANOMALIES IN A DATABASE

Consider the following relation scheme pertaining to the information about a student maintained by a university:

STDINF(Name, Course, Phone_No, Major, Prof, Grade)

Table 2 shows some tuples of a relation on the relation scheme **STDINF(Name, Course,** Phone_No, Major, Prof, Grade). The functional dependencies among its attributes are shown in Figure 3. The key of the relation is Name Course and the relation has, in addition, the following functional dependencies {Name → Phone_No, Name → Major, Name Course → Grade, Course → Prof}.

Name	Course	Phone_No	Major	Prof	Grade
Jones	353	237-4539	Comp Sci	Smith	A
Ng	329	427-7390	Chemistry	Turner	B
Jones	328	237-4539	Comp Sci	Clark	B
Martin	456	388-5183	Physics	James	A
Dulles	293	371-6259	Decision Sci	Cook	C
Duke	491	823-7293	Mathematics	Lamb	B
Duke	356	823-7293	Mathematics	Bond	in prog
Jones	492	237-4539	Comp Sci	Cross	in prog
Baxter	379	839-0827	English	Broes	C

Table 2: Student Data Representation in Relation STDINF

Here the attribute Phone_No, which is not in any key of the relation scheme STDINF, is not functionally dependent on the whole key but only one part of the key, namely, the attribute Name. Similarly, the attributes Major and Prof, which are not in any key of the relation scheme **STDINF** either, are fully functionally dependent on the attributes Name and Course, respectively. Thus the determinants of these functional dependencies are again not the entire key but only part of the key of the relation. Only the attribute Grade is fully functionally dependent on the key **Name Course.**

The relation scheme **STDINF** can lead to several undesirable problems:

- **Redundancy:** The aim of the database system is to reduce redundancy, meaning that information is to be stored only once. Storing information several times leads to the waste of storage space and an increase in the total size of the data stored.

Updates to the database with such redundancies have the potential of becoming inconsistent, as explained below. In the relation of table 2, the Major and Phone_No. of a student are stored several times in the database: once for each course that is or was taken by a student.

- **Update Anomalies:** Multiple copies of the same fact may lead to update anomalies or inconsistencies when an update is made and only some of the multiple copies are updated. Thus, a change in the Phone_No. of Jones must be made, for consistency, in all tuples pertaining to the student Jones. If one of the three tuples of Figure 3 is not changed to reflect the new Phone_No. of Jones, there will be an inconsistency in the data.

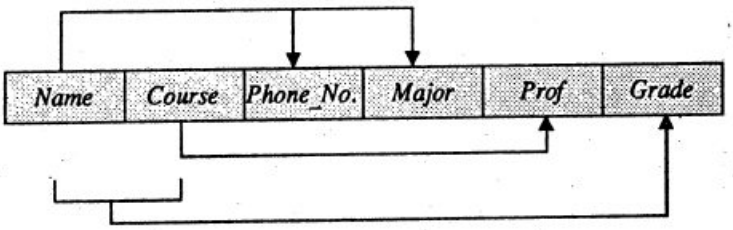


Figure 3: Function dependencies in STDINF

- **Insertion Anomalies:** If this is the only relation in the database showing the association between a faculty member and the course he or she teaches, the fact that a given processor is teaching a given course cannot be entered in the database unless a student is registered in the course. Also, if another relation also establishes a relationship between a course and a professor who teaches that course the information stored in these relations has to be consistent.
- **Deletion Anomalies:** If the only student registered in a given course discontinues the course, the information as to which professor is offering the course will be lost if this is the only relation in the database showing the association between a faculty member and the course she or he teaches. If another relation in the database also establishes the relationship between a course and a professor who teaches that course, the deletion of the last tuple in STDINF for a given course will not cause the information about the course's teacher to be lost.

The problems of database inconsistency and redundancy of data are similar to the problems that exist in the hierarchical and network models. These problems are addressed in the network model by the introduction of virtual fields and in the hierarchical model by the introduction of virtual records. In the relational model, the above problems can be remedied by decomposition. We define decomposition as follows:

Definition: Decomposition

The decomposition of a relation scheme $R = (A_1, A_2, \ldots A_n)$is its replacement by a set of relation schemes $\{R_1, R_2, \ldots R_m\}$, such that $R_1 \leq R$ for $1 \leq i \leq m$ and $R_1 \cup R_2 \cup R_m = R$.

A relation scheme R can be decomposed into a collection of relation schemes $\{R_1, R_2, R_3 \ldots R_m\}$ to eliminate some of the anomalies contained in the original relation R. Here the relation schemes R_1 $(1 \leq i \leq m)$ are subsets of R and the intersection of $R_1 \cap R_j$ for $i \neq j$ need not be empty. Furthermore, the union of R_j $(1 \leq i \leq m)$ is equal to R, i.e. $R = R_1\ R_2 \ldots R_m$.

The problems in the relation scheme STDINF can be resolved if we replace it with the following relation schemes:

STUDENT_INFO (Name, Phone_No, Major)

TRANSCRIPT (Name, Course, Grade)

TEACHER (Course, Prof)

The first relation scheme gives the phone number and the major of each student and such information will be stored only once for each student. Any change in the phone number will thus require a change in only one tuple of this relation.

The second relation scheme stores the grade of each student in each coarse that the student is or was enrolled in., (Note: In oar database we assume that either the student tales the course only once, or if he or she has to repeat it to improve the grade, the TRANSCRIPT relation . stores only the highest grade.)

The third relation scheme records the teacher of each course. One of the disadvantages of replacing the original relation scheme STDINF with the three relation schemes is that the retrieval of certain information requires a natural join operation to be performed. For instance, to find die majors of a student who obtained a grade of A in course 353 requires a join to be performed: (STUDENT_INFO ⋈ TRANSCRIPT). The same information could be derived from the original relation STDINF by selection and projection.

When we replace the original scheme STDINF with the relation schemes STUDENT_INFO. TRANSCRIPT and TEACHER, die consistency and referential integrity constraints have to be enforced. The referential integrity enforcement implies that if a tuple in the relation TRANSCRIPT exists, such as (Jones. 353. in prog), a tuple must exist in STUDENT_INFO with Name = Jones and furthermore, a tuple must exist in STUDENT_INFO with Course = 353. The attribute Name, which forms pan of the key of the relation TRANSCRIPT, is a key of the relation STUDENT_INFO. Such an attribute (or a group of attributes), which establishes a relationship between specific tuples (of the same or two distinct relation,), is called a foreign key. Notice that the attribute Course in relation TRANSCRIPT is also a foreign key. since it b a key of the relation TEACHER.

Note that the decomposition of STDINF into the relation schemes STUDENT (Name, Phone No, Major, Grade) and COURSE (Course. Prof.) Is a bad decomposition for the following reasons:

1. Redundancy and update anomaly, because the data for the attributes Phone no and Major are repeated
2. Loss of information, because we lose the fact that a student has a given grade in a particular course.

2.4 PROPERTIES OF NORMALIZED RELATIONS

Ideal relations after normalization should have the following properties so that the problems mentioned above do not occur for relations in die (ideal) normalized form:

1. No data value should be duplicated in different rows unnecessarily.
2. A value must be specified (and required) for every attribute in a row.
3. Each relation should be self-contained. In other words, if a row from a relation is deleted, important information should not be accidentally lost
4. When a row is added to a relation, other relations in the database should not be affected.
5. A write of an attribute in a tuple may be changed independent of other tuples in the relation and other relations.

The idea of normalizing relations to higher and higher normal forms Is to attain the goals of having a set of ideal relations meeting the above criteria.

2.5 FIRST NORMALIZATION

The relation shown in table 1 is said to be in First Normal Form, abbreviated as 1NF. This form is also called a flat file. There arc no composite attributes, and every attribute is single and describes one property.

Converting a relation to the 1NF form is the first essential step normalization. There arc successive higher normal forms known as 2NF. 3NF, BCNF, 4NF and 5NF. Each fa m is an improvement over the earlier form. In other words, 2NF is an improvement on 1NF. 3NF is an improvement on 2NF. and so on. A higher normal form relation is a subset of lower normal form as Shown in the following figure 4. The higher normalization steps are based on three important concepts:

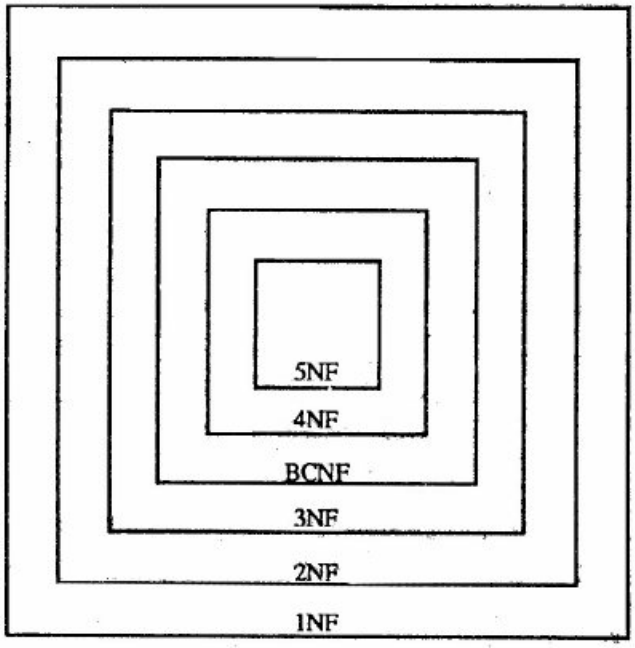


Figure 4: Illustration of successive normal forms of a relation

1. Dependence among attributes in a relation
2. Identification of an attribute or a set of attributes as the key of relation
3. Multivalued dependency between attributes

2.6 SECOND NORMAL FORM RELATION

We will now define a relation in the Second Normal Form (2NF): A relation is said to be in .(2NF if it is in 1NF and non-key attributes ate functionally dependent on the key attribute (s Further, if the key has more than one attribute then no non-key attributes should be functionally dependent upon a part of the key attributes. Consider, for example. the relation given in table t. This relation is in 1NF. The key is (Order no., Item code).The dependency diagram for. attributes of this relation is shown in figure 5. The non-key attribute Price/Unit is functionally dependent on Item code which is part of the relation key. ;Also. the non-key .attribute Order date is functionally dependent on Order no. which is a part of the relation key Thus the relation is net in 2NF. It can be transformed to 2NF by splitting it into three .relations as shown in table 3

In table 3 the relation Orders has Order no, as the key. The relation Oreder details has the composite key Order no, and Item code. In bothe relations the non-key attributes are functionally dependent on the whole key. Observe that by transforming to 2NF relations the

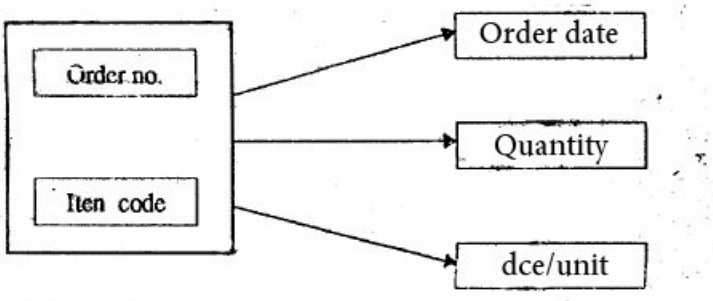


Figure : Dependency diagram for the relation give in table

repetition of Order date (table 1) has been removed. Further, if an order for an item is cancelled, the price of an item is not lost. For example, if Order no. 1886 for Item code 4629 is cancelled in table 1, then the fourth row will be removed and the price of the item is lost. In table 3 only the fourth row of the table 3(b) is omitted. The item price is not lost as it is available in table 3(c). The data of the order is also not lost as it is in table 3(a).

(a) Orders

Order no	*Order date*
1456	260289
1886	040389
1788	040489

(b) Order Details

Order no.	Item Code	Qty.
1456	3687	52
1456	4627	38
1456	3214	20
1886	4629	45
1886	4627	30
1788	4627	40

(c) Prices

Item code	Price/ unit
3687	50.40
4627	60.20
3214	17.50
4629	20.25

Table 3: Splitting of Relation given in table 1 into 2NF Relations

These relations in 2NF form meet all the "ideal" conditions specified. Observe that the three relations obtained as self-contained. There is no duplication of data within a relation.

2.7 THIRD NORMAL FORM

A Third Normal Form normalization will be needed where all attributes in a relation tuple are not functionally dependent only on the key attribute. If two non-key attributes are functionally dependent, then there will be unnecessary duplication of data. Consider the relation given in table 4. Here, Roll no. is the key and all other attributes are

Roll no.	Name	Department	Year	Hostel name
1784	Raman	Physics	1	Ganga
1648	Krishnan	Chemistry	1	Ganga
1768	Gopalan	Mathematics	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Maya	Geology	3	Krishna
1485	Singh	Zoology	4	Godavari

Table 4: A 2NF Form Relation

functionally dependent on it. Thus it is in 2NF. If it is known that in the college all first year students are accommodated in Ganga hostel, all second year students in Kaveri, all third year students in Krishna, and all fourth year students in Godavari, then the non-key attribute Hostel name is dependent on the non-key attribute Year. This dependency is shown in figure 6.

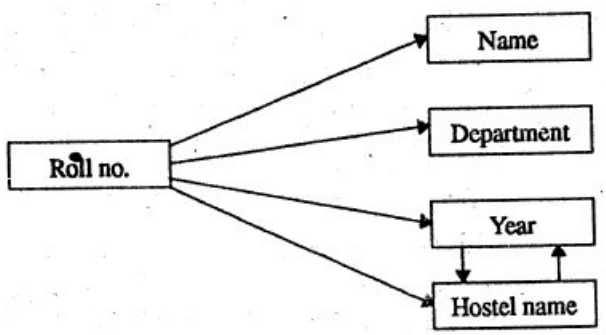


Figure 6: Dependency diagram for the relation

Observe that given the year of student, his hostel is known and vice versa. The dependency of hostel on year leads to duplication of data as is evident from table 4. If it is decided to ask all first year students to move to Kaveri hostel, and all second year students to Ganga hostel, this change should be made in many places in table 4. Also, when a student's year of study changes, his hostel change should also be noted in Table 4. This is undesirable. Table 4 is said to be in 3NF if it is in 2NF and no non-key attribute is functionally dependent on any other non-key attribute. Table 4 is thus not in 3NF. To transform it to 3NF, we should introduce another relation which includes the functionally related non-key attributes. This is shown in table 5.

Roll no.	Name	Department	Year
1784	Raman	Physics	1
1648	Krishnan	Chemistry	1
1768	Gopalan	Mathematics	2
1848	Raja	Botany	2
1682	Maya	Geology	3
1485	Singh	Zoology	4

Year	Hostel name
1	Ganga
2	Kaveri
3	Krishna
4	Godavari

Table 5: Conversion of table 4 into two 3NF relations

It should be stressed again that dependency between attributes is a semantic property and has to be stated in the problem specification. In this example the dependency between Year and Hostel is clearly stated. In case hostel allocated to students do not depend on their year in college, then table 4 is already in 3NF.

Let us consider another example of a relation. The relation **Employee** is given below and its dependency diagram in figure 7.

Employee (**Employee code,** Employee name, Dept., Salary, Project no., Termination date of project).

As can be seen from the figure, the termination date of a project is dependent on the Project no. Thus this relation is not in 3NF. The 3NF relations are:

Employee (**Employee code,** Employee name, Salary, Project no.)

Project (**Project no.** Termination date)

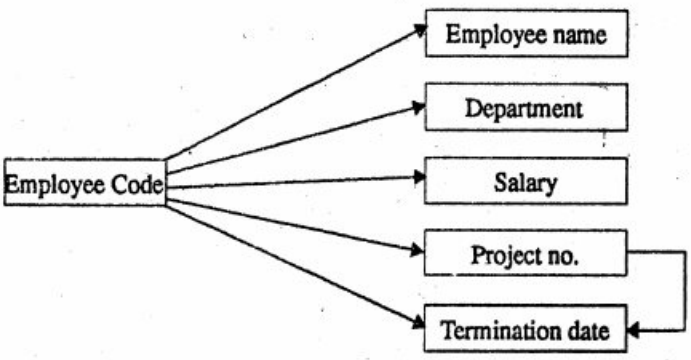


Figure 7: Dependency diagram of employee relation

2.8 BOYCE CODD NORMAL FORM (BCNF)

Assume that a relation has more than one possible key. Assume further that the composite keys have a common attribute. If an attribute of a composite key is dependent on an attribute of the other composite key, a normalization called BCNF is needed. Consider, as an example, the relation **Professor:**

Professor (**Professor code,** Dept., Head of Dept., Parent time)

It is assumed that

1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relationship diagram for the above relation is given in figure 8. Table 6 gives the relation attributes. The two possible composite keys are professor code and Dept. or Professor code and Head of Dept. Observe that department as well as Head of Dept. are not non-key attributes. They are a part of a composite key.

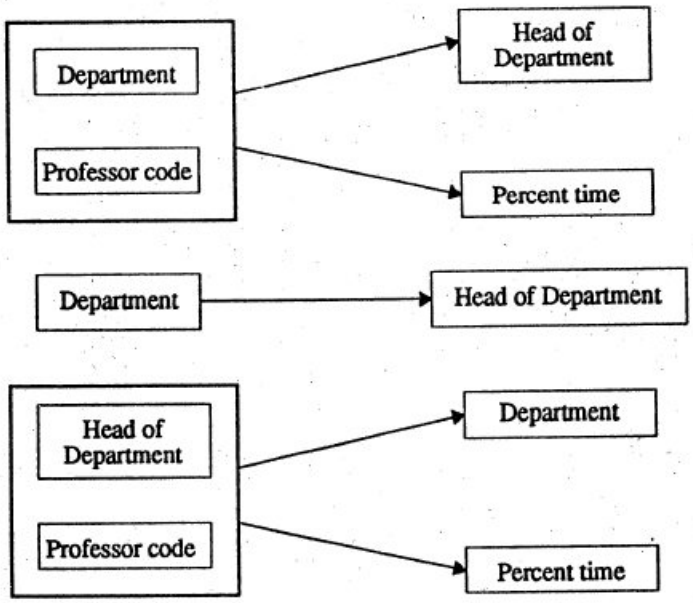


Figure 8: Dependency diagram of Professor relation

Professor Code	Department	Head of Dept.	Parent
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Table 6: Normalization of Relation "Professor"

The relation given in table 6 is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. from Professor relation. The normalized relations are shown in the following table 7.

(a)

Professor code	Department	Percent time
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25
P2	Physics	75
P3	Mathematics	100

(b)

Department	Head of Dept.
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

Table 7: Normalized Professor Relation in BCNF

and the dependency diagrams for these new relations in figure 8. The dependency diagram gives the important clue to this normalization step as is clear from figures 8 and 9.

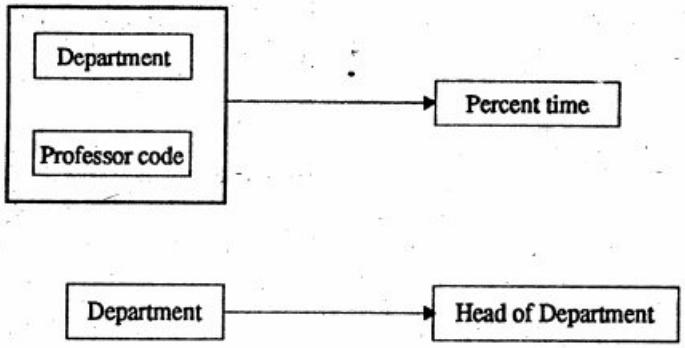


Figure 9: Dependency diagram of Professor relation

2.9 FOURTH AND FIFTH NORMAL FORM

When attributes in a relation have multivalued dependency, further Normalisation to 4NF and 5NF are required. We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organisation. The following are the assumptions:

1. A vendor is capable of supplying many items.
2. A project uses many items.
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.

Table 8 gives a relation for this problem and figure 10 the dependency diagram(s).

Vendor code	Item code	Project no.1
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P2

Table 8: Vendor-supply-projects Relation

The relation given in table 8 has a number of problems. For example:

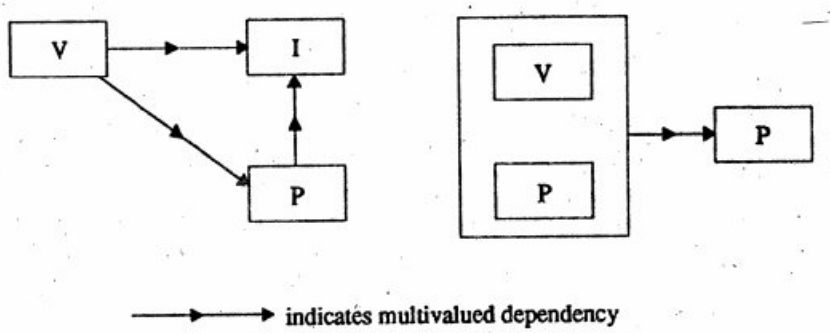


Figure 10 : Dependency diagrams of vendor-supply-project relation

- If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.
- The information about item 1 is stored twice for vendor V3.

Observe that the relation given in Table 8 is in 3NF and also in BCNF. It still has the problems mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multivalued dependency or one independent multivalued dependency with a functional dependency.

Table 8 can be expressed as the two 4NF relations given in Table 9. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

(a) Vendor Supply	
Vendor code	Item code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

(b) Vendor project	
Vendor code	Project no.
V1	P1
V1	P3
V2	P1
V3	P1
V3	P2

Table 9: Vendor-supply-project Relations in 4NF

These relations still have a problem. Even though vendor V1's capability to supply items and his allotment to supply for specified projects may not need it. We thus need another relation which specifies this. This is called 5NF form. The 5NF relations are the relations in Table 9(a) and 9(b) together with the relation given in table 10.

Project no.	Item code
P1	I1
P1	I2
P2	I1
P3	I1
P3	I3

Table 10: 5NF Additional Relation

In table 11 we summarise the normalisation steps already explained.

Input relation	Transformation	Output relation
All relations	Eliminate variable length records. Remove multiattribute lines in table	1NF
1NF relation	Remove dependency of non-key attribute on part of a multiattribute key	2NF
2NF	Remove dependency of non-key attributes on other non-key attributes	3NF
3NF	Remove dependency of an attribute of a multiattribute key on an attribute of another (overlapping) multi-attribute key	BCNF
BCNF	Remove more than one independent multivalued dependency from relation by splitting relation	4 NF
4NF	Add one relation relating attributes with multivalued dependency to the two relations with multivalued dependency	5 NF

Table 11: Summary of Normalisation Steps

2.10 SOME EXAMPLES OF DATABASE DESIGN

Consider a problem where items are supplied by a vendor and checked by a receiving process against orders for detecting any discrepancy. An order file is used to check whether the deliveries are against orders and whether there is any discrepancy. We will now see how these data can be organised as relations. The E-R diagram of figure 11 applies for this case and the relations are:

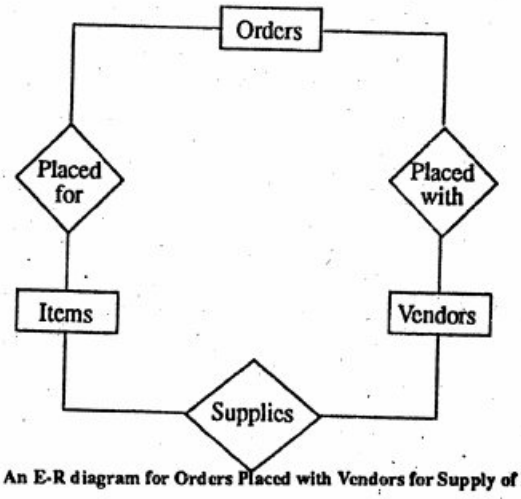


Figure 11: An E-R diagram for Orders Placed with Vendors for Supply of Items

ORDERS (**order no.**, order date)

ORDER PLACED FOR (**order no., item code**)

ORDER PLACED WITH (**order no., vendor code, item code,** qty. ordered, **price/unit,** delivery time allowed)

VENDORS (**vendor no.**, vendor name, vendor address)

ITEMS (**item code,** item name)

SUPPLIES (**vendor code, item code, order no.,** qty. supplied, date of supply)

The key attribute(s) are in bold letter(s) in each relation.

Let us examine whether the relations are in normal form. ORDERS and ORDER PLACED FOR are simple relations. In the relation ORDER PLACED WITH, the key is the composite attributes order no., vendor code and item code. However, order no. and vendor code are functionally related if we assume that a given order no. has only one vendor specified. Further, an order is with a vendor for an item. The price/unit depends on the item and the vendor. Thus we need to modify the relations ORDER PLACED FOR and ORDER PLACED WITH to :

ORDER PLACED FOR (**order no., Item code,** qty. ordered, price/unit, delivery time)

ORDER PLACED WITH (**vendor code, Item code**)

The two relations have composite keys. The non-key fields are not related to one another. In a key with more than one attribute the individual attributes are not functionally dependent. Thus these two relations are in normalised form and do not need any further change.

There is still one problem. Many orders may be given to the same vendor for the same or different items. In order to organise this data we need one more relation

ORDER WITH VENDOR (**order no.**, vendor code)

so that we can find out which vendor supplied against an order.

The relations VENDOR and ITEM are simple and are in normalised form. The relation SUPPLIES is, however, not normalised. Vendor code and order no. are functionally

dependent. There is a multivalued dependency between vendor code and item code as a vendor can supply many items. We thus split the relations into two relations:

ACTUAL VENDOR SUPPLY (**vendor no., item code,** qty. supplied, date of supply)

VENDOR SUPPLY CAPABILITY (**vendor code, item code**)

Observe that the relations VENDOR SUPPLY CAPABILITY and ORDER PLACED WITH have identical attributes. However, VENDOR SUPPLY CAPABILITY relation will have a **(vendor code, item code)** table without a vendor having supplied any item. The relation ORDER PLACED WITH will have a tuple only when a vendor actually supplies an item.

We now consider another problem. Let a database contain the following: Teacher code, Teacher's name, Teacher's address, rank, department, courses taught by the teacher, course name, credits for course, no. of students in the class, course taught in semester no., student no., name, dept., year, and courses taken in semester no. The following information is given on dependencies.

- A teacher may teach more than one course in a semester.
- A teacher is affiliated to only one department.
- A student may take many courses in a semester.
- The same course may have more than one section and different sections may be taught by different teachers.

An entity-relationship diagram for this problem is given in figure 12. The relations corresponding to the E-R diagram are:

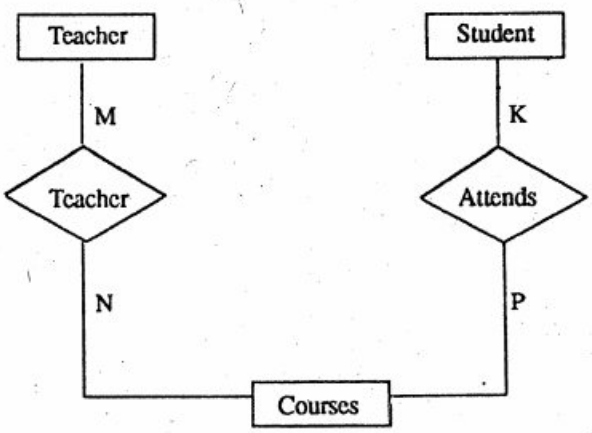


Figure 12 : An E-R diagram for teacher database

TEACHER (**Teacher code, course no.,** no. of rank, dept.)

TEACHES COURSES (**Teacher code, course no.,** no. of students)

COURSES (**course no.,** course name, credits, semester taught)

STUDENT (**student no.,** student's name, dept., year)

STUDENT-COURSES (**student's no., course no.,** semester no.)

TEACHER relation has only one key. All non-key attributes are functionally dependent only on the key. There is no functional dependency among non-key attributes. Thus the relation is normalised in 3NF. (No higher NFs are applicable).

STUDENT relation is also, similarly, in 3NF. In the COURSE relation, course name could also be a key However there is no overlapping multiattribute key. The relation is in 3NF and no further normalisation is required. The relations TEACHES COURSES and STUDENT- COURSES have multiattribute keys, but the relations themselves are in normal form. The only point which is not clear, from these relations, is the relation between teacher and student. This has been missed in the EIR diagram. The relationship is between the

teacher, courses taught and students. In other words, we should be able to answer the question "Which teacher is teaching course no. X to student no. Y?" Let us add a relation

TEACHES TO **(Teacher code, student no., course no.)**

In this relation **Teacher code** and **course no.**, have a multivalued dependency. Similarly, **Teacher code** and **student no.** as well as **student no.** and **course no.** have multivalued dependency. However, TEACHES COURSES **(Teacher code, course no.,** no. of students) and STUDENT-COURSES **(student no., course no.,** semester no.) relations are already in the database. Thus the relation TEACHES TO as it is specified above is sufficient to give the idea that student Y takes course X from Teacher Z.

Check Your Progress

1. What is the basic purpose of 4NF?

   .....

   .....

2. What types of anomalies are found in relational database?

   .....

   .....

   .....

2.11 SUMMARY

In this unit, we pointed out different types of anomalies in the database that could cause an undesirable effect. We also discussed several forms of normalization that could help in removing these anomalies.

2.12 MODEL ANSWERS

1. The 2nd, 3rd and BCNF normal forms deal with functional dependencies only. It is possible for a relation in 3NF to still exhibit update, insertion and deletion anomalies. This can happen when multivalued dependencies are not properly taken care of. In order to eliminate anomalies arise out of these dependencies, the notion of 4NF was developed.
2. There are 3 types of anomalies in database. These are:
   (i) Insertion anomalies
   (ii) Deletion anomalies
   (iii) Update anomalies

2.13 FURTHER READINGS

1. Bipin.C. Desai, *An Introduction to Database System*, Galgotia Publication, New Delhi.
2. V. Rajaraman, *Analysis and Design of Informatic System*, PHI, New Delhi-1995.

UNIT 3 STRUCTURED QUERY LANGUAGE

Structure

3.0 INTRODUCTION

SQL is an acronym for Structured Query Language. It is available in a number of data base management packages based on the relational model of data, for example, in DB2 of the IBM and UNIFY of the UNIFY corporation.

Originally defined by D.D. Chamberlain in 1974, SQL underwent a number of modifications over the years. Today, SQL has become an official ANSI standard.

It allows for data definition, manipulation and data control for a relational database. The data definition facilities of SQL permit the definition of relations and of various alternative views of relations. Further, the data control facility gives features for one user to authorize other users to access his data. This facility also permits assertions to be made about data integrity. All the three major facilities of SQL, namely, data manipulation, data definition and data control are bound together in one integrated language framework.

3.1 OBJECTIVES

After going through this unit you will be able to:

- Differentiate SQL commands
- List data manipulation commands
- List data definition commands
- Make queries using data manipulation commands.

3.2 CATEGORIES OF SQL COMMANDS

SQL commands can be roughly divided into three major categories with regard to their functionality. Firstly, there are those used to create and maintain the database structure. The second category includes those commands that manipulate the data in such structures, and thirdly there are those that control the use of the database. To have all this functionality in a single language is a clear advantage over many other systems straightway, and must certainly contribute largely to the rumour of it being easy to use.

It's worth naming these three fundamental types of commands for future reference. Those that create and maintain the database are grouped into the class called DDL or Data Definition Language statements and those used to manipulate data in the tables, of which there are four are the DML or Data Manipulation Language commands. To control usage of the data the DCL commands (Data Control Language) are used, and it is these three in conjunction plus one or two additions that define SQL. There are therefore no environmental

statements, as one finds so irritating in COBOL: for example, no statements to control program flow (if/then/else, perform, go to) and of course, no equivalent commands to open and close files, and read individual records. At this level then, it is easy to see where SQL gets its **end- user-tool** and **easy-to-use** tags.

The Data Definition Statements

To construct and administer the database there are two major DDL statements - CREATE and DROP, which form the backbone of many commands:

CREATE DATABASE to create a database DROP DATABASE to remove a database CREATE TABLE to create a table DROP TABLE to drop a table CREATE INDEX to create an index on a column DROP INDEX to drop an index CREATE VIEW to create a view DROP VIEW to drop a view.

There may be some additional ones, such as ALTER TABLE or MODIFY DATABASE, which are vendor specific.

The Data Manipulation Statements

To manipulate data in tables directly or through views we use the four standard DML statements:

SELECT DELETE INSERT UPDATE

These statements are now universally accepted, as is their functionality, although the degree to which these commands support this functionality varies somewhat between products compare the functionality of different implementation of UPDATE for example.

Data Control

This deals with three issues

(a) Recovery and Concurrency

Concurrency is concerned with the manner in which multiple users operate upon the data base.

Each individual user can either reflect the updates of a transaction by using the COMMIT or can cancel all the updates of a transaction by using ROLLBACK.

(b) Security

Security has two aspects to it.

The first is the VIEW mechanism. A view of a relation can be created which hides the sensitive information and defines only that part of a relation which should be visible. A user can then be allowed to access this view.

```
CREATE VIEW LOCAL AS
SELECT * FROM SUPPLIER
WHERE SUPPLIER.CITY = 'Delhi'
```

The above view reveals only the suppliers of Delhi.

The second is by using GRANT operation. This shall grant one or more access rights to perform the data manipulative operations on the relations.

(c) Integrity Constraints

Integrity constraints are enforced by the system. For example, one can specify that an attribute of a relation will not take on null values.

3.3 DATA DEFINITION

Data definition in SQL is via the create statement. The statement can be used to create a table, index, or view (i.e., a virtual table based on existing tables). To create a table, the create statement specifies the name of the table and the names and data types of each column

of the table. Its format is :

create table <relation> (<attribute list>)

where the attribute list is specified as:

<attribute list> :: = <attribute name> (<data type>)[not null] <attribute list>

The data types supported by SQL depend on the particular implementation. However, the following data types are generally included: integer, decimal, real (i.e., floating point values), and character strings, both of fixed size and varying length. A number of ranges of values for the integer data type are generally supported, for example, integer and smallint. The decimal value declaration requires the specification of the total number of decimal digits for the value and (optionally), the number of digits to the right of the decimal point. The number of fractional decimal digits is assumed to be zero if only the total number of digits is specified.

<data type> :: = <integer> | <smallint> | <char(n)> | <float> | <decimal (p[,q])>

In addition, some implementations can support additional data types such as bit strings, graphical strings, logical, data, and time. Some DBMSs support the concept of date. One possible implementation of date could be as eight unsigned decimal digits representing the data in the yyyymmdd format. Here yyyy represents the year, mm represents the month and dd represents the day. Two dates can be compared to find the one that is larger and hence occurring later. The system ensures that only legal date values are inserted (19860536 for the date would be illegal) and functions are provided to perform operations such as adding a number of days to a date to come up with another date or subtracting a date from the current date to find the number of days, months, or years. Date constants are provided in either the format given above or as a character string in one of the following formats: mm/dd/yy; mm/dd/yyyy; dd-mmm-yy; dd-mmm-yyyy. In this text we represent a date constant as eight unsigned decimal digits in the format yyyymmdd.

The employee relation for the hotel database can be defined using the create table statement given below. Here, the Empl_No is specified to be **not null** to disallow this unique identifier from having a null value. SQL supports the concept of null values and, unless a column is declared with the not null option, it could be assigned a null value.

create table	**EMPLOYEE**
Empl_No	**integer not null,**
Name	**char** (25),
Skill	**char** (20)
Pay-Rate	**decimal** (10,2))

The definition of an existing relation can be altered by using the **alter** statement. This statement allows a row column to be added to an existing relation. The existing tuples of the altered relation are logically considered to be assigned the null value for the added column. The physical alteration occurs to a tuple only during an update of the record.

alter table existing-table-name

add column-name data-type [....]

alter table EMPLOYEE

add Phone_Number **decimal** (10)

The **create index** statement allows the creation of an index for an already existing relation. The columns to be used in the generation of the index are also specified. The index is named and the ordering for **each column** used in the index can be specified as either ascending or descending. The **cluster** option could be specified to indicate that the records are to be placed in physical proximity to each other. The **unique** option specifies that only one record could exist at any time with a given value for the column(s) specified in the statement to create the index. (Even though this is just an access aid and a wrong place to declare the primary key). Such columns, for instance, could form the primary key of the relation and hence duplicate tuples are not allowed. One case is the ORDER relation where the key is the combination of the attribute Bill#, Dish#. In the case of an existing relation, an attempt to create an index with the unique option will not succeed if the relation does not satisfy this uniqueness criterion. The syntax of the create index statement is shown below:

create [unique] index name-of-index

on existing- table-name

(column-name[ascending or descending]

[,column-name[order]....])

[cluster]

The following statement causes an index called empindex to be built on the columns Name and Pay_Rate. The entries in the index are ascending by Name value and descending by Pay_Rate. In this example there are no restrictions on the number of records with the same Name and Pay_Rate.

Create index empindex

on EMPLOYEE (Name **asc**, Pay_Rate **desc**);

An existing relation or index could be deleted from the database by the **drop** SQL statement. The syntax of the drop statement is as follows:

drop table existing-table-name;

drop index existing- index-name;

3.4 DATA MANIPULATION

Data manipulation capabilities allows one to retrieve and modify contents of the data base. The most important of these is the SELECT operation which allows data to be retrieved from the data base.

The relation definitions that shall be used in the rest of the module are given below.

There are parts which are supplied by suppliers. S contains the details about each supplier. Turnover for a supplier is in terms of lakhs of rupees. Information regarding suppliers of specific parts is contained in SP whereas information about the parts themselves is contained in P.

S

S#	SNAME	SCITY	TURNOVER
10	CAUVERY	BANGALORE	50
11	NARMADA	BOMBAY	100
12	YAMUNA	DELHI	70
13	TAPI	BOMBAY	20

P

P#	WEIGHT	COLOUR	COST	SELLING PRICE
1	25	RED	10	30
2	30	BLUE	15	45
3	45	RED	20	45

SP

S#	P#	QTY
10	1	100
11	1	5
10	2	50
11	2	30
10	3	10
12	3	100
13	1	20

3.4.1 SELECT – The Basic Form

The select statement specifies the method of selecting the tuples of the relations(s). The tuples processed are from one or more relations specified by the form clause of the select statement.

The basic form of SELECT is

Select <target list>

from <relation list>

[where <predicate>]

SELECT lists the attributes to be selected

FROM relations from which information is to be used

WHERE condition. The rows that qualify are those for which the condition evaluates to true.

Condition is a single predicate or a collection of predicates combined using the Boolean operators AND, OR and NOT.

The column names following SELECT are to be retrieved from the relations specified in the FROM part. WHERE specifies the condition that the tuples must satisfy in order to be part of the result.

Below we shall state first the retrieval query in English and they specify its SQL equivalent.

Unqualified Retrieval

1. Get the part numbers of all the parts being supplied.

   SELECT P#

   FROM SP

P#
1
1
2
2
3
3
1

   Part numbers getting repeated? That's right. SELECT does not eliminate duplicate rows (unlike the project operation of the relational algebra). In order to do that

2. Get the part numbers of all the parts being supplied with no duplicates.

   SELECT DISTINCT P#

   FROM SP

P#
1
2
3

   If all the columns of the relation are to be retrieved then one needn't list all of them. A * can be specified after SELECT to indicate retrieval of the entire relation.

3. Get full details of all suppliers.

   SELECT *

   FROM S

   S

S#	SNAME	SCITY	TURNOVER
10	CAUVERY	BANGALORE	50
11	NARMADA	BOMBAY	100
12	YAMUNA	DELHI	70
13	TAPI	BOMBAY	20

The ORDER BY clause

The result of a query can be ordered either in ascending (ASC) order or in descending (DESC) order.

4. Get the supplier numbers and turnover in descending order of turnover.

```
SELECT S#, TURNOVER
FROM S
ORDER BY TURNOVER DESC
```

S#	TURNOVER
11	100
12	70
10	50
13	20

Instead of a column name, the ordinal position of the column in the result can be used. That is, the above query can be rewritten as

5.
```
SELECT S#, TURNOVER
FROM S
ORDER BY 2 DESC
```

The format of the order clause is

ORDER BY {int/col [ASC/DESC], ... }

col – column name

int – ordinal position of the column in the result table

ASC/DESC – Ascending or descending

If there is more than one specification, then the left-to- right specification corresponds to major-to-minor ordering. This is shown below.

6. Get the supplier number and part number in ascending order of supplier number and descending order for the part supplied for each supplier.

```
SELECT   S#, P#, QTY
FROM SP
ORDER BY  S#, P# DESC
```

S#	P#	QTY
10	3	10
10	2	50
10	1	100
11	2	30
11	1	5
12	3	100
13	1	20

Qualified retrieval

The expression following WHERE specifies the condition that must be satisfied. Below we consider a few examples.

7. Get the details of suppliers who operate from Bombay with turnover 50.

```
SELECT S.*
FROM S
WHERE CITY = 'BOMBAY' AND  TURNOVER > 50
```

S#	SNAME	SCITY	TURNOVER
11	NARMADA	BOMBAY	100

The above form is a conjunction of comparison predicates. A comparison predicate is of the form

scalar-expr O scalar-expr

where O is any of the six relational operators

=, < >, <, >, <=, >=

and a scalar expression is an arithmetic expression with

operators as +, –, *, /

operands as col., function, constant

BETWEEN Predicate

8. Get the part numbers weighing between 25 and 35

```
SELECT   P#, WEIGHT
FROM P
WHERE  WEIGHT BETWEEN 25 AND 35
    P#      WEIGHT
    1          25
    2          30
```

The use of BETWEEN gives the range within which the values must lie. If the value should lie outside a range then BETWEEN is to be preceded by NOT. For example,

```
SELECT P#
FROM P
WHERE  WEIGHT   NOT BETWEEN 25 AND 35
    P#          WEIGHT
    3             45
```

would retrieve all part numbers whose weight is less than 25 or greater than 35 as shown above.

LIKE Predicate

This predicate is used for pattern matching. A column of type char can be compared with a string constant. The use of the word LIKE doesn't look for exact match but a form of wild string match. A % or - can appear in the string constant where

% stands for a sequence of n (>=0) characters

– stands for a single character

Examples

ADDRESS LIKE '%Bangalore%' - ADDRESS should have Bangalore somewhere as a part of it if the match is to succeed.

STRANGE STRING LIKE '\-%' ESCAPE'\'

Here, the normal meaning of – is overridden with the use of the escape character. STRANGE above will match with any string beginning with –

9. Get the names and cities of suppliers whose name begin with C

```
SELECT        SNAME,        SCITY
FROM S
WHERE         SNAME LIKE'C%'
SNAME         SCITY
CAUVERY       BANGALORE
```

When the data is to be retrieved from more than one relation then both the relation names is specified in the FROM clause and the join condition in the WHERE part.

10. For each part supplied, get part number and names of all cities supplying the part.

```
SELECT      P#, CITY
FROM        SP, S
WHERE       SP.S# = S.S#
```

P#	SCITY
1	BANGALORE
1	BOMBAY
2	BANGALORE
2	BOMBAY
3	BANGALORE
3	DELHI
1	BOMBAY

How does, then, one specify a join on the same relation?

11. Get pairs of supplier numbers such that both operate from the same city.

```
SELECT      FIRST.S#, SECOND.S#
FROM        S FIRST, S SECOND
WHERE       FIRST.CITY = SECOND.CITY
AND         FIRST.S# <>SECOND.S#
```

FIRST and SECOND are tuple variables, both ranging over S. The last line eliminates a supplier getting compared with himself.

S#	S#
11	13
13	11

But, we see that suppliers with numbers 11 and 13 are getting compared twice. Can that be avoided? How about < instead of <> ?

```
SELECT      FIRST.S#, SECOND.S#
FROM        S FIRST, S SECOND
WHERE       FIRST.CITY = SECOND.CITY
AND         FIRST.S#  SECOND.S#
```

Tests for NULL

An attribute can be tested for the presence or absence of null.

12. Get the supplier numbers whose turnover is null

```
SELECT      S#
FROM        S
WHERE       TURNOVER IS NULL
```

There is no tuple in the result of this query as in the sample

Can the last line in the above query be replaced by

```
WHERE       TURNOVER = NULL
```

Not really! It is incorrect as nothing (even NULL) is equal to NULL.

The format for specifying NULL is

col. ref IS [NOT] NULL

IN Predicate

This is to be used whenever you want to test whether an attribute value is one of a set of values. For example,

13. Get the part numbers that cost 20, 30 or 40 rupees.

```
SELECT      P.P#, SELLING PRICE
FROM        P
WHERE       SELLING PRICE IN (20, 40, 45)
```

P#	SELLING PRICE
2	45
3	45

It's a quicker way of specifying comparison.

The format of the predicate

scalar-expr [NOT] IN (atom list)

3.4.2 Subqueries

The expression following WHERE can be either a simple predicate as explained above or it can be a query itself! This part of the query following WHERE is called a Subquery.

A subquery, which in turn is a query, can have its own subquery and the process of specifying subqueries can continue ad infinitum! More practically, the process ends once the query has been fully expressed as a SQL statement.

Subqueries can appear when using the comparison predicate, the IN predicate and when quantifiers are used (not yet explained).

Comparison Predicate

14. Get the supplier numbers of suppliers who are located in the same city as Tapi.

```
SELECT      S.S#, SNAME
FROM        S
WHERE       S.CITY =
(SELECT     S.CITY
FROM        S
WHERE       SNAME = 'TAPI')
```

The inner select (subquery) retrieves the city of the supplier named Tapi. The outer select (the main one) then compares the city of each supplier in the supplier relation and picks up those where the comparison succeeds.

S#	SNAME
11	NARMADA
13	TAPI

Notice that the subquery appears after the comparison operator. The format of this form of expression

scalar-expr operator subquery

IN Predicate

In this form the subquery selects a set of values. The outer query checks whether the value of a specified attribute is in this set.

15. Get the names of suppliers who supply part 2

```
SELECT    S.SNAME
FROM      S
WHERE     S.S# IN
(SELECT   SP.S#
FROM      SP
WHERE     SP.P# = 2)
```

SNAME
CAUVERY
NARMADA

The above query can be equivalently expressed as

```
SELECT    S.SNAME
FROM      S
WHERE     2 IN
(SELECT   P#
FROM      SP
WHERE     S.S# = S#)
```

S# is unqualified and therefore, refers to SP. That is because every unqualified attribute name is implicitly qualified with the relation name from the nearest applicable FROM clause.

Quantified predicates

The two quantifiers that can be used are the ALL and ANY. Any stands for the existential quantifier and ALL for the universal quantifier.

Lets first look at ANY. It can be specified in a comparison predicate just following the comparison operator. That is,

scalar-expr O ANY subquery

The subquery is first evaluated to give a set of values. The above expression is true if the scalar-expr is O comparable with any of the values that form the result of the subquery.

16. Get the part numbers for parts whose cost is less than the current maximum cost.

```
SELECT    P#, COST
FROM      P
WHERE     COST < ANY
(SELECT   COST
FROM      P)
```

P#	COST
1	10
2	15

The inner select gets the cost of all the parts. In the outer select, a P# is selected if its cost is less than some element of the set selected in the earlier step.

17. Get the supplier names of suppliers who do not supply part 2.

```
SELECT    SNAME
FROM      S
WHERE     2 < > ALL
(SELECT   P#
FROM      SP
WHERE     S# = S.S#)
```

```
SNAME
YAMUNA
TAPI
```

For each supplier, all the parts supplied by him are collected in the inner select. If none of them is equal to P2 then the condition evaluates to true and supplier name forms part of the result.

Existence Test

This kind of an expression is used when it is necessary to find out if the set of values retrieved by using a subquery contains an element or not.

18. Get the supplier names of suppliers who supply at least one part.

```
SELECT    SNAME
FROM      S
WHERE     EXISTS
(SELECT   *
FROM      SP
WHERE     SP.S# = S.S#)
```

For a given supplier, if the subquery selects at least one tuple then the condition (which follows WHERE) evaluates to true. Then, the name of the supplier is selected. IN our data base every supplier is supplying at least one part. So the names of all of them would be part of the result.

3.4.3 Functions

Some standard functions are defined in SQL and can be used when framing queries. There are five built-in functions. These are

COUNT	–	number of values of a column
SUM	–	Sum of values in a column
AVG	–	Average of values in a column
MAX	–	Maximum of all the values in a column
MIN	–	Minimum of all the values in a column

If the function is followed by the word DISTINCT then unique values are used. On the other hand, if ALL follows the function then all the values are used for evaluating the function. ALL is the default.

COUNT(*) has a special meaning in that it counts the number of rows of a relation. COUNT

in any other form must make use of DISTINCT. In other words, except when rows are counted, COUNT always returns the number of distinct values in a column.

19. Get the total number of suppliers

```
SELECT    COUNT(*)
FROM      S
```

COUNT(*) counts the number of tuples of S and hence, the number of suppliers.

20. Get the total quantity of Part 2 that is supplied.

```
SELECT    SUM (SP.QTY)
FROM      SP
WHERE     SP.P# = 2
```

The answer is one of

a) 25 b) 40 c) 80 d) 100

21. Get the part numbers whose cost is greater than the average cost.

```
SELECT    P#
FROM      P
WHERE     COST
(SELECT   AVG(COST)
FROM      P)
```

22. Get the names of suppliers who supply from a city where there is atleast one more supplier.

```
SELECT    SNAME
FROM S    FIRST
WHERE     2
(SELECT   COUNT (CITY)
FROM      S
WHERE     CITY = FIRST.CITY)
```

Some practice exercise before we move on to the next section. Try and write the expression yourself before looking at the solution. Yours might be different from the solution given in the notes. For all you know, yours might be a better solution. So, go-ahead and try. Use any feature that has been covered till now.

1. Get the names of suppliers who supply at least one red part

```
SELECT    SNAME
FROM      S
WHERE     S#  IN
( SELECT  S#
 FROM     SP
WHERE     P# IN
( SELECT  P#
FROM      P
WHERE     COLOUR = 'RED') )
```

2. Get the supplier numbers who supply at least one part supplied by supplier 10.

```
SELECT    DISTINCT S#
FROM      SP
WHERE     P# IN
(SELECT   P#
FROM      SP
WHERE     S# = 10 )
```

3.4.4 GROUP BY Feature

This feature allows one to partition the result into a number of groups such that all rows of the group have the same value in some specified column.

23. Get the part number and the total quantity.

```
SELECT  P#, SUM(QTY)
FROM SP
GROUP BY  P#
```

P#	SUM(QTY)
1	125
2	80
3	110

GROUP BY groups together all the rows which have the same value for P#. The function SUM is then applied to each group. That is, the result consists of a part number along with the total quantity in which it is supplied.

Whenever GROUP BY is used then the phrase WHERE is to be replaced by HAVING. The meaning of HAVING is the same as WHERE except that the condition is now applicable to each group.

24. Get the part numbers for parts supplied by more than one supplier.

```
SELECT      P#
FROM        SP
GROUP BY    P#
HAVING      COUNT(*) > 1
```

Each group contains one or more tuples which have the same part number. COUNT(*) is applied to each such group.

The result before COUNT(*) is applied is

P#
1
2
3

In this case all the part numbers will be selected.

3.4.5 Updating the Database

The contents of the database can be modified by inserting a new tuple, deleting an existing tuple or changing the values of attributes of one or more tuples.

INSERT

The insertion facility allows new tuples to be inserted into given relations. Attributes which are not specified by the insertion statement are given null values. Consider

1. Add a part with number 14, weight 10, coloured red, with the cost and selling price as 20 and 60 respectively.

   INSERT INTO P :

   < 14, 10, 'red', 20, 60 >

The tuple is inserted into P.

If all the fields are not known then a tuple can still be added. The attributes whose values are not specified will have a null value.

INSERT INTO P :

<15, 'GREEN'>

The values for fields the weight, cost and the selling price which are not specified are assumed to be null.

2. Let us assume that there is a relation called RED-PART with one column P#.

   INSERT INTO RED-PART :

   SELECT P#

   FROM P

   WHERE COLOUR = 'red'

The various attributes of P having red colour are identified and inserted into the relation RED-PART.

DELETE

The deletion facility removes specified tuples from the database. Consider

1. Delete supplier 13

   DELETE S

   WHERE S# = 13

   Since S# is the primary key only one tuple will be deleted from S.

2. Delete all suppliers who supply from Bangalore

   DELETE S

   WHERE SCITY = 'BANGALORE'

   Here, more than one supplier can get deleted.

3. Delete all the suppliers

   DELETE S

   The definition of S exists but the relation is empty.

4. Delete all the supplies involving red coloured parts.

   DELETE SP
   WHERE 'red' =
   (SELECT COLOUR
   FROM P
   WHERE P.P# = SP.P#)

UPDATE

When columns are to be modified SET clause is used. This clause specifies the update to be made to selected tuples.

1. Change the city of supplier 13 to Bangalore and increase the turnover by 20 lakhs.

```
UPDATE   S
SET CITY = 'BANGALORE'
TURNOVER = TURNOVER + 20
WHERE    S# = 13
```

2. Increase quantity by 10 for all supplies of red coloured parts.

```
UPDATE   SP
SET QTY = QTY + 10
WHERE    P# IN
( SELECT   P#
FROM       P
WHERE  COLOUR = 'RED')
```

3.4.6 Data Definition Facilities

Data definition facilities permit users to create and drop relations, define alternative views of relations.

CREATE statement allows to define a relation. The name of the relation to be created and its various fields together with their data types must be specified. If a certain attribute is barred from containing null values then a NONULL specification must be made for it.

It must be noted that the word TABLE is used in this syntax instead of RELATION.

Example

```
CREATE TABLE DEPT
(DNO(CHAR(2),NONULL),
DNAME (CHAR (12) VAR),
LOC(CHAR(20) VAR))
```

VIEW

A very important aspect of data definition is the ability to define alternative views of data. The process of specifying an alternative view is very similar to that of framing a query. The derived relation is stored and can be used thereafter as an object of the various commands. It is also possible to define other views on top of the newly created relation.

Example

```
DEFINE VIEW D50 AS
SELECT EMPNO, NAME, JOB
FROM EMP
WHERE DNO = 50
```

D50 contains the employee number, name and job of those employees who are in department 50.

3.5 VIEWS

A view is a virtual table, that is one that does not actually exist. It is made up of a query on other tables in the database. It could include only certain columns or rows from a table or from many tables. A view which restricts the user to certain rows is called a horizontal view and a vertical view restricts the user to certain columns. You are not restricted to purely horizontal or vertical slices of data.

A view can be as complicated as you like. You can have grouped views where the query contains a GROUP BY clause. This makes the view a summary of the data in a table or tables.

If the list of column names is omitted the columns in the view take the same name as in the underlying tables. You must specify column names if the qurey includes calculated columns or two columns with the same name. There are several advantages to views, including :

- **Security :** Users can be given access to only the rows/columns of tables that concern them. The table may contain data for the whole firm but they only see data for their department.
- **Date integrity :** The WITH CHECK OPTION clause is used to enforce the query conditions on any updates to the view. If the view shows data for a particular office the user can only enter data for that office.
- **Simplicity :** Even if a view is a multi-table query, querying the view still looks like a single-table query.
- **Protection from change :** If the structure of the database changes, the user's view of the data can remain the same.

There are two disadvangages to views :

- **Performance :** A view may look like a single table but underneath the DBMS is usually still running multi-table queries. IF the view is complex then even simple queries can take a long time.
- **Update restrictions :** Updating the data through a view may or may not be possible. If the view is complex the DBMS may decide it can't perform updates and make the view read-only.

The ISO standard specifies five conditions that a view must meet in order to allow updates :

- The view must not have a DISTINCT clause
- The view must only name one table in the FROM clause
- All columns must be real columns — no expressions, calculated columns or column functions
- The WHERE clause must not contain a sub-query
- There must be no GROUP BY or HAVING clause

You will find that most dialects of SQL are not quite so restrictive. The underlying principle is that updates are allowed if the rows and columns of the view are traceable back to actual rows and columns in tables.

The format of view statement is as follows :

create view <view name> as query expression

A view is a relation (virtual rather than base) and can be used in query expressions, that is, queries can be written using the view as a relation. Views generally are not stored, since the data in the base relations may change. The base relations on which a view is based are sometimes called the **existing relations.** The definition of a view in a **create view** statement is stored in the system catalog. Having been defined, it can be used as if the view really represented a real relation. However, such a virtual relation defined by a view is recomputed whenever a query refers to it.

Example

(a) For reasons of confidentiality, not all users are permitted to see the Pay_Rate of an employee. For such user the DBA can create a view, for example, EMP_VIEW defined as:

```
create view EMP_VIEW as
(select Empl_No, Name, Skill
from EMPLOYEE)
```

(b) A view can be created for a subset of the tuples of a relation, as in this example. For assigning employees to particular jobs, the manager requires a list of the employees who have not been assigned to any jobs:

```
create view FREE as
(select Empl_No
from EMPLOYEE)
minus
(select Empl_No
from DUTY_ALLOCATION)
```

(c) The view in part (b) above can also be created using the following statements:

```
create view FREE as
(select Empl_No
from EMPLOYEE)
where Empl_No any
(select Empl_No
from DUTY_ALLOCATION)
```

In the above examples, the names of the attributes in the views are implicitly taken from the base relation. The data types of the attribute of the view are inherited from the corresponding attributes in the base relation. We can, however, given new names to the attributes of the view. This is illustrated in the syntax of the create view statement given below:

```
create view VIEW_NAME
(Name1, Name 2, ....)
as (select .....)
```

Here the attributes in the view are given as Name1, Name 2, ... and these names are associated with the existing relation by order correspondence. The definition of a view is accomplished by means of a subquery involving a select statement as given in the syntax above. Since a view can be used in a select statement, a view can be defined on another existing view.

3.6 SUMMARY

Most commercial relational DBMSs support some form of the SQL data manipulation language, and this creates different dialects of SQL. SQL has been standardised; that is, a minimum compatible subset is specified as a standard. In addition, embedded versions of SQL are supported by many commercial DBMSs. This allows application program written in a high-level language such as BASIC, C, COBOL, FORTRAN, Pascal, or PL/I to use the database accessing SQL by means of appropriate preprocessors.

3.7 FURTHER READING

Bipin C. Desai, *An Introduction to Database Management*, Galgotia Publication, New Delhi.

UNIT 4 DISTRIBUTED DATABASES

Structure

4.0 INTRODUCTION

In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with each other through various communication media, such as high-speed buses or telephone line. They do not share main memory, nor do they share a clock.

The processors in a distributed system may vary in size and function. They may include small microcomputers, work station, minicomputers, and large general-purpose computer system. These processors are referred to by a number of different names such as sites, nodes, computers, and so on, depending on the context in which they are mentioned. We mainly use the term site, in order to emphasize the physical distribution of these systems.

A distributed database system consists of a collection of sites, each of which may participate in the execution of transactions which access data at one site, or several sites. The main difference between centralized and distributed database systems is that, in the former, the data resides in one single location, while in the latter, the data resides in several locations. As we shall see, this distribution of data is the cause of many difficulties that will be addressed in this chapter.

4.1 OBJECTIVES

After going through this unit you may able to :

- Differentiate DDBMS and conventional DBMS
- Discuss Network topology for DDBMS
- Discuss advantages and disadvantages of DDBMS
- Distinguish between horizontal and vertial fragmentation.

4.2 STRUCTURE OF DISTRIBUTED DATABASE

A distributed database system consists of a collection of sites, each of which maintains a local databases system. Each site is able to process local transactions, those transactions that access data only in that single site. In addition, a site may participate in the execution of global transactions, those transactions that access data is several sites. The execution of global transactions requires communication among the sites.

The sites in the system can be connected physically in a variety of ways. The various topologies are represented as graphs whose nodes correspond to sites. An edge from node A to node B corresponds to a direct connection between the two sites. Some of the most common configurations are depicted in Figure 1. The major differences among these configurations involve:

- Installation cost. The cost of physically linking the sites in the system.
- Communication cost. The cost in time and money to send a message from site A to site B.
- Reliability. The frequency with which a link or site fails.
- Availability. The degree to which data can be accessed despite the failure of some links or sites.

As we shall see, these differences play an important role in choosing the appropriate mechanism for handling the distribution of data.

The sites of a distributed database system may be distributed physically either over a large geographical area (such as the all Indian states), or over a small geographical area such as a single building or a number of adjacent buildings). The former type of network is referred to as a long-haul network, while the latter is referred to as a local-area network.

Since the sites in long-haul networks are distributed physically over a large geographical area, the communication links are likely to be relatively slow and less reliable as compared with local-area networks. Typical long-haul links are telephone lines, microwave links, and satellite channels. In contrast, since all the sites in local-area networks are close to each other, communication links are of higher speed and lower error rate than their counterparts in long-haul networks. The most common links are twisted pair, baseband coaxial, broadband coaxial, and fiber optics.

Let us illustrate these concepts by considering a banking system consisting of four branches located in four different cities. Each branch has its own computer with a database consisting of all the accounts maintained at that branch. Each such installation is thus a site. There also exists one single site which maintains information about all the branches of the bank. Suppose that the database systems at the various sites are based on the relational model. Thus, each branch maintains (among others) the relation deposite (Deposit-scheme) where

Deposite-scheme = (branch-name, account-number, customer-name, balance)

site containing information about the four branches maintains the relation branch (Branch-scheme), where

Branch-scheme = (branch-name, assets, branch-city)

There are other relations maintained at the various sites which are ignored for the purpose of our example.

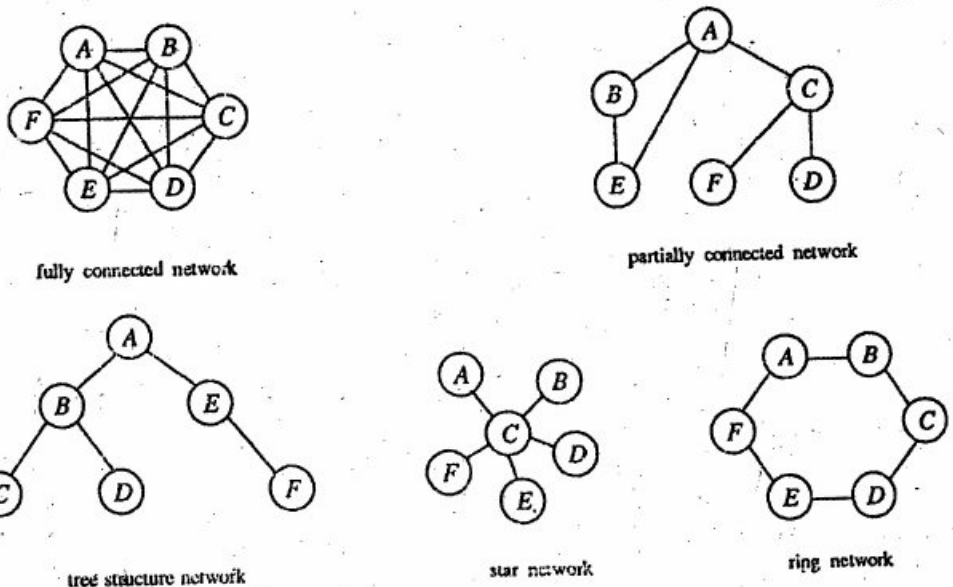


Figure 1 : Network topology

A local transaction is a transaction that accesses accounts in the one single site, at which the transaction was initiated. A global transaction, on the other hand is one which either access accounts in a site different from the one at which the transaction was initiated, or accesses accounts in several different sites. To illustrate the difference between these two types of transactions, consider the transaction to add $ 50 to account number 177 located at the Delhi branch. If the transaction was initiated at the Delhi branch, then it is considered local; otherwise, it is considered global. A transaction to transfer $ 50 from account 177 to account 305, which is located at the Bombay branch, is a global transaction since accounts in two different sites are accessed as a result of its execution.

What makes the above configuration a distributed database system are the facts that:

- The various sites are aware of each other.
- Each site provides an environment for executing both local and global transactions.

4.3 TRADE-OFFS IN DISTRIBUTING THE DATABASE

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speedup of query processing. However, along with these advantages come several disadvantages, including software development cost, greater potential for bugs, and increased processing overhead. In this section, we shall elaborate briefly on each of these.

4.3.1 Advantages of Data Distribution

The primary advantage of distributed database systems is the ability to share and access data in a reliable and efficient manner.

Data Sharing and Distributed Control

If a number of different sites are connected to each other, then a user at one site may be able to access data that is available at another site. For example, in the distributed banking system described in Section 4.2, it is possible for a user in one branch to access data in another branch. Without this capability, a user wishing to transfer funds from one branch to another would have to resort to some external mechanism for such a transfer. This external mechanism would, in effect, be a single centralized database.

The primary advantage to accomplishing data sharing by means of data distribution is that each site is able to retain a degree of control over data stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system. A part of these responsibilities is delegated to the local database administrator for each site. Depending upon the design of the distributed database system, each local administrator may have a different degree of autonomy is often a major advantage of distributed databases.

Reliability and Availability

If one site fails in distributed system, the remaining sites may be able to continue operating. In particular, if data are replicated in several sites, transaction needing a particular data item may find it in several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system.

The failure of one site must be detected by the system, and appropriate action may be needed to recover from the failure. The system must no longer use the service of the failed site. Finally, when the failed site recovers or is repaired, mechanisms must be available to integrate it smoothly back into the system.

Although recovery from failure is more complex in distributed systems than in centralized system, the ability of most of the system to continue to operate despite the failure of one site results in increased availability. Availability is crucial for database systems used for real-time applications. Loss of access to data by, for example, an airline may result in the loss of potential ticket buyers to competitors.

Speedup Query Processing

If a query involves data at several sites, it may be possible to split the query into subqueries that can be executed in parallel by several sites. Such parallel computation allows for faster

processing of a user's query. In those cases in which data is replicated, queries may be directed by the system to the least heavily loaded sites.

4.3.2 Disadvantages of Data Distribution

The primary disadvantage of distributed database systems is the added complexity required to ensure proper coordination among the sites. This increased complexity takes the form of :

- **Software development cost :** It is more difficult to implement a distributed database system and, thus, more costly.
- **Greater potential for bugs :** Since the sites that comprise the distributed system operate in parallel, it is harder to ensure the correctness of algorithms. The potential exists for extremely subtle bugs. The art of constructing distributed algorithms remains an active and important area or research.
- **Increased processing overhead :** The exchange of messages and the additional computation required to achieve intersite coordination is a form of overhead that does not arise in centralized systems.
- In choosing the design for a database system, the designer must balance the advantages against the disadvantages of distribution of data design ranging from fully distributed designs to designs which included large degree of centralization.

4.4 DESIGN OF DISTRIBUTED DATABASES

The principles of database design that we discussed earlier apply to distributed databases as well. In this section, we focus on those design issues that are specific to distributed databases.

Consider a relation that is to be stored in the database. There are several issues involved in storing this relation in the distributed database, including:

- **Replication :** The system maintains several identical replicas (copies) of the relation. Each replica is stored in a different site, resulting in data replication. The alternative to replication is to store only one copy of relation.
- **Fragmentation :** The relation is partitioned into several fragments. Each fragment is stored in a different site.
- **Replication and Fragmentation :** This is a combination of the above two notions. The relation is partitioned into several fragments. The system maintains several identical replicas of each such fragment.

In the following subsections, we elaborate on each of these.

4.4.1 Data 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.

There are a number of advantages and disadvantages to replication.

- **Availability :** If one of the sites containing relation r fails, then the relation r may be found in another site. Thus, the system may continue to process queries involving r despite the failure of one site.
- **Increased parallelism :** In the case where the majority of access to the relation r results in only the reading of the relation, the several sites can process queries involving r in parallel. The more replicas of r there are, the greater the chance that the needed data is found in the site where the transaction is executing. Hence, data replication minimizes movement of data between sites.
- **Increase overhead on update :** The system must ensure that all replicas of a relation r are consistent since otherwise erroneous computations may result. This implies that whenever r is updated, this update must be propagated to all sites containing replicas, resulting in increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary that transactions assure that the balance in a particular account agrees in all sites.

In general, replication enhances the performance of read operations and increases the availability of data to read transactions. However, update transactions incur greater overhead. The problem of controlling concurrent updates by several transactions to replicated data is more complex than the centralized approach to concurrency control. We may simplify the management of replicas of relation r by choosing one of them as the primary copy of r. For example, in a banking system, an account may be associated with the site in which the account has been opened. Similarly, in an airline reservation system, a flight may be associated with the site at which the flight originates.

4.4.2 Data Fragmentation

If the relation r if fragmented, r is divided into a number of fragments $r_1, r_2, ..., r_n$. These fragments contain sufficient information to reconstruct the original relation r. As we shall see, this reconstruction can take place through the application of either the union operation or a special type of join operation on the various fragments. 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 in a special way that we shall discuss. These two schemes can be applied successively to the same relation, resulting in a number of different fragments. Note that some information may appear in several fragments.

Below we discuss the various ways for fragmenting a relation. We shall illustrate these by fragmenting the relation deposit, with scheme:

Deposite-scheme = (branch-name, account-name, customer-name, balance)

The relation deposite (deposite-scheme) is shown in Figure 2.

branch-name	*account number*	*customer-name*	*balance*
Bombay	305	Lowman	500
Bombay	226	Camp	336
Delhi	117	Camp	205
Delhi	402	Khan	10000
Bombay	155	Khan	62
Delhi	408	Khan	1123
Delhi	639	Green	750

Figure 2 : Sample deposite relation

Horizontal Fragmentation

The relation r is partitioned into a number of subsets, $r_1, r_2, ..., r_n$. Each subset consists of a number of tuples of relation r. Each tuple of relation r must belong to one of the fragments, so that the original relation can be reconstructed, if needed.

A fragment may be defined as a selection on the global relation r. That is, a predicate P1 is used to construct fragment ri as follows:

$$r_i = \sigma_{p_i}(r)$$

The reconstruction of the relation r can be obtained by taking the union of all fragments, that is,

$$r = \bigcup_{i=1}^{n} r_i$$

To illustrate this, suppose that the relation r is the deposite relation of Figure 2. This relation can be divided into n different fragments, each of which consists of tuples of accounts belonging to a particular branch. If the

branch-name	*account-number*	*customer-name*	*balance*
Bombay	305	Lowman	500
Bombay	226	Camp	336
Bombay	155	Khan	62

(a)

branch-name	account-number	customer-name	balance
Delhi	177	Camp	205
Delhi	402	Khan	10000
Delhi	408	Khan	1123
Delhi	639	Green	750

(b)

Figure 3 : Horizontal fragmentation of relation of *deposit*

banking system has only two branches, Bombay and Delhi, then there are two different fragments:

$$deposit_1 = \sigma_{branch\text{-}name = \text{"Hillside"}} (deposit)$$
$$deposit_2 = \sigma_{branch\text{-}name = \text{"Valleyview"}} (deposit)$$

these two fragments are shown in Figure 3. Fragment deposit 1 is stored in the Bombay site. Fragment deposit 2 is stored in the Delhi site.

In our example, the fragments are disjoint. By changing the selection predicates used to construction the fregments, we may have a particular tuple of r appear in more than one of the r_1. This is a form of data replication about which we shall say more at the end of this section.

Vertical Fragmentation

In its most simple form, vertical fragmentation is the same as decomposition. Vertical fragmentation of r(R) involves the definition of several subsets $R_1, R_2, ..., R_n$ of R such that $r_i = r$.

Each fragment ri of r is defined by:

$$r_i = \Pi_{R_i} (r)$$

relation r can be reconstructed from the fragments by taking the natural join:

$$r = r_1 \infty r_2 \infty r_3 \infty \ldots \infty r_n$$

branch-name	account-number	customer-name	balance	tuple-id
Bombay	305	Lowman	500	1
Bombay	226	Camp	336	2
Delhi	177	Camp	205	3
Delhi	402	Kahn	10000	4
Bombay	155	Kahan	62	5
Delhi	408	Khan	1123	6
Delhi	639	Green	750	7

Figure 4 : The deposite relation of Figure 4.2 with tuple-ids

More generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-id to the scheme R. A tuple-id is a physical or logical address for a tuple. Since each tupe in r must have a unique address, the tuple-id attribute is a key for the augmented scheme.

In Figure 4, we show the relation deposit', the deposit relation of Figure 2 with tuple-ids added. Figure 5 shows a vertical decomposition of the scheme Deposit-scheme tuple-id into:

Deposit-scheme-3 = (branch-name, customer-name, tuple-id)
Deposit-scheme-4 = (account-number, balance, tuple-id)

The two relation shown in figure 5 result from computing:

$$deposit_3 = \Pi_{Deposit\text{-}scheme\text{-}3} (Deposit')$$
$$deposit_4 = \Pi_{Deposit\text{-}scheme4} (Deposit')$$

branch-name	*customer-name*	*tuple-id*
Bombay	Lowman	1
Bombay	Camp	2
Delhi	Camp	3
Delhi	Khan	4
Bombay	Khan	5
Delhi	Khan	6
Delhi	Green	7

(a)

account-number	*balance*	*tuple-id*
305	500	1
226	336	2
177	205	3
402	10000	4
155	62	5
408	1123	6
639	750	7

(b)

Figure 5 : Vertical fragmentation of relation *deposit*

To reconstruct the original deposit relation from the fragments, we compel

$$\Pi_{Deposit\text{-}scheme}\,(deposit_3 \infty deposit_4)$$

Note that the expression

$$deposit_3 \infty deposit_4$$

is special form of natural join. The join attribute is tuple-id. Since the *tupled-id* value represents an address, it is possible to pair a tuple of deposit 3 with the corresponding tuple of deposit 4 by using the address given by the tuple-id value. This address allows direct retrieval of the tuple without the need for an index. Thus, this natural join may be computed much more efficiently than typical natural joins.

Although the *tuple-id* attribute is important in the implementation of vertical partitioning, it is important that this attribute not be visible to users. If users are given access to tuple-ids, it becomes impossible for the system to change tuple addresses. Furthermore, the accessibility of internal addresses violates the notion of data independence, one of the main virtues of the relational model.

4.5 SUMMARY

A distributed database system consists of a collection of sites, each of which maintain a local database system. Each site is able to process local transaction, those transaction that access data only in that single site. In addition, a site may participate in the execution of global transactions those transactions that access data n several sites. The execution of global transactions requires communication among the sites.

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speed of query processing. However, along with these advantages come several disadvantages, including software development cost, greater potential for bugs, and increased processing overhead. The primary disadvantage of distributed database systems in the added complexity required to ensure proper co-ordination among the sites.

There are several issues involved in storing, a relation in the distributed database, including replication and fragmentation. It is essential that the system minimise the degree to which a user needs to be aware of how a relation is stored.

4.6 FURTHER READING

Henry F. Korth, Abraham Silberschatz, *Database System Concepts*, McGraw Hill International Edition.

Notes

BCA-1.5
Introduction to Database Management Systems

Block

3

Emerging Trends in Database Management Systems

Expert Advisors

Prof. P.S. Grover*
Professor of Computer Science
University of Delhi
Delhi

Brig. V.M. Sundaram
Coordinator
DoE-ACC Centre
New Delhi

Prof. Karmeshu
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Prof. L.M. Patnaik
Indian Institute of Science
Bangalore

Prof. M.M. Pant
Director
School of Computer and
Information Sciences
IGNOU, New Delhi

Dr. S.C. Mehta
Sr. Director
Manpower Development Division
Department of Electronics
Govt. of India
New Delhi

Dr. G. Haider
Director
Information Technology Centre
TCIL, Delhi

Prof. H.M. Gupta
Department of Electrical
Engineering
Indian Institute of Technology
Delhi

Prof. S. Sadagopan
Department of Industrial
Engineering
Indian Institute of Technology
Kanpur

Prof. R.G. Gupta
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Prof. S.K. Wasan
Professor of Computer
Science
Jamia Millia
Delhi

Dr. Sugata Mitra
Principal Scientist
National Institute of
Information Technology
New Delhi

Prof. Sudhir Kaicker
Director
School of Computer and
Systems Sciences
Jawaharlal Nehru University
Delhi

Faculty of the School

Prof. M.M. Pant
Director

Mr. Akshay Kumar
Lecturer

Mr. Shashi Bhushan
Lecturer

Dr. (Mrs.) Bimlesh
Lecturer

Course Preparation Team

Prof. M.M. Pant
Director, SOCIS
IGNOU

Mr. Milind Mahajani
Manager
Information Services
Times of India Group
New Delhi

Dr. N. Parimala
Birla Institute of Technology
and Science, Pilani

Mr. Shashi Bhushan
Lecturer, IGNOU

Block Writers

Utpal Bhattacharya
NIIT
New Delhi

Mr. Shashi Bhushan
Lecturer, IGNOU

Course Coordinator

Mr. Shashi Bhushan
Lecturer, IGNOU

Print Production : Sh. Jitender Sethi, APO, MPDD

March, 2003 (Reprint)

ISBN-81-7263-823-X

BLOCK INTRODUCTION

Since the 1960s, the technology for managing data has evolved from file system to hierarchical, to network, to relational. RDBMSs were originally designed for mainframe computers and business data processing. Many of today's applications are workstation based and involve complex data and operations. For example, computer aided design database require the support of composite objects and different versions of the same objects. A multimedia database may contain variable length text, graphics, images, audio and video data. Finally a knowledge-base requires data rich in Semantics.

During the late 1980s and early 1990s, the image of the classical mainframe computer for computing is decreasing and the trend is towards Client/Server computing.

In this block we have taken up three new emerging topics/disciplines in database. These are Object oriented databases, Client/Server databases and Knowledge databases. In fact there are many more such as Distributed database, Multimedia database, Temporal database, Spatial database, etc. But, we will be taking up these topics in advance course of DBMS to be offered at third year level. In this block there are 3 units:

The first unit introduces object oriented database system. In this unit we have discussed the basic components of OODBMS, how it is different from RDBMS and what are its drawbacks and promises.

The second unit takes up Client/Server database. Apart from talking about basics of Client/Server computing we have also demonstrated how to develop an application in Client/Server environment.

The final unit discusses Knowledge database. The focal points covered in this unit are how a knowledge base system is different from database and what are knowledge representation schemes.

UNIT 1 INTRODUCTION TO OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM

Structure

1.0 INTRODUCTION

Since 1960s, Data Base Management Systems (DBMS) have been widely used in data processing environment. The support of characteristics such as data sharing, independence, consistency, integrity is the main reason for its success which traditional file management system does not inherently offer.

A database system is usually organised according to a data model. In the previous block, we discussed three most popular models: hierarchical, network and relational. The difference among all these three models is in the way of organising records, although they are record based. They were mainly designed to process large amount of relatively simple and fixed format data. DBMS based on these models along with sophisticated indexing and query optimization techniques have served business oriented database application especially well.

RDBMSs were originally designed for mainframe computer and business data processing applications. Moreover, relational systems were optimized for environments with large number of users who issue short queries. But today's application has moved from centralised mainframe computer to networked workstation on every desk. These applications include computer aided design (CAD), multimedia system, software engineering (design of complex project), knowledge database (to be discussed in unit 3 of this block). These operations require complex operations and data structure representation. For example, a multimedia database may contain variable length text, graphics, images, audio and video data. Finaliy a knowledge base system requires data rich in semantics.

Existing commercial DBMS, both small and large scale have proven inadequate for these applications. The traditional database notion of storing data in two-dimensional tables or in flat files breaks down quickly in the face of complex data structures and data types used in today's applications.

Research to model and process complex data has gone in two directions:

(a) extending the functionality of RDBMS

(b) developing and implementing OODBMS that is based on object oriented programming paradigm.

OODBMSs are designed for use in today's application areas such as multimedia, CAD, office automation, etc. In this unit, we will touch up some of the basic issues related to OODBMS.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- define what is object oriented DBMS
- differentiate between RDBMS and OODBMS
- list next generation database systems
- list advantages of object oriented DBMS

1.2 WHAT ARE NEXT GENERATION DATABASE SYSTEM?

Computer sciences has gone through several generation of database management starting with indexed files and later, network and hierarchical data base management systems (DBMS). More recently, relational DBMS revolutionalised the industry by providing powerful data management capabilities based on few simple concepts. Now, we are on the verge of another generation of database system called **Object Oriented DBMS** based on object oriented programming paradigm. This new kind of DBMS, unlike previous DBMS models, manage more complex kind of data for example multimedia objects. The other kind of next generation DBMS is knowledge database management system (KDBMS) which is used to support the management of the shared knowledge. It supports a large number of complex rules for automatic data inferencing (retrieval) and maintenance of data integrity.

The goal of these new DBMS is to support a much wider range of data intensive applications in engineering, graphic representation—scientific and medical.These new DBMS can also support new generations of traditional business applications.

1.3 NEW DATABASE APPLICATIONS

Some applications that require the manipulation of large amounts of data can benefit from using a DBMS. However, the nature of the data in these applications does not fit well into the relational framework.

(1) Design databases : Engineering design databases are useful in computer-aided design/manufacturing/software engineering (CAD/CAM/CASE) systems. In such systems, complex objects can be recursively partitioned into smaller objects. Furthermore, an object can have different representations at different levels of abstraction (equivalent objects). Moreover, a record of an object's evolution (object versions) should be maintained. Traditional database technology does not support the notions of complex objects, equivalent objects, or object versions.

(2) Multimedia databases : In a modern office information or other multi-media system, data include not only text and numbers but also images, graphics and digital audio and video. Such multimedia data are typically stored as sequences of bytes with variable lengths, and segments of data are linked together for easy reference. The variable length data structure cannot fit well into the relational framework, which mainly deals with fixed-format records. Furthermore, applications may require access to multimedia data on the basis of the structure of a graphical item or by following logical links. Conventional query languages were not designed for such applications.

(3) Knowledge bases : Artificial intelligence and expert systems represent information as facts and rules that can be collectively viewed as a knowledge base. In typical Artificial Intelligence applications, knowledge representation requires data structures with rich semantics that go beyond the simple structure of the relational model. Artificial decomposition and mapping would be necessary if a relational DBMS were used. Furthermore, operations in a knowledge base are more complex than those in a traditional database. When a rule is added, the system must check for contradiction and redundancy. Such operations cannot be represented directly by relational operations, and the complexity of checking increases rapidly as the size of the knowledge base grows.

In general, these applications require the representation of complex data elements as well as complex relationships among them. Users in these environments have found relational technology inadequate in terms of flexibility, modelling power, and efficiency.

1.4 WHAT IS OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM ?

Object-oriented technologies in use today include object- oriented programming languages (e.g., C++ and Smalltalk), object- oriented database systems, object-oriented user interfaces (e.g., Macintosh and Microsoft Windows systems) and so on. An object-oriented technology is a technology that makes available to the users facilities that are based on object-oriented concepts. To define object-oriented concepts, we must first understand what an **object** is.

Object

The term object means a combination of data and program that represents some real-world entity. For example, consider an employee named Amit; Amit is 25 years old, and his salary is $25,000. Then Amit may be represented in a computer program as an object. The data part of this object would be (name: Amit, age: 25, salary: $25,000). The program part of the object may be a collection of programs (hire, retrieve the data, change age, change salary, fire). The data part consists of data of any type. For the **Amit** object, string is used for the name, integer for age, and monetary for salary; but in general, even any user-defined type, such as Employee, may be used. In the **Amit** object, the name, age, and salary are called attributes of the object.

Encapsulation

Often, an object is said to **encapsulate** data and program. **This means that the users cannot see the inside the object but can use the object by calling the program part of the object.** This is not much different from **procedure calls** in conventional programming; the users call a procedure by supplying values for input parameters and receive results in output parameters.

Inheritance and Class

The term **object-oriented** roughly means a combination of object encapsulation and inheritance. The term inheritance is sometimes called **reuse**. Inheritance means roughly that a new object may be created by extending an existing object. Now let us understand the term inheritance more precisely. An object has a data part and a program part. **All objects that have the same attributes for the data part and same program part are collectively called a class (or type).** The classes are arranged such that some class may inherit the attributes and program part from some other classes.

Amit, Ankit and Anup are each an Employee object. The data part of each of these objects consists of the attributes Name, Age and salary. Each of these Employee objects has the same program part (hire, retrieve the data, change age, change salary, fire). **Each program in the program part is called a method.** The term class refers to the collection of all objects that have the same attributes and methods. In our example, the Amit, Ankit and Anup objects belong to the class Employee since they all have the same attributes and methods. The class may be used as the type of an attribute of any object. At this time, there is only one class in the system namely, the class Employee; and three objects that belong to the class namely Amit, Ankit and Anup objects.

Inheritance Hierarchy or Class Hierarchy

Now suppose that a user wishes to create two sales employees, Jai and Prakash. But shod employees have an additional attribute namely, **commission.** The sales employees came belong to the class Employee. However, the user can create a new class, Sales Employes such that all attributes and methods associated with the class Employee may be reused and the attribute commission may be added to Sales Employee. The user does this by deeking the class Sales-Employee to be a subclass of the class Employee. The user can now prodeed to create the two sales employees as objects belonging to the class Sale Employee. Two users can create new classes as subclasses of existing classes. **In general**, a class may **inherit from one or more existing classes** and the inheritance structures of classes becomes a directed acyclic graph (DAG); but for simplicity, the inheritance structes is called an **inheritance hierarchy or class hierarchy.**

The power of object-oriented concepts is delivered when encapsulation and inheritance work together.

- Since inheritance makes it possible for different classes to share the same set of attributes and methods, the same program can be run against objects that belong to different classes. This is the basis of the object-oriented user interface that desktop publishing systems and windows management systems provide today. The same set of programs (e.g., open, close, drop, create, move, etc.) apply to different types of data (image, text file, audio, directory, etc.).
- If the users define many classes, and each class has many attributes and methods, the benefit of sharing not only the attributes but also the programs can be dramatic. The attributes and programs need not be defined and written from scratch. New classes can be created by adding attributes and methods of existing classes, thereby reducing the opportunity to introduce new errors to existing classes.

1.5 PROMISES OF OBJECT ORIENTED SYSTEMS

Object-oriented systems make these promises:

- **Reduced maintenance**
  The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviours may be reused and incorporated into new behaviours.
- **Real-world modelling**
  Object-oriented systems tend to model the real world in a more complete fashion than do traditional methods. Objects are organised into classes of objects, and objects are associated with behaviours. The model is based on objects rather than on data and processing.
- **Improved reliability**
  Object-oriented systems promise to be far more reliable than traditional systems, primarily because new behaviours can be built from existing objects.
- **High code reusability**
  When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviours from all superclasses in which it participates.

1.6 PROMISES AND ADVANTAGES OF OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM

An object-oriented programming language (OOPL) provides facilities to create classes for organising objects, to create objects, to structure an inheritance hierarchy to organise classes so that subclasses may inherit attributes and methods from superclasses, and to call methods to access specific objects. Similarly, an object-oriented database system (OODB) should provide facilities to create classes for organising objects, to create objects, to structure an inheritance hierarchy to organise classes so that subclasses may inherit attributes and methods from superclasses, and to call methods to access specific objects. Beyond these, an OODB, because it is a database system, must provide standard database facilities found in today's relational database systems (RDBs), including nonprocedural query facility for retrieving objects, automatic query optimisation and processing, **dynamic schema changes** (changing the class definitions and inheritance structure), **automatic management of access methods** (e.g., B+-tree index, extensible hashing, sorting, etc.) to improve **query processing performance, automatic transaction management, concurrency control, recovery from system crashes, and security and authorisation**, Programming languages, including OOPLs, are designed with one user and a relatively small database in mind. Database systems are designed with many users and very large databases in mind; hence performance, security and authorisation, concurrency control, and dynamic schema changes become important issues. Further, transaction systems are used to maintain critical data accurately; hence, transaction management, concurrency control, and recovery are important facilities.

In so far as a database system is a system software, whose functions are called from application programs written in some host programming languages, we may distinguish two different approaches to designing an OODB. One is to store and manage objects created by programs written in an OOPL. Some of the current OODBs are designed to store and manage objects generated in C++ or Smalltalk programs. Of course, an RDB can be used to store and manage such objects. However, RDBs do not understand objects—in particular, methods and inheritance. Therefore, what may be called an object manager or an object-oriented layer software needs to written to be manage methods and inheritance and to translate objects to tuples (rows) of a relation (table). But the object manager and RDB combined are in effect an OODB (with poor performance, of course).

Another approach is to make object-oriented facilities available to users of non-OOPLs. The users may create classes, objects, inheritance hierarchy, and so on, and the database system will store and manage those objects and classes. This approach in effect turns non-OOPLs (e.g., C, FORTRAN, COBOL, etc.) into object-oriented languages. In fact, C++ has turned C into an OOPL, and CLOS has added object-oriented programming facilities to Common LISP. An OODB designed using this approach can of course be used to store and manage objects created by programs written in an OOPL. Although a translation layer would need to be written to map the OOPL objects to objects of the database system, the layer should be much less complicated than the object manager layer that an RDB would require.

In view of the fact that C++, despite its growing popularity, is not the only programming language that database application programmers are using or will ever use, and there is a significant gulf between a programming language and a database system that will deliver the power of object-oriented concepts to database application programmers. Regardless of the approach, OODBs, if done right, can bring about a quantum jump in the productivity of database application programmers and even in the performance of the application programs.

One source of the technological quantum jump is the reuse of a database design and program that object-oriented concepts make possible for the first time in the evolving history of database technologies. Object-oriented concepts are fundamentally designed to reduce the difficulty of developing and evolving complex software systems or designs. Encapsulation and inheritance allow attributes (i.e., database design) and programs to be reused as the basis for building complex databases and programs. This is precisely the goal that has driven the data management technology from file systems to relational database systems during the past three decades. An OODB has the potential to satisfy the objective of reducing the difficulty of designing and evolving very large and complex databases.

Another source of the technological jump is the powerful data type facilities implicit in the object-oriented concepts of encapsulation and inheritance.

Advantages of Object-Oriented Databases

Systems developed with object-oriented languages have many benefits, as previously discussed. Yet, as also described, these systems have particular attributes that can be complemented with object-oriented databases. These attributes include lack of persistence, inability to share objects among multiple users, limited version control, and lack of access to other data, for example, data in other databases.

In systems designed with object-oriented languages, objects are created during the running of a program and are destroyed when the program ends. Providing a database that can store the objects between runs of a program offers both increased flexibility and increased security. The ability to store the objects also allows the objects to be shared in a distributed environment. An object-oriented database can allow only the actively used objects to be loaded into memory and thus minimizes or preempts the need for virtual memory paging. This is especially useful in large-scale systems. Persistent objects also allow objects to be stored for each version. This version control is useful not only for testing applications, but also for many object-oriented design applications where version control is a functional requirement of the application itself. Access to other data sources can also be facilitated with object-oriented databases, especially those built as hybrid relational systems, which can access relational tables as well as other object types.

Object-oriented databases also offer many of the benefits that were formerly found only in expert systems. With an object-oriented database, the relationships between objects and the constraints on objects are maintained by the database management system, that is, the objects themselves. The rules associated with the expert system are essentially replaced by the object schema and the methods. As many expert systems currently do not have adequate

database support, object-oriented databases afford the possibility of offering expert system functionality with much better performance.

Object-oriented databases offer benefits over current hierarchical and relational database models. They enable support of complex applications not supported well by the other models. They enhance programmability and performance, improve navigational access, and simplify concurrency control. They lower the risks associated with **referential integrity**, and they provide a better user metaphor than the relational model.

Object-oriented databases by definition allow the inclusion of more of the code (i.e. the object's methods) in the database itself. This incremental knowledge about the application has a number of potential benefits of the database system itself, including the ability to optimize query processing and to control the concurrent execution of transactions.

Performance, always a significant issue in system implementation, may be significantly improved by using an object-oriented model instead of a relational model. The greatest improvement can be expected in applications with high data complexity and large numbers of inter-relationships. Clustering, or locating the related objects in close proximity, can be accomplished through the class hierarchy or by other interrelations. Caching, or the retention of certain objects in memory or storage, can be optimised by anticipating that the user or application may retrieve a particular instance of the class. When there is high data complexity, clustering and caching techniques in object databases gain tremendous performance benefits that relational databases, because of their fundamental architecture, will never be able to approach.

Object-oriented databases can store not only complex application components but also larger structures. Although relational systems can support a large number of tuples (i.e. rows in a table), individual types are limited in size. Object-oriented databases with large objects do not suffer a performance degradation because the objects do not need to be broken apart and reassembled by applications, regardless of the complexity of the properties of the application objects.

Since objects contain direct references to other objects, complex data set can be efficiently assembled using these direct references. The ability to search by direct references significantly improves navigational access. In contrast, complex data sets in relational databases must be assembled by the application program using the slow process of joining tables.

For the programmer, one of the challenges in building a database is the data manipulation language (DML) of the database. DMLs for relational databases usually differ from the programming language used to construct the rest of the application. This contrast is due to differences in the programming paradigms and mismatches of type systems. The programmer must learn two languages, two tool sets, and two paradigms because neither alone has the functionality to build an entire application. Certain types of programming tools, such as application generators and fourth-generation languages (4GLs) have emerged to produce code for the entire application, thereby bridging the mismatch between the programming language and the DML, but most of these tools compromise the application programming process.

With object-oriented databases much of this problem is eliminated. The DML can be extended so that more of the application can be written in the DML. Or an object-oriented application language, of example C++ can be extended to be the DML. More or the application can be built into the database itself. Movement across the programming interface between the database the application then occurs in a single paradigm with a common set of tools. Class libraries can also assist the programmer in speeding the creation of databases. Class libraries encourage reuse of existing code and help to minimise the cost of later modifications. Programming is easier because the data structures model the problem more closely. Having the data and procedures encapsulated in a single object makes it less likely that a change to one object will affect the integrity of other objects in the database. Concurrency control is also simplified with an object-oriented database. In a relational database, the application needs to lock each record in each table explicitly because related data re-represented across a number of tables. Integrity, a key requirement for databases, can be better supported with an object-oriented database, because the application can lock all the relevant data in one operation. Referential integrity is better supported in an object-oriented database because the pointers are maintained and updated by the database itself. Finally, object-oriented databases offer a better user metaphor than relational databases. The tuple or table, although enabling a well-defined implementation strategy, is not an intuitive modelling

framework, especially outside the domain of numbers. Objects offer a more natural and encompassing modelling metaphor.

1.7 DEFICIENCIES OF RELATIONAL DATA BASE MANAGEMENT SYSTEM

The data type facilities in fact are the keys to eliminating three of the important deficiencies of RDBs. These are summarized below, we will discuss these points in greater detail later.

- RDBs force the users to represent hierarchical data (or complex nested data or compound data) such as bill of materials in terms of tuples in multiple relations. This is awkward to start with. Further, to retrieve data thus spread out in multiple relations. RDBs must resort to joins, a generally expensive operation. The data type of an attribute of an object in OOPLs may be a primitive type or an arbitrary user-defined type (class). The fact that an object may have an attribute whose value may be another object naturally leads to nested object representation, which in turn allow hierarchical data to be naturally (i.e., hierarchically) represented.

- RDBs offer a set of primitive built-in data types for use as domains of columns of relation, but they do not offer any means of adding user-defined data types. The built-in data types are basically all numbers and short symbols. RDBs are not designed to allow new data types to be added and therefore often require major surgery to the system architecture and code to add any new data type. Adding a new data type to a database system means allowing its use as the data type of an attribute—that is, storage of data of that type, querying, and updating of such data. Object encapsulation in OOPLs does not impose any restriction on the types of data that the data may be primitive types or user-defined types. Further, new data types may be created as new classes, possibly even as subclasses of existing classes, inheriting their attributes and methods.

- Object encapsulation is the basis for the storage and management of programs as well as data in the database. RDBs now support stored procedures—that is, they allow programs to be written in some procedural language and stored in the database for later loading and execution. However, the stored procedures in RDBs are not encapsulated with data—that is, they are not associated with any relation or any tuple of a relation. Further, since RDBs do not have the inheritance mechanism, the stored procedures cannot automatically be reused.

1.8 DIFFERENCE BETWEEN RELATIONAL DATABASE MANAGEMENT SYSTEM AND OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM

RDBMSs were never designed to allow for the nested structure. These types of applications are extensively found in CAD/CAE, aerospace, etc. OODBM can easily support these applications. Moreover, it is much easier and natural to navigate through these complex structures in form of objects that model the real world in OODBMS rather than table, tuples and records in RDBMS.

It is hard to confuse a relational database with an object-oriented database. The normalised relational model is based on a fairly elegant mathematical theory. Relational databases derive a virtual structure at run time based on values from sets of data stored in tables. Databases construct views of the data by selecting data from multiple tables and loading it into a single table (OODBs traverse the data from object to object).

Relational databases have a limited number of simple, built-in data types, such as **integer** and **string**, and a limited number of built-in operations that can handle these data types. You can create complex data types in a relational database, but you must do it on a linear basis, such as combining fields into records. And the operations on these new complex types are restricted, again, to those defined for the basic types (as opposed to arbitrary data types or **subclassing** with inheritance as found in OODBs).

The object model supports browsing of object class libraries, which allows the reuse, rather

than the reinvention, of commonly used data elements. Objects in an OODB survive multiple sessions; they are persistent. If you delete an object stored in a relational database, other objects may be left with references to the deleted one and may now be incorrect. The integrity of the data thus becomes suspect and creates inconsistent versions.

In the relational database, complex objects must be broken up and stored in separate tables. This can only be done in a sequential procedure with the next retrieval replying on the outcome of the previous. The relational database does not understand a global request and thus cannot optimise multiple requests, OODBs can issue a single message (request) that contains multiple transactions.

The relational model, however, suffers at least one major disadvantage. It is difficult to express the semantics of complex objects with only a table model for data storage. Although relational databases are adequate for accounting or other typical transaction-processing applications where the data types are simple and few in number, the relational model offers limited help when data types become numerous and complex.

Object-oriented databases are favoured for applications where the relationships among elements in the database carry the key information. Relational databases are favoured when the values of the database elements carry the key information. That is, object-oriented models capture the structure of the data; relational models organise the data itself. If a record can be understood in isolation, then the relational database is probably suitable. If a record makes sense only in the context of other records, then an object-oriented database is more appropriate.

Engineering and technical applications were the first applications to require databases that handle complex data types and capture the structure of the data. Applications such as mechanical and electrical computer-aided design (MCAD and ECAD) have always used nontraditional forms of data, representing such phenomena as three-dimensional images and VLSI circuit designs. Currently these application programs store their data in application-specific file structures. The data-intensiveness of these applications is not only in the large amount of data that need to be programmed into the database, but in the complexity of the data itself. In these design-based applications, relationships among elements in the database carry key information for the user. Functional requirements for complex cross references, structural dependences, and version management all require a richer representation than what is provided by hierarchical or relational databases.

Check Your Progress

1. What are the drawbacks of current commercial databases?

.....

.....

.....

.....

2. What is the meaning of multi-media data?

.....

.....

.....

.....

.....

3. List few requirements for multi-media data management.

.....

.....

.....

.....

.....

1.9 ALTERNATIVE OBJECT-ORIENTED DATABASE STRATEGIES

There are at least six approaches for incorporating object orientation capabilities in databases:

1. **Novel database data model/data language approach :** The most aggressive approach is to develop entirely new database language and database management system with object orientation capabilities. Most of the research project in object-oriented databases have pursued this approach. In the industry introduces novel DML (Data Manipulation Language) and DDL (Data Definition Language) constructs for a data model based on semantic and functional data models.

2. **Extending an existing database language with object orientation capabilities :** A number of programming languages have been extended with object-oriented constructs. C++ flavors (an extension of LISP), and Object Pascal are examples of this approach in programming languages. It is conceivable to follow a similar strategy with database languages. Since SQL is a standard and the most popular database language, the most reasonable solution is to extend this language with object-oriented constructs, reflecting the object orientation capabilities of the underlying database management system. This approach being pursued by most vendors of relational systems, as they evolve the next generation products. There have been many such attempts incorporating inheritance, function composition for nested entities, and even some support of encapsulation in an SQL framework.

3. **Extending an existing object-oriented programming language with database capabilities :** Another approach is to introduce database capabilities to an existing object-oriented language. The object orientation features abstract data typing, inheritance, object identity—will already be supported by the object-oriented language. The extensions will incorporate database features (querying, transaction support, persistence, and so on).

4. **Embedding object-oriented database language constructs in a host (conventional) language :** Database languages can be embedded in host programming languages. For example, SQL statements can be embedded in PL/I C, FORTRAN and Ada. The types of SQL (that is relations and rows in relations) are quite different from the type systems of these host languages. Some object-oriented databases have taken a similar approach with a host language and an object-oriented database language.

1.10 SUMMARY

During the past decade, object oriented technology has found its way into database user interface, operating system, programming languages, expert system and the like. Object Oriented database product is already in the market for several years and several vendors of RDBMS are now declaring that they will extend their products with object oriented capabilities. In spite of all these claims there is no wide acceptability of OODBMS because of lack of industry standard. This technology is still evolving and take some more time to get fully settled.

1.11 MODEL ANSWERS

Check Your Progress

1. Most of the current commercial database systems suffer from an inability to manage arbitrary types of data, arbitrary large data and data stored on devices other than magnetic disks. They understand a relatively limited set of data types such as integer, real data, monetary unit, short strings. Further they are not designed to manage data stored on such increasingly important storage devices such as CD-ROM and Videodisks.

2. Broadly, multimedia data means arbitrary data types and data from arbitrary data sources. Arbitrary data types include the numeric data and short string data supported in conventional database systems; large unstructured data, such as charts, graphs, tables, and arrays; and compound documents that are comprised of such data. Arbitrary data

sources include a native database; external (remote) databases; host file base; data input, storage, and presentation (output) devices; and even data-generating and data-consuming programs (such as a text processing system).

3a) The ability to represent arbitrary data types (including compound documents) and specification of procedures (programs) that interact with arbitrary data sources.

b) The ability to query, update, insert and delete multimedia data (including retrieval of multimedia data via associative search within multimedia data; minimally, text).

c) The ability to specify and execute abstract operations on multimedia data; for example, to play, fast forward, pause, and rewind such one-dimensional data as audio and text; to display, expand and condense such two-dimensional data as a bit-mapped image.

d) The ability to deal with heterogeneous data sources in a uniform manner; this includes access to data in these sources and migration of data from one data source to another.

1.12 FURTHER READINGS

1. Modern Database Systems—the Object Model , Interoperability and Beyond, By WON KIM, Addison Wesley, 1995.

2. Object-Oriented DBMS : Evolution & Perform ance Issues, A.R.Hurson & Simin H. Pakzad, IEEE Computer, Feb. 1993.

UNIT 2 INTRODUCTION TO CLIENT/ SERVER DATABASE

Structure

2.0 INTRODUCTION

The concept behind the Client/Server solution is concurrent, cooperative processing. It is an approach, that presents a single systems view from a user's viewpoint, involves processing on multiple, interconnected machines provides coordination of activities in a manner transparent to end users.

This unit is broadly divided into three parts. The first part (sections 2.2 and 2.3) address as the basics of client server computing. The second part (section 2.4) discusses the critical products used in implementing client/server model. The focal point of the last part is to develop an application in client server environment.

2.1 OBJECTIVES

At the end of this course, the reader should be able to understand

- the broad level issues in Client' Server computing
- the product components of Client' Server architecture
- how to develop application in Client' Server model
- discuss the possible emerging scenario in Client' Server computing.

2.2 EVOLUTION OF CLIENT/SERVER

Mainframe Scenario

After twenty years of existence starting in the middle of seventies, the computer based application proliferated the business and scientific application throughout the world. The

scenario was dominated by mainframe computers. The development of hardware has always outplaced the development of software yet user requirements did outgrow the capacity of the mainframe computers coupled with the fact that better hardware releases were being launched at rapid succession. The large EDP houses typically opted for the better version of hardware every alternate year. The new model of the hardware would take over the major share of applications rendering the earlier model unutilized or underutilized.

PCs as Environment for Business Computing

In the late seventies first version of PC with 64 KB of main memory were launched, they were typically used to do wordprocessing jobs and spreadsheet calculations. In 1980, IBM launched its 640 KB PC, this is single most important development in the field of computers, which revolutionised the concept of computing profoundly. In 1980 people did not think that the PCs could really become a serious computing environment because of the advancements of technologies in many other related fields. In the decade of 80s PCs grew in power and speed in leaps and bounds. Because of the standard environment and non-proprietary architecture and also because of the very low price tag PCs and software that runs on PC spreaded at a rate which has never been witnessed in field of Information Technology.

Emergence of Open Systems

Till the middle of 70s for over two and half decades proprietary networking solutions dominated the networking scenario. The solutions used to be very expensive, each company used to set its own networking and connectivity standards. Each company believed in giving the complete network, software and networking solution to the end client rendering the solutions extremely high priced and beyond the budget of most Information Technology organisations. Further this did not enable sharing machines from multiple vendors on a network. The advent of non-proprietary standards in network and software product components allowed increasing use of open system. The chip, the peripherals, the architecture, the networking protocols, the operating system even the software components became standard. These developments allowed growth of non-proprietary solution, network solution involving network and software components from multiple vendors. This also enabled usage of downsized environments. PC users grew to upsized environment with Novell netware among other software as the servers.

In 80s networking of PCs in local area network (LAN) or connectivity between PCs running between DOS and VAX running VMS and PCs, connectivity between almost all standard machines gave rise to a new way of looking at computing. The R & D labs dealing with software started working on solution which would distribute the computing load on multiple machines on network. Client/Server architecture took birth around these developments. It basically tries to utilize and distribute computing requirements on PCs, UNIX servers, VMS servers, even mainframe depending on the computing requirements.

2.3 EMERGENCE OF CLIENT' SERVER ARCHITECTURE

Some of the pioneering work that was done by some of the relational database vendors allowed the computing to be distributed on multiple computers on network using contemporary technologies involving:

- Low Cost, High Performance PCs and Servers
- Graphical User Interfaces
- Open Systems
- Object-Orientation
- Workgroup Computing
- EDI and E-Mail
- Relational Databases
- Networking and Data Communication

2.4 THE CLIENT/SERVER COMPUTING

In this application we will take up basics of Client/Server model: how to define client and

server, objectives of client/server model, the difference between mainframe based computing and client/server computing etc.

2.4.1 Basics of Client/Server Computing Paradigm

A Client is an application that initiates peer to peer communication and users usually involve client software when they use a network service. Most client software consists of conventional application programs. Each time a client application executes, it contact a server, sends a request and awaits a response. When the response arrives, the client continues processing. Clients are often easier to build than servers and usually require no special system privileges to operate.

By comparison, a server is any program that provides services to requesting processes in client. It waits for incoming communication requests from a client. It receives a client's request, perhaps the necessary computation and returns the result to the client.

Generally, it does not send information to the requester until the requesting process tells it to do so. But the server must also manage sychronisation of services as well as communication once a request has been initiated.

The client may initiate a transaction with the server, while normally the server does not initiate a transaction with the client.

The client is therefore the more active partner in this association, requesting specific functions, accepting corresponding results from the server, and acknowledging the completion of services. The client, however, does not manage the synchronization of services and associated communication. Because servers often need to access data, server software usually requires special system privileges. Because a server executes with special system privilege, care must be taken to ensure that it does not inadvertently pass privileges on to the clients that use it. For example, a file server that operates as a privileged program must contain code to check whether a given file can be accessed by a given client, the server cannot rely on the usual operating system because its privileged status overrides them. Servers must contain code that handle the issue of:

- Authentication — Verifying the identity of a client
- Authorisation — Determining whether a given client is permitted to access the service by the server supplies.
- Data Security — Generating that data is not unintentionally revealed or compromised.
- Privacy — Keeping information about an individual from unauthorised access.
- Protection — Guaranteeing that network application cannot abuse system resources.

The general case of client/server implementation is shown in figure.

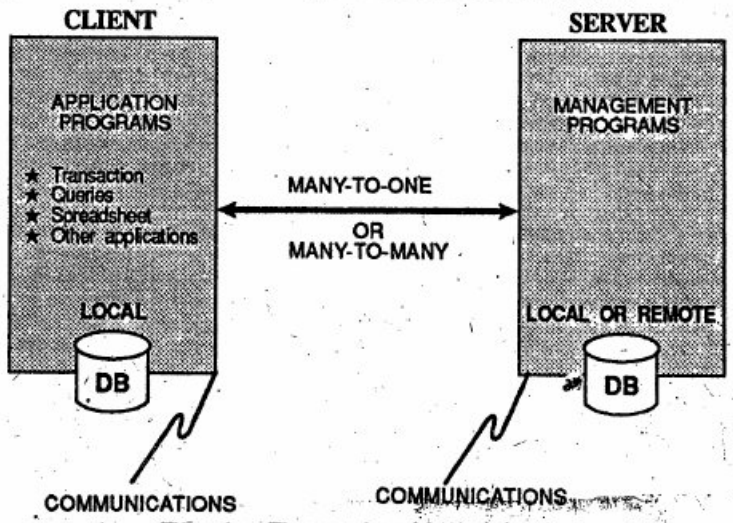


Figure 1 : The general case of client/server implementation

With the client/server computing model of distributed operations

- Many clients may share one server.
- The same client may access many servers, both logical and remote.

The client, the server, or both can be a workstation. The server can also be a supermicro, a database computer, or a supercomputer. Purposeful old minis and obsolete mainframes have not been included in the definition.

Since server-based supports can be complex, the term server does not necessarily refer to a piece of hardware, a database unit, a gateway, or a special-purpose processor dedicated to run software. The concept is much broader: Server requires both software and hardware for a range of functions—though typically each server is specialised.

The technical solutions should be able to assure networkwide shared access by applications processes that address databases, number printers, gateways, and other resources. The environment can be:

- Simple, such as a small work group sharing applications and peripherals

or

- Complex, resulting from wider sharing across multiple systems and topologies

Whether the solution we are after is simple or complex, totally new or a conversion of mainframe applications, a basic design principle is never to build a system to support the current organisational divisions and their departments. A great deal of the necessary flexibility can be provided by solutions that are modular and independent of current structures.

A different way of making this statement is that the information technology (IT) solutions we develop must be organisation-independent:

- They should not reflect the current organisation chart.
- They should be immune to structural changes—hence flexible.
- They should integrate with existing resources, assuring the end user of seamless access to them.

The goal of the solutions through client/server computing is to enable a client program anywhere in a network to request services from anywhere else in the network in a way that is both transparent and independent of any particular interconnected software and hardware. This reference raises a number of questions:

- What computers, communications, and software solutions should be used to create a client/server network ?
- What range of functionality should be targeted to optimise cost-effectiveness ?
- How can the system be kept flexible to develop new applications, add new users, and enhance its own structure ?
- How can we ensure that server platforms and client stations will deliver the high availability demanded by the most competitive applications ?
- What facilities will be able to manage all the nodes and links in the client/server network, distributing the software and controlling the physical assets ?

Not all approaches that represent themselves as client/server models can answer these questions in an able manner. Without any doubt, exceptions should be made of mainframes and nonintelligent terminals connected to mainframes or minis.

Another major exception is the stand-alone workstation. Under no stretch of the imagination can it qualify as a client/server model, although some vendors try to sell it as such. A PC or any other unit that is not networked is not a workstation.

Integrating what has been said so far, we are converging toward definition that **client/servers are excellent multiuser systems with a flexible but all defined applications perspective.** These applications must be designed to work together through adherence to rules.

The foregoing concepts are not necessarily new. To a substantial extent, they have existed for four decades in computing. What is new is the truly peer-to-peer structure of the implementation environment.

The wider acceptance of the outlined solutions largely depends on the functionality provided by the system as a whole. **What makes the client/server architecture distinct from mainframe-based processes are its distributed but cooperative applications characteristics:**

- Clients and servers function across platforms within the network, whether in a local or in a wide area.
- Distributed software artifacts execute on multiple platforms within the supported architecture.
- Processes on the network can be dynamically distributed to the most appropriate (and available) platform for execution.

Graphics applications can be assisted through graphics processors. A numerically intensive process within an application can be migrated from a client to the network's number cruncher server, and a complex database query may access a different database server if the information elements it requires are themselves distributed. This emphasises the need for first- class solutions in networking.

2.4.2 Why need Client/Server Computing?

Client/Server (C/S) architecture involves running the application on multiple machines in which each machine with its component software handles only a part of the job. Client machine is basically a PC or a workstation that provides presentation services and the appropriate computing, connectivity and interfaces while the server machine provides database services, connectivity and computing services to multiple users. Both client machines and server machines are connected to the same network. As the users grow more client machines can be added to the network while as the load on the database machine increases more servers can be connected to the network. The client could be character terminals or GUI PCs or workstations connected to the network. Server machines are slightly more heavy duty machines which gives database services to the client requests.

The network need not be Local Area Network (LAN) only, it can be on much wider distributed Wide Area Network (WAN) across multiple cities. The client and server machines communicate through standard application program interfaces (API) and remote procedure calls (RPC). The language through which RDBMS based C/S environment communicate is known as structured query language (SQL).

2.4.3 Advantages of Client/Server Computing

C/S computing caters to low cost and user friendly environment. It can be used to develop highly complex multiuser database application being handled by any mainframe computers until about 5 years back. It offers expandability. It ensures that the performance degradation is not so much with increased load. It allows connectivity with the heterogeneous machines and also with real time date feeders like ATMs, Numerical machines. It allows the database management including security, performance, backup, server enforced integrity to be part of the database machine avoiding the requirement to write large number of redundant piece of code dealing with database field validation and referential integrity. Since PCs can be used as clients, the application can be connected to the spreadsheets and other applications through Dynamic Data Exchange (DDE) and Object Linking and Embedding (OLE). If the load on database machine grows, the same application can be run slightly upgraded machine like disk machine provided it offers the same version of RDBMSs on diverse machines, legacy applications on old machines or geographically separated can meet all requirements of an enterprise.

2.4.4 Components of Client/Server Computing

The Server

The server machines could be running NOVELL LAN or INTEL based server or UNIX from SCO or AT&T or UNIX being run on RISC machines like HP, SUN Microsystems, IBM, Compaq etc.

These server machines should be running on RDBMS engine like Sybase, Oracle, Informix etc. The server machine takes care of data storage, backup, recovery and database services. It is typically a multiuser machine catering to large number of requests submitted from the client machine or executing requests for RPCs/Stored procedures. The database engine executes the requests and sends the result to the client machine and the presentation service of the client machine puts the received data in required format. Some of the databases take care of the file handling, the task and user handling themselves. The server also allows certain constraints at table level or field level to be incorporated. The field level validations are generally called rules e.g. if employee code is 4 chars, all numeric starting with digit other than 0. In employee table this constraint can be attached to the field itself. It would eliminate writing a code for field validation on this field in each table. If the existence of the employee code is to be checked before entering employee pay details for a month, it would involve two tables : employee pay detail and employee master. These types of checks are called referential integrity constraints. If these constraints can be incorporated in the database, then we can reduce large number of application code. More than that, it will be ensured that application errors do not effect the integrity/reliability of the data stored in the database.

These types of integrity checks are called **Database Triggers**. The advanced RDBMSs also allow on-line database backup, schema modification and performance and tuning. Database stored procedures are certain more repetitively executed pieces of code stored in the database itself, written in the extended form of SQL called T-SQL in Sybase. PL/SQL in Oracle 7. These fast and compiled server resident procedures improve performance by reducing network traffic and by allowing a single copy of the procedure to large number of users. Stored procedures can be executed by client machine. Remote procedure calls on the contrary are generally invoked by servers which enables distributed database processing when the information is available on multiple servers. RPCs can handle the situation very efficiently. In the earlier versions of RDBMSs process for users design was followed, so number of processes on a machine would be directly proportional to the number of users currently using the RDBMS. This resulted in tremendous degradation of performance as number of users grew in number. The reason being, after sometime the user processes go into swap. Sybase pioneered the multithreaded RDBMS design in which, irrespective of the number of users taking database services, the database process code just be one. Today, most of the important RDBMSs provide for multithreaded server design. The multithreaded architecture combined with server integrity, Client Server Architecture, connectivity to networking protocol, connectivity to heterogeneous databases has resulted in major movement towards enterprise wide computing.

2.5 THE CRITICAL PRODUCTS

In this section we will briefly look at some tools to implement client/server environment.

2.5.1 Object Oriented Technology (OOT)

The fundamental ideas underlying OOT are:

- Abstraction
- Objects
- Encapsulation
- Classes and Instance
- Inheritance
- Message
- Methods

How OOT differs from structured programming?

Structured Programming

Data and code are separate and code operates on data.

OOT

Data and procedures are together and the object responds to messages.

Abstraction

It is the act of removing certain distinctions between objects so that we can see commonalties. The result of an abstraction process is a concept or object type. One of the forms of abstraction is Data Abstraction. Here only the selected properties of the object are made visible to the outside world and their internal representation are hidden. The object model has a greater advantage over conventional languages that the lower level implementation details are not visible.

Object

An object is any thing, real or abstract, about which we store data and those methods that manipulate the data. Its an encapsulated abstraction that includes state information and a clearly defined set of access protocol (messages to which object responds). It is a software package which contains related data and procedures. The basic concept is to define software objects that can intersect with each other just as their real-world counterparts do.

An object type is a category of object. An object is an instance of an object type.

Encapsulation

Packaging data and methods together, is called Encapsulation.

Its advantages are:

- Unnecessary details are hidden.
- Unintentional data modification is avoided i.e. provides security and reliability.
- Presents interference with the internals and also hides the complexity of the components. Thus, encapsulation is important because it separates how an object behaves, from how it is implemented.

Classes

A class is an implementation of an object type and is defined by and is defined by a class description that defines both the attributes and messages for an object that class. It specifies a data structure and the permissible operational methods that apply to each of its objects. Classes can also be objects in some object oriented language.

An object is an instance of a class. The properties of any instance (object) are given by the class description of its class. Thus,

- Class is template that helps us to create objects.
- Classes have names (class identifier) that indicates the kind of objects they represent.
- Classes may be arranged in hierarchy with subclass representing more specific kinds of objects than their super class.

Inheritance

Inheritance allows the developer to create a new class for object from an existing one by inheriting the behaviour and then modifying or adding to it. It provides an ability to create classes that will automatically model themselves on other classes. Sometimes a class inherits properties of more than one superclass, then its called MULTIPLE INHERITANCE. This inheritance leads to a "Class Hierarchy". It is a network of classes that starts with the most general as the uppermost branches and descends to the bottom leaves which are most specific. The power of an Object Oriented environment is defined by the Class Hierarchy and its capabilities.

Its advantages are:

- Reusability of code
- Avoid duplication of code
- Reduce cost of maintenance

Message

A message is

- A specific symbol, identifier or key-word(s) with or without parameters that represents an action to be taken by an object.
- Only way to communicate with an object is through message passing.
- An object knows what another intrinsic property/capability object has, but not how it does it.
- A message is not restricted to one recipient object but to multiple object.

In a conventional programming language an operation is invoked by calling a named procedure and supplying with it the data to act on. If the wrong data are supplied the wrong results will be returned.

Methods

They are often called SELECTORS since when they are called by name they allow the system to select which code is to be executed.

- Methods are description of operations.
- Method appears as a component of object.
- There is a 1-1 correspondence between messages and methods that are executed when a message is received by a given object.
- The same message might result in different method.

These concepts have been also explained in the previous unit.

2.5.2 Distributed Computing Environment

Distributing computing environment integrates computers in geographically distant location and underlying applications. These computers could be of different types with different operating systems, even the RDBMSs. It can also utilize heterogeneous client environment like Powerbuilder, VisualBasic, Uniface etc.

Issues involve:

- Networking multiple machines.
- Integrating RDBMS applications using global dictionary or two-phase commit (2PC) or replicated server/table.

Global Dictionary

The data dictionary of these geographically separate databases is stored centrally to take action on distributed transactions. This caters to location transparency e.g. if the salary of all the people in a corporation belonging to grade G2 is higher by 25%, the transaction statement need not specify the location name where G2's data has to be changed. A statement similar to increased salary of G2 by 25% globally will do.

Two Phase Commit (2PC)

In 2PC, the commit server in phase one checks out the availability of participating servers in all the locations. In phase two, after sending the transactions to the individual participating servers and receiving OK from them on updation, the commit server END COMMIT status to all the participating servers.

Replicated Server/Table

In this method, the tables required for distributed transaction are copied/replicated to all participating server sites. The replication server ensures that when there is a change in the replicated table, the change transmitted to all the locations. This enables reduced network traffic and lays prone to application storage owing to network failure.

The communication between heterogeneous database is done through database gateways.

2.5.3 Application Programming Interface (API)

It is simply a specification of a set of functions that allow client and server processes to communicate. It hides the underlying platform hardware and software from the developer. APIs show the developer a single-system image across a heterogeneous network of processors. e.g. Open Database Connectivity (ODBC). The primary advantage of developing the client application using a standard API is that the resulting application can use any back-end database server rather than just a specific server. The primary disadvantage is that they generally include the **least common denominator** of all tools and database servers that support the standard. So consider the use of API only if two or more databases servers are used.

2.5.4 Multithreaded Processes

A single process can have multiple threads of execution. This simply means that a single multithreaded process can do the work of multiple single-threaded processes. The advantage of a multithreaded process is that it can do the work of many single-threaded processes but requires far less system overheads. If a database server uses multithreaded server processes, it can support large number of clients with minimal system overheads. The user processes and server processes are different and one server process can serve multiple user processes. This configuration is called Multithreaded Architecture.

2.5.5 Remote Procedure Calls (RPC)

With RPC one component communicate with a remote component using simple procedure calls. This involves peer to peer messaging. If an application issues a functional request and this request is embedded in an RPC, the requested function can be located anywhere in the enterprise, the caller is authorised to access. The advantage of this process to process communication is evident when processors are involved in any simultaneous processes. New client applications that use object-level relationships between processes provide need for this type of communication e.g. a client requests information from a server by connecting to the server, making the request by calling low-level procedure native to the database server, and then disconnecting. To respond to a request, the server connects to the application and calls a low-level procedure in the application.

RPC is typically transparent to a user, making it very easy to use. The RPC provides facility for the invocation and execution of requests from processors running different operating systems and using different hardware platforms from the callers.

2.5.6 Dynamic Data Exchange (DDE)

Through a set of APIs, windows provide calls that support to the DDE protocol for message-based exchange of data among applications. DDE can be used to construct HOT-LINKS between applications where data can be fed from window to window without operation intervention. DDE support WARM-LINKS that we can create so the server application notifies the client that the data has changed and client can issue an explicit request to receive it. We create REQUEST-LINKS to allow direct copy and paste operation between a server and client without the need for an immediate clipboard. No notification of change in data by the server application is provided. EXECUTIVE LINKS cause the execution of one application to be controlled by the another. This provides an easy to use batch processing capability. Thus, using DDE, applications can share data, execute commands remotely and check error conditions.

2.5.7 Object Linking and Embedding (OLE)

Linking is one way of attaching information from one application to another. Link can be locked, broken or reconnected. Linked information is stored in source application.

Embedding makes the information to be embedded, a part of the destination document, thus increases its size.

OLE is designed to let users focus on data rather than on the software required to manipulate the data. A document becomes a collection of objects, rather than a file. Applications that are OLE-capable provide an API that passes the description of the object to any other application that requests the object. OLE is perhaps the most powerful way to share information between documents. In order to link an object created in another application to another file, but applications need to be running in the same environment i.e. either DDE or OLE.

Advantages of OLE

- We can display an embedded or linked object as an icon instead of its full size.
- We can convert an embedded or linked to a different application.

2.6 DEVELOPING AN APPLICATION

C/S application developments requires broadly dividing the application into two categories:

- Server Coding
- Client Coding

Server Coding :

Creation of Database

It has two major phases—the design phase and the creation phase. The design phase includes planning file limits, size and location of the initial data files, size and location of the new database transaction log groups and members, determining the character set to store database data. Once we have planned a new database we can execute it using the SQL commands.

Creation of Tables

Once database is created, the tables to be kept under this database are designed. The table is comprised of columns and the properties of these columns are decided i.e. whether the field is null or not null, which is the primary/foreign key etc.

Creation of Database Triggers

Triggers ensure that when a specific action is performed, related actions are performed. It also ensures that centralised, global operations should be fired for the triggering statement, regardless of which user or database application issues the statement. By default, triggers are automatically enabled when they are created. A pre-defined or user-defined error conditions or exception may be raised during execution of a trigger, if so, all effects of the trigger execution are rolled back, unless the exception is specifically handled.

Creation of Stored Procedures

A stored procedure is a schema object that logically groups a set of SQL and PL/SQL programming language in Oracle and T-SQL statements in Sybase together to perform a specific task these are created in a user schema and stored in a database for continued use. These can be invoked by calling explicitly in the code of a database application, by another procedure or function or by a trigger. It is defined to complete a single, focussed task. It should not duplicate functionality provided by another feature of the server e.g. defining procedures to enforce data integrity rules that may be enforced using integrity constraints.

Creation of Server Enforced Validation Checks

This is done through database integrity rules. It guarantees that the data in a database adheres to a predefined set of constraints. Its a rule for a column of a table which prevents invalid data-entry into the tables of a database. It is stored in a data dictionary. It supports entity integrity and referential integrity. Rules depend on type of data and condition specified at time of access of data and frequently accessed as a check on transaction and not as a constraint on the database.

Creation of Indexes

Indexes are used to provide quick access to rows in a table. It provide faster access to data for operations that return a small portion of the rows of a table. Indexes should be created after loading data in a table. Index those columns which re-used for joins to improve performance on joins of multiple tables.

Creation of Views

Views are created to see the same data that is in database tables, but with a different perspective. A view is a virtual table, deriving its data from base tables. Views are used to limit access to specific table columns and create value-based security by defining a view for

a specific rows. Views can also be used to derive other columns not present in any table e.g. calculated field.

2.7 STRUCTURED QUERY LANGUAGE (SQL)

It has emerged as the standard for query language for relational DBMSs. Its original version was called SEQUEL. It is still pronounced as SEQUEL.SQL is both the data definition and data manipulation language of a number of relational database systems e.g. Oracle, Ingres, Sybase, Informix etc.

Note : In this discussion, we would be taking examples for Hotel database having two tables:

Employee (Emp_no, Name, skill, Pay_rate)

Duty_allocation(Posting_no, Emp_no, Day, Shift)

2.7.1 Data Definition Language (DDL)

Data definition in SQL is via the create statement.

Create A Table:

Syntax:

Create table < relation (attribute list)

< attribute list > = < attribute name > (<data type>)
[< attribute list >]

< data type > = < integer > | < smallint > | < char(n) > | < varchar > | < float > | < decimal > (p [, q]) >

Note: In above syntax, relation means table (i.e. file), while attribute means fields or columns. In addition, some data types may be implementation dependent.

For example, the employee relation for the Hotel database can be created as:

```
create table Employee
(Emp_no     integer not null,
Name        char (25),
Skill       char (20),
Pay-rate    decimal (10,2))
```

Alter Table

The definition of an existing relation can be altered by using the alter statement. It allows the new column to be added. The physical alteration occurs only during an update of the record.

Syntax:

```
alter table existing_table_name
add column_name data type{...}
```

For example, to add phone_number attribute to the employee relation alter table Employee add phone_number decimal(10)

Create Index

It allows the creation of an index for an already existing table.

Syntax:

```
create[unique] index name_of_index
    on existing_table_name
(column_name[ascending or descending]
[,column_name[order],...]) [cluster]
```

Cluster option is used to indicate that the records are to be placed in physical proximity to each other.

For example, create an index (named empindex) on Employee relation using columns: Name and Pay_rate

```
Create index empindex
on Employee (Name asc, Pay_rate desc)
```

Drop Table/Index

It is used to delete relation/index from the database.

Syntax:

```
drop table existing_table_name
drop index existing_index_name
```

2.7.2 Data Manipulation Language (DML)

Select Statement

It is the only data retrieval statement in SQL. It is based on relational calculus and entails selection, join and projection.

Syntax :

```
select [distinct/unique] <target list>
from <relation list>
[where <predicate>]
[order by attribute_name desc/asc]
[group by attribute_name]
[having value_expression]
```

For example, find the values for attribute Name in the employee relation.

```
select Name
from Employee
```

For example, get Duty_allocation details in ascending order of Day for Emp_no 123461 for the month of April 1986 as well as for all employees for shift 3 regardless of dates.

```
select *
from Duty_allocation
where   (Emp_no = 123461 and Day  19860401 and
          Day  = 19860430) or shift = 3)
order by Day asc
```

Update Statement

Syntax :

```
update <relation> set <target_value_list
[where <predicate> ]
<target_value_list> = <attribute name> =
    <value exp> [, <target_value_list>]
```

For example, change Pay_rate to 8 of the employee Ron in the Employee relation.

```
update Employee
    set Pay_rate = 8
     where Name = 'Ron'
```

Delete Statement

It deletes on or more records in the relation.

Syntax :

```
delete < relation >
[where < predicate >]
```

If where clause is left out, all the tuples in the relation are deleted. In this case, the relation is still known to the database although it is an empty relation.

For example, delete the tuple for employee Ron in the Employee relation.

```
delete < relation >
[where < predicate >]
```

Insert Statement

It is used to insert new tuples in a specified relation.

Syntax :

```
insert into < relation > (< target list >)
value (< values list >)
< value list > = < value expression > [, < target list >]
```

We can replace value clause by select statement.

e.g. Inset a tuple for the employee Ron.

```
insert into Employee
values (123456, 'Ron', 'waiter', 8)
```

Condition Specification

SQL supports the following Boolean and comparison operators: and, or, not, =, < >, >=>, <, <=, like. If more than one of the Boolean operators appear together, not has the highest priority while or has the lowest. Parentheses may be used to indicate the desired order of evaluation.

Arithmetic and Aggregate O, rators

Avg

Min

Max

Sum

Count

For example, find the average pay rate for employee working as a chef.

```
select avg (Pay_rate)
from Employee
where skill = 'chef'
```

For example, get the number of distinct pay rates from the Employee relation.

```
select count (distinct Pay_rate)
from Employee
```

For example, get minimum and maximum pay-rates.

```
select min(Pay_rate), max(Pay_rate)
from employee
```

Join

SQL does not have a direct representation of the JOIN operator. However, the type of join can be specified by an appropriate predicate in the where clause of the select statement.

For example, retrieve the shift details for employee RON.

```
select Posting_No, Day, Shift
from Duty-allocation, Employee
where Duty-allocation. Emp_No
    =   Employee.Emp_No
        and
        Name = 'Ron'
```

SQL uses the concept of tuple variables from relational calculus. In SQL a tuple variable is defined in the from clause of the select statement.

For example, get employees whose rate of pay is more than or equal to the pay of employee Pierre.

```
select e1.Name, e2.Pay-rate
from Employee e1, Employee e2
where e1.Pay- rate > e2.Pay
    and
e2. Name = 'Pierre'
```

Set Manipulation

SQL provides following set of operators:

- Any
- In
- Exists
- Not exists
- Union
- Minus
- Intersects
- Contains

When using these operators, remember that the statement 'select...' returns a set of tuples.

Any

It allows the testing of a value against a set of values.

For example, get the names and pay rates of employees with employee number less than 123469 whose rate of pay is more than the rate a pay of at least one employee with employee-No >= 123460.

```
select Name, Par-rate
from Employee
where Emp-No  123480
        and
        Pay-rate  > any (select Pay-rate
                         from Employee
                         where Employee >= 123460
```

In

Its equivalent to = any.

For example, get employees who are working either on the date 19860419 or 19860420

```
select Emp_No
from Duty-allocation
where Day in (19860419, 19860420)
```

Contains

It is used to test for the containment of one set in another

For example, find the names of all the employees who are assigned to all the positions that require chef's skill.

```
select e.Name
from employee e
where
        (select Posting_no
        from Duty_allocation d
        where e.Emp_no = d.Emp_no
        contains
                (select p.Posting_no
                from Position p
                where p.skill = 'chef')
```

* Position is another relation of Hotel database:

Position (Posting_no, skill)

All

For example, find the employees with the lower pay-rate

```
select Emp_No, Name, Pay-rate
from Employee
where Pay-rate <= all
    (select*
Pay-rate from Employee)
```

Not In

It is equivalent to # all

Not Contain

It is complement of contains

Exists

exists (select x from ...)

It evaluates to true if and only if the result of "select x from ..." is not empty.

For example, find the names and pay-rate of all the employees who are allocated a duty.

```
select Name, Pay-rate
from Employee
 where exists
        (select *
        from Duty-allocation
        where Employee.Emp_no =
            Duty_allocation.Emp_no
```

Not Exists

It is complement of exists.

For example, find the name of pay rate of all the employee who are not allocated a duty.

```
select Name, Pay-rate
from Employee
where not exists
        (select *
        from Duty-allocation
        where Employee.Emp_no =
            Duty_allocation.Empl_no)
```

Union

Duplicates are removed from the result of a union.

For example, get employees who are waiters or working at posting-no 321.

```
select Emp-No
from Employee
where skill = 'waiter'
 union
        (select Emp-No
        from Duty-allocation
        where Posting-No = 321)
```

Minus

For example, get a list of employees not assigned a duty.

```
select Emp-No
from Employee
minus
    (select Emp-No
    from Duty-allocation)
```

Intersect

For example, get a list of names of employees with the skill of chef who are assigned a duty.

```
select Name
from Employee
 where Emp_no in
    ((select Emp_No
        from Employee
        where skill = 'chef'
        intersect
        (select Emp-No
        from Duty-allocation))
```

Categorization

Sometimes we need to group the tuples with some common property and perform some group operations on them. For this, we use group by and having options in where clause.

For example, get a count of different employees on each shift.

```
select shift, count(distinct Emp_No)
from Duty-allocation
group by shift
```

For example, get a count of different employees on each shift.

```
select shift, count(distinct Emp-No)
from Duty-allocation
group by shift
```

For example, get employee number of all employees working on at least two dates.

```
select Emp-No
from Duty-allocation
group by Emp-No
having count(*) > 1
```

View

Conceptual or physical relations are called base relations. Any relation that is no part of the physical database i.e. a virtual relation, is made available to the users as a view. A view can be defined using a query expression.

```
create view <view name>
as <query expression>
```

For example, create a view named Emp_view containing the fields Emp_No and Name from Employee relation.

```
create view Emp_view
(select Emp_No, Name
from Employee)
```

DROP VIEW

Drop view view-name

2.8 CLIENT/SERVER: WHERE TO NEXT?

Client/Server Computing has a great future ahead. The successful organizations have to be market driven and competitive in the times to come, and they will use Client/Server Computing as the enabling technology to add values to their business.

In future cheap and powerful workstations will be available to all end users to be used as clients to access the information on the servers which are distributed globally. The future Client/Server Information System will provide the information from data in its original form, e.g. image, video, graphics, documents, spreadsheets etc. without the need to be specific about the software used to store and process each of these.

The future trends in networking show that there is going to be an explosion in the number network users and more than 70% users, and obviously most of them will use Client/Server as the underlying technology. The networks of the future will support much higher bandwidth (of the order of 100 Mbps) by using the technologies like corporate networks will cut across the boundaries of cities or even countries and they will be connected to major networks around the world. An organization living in isolation will not survive in future.

The future Client/Server Information Systems will use the object oriented programming—OOP. Techniques to make zero defect applications. The OOPs will provide the capability to reuse previously tested components. The reuse of already tested components is quite common in most engineering and manufacturing applications (or even the hardware design), the OOPs makes it for the software development too.

The future Client/Server will cover the systems such as Expert Systems, Geographic Information Systems, Point-of- Services, Imaging, Text Retrieval, Document Management Systems or Electronic Filing Systems, Executive Information Systems, Decision Support Systems etc., alongwith the data handling in OLTP (On Line Transaction Processing) and real time environments.

Check Your Progress

1. Discuss trade-off between mainframe and Client/Server environment.

.....

.....

.....

.....

2.9 SUMMARY

The first major challenge for business today is staying competitive in a changing liberalized global economy. Success, even survival, depends on how quickly and accurately one can get up-to-date information so that major business decision can be taken without any delay.

The availability of a vast amount of knowledge and information needs integration of computer and communication systems.

There is a paradigm shift today in using technology for finding solutions/information from the earlier days.

- In the 1960s by centralized mainframes
- In the 1970s by minicomputers, as distributed data processing
- In the 1980s by the personal computers (PCs) and Local Area Networks (LANs)
- In the 1990s by Client/Server architecture which use as server product ranging from UNIX and NT box's to Database Computers (DBCs) and supercomputers.

Companies that have moved out of this mainframe system to Client/Server architecture have found three major advantages:

- Client/Server technology is more flexible and responsive to user needs
- A significant reduction in data processing costs
- An increase in business competitiveness as the market edge turns towards merchandising

2.10 MODEL ANSWERS

1. In a mainframe all operations take place on one system. This type of environment is being used for the last 30 years, but it is:

- Highly costly
- Highly inflexible
- Quite slow because of contention
- Less reliable

By contrast to this monolithic approach Client/Server (shown in figure 2) provides a low-priced robust solution to user requirements. This approach permits downsizing production subsystem while allowing the clients and servers the necessary tools and facilities to control, manage and tune the environment in which they operate.

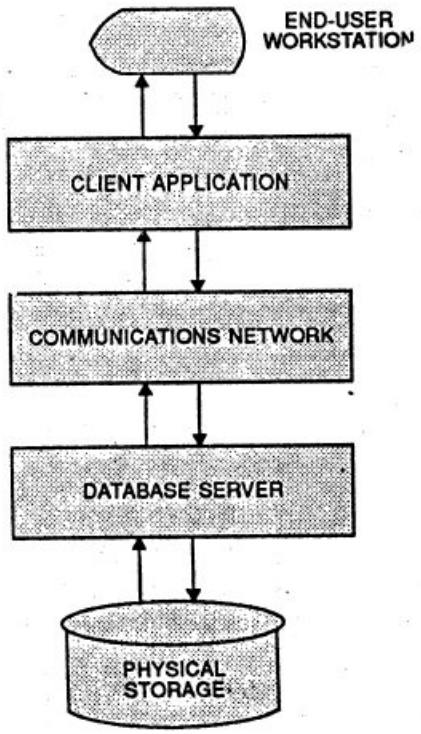


Figure 2 : A layered approach to computing enhances functionality and increases flexibility, speed and reliability at low cost.

Most Client/Server solution are also very attentive in matters of security. Access to any resource can be defined to the file level, with such access being controlled through identification and authorisation. Logically defined closed use groups can be setup to enable the enhancing of security measures by network administrators.

2.11 FURTHER READINGS

1. Developing Client/Server Applications by W.H.Inmon
2. Guide to Client/Server Database by Joe Salcmi
3. Mastering Oracle and Client/Server Computing by Steven M.Bobrowski
4. BSG Client/Server Computing by SHL System House
5. Featuring SQL standard by Hayden Book
6. Datapro report on Client/Server Computing by Emerging Trends, Solutions and Strategies.
7. Datapro report on Client/Server Challenge by FDDI or Ethernet Switches
8. Datapro report on The Hidden Cost of Client/Server Computing

9. Client/Server Computing by Patrick Smith and Steve Guengerich

10. GUI based Design and Development for Client/Server Applications using Power Builder, SQL Windows, Visual Basic, PARTS Workbench by Jonathan S.Sayles, Steve Karlen, Peter Molchan and Gary Bilodeau.

11. Beyond LANS Client/Server Computing By Dimitris N. Chorafas; McGraw-Hill Series on Computer Communication;1994.

UNIT 3 INTRODUCTION TO KNOWLEDGE DATABASES

Structure

3.0 INTRODUCTION

A knowledge base management system (KBMS) is a computer system that manages the knowledge in a given domain of interest and exhibits reasoning power to the level of a human expert in this domain. In typical Artificial Intelligence (AI) application, knowledge representation requires data structures with rich semantics that go beyond the simple structure of the relational model. AI is the part of computer science with designing intelligent computer systems, that is systems that exhibit the characteristics we associate with intelligence in human behaviour. Furthermore, operations in a knowledge base are more complex than those in traditional database. When a rule is added the system must check for contradiction and redundancy. Such operations cannot be represented directly by relational operations and the complexity of checking increases rapidly as the size of knowledge base grows.

3.1 OBJECTIVES

After going through this unit you will be able to :

- define what is knowledge, hypothesis and belief
- explain what is a Knowledge base system
- differentiate between Knowledge base system and a database system
- list several knowledge representation schemes

3.2 DEFINITION AND IMPORTANCE OF KNOWLEDGE

Definition and Importance of Knowledge

Knowledge can be defined as the body of facts and principles accumulated by human-kind or the act, fact or state of knowing. While this definition may be true, it is far from complete. We know that knowledge is much more than this. It is having a familiarity with **language, concepts, procedures, rules, ideas, abstractions, places, customs, facts and associations,** coupled with an ability to use these notions effectively in modelling different aspects of the world. Without this ability, the facts and concepts are meaningless and, therefore, worthless. The meaning of knowledge is closely related to the meaning of intelligence. Intelligence requires the possession of and access to knowledge. And a characteristic of intelligent people is that they possess much knowledge.

In biological organisms, knowledge is likely stored as complex structures of interconnected

neurons. The structures correspond to symbolic representation of the knowledge possessed by the organism, the facts, rules and so on. The average humane brain weighs about 3.3 pounds and contains an estimated number of 10^{12} neurons. The neurons and their interconnection capabilities provide about 10^{14} bits of potential storage capacity.

In computers, knowledge is also stored as symbolic structures, but in the form of collections of magnetic spots and voltage states. State-of-the-art storage in computers is in the range of 10^{12} bits with capacities doubling about every three to four years. The gap between human and computer storage capacities is narrowing rapidly. Unfortunately, there is still a wide gap between representation schemes and efficiencies.

A common way to represent knowledge external to a computer or a humane is in the form of written language. For example, some facts and relations represented in printed English are

> Jancy is tall.
>
> Ram loves Sita.
>
> Som has learned to use recursion to manipulate Binary tree in several programming languages.

The first item of knowledge above expresses a simple fact, an attribute possessed by a person. The second item expresses a complex binary relation between two persons. The third item is the most complex, expressing relations between a person and more abstract programming concepts. To truly understand and make use of this knowledge, a person needs other world knowledge and the ability to reason with it.

Knowledge may be **declarative or procedural. Procedural knowledge is compiled knowledge related to the performance of some task.** For example, the steps used to solve an algebraic equation are expressed as procedural knowledge. **Declarative knowledge, on the other hand, is passive knowledge expressed as statements of facts about the world.** Personnel data in a database is typical of declarative knowledge. Such data are explicit pieces of independent knowledge.

Frequently, we will be interested in the use of **heuristic knowledge,** a special type of knowledge used by humans to solve complex problems. **Heuristics are the knowledge used to make good judgments, or the strategies, tricks or "rules of thumb" used to simplify the solution of problems.** Heuristics are usually acquired with much experience. For example, in locating a fault in a TV set, an experienced technician will not start by making numerous voltage checks when it is clear that the sound is present but the picture is not, but instead will immediately reason that the high voltage flyback transformer or related component is the culprit. This type of reasoning may not always be correct, but it frequently is, and then it leads to a quick solution.

Knowledge should not be confused with data. Some scientists emphasize this difference with the following example. A physician treating a patient uses both knowledge and data. The data is the patient's record, including patient history, measurements of vital signs, drugs given, response to drugs, and so on, whereas the knowledge is what the physician has learned in medical school and in the years of internship, residency, specialization, and practice. Knowledge is what the physician now learns in journals. It consists of facts, prejudices, beliefs, and most importantly, heuristic knowledge.

Thus, we can say that knowledge includes and requires the use of data and information. But it is more. It combines relationships, correlations, dependencies, and the notion of gestalt with data and information.

Even with the above distinction, we have been using knowledge in its broader sense up to this point. At times, however, it will be useful or even necessary to distinguish between knowledge and other concepts such as **belief** and **hypotheses.** For such cases we make the following distinctions. We define belief as essentially any meaningful and coherent expression that can be represented. Thus, a belief may be true or false. We define a hypothesis as a justified belief that is not known to be true. Thus, a **hypothesis is a belief which is backed up with some supporting evidence, but it may still be false.** Finally, we define knowledge as true justified belief.

Two other knowledge terms which we shall occasionally use are epistemology and metaknowledge. Epistemology is the study of the nature of knowledge, whereas metaknowledge is knowledge about knowledge, that is, knowledge about what we know.

In this section we have tried to give a broader definition of knowledge than that commonly found in dictionaries. Clearly, we have not offered a scientific definition, we are not able to measure knowledge. How then will we know when a system has enough knowledge to perform a specified task? Can we expect to build intelligent systems without having a more precise definition of either knowledge or intelligence? In spite of our ignorance about knowledge, the answer is definitely yes.

Finally, our overall picture of knowledge cannot be complete without also knowing the meaning of closely related concepts such as understanding, learning, thinking, remembering, and reasoning. These concepts all depend on the use of knowledge. But then just what is learning, or reasoning, or understanding? Here too we will find dictionary definitions lacking. And, as in the case of knowledge and intelligence, we cannot give scientific definitions for any of these terms either.

The Importance of Knowledge

AI has given new meaning and importance to knowledge. Now, for the first time, it is possible to "package" specialized knowledge and sell it with a system that can use it to reason and draw conclusions. The potential of this important development is only now beginning to be realised. Imagine being able to purchase an untiring, reliable advisor that gives high level professional advice in specialised areas, such as manufacturing techniques, sound financial strategies, ways to improve one's health, top marketing sectors and strategies, optimal farming plans, and many other important matters. We are not far from the practical realisation of this, and those who create and market such systems will have more than just an economic advantage over the rest of the world.

3.3 WHAT IS A KNOWLEDGE BASE SYSTEM?

One of the important lessons learned in AI during the 1960s was that general purpose problem solvers which used a limited number of laws or axioms were too weak to be effective in solving problems of any complexity. This realisation eventually led to the design of what is now known as Knowledge base system, systems that depend on a rich base of knowledge to perform difficult tasks.

Edward Feigenbaum summarised this new thinking in a paper at the International Joint Conference on Artificial Intelligence (IJCAI) in 1977. He emphasised the fact that the real power of an expert system comes from the knowledge it possesses rather than the particular inference schemes and other formalisms it employs. This new view of AI systems marked the turning point in the development of more powerful problem solvers. It formed the basis for some of the new emerging **expert systems** being developed during the 1970s including MYCIN, an expert system developed to diagnose infectious blood diseases. An expert system contains knowledge of experts in a particular domain along with an inferencing mechanism and an explanation sub-system. It is also called knowledge base system.

Since this realisation, much of the work done in AI has been related to so-called Knowledge base systems, including work in vision, learning, general problem solving and **natural language understanding.** This in turn has led to more emphasis being placed on research related to knowledge representation, memory organisation, and the use and manipulation of knowledge.

Knowledge base systems get their power from the expert knowledge that has been coded into facts, rules, heuristics, and procedures. The knowledge is stored in a knowledge base separate from the control and inferencing components. This makes it possible to add new knowledge or refine existing knowledge without recompiling the control and inferencing programs. This greatly simplifies the construction and maintenance of Knowledge base systems

In the knowledge lies the power! This was the message learned a few farsighted researchers at Stanford University during the late 1960s and early 1970s.

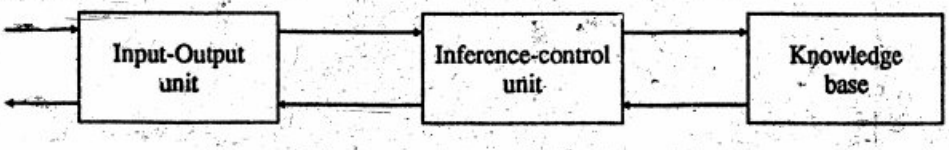


Figure 1 : Components of a Knowledge-based system.

The proof of their message was provided in the first Knowledge base expert systems which were shown to be more than toy problem solvers. These first systems were real world problem solvers, tackling such tasks as determining complex chemical structures given only the atomic constituents and mass spectra data from samples of the compounds and later performing medical diagnoses of infectious blood diseases.

Using the analogy of a DBMS, we can define a **knowledge base management system (KBMS)** as a computer system used to manage and manipulate shared knowledge. A knowledge base system's manipulation facility includes a **reasoning facility**, usually including aspects of one or more of the following forms of reasoning : deductive, inductive, or abductive. **Deductive reasoning** implies that a new fact can be inferred from a given set of facts or knowledge using known rules of inference. For instance, a given proposition can be found to be true or false in light of existing knowledge in the form of other propositions believed to be either true or false. **Inductive reasoning** is used to prove something by first proving a base fact and then the increment step; having proved these, we can prove a generalized fact. **Abductive reasoning** is used in generating a hypothesis to explain observations. Like deductive reasoning, it points to possible inferences from related concepts; however, unlike deductive reasoning, the number of inferences could be more than one. The likelihood of knowing which of these inferences corresponds to the current state of the system can be gleaned from the explanations generated by the system. These explanations can facilitate choosing among these alternatives and arriving at the final conclusion.

In addition to the reasoning facility, a knowledge base system may incorporate an **explanation facility** so that the user can verify whether the reasoning used by the system is consistent and complete. The reasoning facility also offers a form of tutoring to the uninitiated user. The so-called expert systems and the associated expert system generation facilities are one form of knowledge base systems that have emerged from research labs and are being marketed commercially. Since a KBMS includes reasoning capacity, there is a clear benefit in incorporating this reasoning power in database application programs in languages such as COBOL and Pascal.

Most knowledge base systems are still in the research stage. The first generation of commercial KBMSs are just beginning to emerge and integration of a KBMS with a DBMS is a current research problem. However, some headway has been made in the integration of expert systems in day-to-day database applications.

3.4 DIFFERENCE BETWEEN A KNOWLEDGE BASE SYSTEM AND A DATABASE SYSTEM

There is no consensus on the difference between a knowledge base system and a database system. In a DBMS, the starting point is a data model to represent the data and the interrelationships between them; similarly, the starting point of a KBMS is a **knowledge representation scheme.** The requirements for any knowledge representation scheme should provide some mechanism to organise knowledge in appropriate hierarchies or categories, thus allowing easy access to associated concepts. In addition, since knowledge can be expressed as rules and exceptions to rules, **exception-handling features** must be present in the knowledge stored in the system must be insulated from changes in usage in its physical or logical structure. This concept is similar to the data independence concept used in a DBMS. To data, little headway has been made in this aspect of a KBMS.

A KBMS is developed to solve problem for a finite domain or portion of the real world. In developing such a system, the designer selects a significant objects and relationships among these objects. In addition to this domain-specific knowledge, general knowledge such as concepts of up, down, far, near, cold, hot, on top of, and besides must be incorporated in the KBMS. Another type of knowledge, which we call common sense, has yet to be successfully incorporated in the KBMS.

The DBMS and KBMS have similar architectures; both contain a component to model the information being managed by the system and have a subsystem to respond to queries. Both systems are used to model or represent a portion of the real world of interest to the application. A database system, in addition to storing facts in the form of data, has limited capability of establishing associations between these data. These associations could be pre-established as in the case of the network and hierarchical models, or established using

common values of shared domains as in the relational model. A knowledge base system exhibits similar associative capability. However, this capability of establishing associations between data and thus a means of interpreting the information contained is at a much higher level in a knowledge base system, ideally at the level of a knowledgeable human agent.

One difference between the DBMS and KBMS that has been proposed is that the knowledge base system handles a rather small amount of knowledge, whereas a DBMS efficiently (as measured by response performance) handles large amounts of shared data. However, this distinction is fallacious since the amount of knowledge has no known boundaries and what this says is that existing knowledge base systems handle a very small amount of knowledge. This does not mean that at some future date we could not develop knowledge base systems to efficiently handle much larger amounts of shared knowledge.

In a knowledge base system, the emphasis is placed on a robust knowledge representation scheme and extensive reasoning capability. Robust signifies that the scheme is rich in expressive power and at the same time it is efficient. In a DBMS, emphasis is on efficient access and management of the data that model a portion of the real world. A knowledge base system is concerned with the meaning of information, whereas a DBMS is interested in the information contained in the data. However, these distinctions are not absolute.

For our purposes, we can adopt the following informal definition of a KBMS. The important point in this definition is that we are concerned with what the system does rather than how it is done.

A knowledge base management system is a computer system that manages the knowledge in a given domain or field of interest and exhibits reasoning power to the level of a human expert in this domain.

A KBMS, in addition, provides the user with an integrated language, which serves the purpose of the traditional DML of the existing DBMS and has the power of a high-level application language. A database can be viewed as a very basic knowledge base system in so far as it manages facts. It has been recognised that there should be an integration of the DBMS technology with the reasoning aspect in the development of shared knowledge bases. Database technology has already addressed the problems of improving system performance, concurrent access, distribution, and friendly interface; these features are equally pertinent in a KBMS. There will be a continuing need for current DBMSs and their functionalities co-existing with an integrated KBMS. However, the reasoning power of a KBMS can improve the ease of retrieval of pertinent information from a DBMS.

3.5 KNOWLEDGE REPRESENTATION SCHEMES

Knowledge is the most vital part of **Knowledge Base System** or **Expert System.** These systems contain large amounts of knowledge to achieve high performance. A suitable **Knowledge Representation** scheme is necessary to represent this vast amount of knowledge and to perform inferencing over the Knowledge Base (KB). A Knowledge Representation scheme means a set of syntactic and semantic conventions to describe various objects. The syntax provides a set of rules for combining symbols and arrangements of symbols to form expressions.

Knowledge Representation is a non-trivial problem, which continues to engage some of the best minds in this field even after the successful development of many a Knowledge Base System. Some of the important issues in Knowledge Representation are the following:

i) **Expressive Adequacy :** What knowledge can be and cannot be represented in a particular Knowledge Representation scheme ?

ii) **Reasoning Efficiency :** How much effort is required to perform inferencing over the KB? There is generally a trade off between expressive adequacy and reasoning efficiency.

iii) **Incompleteness :** What can be left unsaid about a domain and how does one perform inferencing over incomplete knowledge ?

iv) **Real World Knowledge :** How can we deal with attitudes such as beliefs, desires and intentions ?

Major Knowledge Representation schemes are based on **production rules, frames, semantic nets and logic.** Facts and rules can be represented in these Knowledge Representation schemes. Inference Engines using **forward chaining, backward chaining** or a combination thereof are used alongwith these Knowledge Representation schemes to build actual Expert System. We will briefly describe these Knowledge Representation **schemes and inferencing engines.**

3.5.1 Rule Based Representation

A rule based system is also called production rule system. Essentially, it has three parts, working memory, rule memory or production memory and interpreter. Working memory contains facts about the domain. These are in the form of triples of objects, attribute and value. These facts are modified during the process of execution. Some new facts may be added as conclusions.

Production memory contains IF-THEN rules. IF part contains a set of conditions connected by AND. Each condition can have different other conditions connected by AND or OR. Each condition can give either true or false as its value. THEN part has a set of conclusions or actions. Conclusions may change values of some entity or may create new facts.

A rule can be fired when all the conditions in it are true. If any of the conditions is not true or unknown, the rule cannot be fired. If it is unknown, the system will try to determine its value. Once a rule has fired, all its conclusions and actions are executed.

For firing a rule, the system looks into its database. If a rule has some of its conditions satisfied, it is a candidate for further exploration. There may be more than one such rule. This conflict is resolved by some strategy like choosing the rule which contains the maximum number of satisfied conditions, or there may be metarules which may be domain dependent to move the reasoning in a particular direction.

Rules may be used in both **forward and backward reasoning.** When it is used in **forward mode,** the system starts with a given set of initial data and infers as much information as possible by application of various rules. Again new data are used to infer further. At any point system may ask the user to supply more information, if goal state has not been reached and no more rules can be applied. System keeps on checking for goal state at each firing of rules. Once goal state has been detected reasoning comes to an end. In **backward reasoning** mode, reasoning starts with the goal and rules are selected if they have the goal in their right hand side (RHS). To achieve the goal, left hand side (LHS) conditions have to be true. These conditions become new sub-goals. Now the system tries to achieve these sub-goals before trying the main goal. At some point it may not be possible to establish goal by application of rules. In this situation the system asks the user to supply the information.

It may be noted that these rules are not IF-THEN programming constructs available in most of the procedural Programming languages. These are different in the sense that they are not executed sequentially. Their execution depends on the state of the database which determines which are the candidates rules. Another difference is that IF-part is a complex pattern and not just a Boolean expression.

Rules have been used in many classical systems like **MYCIN, RI/XCON** etc. Even today it is the most frequent used **Knowledge Representation** scheme. The reason is that most of the time, experts find it easier to give knowledge in the form of rules. Further, rules can be easily used for explanations.

One problem with rules is that when they grow very large in number it becomes difficult to maintain them because KB is unstructured. Some techniques like context in **MYCIN** solve the problem to some extent.

3.5.2 Frame Based Representation

The concept of frame is quite simple. When we encounter a new situation, we do not analyse it from scratch. Instead we have a large number of structures or (records) in memory representing our experiences. We try to match the current situation with these structures and then the most appropriate one is chosen. Further details may be added to this chosen structure so that it can exactly describe the situation. A computer representation of this common knowledge is called a frame.

It is convenient to create a knowledge base about situations by breaking it into modular

chunks, called **frames.** Individual frames may be regarded as a record or structure.

Each frame contains slots that identify the type of situations or specify the parameters of a particular situation.

A frame describes a class of objects such as **ROOM** or **BUILDING.** It consists of various slots which describe one aspect of the object. A slot may have certain condition which should be met by the filler. A slot may also have default value which is used when the slot value is not available or cannot be obtained by any other way. If added procedure describes what is to be done if slots get a value. Such information is called facet of slot.

An example is presented below:

```
{CHAIR
    IS-A      :  FURNITURE
    COLOUR    :  BROWN
    MADE-OF   :  WOOD
    LEG       :  4
    ARMS      :  default : 0
    PRICE     :  100
```

Reasoning with the knowledge stored in a frame requires choosing an appropriate frame for the given situation. Some of the ways information may be inferred are the following :

a) If certain information is missing from current situation, it can be inferred. For example, if we have established that the given object is a room, we can infer that room has got a door.

b) Slots in a frame describe components of situation. If we want to build a situation then information associated with the slots can be used to build components of the situation.

c) If there is any additional feature in the object which can be discovered using a typical frame, it may require special attention. For example, a man with a tail is not a normal man.

3.5.3 Semantic Nets

Semantic net representation was developed for natural language understanding. Semantic net was originally designed to represent the meaning of English words. It has been used in many expert systems too. It is used for representation of declarative knowledge. In semantic nets, the knowledge is represented as a set of nodes and links. A node represents an object or concept and a link represents relationship between two objects (nodes).

Moreover, any node may be linked to any number of other nodes, so giving rise to a formation of network of facts. An example is shown in the following figure.

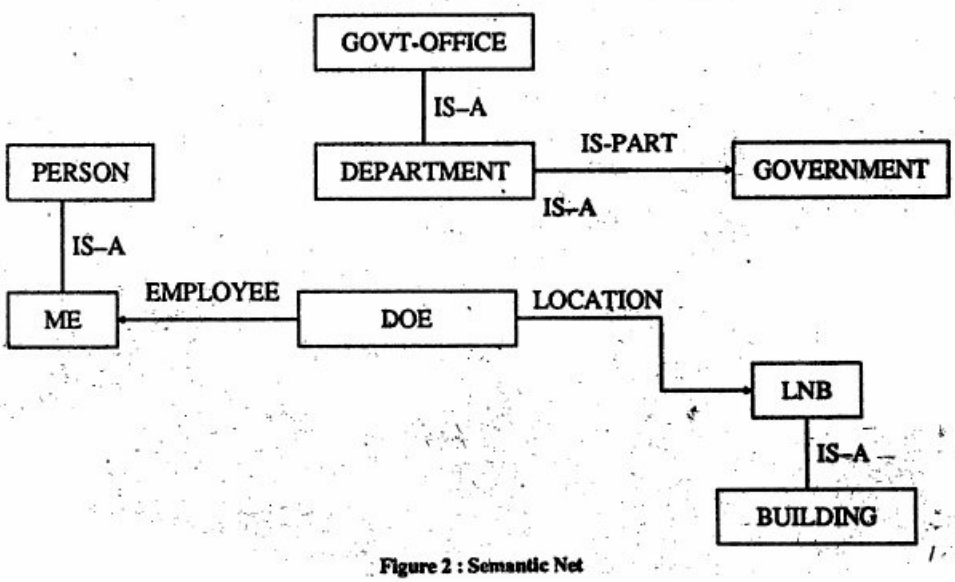


Figure 2 : Semantic Net

A semantic net as shown in figure 2, cannot be represented like this in computer. Every pair and its link are stored separately. For example, IS—A (DOE, Department) in PROLOG represents

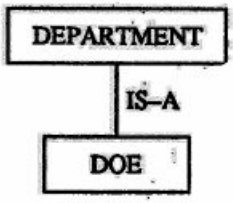


Figure 3 : One-way link representations

The link as shown in the figure is a one-way link. If we want an answer to "who is my employer ?" the system will have to check all the links coming to node ME. This is not computationally efficient. Hence reverse links are also stored. In this case we add

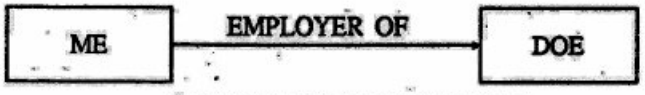


Figure 4 : Representation of a reverse link

In LISP the basic semantic network unit may be programmed as an atom/property list combination. The unit given in department—DOE semantic networks would be composed of "DOE" as the atom, "IS–A" as a property and "Department" as the value of that property. The value "Department" is of course, an atom in its own right and this may have a property list associated it as well. "Is-a" relationship indicates that one concept is an attribute of the other. Other links (relationship) of particular use for describing object concepts are "has", indicating that one concept is a part of the other. Using such relations, it is possible to represent complex set of facts through semantic network. The following figure illustrates one possible representation of facts about an employee "AKSHAY". These include—

"Akshay is a bank manager"

"Akshay works in the State Bank of India locate din IGNOU Campus"

"Akshay is 26 years old"

"Akshay has blue eyes"

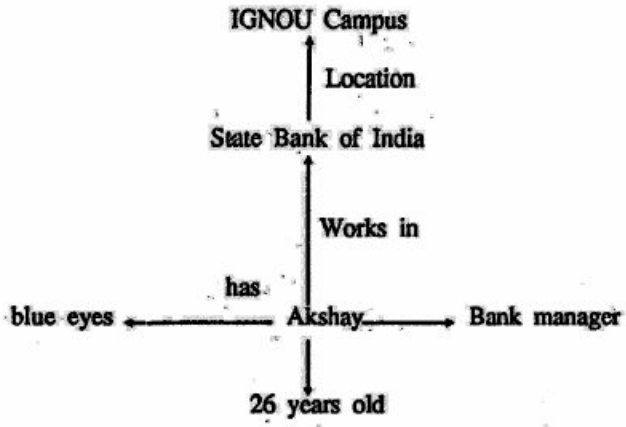


Figure 5 : Representation of complex sets of foots through semantic nets

When we have to represent more than two argument relationships, we break it down into arguments relationships.

SCORE (INDIA AUSTRALIA (250 150))

can be written as

participant (match-1 INDIA)

participant (match-1 AUSTRALIA)

score (match-1 (250 150))

As with an **Knowledge Representation** scheme, the problem solving power comes from the ability of the program to manipulate knowledge to solve a problem. Intersection search is used to find the relationship between two objects. In this case activation sphere grows from the two modes and intersects each other at some time. The corresponding paths give the correct relationship. More techniques have been developed to perform more directed search.

3.5.4 Knowledge Representation Using Logic

Traditionally logic has been studied by philosophers and mathematicians in different countries in order to describe and understand the world around us. Today computer scientists are using this tool to teach a computer about the world. Here we discuss propositional and predicate logic.

We can easily represent real-world facts as logical propositions in propositional logic. In propositional logic we deal with propositions like

It is raining. **(RAINING)**

It is sunny. **(SUNNY)**

It is windy. **(WINDY)**

If it is raining then it is not sunny.

(RAINING ——>~SUNNY)

Given the fact that "It is raining" we can deduce that it is not sunny.

But the representational power of propositional logic is quite limited. For example, suppose we have to represent

Vivek is a man.

Anurag is a man.

We may represent them in computer as **VIVEKMAN** and **ANURAGMAN**. But from these, we don't get any information about similarity between Vivek and Anurag. A better way of representation is

MAN (VIVEK)

MAN (ANURAG)

Consider the sentence 'All men are mortal'. This requires quantification like

The form of logic with these and certain other extra features is called predicate logic. The basic elements are described here.

Here capital letter P, Q etc. stand for predicates. A predicate is of the form predicate-name (arg1..argn)

It can have values true or false. A predicate represents among the arguments.

AND $\wedge$ (P $\wedge$ is true when both are true)

OR $\vee$ (P $\vee$ Q is true when atleast one is true)

NOT ~ (~P is true when P is false)

Implies (P $\rightarrow$ Q is true unless P is true and Q is false) means for all the values of X, P holds.

P(X) means there exists at least one value of X for which P holds.

means only some values are true.

The predicate logic has the following two properties:

a) Completeness : If P is a theorem of predicate logic then it can be derived only by the inferences rules available in predicate logic;

b) Soundness : There is no P such that both P and NOT P are theorems.

The decidability property of propositional logic does not carry over into predicate logic. The following inferences rules are some of the important rules available in predicate logic :

Modus Ponens : If P $\rightarrow$ Q and P is true then Q is true

Modus Tollens : If $P \rightarrow Q$ and Q is false then P is false

Chaining : If $P \vee Q$ and $(NOT\ P) \vee Q$ then Q is true

Reduce to : P and NOT P reduces to { }.

Most of the AI theorem provers for clausal form use resolution as the only way of inferencing. This subsumes the above five rules of inference. Resolution proves a theorem by refutation. First a normal form of clauses is obtained and then negation of the theorem is added to it. If it leads to a contradiction then the theorem is proved. A discussion of detailed algorithm is beyond the scope of this handout.

Let us now explore the use of predicate logic as a way of representing knowledge by looking at a specific example.

i) Anil is a Manager

ii) Anil is a disciplined

iii) All staff are either loyal to Anil or hate him

iv) Everyone is loyal to someone

The facts described by these sentences, can be represented in Predicate logic as follows:

i) Anil is a Manager
Manager (Anil)

ii) Anil is a disciplined
disciplined (Anil)

iii) All staff are either loyal to Anil or hate him.

iv) Everyone is loyal to someone

v) Predicate logic is useful for representing simple English sentences into a logical statement. But, it creates ambiguity for complicated sentences.

Check Your Progress

1. Define and describe the difference between knowledge, belief, hypothesis and data.

.....

.....

.....

2. What is the difference between declarative and procedural knowledge ?

.....

.....

.....

3. What are the techniques available for the knowledge representation ?

.....

.....

.....

.....

How the knowledge representation is done through Semantic Network ?

.....

.....

.....

.....

3.6 SUMMARY

In this unit, we defined a knowledge base system as a computer system used for the management and manipulation of shared knowledge. We compared a knowledge base system with a DBMS and pointed out similarities and differences. We also considered the different schemes used to represent knowledge. The semantic network, first order logic, rule based system, frames and procedural representation.

3.7 MODEL ANSWERS

Check Your Progress

1. Knowledge can be defined as the body of facts and principles accumulated by human-kind or the act, fact, or state of knowing. While this definition may be true, it is far from complete. We know that knowledge is much more than this. It is having a familiarity with language, concepts, procedures, rules, ideas, abstractions, places, customs, facts, and associations, coupled with an ability to use these notions effectively in modelling different aspects of the world. Without this ability, the facts and concepts are meaningless and therefore worthless. The meaning of knowledge is closely related to the meaning of intelligence. Intelligence requires the possession of and access of knowledge. And a characteristic of intelligent people is that they possess much knowledge.

   Thus, we can say that knowledge includes and requires the use of data and information. But it is more. It combines relationships, correlations, dependencies, and the notion of gestalt with data and information.

2. Knowledge may be declarative or procedural. Procedural knowledge is compiled knowledge related to the performance of some task. For example, the steps used to solve an algebraic equation are expressed as procedural knowledge. Declarative knowledge, on the other hand, is passive knowledge expresses as statements of facts about the world. Personnel data in a database is typical of declarative knowledge. Such data are explicit pieces of independent knowledge.

3. The following are knowledge representation techniques:

   i) Rule Based Representation

   ii) Frame Based Representation

   iii) Semantic Nets

   iv) Knowledge Representation Using Logics

4. In semantic nets, the knowledge is represented as a set of nodes and links. A node represents an object or concept and a link represents relationship between two objects (nodes).

3.8 FURTHER READING

1. An Introduction to Database Systems by Bipin C. Desai, Galgotia Publications Pvt. Ltd.