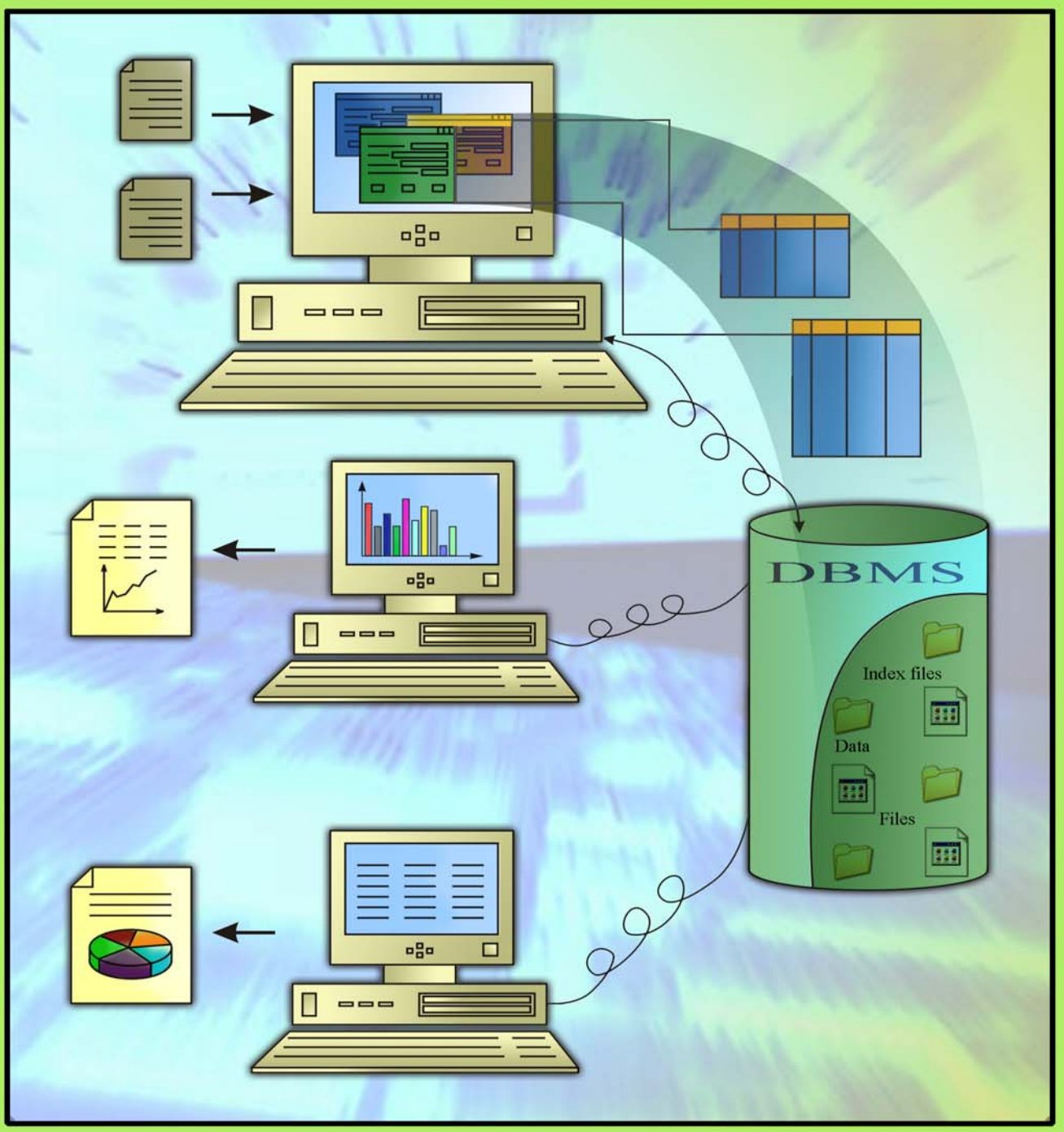
Indira Gandhi National Open University
School of Computer and Information Sciences

MCS - 023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS



The Database Management System Concepts 1

Indira Gandhi
National Open University
School of Computer &
Information Sciences

MCS-023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

Block

1

THE DATABASE MANAGEMENT SYSTEM CONCEPTS

Programme / Course Design Committee

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Prof. M. Balakrishnan, IIT, Delhi
Prof. Harish Karnick, IIT, Kanpur
Prof. C. Pandurangan, IIT, Madras
Dr. Om Vikas, Sr. Director,
Ministry of CIT, Delhi
Prof. P. S. Grover, Sr. Consultant,
SOCIS, IGNOU

Faculty of School of Computer and Information Sciences

Shri Shashi Bhushan
Shri Akshay Kumar
Prof. Manohar Lal
Shri V.V. Subrahmanyam
Shri P.Venkata Suresh

Block Preparation Team

Mr.Milind Mahajani (Content Editor)
Software Consultant
New Delhi

Ms.Ranjana Sharma
Ansal Institute of Technology
Gurgaon

Dr. Archana Singhal
GGSIP University
New Delhi

Shri Akshay Kumar
IGNOU

Shri V.V.Subrahmanyam
IGNOU

Prof. (Retd) A.K.Verma (Language Editor)
New Delhi

Course Coordinator: Shri Akshay Kumar

Block Production Team

Shri T.R. Manoj, Section Officer (Pub.) and Shri H.K. Som, Consultant

June, 2005

ISBN—81-266-1825-6

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

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Director, SOCIS.

COURSE INTRODUCTION

Database systems are pervasive. They are present in every segment of commercial, academic and virtual world. They are required as the backbone of any information system, enterprise resource planning, research activities and other activity that require permanence of data storage. Database management systems manage data more efficiently and effectively. Presently almost all popular commercially available database management system support relational model although many extensions of this model have been in existence. Some of these DBMSs have been extended to include many advanced features such as object-oriented support for graphical objects, vary large objects etc.

This course is an attempt to provide you with the basic information about relational database management system and their development. This course also provides the basic conceptual background necessary to design and develop simple database system. Some of the basic objectives of the course are to:

- provide an introduction to the needs of DBMS
- Describe the features of Relational and ER models
- Write SQL queries, create forms and queries
- Draw ER diagrams and design databases
- Discuss the features for Recovery, Concurrency and Security in DBMS.

The course is divided into four blocks.

Block 1 covers the basic concepts of RDBMS including the three level database Architecture, the two most important models for databases; the relational and entity relationship models, the integrity requirements of databases, issues relating to normalisation of database and basics about the conventional file organisation.

Block 2 is an attempt to provide you details on the structured query language (SQL), the standard query language of commercial DBMS. In addition, the concepts of Database Recovery, transaction management and security have also been covered in this Block. This block also introduces the concepts of the distributed and client server database.

Block 3 and **Block 4** are oriented towards development of a database system using DBMS software. Before, reading these blocks, you must thoroughly go through the concepts given in block 1 and block 2 as they will be required for any relational database management system application development. Block 3 focuses on database development steps involving creation of tables, relationships among tables, forms, reports and queries using the tools of a DBMS. Block 4 on the other hand highlights the complete process of development of a Database system for an organisation. It also briefs the process of the system development.

RDBMS technologies are growing at a very fast pace. You must keep up pace with it. Specially in the client-server levels of databases, server side will be the designs you undertake here, whereas client side development may involve various tools that may be available with the DBMS such as report writers, form designers, ER diagramming tools etc. You must try to practice on available DBMSs on Linux and Windows operating systems.

Further Readings to the Course:

1. Database Management system, 4th edition, R Elmasri, Shamkant Navate, Pearson.
2. Database Management System by Korth and Silberschaz,3rd/4th edition Tata McGraw Hill.
3. Database Management System, 7th edition 2003, C.J. Date, Addison Wesley.
4. Database Management System by Bipin Desai, BPB.
5. Database Systems, 3rd edition, Thomas Connolly, C.Begg, Pearson.
6. Database Systems, The Complete Book, Hector Garcia, Molina, Ullman et al.
7. Reference Manuals of Commercial RDBMS and VB or any Introductory book on VB.

BLOCK INTRODUCTION

This being the first block of the course, an attempt has been made to define and consolidate the basic concepts relating to a Database Management System. Please note that this block along with Block 2 provides the backbone for your practical implementations later, therefore, must be given maximum attention. One must be clear about these basic concepts in order to use a lot of functions and facilities which does exist in an Relational Database Management System (RDBMS). This block also visits the concepts relating to relational model, database design models, integrity control, normalisation and various other principles which will be extremely useful from the viewpoint of database design.

The block is divided into four units.

Unit 1 defines the basic concepts relating to RDBMS. One of the main concepts defined in the block is the three level database architecture. This is the basic building block for any database. It highlights the implementation of one of the most important concept or rather advantage of database system: the data independence. We have also discussed in brief about the client server database architecture.

Unit 2 focuses on the conceptual models of data. The key models discussed in this Unit are the Relational Model and ER Model. The Relational Model is a mathematically proven model of implementation and is the basis of RDBMS. On the other hand the ER model is the primary model for logical database design in the industry.

Unit 3 describes the key concepts with respect to database integrity and the principles of normalisation. The idea of integrity is to have consistent data in a database system that does not violate any constraint. Normalisation also attempts to help a database designer in minimising redundancies in the database.

Unit 4 tries to explain the basic file organisation concepts with respect to databases. One of the key concepts discussed in this unit is “Index”. The unit explains how an index helps in faster access of data from a file.

UNIT 1 BASIC CONCEPTS

Structure **Page Nos.**

1.0 INTRODUCTION

Databases and database systems have become an essential part of our everyday life. We encounter several activities that involve some interaction with a database almost daily. The examples include deposit and/or withdrawal from a bank, hotel, airline or railway reservation, accessing a computerised library, order a magazine subscription from a publisher, purchase items from supermarkets. In all the above cases a database is accessed. These may be called **Traditional Database Applications.** In these types of databases, the information stored and accessed is textual or numeric. However, with advances in technology in the past few years, different databases have been developed such as **Multimedia Databases** that store pictures, video clips and sound messages; **Geographical Information Systems (GIS)** that can store maps, weather data and satellite images, etc., and Real time databases that can control industrial and manufacturing processes. In this unit, we will be introducing the concepts involved in the Database Management System.

1.1 OBJECTIVES

After going through this unit you should be able to:

- describe the File Based system and its limitations;
- describe the structure of DBMS;
- define the functions of DBA;
- explain the three-tier architecture of DBMS, and
- identify the need for three-tier architecture.

1.2 NEED FOR A DATABASE MANAGEMENT SYSTEM

A Database is an organised, persistent collection of data of an organisation. The database management system manages the database of an enterprise. But why do we need the database management system? To describe it, let us first discuss the alternative to it, that is the file-based system.

1.2.1 The File Based System

File based systems are an early attempt to computerise the manual filing system. For example, a manual file can be set up to hold all the correspondence relating to a particular matter as a project, product, task, client or employee. In an organisation there could be many such files which may be labeled and stored. The same could be done at homes where file relating to bank statements, receipts, tax payments, etc., could be maintained.

What do we do to find information from these files? For retrieval of information, the entries could be searched sequentially. Alternatively, an indexing system could be used to locate the information.

The manual filing system works well when the number of items to be stored is small. It even works quite well when the number of items stored is quite large and they are only needed to be stored and retrieved. However, a manual file system crashes when cross-referencing and processing of information in the files is carried out. For example, in a university a number of students are enrolled who have the options of doing various courses. The university may have separate files for the personal details of students, fees paid by them, the number and details of the courses taught, the number and details of each faculty member in various departments. Consider the effort to answer the following queries.

- Annual fees paid by the students of Computer science department.
- Number of students requiring transport facility from a particular area.
- This year's turnover of students as compared to last year.
- Number of students opting for different courses from different departments.

Please refer to *Figure 1*. The answer to all the questions above would be cumbersome and time consuming in the file based system.

1.2.2 Limitations of File Based System

The file-based system has certain limitations. The limitations are listed as follows:

- **Separation and isolation of data**: When the data is stored in separate files it becomes difficult to access. It becomes extremely complex when the data has to be retrieved from more than two files as a large amount of data has to be searched.
- **Duplication of data:** Due to the decentralised approach, the file system leads to uncontrolled duplication of data. This is undesirable as the duplication leads to wastage of a lot of storage space. It also costs time and money to enter the data more than once. For example, the address information of student may have to be duplicated in bus list file data (*Figure 1*).
- **Inconsistent Data:** The data in a file system can become inconsistent if more than one person modifies the data concurrently, for example, if any student

changes the residence and the change is notified to only his/her file and not to bus list. Entering wrong data is also another reason for inconsistencies.

- **Data dependence:** The physical structure and storage of data files and records are defined in the application code. This means that it is extremely difficult to make changes to the existing structure. The programmer would have to identify all the affected programs, modify them and retest them. This characteristic of the File Based system is called **program data dependence.**

- **Incompatible File Formats:** Since the structure of the files is embedded in application programs, the structure is dependent on application programming languages. Hence the structure of a file generated by COBOL programming language may be quite different from a file generated by 'C' programming language. This incompatibility makes them difficult to process jointly. The application developer may have to develop software to convert the files to some common format for processing. However, this may be time consuming and expensive.

- **Fixed Queries:** File based systems are very much dependent on application programs. Any query or report needed by the organisation has to be developed by the application programmer. With time, the type and number of queries or reports increases. Producing different types of queries or reports is not possible in File Based Systems. As a result, in some organisations the type of queries or reports to be produced is fixed. No new query or report of the data could be generated.

Besides the above, the maintenance of the File Based System is difficult and there is no provision for security. Recovery is inadequate or non-existent.

Figure 1: File based system versus database system

1.2.3 The Database Approach

In order to overcome the limitations of a file system, a new approach was required. Hence a database approach emerged. *A database is a persistent collection of logically related data.* The initial attempts were to provide a centralised collection of data. A database has a self-describing nature. It contains not only the data but also the complete definition of the database structure and constraints, which are stored in a system catalog. A DBMS manages this data. It allows data sharing and integration of data of an organisation in a single database. *DBMS controls access to this data and thus needs to provide features for database creation, data manipulation such as data value modification, data retrieval, data integrity and security etc*. Let us describe some of the advantages of the database approach.

The database approach has many advantages. Let us discuss these in more detail.

Reduction of Redundancies

In a file processing system, each user group maintains its own files resulting in a considerable amount of redundancy of the stored data. This results in wastage of storage space but more importantly may result in data inconsistencies. Also, the same data has to be updated more than once resulting in duplication of effort. The files that represent the same data may become inconsistent as some may be updated whereas others may not be.

In database approach data can be stored at a single place or with controlled redundancy under DBMS, which saves space and does not permit inconsistency.

Shared Data

A DBMS allows the sharing of database under its control by any number of application programs or users. A database belongs to the entire organisation and is shared by all authorised users (may not be the complete data, why?). This scheme can be best explained with the help of a logical diagram (*Figure 2*). New applications can be built and added to the current system and data not currently stored can be stored.

Data Independence

In the file-based system, the descriptions of data and logic for accessing the data are built into each application program making the program more dependent on data. A change in the structure of data may require alterations to programs. Database Management systems separates data descriptions from data. Hence it is not affected by changes. This is called Data Independence, where details of data are not exposed. DBMS provides an abstract view and hides details. For example, logically we can say that the interface or window to data provided by DBMS to a user may still be the same although the internal structure of the data may be changed. (Refer to *Figure 2*).

Improved Integrity

Data Integrity refers to validity and consistency of data. Data Integrity means that the data should be accurate and consistent. This is done by providing some checks or constraints. These are consistency rules that the database is not permitted to violate. Constraints may apply to data items within a record or relationships between records. For example, the age of an employee can be between 18 and 70 years only. While entering the data for the age of an employee, the database should check this. However, if Grades of any student are entered, the data can be erroneously entered as Grade C for Grade A. In this case DBMS will not be able to provide any check as both A and C are of the same data type and are valid values.

Efficient Data Access

DBMS utilises techniques to store and retrieve the data efficiently at least for unforeseen queries. A complex DBMS should be able to provide services to end users, where they can efficiently retrieve the data almost immediately.

Multiple User Interfaces

Since many users having varying levels of technical knowledge use a database, a DBMS should be able to provide a variety of interfaces. This includes ─

a. query language for casual users,

b. programming language interfaces for application programmers,

c. forms and codes for parametric users,

d. menu driven interfaces, and

e. natural language interfaces for standalone users, these interfaces are still not available in standard form with commercial database.

Figure 2: User interaction to DBMS

Representing complex relationship among data

A database may include varieties of data interrelated to each other in many ways. A DBMS must have the capability to represent a variety of relationships among the data as well as to retrieve and update related data easily and efficiently.

Improved Security

Data is vital to any organisation and also confidential. In a shared system where multiple users share the data, all information should not be shared by all users. For example, the salary of the employees should not be visible to anyone other than the department dealing in this. Hence, database should be protected from unauthorised users. This is done by Database Administrator (DBA) by providing the usernames and passwords only to authorised users as well as granting privileges or the type of operation allowed. This is done by using security and authorisation subsystem. Only authorised users may use the database and their access types can be restricted to only retrieval, insert, update or delete or any of these. For example, the Branch Manager of any company may have access to all data whereas the Sales Assistant may not have access to salary details.

Improved Backup and Recovery

A file-based system may fail to provide measures to protect data from system failures. This lies solely on the user by taking backups periodically. DBMS provides facilities for recovering the hardware and software failures. A backup and recovery subsystem is responsible for this. In case a program fails, it restores the database to a state in which it was before the execution of the program.

Support for concurrent transactions

A transaction is defined as the unit of work. For example, a bank may be involved in a transaction where an amount of Rs.5000/- is transferred from account X to account Y. A DBMS also allows multiple transactions to occur simultaneously.

☞ Check Your Progress 1

1) What is a DBMS?

……

……

……

……

……

2) What are the advantages of a DBMS?

……

……

……

……

……..

3) Compare and contrast the traditional File based system with Database approach.

……

……

……

……

……..

1.3 THE LOGICAL DBMS ARCHITECTURE

Database Management Systems are very complex, sophisticated software applications that provide reliable management of large amounts of data. To describe general database concepts and the structure and capabilities of a DBMS better, the architecture of a typical database management system should be studied.

There are two different ways to look at the architecture of a DBMS: the *logical DBMS architecture* and the *physical DBMS architecture*. The logical architecture deals with the way data is stored and presented to users, while the physical architecture is concerned with the software components that make up a DBMS.

1.3.1 Three Level Architecture of DBMS or Logical DBMS Architecture

The logical architecture describes how data in the database is perceived by users. It is not concerned with how the data is handled and processed by the DBMS, but only with how it looks. The method of data storage on the underlying file system is not revealed, and the users can manipulate the data without worrying about where it is located or how it is actually stored. This results in the database having different levels of abstraction.

The majority of commercial Database Management S ystems available today are based on the ANSI/SPARC generalised DBMS architecture, as proposed by the ANSI/SPARC Study Group on Data Base Management Systems. Hence this is also called as the ANSI/SPARC model. It divides the system into three levels of abstraction: the *internal* or *physical level*, the *conceptual level*, and the *external* or *view level*. The diagram below shows the logical architecture for a typical DBMS.

The External or View Level

The external or view level is the highest level of abstraction of database. It provides a window on the conceptual view, which allows the user to see only the data of interest to them. The user can be either an application program or an end user. There can be many external views as any number of external schema can be defined and they can overlap each other. It consists of the definition of logical records and relationships in the external view. It also contains the methods for deriving the objects such as entities, attributes and relationships in the external view from the Conceptual view.

Figure 3: Logical DBMS Architecture

The Conceptual Level or Global level

The conceptual level presents a logical view of the entire database as a unified whole. It allows the user to bring all the data in the database together and see it in a consistent manner. Hence, there is only one conceptual schema per database. The first stage in the design of a database is to define the conceptual view, and a DBMS provides a data definition language for this purpose. It describes all the records and relationships included in the database.

The data definition language used to create the conceptual level must not specify any physical storage considerations that should be handled by the physical level. It does not provide any storage or access details, but defines the information content *only*.

The Internal or Physical Level

The collection of files permanently stored on secondary storage devices is known as the physical database. The physical or internal level is the one closest to physical storage, and it provides a low-level description of the physical database, and an interface between the operating systems file system and the record structures used in higher levels of abstraction. It is at this level that record types and methods of storage are defined, as well as how stored fields are represented, what physical sequence the stored records are in, and what other physical structures exist.

1.3.2 Mappings between Levels and Data Independence

The three levels of abstraction in the database do not exist independently of each other. There must be some correspondence, or mapping, between the levels. There are two types of mappings: the conceptual/internal mapping and the external/conceptual mapping.

The conceptual/internal mapping lies between the conceptual and internal levels, and defines the correspondence between the records and the fields of the conceptual view and the files and data structures of the internal view. *If the structure of the stored database is changed, then the conceptual/ internal mapping must also be changed accordingly so that the view from the conceptual level remains constant.* It is this mapping that provides physical data independence for the database. For example, we may change the internal view of student relation by breaking the student file into two files, one containing enrolment, name and address and other containing enrolment, programme. However, the mapping will make sure that the conceptual view is restored as original. The storage decision is primarily taken for optimisation purposes.

The external/conceptual view lies between the external and conceptual levels, and defines the correspondence between a particular external view and the conceptual view. Although these two levels are similar, some elements found in a particular external view may be different from the conceptual view. For example, several fields can be combined into a single (virtual) field, which can also have different names from the original fields. If the structure of the database at the conceptual level is changed, then the external/conceptual mapping must change accordingly so that the view from the external level remains constant. It is this mapping that provides logical data independence for the database. For example, we may change the student relation to have more fields at conceptual level, yet this will not change the two user views at all.

It is also possible to have another mapping, where one external view is expressed in terms of other external views (this could be called an external/external mapping). This is useful if several external views are closely related to one another, as it allows you to avoid mapping each of the similar external views directly to the conceptual level.

1.3.3 The need for three level architecture

The objective of the three level architecture is to separate each user's view of the database from the way the database is physically represented.

- **Support of multiple user views:** Each user is able to access the same data, but have a different customized view of the data. Each user should be able to change the way he or she views the data and this change should not affect other users.
- **Insulation between user programs and data that does not concern them:** Users should not directly deal with physical storage details, such as indexing or hashing. The user's interactions with the database should be independent of storage considerations.

Insulation between conceptual and physical structures

It can be defined as:

1. The Database Administrator should be able to change the storage structures without affecting users' views.
2. The internal structure of the database should be unaffected by the changes to the physical aspects of the storage, such as changing to a new storage device.
3. The DBA should be able to change the conceptual structure of the database without affecting all users.

1.4 PHYSICAL DBMS ARCHITECTURE

The physical architecture describes the software components used to enter and process data, and how these software components are related and interconnected. Although it is not possible to generalise the component structure of a DBMS, it is possible to identify a number of key functions which are common to most database management systems. The components that normally implement these functions are shown in *Figure 4*, which depicts the physical architecture of a typical DBMS.

Figure 4: DBMS Structure

Based on various functions, the database system may be partitioned into the following modules. Some functions (for example, file systems) may be provided by the operating system.

1.4.1 DML Precompiler

All the Database Management systems have two basic sets of Languages – Data Definition Language (DDL) that contains the set of commands required to define the format of the data that is being stored and Data Manipulation Language (DML) which defines the set of commands that modify, process data to create user definable output. The DML statements can also be written in an application program. The DML precompiler converts DML statements (such as SELECT…FROM in Structured Query Language (SQL) covered in Block 2) embedded in an application program to normal procedural calls in the host language. The precompiler interacts with the query processor in order to generate the appropriate code.

1.4.2 DDL Compiler

The DDL compiler converts the data definition statements (such as CREATE TABLE …. in SQL) into a set of tables containing metadata tables. These tables contain

information concerning the database and are in a form that can be used by other components of the DBMS. These tables are then stored in a system catalog or data dictionary.

1.4.3 File Manager

File manager manages the allocation of space on disk storage. It establishes and maintains the list of structures and indices defined in the internal schema that is used to represent information stored on disk. However, the file manager does not directly manage the physical output and input of data. It passes the requests on to the appropriate access methods, which either read data from or write data into the system buffer or cache. 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.

1.4.4 Database Manager

It is the interface between low-level data, application programs and queries. Databases typically require a large amount of storage space. It is stored on disks, as the main memory of computers cannot store this information. 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 the 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 responsible for interfacing with the database file system to the user queries. 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. 1000/-). Similarly the number of holidays per year an employee may be having should not exceed 8 days. These constraints must be 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 software 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.

The above functions are achieved through the database manager. The major components of a database manager are:

- **Authorisation control:** This module checks that the user has necessary authorisation to carry out the required function.
- **Command Processor:** Once the system has checked that the user has authority to carry out the operation, control is passed to the command processor, which converts commands to a logical sequence of steps.
- **Integrity checker:** For an operation that changes the database, the integrity checker checks that the requested operation satisfies all necessary integrity constraints such as key constraints.
- **Query Optimiser:** This module determines an optimal strategy for the query execution.
- **Transaction Manager:** This module performs the required processing of operations of various transactions. The transaction manager maintains tables of authorisation Concurrency. The DBMS may use authorisation tables to allow the transaction manager to ensure that the user has permission to execute the desired operation on the database. The authorisation tables can only be modified by properly authorised user commands, which are themselves checked against the authorisation tables.

Figure 5: Components of Database Manager

- **Scheduler:** This module is responsible for ensuring that concurrent operations or transactions on the database proceed without conflicting with one another. It controls the relative order in which transaction operations are executed. A database may also support concurrency control tables to prevent conflicts when simultaneous, conflicting commands are executed. The DBMS checks the concurrency control tables before executing an operation to ensure that the data used by it is not locked by another statement.

- **Recovery Manager:** This module ensures that the database remains in a consistent state in the presence of failures. It is responsible for transaction commit and abort, that is success or failure of transaction.

- **Buffer Manager:** This module is responsible for the transfer of data between main memory and secondary storage, such as disk and tape. The recovery manager and the buffer manager are sometimes collectively referred to as *data manager*. The buffer manager is sometimes known as *cache manager*.

1.4.5 Query Processor

The query language processor is responsible for receiving query language statements and changing them from the English-like syntax of the query language to a form the DBMS can understand. The query language processor usually consists of two separate parts: the parser and the query optimizer.

The parser receives query language statements from application programs or command-line utilities and examines the syntax of the statements to ensure they are correct. To do this, the parser breaks a statement down into basic units of syntax and examines them to make sure each statement consists of the proper component parts. If the statements follow the syntax rules, the tokens are passed to the query optimizer.

The query optimiser examines the query language statement, and tries to choose the best and most efficient way of executing the query. To do this, the query optimizer will generate several query plans in which operations are performed in different orders, and then try to estimate which plan will execute most efficiently. When making this estimate, the query optimizer may examine factors such as: CPU time, disk time, network time, sorting methods, and scanning methods.

1.4.6 Database Administrator

One of the main reasons for having the database management system is to have control of both data and 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 defines the global view or conceptual level of the database. The DBA also 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 due to changes 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 conceptual and external levels, are also defined by the DBA. 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.

Thus, the functions of DBA are:

- **Schema definition:** Creation of the original database schema is accomplished by writing a set of definitions 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 is accomplished by writing a set of definitions which are translated by the data storage and definition language compiler.
- **Schema and Physical organisation modification:** DBA involves 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 definitions 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:** DBA allows the granting of different types of authorisation for data access to the various users of the database.
- **Integrity constraint specification:** The DBA specifies the constraints. These constraints are kept in a special system structure, the data dictionary that is consulted by the database manager prior to any data manipulation. Data Dictionary is one of the valuable tools that the DBA uses to carry out data administration.

1.4.7 Data files Indices and Data Dictionary

The data is stored in the data files. The indices are stored in the index files. Indices provide fast access to data items. For example, a book database may be organised in the order of Accession number, yet may be indexed on Author name and Book titles.

Data Dictionary: A Data Dictionary stores information about the structure of the database. It is used heavily. Hence a good data dictionary should have a good design and efficient implementation. It is seen that when a program becomes somewhat large in size, keeping track of all the available names that are used and the purpose for which they were used becomes more and more difficult. After a significant time if the same or another programmer has to modify the program, it becomes extremely difficult.

The problem becomes even more difficult when the number of data types that an organisation has in its database increases. The data of an organisation is a valuable corporate resource and therefore some kind of inventory and catalog 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. A dictionary 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 items, how they fit into the data structure and how they relate to other entities in the database. In DBMS, the data dictionary stores 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 is also recorded in Data dictionary.

The Database administrator (DBA) uses the data dictionary in every phase of a database life cycle, starting from the 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 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. A data dictionary is a road map which guides users to access information within a large database.

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.
5. Statistics about database and its usage including number of records, etc.

A data dictionary is implemented as a database so that users can query its contents.

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.

☞ Check Your Progress 2

1) What are the major components of Database Manager?

……

……

……

2) Explain the functions of the person who has the control of both data and programs accessing that data.

……

……

1.5 COMMERCIAL DATABASE ARCHITECTURE

At its most basic level the DBMS architecture can be broken down into two parts: the *back end* and the *front end*.

The back end is responsible for managing the physical database and providing the necessary support and mappings for the internal, conceptual, and external levels described in a later section. Other benefits of a DBMS, such as security, integrity, and access control, are also the responsibility of the back end.

The front end is really just any application that runs on top of the DBMS. These may be applications provided by the DBMS vendor, the user, or a third party. The user interacts with the front end, and may not even be aware that the back end exists. This interaction is done through Applications and Utilities which are the main interface to the DBMS for most users.

There are three main sources of applications and utilities for a DBMS:

a. *Vendor applications and utilities* are provided for working with or maintaining the database, and usually allow users to create and manipulate a database without the need to write custom applications.

b. *User applications* are generally custom-made application programs written for a specific purpose using a conventional programming language. This programming language is coupled to the DBMS query language through the *application program interface (API)*. This allows the user to utilise the power of the DBMS query language with the flexibility of a custom application.

c. *Third party applications* may be similar to those provided by the vendor, but with enhancements, or they may fill a perceived need that the vendor hasn't created an application for. They can also be similar to user applications, being written for a specific purpose they think a large majority of users will need.

The most common applications and utilities used with a database can be divided into several well-defined categories. These are:

- **Command Line Interfaces:** These are character-based, interactive interfaces that let you use the full power and functionality of the DBMS query language directly. They allow you to manipulate the database and perform ad-hoc queries and see the results immediately. They are often the only method of exploiting the full power of the database without creating programs using a conventional programming language.
- **Graphical User Interface (GUI) tools:** These are graphical, interactive interfaces that hide the complexity of the DBMS and query language behind an intuitive, easy to understand, and convenient interface. This allows casual users the ability to access the database without having to learn the query language, and it allows advanced users to quickly manage and manipulate the database without the trouble of entering formal commands using the query language. However, graphical interfaces usually do not provide the same level of functionality as a command line interface because it is not always possible to implement all commands or options using a graphical interface.
- **Backup/Restore Utilities:** These are designed to minimise the effects of a database failure and ensure a database is restored to a consistent state if a failure does occur. Manual backup/restore utilities require the user to initiate the backup, while automatic utilities will back up the database at regular intervals without any intervention from the user. Proper use of a backup/restore utility allows a DBMS to recover from a system failure correctly and reliably.
- **Load/Unload Utilities:** These allow the user to unload a database or parts of a database and reload the data on the same machine, or on another machine in a different location. This can be useful in several situations, such as for creating backup copies of a database at a specific point in time, or for loading data into a new version of the database or into a completely different database. These load/unload utilities may also be used for rearranging the data in the database to improve performance, such as clustering data together in a particular way or reclaiming space occupied by data that has become obsolete.
- **Reporting/Analysis Utilities:** These are used to analyse and report on the data contained in the database. This may include analysing trends in data, computing values from data, or displaying data that meets some specified criteria, and then displaying or printing a report containing this information.

1.6 DATA MODELS

After going through the database architecture, let us now dwell on an important question: how is the data organised in a database? There are many basic structures that exist in a database system. They are called the database models. A database model defines

- The logical data structure
- Data relationships

- Data consistency constraints.

The following Table defines various types of Data Models

Model Type	Examples
Object-based Models: Use objects as key data representation components.	• Entity-Relationship Model: It is a collection of real world objects called entities and their relationships. It is mainly represented in graphical form using E-R diagrams. This is very useful in Database design. These have been explained in Unit 2 of this Block 1. • Object-Oriented Model: Defines the database as a collection of objects that contains both data members/values and operations that are allowed on the data. The interrelationships and constraints are implemented through objects, links and message passing mechanisms. Object-Models are useful for databases where data interrelationship are complex, for example, Computer Assisted Design based components.
Record based Logical Models: Use records as the key data representation components	Relational Model: It represents data as well as relationship among data in the form of tables. Constraints are stored in a meta-data table. This is a very simple model and is based on a proven mathematical theory. This is the most widely used data base model and will be discussed in more detail in the subsequent units. Network Model: In this model data is represented as records and relationship as links. A simple network model example is explained in *Figure 6*. It shows a sample diagram for such a system. This model is a very good model as far as conceptual framework is concerned but is nowadays not used in database management systems.
Hierarchical Data Representation Model	Hierarchical Model: It defines data as and relationships through hierarchy of data values. *Figure 7* shows an example of hierarchical model. These models are now not used in commercial DBMS products.

Figure 6: An example of Network Model

Figure 7: An example of Hierarchical Model

☞ Check your Progress 3

State whether the following are **True** or **False.**

	T	F
1) The external schema defines how and where data are organised in physical data storage.	☐	
2) A schema separates the physical aspects of data storage from the logical aspects of data representation.	☐	
3) The conceptual schema defines a view or views of the database for particular users.	☐	
4) A collection of data designed to be used by different people is called a database.	☐	
5) In a database, the data are stored in such a fashion that they are independent of the programs of people using the data.	☐	
6) Using a database redundancy can be reduced.	☐	
7) The data in a database cannot be shared.	☐	
8) Security restrictions are impossible to apply in a database.	☐	
9) In a database data integrity can be maintained.	☐	
10) Independence means that the three levels in the schema (internal, conceptual and external) should be independent of each other so that the changes in the schema at one level should not affect the other levels.	☐	

1.7 SUMMARY

Databases and database systems have become an essential part of our everyday life. We encounter several activities that involve some interaction with a database almost daily. File based systems were an early attempt to computerise the manual filing system. This system has certain limitations. In order to overcome the limitations of file-based system, a new approach, a database approach, emerged. A database is a collection of logically related data. This has a large number of advantages over the file-based approach. These systems are very complex, sophisticated software applications that provide reliable management of large amounts of data.
There are two different ways to look at the architecture of a DBMS: the *logical DBMS architecture* and the *physical DBMS architecture*. The logical architecture deals with

the way data is stored and presented to users, while the physical architecture is concerned with the software components that make up a DBMS.

The physical architecture describes the software components used to enter and process data, and how these software components are related and interconnected. At its most basic level the physical DBMS architecture can be broken down into two parts: the *back end* and the *front end*.

The logical architecture describes how data in the database is perceived by users. It is not concerned with how the data is handled and processed by the DBMS, but only with how it looks. The method of data storage on the underlying file system is not revealed, and the users can manipulate the data without worrying about where it is located or how it is actually stored. This results in the database having different levels of abstraction such as the *internal* or *physical level*, the *conceptual level*, and the *external* or *view level*. The objective of the three level architecture is to separate each user's view of the database from the way the database is physically represented.

Finally, we have a brief introduction to the concepts of database Models.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) DBMS manages the data of an organisation. It allows facilities for defining, updating and retrieving data of an organisation in a sharable, secure and reliable way.

2)
- Reduces redundancies
- Provides environment for data independence
- Enforces integrity
- Security
- Answers unforeseen queries
- Provides support for transactions, recovery etc.

3)

File Based System	Database Approach
Cheaper	Costly
Data dependent	Data independent
Data redundancy	Controlled data redundancy
Inconsistent Data	Consistent Data
Fixed Queries	Unforeseen queries can be answered

Check Your Progress 2

1) Integrity enforcement, control of file manager, security, backup, recovery, concurrency control, etc.

2) A database administrator is normally given such controls. His/her functions are: defining database, defining and optimising storage structures, and control of security, integrity and recovery.

Check Your Progress 3

1. False 2. True 3. False 4. False 5. True 6. True 7. False 8. False 9. True 10. True

UNIT 2 RELATIONAL AND E-R MODELS

Structure **Page Nos.**

2.0 INTRODUCTION

In the first unit of this block, you have been provided with the details of the Database Management System, its advantages, structure, etc. This unit is an attempt to provide you information about relational and E-R models. The relational model is a widely used model for DBMS implementation. Most of the commercial DBMS products available in the industry are relational at core. In this unit we will discuss the terminology, operators and operations used in relational model.

The second model discussed in this unit is the E-R model, which primarily is a semantic model and is very useful in creating raw database design that can be further normalised. We discuss DBMS E-R diagrams, and their conversion to tables in this unit.

2.1 OBJECTIVES

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

- describe relational model and its advantages;
- perform basic operations using relational algebra;
- draw an E-R diagram for a given problem;
- convert an E-R diagram to a relational database and vice versa.

2.2 THE RELATIONAL MODEL

A model in database system basically defines the structure or organisation of data and a set of operations on that data. Relational model is a simple model in which database is represented as a collection of "Relations", where each relation is represented by a two dimensional table. Thus, because of its simplicity it is most commonly used. The following table represents a simple relation:

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	b-4,Modi Nagar
2	Sharad Gupta	30	Pocket 2, Mayur Vihar.
3	Vibhu Datt	36	c-2, New Delhi

Figure 1: A Sample Person Relation

Following are some of the advantages of relational model:

- **Ease of use**

  The simple tabular representation of database helps the user define and query the database conveniently. For example, you can easily find out the age of the person whose first name is "Vibhu".

- **Flexibility**

  Since the database is a collection of tables, new data can be added and deleted easily. Also, manipulation of data from various tables can be done easily using various basic operations. For example, we can add a telephone number field in the table at *Figure 1*.

- **Accuracy**

  In relational databases the relational algebraic operations are used to manipulate database. These are mathematical operations and ensure accuracy (and less of ambiguity) as compared to other models. These operations are discussed in more detail in Section 2.4.

2.2.1 Domains, Attributes Tuple and Relation

Before we discuss the relational model in more detail, let us first define some very basic terms used in this model.

Tuple

Each row in a table represents a record and is called a tuple. A table containing 'n' attributes in a record is called n-tuple.

Attribute

The name of each column in a table is used to interpret its meaning and is called an attribute. Each table is called a relation.

For example, *Figure 2* represents a relation PERSON. The columns PERSON_ID, NAME, AGE and ADDRESS are the attributes of PERSON and each row in the table represents a separate tuple (record).

Relation Name: PERSON

PERSON_ID	NAME	AGE	ADDRESS	TELEPHONE
1	Sanjay Prasad	35	b-4,Modi Nagar	011-25347527
2	Sharad Gupta	30	Pocket 2, Mayur Vihar.	023-12245678
3	Vibhu Datt	36	c-2, New Delhi	033-1601138

Figure 2: An extended PERSON relation

Domain

A domain is a set of permissible values that can be given to an attribute. So every attribute in a table has a specific domain. Values to these attributes cannot be assigned outside their domains.

In the example above if domain of PERSON_ID is a set of integer values from 1 to 1000 than a value outside this range will not be valid. Some other common domains may be age between 1 and 150. The domain can be defined by assigning a type or a format or a range to an attribute. For example, a domain for a number 501 to 999 can be specified by having a 3-digit number format having a range of values between 501 and 999. However, please note the domains can also be non-contiguous. For example, the enrolment number of IGNOU has the last digit as the check digit, thus the nine-digit enrolment numbers are non-continuous.

Relation

A relation consists of:

- Relational Schema
- Relation instance

Relational Schema:

A relational schema specifies the relation's name, its attributes and the domain of each attribute. If R is the name of a relation and A_1, $A_2 \ldots A_n$ is a list of attributes representing R then $R(A_1, A_2 \ldots A_n)$ is called a relational schema. Each attribute in this relational schema takes a value from some specific domain called Domain (Ai).

For example, the relational schema for relation PERSON as in *Figure 1* will be:

PERSON(PERSON_ID:integer, NAME: string, AGE:integer, ADDRESS:string)

Total number of attributes in a relation denotes the degree of a relation. Since the PERSON relation contains four attributes, so this relation is of degree 4.

Relation Instance or Relation State:

A relation instance denoted as **r** is a collection of tuples for a given relational schema at a specific point of time.

A relation state **r** of the relation schema R (A_1,A_2,…….A_N), also denoted by r(R) is a set of n-tuples

$r = \{t_1, t_2, \ldots\ldots\ldots\ldots t_m\}$
Where each n-tuple is an ordered list of n values
$t = \langle v_1, v_2, \ldots\ldots.., v_n \rangle$
where each v_i belongs to domain (A_i) or contains null values.

The relation schema is also called *'intension'* and relation state is also called *'extension'*.

Let us elaborate the definitions above with the help of examples:

Example 1:

RELATION SCHEMA For STUDENT:

STUDENT (RollNo: string, name: string, login: string, age: integer)

RELATION INSTANCE

STUDENT

	ROLLNO	NAME	LOGIN	AGE
t_1	3467	Shikha	Noorie_jan@yahoo	20
t_2	4677	Kanu	Golgin_atat@yahoo	20

Where t_1 = (3467, shikha, Noorie-jan@yahoo.com , 20) for this relation instance, m = 2 and n = 4.

Example 2:

RELATIONAL SCHEMA For PERSON:

PERSON (PERSON_ID: integer, NAME: string, AGE: integer, ADDRESS: string)

RELATION INSTANCE
In this instance, m = 3 and n = 4

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	b-4,Modi Nagar
2	Sharad Gupta	30	Pocket 2, Mayur Vihar.
3	Vibhu Datt	36	c-2, New Delhi

Thus current relation state reflects only the valid tuples that represent a particular state of the real world. However, Null values can be assigned for the cases where the values are unknown or missing.

Ordering of tuples

In a relation, tuples are not inserted in any specific order. **Ordering of tuples is not defined as a part of a relation definition**. However, records may be organised later according to some attribute value in the storage systems. For example, records in PERSON table may be organised according to PERSON_ID. Such data or organisation depends on the requirement of the underlying database application. However, for the purpose of display we may get them displayed in the sorted order of age. The following table is sorted by age. It is also worth mentioning here that relational model **does not allow duplicate tuples**.

PERSON

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	33	Pocket 2, Mayur Vihar.
1	Sanjay Prasad	35	b-4,Modi Nagar
3	Vibhu Datt	36	c-2, New Delhi

2.2.2 Super Keys, Candidate Keys and Primary Keys for the Relations

As discussed in the previous section ordering of relations does not matter and all tuples in a relation are unique. However, can we uniquely identify a tuple in a relation? Let us discuss the concepts of keys that are primarily used for the purpose as above.

What Are Super Keys?

A **super key** is an attribute or set of attributes used to identify the records uniquely in a relation.

For Example, in the Relation PERSON described earlier PERSON_ID is a super key since PERSON_ID is unique for each person. Similarly (PERSON_ID, AGE) and (PERSON_ID, NAME) are also super keys of the relation PERSON since their combination is also unique for each record.

Candidate keys:

Super keys of a relation can contain extra attributes. Candidate keys are minimal super key, i.e. such a key contains no extraneous attribute. An attribute is called extraneous if even after removing it from the key, makes the remaining attributes still has the properties of a key.

The following properties must be satisfied by the candidate keys:

- A candidate key must be **unique**.
- A candidate key's value must **exist.** It cannot be null. (This is also called entity integrity rule)
- A candidate key is a minimal set of attributes.
- The value of a candidate key must be **stable.** Its value cannot change outside the control of the system.

A relation can have more than one candidate keys and one of them can be chosen as a **primary key.**

For example, in the relation PERSON the two possible candidate keys are PERSON-ID and NAME (assuming unique names in the table). PERSON-ID may be chosen as the primary key.

2.3 RELATIONAL CONSTRAINTS

There are three types of constraints on relational database that include:

- DOMAIN CONSTRAINT
- PRIMARY KEY CONSTRAINT
- INTEGRITY CONSTRAINT

2.3.1 Domain Constraint

It specifies that each attribute in a relation must contain an atomic value only from the corresponding domains. The data types associated with commercial RDBMS domains include:

1) Standard numeric data types for integer (such as short- integer, integer, long integer)
2) Real numbers (float, double precision floats)
3) Characters
4) Fixed length strings and variable length strings.

Thus, domain constraint specifies the condition that we want to put on each instance of the relation. So the values that appear in each column must be drawn from the domain associated with that column.
For example, consider the relation:

STUDENT

ROLLNO	NAME	LOGIN	AGE
4677	Kanu	Golgin_atat@yahoo.com	20
3677	Shikha	Noorie_jan@yahoo.com	20

In the relation above, AGE of the relation STUDENT always belongs to the integer domain within a specified range (if any), and not to strings or any other domain. Within a domain non-atomic values should be avoided. This sometimes cannot be checked by domain constraints. For example, a database which has area code and phone numbers as two different fields will take phone numbers as-

Area code	Phone
11	29534466

A non-atomic value in this case for a phone can be 1129534466, however, this value can be accepted by the Phone field.

2.3.2 Key Constraint

This constraint states that the key attribute value in each tuple must be unique, i.e., no two tuples contain the same value for the key attribute. This is because the value of the primary key is used to identify the tuples in the relation.

Example 3: If A is the key attribute in the following relation R than A1, A2 and A3 must be unique.

R

A	B
A1	B1
A3	B2
A2	B2

Example 4: In relation PERSON, PERSON_ID is primary key so PERSON_ID cannot be given as the same for two persons.

2.3.3 Integrity Constraint

There are two types of integrity constraints:

- Entity Integrity Constraint
- Referential Integrity Constraint

Entity Integrity Constraint:

It states that **no primary key value can be null**. This is because the primary key is used to identify individual tuple in the relation. So we will not be able to identify the records uniquely containing null values for the primary key attributes. This constraint is specified on one individual relation.

Example 5: Let R be the Table

A#	B	C
Null	B1	C1
A2	B2	C2
Null	B3	C3
A4	B4	C3
A5	B1	C5

Note:

1) '#' identifies the Primary key of a relation.

In the relation R above, the primary key has null values in the tuples t_1 & t_3. NULL value in primary key is not permitted, thus, relation instance is an invalid instance.

Referential integrity constraint

It states that the tuple in one relation that refers to another relation must refer to an existing tuple in that relation. This constraint is specified on two relations (not necessarily distinct). It uses a concept of foreign key and has been dealt with in more detail in the next unit.

Example 6:

R

A#	B	C^
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C3
A5	B1	C5

S

E	C#
E1	C1
E2	C3
E3	C5
E2	C2

Note:

1) '#' identifies the Primary key of a relation.
2) '^' identifies the Foreign key of a relation.

In the example above, the value of C^ in every R tuple is matching with the value of C# in some S tuple. If a tuple having values (A6, B2, C4) is added then it is invalid since referenced relation S doesn't include C4. Thus, it will be a violation of referential integrity constraint.

2.3.4 Update Operations and Dealing with Constraint Violations

There are three basic operations to be performed on relations:

- Insertion
- Deletion
- Update

The INSERT Operation:

The insert operation allows us to insert a new tuple in a relation. When we try to insert a new record, then any of the following four types of constraints can be violated:

- Domain constraint: If the value given to an attribute lies outside the domain of that attribute.
- Key constraint: If the value of the key attribute in new *tuple t* is the same as in the existing tuple in *relation R*.
- Entity Integrity constraint: If the primary key attribute value of new *tuple t* is given as *null*.
- Referential Integrity constraint: If the value of the foreign key in t refers to a tuple that doesn't appear in the referenced relation.

Dealing with constraints violation during insertion:

If the *insertion* violates one or more constraints, then two options are available:

- Default option: - *Insertion can be rejected and the reason of rejection can also be explained to the user by* DBMS.

- Ask the user to correct the data, resubmit, also give *the reason for rejecting the insertion*.

Example 7:

Consider the Relation PERSON of Example 2:

PERSON

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	b-4,Modi Nagar
2	Sharad Gupta	30	Pocket 2, Mayur Vihar.
3	Vibhu Datt	36	c-2, New Delhi

(1) Insert<1, 'Vipin', 20, 'Mayur Vihar'> into PERSON
Violated constraint: - Key constraint
Reason: - Primary key 1 already exists in PERSON.
Dealing: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(2) Insert<'null', 'Anurag', 25, 'Patparganj'> into PERSON
Violated constraint: - Entity Integrity constraint
Reason: - Primary key is 'null'.
Dealing: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(3) Insert<'abc', 'Suman', 25, 'IP college'> into PERSON
Violated constraint: - Domain constraint
Reason: - value of PERSON_ID is given a string which is not valid.

(4) Insert <10, 'Anu', 25, 'Patpatganj'> into PERSON
Violated constraint: - None

Note: In all first 3 cases of constraint violations above DBMS could reject the insertion.

The Deletion Operation:

Using the delete operation some existing records can be deleted from a relation. To delete some specific records from the database a condition is also specified based on which records can be selected for deletion.

Constraints that can be violated during deletion

Only one type of constraint can be violated during deletion, it is referential integrity constraint. It can occur when you want to delete a record in the table where it is referenced by the foreign key of another table. Please go through the example 8 very carefully.

Dealing with Constraints Violation

If the *deletion* violates referential integrity constraint, then three options are available:

- Default option: - *Reject the deletion*. It is the job of the DBMS to explain to the user why the deletion was rejected.
- *Attempt to cascade (or propagate) the deletion* by deleting tuples that reference the tuple that is being deleted.
- Change the value of *referencing attribute* that causes the violation.

Example 8:

Let R:

A#	B	C^
A1	B1	C1
A2	B3	C3
A3	B4	C3
A4	B1	C5

Q

C#	D
C1	D1
C3	D2
C5	D3

Note:

1) '#' identifies the Primary key of a relation.
2) '^' identifies the Foreign key of a relation.

(1) Delete a tuple with C# = 'C1' in Q.
Violated constraint: - Referential Integrity constraint

Reason: - Tuples in relation A refer to tuple in Q.
Dealing: - Options available are

1) Reject the deletion.
2) DBMS may automatically delete all tuples from relation *Q* and *S* with C # = 'C1'. This is called cascade detection.
3) The third option would result in putting NULL value in R where C1 exist, which is the first tuple R in the attribute C.

The Update Operations:

Update operations are used for modifying database values. The constraint violations faced by this operation are logically the same as the problem faced by Insertion and Deletion Operations. Therefore, we will not discuss this operation in greater detail here.

2.4 RELATIONAL ALGEBRA

Relational Algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval request. The result of retrieval is a new relation, formed from one or more relations. **These operations can be classified in two categories:**

- Basic Set Operations
  1) UNION
  2) INTERSECTION
  3) SET DIFFERENCE
  4) CARTESIAN PRODUCT

- Relational Operations
  1) SELECT
  2) PROJECT
  3) JOIN
  4) DIVISION

2.4.1 Basic Set Operation

These are the binary operations; i.e., each is applied to two sets or relations. These two relations should be union compatible except in case of *Cartesian Product*. Two relations R (A_1, A_2,….., A_n) and S (B_1, B_2,……,B_n) are said to be union compatible if they have the ***same degree n*** and domains of the corresponding attributes are also the same i.e.***Domain (Ai) = Domain (Bi) for 1<=i<=n***.

UNION

If R1 and R2 are two union compatible relations then R3 = R1 ∪ R2 is the relation containing tuples that are either in R1 or in R2 or in both.

In other words, R3 will have tuples such that R3 = {t | R1 ϶ t ∨ R2 ϶ t}.

Example 9:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R3 = R1 ∪ R2 is

Q

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A7	B7

Note: 1) Union is a commutative operation, i.e,

$R \cup S = S \cup R$.

2) Union is an associative operation, i.e.

$R \cup (S \cup T) = (R \cup S) \cup T$.

Intersection

If R1 and R2 are two union compatible functions or relations, then the result of R3 = R1 ∩R2 is the relation that includes all tuples that are in both the relations In other words, R3will have tuples such that R3 = {t | R1 ϶ t ∧ R2 ϶ t}.

Example 10:

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R3 = R1 ∩ R2 is

A	B
A1	B1
A2	B2
A4	B4

Note: 1) Intersection is a commutative operation, i.e.,

$R1 \cap R2 = R2 \cap R1$.

2) Intersection is an associative operation, i.e.,

$R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$

Set Difference

If R1 and R2 are two union compatible relations or relations then result of R3 =R1– R2 is the relation that includes only those tuples that are in R1 but not in R2.

In other words, R3 will have tuples such that $R3 = \{t \mid R1 \ni t \wedge t \notin R2\}$.

Example 11:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R1-R2 =

A	B
A3	B3

R2-R1=

A	B
A7	B7

Note: -1) Difference operation is not commutative, i.e.,

$$\mathbf{R1 - R2 \neq R2 - R1}$$

2) Difference operation is not associative, i. e.,

$$\mathbf{R1 - (R2 - R3) \neq (R1 - R2) - R3}$$

2.4.2 Cartesian Product

If R1 and R2 are two functions or relations, then the result of $R3 = R1 \times R2$ is the combination of tuples that are in R1 and R2. The product is commutative and associative.

Degree (R3) =Degree of (R1) + Degree (R2).
In other words, R3 will have tuples such that $R3 = \{t_1 \| t_2 \mid R1 \ni t_1 \wedge R2 \ni t_2\}$.

Example 12:

R1

C
C1
C2

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R3 = R1 × R2 is

A	B	C
A1	B1	C1
A1	B1	C2
A2	B2	C1
A2	B2	C2
A3	B3	C1
A3	B3	C2
A4	B4	C1
A4	B4	C2

2.4.3 Relational Operations

Let us now discuss the relational operations:

SELECT

The select operation is used to select some specific records from the databse based on some criteria. This is a unary operation mathematically denoted as σ.

Syntax:

σ **<Selection condition> (Relation)**

The Boolean expression is specified in <Select condition> is made of a number of clauses of the form:

<attribute name><comparison operator><constant value> or

<attribute name><comparison operator><attribute name>

Comparison operators in the set $\{\leq, \geq, \neq, =, <, <\}$ apply to the attributes whose domains **are *ordered value* like integer.**

Example 13:

Consider the relation PERSON. If you want to display details of persons having age less than or equal to 30 than the select operation will be used as follows:

σ **AGE <=30 (PERSON)**

The resultant relation will be as follows:

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar.

Note:

1) Select operation is commutative; i.e.,
σ <condition1> (σ <condition2> (R)) = σ <condition2> (σ <condition1> (R))
Hence, Sequence of select can be applied in any order

2) More than one condition can be applied using Boolean operators AND & OR etc.

The PROJECT operation

The project operation is used to select the records with specified attributes while discarding the others based on some specific criteria. This is denoted as Π**.**

Π **List of attribute for project (Relation)**

Example 14:

Consider the relation PERSON. If you want to display only the names of persons then the project operation will be used as follows:

Π **Name (PERSON)**

The resultant relation will be as follows:

NAME
Sanjay Prasad
Sharad Gupta
Vibhu Datt

Note: -

1) $\Pi_{<List1>}(\Pi_{<list2>}(R)) = \Pi_{<list1>}(R)$

As long as<list2> contains attributes in <list1>.

The JOIN operation

The JOIN operation is applied on two relations. When we want to select related tuples from two given relation join is used. This is denoted as ⋈. The join operation requires that both the joined relations must have at least one domain compatible attributes.

Syntax:

$R1 \bowtie_{<join\ condition>} R2$ is used to combine related tuples from two relations R1 and R2 into a single tuple.
<join condition> is of the form:
<condition>AND<condition>AND……..AND<condition>.

- Degree of Relation:

Degree ($R1 \bowtie_{<join\ condition>} R2$) <= Degree (R1) + Degree (R2).

- Three types of joins are there:

a) **Theta join**

When each condition is of the form A θ B, A is an attribute of R1 and B is an attribute of R2 and have the same domain, and θ is one of the comparison operators **{**$\leq, \geq, \neq$**, =, <, <}.**

b) **Equijoin**

When each condition appears with equality condition (=) only.

c) **Natural join (denoted by R*S)**

When two join attributes have the same name in both relations. (That attribute is called Join attribute), only one of the two attributes is retained in the join relation. The

join condition in such a case is = for the join attribute. The condition is not shown in the natural join.

Let us show these operations with the help of the following example.

Example 15:

Consider the following relations**:**

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID
100	Kanupriya	234, Saraswati Vihar.	CS1
101	Rajni Bala	120, Vasant Kunj	CS2
102	Arpita Gupta	75, SRM Apartments.	CS4

COURSE

COURSE ID	COURSE_NAME	DURATION
CS1	MCA	3yrs
CS2	BCA	3yrs
CS3	M.Sc.	2yrs
CS4	B.Sc.	3yrs
CS5	MBA	2yrs

If we want to display name of all the students along with their course details then natural join is used in the following way:

STUDENT ⋈ COURSE

Resultant relation will be as follows:

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID	COURSE_NAME	DURATION
100	Kanupriya	234, Saraswati Vihar.	CS1	MCA	3yrs
101	Rajni Bala	120, Vasant Kunj	CS2	BCA	3yrs
102	Arpita Gupta	75, SRM Apartments.	CS4	B.Sc.	3yrs

There are other types of joins like outer joins. You must refer to further reading for more details on those operations. They are also explained in Block 2 Unit 1.

The DIVISION operation:

To perform the division operation R1 ÷ R2, R2 should be a proper subset of R1. In the following example R1 contains attributes A and B and R2 contains only attribute B so R2 is a proper subset of R1. If we perform R1 ÷ R2 than the resultant relation will contain those values of A from R1 that are related to all values of B present in R2.

Example 16:

Let R1

A	B
A1	B1
A1	B2
A2	B1
A3	B1
A4	B2
A5	B1
A3	B2

If R2 R3= R1 ÷ R2

B
B1
B2

A
A1
A3

(a)

If R2 R3= R1 ÷ R2

B
B1

A
A1
A2
A3
A5

(b)

If R2 R3= R1 ÷ R2

A

B
B1
B2
B3

(c)

If R2 R3= R1 ÷ R2

B

A
A1
A2
A3
A4
A5

(d)

Figure 3: The Division Operation

Note:

Degree of relation: Degree (R ÷ S)=Degree of R – Degree of S.

☞ Check Your Progress 1

1) A database system is fully relational if it supports _____ and _____.

2) A Relation resembles a _____ a tuple resembles a _____ and an attribute resembles a _____.

3) A candidate key which is not a primary key is known as a _____key.

4) Primitive operations are union, difference, product, selection and projection. The definition of A intersects B can be_____.

5) Which one is not a traditional set operator defined on relational algebra?

   a. Union
   b. Intersection
   c. Difference
   d. Join

6) Consider the tables **S**uppliers, **P**arts, pro**J**ect and **SPJ** relation instances in the relations below. (Underline represents a key attribute. The SPJ relation have three Foreign keys: SNO, PNO and JNO.)

S

SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune
S4	Sita	Delhi
S5	Anand	Agra

P

PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Screw	White	Bombay
P4	Screw	Blue	Delhi
P5	Camera	Brown	Pune
P6	Cog	Grey	Delhi

J

JNO	JNAME	CITY
J1	Sorter	Pune
J2	Display	Bombay
J3	OCR	Agra
J4	Console	Agra
J5	RAID	Delhi
J6	EDS	Udaipur
J7	Tape	Delhi

SPJ

SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J2	400
S2	P2	J7	200
S2	P3	J3	500
S3	P3	J5	400
S3	P4	J3	500
S3	P5	J3	600
S3	P6	J4	800
S4	P6	J2	900
S4	P6	J1	100
S4	P5	J7	200
S5	P5	J5	300
S5	P4	J6	400

Using the sample data values in the relations above, tell the effect of each of the following operation:

a) UPDATE Project J7 in J setting CITY to Nagpur.
b) UPDATE part P5 in P, setting PNO to P4.
c) UPDATE supplier S5 in S, setting SNO to S8, if the relevant update rule is RESTRICT.
d) DELETE supplier S3 in S, if the relevant rule is CASCADE.
e) DELETE part P2 in P, if the relevant delete rule is RESTRICT.
f) DELETE project J4 in J, if the relevant delete rule is CASCADE.
g) UPDATE shipment S1-P1-J1 in SPJ, setting SNO to S2. (shipment S1-P1-J1 means that in SPJ table the value for attributes SNO, PNO and JNO are S1, P1 and J1 respectively)
h) UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J7.
i) UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J8
j) INSERT shipment S5-P6-J7 in SPJ.
k) INSERT shipment S4-P7-J6 in SPJ
l) INSERT shipment S1-P2-jjj (where jjj stands for a default project number).

……

……

……

7) Find the name of projects in the relations above, to which supplier S1 has supplied.

……

……

……

……

2.5 ENTITY RELATIONSHIP (ER) MODEL

Let us first look into some of the main features of ER model.

- Entity relationship model is a high-level conceptual data model.
- It allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships.
- It is widely used to develop an initial design of a database.
- It provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and precise description of information that can be easily implemented in a database system.
- It describes data as a collection of entities, relationships and attributes.

Let us explain it with the help of an example application.

An Example Database Application

We will describe here an example database application containing COLLEGE database and use it in illustrating various E-R modeling concepts.
College database keeps track of Students, faculty, Departments and Courses organised by various departments. Following is the description of COLLEGE database:

College contains various departments like Department of English, Department of Hindi, Department of Computer Science etc. Each department is assigned a unique id and name. Some faculty members are also appointed to each department and one of them works as head of the department.

There are various courses conducted by each department. Each course is assigned a unique id, name and duration.

- Faculty information contains name, address, department, basic salary etc. A faculty member is assigned to only one department but can teach various courses of other department also.
- Student's information contain Roll no (unique), Name, Address etc. A student can opt only for one course.
- Parent (gurdian) information is also kept along with each student. We keep each guardian's name, age, sex and address.

2.5.1 Entities

Let us first be aware of the question:

What are entities?

- An entity is an object of concern used to represent the things in the real world, e.g., car, table, book, etc.
- An entity need not be a physical entity, it can also represent a concept in real world, e.g., project, loan, etc.
- It represents a class of things, not any one instance, e.g., 'STUDENT' entity has instances of 'Ramesh' and 'Mohan'.

Entity Set or Entity Type:

A collection of a similar kind of entities is called an **Entity Set or entity type.**

Example 16:

For the COLLEGE database described earlier objects of concern are Students, Faculty, Course and departments. The collections of all the students entities form a entity set STUDENT. Similarly collection of all the courses form an entity set COURSE.

Entity sets need not be disjoint. For example – an entity may be part of the entity set STUDENT, the entity set FACULTY, and the entity set PERSON.

Entity identifier key attributes

An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a key attribute and its values can be used to identify each entity uniquely.

Strong entity set: The entity types containing a key attribute are called strong entity types or regular entity types.

- EXAMPLE: The Student entity has a key attribute RollNo which uniquely identifies it, hence is a strong entity.

2.5.2 Attributes

Let us first answer the question:

What is an attribute?

An attribute is a property used to describe the specific feature of the entity. So to describe an entity entirely, a set of attributes is used.

For example, a student entity may be described by the student's name, age, address, course, etc.

An entity will have a value for each of its attributes. For example for a particular student the following values can be assigned:

RollNo: 1234
Name: Supriya
Age: 18
Address: B-4, Mayapuri, Delhi.
Course: B.Sc. (H)
••

Domains:

- Each simple attribute of an entity type contains a possible set of values that can be attached to it. This is called the domain of an attribute. An attribute cannot contain a value outside this domain.
- EXAMPLE- for PERSON entity PERSON_ID has a specific domain, integer values say from 1 to 100.

Types of attributes

Attributes attached to an entity can be of various types.

Simple

The attribute that cannot be further divided into smaller parts and represents the basic meaning is called a simple attribute. For example: The 'First name', 'Last name', age attributes of a person entity represent a simple attribute.

Composite

Attributes that can be further divided into smaller units and each individual unit contains a specific meaning.

For example:-The NAME attribute of an employee entity can be sub-divided into First name, Last name and Middle name.

Single valued

Attributes having a single value for a particular entity. For Example, Age is a single valued attribute of a student entity.

Multivalued

Attributes that have more than one values for a particular entity is called a multivalued attribute. Different entities may have different number of values for these kind of attributes. For multivalued attributes we must also specify the minimum and maximum number of vales that can be attached. For Example phone number for a person entity is a multivalued attribute.

Stored

Attributes that are directly stored in the data base.

For example, 'Birth date' attribute of a person.

Derived

Attributes that are not stored directly but can be derived from stored attributes are called derived attributes. For Example, The years of services of a 'person' entity can be determined from the current date and the date of joining of the person. Similarly, total salary of a 'person' can be calculated from 'basic salary' attribute of a 'person'.

2.5.3 Relationships

Let us first define the term relationships.

What Are Relationships?

A relationship can be defined as:

- a connection or set of associations, or
- a rule for communication among entities:

Example: In college the database, the association between student and course entity, i.e., "Student opts course" is an example of a relationship.

Relationship sets

A relationship set is a set of relationships of the same type.

For example, consider the relationship between two entity sets *student* and course. Collection of all the instances of relationship opts forms a relationship set called relationship type.

Degree

The degree of a relationship type is the number of participating entity types.

The relationship between two entities is called binary relationship. A relationship among three entities is called ternary relationship.

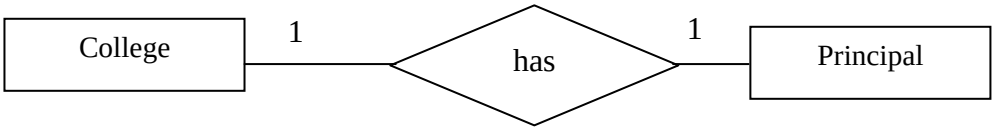
Similarly relationship among n entities is called n-ry relationship.

Relationship Cardinality

Cardinality specifies the number of instances of an entity associated with another entity participating in a relationship. Based on the cardinality binary relationship can be further classified into the following categories:

- **One-to-one**: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

Example 17: Relationship between college and principal

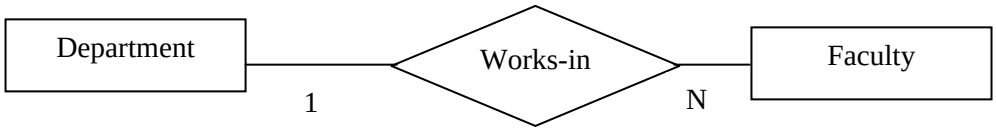


One college can have at the most one principal and one principal can be assigned to only one college.

Similarly we can define the relationship between university and Vice Chancellor.

- **One-to-many**: An entity in A is associated with any number of entities in B. An entity in B is associated with at the most one entity in A.

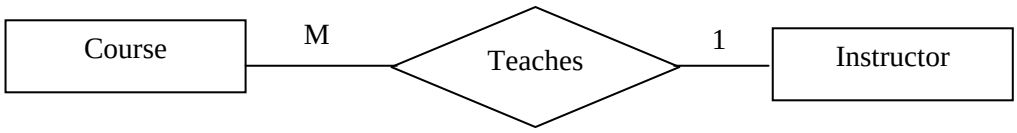
Example 18: Relationship between department and faculty.



One department can appoint any number of faculty members but a faculty member is assigned to only one department.

- **Many-to-one**: An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.

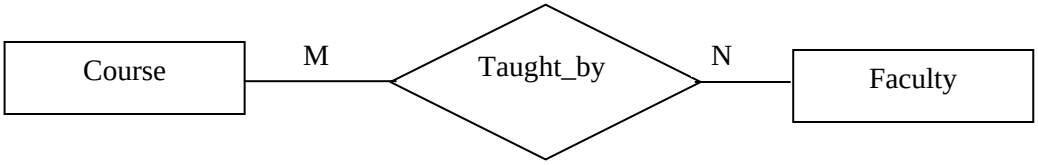
Example 19: Relationship between course and instructor. An instructor can teach various courses but a course can be taught only by one instructor. Please note this is an assumption.



Many-to-many: Entities in A and B are associated with any number of entities from each other.

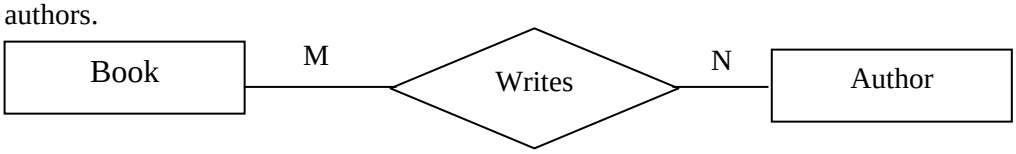
Example 20:

Taught_by Relationship between course and faculty.
One faculty member can be assigned to teach many courses and one course may be taught by many faculty members.



Relationship between book and author.

One author can write many books and one book can be written by more than one authors.



2.5.4 More about Entities and Relationships

Recursive relationships

When the same entity type participates more than once in a relationship type in different roles, the relationship types are called recursive relationships.

Participation constraints

The participation Constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. There are 2 types of participation constraints:

Total: When all the entities from an entity set participate in a relationship type, is called total participation, for example, the participation of the entity set student in the relationship set must ‘opts’ is said to be total because every student enrolled must opt for a course.

Partial: When it is not necessary for all the entities from an entity set to participate in a relationship type, it is called partial participation. For example, the participation of the entity set student in ‘represents’ is partial, since not every student in a class is a class representative.

WEAK ENTITY

Entity types that do not contain any key attribute, and hence cannot be identified independently, are called weak entity types. A weak entity can be identified uniquely only by considering some of its attributes in conjunction with the primary key attributes of another entity, which is called the identifying owner entity.

Generally a partial key is attached to a weak entity type that is used for unique identification of weak entities related to a particular owner entity type. The following restrictions must hold:

- The owner entity set and the weak entity set must participate in one to many relationship set. This relationship set is called the identifying relationship set of the weak entity set.
- The weak entity set must have total participation in the identifying relationship.

EXAMPLE: Consider the entity type dependent related to employee entity, which is used to keep track of the dependents of each employee. The attributes of dependents are: name, birth date, sex and relationship. Each employee entity is said to own the dependent entities that are related to it. However, please note that the 'dependent' entity does not exist of its own, it is dependent on the Employee entity. In other words we can say that in case an employee leaves the organisation all dependents related to him/her also leave along with. Thus, the 'dependent' entity has no significance without the entity 'employee'. Thus, it is a weak entity. The notation used for weak entities is explained in Section 2.6.

Extended E-R features:

Although, the basic features of E-R diagrams are sufficient to design many database situations. However, with more complex relations and advanced database applications, it is required to move to enhanced features of E-R models. The three such features are:

- Generalisation
- Specialisation, and
- Aggregation

In this unit, we have explained them with the help of an example. More detail on them are available in the further readings and MCS-043.

Example 21: A bank has an account entity set. The accounts of the bank can be of two types:

- Savings account
- Current account

The statement as above represents a specialisation/generalisation hierarchy. It can be shown as:

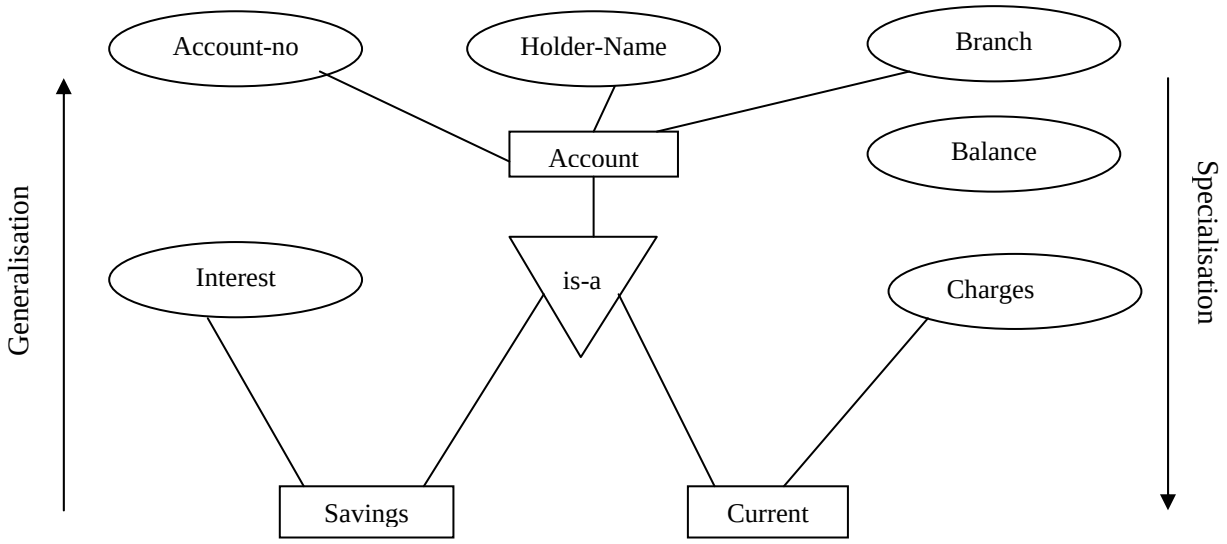


Figure 4: Generalisation and Specialisation hierarchy

But how are these diagrams converted to tables? This is shown in section 2.7.

Aggregation: One limitation of the E-R diagram is that they do not allow representation of relationships among relationships. In such a case the relationship along with its entities are promoted (aggregated to form an aggregate entity which can be used for expressing the required relationships). A detailed discussion on aggregation is beyond the scope of this unit you can refer to the further readings for more detail.

2.5.5 Defining Relationship For College Database

Using the concepts defined earlier, we have identified that strong entities in COLLEGE database are STUDENT, FACULTY, COURSE and DEPARTMENT. This database also has one weak entity called GUARDIAN. We can specify the following relationships:

1. **Head-of**, is a 1:1 relationship between FACULTY and DEPARTMENT. Participation of the entity FACULTY is partial since not all the faculty members participate in this relationship, while the participation from department side is total, since every department has one head.

2. **Works_in**, is a 1:N relationship between DEPARTMENT and FACULTY. Participation from both side is total.

3. **Opts,** is a 1:N relationship between COURSE and STUDENT. Participation from student side is total because we are assuming that each student enrolled opts for a course. But the participation from the course side is partial, since there can be courses that no student has opted for.

4. **Taught_by**, is a M: N relationship between FACULTY and COURSE, as a faculty can teach many courses and a course can be taught by many faculty members.

5. **Enrolled**, is a 1:N relationship between STUDENT and DEPARTMENT as a student is allowed to enroll for only one department at a time.

6. **Has**, is a 1:N relationship between STUDENT and GUARDIAN as a student can have more than one local guardian and one local guardian is assumed to be related to one student only. The weak entity Guardian has total participation in the relation “Has”.

So now, let us make an E-R diagram for the college database in the next section.

2.6 E-R DIAGRAM

Let us now make the E-R diagram for the student database as per the description given in the previous section.

We can also express the overall logical structure of a database using ER model **graphically** with the help of an E-R diagram.

ER diagrams are composed of:

- **rectangles** representing entity sets.
- **ellipses** representing attributes.
- **diamonds** representing relationship sets.

Figure 5: ER diagram of COLLEGE database

Please note that the relationship head-of has an attribute “Date-from”. Let us define the symbols used in the ER diagram:

Figure 6: Symbols of E-R diagrams.

2.7 CONVERSION OF ER DIAGRAM TO RELATIONAL DATABASE

For every ER diagram we can construct a relational database which is a collection of tables. Following are the set of steps used for conversion of ER diagram to a relational database.

Conversion of entity sets:

I) For each strong entity type E in the ER diagram, we create a relation R containing all the simple attributes of E. The primary key of the relation R will be one of the key attributes of R.

For example, the STUDENT, FACULTY, COURSE and DEPARTMENT tables in *Figure 7.*

STUDENT

ROLLNO: Primary Key	NAME	ADDRESS

FACULTY

ID: Primary Key	NAME	ADDRESS	BASIC_SAL

COURSE

COURSE_ID: Primary Key	COURSE_NAME	DURATION

DEPARTMENT

D_NO: Primary Key	D_NAME

Figure 7: Conversion of Strong Entities

II) For each weak entity type W in the E R Diagram, we create another relation R that contains all simple attributes of W. If E is an owner entity of W then key attribute of E is also included in R. This key attribute of R is set as a foreign key attribute of R. Now the combination of primary key attribute of owner entity type and partial key of weak entity type will form the key of the weak entity type.

Figure 8 shows the weak entity GUARDIAN, where the key field of student entity RollNo has been added.

RollNo Name (Primary Key)		**Address**	**Relationship**

Figure 8: Conversion of weak entity to table.

Conversion of relationship sets:

Binary Relationships:

I) One-to-one relationship:

For each 1:1 relationship type R in the ER diagram involving two entities E1 and E2 we choose one of entities (say E1) preferably with total participation and add primary key attribute of another entityE2 as a foreign key attribute in the table of entity (E1). We will also include all the simple attributes of relationship type R in E1 if any.
For example, the DEPARTMENT relationship has been extended to include head-Id and attribute of the relationship. Please note we will keep information in this table of only current head and Date from which s/he is the head. (Refer to *Figure 9*).

There is one Head_of 1:1 relationship between FACULTY and DEPARTMENT. We choose DEPARTMENT entity having total participation and add primary key attribute ID of FACULTY entity as a foreign key in DEPARTMENT entity named as Head_ID. Now the DEPARTMENT table will be as follows:

DEPARTMENT

D_NO	**D_NAME**	**Head_ID**	**Date-from**

Figure 9: Converting 1:1 relationship

II) One-to-many relationship:

For each 1: n relationship type R involving two entities E1 and E2, we identify the entity type (say E1) at the n-side of the relationship type R and include primary key of the entity on the other side of the relation (say E2) as a foreign key attribute in the table of E1. We include all simple attributes (or simple components of a composite attributes of R (if any) in the table of E1).

For example, the works_in relationship between the DEPARTMENT and FACULTY. For this relationship choose the entity at N side, i.e., FACULTY and add primary key attribute of another entity DEPARTMENT, i.e., DNO as a foreign key attribute in FACULTY. Please refer to *Figure 10*.

FACULTY (CONTAINS WORKS_IN RELATIONSHIP)

ID	**NAME**	**ADDRESS**	**BASIC_SAL**	**DNO**

Figure 10: Converting 1:N relationship

III) Many-to-many relationship:

For each m:n relationship type R, we create a new table (say S) to represent R. We also include the primary key attributes of both the participating entity types as a foreign key attribute in S. Any simple attributes of the m:n relationship type (or simple components of a composite attribute) is also included as attributes of S.
For example, the m: n relationship taught-by between entities COURSE and FACULTY should be represented as a new table. The structure of the table will include primary key of COURSE and primary key of FACULTY entities. (Refer to *Figure 11*).

A new table TAUGHT-BY will be created as: Primary key of TAUGHT-By table.

ID	**Course_ID**
{Primary key of FACULTY table}	{Primary key of COURSE table}

Please note that since there are no attributes to this relationship, therefore no other fields.

Figure 11: Converting N:N relationship

n-ary Relationship:

For each n-ary relationship type R where n>2, we create a new table S to represent R. We include as foreign key attributes in s the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n-ary relationship type (or simple components of complete attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. *Figure 11* is a special case of n-ary relationship: a binary relation.

Multivalued attributes:

For each multivalued attribute ‘A’, we create a new relation R that includes an attribute corresponding to plus the primary key attribute k of the relation that represents the entity type or relationship type that has as an attribute. The primary key of R is then combination of A and k.
For example, if a STUDENT entity has RollNo, Name and PhoneNumber where phone number is a multi-valued attribute then we will create a table PHONE (**RollNo, Phone-No)** where primary key is the combination. Please also note that then in the STUDENT table we need not have phoneNumber, instead it can be simply (Roll No, Name) only.

Converting Generalisation / Specialisation hierarchy to tables:

A simple rule for conversation may be to decompose all the specialised entities into tables in case they are disjoint. For example, for the *Figure 4* we can create the two tables as:

Saving_account (account-no holder-name, branch, balance, interest).

Current_account (account-no, holder-name, branch, balance, charges).

The other way might be to create tables that are overlapping for example, assuming that in the example of *Figure 4* if the accounts are not disjoint then we may create three tables:

account (account-no, holder-name, branch, balance)

saving (account-no, interest)

current (account-no, charges)

☞ Check Your Progress 2

1) A Company ABC develops applications for the clients and their own use. Thus, the company can work in "user mode" and "developer mode". Find the possible entities and relationships among the entities. Give a step by step procedure to make an E-R diagram for the process of the company when they develop an application for the client and use the diagram to derive appropriate tables.

……..…

……..…

……..…

…….

2) An employee works for a department. If the employee is a manager then s/he manages the department. As an employee the person works for the project, and the various departments of a company control those projects. An employee can have many dependents. Draw an E-R diagram for the above company. Try to find out all possible entities and relationships among them.

……..…

……..…

……..…

…….

3) A supplier located in only one-city supplies various parts for the projects to different companies located in various cities. Let us name this database as "supplier-and-parts". Draw the E-R diagram for the supplier and parts database.

……..……

……..……

……..…….

…….

2.8 SUMMARY

This unit is an attempt to provide a detailed viewpoint of data base design. The topics covered in this unit include the relational model including the representation of relations, operations such as set type operators and relational operators on relations. The E-R model explained in this unit covers the basic aspects of E-R modeling. E-R modeling is quite common to database design, therefore, you must attempt as many problems as possible from the further reading. E-R diagram has also been extended. However, that is beyond the scope of this unit. You may refer to further readings for more details on E-R diagrams.

2.9 SOLUTIONS/ ANSWERS

Check Your Progress 1

1. Relational databases and a language as powerful as relational algebra
2. File, Record, Field
3. Alternate
4. A minus (A minus B)

5. (d)
6.
   a) Accepted
   b) Rejected (candidate key uniqueness violation)
   c) Rejected (violates RESTRICTED update rule)

d) Accepted (supplier S3 and all shipments for supplier S3 in the relation SPJ are deleted)
e) Rejected (violates RESTRICTED delete rule as P2 is present in relation SPJ)
f) Accepted (project J4 and all shipments for project J4 from the relation SPJ are deleted)
g) Accepted
h) Rejected (candidate key uniqueness violation as tuple S5-P5-J7already exists in relation SPJ)
i) Rejected (referential integrity violation as there exists no tuple for J8 in relation J)
j) Accepted
k) Rejected (referential integrity violation as there exists no tuple for P7 in relation P)
l) Rejected (referential integrity violation – the default project number jjj does not exist in relation J).

7) The answer to this query will require the use of the relational algebraic operations. This can be found by selection supplies made by S1 in SPJ, then taking projection of resultant on JNO and joining the resultant to J relation. Let us show steps:

Let us first find out the supplies made by supplier S1 by selecting those tuples from SPJ where SNO is S1. The relation operator being:

$$SPJT = \sigma_{<SNO = 'S1'>} (SPJ)$$

The resulting temporary relation SPJT will be:

SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S1	P1	J4	700

Now, we take the projection of SPJT on PNO

$$SPJT2 = \Pi_{JNO}(SPJT)$$

The resulting temporary relation SPJT will be:

JNO
J1
J4

Now take natural JOIN this table with J:

RESULT = SPJT2 JOIN J

The resulting relation RESULT will be:

JNO	JNAME	CITY
J1	Sorter	Pune
J4	Console	Agra

Check Your Progress 2

1) Let us show the step by step process of development of Entity-Relationship Diagram for the Client Application system. The first two important entities of the system are **Client** and **Application**. Each of these terms represents a noun, thus, they are eligible to be the entities in the database.

But are they the correct entity types? Client and Application both are **independent** of anything else. So the entities, clients and applications form an entity set.

But how are these entities related? Are there more entities in the system?
Let us first consider the relationship between these two entities, if any. Obviously the relationship among the entities depends on interpretation of written requirements. Thus, we need to define the terms in more detail.
Let us first define the term **application.** Some of the questions that are to be answered in this regard are: Is the **Accounts Receivable (AR)** an application? Is the AR system installed at each client site regarded as a different application? Can the same application be installed more than once at a particular client site?

Before we answer these questions, do you notice that another entity is in the offering? The client **site** seems to be another candidate entity. This is the kind of thing you need to be sensitive to at this stage of the development of entity relationship modeling process.

So let use first deal with the relationship between Client and Site before coming back to Application.

Just a word of advice: "It is often easier to tackle what seems likely to prove simple before trying to resolve the apparently complex."

Each Client can have many sites, but each site belongs to one and only one client.

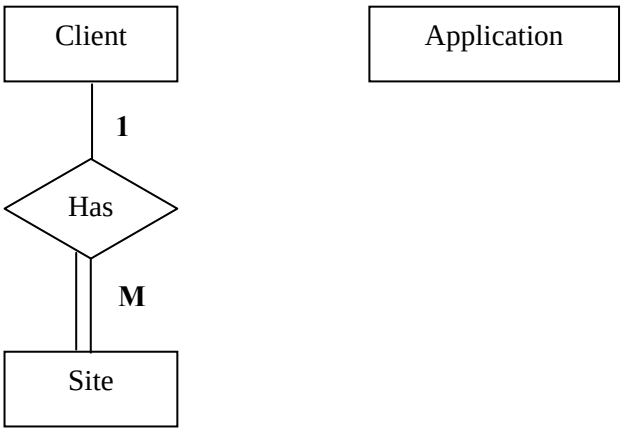


Figure 12: A One-to-Many Relationship

Now the question arises what entity type is Site? We cannot have a site without a client. If any site exists without a client, then who would pay the company? This is a good example of an existential dependency and one-to-many relationship. This is illustrated in the diagram above.

Let us now relate the entity Application to the others. Please note the following fact about this entity:

An application may be installed at many client sites. Also more than one application can be installed at these sites. Thus, there exists a many-to-many relationship between Site and Application:

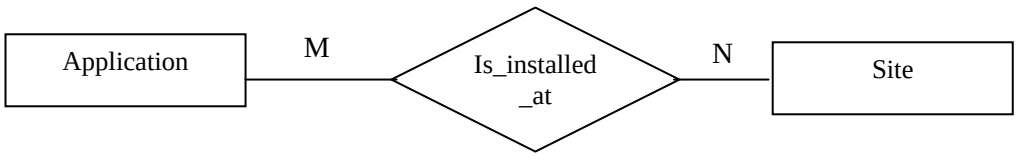


Figure 13: A Many-to-Many Relationship

However, the M: M relationship “is_installed_at” have many attributes that need to be stored with it, specifically relating to the available persons, dates, etc. Thus, it may be a good idea to promote this relationship as an Entity.

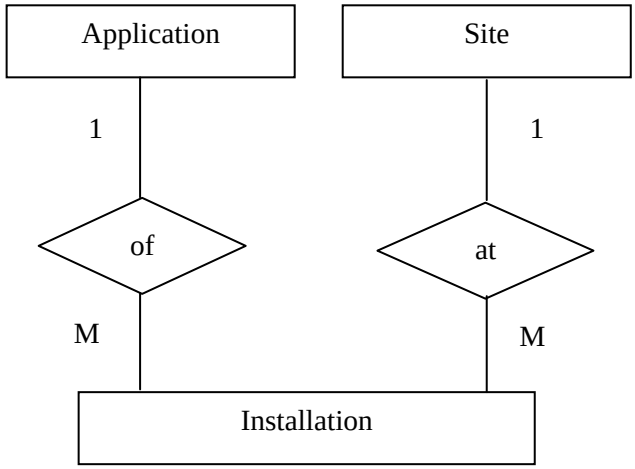


Figure 14: The INSTALLATION ENTITY

The *Figure* 14 above consists of the following relationships:

of - relationship describing installation of applications and
at - relationship describing installation at sites.

Please note that entities can arise in one of two ways. Either we draw them because they were named in the specification, or as a result of resolving a M:M relationship, as in this case. When we create a new entity in this way, we have to find a suitable name for it. This can sometimes be based on the **verb** used to describe the M:M. Thus, the statement **we can install the application at many sites** we can choose the verb install and covert it to related noun **Installation**.

But what is the type of this Installation entity?

The best person to do so is the user of this entity. By identifying type we mean to define the most effective way of uniquely identifying each record of this type. So now the questions that need to be answered are:

- How do we want to identify each Installation?
- Is an Installation independent of any other entity, that is, can an entity Installation exist without being associated with the entities Client, Site and Application?

In the present design, there cannot be an Installation until we can specify the Client, Site and Application. But since Site is existentially dependent on Client or in other words, Site is subordinate to Client, the Installation can be identified by (Client) Site (it means Client or Site) and Application. We do not want to allow more than one record for the same site and application.

But what if we also sell multiple copies of packages for a site? In such a case, we need to keep track of each individual copy (license number) at each site. So we need another entity. We may even find that we need separate entities for Application and Package. This will depend on what attributes we want to keep in each as our requirements differ in respect of each of these.

Figure 15: **Adding More Entities**

Let us define the Relationships in the Figure 15.
Has: describes that each application has one or more licenses
Buys: describes each site buys the licensed copies.

We might decide that License should be subordinate to Package: the best unique identifier for the entity could be Package ID and License serial number. We could also define License as identified by Client Site, Application and the serial number of the licensed copy. The only objection to the subordinate identification is that one of the key elements has an externally assigned value: the license numbers issued by the companies to whom the package belongs. We do not issue license numbers, and thus have no control over their length and data type (Can it be up to 15 mixed alpha-numeric characters?). Also we have no control over their uniqueness and changeability. Please note that **it is always safer to base primary keys on internally assigned values.**

What if we make License subordinate to Package, but substitute our own Copy serial number for the software manufacturer's License number? It also seems that the client site is not an essential part of the identifier, as the client is free to move the copy of a package to another site. Thus, we should definitely not make client/site part of the primary key of License.

Hence the discussion goes on till you get the final E-R diagram for the problem above. Please keep thinking and refining your reasoning. Here we present the final E-R diagram for the problem. Please note that knowing and thinking about a system is essential for making good ER diagrams.

Figure 16: The Entity Types along with the attributes and keys

Let us list each entity identified so far, together with its entity type, primary keys, and any foreign keys.

Entity	Type	Primary Key	Foreign Keys
Client	Independent	Client ID	
Site	Subordinate	Client ID, Site No	
Application	Independent	Application ID	

Package	Independent	Package ID	
Installation	Combination	Client ID, Site No, Application ID	Client ID, Site No, Application ID
License	Subordinate	Package ID, Copy No	Package ID

2) The E-R diagram is given below:

Figure 17: The E-R diagram for EMPLOYEE-DEPARTMENT database

In the above E-R diagram, EMPLOYEE is an entity, who works for the department, i.e., entity DEPARTMENT, thus **works_for** is many-to-one relationship, as many employees work for one department. Only one employee (i.e., Manager) manages the department, thus **manages** is the one-to-one relationship. The attribute Emp_Id is the primary key for the entity EMPLOYEE, thus Emp_Id is unique and not-null. The candidate keys for the entity DEPARTMENT are **Dept_name** and **Dept_Id**. Along with other attributes, NumberOfEmployees is the derived attribute on the entity DEPT, which could be recognised the number of employees working for that particular department. Both the entities EMPLOYEE and DEPARTMENT participate totally in the relationship works_for, as at least one employee work for the department, similarly an employee works for at least one department.

The entity EMPLOYEES work for the entity PROJECTS. Since many employees can work for one or more than one projects simultaneously, thus works_for is the N:N relationship. The entity DEPARTMENT controls the entity PROJECT, and since one department controls many projects, thus, controls in a 1:N relationship. The entity EMPLOYEE participate totally in the relationship works_for, as at least one employee

works for the project. A project has no meaning if no employee is working in a project.

The employees can have many dependents, but the entity DEPENDENTS cannot exist without the existence of the entity EMPLOYEE, thus, DEPENDENT is a weak entity. We can very well see the primary keys for all the entities. The underlined attributes in the eclipses represent the primary key.

3. The E-R diagram for supplier-and-parts database is given as follows:

Figure 18: E-R diagram for Supplier-Project and entities

UNIT 3 DATABASE INTEGRITY AND NORMALISATION

Structure **Page Nos.**

3.0 INTRODUCTION

In the previous unit, we have discussed relations. Relations form the database. They must satisfy some properties, such as no duplicate tuples, no ordering of tuples, and atomic attributes, etc. Relations that satisfy these basic requirements may still have some undesirable characteristics such as data redundancy, and anomalies.

What are these undesirable characteristics and how can we eliminate them from the database system? This unit is an attempt to answer this question. However, please note that most of these undesirable properties do not arise if the database modeling has been carried out very carefully using some technique like the Entity-Relationship Model that we have discussed in the previous unit. It is still important to use the techniques in this unit to check the database that has been obtained and ensure that no mistakes have been made in modeling.

The central concept in these discussions is the concept of Database integrity, the notion of functional dependency, which depends on understanding the semantics of the data and which deals with what information in a relation is dependent on what other information in the relation. Our prime objective in this unit is to define the concept of data integrity, functional dependence and then define normal forms using functional dependencies and using other types of data dependencies.

3.1 OBJECTIVES

After going through this unit you should be able to

- define the concept of entity integrity;

- describe relational database referential integrity constraints;
- define the concept of functional dependency, and
- show how to use the dependencies information to decompose relations; whenever necessary to obtain relations that have the desirable properties that we want without losing any of the information in the original relations.

3.2 RELATIONAL DATABASE INTEGRITY

A database is a collection of data. But, is the data stored in a database trustworthy? To answer that question we must first answer the question. What is integrity?

Integrity simply means to maintain the consistency of data. Thus, integrity constraints in a database ensure that changes made to the database by authorised users do not compromise data consistency. Thus, integrity constraints do not allow damage to the database.

There are primarily two integrity constraints: the entity integrity constraint and the referential integrity constraint. In order to define these two, let us first define the term Key with respect to a Database Management System.

3.2.1 The Keys

Candidate Key: In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following two properties:

(1)	*Uniqueness*	No two distinct tuples in R have the same value for the candidate key
(2)	*Irreducible*	No proper subset of the candidate key has the uniqueness property that is the candidate key.

Every relation must have at least one candidate key which cannot be reduced further. Duplicate tuples are not allowed in relations. Any candidate key can be a composite key also. For Example, (student-id + course-id) together can form the candidate key of a relation called marks (student-id, course-id, marks).

Let us summarise the properties of a candidate key.

Properties of a candidate key

- A candidate key must be unique and irreducible
- A candidate may involve one or more than one attributes. A candidate key that involves more than one attribute is said to be composite.

But why are we interested in candidate keys?

Candidate keys are important because they provide the basic **tuple-level identification** mechanism in a relational system.

For example, if the enrolment number is the candidate key of a STUDENT relation, then the answer of the query: “Find student details from the STUDENT relation having enrolment number A0123” will output at most one tuple.

Primary Key

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate keys, if any, are called **alternate keys.**

Foreign Keys

Let us first give you the basic definition of foreign key.

Let R2 be a relation, then a foreign key in R2 is a subset of the set of attributes of R2, such that:

1. There exists a relation R1 (R1 and R2 not necessarily distinct) with a candidate key, and
2. For all time, each value of a foreign key in the current state or instance of R2 is identical to the value of Candidate Key in some tuple in the current state of R1.

The definition above seems to be very heavy. Therefore, let us define it in more practical terms with the help of the following example.

Example 1

Assume that in an organisation, an employee may perform different roles in different projects. Say, RAM is doing coding in one project and designing in another. Assume that the information is represented by the organisation in three different relations named EMPLOYEE, PROJECT and ROLE. The ROLE relation describes the different roles required in any project.

Assume that the relational schema for the above three relations are:

EMPLOYEE (EMPID, Name, Designation)
PROJECT (PROJID, Proj_Name, Details)
ROLE (ROLEID, Role_description)

In the relations above EMPID, PROJID and ROLEID are unique and not NULL, respectively. As we can clearly see, we can identify the complete instance of the entity set employee through the attribute EMPID. Thus EMPID is the primary key of the relation EMPLOYEE. Similarly PROJID and ROLEID are the primary keys for the relations PROJECT and ROLE respectively.

Let ASSIGNMENT is a relationship between entities EMPLOYEE and PROJECT and ROLE, describing which employee is working on which project and what the role of the employee is in the respective project. Figure 1 shows the E-R diagram for these entities and relationships.

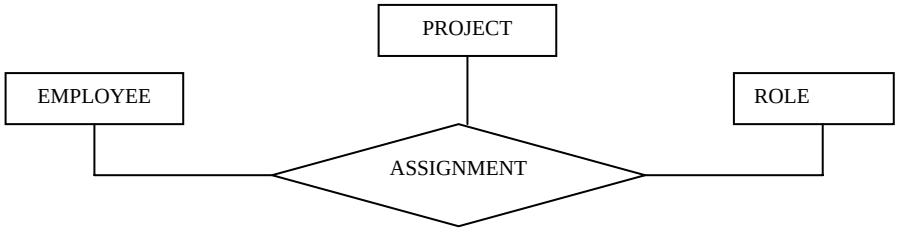


Figure 1: E-R diagram for employee role in development team

Let us consider sample relation instances as:

EMPLOYEE

EMPID	Name	Designation
101	RAM	Analyst
102	SITA	Receptionist
103	ARVIND	Manager

PROJECT

PROJID	Proj_name	Details
TCS	Traffic Control System	For traffic shaping.
LG	Load Generator	To simulate load for input in TCS.
B++1	B++_TREE	ISS/R turbo sys

ROLE

ROLEID	Role_descrption
1000	Design
2000	Coding
3000	Marketing

ASSIGNMENT

PROJID	Proj_name	Details
101	TCS	1000
101	LG	2000
102	B++1	3000

Figure 2: An example relation

We can define the relational scheme for the relation ASSIGNMENT as follows:

ASSIGNMENT (EMPID, PROJID, ROLEID)
Please note now that in the relation ASSIGNMENT (as per the definition to be taken as R2) EMPID is the foreign key in ASSIGNMENT relation; it references the relation EMPLOYEE (as per the definition to be taken as R1) where EMPID is the primary key. Similarly PROJID and ROLEID in the relation ASSIGNMENT are **foreign keys** referencing the relation PROJECT and ROLE respectively.

Now after defining the concept of foreign key, we can proceed to discuss the actual integrity constraints namely Referential Integrity and Entity Integrity.

3.2.2 Referential Integrity

It can be simply defined as:

The database must not contain any unmatched foreign key values.

The term "unmatched foreign key value" means a foreign key value for which there does not exist a **matching value** of the relevant candidate key in the relevant target (referenced) relation. For example, any value existing in the EMPID attribute in ASSIGNMENT relation must exist in the EMPLOYEE relation. That is, the only EMPIDs that can exist in the EMPLOYEE relation are 101, 102 and 103 for the present state/ instance of the database given in *Figure 2*. If we want to add a tuple with EMPID value 104 in the ASSIGNMENT relation, it will cause violation of referential integrity constraint. Logically it is very obvious after all the employee 104 does not exist, so how can s/he be assigned any work.

Database modifications can cause violations of referential integrity. We list here the test we must make for each type of database modification to preserve the referential-integrity constraint:

Delete

During the deletion of a tuple two cases can occur:

Deletion of tuple in relation having the foreign key: In such a case simply delete the desired tuple. For example, in ASSIGNMENT relation we can easily delete the first tuple.

Deletion of the target of a foreign key reference: For example, an attempt to delete an employee tuple in EMPLOYEE relation whose EMPID is 101. This employee appears not only in the EMPLOYEE but also in the ASSIGNMENT relation. Can this tuple be deleted? If we delete the tuple in EMPLOYEE relation then two unmatched tuples are left in the ASSIGNMENT relation, thus causing violation of referential integrity constraint. Thus, the following two choices exist for such deletion:

RESTRICT – The delete operation is "restricted" to only the case where there are no such matching tuples. For example, we can delete the EMPLOYEE record of EMPID 103 as no matching tuple in ASSIGNMENT but not the record of EMPID 101.

CASCADE – The delete operation "cascades" to delete those matching tuples also.

For example, if the delete mode is CASCADE then deleting employee having EMPID as 101 from EMPLOYEE relation will also cause deletion of 2 more tuples from ASSIGNMENT relation.

Insert

The insertion of a tuple in the target of reference does not cause any violation. However, insertion of a tuple in the relation in which, we have the foreign key, for example, in ASSIGNMENT relation it needs to be ensured that all matching target candidate key exist; otherwise the insert operation can be rejected. For example, one of the possible ASSIGNMENT insert operations would be (103, LG, 3000).

Modify

Modify or update operation changes the existing values. If these operations change the value that is the foreign key also, the only check required is the same as that of the Insert operation.

What should happen to an attempt to update a candidate key that is the target of a foreign key reference? For example, an attempt to update the PROJID "LG" for which there exists at least one matching ASSIGNMENT tuple? In general there are the same possibilities as for DELETE operation:

RESTRICT: The update operation is "restricted" to the case where there are no matching ASSIGNMENT tuples. (it is rejected otherwise).

CASCADE – The update operation "cascades" to update the foreign key in those matching ASSIGNMENT tuples also.

3.2.3 Entity Integrity

Before describing the second type of integrity constraint, viz., Entity Integrity, we should be familiar with the concept of **NULL.**

Basically, NULL is intended as a basis for dealing with the problem of missing information. This kind of situation is frequently encountered in the real world. For example, historical records sometimes have entries such as "Date of birth unknown", or police records may include the entry "Present whereabouts unknown." Hence it is necessary to have some way of dealing with such situations in database systems. Thus Codd proposed an approach to this issue that makes use of special markers called NULL to represent such missing information.

A given attribute in the relation might or might not be allowed to contain NULL. But, can the Primary key or any of its components (in case of the primary key is a composite key) contain a NULL? To answer this question an **Entity Integrity Rule** states: No component of the primary key of a relation is allowed to accept NULL. In other words, the definition of every attribute involved in the primary key of any basic relation must explicitly or implicitly include the specifications of NULL NOT ALLOWED.

Foreign Keys and NULL

Let us consider the relation:

DEPT		
DEPT ID	**DNAME**	**BUDGET**
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP			
EMP ID	**ENAME**	**DEPT ID**	**SALARY**
E1	Rahul	D1	40K
E2	Aparna	D1	42K
E3	Ankit	D2	30K
E4	Sangeeta		35K

Suppose that Sangeeta is not assigned any Department. In the EMP tuple corresponding to Sangeeta, therefore, there is no genuine department number that can serve as the appropriate value for the DEPTID foreign key. Thus, one cannot determine DNAME and BUDGET for Sangeeta's department as those values are NULL. This may be a real situation where the person has newly joined and is

undergoing training and will be allocated to a department only on completion of the training. Thus, NULL in foreign key values may not be a logical error.

So, the foreign key definition may be redefined to include NULL as an acceptable value in the foreign key for which there is no need to find a matching tuple.

Are there any other constraints that may be applicable on the attribute values of the entities? Yes, these constraints are basically related to the domain and termed as the domain constraints.

Domain Constraints

Domain constraints are primarily created for defining the logically correct values for an attribute of a relation. The relation allows attributes of a relation to be confined to a range of values, for example, values of an attribute age can be restricted as Zero to 150 or a specific type such as integers, etc. These are discussed in more detail in Block 2 Unit 1.

☞ Check Your Progress 1

Consider supplier-part-project database;

SUPPLIERS

S.NO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune
S4	Sita	Delhi
S5	Anand	Agra

PROJECTS

PROJNO	JNAME	CITY
PROJ1	Sorter	Pune
PROJ2	Display	Mumbai
PROJ3	OCR	Agra
PROJ4	Console	Agra
PROJ5	RAID	Delhi
PROJ6	EDS	Udaipur
PROJ7	Tape	Delhi

PARTS

P.NO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Screw	White	Mumbai
P4	Screw	Blue	Delhi
P5	Cam	Brown	Pune
P6	Cog	Grey	Delhi

SUP_PAR_PROJ

SNO	PNO	PROJ NO	QUANTITY
S1	P1	PROJ1	200
S1	P1	PROJ4	700
S2	P3	PROJ2	400
S2	P2	PROJ7	200
S2	P3	PROJ3	500
S3	P3	PROJ5	400
S3	P4	PROJ3	500
S3	P5	PROJ3	600
S3	P6	PROJ4	800
S4	P6	PROJ2	900
S4	P6	PROJ1	100
S4	-	PROJ7	200
S5	P5	PROJ5	300
S6	P4	PROJ6	400

1) What are the Candidate keys and PRIMARY key to the relations?

……

……

……

……

2) What are the entity integrity constraints? Are there any domain constraints?

……

……

……

……

3) What are the referential integrity constraints in these relations?

……

……

……

……

4) What are referential actions you would suggest for these referential integrity constraints?

……

……

……

……

3.3 REDUNDANCY AND ASSOCIATED PROBLEMS

Let us consider the following relation STUDENT.

Enrolment no	Sname	Address	Cno	Cname	Instructor	Office
050112345	Rahul	D-27, main Road Ranchi	MCS–011	Problem Solution	Nayan Kumar	102
050112345	Rahul	D-27, Main Road Ranchi	MCS–012	Computer Organisation	Anurag Sharma	105
050112345	Rahul	D-27, Main Road Ranchi	MCS-014	SSAD	Preeti Anand	103
050111341	Aparna	B-III, Gurgaon	MCS-014	SSAD	Preeti Anand	103

Figure 3: A state of STUDENT relation

The above relation satisfies the properties of a relation and is said to be in first normal form (or 1NF). Conceptually it is convenient to have all the information in one relation since it is then likely to be easier to query the database. But the relation above has the following undesirable features:

Data Redundancy-A lot of information is being repeated in the relation. For example, the information that MCS-014 is named SSAD is repeated, address of Rahul is "D-27, Main road, Ranchi" is being repeated in the first three records. Please find the other duplicate information. So we find that the student name, address, course name, instructor name and office number are being repeated often, thus, the table has **Data Redundancy**.

Please also note that every time we wish to insert a student record for a subject taken by him/her, say for example, MCS-013, we must insert the name of the course as well as the name and office number of its instructor. Also, every time we insert a new course we must repeat the name and address of the student who has taken it. This repetition of information results in problems in addition to the wastage of space. Check these problems in the STUDENT relation. What are these problems? Let us define them.

These problems are called database anomalies. There are three anomalies in database systems:

1. *Update Anomaly*: This anomaly is caused due to data redundancy. Redundant information makes updates more difficult since, for example, changing the name of the instructor of MCS-014 would require that all tuples containing MCS-014 enrolment information be updated. If for some reason, all tuples are not updated, we might have a database that gives two names of instructor for the subject MCS-014 which is inconsistent information. This problem is called update anomaly. An update anomaly results in **data inconsistency.**

2. *Insertion Anomaly*: Inability to represent certain information-The primary key of the above relation be (enrolment number, Cno). Any new tuple to be inserted in the relation must have a value for the primary key since entity integrity constraint requires that a key may not be totally or partially NULL. However, in the given relation if one wanted to insert the number and name of a new course in the database, it would not be possible until a student enrols in that course. Similarly information about a new student cannot be inserted in the database until the student enrols in a course. These problems are called insertion anomalies.

3. *Deletion Anomalies*: Loss of Useful Information: In some instances, useful information may be lost when a tuple is deleted. For example, if we delete the tuple corresponding to student 050111341 enrolled for MCS-014, we will lose relevant information about the student viz. enrolment number, name and address of this student. Similarly deletion of tuple having Sname "Rahul" and Cno 'MCS-012" will result in loss of information that MCS-012 is named computer organisation having an instructor "Anurag Sharma", whose office number is 105. This is called deletion anomaly.

The anomalies arise primarily because the relation STUDENT has information about students as well as subjects. One solution to the problems is to decompose the relation into two or more smaller relations. But what should be the basis of this decomposition? To answer the questions let us try to formulate how data is related in the relation with the help of the following *Figure 4*:

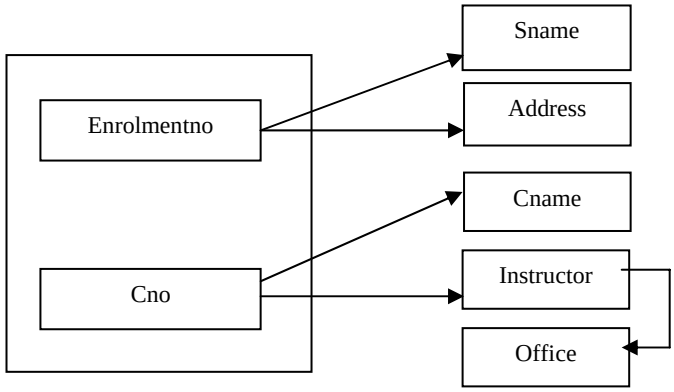


Figure 4: The dependencies of relation in Figure 3

Please note that the arrows in *Figure 4* are describing data inter-relationship. For example, enrolmentno column is unique for a student so if we know the enrolment no of a student we can uniquely determine his/her name and address. Similarly, the course code (Cno) uniquely determines course name (Cname) and Instructor (we are assuming that a course is taught by only one instructor). Please also note one important interrelationship in *Figure 4* that is, the Office (address) of an instructor is dependent on Instructor (name), assuming unique instructor names. The root cause of the presence of anomalies in a relation is determination of data by the components of the key and non-key attributes.

Normalisation involves decomposition of a relation into smaller relations based on the concept of functional dependence to overcome undesirable anomalies.

Normalisation sometimes can affect performance. As it results in decomposition of tables, some queries desire to join these tables to produce the data once again. But such performance overheads are minimal as Normalisation results in minimisation of data redundancy and may result in smaller relation sizes. Also DBMSs implements optimised algorithms for joining of relations and many indexing schemes that reduce the load on joining of relations. In any case the advantages of normalisaiton normally overweigh the performance constraints. Normalisation does lead to more efficient updates since an update that may have required several tuples to be updated, whereas normalised relations, in general, require the information updating at only one place.

A relation that needs to be normalised may have a very large number of attributes. In such relations, it is almost impossible for a person to conceptualise all the information and suggest a suitable decomposition to overcome the problems. Such relations need an algorithmic approach of finding if there are problems in a proposed database design and how to eliminate them if they exist. The discussions of these algorithms are beyond the scope of this Unit, but, we will first introduce you to the basic concept that supports the process of Normalisation of large databases. So let us first define the concept of functional dependence in the subsequent section and follow it up with the concepts of normalisation.

3.4 SINGLE-VALUED DEPENDENCIES

A database is a collection of related information and it is therefore inevitable that some items of information in the database would depend on some other items of information. The information is either single-valued or multivalued. The name of a person or his date of birth is single-valued facts; qualifications of a person or subjects that an instructor teaches are multivalued facts. We will deal only with single-valued facts and discuss the concept of functional dependency.

Let us define this concept logically.

Functional Dependency

Consider a relation R that has two attributes A and B. The attribute B of the relation is functionally dependent on the attribute A if and only if for each value of A, no more than one value of B is associated. In other words, the value of attribute A uniquely determines the value of attribute B and if there were several tuples that had the same value of A then all these tuples will have an identical value of attribute B. That is, if **t1** and **t2** are two tuples in the relation R where **t1(A) = t2(A),** then we must have **t1(B) = t2(B).**

Both, A and B need not be single attributes. They could be any subsets of the attributes of a relation R. The FD between the attributes can be written as:

R.A → R.B or simply A → B, if B is functionally dependent on A (or A functionally determines B). Please note that functional dependency does not imply a one-to-one relationship between A and B.

For example, the relation in *Figure 3*, whose dependencies are shown in *Figure 4*, can be written as:

Enrolmentno → Sname
Enrolmentno → Address
Cno → Cname
Cno → Instructor

Instructor→Office

These functional dependencies imply that there can be only one student name for each **Enrolmentno**, only one address for each student and only one subject name for each **Cno**. It is of course possible that several students may have the same name and several students may live at the same address.

If we consider **Cno → Instructor**, the dependency implies that no subject can have more than one instructor (perhaps this is not a very realistic assumption). Functional dependencies therefore place constraints on what information the database may store. In the example above, you may be wondering if the following FDs hold:

Sname → Enrolmentno **(1)**
Cname → Cno **(2)**

Certainly there is nothing in the given instance of the database relation presented that contradicts the functional dependencies as above. However, whether these FDs hold or not would depend on whether the university or college whose database we are considering allows duplicate student names and course names. If it was the enterprise policy to have unique course names than **(2)** holds. If duplicate student names are possible, and one would think there always is the possibility of two students having exactly the name, then (**1)** does not hold.

A simple example of the functional dependency above is when A is a primary key of an entity (e.g., enrolment number: Enrolment no) and B is some single-valued property or attribute of the entity (e.g., student name: Sname). **A → B** then must always hold. (Why?)

Functional dependencies also arise in relationships. Let C be the primary key of an entity and D be the primary key of another entity. Let the two entities have a relationship. If the relationship is one-to-one, we must have both **C → D** and **D → C** . If the relationship is many-to-one (Con many side), we would have **C → D** but not **D → C** . For many-to-many relationships, no functional dependencies hold.

For example, consider the following E-R diagram:

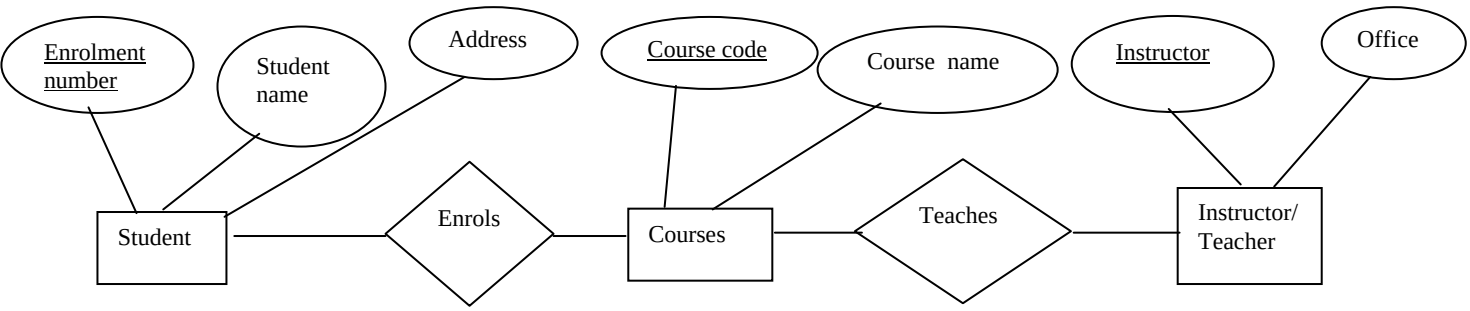


Figure 5: E-R diagram for student course Teacher

In the ER diagram as above, the following FDs exist:

FDs in Entities:

Student entity:

Enrolment number → Student name, Address

Course Entity

Course code → Course name

Instructor/ Teacher Entity:

Instructor → Office

FDs in Relationships:

Enrols Relationship: None as it is many to Many
Teaches Relationship:
Course code → Instructor
Instructor → Course code

The next question is: How do we identify the functional dependence in a database model?

Functional dependencies arise from the nature of the real world that the database models. Often A and B are facts about an entity where A might be some identifier for the entity and B some characteristic. Functional dependencies cannot be automatically determined by studying one or more instances of a database. They can be determined only by a careful study of the real world and a clear understanding of what each attribute means.

There are no thumb rules for determining FDs.

Functional dependence is an important concept and a large body of formal theory has been developed about it.

3.5 SINGLE VALUED NORMALISATION

Codd in the year 1972 presented three normal forms (1NF, 2NF, and 3NF). These were based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multivalue and join dependencies and were proposed later. In this section we will cover normal forms till BCNF only. Fourth and fifth normal forms are beyond the scope of this course.
For all practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed for most common situations. It should be clearly understood that there is no obligation to normalise relations to the highest possible level. Performance should be taken into account and sometimes an organisation may take a decision not to normalise, say, beyond third normal form. But, it should be noted that such designs should be careful enough to take care of anomalies that would result because of the decision above.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). A sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 2NF or 3NF.

Normalisation results in decomposition of the original relation. It should be noted that decomposition of relation has to be always based on principles, such as functional dependence, that ensure that the original relation may be reconstructed from the decomposed relations if and when necessary. Careless decomposition of a relation can result in loss of information. We will discuss this in detail in the later section.

Let us now define these normal forms in more detail.

3.5.1 The First Normal Form (1NF)

Let us first define 1NF:

Definition: A relation (table) is in 1NF if

1. **There are no duplicate rows or tuples in the relation.**

2. **Each data value stored in the relation is single-valued**
3. **Entries in a column (attribute) are of the same kind (type).**

Please note that in a 1NF relation the order of the tuples (rows) and attributes (columns) does not matter.

The first requirement above means that the relation **must have a key**. The key may be single attribute or composite key. It may even, possibly, contain all the columns. The first normal form defines only the basic structure of the relation and does not resolve the anomalies discussed in section 3.3.

The relation STUDENT (Enrolmentno, Sname, Address, Cno, Cname, Instructor, Office) of *Figure 3* is in 1NF. The primary key of the relation is (Enrolmentno+Cno).

3.5.2 The Second Normal Form (2NF)

The relation of *Figure 3* is in 1NF, yet it suffers from all anomalies as discussed earlier so we need to define the next normal form.

Definition: A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation.

Some of the points that should be noted here are:

- A relation having a single attribute key has to be in 2NF.
- In case of composite key, partial dependency on key that is part of the key is not allowed.
- 2NF tries to ensure that information in one relation is about one thing
- Non-key attributes are those that are not part of any candidate key.

Let us now reconsider *Figure 4*, which defines the FDs of the relation to the relation STUDENT (Enrolmentno, Sname, Address, Cno, Cname, Instructor, Office). These FDs can also be written as:

Enrolmentno	→	Sname, Address	(1)
Cno	→	Cname, Instructor	(2)
Instructor	→	Office	(3)

The key attributes of the relation are (Enrolmentno + Cno). Rest of the attributes are non-key attributes. For the 2NF decomposition, we are concerned with the FDs (1) and (2) as above as they relate to partial dependence on the key that is (Enrolmentno + Cno). As these dependencies (also refer to *Figure 4*) shows that relation in not in 2NF and hence suffer from all the three anomalies and redundancy problems as many non-key attributes can be derived from partial key attribute. To convert the relation into 2NF, let us use FDs. As per FD (1) the Enrolment number uniquely determines student name and address, so one relation should be:
STUDENT1 (Enrolmentno, Sname, Address)

Now as per FD (2) we can decompose the relation further, but what about the attribute 'Office'?

We find in FD (2) that Course code (Cno) attribute uniquely determines the name of instructor (refer to FD 2(a)). Also the FD (3) means that name of the instructor uniquely determines office number. This can be written as:

Cno	→	Instructor	(2 (a)) (without Cname)
Instructor	→	Office	(3)
⇒ Cno	→	Office	(This is transitive dependency)

Thus, FD (2) now can be rewritten as:

Cno	→	Cname, Instructor, Office	(2′)

This FD, now gives us the second decomposed relation:

COU_INST (Cno, Cname, Instruction, Office)

Thus, the relation STUDENT has been decomposed into two relations:

STUDENT1 (Enrolmentno, Sname, Address)
COU_INST (Cno, Cname, Instruction, Office)

Is the decomposition into 2NF complete now?

No, how would you join the two relations created above any way? Please note we have super FDs as, because (Enrolmentno + Cno) is the primary key of the relation STUDENT:

Enrolmentno, Cno → ALL ATTRIBUTES

All the attributes except for the key attributes that are Enrolmentno and Cno, however, are covered on the right side of the FDs (1) (2) and (3), thus, making the FD as redundant. But in any case we have to have a relation that joins the two decomposed relations. This relation would cover any attributes of Super FD that have not been covered by the decomposition and the key attributes. Thus, we need to create a joining relation as:

COURSE_STUDENT (Enrolmentno, Cno)

So, the relation STUDENT in 2NF form would be:

STUDENT1 (Enrolmentno, Sname, Address)	2NF(a)
COU_INST (Cno, Cname, Instruction, Office)	2NF(b)
COURSE_STUDENT (Enrolmentno, Cno)	2NF(c)

3.5.3 The Third Normal Form (3NF)

Although, transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies, it does not necessarily remove all anomalies. Thus, further Normalisation is sometimes needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly related to the candidate keys of the relation.

Definition: A relation is in third normal form, if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on each candidate key of the relation**.**

But what is **non-transitive** dependence?
Let A, B and C be three attributes of a relation R such that **A → B** and **B → C**. From these FDs, we may derive **A → C**. This dependence **A → C** is transitive.

Now, let us reconsider the relation 2NF (b)

COU_INST (Cno, Cname, Instruction, Office)

Assume that Cname is not unique and therefore Cno is the only candidate key. The following functional dependencies exists

Cno	→	**Instructor**	(2 (a))
Instructor	→	**Office**	(3)
Cno	→	Office	(This is transitive dependency)

We had derived Cno → Office from the functional dependencies 2(a) and (3) for decomposition to 2NF. The relation is however not in 3NF since the attribute 'Office' is not directly dependent on attribute 'Cno' but is transitively dependent on it and

should, therefore, be decomposed as it has all the anomalies. The primary difficulty in the relation above is that an instructor might be responsible for several subjects, requiring one tuple for each course. Therefore, his/her office number will be repeated in each tuple. This leads to all the problems such as update, insertion, and deletion anomalies. To overcome these difficulties we need to decompose the relation 2NF(b) into the following two relations:

COURSE (Cno, Cname, Instructor)
INST (Instructor, Office)

Please note these two relations and 2NF (a) and 2NF (c) are already in 3NF. Thus, the relation STUDENT in 3 NF would be:

STUDENT1 (Enrolmentno, Sname, Address)
COURSE (Cno, Cname, Instructor)
INST (Instructor, Office)
COURSE_STUDENT (Enrolmentno, Cno)

The 3NF is usually quite adequate for most relational database designs. There are however some situations where a relation may be in 3 NF, but have the anomalies. For example, consider the relation NEWSTUDENT (Enrolmentno, Sno, Sname, Cno, Cname) having the set of FDs:

Enrolmentno	→	Sname
Sname	→	Enrolmentno
Cno	→	Cname
Cname	→	Cno

The relation is in 3NF. Why? Please refer to the functional diagram for this relation given in *Figure 6*.

Error!

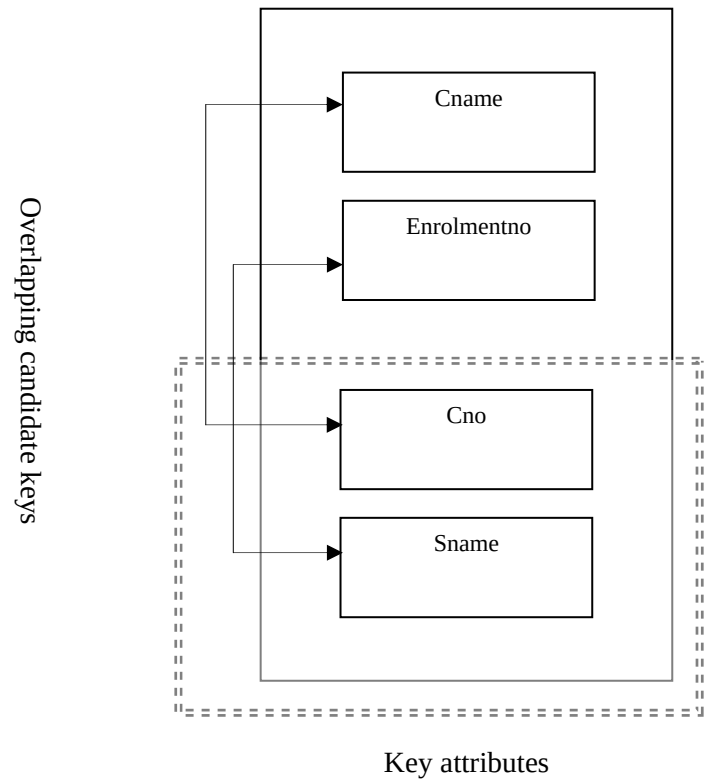


Figure 6: Functional Diagram for NEWSTUDENT relation

All the attributes of this relation are part of candidate keys, but have dependency between the non-overlapping portions of overlapping candidate keys. Thus, the 3NF may not eliminate all the redundancies and inconsistencies. Thus, there is a need of further Normalisation using the BCNF.

3.5.4 Boyce-Codd Normal Form (BCNF)

The relation NEWSTUDENT (Enrolmentno, Sno, Sname, Cno, Cname) has all attributes participating in candidate keys since all the attributes are assumed to be unique. We therefore had the following candidate keys.

(Enrolmentno, Cno)
(Enrolmentno, Cname)
(Sname, Cno)
(Sname, Cname)

Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF. However, the relation suffers from the anomalies (please check it yourself by making the relational instance of the NEWSTUDENT relation).

The difficulty in this relation is being caused by dependence within the candidate keys.

Definition: A relation is in BCNF, if it is in 3NF and if every determinant is a candidate key.

- A determinant is the left side of an FD
- Most relations that are in 3NF are also in BCNF. A 3NF relation is not in BCNF if all the following conditions apply.

(a) The candidate keys in the relation are composite keys.
(b) There is more than one overlapping candidate keys in the relation, and some attributes in the keys are overlapping and some are not overlapping.
(c) There is a FD from the non-overlapping attribute(s) of one candidate key to non-overlapping attribute(s) of other candidate key.

Let us recall the NEWSTUDENT relation:

NEWSTUDENT (Enrolmentno, Sno, Sname, Cno, Cname)

Set of FDs:

Enrolmentno	→	Sname	(1)
Sname	→	Enrolmentno	(2)
Cno	→	Cname	(3)
Cname	→	Cno	(4)

The relation although in 3NF, but is not in BCNF and can be decomposed on any one of the FDs in (1) & (2); and any one of the FDs in (3) & (4) as:

STUD1 (Enrolmentno, Sname)
COUR1 (Cno, Cname)

The third relation that will join the two relation will be:
ST_CO(Enrolmentno, Cno)

Since this is a slightly difficult form, let us give one more example, for BCNF.

Consider for example, the relation:

ENROL(Enrolmentno, Sname, Cno, Cname, Dateenrolled)

Let us assume that the relation has the following candidate keys:

(Enrolmentno, Cno)
(Enrolmentno, Cname)
(Sname, Cno)
(Sname, Cname)

(We have assumed Sname and Cname are unique identifiers).

The relation has the following set of dependencies:

Enrolmentno	→	Sname
Sname	→	Enrolmentno
Cno	→	Cname
Cname	→	Cno
Enrolmentno, Cno	→	Dateenrolled

The relation is in 3NF but not in BCNF because there are dependencies. The relation suffers from all anomalies. Please draw the relational instance and checks these problems. The BCNF decomposition of the relation would be:

STUD1 (Enrolment no, Sname)
COU1 (Cno, Cname)
ENROL1 (Enrolmentno, Cno, Dateenrolled)

We now have a relation that only has information about students, another only about subjects and the third only about relationship enrolls.

Higher Normal Forms:

Are there more normal forms beyond BCNF? Yes, however, these normal forms are not based on the concept of functional dependence. Further normalisation is needed if the relation has Multi-valued, join dependencies, or template dependencies. These topics are beyond the scope of this unit you can refer to further readings for more detail on these. MCS-043 contains more discussion on these topics.

☞ Check Your Progress 2

1) Consider the following relation
LIBRARY (member_id, member_name, book_code, book_name, issue_date, return_date)

The relation stores information about the issue and return of books in a Library to its members. A member can be issued many books. What are the anomalies in the relation above?

……..…

……..…

2) What are the functional dependencies in the relation above? Are there any constraints, specially domain constraint, in the relation?

……..…

……..…

3) Normalise the relation of problem 1.

……

……

……

……..….

3.6 DESIRABLE PROPERTIES OF DECOMPOSITION

We have used normalisation to decompose relations in the previous section. But what is decomposition formally? Decomposition is a process of splitting a relation into its projections that will not be disjoint. Please recall the Relational projection operator given in Unit 2, and remember no duplicate tuples are allowed in a projection.

Desirable properties of decomposition are:

- Attribute preservation
- Lossless-join decomposition
- Dependency preservation
- Lack of redundancy

3.6.1 Attribute Preservation

This is a simple and obvious requirement that involves preserving all the attributes that were there in the relation that is being decomposed.

3.6.2 Lossless-Join Decomposition

Let us show an intuitive decomposition of a relation. We need a better basis for deciding decompositions since intuition may not always be correct. We illustrate how a careless decomposition may lead to problems including loss of information.

Consider the following relation
ENROL (stno, cno, date-enrolled, room-no, instructor)

Suppose we decompose the above relation into two relations enrol and enrol2 as follows:

ENROL1 (stno, cno, date-enrolled)

ENROL2 (date-enrolled, room-no, instructor)

There are problems with this decomposition but we do not wish to focus on this aspect at the moment. Let an instance of the relation ENROL be:

St no	cno	Date-enrolled	Room-no	Instructor
1123	MCS-011	20-06-2004	**1**	Navyug
1123	MCS-012	26-09-2004	**2**	Anurag Sharma
1259	MCS-011	26-09-2003	**1**	Preeti Anand
1134	MCS-015	30-10-2005	**5**	Preeti Anand
2223	MCS-016	05-02-2004	**6**	Shashi Bhushan

Figure 7: A sample relation for decomposition

Then on decomposition the relations ENROL1 and ENROL2 would be:

ENROL1

St no	Cno	Date-enrolled
1123	MCS-011	20-06-2004
1123	MCS-012	26-09-2004
1259	MCS-011	26-09-2003
1134	MCS-015	30-10-2005
2223	MCS-016	05-02-2004

ENROL2

Date-enrolled	Room-no	Instructor
20-06-2004	**1**	Navyug
26-09-2004	**2**	Anurag Sharma
26-09-2003	**1**	Preeti Anand
30-10-2005	**5**	Preeti Anand
05-02-2004	**6**	Shashi Bhushan

All the information that was in the relation ENROL appears to be still available in ENROL1 and ENROL2 but this is not so. Suppose, we wanted to retrieve the student numbers of all students taking a course from Preeti Anand, we would need to join ENROL1 and ENROL2. For joining the only common attribute is Date-enrolled. Thus, the resulting relation obtained will not be the same as that of *Figure 7*. (Please do the join and verify the resulting relation).

The join will contain a number of spurious tuples that were not in the original relation. Because of these additional tuples, we have lost the right information about which students take courses from Preeti Anand. (Yes, we have more tuples but less information because we are unable to say with certainty who is taking courses from Preeti Anand). Such decompositions are called **lossy decompositions**. A nonloss or lossless decomposition is that which guarantees that the join will result in exactly the same relation as was decomposed. One might think that there might be other ways of recovering the original relation from the decomposed relations but, sadly, no other operators can recover the original relation if the join does not (why?).

We need to analyse why the decomposition is lossy. The common attribute in the above decompositions was Date-enrolled. The common attribute is the glue that gives us the ability to find the relationships between different relations by joining the relations together. **If the common attribute have been the primary key of at least one of the two decomposed relations, the problem of losing information would not have existed**. The problem arises because several enrolments may take place on the same date.

The dependency based decomposition scheme as discussed in the section 3.5 creates lossless decomposition.

3.6.3 Dependency Preservation

It is clear that the decomposition must be lossless so that we do not lose any information from the relation that is decomposed. Dependency preservation is another important requirement since a **dependency is a constraint** on the database. If all the attributes appearing on the left and the right side of a dependency appear in the same relation, then a dependency is considered to be preserved. Thus, dependency preservation can be checked easily. Dependency preservation is important, because as stated earlier, dependency is a constraint on a relation. Thus, if a constraint is split over more than one relation (dependency is not preserved), the constraint would be difficult to meet. We will not discuss this in more detail in this unit. You may refer to the further readings for more details. However, let us state one basic point:
"A decomposition into 3NF is lossless and dependency preserving whereas a decomposition into BCNF is lossless but may or may not be dependency preserving."

3.6.4 Lack of Redundancy

We have discussed the problems of repetition of information in a database. Such repetition should be avoided as much as possible. Let us state once again that redundancy may lead to inconsistency. On the other hand controlled redundancy

sometimes is important for the recovery in database system. We will provide more details on recovery in Unit 3 of Block 2.

3.7 RULES OF DATA NORMALISATION

Let us now summarise Normalisation with the help of several clean rules. The following are the basic rules for the Normalisation process:

1. **Eliminate Repeating Groups:** Make a separate relation for each set of related attributes, and give each relation a primary key.
2. **Eliminate Redundant Data:** If an attribute depends on only part of a multi-attribute key, remove it to a separate relation.
3. **Eliminate Columns Not Dependent On Key:** If attributes do not contribute to a description of the key, remove them to a separate relation.
4. **Isolate Independent Multiple Relationships:** No relation may contain two or more **1:n** or **n:m** relationships that are not directly related.
5. **Isolate Semantically Related Multiple Relationships:** There may be practical constrains on information that justify separating logically related many-to-many relationships.

Let's explain these steps of Normalisation through an example:

Let us make a list of all the employees in the company. In the original employee list, each employee name is followed by any databases that the member has experience with. Some might know many, and others might not know any.

Emp-ID	Emp-Name	Database-Known	Department	Department-Loc
1	Gurpreet Malhotra	Oracle,	A	N-Delhi
2	Faisal Khan	Access	A	N-Delhi
3	Manisha Kukreja	FoxPro	B	Agra
4	Sameer Singh	DB2, Oracle	C	Mumbai

For the time being we are not considering department and Department-Loc till step 3.

3.7.1 Eliminate Repeating Groups

The query is, "Find out the list of employees, who knows DB2".
For this query we need to perform an awkward scan of the list looking for references to DB2. This is inefficient and an extremely untidy way to retrieve information.

We convert the relation to 1NF. Please note that in the conversion Department and Department-loc fields will be part of Employee relation. The Emp-ID in the Database relation matches the primary key in the employee relation, providing a foreign key for relating the two relations with a join operation. Now we can answer the question by looking in the database relation for "DB2" and getting the list of Employees. Please note that in this design we need not add D-ID (Database ID). Just the name of the database would have been sufficient as the names of databases are unique.

Employee Relation	
Emp-ID	**Emp-Name**
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh

Database Relation		
D-ID	**Emp-ID**	**Database-name**
1	2	Access
2	4	DB2
3	3	FoxPro
4	1	Oracle
4	4	Oracle

3.7.2 Eliminate Redundant Data

In the "Database Relation" above, the primary key is made up of the Emp-ID and the D-ID. The database-name depends only on the D-ID. The same database-name will appear redundantly every time its associated D-ID appears in the Database Relation. The database relation has redundancy, for example D-ID value 4 is oracle is repeated twice. In addition, it also suffers insertion anomaly that is we cannot enter Sybase in the relation as no employee has that skill.

The deletion anomaly also exists. For example, if we remove employee with Emp-ID 3; no employee with FoxPro knowledge remains and the information that D-ID 3 is the code of FoxPro will be lost.

To avoid these problems, we need **second normal form**. To achieve this, we isolate the attributes depending on both parts of the key from those depending only on the D-ID. This results in two relations: "Database" which gives the name for each D-ID, and "Emp-database" which lists the databases for each employee. The employee relation is already in 2NF as all the EMP-ID determines all other attributes.

Employee Relation	
Emp-ID	**Emp-Name**
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh

Emp-database Relation	
Emp-ID	**D-ID**
2	1
4	2
3	3
1	4
4	4

Database Relation	
D-ID	**Database**
1	Access
2	DB2
3	FoxPro
4	Oracle

3.7.3 Eliminate Columns Not Dependent On Key

The Employee Relation satisfies -

First normal form - As it contains no repeating groups.

Second normal form - As it doesn't have a multi-attribute key.

The employee relation is in 2NF but not 3NF. So we consider this table only after adding the required attributes.

Employee Relation			
Emp-ID	**Emp-Name**	**Department**	**Department-Loc**
1	Gurpreet Malhotra	A	N-Delhi
2	Faisal Khan	A	N-Delhi
3	Manisha Kukreja	B	Agra
4	Sameer Singh	C	Mumbai

The key is Emp-ID, and the Dept-Name and location describe only about Department, not an Employee. To achieve the third normal form, they must be moved into a separate relation. Since they describe a department, thus the attribute Department becomes the key of the new "Department" relation.

The motivation for this is the same for the second normal form: we want to avoid update, insertion and deletion anomalies.

Employee-List	
Emp-ID	**Emp-Name**
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh

Department-Relation		
Dept-ID	**Department**	**Department Loc**
1	A	N-Delhi
2	B	Agra
3	C	Mumbai

The rest of the Relation remains the same.

The last two steps: Isolate Independent Multiple Relationships and Isolate Semantically Related Multiple Relationships, converts the relations to higher normal form and are therefore not discussed here. They are not even required for the current example.

☞ Check Your Progress 3

1) What is the need of dependency preservation?

……

……

……

……

……

2) What is a lossless decomposition?

……

……

……

……

3) What are the steps for Normalisation till BCNF?

……

……

……

……

3.8 SUMMARY

This unit converts the details of Database integrity in detail. It covers aspects of keys, entity integrity and referential integrity. The role of integrity constraints is to make data more consistent.

A **functional dependency (FD)** is a many-to-one relationship between two sets of attributes of a given relation. Given a relation R, the FD A → B (where A and B are subsets of the attributes of R) is said to hold in R if and only if, whenever two tuples of R have the same value for A, and they also have the same value for B.

We discussed Single valued Normalisation and the concepts of **first, second, third,** and **Boyce/Codd normal forms.** The purpose of Normalisation is to avoid **redundancy**, and hence to avoid certain **update insertion and detection anomalies.** We have also discussed the desirable properties of a decomposition of a relation to its normal forms. The decomposition should be attribute preserving, dependency preserving and lossless.

We have also discussed various rules of data Normalisation, which help to normalise the relation cleanly. Those rules are eliminating repeating groups, eliminate redundant data, eliminate columns not dependent on key, isolate independent multiple relationship and isolate semantically related multiple relationships.

3.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1)

Relation	Candidate Keys	Primary key
SUPPLIERS	SNO, SNAME*	SNO
PARTS	PNO, PNAME*	PNO
PROJECTS	PROJ NO, JNAME*	PROJNO
SUP_PAR_PROJ	(SNO+PNO+PROJNO)	SNO+PNO+PROJNO

* Only if the values are assumed to be unique, this may be incorrect for large systems.

2) SNO in SUPPLIERS, PNO in PARTS, PROJNO in PROJECTS and (SNO+PNO+PROjNO) in SUP_PAR_PROJ should not contain NULL value. Also no part of the primary key of SUP_PAR_PROJ) that is SNO or PNO or PROJNO should contain NULL value. Please note SUP_PAR_PROJ relation is violating this constraint in the 12^{th} tuple which is not allowed.

3) Foreign keys exist only in SUP_PAR_PROJ where SNO references SNO of SUPPLIERS; PNO references PNO of PARTS; and PROJNO references PROJNO of PROJECTS. The referential integrity necessitates that all matching foreign key values must exist. The SNO field of SUP_PAR_PROJ in last tuple contains S6 which has no matching SNO in SUPPLIERS (valid values are from S1 to S5). So there is a violation of referential integrity constraint.

4) The proposed referential actions are:

For Delete and update, we must use RESTRICT, otherwise we may lose the information from the SUP_PAR_PROJ relation. Insert does not create a problem, only referential integrity constraints must be met.

Check Your Progress 2

1) The database suffers from all the anomalies; let us demonstrate these with the help of the following relation instance or state of the relation:

Member ID	Member Name	Book Code	Book Title	Issue Date	Return Date
A 101	Abishek	0050	DBMS	15/01/05	25/01/05
R 102	Raman	0125	DS	25/01/05	29/01/05
A 101	Abishek	0060	Multimedia	20/01/05	NULL
R 102	Raman	0050	DBMS	28/01/05	NULL

Is there any data redundancy?

Yes, the information is getting repeated about member names and book details. This may lead to any update anomaly in case of changes made to data value of the book. Also note the library must be having many more books that have not been issued yet. This information cannot be added to the relation as the primary key to the relation is:

(member_id + book_code + issue_date). (This would involve the assumption that the same book can be issued to the same student only once in a day).

Thus, we cannot enter the information about book_code and title of that book without entering member_id and issue_date. So we cannot enter a book that has not been issued to a member so far. This is insertion anomaly. Similarly, we cannot enter member_id if a book is not issued to that member. This is also an insertion anomaly.

As far as the deletion anomaly is concerned, suppose Abishek did not collect the Multimedia book, so this record needs to be deleted from the relation (tuple 3). This deletion will also remove the information about the Multimedia book that is its book code and title. This is deletion anomaly for the given instance.

2) The FDs of the relation are:

member_id → member_name (1)

book_code → book_name (2)

book_code, member_id, issue_date → return_date (3)

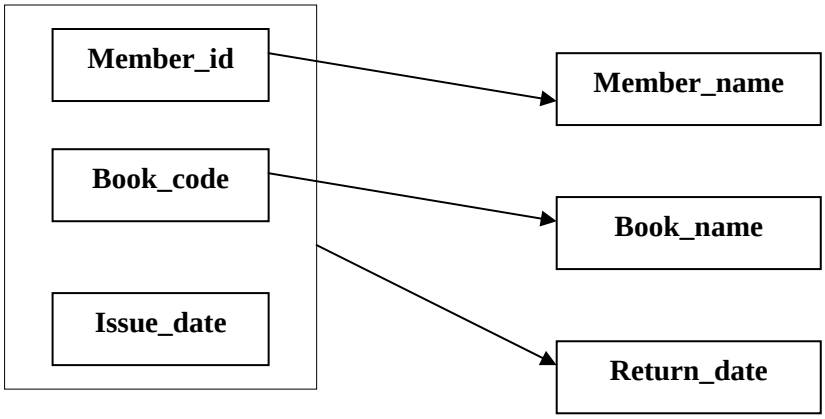
Why is the attribute issue_date on the left hand of the FD above?
Because a student for example Abishek can be issued the book having the book_code 0050 again on a later date, let say in February after Raman has returned it. Also note that return_date may have NULL value, but that is allowed in the FD. FD only necessitates that the value may be NULL or any specific data that is consistent across the instance of the relation.

Some interesting domain and procedural constraints in this database are:

- A book cannot be issued again unless it is returned
- A book can be returned only after the date on which it has been issued.
- A member can be issued a limited/maximum number of books at a time.

You will study in Book 2 Unit 1, how to put some of these constraints into domains using SQL.

3) The table is in 1NF. Let us draw the dependency diagram for the table.



1NF to 2NF:

The member_name and book_name attributes are dependent on part of the key so the decomposition based on specific FDs would be:

MEMBER(member_id, member_name) [Reason: FD (1)]
BOOK (book_code, book_name) [Reason: FD (2)]
ISSUE_RETURN (member_id, book_code, issue_date, return_date)
[Reason: FD (3)]

2NF to 3 NF and BCNF:

All the relations are in 3NF and BCNF also. As there is no dependence among non-key attributes and there is no overlapping candidate key.

Please note that the decomposed relations have no anomalies. Let us map the relation instance here:

MEMBER

Member - ID	Members Name
A 101	Abhishek
R 102	Raman

BOOK

Book - Code	Book – Name
0050	DBMS
0060	Multimedia
0125	OS

ISSUE_RETURN

Member - ID	Book - Code	Issue - Date	Return – Date
A 101	0050	15/01/05	25/01/05
R 102	0125	25/01/05	29/01/05
A 101	0060	20/01/05	NULL
R 102	0050	28/01/05	NULL

i) There is no redundancy, so no update anomaly is possible in the relations above.

ii) The insertion of new book and new member can be done in the BOOK and MEMBER tables respectively without any issue of book to member or vice versa. So no insertion anomaly.

iii) Even on deleting record 3 from ISSUE_RETURN it will not delete the information the book 0060 titled as multimedia as this information is in separate table. So no deletion anomaly.

Check Your Progress 3

1) Dependency preservation within a relation helps in enforcing constraints that are implied by dependency over a single relation. In case you do not preserve dependency then constraints might be enforced on more than one relation that is quite troublesome and time consuming.

2) A lossless decomposition is one in which you can reconstruct the original table without loss of information that means exactly the same tuples are obtained on taking join of the relations obtained on decomposition. Lossless decomposition requires that decomposition should be carried out on the basis of FDs.

3) The steps of Normalisation are:

   1. Remove repeating groups of each of the multi-valued attributes.
   2. Then remove redundant data and its dependence on part key.
   3. Remove columns from the table that are not dependent on the key that is remove transitive dependency.
   4. Check if there are overlapping candidate keys. If yes, check for any duplicate information, and remove columns that cause it.

UNIT 4 FILE ORGANISATION IN DBMS

Structure **Page Nos.**

4.0 INTRODUCTION

In earlier units, we studied the concepts of database integrity and how to normalise the tables. Databases are used to store information. Normally, the principal operations we need to perform on database are those relating to:

- Creation of data
- Retrieving data
- Modifying
- Deleting some information which we are sure is no longer useful or valid.

We have seen that in terms of the logical operations to be performed on the data, relational tables provide a good mechanism for all of the above tasks. Therefore the storage of a database in a computer memory (on the hard disk, of course), is mainly concerned with the following issues:

- The need to store a set of tables, where each table can be stored as an independent file.
- The attributes in a table are closely related and, therefore, often accessed together. Therefore it makes sense to store the different attribute values in each record contiguously. In fact, is it necessary that the attributes must be stored in the same sequence, for each record of a table?
- It seems logical to store all the records of a table contiguously. However, since there is no prescribed order in which records must be stored in a table, we may choose the sequence in which we store the different records of a table.

We shall see that the point (iii) among these observations is quite useful. Databases are used to store information in the form of files of records and are typically stored on magnetic disks. This unit focuses on the file Organisation in DBMS, the access methods available and the system parameters associated with them. File Organisation is the way the files are arranged on the disk and access method is how the data can be retrieved based on the file Organisation.

4.1 OBJECTIVES

After going through this unit you should be able to:

- define storage of databases on hard disks;
- discuss the implementation of various file Organisation techniques;
- discuss the advantages and the limitation of the various file Organisation techniques;
- describe various indexes used in database systems, and
- define the multi-key file organisation.

4.2 PHYSICAL DATABASE DESIGN ISSUES

The database design involves the process of logical design with the help of E-R diagram, normalisation, etc., followed by the physical design.

The Key issues in the Physical Database Design are:

- The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data for the DBMS.
- The goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security and recoverability.

Some of the basic inputs required for Physical Database Design are:

- Normalised relations
- Attribute definitions
- Data usage: entered, retrieved, deleted, updated
- Requirements for security, backup, recovery, retention, integrity
- DBMS characteristics.
- Performance criteria such as response time requirement with respect to volume estimates.

However, for such data some of the Physical Database Design Decisions that are to be taken are:

- Optimising attribute data types.
- Modifying the logical design.
- Specifying the file Organisation.
- Choosing indexes.

Designing the fields in the data base

The following are the considerations one has to keep in mind while designing the fields in the data base.

- Choosing data type
- Coding, compression, encryption
- Controlling data integrity
- Default value
  - Range control
  - Null value control
  - Referential integrity
- Handling missing data
  - Substitute an estimate of the missing value
  - Trigger a report listing missing values
  - In programs, ignore missing data unless the value is significant.

Physical Records

These are the records that are stored in the secondary storage devices. For a database relation, physical records are the group of fields stored in adjacent memory locations

and retrieved together as a unit. Considering the page memory system, data page is the amount of data read or written in one I/O operation to and from secondary storage device to the memory and vice-versa. In this context we define a term blocking factor that is defined as the number of physical records per page.

The issues relating to the Design of the Physical Database Files

Physical File is a file as stored on the disk. The main issues relating to physical files are:

- Constructs to link two pieces of data:
  - Sequential storage.
  - Pointers.
- File Organisation: How the files are arranged on the disk?
- Access Method: How the data can be retrieved based on the file Organisation?

Let us see in the next section how the data is stored on the hard disks.

4.3 STORAGE OF DATABASE ON HARD DISKS

At this point, it is worth while to note the difference between the terms file Organisation and the access method. A file organisation refers to the organisation of the data of a file into records, blocks, and access structures; this includes the way records and blocks are placed on the storage medium and interlinked. An access method, on the other hand, is the way how the data can be retrieved based on the file Organisation.

Mostly the databases are stored persistently on magnetic disks for the reasons given below:

- The databases being very large may not fit completely in the main memory.
- Storing the data permanently using the non-volatile storage and provide access to the users with the help of front end applications.
- Primary storage is considered to be very expensive and in order to cut short the cost of the storage per unit of data to substantially less.

Each hard drive is usually composed of a set of disk platters. Each disk platter has a layer of magnetic material deposited on its surface. The entire disk can contain a large amount of data, which is organised into smaller packages called BLOCKS (or pages). On most computers, one block is equivalent to 1 KB of data (= 1024 Bytes).

A block is the smallest unit of data transfer between the hard disk and the processor of the computer. Each block therefore has a fixed, assigned, address. Typically, the computer processor will submit a read/write request, which includes the address of the block, and the address of RAM in the computer memory area called a buffer (or cache) where the data must be stored / taken from. The processor then reads and modifies the buffer data as required, and, if required, writes the block back to the disk. Let us see how the tables of the database are stored on the hard disk.

How are tables stored on Disk?

We realise that each record of a table can contain different amounts of data. This is because in some records, some attribute values may be 'null'. Or, some attributes may be of type varchar (), and therefore each record may have a different length string as the value of this attribute. Therefore, the record is stored with each subsequent attribute separated by the next by a special ASCII character called a field separator. Of course, in each block, we may place many records. Each record is separated from the next, again by another special ASCII character called the record separator. Let us see in the next section about the types of file Organisation briefly.

4.4 FILE ORGANISATION AND ITS TYPES

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 secondary memory. A file organisation is a technique to organise data in the secondary memory. In this section, 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.
File Organisation is a way of arranging the records in a file when the file is stored on the disk. Data files are organized so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage is required to make it possible. Selection of File Organisations is dependant on two factors as shown below:

- Typical DBMS applications need a small subset of the DB at any given time.
- When a portion of the data is needed it must be located on disk, copied to memory for processing and rewritten to disk if the data was modified.

A file of record is likely to be accessed and modified in a variety of ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently. A DBMS supports several file Organisation techniques. The important task of the DBA is to choose a good Organisation for each file, based on its type of use.

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 *Figure1* illustrates different file organisations based on an access key.

Figure 1: File Organisation techniques

Let us discuss some of these techniques in more detail:

4.4.1 Heap files (unordered file)

Basically these files are unordered files. It is the simplest and most basic type. These files consist of randomly ordered records. The records will have no particular order. The operations we can perform on the records are insert, retrieve and delete. The features of the heap file or the pile file Organisation are:

- New records can be inserted in any empty space that can accommodate them.
- When old records are deleted, the occupied space becomes empty and available for any new insertion.
- If updated records grow; they may need to be relocated (moved) to a new empty space. This needs to keep a list of empty space.

Advantages of heap files

1. This is a simple file Organisation method.
2. Insertion is somehow efficient.
3. Good for bulk-loading data into a table.
4. Best if file scans are common or insertions are frequent.

Disadvantages of heap files

1. Retrieval requires a linear search and is inefficient.
2. Deletion can result in unused space/need for reorganisation.

4.4.2 Sequential File Organisation

The most basic way to organise the collection of records in a file is to use sequential Organisation. Records of the file are stored in sequence by the primary key field values. They are accessible only in the order stored, i.e., in the primary key order. This kind of file Organisation works well for tasks which need to access nearly every record in a file, e.g., payroll. Let us see the advantages and disadvantages of it.
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

A sequential file maintains the records in the logical sequence of its primary key values. Sequential files are inefficient for random access, however, are suitable for sequential access. A sequential file can be stored on devices like magnetic tape that allow sequential access.

On an average, to search a record in a sequential file would require to look into half of the records of the file. However, if a sequential file is stored on a disk (remember disks support direct access of its blocks) with keyword stored separately from the rest of record, then only those disk blocks need to be read that contains the desired record

or records. This type of storage allows binary search on sequential file blocks, thus, enhancing the speed of access.

Updating a sequential file usually creates a new file so that the record sequence on primary key is maintained. The update operation first copies the records till the record after which update is required into the new file and then the updated record is put followed by the remainder of records. Thus method of updating a sequential file automatically creates a backup copy.

Additions in the sequential files are also handled in a similar manner to update. Adding a record requires shifting of all records from the point of insertion to the end of file to create space for the new record. On the other hand deletion of a record requires a compression of the file space.

The basic advantages of sequential file is the sequential processing, as next record is easily accessible despite the absence of any data structure. However, simple queries are time consuming for large files. A single update is expensive as new file must be created, therefore, to reduce the cost per update, all updates requests are sorted in the order of the sequential file. This update file is then used to update the sequential file in a single go. The file containing the updates is sometimes referred to as a transaction file.

This process is called the batch mode of updating. In this mode each record of master sequential file is checked for one or more possible updates by comparing with the update information of transaction file. The records are written to new master file in the sequential manner. A record that require multiple update is written only when all the updates have been performed on the record. A record that is to be deleted is not written to new master file. Thus, a new updated master file will be created from the transaction file and old master file.

Thus, update, insertion and deletion of records in a sequential file require a new file creation. Can we reduce creation of this new file? Yes, it can easily be done if the original sequential file is created with holes which are empty records spaces as shown in the *Figure 3*. Thus, a reorganisation can be restricted to only a block that can be done very easily within the main memory. Thus, holes increase the performance of sequential file insertion and deletion. This organisation also support a concept of overflow area, which can store the spilled over records if a block is full. This technique is also used in index sequential file organisation. A detailed discussion on it can be found in the further readings.

Figure 3: A file with empty spaces for record insertions

Advantages of Sequential File Organisation

- It is fast and efficient when dealing with large volumes of data that need to be processed periodically (batch system).

Disadvantages of sequential File Organisation

- Requires that all new transactions be sorted into the proper sequence for sequential access processing.
- Locating, storing, modifying, deleting, or adding records in the file require rearranging the file.
- This method is too slow to handle applications requiring immediate updating or responses.

4.4.3 Indexed (Indexed Sequential) File Organisation

It organises the file like a large dictionary, i.e., records are stored in order of the key but an index is kept which also permits a type of direct access. The records are stored sequentially by primary key values and there is an index built over the primary key field.

The retrieval of a record from a sequential file, on average, requires access to half the records in the file, making such inquiries 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 index value, address pairs. Indexing associates a set of objects to a set of orderable quantities, that are usually smaller in number or their properties. Thus, an index is a mechanism for faster search. Although the indices and the data blocks are kept together physically, they are logically distinct. Let us use the term an index file to describes the indexes and let us refer to data files as data records. An index can be small enough to be read into the main memory.

A sequential (or sorted on primary keys) file that is indexed on its primary key is called an index sequential file. The index allows for random access to records, while the sequential storage of the records of the file provides easy access to the sequential records. An additional feature of this file system is the over flow area. The overflow area provides additional space for record addition without the need to create.

4.4.4 Hashed File Organisation

Hashing is the most common form of purely random access to a file or database. It is also used to access columns that do not have an index as an optimisation technique. Hash functions calculate the address of the page in which the record is to be stored based on one or more fields in the record. The records in a hash file appear randomly distributed across the available space. It requires some hashing algorithm and the technique. Hashing Algorithm converts a primary key value into a record address. The most popular form of hashing is division hashing with chained overflow.

Advantages of Hashed file Organisation

1. Insertion or search on hash-key is fast.
2. Best if equality search is needed on hash-key.

Disadvantages of Hashed file Organisation

1. It is a complex file Organisation method.
2. Search is slow.
3. It suffers from disk space overhead.
4. Unbalanced buckets degrade performance.
5. Range search is slow.

☞ Check Your Progress 1

1) Mention the five operations which show the performance of a sequential file Organisation along with the comments.

……

……

2) What are Direct-Access systems? What can be the various strategies to achieve this?

……

……

……

3) What is file Organisation and what are the essential factors that are to be considered?

……

……

……

……

4.5 TYPES OF INDEXES

One of the term used during the file organisation is the term index. In this section, let us define this term in more detail.

We find the index of keywords at the end of each book. Notice that this index is a sorted list of keywords (index values) and page numbers (address) where the keyword can be found. In databases also an index is defined in a similar way, as the <index value, address> pair.

The basic advantage of having sorted index pages at the end of the book is that we can locate a desired keyword in the book. We could have used the topic and sub-topic listed in the table of contents, but it is not necessary that the given keyword can be found there; also they are not in any sorted sequence. If a keyword is not even found in the table of contents then we need to search each of the pages to find the required keyword, which is going to be very cumbersome. Thus, an index at the back of the book helps in locating the required keyword references very easily in the book.

The same is true for the databases that have very large number of records. A database index allows fast search on the index value in database records. It will be difficult to locate an attribute value in a large database, if index on that value is not provided. In such a case the value is to be searched record-by-record in the entire database which is cumbersome and time consuming. It is important to note here that for a large database the entire records cannot be kept in the main memory at a time, thus, data needs to be transferred from the secondary storage device which is more time consuming. Thus, without an index it may be difficult to search a database.

An index contains a pair consisting of index value and a list of pointers to disk block for the records that have the same index value. An index contains such information for every stored value of index attribute. An index file is very small compared to a data file that stores a relation. Also index entries are ordered, so that an index can be searched using an efficient search method like binary search. In case an index file is very large, we can create a multi-level index, that is index on index. Multi-level indexes are defined later in this section.

There are many types of indexes those are categorised as:

Primary index	Single level index	Spare index
Secondary index	Multi-level index	Dense index
Clustering index		

A primary index is defined on the attributes in the order of which the file is stored. This field is called the ordering field. A primary index can be on the primary key of a file. If an index is on attributes other than candidate key fields then several records may be related to one ordering field value. This is called clustering index. It is to be noted that there can be only one physical ordering field. Thus, a file can have either the primary index or clustering index, not both. Secondary indexes are defined on the

non-ordering fields. Thus there can be several secondary indexes in a file, but only one primary or clustering index.

Primary index

A primary index is a file that contains a sorted sequence of records having two columns: the ordering key field; and a block address for that key field in the data file. The ordering key field for this index can be the primary key of the data file. Primary index contains one index entry for each value of the ordering key field. An entry in primary index file contains the index value of the first record of the data block and a pointer to that data block.

Let us discuss primary index with the help of an example. Let us assume a student database as (Assuming that one block stores only four student records.):

	Enrolment Number	Name	City	Progra-mme
BLOCK 1	2109348	ANU VERMA	CHENNAI	CIC
	2109349	ABHISHEK KUMAR	CALCUTTA	MCA
	2109351	VIMAL KISHOR	KOCHI	BCA
	2109352	RANJEETA JULIE	KOCHI	CIC
BLOCK 2	2109353	MISS RAJIYA BANU	VARANASI	MBA
	2238389	JITENDAR KASWAN	NEW DELHI	MBA
	2238390	RITURAJ BHATI	VARANASI	MCA
	2238411	AMIT KUMAR JAIN	NEW DELHI	BCA
BLOCK 3	2238412	PAWAN TIWARI	AJMER	MCA
	2238414	SUPRIYA SWAMI	NEW DELHI	MCA
	2238422	KAMLESH KUMAR	MUMBAI	BSC
	2258014	DAVEN SINGHAL	MUMBAI	BCA
BLOCK 4	2258015	S SRIVASTAVA	MUMBAI	BCA
	2258017	SHWETA SINGH	NEW DELHI	BSC
	2258018	ASHISH TIWARI	MUMBAI	MCA
	2258019	SEEMA RANI	LUCKNOW	MBA
…	…	…	…	…
BLOCK r	2258616	NIDHI	AJMER	BCA
	2258617	JAGMEET SINGH	LUCKNOW	MCA
	2258618	PRADEEP KUMAR	NEW DELHI	BSC
	2318935	RAMADHAR	FARIDABAD	MBA
…	…	…	…	…
BLOCK N-1	2401407	BRIJMISHRA	BAREILLY	CIC
	2401408	AMIT KUMAR	BAREILLY	BSC
	2401409	MD. IMRAN SAIFI	AURANGABAD	BCA
	2401623	ARUN KUMAR	NEW DELHI	MCA
BLOCK N	2401666	ABHISHEK RAJPUT	MUMBAI	MCA
	2409216	TANNUJ SETHI	LUCKNOW	MBA
	2409217	SANTOSH KUMAR	ALMORA	BCA
	2409422	SAKSHI GINOTRA	MUMBAI	BSC

Figure 4: A Student file stored in the order of student enrolment numbers

The primary index on this file would be on the ordering field – enrolment number. The primary index on this file would be:

Figure 5: The Student file and the Primary Index on Enrolment Number

Please note the following points in the figure above.

- An index entry is defined as the attribute value, pointer to the block where that record is stored. The pointer physically is represented as the binary address of the block.
- Since there are four student records, which of the key value should be stored as the index value? We have used the first key value stored in the block as the index key value. This is also called the anchor value. All the records stored in the given block have ordering attribute value as the same or more than this anchor value.
- The number of entries in the primary index file is the same as the number of disk block in the ordered data file. Therefore, the size of the index file is small. Also notice that not all the records need to have an entry in the index file. This type of index is called non-dense index. Thus, the primary index is non-dense index.

- To locate the record of a student whose enrolment number is 2238422, we need to find two consecutive entries of indexes such that index value1 < 2238422 < index value 2. In the figure above we find the third and fourth index values as: 2238412 and 2258015 respectively satisfying the properties as above. Thus, the required student record must be found in Block 3.

But, does primary index enhance efficiency of searching? Let us explain this with the help of an example (Please note we will define savings as the number of block transfers as that is the most time consuming operation during searching).

Example 1: An ordered student file (ordering field is enrolment number) has 20,000 records stored on a disk having the Block size as 1 K. Assume that each student record is of 100 bytes, the ordering field is of 8 bytes, and block pointer is also of 8 bytes, find how many block accesses on average may be saved on using primary index.

Answer:

Number of accesses without using Primary Index:

Number of records in the file = 20000
Block size = 1024 bytes
Record size = 100 bytes
Number of records per block = integer value of [1024 / 100] = 10
Number of disk blocks acquired by the file = [Number of records / records per block]
= [20000/10] = 2000
Assuming a block level binary search, it would require $\log_2 2000$ = about11 block accesses.

Number of accesses with Primary Index:

Size of an index entry = 8+8 = 16 bytes

Number of index entries that can be stored per block

= integer value of [1024 / 16] = 64

Number of index entries = number of disk blocks = 2000

Number of index blocks = ceiling of [2000/ 64] = 32

Number of index block transfers to find the value in index blocks = $\log_2 32 = 5$

One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with primary index = 5 + 1 = 6.

Thus, the Primary index would save about 5 block transfers for the given case.

Is there any disadvantage of using primary index? Yes, a primary index requires the data file to be ordered, this causes problems during insertion and deletion of records in the file. This problem can be taken care of by selecting a suitable file organisation that allows logical ordering only.

Clustering Indexes.

It may be a good idea to keep records of the students in the order of the programme they have registered as most of the data file accesses may require student data of a particular programme only. An index that is created on an ordered file whose records of a file are physically ordered on a non-key field (that is the field does not have a distinct value for each record) is called a clustering index. *Figures 6 & 7* show the clustering indexes in the same file organised in different ways.

Figure 6: A clustering Index on Programme in the Student file

Please note the following points about the clustering index as shown in the *Figure 6*.

- The clustering index is an ordered file having the clustering index value and a block pointer to the first block where that clustering field value first appears.
- Clustering index is also a sparse index. The size of clustering index is smaller than primary index as far as number of entries is concerned.

In the Figure 6 the data file have blocks where multiple Programme students exist. We can improve upon this organisation by allowing only one Programme data in one block. Such an organisation and its clustering index is shown in the following *Figure 7*:

Figure 7: Clustering index with separate blocks for each group of records with the same value for the clustering field

Please note the following points in the tables:

- Data insertion and deletion is easier than in the earlier clustering files, even now it is cumbersome.
- The additional blocks allocated for an index entry are in the form of linked list blocks.

- Clustering index is another example of a non-dense index as it has one entry for every distinct value of the clustering index field and not for every record in the file.

Secondary Indexes

Consider the student database and its primary and clustering index (only one will be applicable at a time). Now consider the situation when the database is to be searched or accessed in the alphabetical order of names. Any search on a student name would require sequential data file search, thus, is going to be very time consuming. Such a search on an average would require reading of half of the total number of blocks. Thus, we need secondary indices in database systems. A secondary index is a file that contains records containing a secondary index field value which is not the ordering field of the data file, and a pointer to the block that contains the data record. Please note that although a data file can have only one primary index (as there can be only one ordering of a database file), it can have many secondary indices.

Secondary index can be defined on an alternate key or non-key attributes. A secondary index that is defined on the alternate key will be dense while secondary index on non-key attributes would require a bucket of pointers for one index entry. Let us explain them in more detail with the help of *Figures 8*.

Figure 8: A dense secondary Index on a non-ordering key field of a file (The file is organised on the clustering field "Programme"

Please note the following in the *Figure 8*.

- The names in the data file are unique and thus are being assumed as the alternate key. Each name therefore is appearing as the secondary index entry.
- The pointers are block pointers, thus are pointing to the beginning of the block and not a record. For simplicity of the figure we have not shown all the pointers
- This type of secondary index file is dense index as it contains one entry for each record/district value.
- The secondary index is larger than the Primary index as we cannot use block anchor values here as the secondary index attributes are not the ordering attribute of the data file.
- To search a value in a data file using name, first the index file is (binary) searched to determine the block where the record having the desired key value can be found. Then this block is transferred to the main memory where the desired record is searched and accessed.
- A secondary index file is usually larger than that of primary index because of its larger number of entries. However, the secondary index improves the search time to a greater proportion than that of primary index. This is due to the fact that if primary index does not exist even then we can use binary search on the blocks as the records are ordered in the sequence of primary index value. However, if a secondary key does not exist then you may need to search the records sequentially. This fact is demonstrated with the help of Example 2.

Example 2: Let us reconsider the problem of Example 1 with a few changes. An un-ordered student file has 20,000 records stored on a disk having the Block size as 1 K. Assume that each student record is of 100 bytes, the secondary index field is of 8 bytes, and block pointer is also of 8 bytes, find how many block accesses on average may be saved on using secondary index on enrolment number.

Answer:

Number of accesses without using Secondary Index:

Number of records in the file = 20000
Block size = 1024 bytes
Record size = 100 bytes
Number of records per block = integer value of [1024 / 100] = 10
Number of disk blocks acquired by the file = [Number of records / records per block]
= [20000/10] = 2000
Since the file is un-ordered any search on an average will require about half of the above blocks to be accessed. Thus, average number of block accesses = 1000

Number of accesses with Secondary Index:

Size of an index entry = 8+8 = 16 bytes
Number of index entries that can be stored per block
= integer value of [1024 / 16] = 64
Number of index entries = number of records = 20000
Number of index blocks = ceiling of [20000/ 64] = 320
Number of index block transfers to find the value in index blocks =
ceiling of $[\log_2 320] = 9$

One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = 9 + 1 = 10

Thus, the Secondary index would save about 1990 block transfers for the given case. This is a huge saving compared to primary index. Please also compare the size of secondary index to primary index.

Let us now see an example of a secondary index that is on an attribute that is not an alternate key.

Figure 9: A secondary index on a non-key field implemented using one level of indirection so that index entries are fixed length and have unique field values (The file is organised on the primary key.

A secondary index that needs to be created on a field that is not a candidate key can be implemented using several ways. We have shown here the way in which a block of pointer records is kept for implementing such index. This method keeps the index entries at a fixed length. It also allows only a single entry for each index field value. However, this method creates an extra level of indirection to handle the multiple pointers. The algorithms for searching the index, inserting and deleting new values into an index are very simple in such a scheme. Thus, this is the most popular scheme for implementing such secondary indexes.

Sparse and Dense Indexes

As discussed earlier an index is defined as the ordered <index value, address> pair. These indexes in principle are the same as that of indexes used at the back of the book. The key ideas of the indexes are:

- They are sorted on the order of the index value (ascending or descending) as per the choice of the creator.
- The indexes are logically separate files (just like separate index pages of the book).
- An index is primarily created for fast access of information.
- The primary index is the index on the ordering field of the data file whereas a secondary index is the index on any other field, thus, more useful.

But what are sparse and dense indexes?

Sparse indices are those indices that do not include all the available values of a field. An index groups the records as per the index values. A sparse index is the one where the size of the group is one or more, while in a dense index the size of the group is 1. In other words a dense index contains one index entry for every value of the indexing attributes, whereas a sparse index also called non-dense index contains few index entires out of the available indexing attribute values. For example, the primary index on enrolment number is sparse, while secondary index on student name is dense.

Multilevel Indexing Scheme

Consider the indexing scheme where the address of the block is kept in the index for every record, for a small file, this index would be small and can be processed efficiently in the main memory. However, for a large file the size of index can also be very large. In such a case, one can create a hierarchy of indexes with the lowest level index pointing to the records, while the higher level indexes point to the indexes on indexes. The following figure shows this scheme.

Figure 10: Hierarchy of Indexes

Please note the following points about the multi-level indexes:

- The lowest level index points to each record in the file; thus is costly in terms of space.
- Updating, insertion and deletion of records require changes to the multiple index files as well as the data file. Thus, maintenance of the multi-level indexes is also expensive.

After discussing so much about the indexes let us now turn our attention to how an index can be implemented in a database. The indexes are implemented through B Trees. The following section examines the index implementation in more detail.

4.6 INDEX AND TREE STRUCTURE

Let us discuss the data structure that is used for creating indexes.

Can we use Binary Search Tree (BST) as Indexes?

Les us first reconsider the binary search tree. A BST is a data structure that has a property that all the keys that are to the left of a node are smaller than the key value of the node and all the keys to the right are larger than the key value of the node.

To search a typical key value, you start from the root and move towards left or right depending on the value of key that is being searched. Since an index is a <value, address> pair, thus while using BST, we need to use the value as the key and address field must also be specified in order to locate the records in the file that is stored on the secondary storage devices. The following figure demonstrates the use of BST index for a University where a dense index exists on the enrolment number field.

Figure 11: The Index structure using Binary Search Tree

Please note in the figure above that a key value is associated with a pointer to a record. A record consists of the key value and other information fields. However, we don't store these information fields in the binary search tree, as it would make a very large tree. Thus, to speed up searches and to reduce the tree size, the information fields of records are commonly stored into files on secondary storage devices. The connection between key values in the BST to its corresponding record in the file is established with the help of a pointer as shown in *Figure11*. Please note that the BST structure is key value, address pair.

Now, let us examine the suitability of BST as a data structure to implement index. A BST as a data structure is very much suitable for an index, if an index is to be contained completely in the primary memory. However, indexes are quite large in nature and require a combination of primary and secondary storage. As far as BST is concerned it might be stored level by level on a secondary storage which would require the additional problem of finding the correct sub-tree and also it may require a number of transfers, with the worst condition as one block transfer for each level of a tree being searched. This situation can be drastically remedied if we use B -Tree as data structure.

A B-Tree as an index has two advantages:

- It is completely balanced
- Each node of B-Tree can have a number of keys. Ideal node size would be if it is somewhat equal to the block size of secondary storage.

The question that needs to be answered here is what should be the order of B-Tree for an index. It ranges from 80-200 depending on various index structures and block size.

Let us recollect some basic facts about B-Trees indexes.

The basic B-tree structure was discovered by R.Bayer and E.McCreight (1970) of Bell Scientific Research Labs and has become one of the popular structures for organising an index structure. Many variations on the basic B-tree structure have been developed.

The B-tree is a useful balanced sort-tree 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 techniques of organising a file and its index called the B-tree is the standard Organisation for indexes in a database system."

A B-tree of order N is a tree in which:

- Each node has a maximum of N children and a minimum of the ceiling of [N/2] children. However, the root node of the tree can have 2 to N children.
- Each node can have one fewer keys than the number of children, but a maximum of N-1 keys can be stored in a node.
- The keys are normally arranged in an increasing order. All keys in the sub tree to the left of a key are less than the key, and all the keys in the sub-tree to the right of a key are higher then the value of the key.
- If a new key is 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. If the parent node is the root, a new root node is created.
- All the leaves of B-tree are on the same level. There is no empty sub-tree above the level of the leaves. Thus a B-tree is completely balanced.

Figure 12: A B-Tree as an index

A B-Tree index is shown in Figure 12. The B-Tree has a very useful variant called B+Tree, which have all the key values at the leaf level also in addition to the higher level. For example, the key value 1010 in *Figure 12* will also exist at leaf level. In addition, these lowest level leaves are linked through pointers. Thus, B+tree is a very useful structure for index-sequential organisation. You can refer to further readings for more detail on these topics.

4.7 MULTI-KEY FILE ORGANISATION

Till now we have discussed file organisations having the single access key. But is it possible to have file organisations that allows access of records on more than one key field? In this section, we will introduce two basic file Organisation schemes that allow records to be accessed by more than one key field, thus, allowing multiple access paths each having a different key. These are called multi-key file Organisations. These file organisation techniques are at the heart of database implementation.

There are numerous techniques that have been used to implement multi-key file Organisation. Most of these techniques are based on building indexes to provide direct access by the key value. Two of the commonest techniques for this Organisation are:

- Multi-list file Organisation
- Inverted file Organisation

Let us discuss these techniques in more detail. But first let us discuss the need for the Multiple access paths.

4.7.1 Need for Multiple Access Paths

In practice, most of the online information systems require the support of multi-key files. For example, consider a banking database application having many different kinds of users such as:

- Teller
- Loan officers
- Brach manager
- Account holders

All of these users access the bank data however in a different way. Let us assume a sample data format for the Account relation in a bank as:

Account Relation:

Account Number	Account Holder Name	Branch Code	Account type	Balance	Permissible Loan Limit

A teller may access the record above to check the balance at the time of withdrawal. S/he needs to access the account on the basis of branch code and account number. A loan approver may be interested in finding the potential customer by accessing the records in decreasing order of permissible loan limits. A branch manager may need to find the top ten most preferred customers in each category of account so may access the database in the order of account type and balance. The account holder may be interested in his/her own record. Thus, all these applications are trying to refer to the same data but using different key values. Thus, all the applications as above require the database file to be accessed in different format and order. What may be the most efficient way to provide faster access to all such applications? Let us discuss two approaches for this:

- By Replicating Data
- By providing indexes.

Replicating Data

One approach that may support efficient access to different applications as above may be to provide access to each of the applications from different replicated files of the data. Each of the file may be organised in a different way to serve the requirements of a different application. For example, for the problem above, we may provide an indexed sequential account file having account number as the key to bank teller and the account holders. A sequential file in the order of permissible loan limit to the Loan officers and a sorted sequential file in the order of balance to branch manager.
All of these files thus differ in the organisation and would require different replica for different applications. However, the Data replication brings in the problems of inconsistency under updating environments. Therefore, a better approach for data access for multiple keys has been proposed.

Support by Adding Indexes

Multiple indexes can be used to access a data file through multiple access paths. In such a scheme only one copy of the data is kept, only the number of paths is added with the help of indexes. Let us discuss two important approaches in this category: Multi-List file organisation and Inverted file organisation.

4.7.2 Multi-list file Organisation

Multi-list file organisation is a multi-index linked file organisation. A linked file organisation is a logical organisation where physical ordering of records is not of concern. In linked organisation the sequence of records is governed by the links that determine the next record in sequence. Linking of records can be unordered but such a linking is very expensive for searching of information from a file. Therefore, it may be a good idea to link records in the order of increasing primary key. This will facilitate insertion and deletion algorithms. Also this greatly helps the search performance. In addition to creating order during linking, search through a file can be further facilitated by creating primary and secondary indexes. All these concepts are supported in the multi-list file organisation. Let us explain these concepts further with the help of an example.

Consider the employee data as given in *Figure 13*. The record numbers are given as alphabets for better description. Assume that the Empid is the key field of the data records. Let us explain the Multi-list file organisation for the data file.

Record Number	Empid	Name	Job	Qualification	Gender	City	Married/ Single	Salary
A	800	Jain	Software Engineer	B. Tech.	Male	New Delhi	Single	15,000/-
B	500	Inder	Software Manager	B. Tech.	Female	New Delhi	Married	18,000/-
C	900	Rashi	Software Manager	MCA	Female	Mumbai	Single	16,000/-
D	700	Gurpreet	Software Engineer	B. Tech.	Male	Mumbai	Married	12,000/-
E	600	Meena	Software Manager	MCA	Female	Mumbai	Single	13,000/-

Figure 13: Sample data for Employee file

Since, the primary key of the file is Empid, therefore the linked order of records should be defined as B (500), E(600), D(700), A(800), C(900). However, as the file size will grow the search performance of the file would deteriorate. Therefore, we can create a primary index on the file (please note that in this file the records are in the logical sequence and tied together using links and not physical placement, therefore, the primary index will be a linked index file rather than block indexes).

Let us create a primary index for this file having the Empid values in the range:

>= 500 but < 700
> = 700 but < 900
>= 900 but < 1100

The index file for the example data as per Figure 13 is shown in *Figure 14*.

Figure 14: Linking together all the records in the same index value.

Please note that in the *Figure 14*, those records that fall in the same index value range of Empid are linked together. These lists are smaller than the total range and thus will improve search performance.

This file can be supported by many more indexes that will enhance the search performance on various fields, thus, creating a multi-list file organisation. *Figure 15* shows various indexes and lists corresponding to those indexes. For simplicity we have just shown the links and not the complete record. Please note the original order of the nodes is in the order of Empid's.

Figure 15: Multi-list representation for figure 13

An interesting addition that has been done in the indexing scheme of multi-list organisation is that an index entry contains the length of each sub-list, in addition to the index value and the initial pointer to list. This information is useful when the query contains a Boolean expression. For example, if you need to find the list of Female employees who have MCA qualifications, you can find the results in two ways. Either you go to the Gender index, and search the Female index list for MCA qualification or you search the qualification index to find MCA list and in MCA list search for Female candidates. Since the size of MCA list is 2 while the size of Female list is 3 so the preferable search will be through MCA index list. Thus, the information about the length of the list may help in reducing the search time in the complex queries.

Consider another situation when the Female and MCA lists both are of about a length of 1000 and only 10 Female candidates are MCA. To enhance the search performance of such a query a combined index on both the fields can be created. The values for this index may be the Cartesian product of the two attribute values.

Insertion and deletion into multi-list is as easy/hard as is the case of list data structures. In fact, the performance of this structure is not good under heavy insertion and deletion of records. However, it is a good structure for searching records in case the appropriate index exist.

4.7.3 Inverted File Organisation

Inverted file organisation is one file organisation where the index structure is most important. In this organisation the basic structure of file records does not matter much. This file organisation is somewhat similar to that of multi-list file organisation with the key difference that in multi-list file organisation index points to a list, whereas in inverted file organisation the index itself contains the list. Thus, maintaining the proper index through proper structures is an important issue in the design of inverted file organisation. Let us show inverted file organisation with the help of data given in *Figure 13*.

Let us assume that the inverted file organisation for the data shown contains dense index. *Figure 16* shows how the data can be represented using inverted file organisation.

Figure 16: Some of the indexes for fully inverted file

Please note the following points for the inverted file organisation:

- The index entries are of variable lengths as the number of records with the same key value is changing, thus, maintenance of index is more complex than that of multi-list file organisation.
- The queries that involve Boolean expressions require accesses only for those records that satisfy the query in addition to the block accesses needed for the indices. For example, the query about Female, MCA employees can be solved by the Gender and Qualification index. You just need to take intersection of record numbers on the two indices. (Please refer to *Figure 16*). Thus, any complex query requiring Boolean expression can be handled easily through the help of indices.

Let us now finally differentiate between the two-multi-list and inverted file organisation**.**

Similarities:

Both organisations can support:

- An index for primary and secondary key
- The pointers to data records may be direct or indirect and may or may not be sorted.

Differences:

The indexes in the two organisations differ as:

- In a Multi-list organisation an index entry points to the first data record in the list, whereas in inverted index file an index entry has address pointers to all the data records related to it.
- A multi-list index has fixed length records, whereas an inverted index contains variable length records

However, the data records do not change in an inverted file organisation whereas in the multi-list file organisation a record contains the links, one per created index.

Some of the implications of these differences are:

- An index in a multi-list organisation can be managed easily as it is of fixed length.
- The query response of inverted file approach is better than multi-list as the query can be answered only by the index. This also helps in reducing block accesses.

4.8 IMPORTANCE OF FILE ORGANISATION IN DATABASE

To implement a database efficiently, there are several design tradeoffs needed. One of the most important ones is the file Organisation. For example, if there were to be an application that required only sequential batch processing, then the use of indexing techniques would be pointless and wasteful.

There are several important consequences of an inappropriate file Organisation being used in a database. Thus using replication would be wasteful of space besides posing the problem of inconsistency in the data. The wrong file Organisation can also–

- Mean much larger processing time for retrieving or modifying the required record
- Require undue disk access that could stress the hardware

Needless to say, these could have many undesirable consequences at the user level, such as making some applications impractical.

☞ Check Your Progress 2

1) What is the difference between BST-Tree and B tree indexes?

……..…

……..…

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

……..…

……..…

…….

3) **State or True or False** T/ F

a) A primary index is index on the primary key of an unordered data file ☐

b) A clustering index involves ordering on the non-key attributes. ☐

c) A primary index is a dense index. ☐

d) A non-dense index involves all the available attribute values of ☐

the index field.

e) Multi-level indexes enhance the block accesses than simple indexes. ☐

f) A file containing 40,000 student records of 100 bytes each having the 1k-block size. It has a secondary index on its alternate key of size 16 bits per index entry. For this file search through the secondary index will require 20 block transfers on an average. ☐

g) A multi-list file increases the size of the record as the link information is added to each record. ☐

h) An inverted file stores the list of pointers to records within the index. ☐

i) Multi-list organisation is superior than that of inverted organisation for queries having Boolean expression. ☐

4.9 SUMMARY

In this unit, we discussed the physical database design issues in which we had addressed the file Organisation and file access method. After that, we discussed the various file Organisations: Heap, Sequential, Indexed Sequential and Hash, their advantages and their disadvantages.

An index is a very important component of a database system as one of the key requirements of DBMS is efficient access to data, therefore, various types of indexes that may exist in database systems were explained in detail. Some of these are: Primary index, clustering index and secondary index. The secondary index results in better search performance, but adds on the task of index updating. In this unit, we also discussed two multi-key file organisations viz. multi-list and inverted file organisations. These are very useful for better query performance.

4.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Operation	Comments
File Creation	It will be efficient if transaction records are ordered by record key
Record Location	As it follows the sequential approach it is inefficient. On an average, half of the records in the file must be processed
Record Creation	It has to browse through all the records. Entire file must be read and write. Efficiency improves with greater number of additions. This operation could be combined with deletion and modification transactions to achieve greater efficiency.
Record Deletion	Entire file must be read and write. Efficiency improves with greater number of deletions. This operation could be combined with addition and modification transactions to achieve greater efficiency.
Record Modification	Very efficient if the number of records to be modified is high and the records in the transaction file are ordered by the record key.

2) Direct-access systems do not search the entire file; rather they move directly to the needed record. To be able to achieve this, several strategies like relative addressing, hashing and indexing can be used.

3) It is a technique for physically arranging the records of a file on secondary storage devices. When choosing the file Organisation, you should consider the following factors:

1. Data retrieval speed
2. Data processing speed
3. Efficiency usage of storage space
4. Protection from failures and data loss
5. Scalability
6. Security

Check Your Progress 2

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.

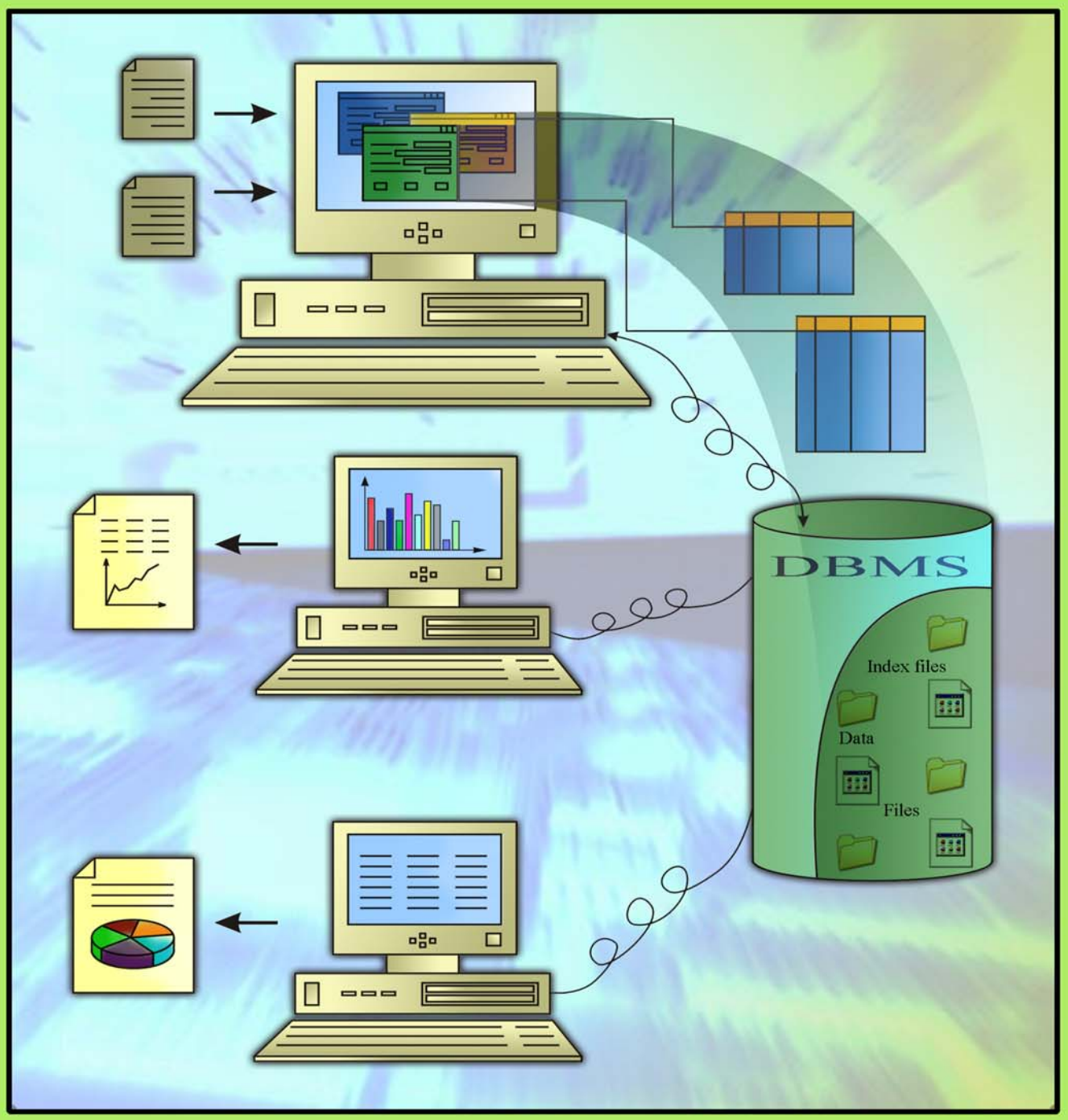
2) Sequential access to the keys of a B-tree is much slower than sequential access to the keys of a B+ tree, since the latter have all the keys linked in sequential order in the leave.

3) (a) False
(b) True
(c) False
(d) False
(e) False
(f) False
(g) True
(h) True
(i) False

SOCIAS-IGNOU/P.O.10.5T/MAY, 2004

Indira Gandhi National Open University
School of Computer and Information Sciences

MCS - 023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS



The Database Management System

2

Indira Gandhi
National Open University
School of Computer &
Information Sciences

MCS-024
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

Block

2

STRUCTURED QUERY LANGUAGE AND TRANSACTION MANAGEMENT

Programme / Course Design Committee

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Prof. M. Balakrishnan, IIT, Delhi
Prof. Harish Karnick, IIT, Kanpur
Prof. C. Pandurangan, IIT, Madras
Dr. Om Vikas, Sr. Director,
Ministry of ICT, Delhi
Prof. P. S. Grover, Sr. Consultant,
SOCIS, IGNOU

Faculty of School of Computer and Information Sciences

Shri Shashi Bhushan
Shri Akshay Kumar
Prof. Manohar Lal
Shri V.V. Subrahmanyam
Shri P.Venkata Suresh

Block Preparation Team

Dr. D.K. Lobiyal (Content Editor)
School of Computer & System Sciences
Jawaharlal Nehru University
New Delhi

Shri M.P. Mishra
SOCIS, IGNOU

Prof. Adarsh Kumar Verma
(Language Editor)

Course Coordinator: Shri M.P. Mishra

Block Production Team

Shri T.R. Manoj, Section Officer (Pub.) and Shri H.K. Som, Consultant

Acknowledgements

To Shri Akshay Kumar, SOCIS for his valuable comments and suggestions in Unit 1 and Unit 2.

April, 2005

ISBN—81-266-1747-0

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Director, SOCIS.

BLOCK INTRODUCTION

In the previous block, you have gone through the basic concepts of database and database management system. Some of the key concepts introduced in Block 1 were three level architecture, data independence, integrity constraints, normalisation, ER model and file organization. However, information about database management system would be incomplete if we do not discuss about how to get information into and out of them in a structured way, and transaction management and security. This block deals with theses issues of database Management system.

Unit 1 of this block is devoted to a standard query Language called Structured Query Language (SQL), which is the most acceptable standard across all commercial database management systems. This unit covers almost all the important features required for database creation, updating and retrieval. The unit also discusses about the complex queries.

Unit 2 discusses the concept of transaction and their management. Transactions are everywhere in the Industry, so how do we deal with transactions in database specially when concurrent transactions are going on? This unit attempts to provide the answer to this question.

Unit 3 provides information with respect to database security and recovery. These two topics are very important for you as they are mainly the responsibility of database administrator, a job where an MCA profile with some experience fits very well.

Unit 4 provides information about two important database approaches, the distributed database system and the client server system. The distributed database system is relational in nature so they are conceptually the logical extensions of relational system. Clint server model is one of the most acceptable implementation Model in the industry. Almost all commercial DBMSs are available in client server mode.

UNIT 1 THE STRUCTURED QUERY LANGUAGE

Structure **Page Nos.**

1.0 INTRODUCTION

Database is an organised collection of information about an entity having controlled redundancy and serves multiple applications. DBMS (database management system) is an application software that is developed to create and manipulate the data in database. A query language can easily access a data in a database. SQL (Structured Query Language) is language used by most relational database systems. IBM developed the SQL language in mid-1979. All communication with the clients and the RDBMS, or between RDBMS is via SQL. Whether the client is a basic SQL engine or a disguised engine such as a GUI, report writer or one RDBMS talking to another, SQL statements pass from the client to the server. The server responds by processing the SQL and returning the results. The advantage of this approach is that the only network traffic is the initial query and the resulting response. The processing power of the client is reserved for running the application.

SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL). It is a fourth generation language. SQL has many more features and advantages. Let us discuss the SQL in more detail in this unit. It should be noted that many commercial DBMS may or may not implement all the details given in this unit. For example, MS-ACCESS does not support some of these features. Even some of the constructs may not be portable, please consult the DBMS Help for any such difference.

1.1 OBJECTIVES

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

- create, modify and delete database schema objects;
- update database using SQL command;
- retrieve data from the database through queries and sub-queries;
- handle join operations;
- control database access;
- deal with database objects like Tables, Views, Indexes, Sequences, and Synonyms using SQL.

1.2 WHAT IS SQL?

Structured Query Language (SQL) is a standard query language. It is commonly used with all relational databases for data definition and manipulation.

All the relational systems support SQL, thus allowing migration of database from one DBMS to another. In fact, this is one of the reasons of the major success of Relational DBMS. A user may be able to write a program using SQL for an application that involves data being stored in more than one DBMSs, provided these DBMS support standard SQL. This feature is commonly referred to as portability. However, not all the features of SQL implemented in one RDBMS are available in others because of customization of SQL. However, please note there is just ONE standard SQL.

SQL provides an interface where the user can specify "What" are the expected results. The query execution plan and optimisation is performed by the DBMS. The query plan and optimisation determines how a query needs to be executed. For example, if three tables X, Y and Z are to be joined together then which plan (X JOIN Y) and then Z or X JOIN (Y JOIN Z) may be executed. All these decisions are based on statistics of the relations and are beyond the scope of this unit.

SQL is called a non-procedural language as it just specifies what is to be dome rather than how it is to be done. Also, since SQL is a higher-level query language, it is closer to a language like English. Therefore, it is very user friendly.

The American National Standard Institute (ANSI) has designed standard versions of SQL. The first standard in this series was created in 1986. It was called SQL-86 or SQL1. This standard was revised and enhanced later and SQL-92 or SQL-2 was released. A newer standard of SQL is SQL3 which is also called SQL- 99. In this unit we will try to cover features from latest standards. However, some features may be found to be very specific to certain DBMSs.

Some of the important features of SQL are:

- It is a non procedural language.
- It is an English-like language.
- It can process a single record as well as sets of records at a time.
- It is different from a third generation language (C& COBOL). All SQL statements define what is to be done rather than how it is to be done.
- SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL).
- It insulates the user from the underlying structure and algorithm.
- SQL has facilities for defining database views, security, integrity constraints, transaction controls, etc.

There are many variants of SQL, but the standard SQL is the same for any DBMS environment. The following table shows the differences between SQL and one of its superset SQL*Plus which is used in Oracle. This will give you an idea of how various vendors have enhanced SQL to an environment. The non-standard features of SQL are not portable across databases, therefore, should not be used preferably while writing SQL queries.

Difference between SQL and SQL*Plus

SQL	**SQL*Plus**
SQL is a language	SQL *Plus is an environment
It is based on ANSI (American National Standards Institute) standard	It is oracle proprietary interface for executing SQL statements

SQL	
It is entered into the SQL buffer on one or more lines	It is entered one line at a time, not stored in the SQL buffer
SQL cannot be abbreviated	SQL*Plus can be abbreviated
It uses a termination character to execute command immediately	It does not require termination character. Commands are executed immediately.
SQL statements manipulate data and table definition in the database	It does not allow manipulation of values in the database

1.3 DATA DEFINITION LANGUAGE

As discussed in the previous block, the basic storage unit of a relational database management system is a table. It organises the data in the form of rows and columns. But what does the data field column of table store? How do you define it using SQL?

The Data definition language (DDL) defines a set of commands used in the creation and modification of schema objects such as tables, indexes, views etc. These commands provide the ability to create, alter and drop these objects. These commands are related to the management and administrations of the databases. Before and after each DDL statement, the current transactions are implicitly committed, that is changes made by these commands are permanently stored in the databases. Let us discuss these commands in more detail:

CREATE TABLE Command

Syntax:

```
CREATE TABLE <table name>(
Column_name1 data type (column width) [constraints],
Column_name2 data type (column width) [constraints],
Column_name3 data type (column width) [constraints],
……..
);
```

Where table name assigns the name of the table, column name defines the name of the field, data type specifies the data type for the field and column width allocates specified size to the field.

Guidelines for creation of table:

- Table name should start with an alphabet.
- In table name, blank spaces and single quotes are not allowed.
- Reserve words of that DBMS cannot be used as table name.
- Proper data types and size should be specified.
- Unique column name should be specified.

Column Constraints: NOT NULL, UNIQUE, PRIMARY KEY, CHECK, DEFAULT, REFERENCES,

On delete Cascade: Using this option whenever a parent row is deleted in a referenced table then all the corresponding child rows are deleted from the referencing table. This constraint is a form of referential integrity constraint.

Example 1:

```
CREATE TABLE product
    (
pno number (4) PRIMARY KEY,
pname char (20) NOT NULL,
qoh number (5) DEFAULT (100),
```

```
    class char (1) NOT NULL,
    rate number (8,2) NOT NULL,
    CHECK ((class='A' AND rate<1000) OR
    (class='B' AND rate>1000 AND rate<4500) OR
    (class='C' AND rate>4500))
);
```

The command above creates a table. Primary key constraint ensures that product number (pno) is not null and unique (both are the properties of primary key). Please note the use of data type char (20). In many implementations of SQL on commercial DBMS like SQL server and oracle, a data type called varchar and varchar2 is used respectively. Varchar basically is variable length character type subject to a maximum specified in the declarations. We will use them at most of the places later.

Please note the use of check constraints in the table created above. It correlates two different attribute values.

Example 2:

```
CREATE TABLE prodtrans
(
pno number (4)
ptype char (1) CHECK (ptype in ('I','R','S')),
qty number (5)
FOREIGN KEY pno REFERENCES product (pno)
ON DELETE CASCADE);
```

In the table so created please note the referential constraint on the foreign key **pno** in **prodtrans** table to **product** table. Any product record if deleted from the product table will trigger deletion of all the related records (ON DELETE CASCADE) in the **prodtrans** table.

ALTER TABLE Command: This command is used for modification of existing structure of the table in the following situation:

- When a new column is to be added to the table structure.
- When the existing column definition has to be changed, i.e., changing the width of the data type or the data type itself.
- When integrity constraints have to be included or dropped.
- When a constraint has to be enabled or disabled.

Syntax

ALTER TABLE <table name> ADD (<column name> <data type>…);
ALTER TABLE <table name> MODIFY (<column name> <data type>…);
ALTER TABLE <table name> ADD CONSTRAINT <constraint name> < constraint type>(field name);
ALTER TABLE <table name> DROP<constraint name>;
ALTER TABLE <table name> ENABLE/DISABLE <constraint name>;

You need to put many constraints such that the database integrity is not compromised.

Example 3:

```
ALTER TABLE emp MODIFY (empno NUMBER (7));
```

DROP TABLE Command:

When an existing object is not required for further use, it is always better to eliminate it from the database. To delete the existing object from the database the following command is used.
Syntax:

DROP TABLE<table name>;

Example 4:

DROP TABLE emp;

1.4 DATA MANIPULATION LANGUAGE

Data manipulation language (DML) defines a set of commands that are used to query and modify data within existing schema objects. In this case commit is not implicit that is changes are not permanent till explicitly committed. DML statements consist of SELECT, INSERT, UPDATE and DELETE statements.

SELECT Statement

This statement is used for retrieving information from the databases. It can be coupled with many clauses. Let us discuss these clauses in more detail:

1. **Using Arithmetic operator**

Example 5:

```
SELECT ENAME, SAL, SAL+300
 FROM EMP;
```

2. **Operator Precedence**

The basic operators used in SQL are * / + -
Operators of the same priority are evaluated From Left to right
Parentheses are used to force prioritized evaluation.

Example 6:

```
SELECT ENAME, SAL, 12*SAL+100
FROM EMP;

SELECT ENAME, SAL, 12*(SAL+100)
FROM EMP;
```

3. **Using Column aliases**

Example 7:

To print column names as NAME and ANNUAL SALARY

```
SELECT ENAME "NAME", SAL*12 "ANNUAL SALARY"
FROM EMP;
```

4. **Concatenation operator**

Example 8:

Printing name and job as one string as column name employees:

```
SELECT ENAME||JOB "EMPLOYEES"
FROM EMP;
```

5. **Using Literal Character String**

Example 9:

To print <name> IS A <job> as one string with column name employee

```
SELECT ENAME || ' IS A ' || JOB AS "EMPLOYEE"
FROM EMP;
```

6. **To eliminate duplicate rows (distinct operator)**

Example 10:

```
SELECT DISTINCT DEPTNO
FROM EMP;
```

7. **Special comparison operator** used in where Clause

   a. **between. …and…**It gives range between two values (inclusive)

Example 11:

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL BETWEEN 1000 AND 1500;
```

   b. **In (list):** match any of a list of values

Example 12:

```
SELECT EMPNO, ENAME, SAL, MGR
FROM EMP
WHERE MGR IN (7902, 7566, 7788);
```

7902, 7566, and 7788 are Employee numbers

   c. **Like:** match a character pattern

- Like operator is used only with char and Varchar2 to match a pattern
- % Denotes zero or many characters
- _ Denotes one character
- Combination of % and_can also be used

Example 13:

(I) List the names of employees whose name starts with ‘s’

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE ‘S%’;
```

(II) List the ename ending with ‘s’

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE ‘%S’;
```

(III) List ename having I as a second character

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE ‘_I%’;
```

d. **Is null operator**

Example 14:

to find employee whose manage-id is not specified

```
SELECT ENAME, MGR FROM EMP
WHERE MGR IS NULL;
```

8 **Logical Operators**

Rules of Precedence:

Order evaluated	Operator
1	All comparison operators

2	NOT
3	AND
4	OR

Example 15: To print those records of salesman or president who are having salary above 15000/-

```
Select ename, job, sal from emp
Where job = 'SALESMAN' or job = 'PRESIDENT'
And sal>15000;
```

The query formulation as above is incorrect for the problem statement. The correct formulation would be:

```
SELECT ENAME, JOB, SAL FROM EMP
WHERE (JOB = 'SALESMAN' OR JOB = 'PRESIDENT')
AND SAL>15000;
```

9. **Order by clause**

- It is used in the last portion of select statement
- By using this rows can be sorted
- By default it takes ascending order
- DESC: is used for sorting in descending order
- Sorting by column which is not in select list is possible
- Sorting by column Alias

Example 16:

```
SELECT EMPNO, ENAME, SAL*12 "ANNUAL"
FROM EMP
ORDER BY ANNUAL;
```

Example 17: Sorting by multiple columns; ascending order on department number and descending order of salary in each department.

```
SELECT ENAME, DEPTNO, SAL
FROM EMP
ORDER BY DEPTNO, SAL DESC;
```

10. **Aggregate functions**

- Some of these functions are count, min, max, avg.
- These functions help in getting consolidated information from a group of tuples.

Example 18: Find the total number of employees.

```
SELECT COUNT(*)
FROM EMP;
```

Example 19: Find the minimum, maximum and average salaries of employees of department D1.

```
SELECT MIN(SAL), MAX(SAL), AVG(SAL)
FROM EMP
WHERE DEPTNO = 'D1' ;
```

11. **Group By clauses**

- It is used to group database tuples on the basis of certain common attribute value such as employees of a department.
- WHERE clause still can be used, if needed.

Example 20: Find department number and Number of Employees working in that department.

```
SELECT DEPTNO, COUNT(EMPNO)
FROM EMP
GROUP BY DEPTNO;
```

Please note that while using group by and aggregate functions the only attributes that can be put in select clause are the aggregated functions and the attributes that have been used for grouping the information. For example, in the example 20, we cannot put ENAME attribute in the SELECT clause as it will not have a distinct value for the group. Please note the group here is created on DEPTNO.

12. Having clause

- This clause is used for creating conditions on grouped information.

Example 21: Find department number and maximum salary of those departments where maximum salary is more than Rs 20000/-.

```
SELECT DEPTNO, MAX(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MAX(SAL) > 20000;
```

INSERT INTO command:

- Values can be inserted for all columns or for the selected columns
- Values can be given through sub query explained in section 1.8
- In place of values parameter substitution can also be used with insert.
- If data is not available for all the columns, then the column list must be included following the table name.

Example 22: Insert the employee numbers, an increment amount of Rs.500/- and the increment date-today (which is being entered through function SYSDATE) of all the managers into a table INCR (increment due table) from the employee file.

```
INSERT INTO INCR
SELECT EMPNO, 500, SYSDATE FROM EMP
WHERE JOB = 'MANAGER';
```

Example 23: Insert values in a table using parameter substitution (& operator is used for it 1, 2 are the field numbers).

```
INSERT INTO EMP
VALUES (&1,'&2','&3', &4, &5, &6,NULL, &7);
```

Please note these values needs to be supplied at run time.

UPDATE Command:

Syntax is

```
UPDATE <table name>
SET <column name> = <value>
WHERE <condition>;
```

Sub query in the UPDATE command:

Example 24: Double the commission for employees, who have got at least 2 increments.

```
UPDATE EMP
SET COMM = COMM * 2
WHERE 2 <= (        SELECT COUNT (*) FROM INCR
WHERE INCR.EMPNO = EMP.EMPNO
GROUP BY EMPNO);
```

Please note the use of subquery that counts the number of increments given to each employee stored in the INCR table. Please note the comparison, instead of ……>=2, we have written reverse of it as 2 <= …..

DELETE Command

In the following example, the deletion will be performed in EMP table.No deletion will be performed in the INCR table.

Example 25: Delete the records of employees who have got no increment.

```
DELETE FROM EMP
WHERE EMPNO NOT IN (SELECT EMPNO FROM INCR);
```

☞ Check Your Progress 1

1) What are the advantages of SQL? Can you think of some disadvantages of SQL?

……

……

……

……..

2) Create the Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:
   (a) Type must be one of Single, Double, or Family.
   (b) Price must be between Rs.100/- and Rs.1000/-.
   (c) roomNo must be between 1 and 100.
   (d) booking dateFrom and dateTo must be greater than today's date.
   (e) The same room cannot be double booked.
   (f) The same guest cannot have overlapping bookings.

……

……

……

……

……

……..

3) Define the function of each of the clauses in the SELECT statement. What are the restrictions imposed on these clauses?

……..

……..

……

……

……..

4) Consider the supplier relations.

S

SNO (Supplier Number)	SNAME (Supplier Name)	STATUS	CITY
S1	Prentice Hall	30	Calcutta
S2	McGraw Hill	30	Mumbai
S3	Wiley	20	Chennai
S4	Pearson	40	Delhi
S5	Galgotia	10	Delhi

SP

SNO	PNO (Part Number)	Quantity
S1	P1	300
S1	P2	200
S2	P1	100
S2	P2	400
S3	P2	200
S4	P2	200

a) Get supplier numbers for suppliers with status > 20 and city is Delhi
b) Get Supplier Numbers and status for suppliers in Delhi in descending order of status.
c) Get all pairs of supplier numbers such that the two suppliers are located in the same city. (Hint: It is retrieval involving join of a table with itself.)
d) Get unique supplier names for suppliers who supply part P2 without using IN operator.
e) Give the same query above by using the operator IN.
f) Get part numbers supplied by more than one supplier. (Hint : It is retrieval with sub-query block, with inter block reference and same table involved in both blocks)
g) Get supplier numbers for suppliers who are located in the same city as supplier S1. (Hint: Retrieval with sub query and unqualified comparison operator).
h) Get supplier names for suppliers who supply part P1. (Hint : Retrieval using EXISTS)
i) Get part numbers for parts whose quantity is greater than 200 or are currently supplied by S2. (Hint: It is a retrieval using union).
j) Suppose for the supplier S5 the value for status is NULL instead of 10. Get supplier numbers for suppliers greater than 25. (Hint: Retrieval using NULL).
k) Get unique supplier numbers supplying parts. (Hint: This query is using the built-in function count).
l) For each part supplied, get the part number and the total quantity supplied for that part. (Hint: The query using GROUP BY).
m) Get part numbers for all parts supplied by more than one supplier. (Hint: It is GROUP BY with HAVING).
n) For all those parts, for which the total quantity supplied is greater than 300, get the part number and the maximum quantity of the part supplied in a single supply. Order the result by descending part number within those maximum quantity values.
o) Double the status of all suppliers in Delhi. (Hint: UPDATE Operation).
p) Let us consider the table TEMP has one column, called PNO. Enter into TEMP part numbers for all parts supplied by S2.
q) Add part p7.
r) Delete all the suppliers in Mumbai and also the suppliers concerned.

……

……

1.5 DATA CONTROL

The data control basically refers to commands that allow system and data privileges to be passed to various users. These commands are normally available to database administrator. Let us look into some data control language commands:

Create a new user:

CREATE USER < user name > IDENTIFIED BY < Password>

Example 26:

CREATE USER MCA12 IDENTIFIED BY W123

Grant: It is used to provide database access permission to users. It is of two types (1) system level permission (2) Object level permission.

Example 27:

GRANT CREATE SESSION TO MCA12;
(This command provides system level permission on creating a session – not portable)

GRANT SELECT ON EMP TO MCA12;
(Object level permission on table EMP)

GRANT SELECT, INSERT, UPDATE, DELETE ON EMP TO MCA12;

GRANT SELECT, UPDATE ON EMP TO MCA12, MCA13;
(Two users)
GRANT ALL ON EMP TO PUBLIC;
(All permission to all users, do not use it. It is very dangerous for database)

Revoke: It is used to cancel the permission granted.

Example 28:

REVOKE ALL ON EMP FROM MCA12;
(All permissions will be cancelled)
You can also revoke only some of the permissions.

Drop: A user-id can be deleted by using drop command.

Example 29:

DROP USER MCA12;

Accessing information about permissions to all users

(1) Object level permissions: With the help of data dictionary you can view the permissions to user. Let us take the table name from oracle. In oracle the name of the table containing these permissions is user_tab_privs.
DESCRIBE USER_TAB_PRIVS ;
SELECT * FROM USER_TAB_PRIVS;

(2) System level permissions: With the help of data dictionary you can see them. Let us take the table name as user_sys_privs (used in oracle).
DESCRIBE USER_SYS_PRIVS ;

SELECT * FROM USER_SYS_PRIVS ;

All these commands are very specific to a data base system and may be different on different DBMS.

1.6 DATABASE OBJECTS: VIEWS, SEQUENCES, INDEXES AND SYNONYMS

Some of the important concepts in a database system are the views and indexes. Views are a mechanism that can support data independence and security. Indexes help in better query responses. Let us discuss about them along with two more concepts sequences and synonyms in more details.

1.6.1 Views

A view is like a window through which data from tables can be viewed or changed. The table on which a view is based is called Base table. The view is stored as a SELECT statement in the data dictionary. When the user wants to retrieve data, using view. Then the following steps are followed.

1) View definition is retrieved from data dictionary table. For example, the view definitions in oracle are to be retrieved from table name USER-VIEWS.
2) Checks access privileges for the view base table.
3) Converts the view query into an equivalent operation on the underlying base table

Advantages:

- It restricts data access.
- It makes complex queries look easy.
- It allows data independence.
- It presents different views of the same data.

Type of views:

Simple views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain Functions	No	Yes
Contain groups of data	No	Yes
Data Manipulation	IS allowed	Not always

Let us look into some of the examples. To see all the details about existing views in Oracle:

SELECT* FROM USER_VIEWS;

Creating a view:

- A query can be embedded within the CREATE VIEW STATEMENT
- A query can contain complex select statements including join, groups and sub-queries
- A query that defines the view **cannot contain** an order by clause.
- DML operation (delete/modify/add) cannot be applied if the view contains any of the following:

Delete (You can't delete if view contains following)	*Modify (you cannot modify if view contains following)*	*Insert (you cannot insert if view contains following)*
• Group functions • A group by clause • A distinct keyword	• Group functions • A group by clause • A distinct keyword • Columns defined by Expressions	• Group functions • A group by clause • A distinct keyword • Columns defined by Expressions • There are Not Null Columns in the base tables that are not selected by view.

Example 30: Create a view named employee salary having minimum, maximum and average salary for each department.

```
CREATE VIEW EMPSAL (NAME, MINSAL, MAXSAL, AVGSAL) AS
        SELECT D.DNAME, MIN(E.SAL),MAX(E.SAL),AVG(E.SAL)
        FROM EMP E, DEPT D
        WHERE E.DEPTNO=D.DEPTNO
         GROUP BY D.DNAME;
```

To see the result of the command above you can give the following command:

```
SELECT * FROM EMPSAL;
```

You may get some sample output like:

NAME	MINSAL	MAXSA	AVGSAL
ACCOUNTING	1300	5000	2916.6667
RESEARCH	800	3000	2175
SALES	950	2850	1566.6667

To see the structure of the view so created, you can give the following command:

```
DESCRIBE EMPSAL;
```

Name	Null?	Type
NAME		VARCHAR2 (14)
MINSAL		NUMBER
MAXSAL		NUMBER
AVGSAL		NUMBER

Creating views with check option: This option restricts those updates of data values that cause records to go off the view. The following example explains this in more detail:

Example 31: To create a view for employees of Department = 30, such that user cannot change the department number from the view:

```
CREATE OR REPLACE VIEW EMPD30 AS
        SELECT EMPNO EMPL_NUM, ENAME NAME, SAL SALARY
        FROM EMP
        WHERE DEPTNO=30
        WITH CHECK OPTION;
```

Now the user cannot change the department number of any record of view EMPD30. If this change is allowed then the record in which the change has been made will go off the view as the view is only for department-30. This restricted because of use of WITH CHECK OPTION clause

Creating views with Read only option: In the view definition this option is used to ensure that no DML operations can be performed on the view.

Creating views with Replace option: This option is used to change the definition of the view without dropping and recreating it or regranting object privileges previously granted on it.

1.6.2 Sequences

Sequences:

- automatically generate unique numbers
- are sharable
- are typically used to create a primary key value
- replace application code
- speed up the efficiency of accessing sequence Values when cached in memory.

Example 32: Create a sequence named SEQSS that starts at 105, has a step of 1 and can take maximum value as 2000.

```
CREATE SEQUENCE SEQSS
START WITH 105
INCREMENT BY 1
 MAX VALUE 2000;
```

How the sequence so created is used? The following sequence of commands try to demonstrate the use of the sequence SEQSS.

Suppose a table person exists as:

```
SELECT * FROM PERSON;
Output:        CODE    NAME          ADDRESS
                ------  ------------- -------------
                 104     RAMESH    MUMBAI
```

Now, if we give the command:

```
INSERT INTO PERSON
        VALUES (SEQSS.NEXTVAL, &NAME, &ADDRESS)
```

On execution of statement above do the following input:

```
Enter value for name: 'Rakhi'
Enter value for address: 'New Delhi'
```

Now, give the following command to see the output:

```
SELECT * FROM PERSON;
  CODE NAME          ADDRESS
--------- -------------- ----------------
    104 RAMESH         MUMBAI
    105 Rakhi          NEW DELHI
```

The descriptions of sequences such as minimum value, maximum value, step or increment are stored in the data dictionary. For example, in oracle it is stored in the table user_sequences. You can see the description of sequences by giving the SELECT command.

Gaps in sequence values can occur when:

- A rollback occurs that is when a statement changes are not accepted.
- The system crashes
- A sequence is used in another table.

Modify a sequence:

```
ALTER SEQUENCE SEQSS
INCREMENT 2
MAXVALUE 3000;
```

Removing a sequence:

```
DROP SEQUENCE SEQSS;
```

1.6.3 Indexes and Synonyms

Some of the basic properties of indexes are:

- An Index is a schema Object
- Indexes can be created explicitly or automatically
- Indexes are used to speed up the retrieval of rows
- Indexes are logically and physically independent of the table. It means they can be created or dropped at any time and have no effect on the base tables or other indexes.
- However, when a table is dropped corresponding indexes are also dropped.

Creation of Indexes

Automatically: When a primary key or Unique constraint is defined in a table definition then a unique index is created automatically.

Manually: User can create non-unique indexes on columns to speed up access time to rows.

Example 33: The following commands create index on employee name and employee name + department number respectively.

```
CREATE INDEX EMP_ENAME_IDX ON EMP (ENAME);
CREATE INDEX EMP_MULTI_IDX ON EMP (ENAME, DEPTNO);
```

Finding details about created indexes: The data dictionary contains the name of index, table name and column names. For example, in Oracle a user-indexes and user-ind-columns view contains the details about user created indexes.

Remove an index from the data dictionary:

```
DROP INDEX EMP_ENAME_IDX;
```

Indexes cannot be modified.

Synonyms

It permits short names or alternative names for objects.

Example 34:

```
CREATE SYNONYM D30
FOR EMPD30;
```

Now if we give command:

```
SELECT * FROM D30;
```

The output will be:

```
NAME                MINSAL    MAXSAL     AVGSAL
------------------- ----------  ----------  ----------
ACCOUNTING       1300         5000       2916.6667
RESEARCH          800          3000       2175
SALES             950          2850       1566.6667
```

Removing a Synonym:

```
DROP SYNONYM D30;
```

1.7 TABLE HANDLING

In RDBMS more than one table can be handled at a time by using join operation. Join operation is a relational operation that causes two tables with a common domain to be combined into a single table or view. SQL specifies a join implicitly by referring the matching of common columns over which tables are joined in a WHERE clause. Two tables may be joined when each contains a column that shares a common domain with the other. The result of join operation is a single table. Selected columns from all the tables are included. Each row returned contains data from rows in the different input tables where values for the common columns match. An important rule of table handling is that there should be one condition within the WHERE clause for each pair of tables being joined. Thus if two tables are to be combined, one condition would be necessary, but if three tables (X, Y, Z) are to be combined then two conditions would be necessary because there are two pairs of tables (X-Y and Y-Z) OR (X-Z and Y-Z), and so forth. There are several possible types of joins in relational database queries. Four types of join operations are described below:

(1) **Equi Join**: A join in which the joining condition is based on equality between values in the common columns. Common columns appear (redundantly) in the result table. Consider the following relations:

- customer (custid, custname, ……..) and
- order (custid, ordered,……..)

Example 35: What are the names of all customers who have placed orders?

The required information is available in two tables, customer and order. The SQL solution requires joining the two table using equi join.

```
SELECT CUSTOMER.CUTOID, ORDER.CUSTOID,
                                    CUSTONAME, ORDERID
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CUSTOID=ORDER.CUSTOID;
```

The output may be:

Customer.custoid	order.custoid	custoname	orderid
10	10	Pooja Enterprises	1001
12	12	Estern Enterprises	1002
3	3	Impressions	1003

(2) **Natural Join**: It is the same like Equi join except one of the duplicate columns is eliminated in the result table. The natural join is the most commonly used form of join operation.

Example 36:

```
SELECT CUSTOMER.CUTOID, CUSTONAME, ORDERID
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CUSTOID=ORDER.CUSTOID;
```

Output:

custoid	custoname	orderid
10	Pooja Enterprises	1001
12	Estern Enterprises	1002
3	Impressions	1003

(3) **Outer Join**: The use of Outer Join is that it even joins those tuples that do not have matching values in common columns are also included in the result table. Outer join places null values in columns where there is not a match between

tables. A condition involving an outer join is that it cannot use the IN operator or cannot be linked to another condition by the OR operator.

Example 37: The following is an example of left outer join (which only considers the non-matching tuples of table on the left side of the join expression).

```
SELECT CUSTOMER.CUTOID, CUSTONAME, ORDERID
FROM CUSTOMER LEFT OUTER JOIN ORDER
WHERE CUSTOMER.CUSTOID =   ORDER.CUSTOID;
```

Output: The following result assumes a CUSTID in CUSTOMER table who have not issued any order so far.

CUSTOID	CUSTONAME	ORDERID
10	Pooja Enterprises	1001
12	Estern Enterprises	1002
3	Impressions	1003
15	South Enterprises	NULL

The other types of outer join are the Right outer join or complete outer join.

(4) **Self-Join**: It is a join operation where a table is joined with itself. Consider the following sample partial data of EMP table:

EMPNO	ENAME	MGRID	…..
1	Nirmal	4	
2	Kailash	4	
3	Veena	1	
4	Boss	NULL	
…..	…..	…	

Example 38: Find the name of each employee's manager name.

```
SELECT WORKER.ENAME || 'WORK FOR' || MANAGER.ENAME
FROM EMP WORKER, EMP MANAGER
WHERE WORKER.MGR=MANAGER.EMPNO;
```

Output:

Nirmal works for Boss
Kailash works for Boss
Veena works for Nirmal

☞ Check Your Progress 2

1) Discuss how the Access Control mechanism of SQL works.

……..

……..

……..

……..

2) Consider Hotel schema consisting of three tables Hotel, Booking and Guest,

```
CREATE TABLE Hotel
        hotelNo         HotelNumber         NOT NULL,
        hotelName       VARCHAR(20)         NOT NULL,
        city            VARCHAR(50)         NOT NULL,

        PRIMARY KEY (hotelNo));
```

```
CREATE TABLE Booking(
hotelNo         HotelNumbers        NOT NULL,
guestNo         GuestNumbers        NOT NULL,
dateFrom        BookingDate         NOT NULL,
dateTo          BookingDate         NULL,
roomNo          RoomNumber          NOT NULL,

PRIMARY KEY (hotelNo, guestNo, dateFrom),
FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (guestNo) REFERENCES Guest
        ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
        ON DELETE NO ACTION ON UPDATE CASCADE);
CREATE TABLE Guest(
guestNo         GuestNumber         NOT NULL,
guestName       VARCHAR(20)         NOT NULL,
guestAddress    VARCHAR(50)         NOT NULL
PRIMARY KEY (guestno));

CREATE TABLE Room(
roomNo          RoomNumber          NOT NULL,
hotelNo         HotelNumbers        NOT NULL,
type            RoomType            NOT NULL DEFAULT 'S'
price           RoomPrice           NOT NULL,
PRIMARY KEY (roomNo, hotelNo),
FOREIGN KEY (hotelNo) REFERENCES Hotel
ON DELETE CASCADE ON UPDATE CASCADE);
```

Create a view containing the hotel name and the names of the guests staying at the hotel.

……

……

……

……

……

……

……

……..

3) Give the users Manager and Director full access to views HotelData and BookingOutToday, with the privilege to pass the access on to other users.

……

……

……

……

……

1.8 NESTED QUERIES

By now we have discussed the basic commands including data definition and data manipulations. Now let us look into some more complex queries in this section.

Sub-queries

Some of the basic issues of sub-queries are:

- A sub-query is a SELECT statement that is embedded in a clause of another SELECT statement. They are often referred to as a NESTED SELECT or SUB SELECT or INNER SELECT.

- The sub-query (inner query) executes first before the main query. The result of the sub-query is used by the main query (outer query).

- Sub-query can be placed in WHERE or HAVING or FROM clauses.

- Format of using sub-queries:
  SELECT<select_list>
  FROM<table>
  WHERE expr OPERATOR
  (SELECT <select_list>
  FROM <TABLE>WHERE);
  Operator includes a comparison operator (single or multiple row operators)
  Single row operators: >, =, >=, <, <=, <>
  Multiple row operators: IN, ANY, ALL

- Order by clause cannot be used in sub-query, if specified it must be the last clause in the main select statement.

- Types of sub-queries:

  - Single row sub-query: It returns only one row from the inner select statement.
  - Multiple row sub-queries: it returns more than one row from the inner select statement
  - Multiple column sub-queries: it returns more than one column from the inner select statement.

  Single row operators are used with single row sub queries and multiple row operators are used with multiple row sub queries.

- The Outer and Inner queries can get data from different tables.

- Group Functions can be used in sub queries.

Consider the following partial relation EMP. Let us create some sub-queries for them

EMPNO	ENAME	JOB	SAL	DEPTNO
7566	Nirmal	MANAGER	2975	10
7788	Kailash	ANALYST	3000	10
7839	Karuna	PRESIDENT	5000	20
7902	Ashwin	ANALYST	3000	20
7905	Ashwini	MANAGER	4000	20

Example 39: Get the details of the person having the minimum salary.

```
SELECT ENAME, JOB, SAL
FROM EMP
 WHERE SAL =   (      SELECT MIN (SAL)
                      FROM EMP);
```

Output:

ENAME	JOB	SAL
Nirmal	MANAGER	2975

Example 40: Display the employees whose job title is the same as that of employee 7566 and salary is more than the salary of employee 7788.

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB  =   (      SELECT JOB
                      FROM EMP
                      WHERE EMPPNO = 7566)
          AND SAL >  (  SELECT SAL
                      FROM EMP
                      WHERE EMPPNO=7788);
```

Output: Job title for the employee 7566 happens to be 'MANAGER')

ENAME	JOB
Ashwini	MANAGER

Having Clause with sub queries: First we recollect the GROUP BYclause. The following query finds the minimum salary in each department.

```
SELECT DEPTNO, MIN(SAL)
FROM.EMP
GROUP BY DEPTNO;
```

Output:

DEPTNO	SAL
10	2975
20	3000

Example 41: To find the minimum salary in those departments whose minimum salary is greater than minimum salary of department number 10.

```
SELECT DEPTNO, MIN(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MIN(SAL) >  (        SELECT MIN (SAL)
                            FROM EMP
                            WHERE DEPTNO = 10);
```

Output:

DEPTNO	SAL
20	3000

Example 42: Find the name, department number and salary of employees drawing minimum salary in that department.

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE SAL IN (SELECT MIN (SAL)
                  FROM EMP
                  GROUP BY DEPTNO);
```

Output:

ENAME	SAL	DEPTNO
Nirmal	2975	10
Ashwin	3000	20

Find the salary of employees employed as an ANALYST
SELECT SAL FROM EMPWHERE JOB= ' ANALYST '

Output:

SAL
3000
3000

Example 43: Find the salary of employees who are not 'ANALYST' but get a salary less than or equal to any person employed as 'ANALYST'.

```
SELECT EMPNO, ENAME, JOB, SAL
FROMEMP
WHERE SAL <= ANY ( SELECT SAL
                   FROM EMP WHERE JOB = 'ANALYST' )
    AND JOB<>'ANALYST' ;
```

Output:

EMPNO	ENAME	JOB	SAL
7566	Nirmal	MANAGER	2975

Find the average salary in each department

SELECT DEPTNO, AVG(SAL) FROM EMP GROUP BY DEPTNO;
Result:

DEPTNO	SAL
10	2987.5
20	4000

Example 44: Find out the employee who draws a salary more than the average salary of all the departments.

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL> ALL (SELECT AVG (SAL)
                FROM EMP
                GROUP BY DEPTNO);
```

Output:

EMPNO	ENAME	JOB	SAL
7839	Karuna	PRESIDENT	5000

Example 45: Find the employee name, salary, department number and average salary of his/her department, for those employees whose salary is more than the average salary of that department.

```
SELECT A.ENAME, A.SAL, A.DEPTNO, B.AVGSAL
FROM EMP A, (   SELECT DEPTNO, AVG (SAL)  AVGSAL
                FROM EMP
                GROUP BY DEPTNO) B
WHERE A.DEPTNO=B.DEPTNO AND A.SAL> B. AVGSAL;
```

Output:

ENAME	SAL	DEPTNO	AVGSAL
Kailash	3000	10	2987.5
Karuna	5000	20	4000

Multiple column Queries:
Syntax:

```
SELECT COLUMN1, COL2,……
FROM TABLE
WHERE (COLUMN1, COL2, …) IN
                (SELECT COLUMN1, COL2,….
                 FROM TABLE
                 WHERE <CONDITION>);
```

Example 46: Find the department number, name, job title and salary of those people who have the same job title and salary as those are in department 10.

```
SELECT DEPTNO,ENAME, JOB, SAL
FROM EMP
WHERE (JOB, SAL) IN  (      SELECT JOB, SAL
                            FROM EMP
                            WHERE DEPTNO=10);
```

Output:

DEPTNO	ENAME	JOB	SAL
10	Nirmal	MANAGER	2975
10	Kailash	ANALYST	3000
20	Ashwin	ANALYST	3000

☞ Check Your Progress 3

1) What is the difference between a sub-query and a join? Under what circumstances would you not be able to use a sub-query?

……

……

……

2) Use the Hotel schema defined in question number 2 (Check Your Progress 2) and answer the following queries:

- List the names and addresses of all guests in Delhi, alphabetically ordered by name.
- List the price and type of all rooms at the GRAND Hotel.
- List the rooms that are currently unoccupied at the Grosvenor Hotel.
- List the number of rooms in each hotel.
- What is the most commonly booked room type for hotel in Delhi?
- Update the price of all rooms by 5%.

……

……

……

3) Consider the following Relational database.

Employees (eno, ename, address, basic_salary)
Projects (Pno, Pname, enos-of-staff-alotted)
Workin (pno, eno, pjob)

Two queries regarding the data in the above database have been formulated in SQL. Describe the queries in English sentences.

(i)
```
SELECT ename
FROM employees
WHERE eno IN ( SELECT eno
                FROM workin
                GROUP BY eno
 HAVING COUNT (*) =  (SELECT COUNT (*) FROM projects));
```

(ii)
```
SELECT Pname
FROM projects
WHERE Pno IN ( SELECT Pno
FROM projects
MINUS
GROUP BY eno
(SELECT DISTINCT Pno FROM workin));
```

……..

……..

…….

……..

……..

…….

1.9 SUMMARY

This unit has introduced the SQL language for relational database definition, manipulation and control. The SQL environment includes an instance of an SQL DBMS with accessible databases and associated users and programmers. Each schema definition that describes the database objects is stored in data dictionary/ system catalog. Information contained in system catalog is maintained by the DBMS itself, rather than by the users of DBMS.

The data definition language commands are used to define a database, including its creation and the creation of its tables, indexes and views. Referential integrity constraints are also maintained through DDL commands. The DML commands of SQL are used to load, update and query the database. DCL commands are used to establish user access to the database. SQL commands directly affect the base tables, which contain the raw data, or they may affect database view, which has been created. The basic syntax of an SQL SELECT statement contains the following keywords: SELECT, FROM, WHERE, ORDER BY, GROUP BY and HAVING.

SELECT determines which attributes will be displayed in the query result table. FROM determines which tables or views will be used in the query. WHERE sets the criteria of the query, including any joins of multiple tables, which are necessary. ORDER BY determines the order in which the result will be displayed. GROUP BY is used to categorize results. HAVING is used to impose condition with GROUP BY.

1.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Advantages

- A standard for database query languages
- (Relatively) Easy to learn
- Portability
- SQL standard exists
- Both interactive and embedded access
- Can be used by specialist and non-specialist.

Yes, SQL has disadvantages. However, they are primarily more technical with reference to Language features and relational model theories. We are just putting them here for reference purposes.

Disadvantages

- Impedance mismatch – mixing programming paradigms with embedded access
- Lack of orthogonality – many different ways to express some queries
- Language is becoming enormous (SQL-92 is 6 times larger than predecessor)
- Handling of nulls in aggregate functions is not portable
- Result tables are not strictly relational – can contain duplicate tuples, imposes an ordering on both columns and rows.

2. CREATE DOMAIN RoomType AS CHAR(1) ……..*[Constraint (a)]*
CHECK(VALUE IN (S, F, D));

CREATE DOMAIN HotelNumbers AS HotelNumber
CHECK(VALUE IN (SELECT hotelNo FROM Hotel));
[An additional constraint for hotel number for the application]

CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
CHECK(VALUE BETWEEN 1000 AND 10000);

CREATE DOMAIN RoomNumber AS VARCHAR(4)
CHECK(VALUE BETWEEN '1' AND '100');

[Constraint (c), one additional character is kept instead of 3 we have used 4characters but no space wastage as varchar]

```
CREATE TABLE Room(
        roomNo      RoomNumber      NOT NULL,
        hotelNo     HotelNumbers    NOT NULL,
        type        RoomType        NOT NULL DEFAULT S,
        price       RoomPrice       NOT NULL,
        PRIMARY KEY (roomNo, hotelNo),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
                ON DELETE CASCADE ON UPDATE CASCADE);
        CREATE DOMAIN GuestNumber AS CHAR(4);
```

```
CREATE TABLE Guest(
        guestNo         GuestNumber         NOT NULL,
        guestName       VARCHAR(20)         NOT NULL,
        guestAddress    VARCHAR(50)         NOT NULL);

CREATE DOMAIN GuestNumbers AS GuestNumber
        CHECK(VALUE IN (SELECT guestNo FROM Guest));
                [A sort of referential constraint expressed within domain]

CREATE DOMAIN BookingDate AS DATETIME
        CHECK(VALUE > CURRENT_DATE);        [constraint (d) ]

CREATE TABLE Booking(
        hotelNo         HotelNumbers        NOT NULL,
        guestNo         GuestNumbers        NOT NULL,
        dateFrom        BookingDate         NOT NULL,
        dateTo          BookingDate         NULL,
        roomNo          RoomNumber          NOT NULL,
        PRIMARY KEY (hotelNo, guestNo, dateFrom),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
                ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (guestNo) REFERENCES Guest
                ON DELETE NO ACTION ON UPDATE CASCADE,
        FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
                ON DELETE NO ACTION ON UPDATE CASCADE,
        CONSTRAINT RoomBooked
                CHECK (NOT EXISTS ( SELECT *
                                FROM Booking b
                                WHERE b.dateTo > Booking.dateFrom AND
                                b.dateFrom < Booking.dateTo AND
                                b.roomNo = Booking.roomNo AND
                                b.hotelNo = Booking.hotelNo)),
        CONSTRAINT GuestBooked
                CHECK (NOT EXISTS ( SELECT *
                                FROM Booking b
                                WHERE b.dateTo > Booking.dateFrom AND
                                b.dateFrom < Booking.dateTo AND
                                b.guestNo = Booking.guestNo)));
```

3.

FROM	Specifies the table or tables to be used.
WHERE	Filters the rows subject to some condition.
GROUP BY	Forms groups of rows with the same column value.
HAVING	Filters the groups subject to some condition.
SELECT	Specifies which columns are to appear in the output.
ORDER BY	Specifies the order of the output.

If the SELECT list includes an aggregate function and no GROUP BY clause is being used to group data together, then no item in the SELECT list can include any reference to a column unless that column is the argument to an aggregate function.

When GROUP BY is used, each item in the SELECT list must be single-valued per group. Further, the SELECT clause may only contain:

- Column names.
- Aggregate functions.
- Constants.
- An expression involving combinations of the above.

All column names in the SELECT list must appear in the GROUP BY clause unless the name is used only in an aggregate function.

3. Please note that some of the queries are sub-queries and queries requiring join. The meaning of these queries will be clearer as you proceed further with the Unit.

a)
```
SELECT SNO
FROM S
WHERE CITY = 'Delhi'
AND STATUS > 20;
```

Result:

SNO
S4

b)
```
SELECT SNO, STATUS
FROM S
WHERE CITY = 'Delhi'
ORDER BY STATUS DESC;
```

Result:

SNO	STATUS
S4	40
S5	10

c)
```
SELECT FIRST.SNO, SECOND.SNO
FROM S FIRST, S SECOND
WHERE FIRST.CITY = SECOND.CITY AND FIRST.SNO <
SECOND.SNO;
```

Please note that if you do not give the condition after AND you will get some unnecessary tuples such as: (S4, S4), (S5, S4) and (S5, S5).

Result:

SNO	SNO
S4	S5

d)
```
SELECT DISTINCT SNAME
FROM S, SP
WHERE S.SNO = SP.SNO
AND SP.PNO = 'P2';
```

Result:

SNAME
Prentice Hall

McGraw Hill
Wiley
Pearson

```
OR
SELECT SNAME
FROM S
WHERE SNO = ANY (   SELECT SNO
                    FROM SP
                    WHERE PNO = 'P2');
```

e)
```
SELECT SNAME
FROM S
WHERE SNO IN (      SELECT SNO
                    FROM SP
                    WHERE PNO = 'P2' );
```

f)
```
SELECT DISTINCT PNO
FROM SP SPX
WHERE PNO IN (      SELECT PNO
                    FROM SP
                WHERE SP.SNO = SPX.SNO AND SPX.SNO < SP.SNO );
```

This query can also be answered using count and group by. Please formulate that.

Result:

PNO
P1
P2

g)
```
SELECT SNO
FROM S
WHERE CITY = (      SELECT CITY
                    FROM S
                    WHERE SNO = 'S1' );
```

Result:

SNO
S1

h)
```
SELECT SNAME
FROM S
WHERE EXISTS (      SELECT *
                    FROM SP
                    WHERE SNO = S.SNO AND PNO = 'P1');
```

Result:

SNAME
Prentice Hall
McGraw Hill

i)
```
SELECT PNO
FROM SP
WHERE QUANTITY > 200 UNION (   SELECT PNO
                               FROM SP
                               WHERE SNO = S2 );
```

Result:

PNO
P1
P2

j)
```
SELECT SNO
FROM S
WHERE STATUS > 25 OR STATUS IS NULL;
```

Result:

SNO
S1
S2
S4
S5

k)
```
SELECT COUNT (DISTINCT SNO)
FROM SP;
```

Result: 4

l)
```
SELECT PNO, SUM(QUANTITY)
FROM SP
GROUP BY PNO;
```

Result:

PNO	SUM
P1	400
P2	1000

m)
```
SELECT PNO
FROM SP
GROUP BY PNO
HAVING COUNT(*) > 1 ;
```

The query is a same as that of part (f)

n)
```
SELECT PNO, MAX(QUANTITY)
FROM SP
WHERE QUANTITY > 200
GROUP BY PNO
HAVING SUM(QUANTITY) > 300
ORDER BY 2, PNO DESC;
```

o)
```
UPDATE  S
SET STATUS = 2 * STATUS
WHERE CITY = 'Delhi';
```

p)
```
INSERT INTO TEMP
SELECT PNO
FROM SP
WHERE SNO = 'S2';
```

q)
```
INSERT INTO SP( SNO,PNO,QUANTITY) < 'S5','P7',100> ;
```

Please note that part cannot be added without a supply in the present case.

Actually there should be another table for Parts

r)
```
DELETE S, SP
WHERE SNO = (       SELECT SNO
                    FROM S
                    WHERE CITY = 'Mumbai');
```

Check Your Progress 2

1) Each user has an authorization identifier (allocated by DBA).
Each object has an owner. Initially, only owner has access to an object but the owner can pass privileges to carry out certain actions on to other users via the GRANT statement and take away given privileges using REVOKE.

2)
```
CREATE VIEW HotelData(hotelName, guestName) AS
SELECT h.hotelName, g.guestName
        FROM Hotel  h, Guest g, Booking b
        WHERE h.hotelNo = b.hotelNo AND g.guestNo = b.guestNo AND
                b.dateFrom <= CURRENT_DATE AND
                b.dateTo >= CURRENT_DATE;
```

3)
```
GRANT ALL PRIVILEGES ON HotelData
        TO Manager, Director WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON BookingOutToday
        TO Manager, Director WITH GRANT OPTION;
```

Check Your Progress 3

1) With a sub-query, the columns specified in the SELECT list are restricted to one table. Thus, cannot use a sub-query if the SELECT list contains columns from more than one table. But with a join operation SELECT list contains columns from more than two tables.

2) Answers of the queries are:

- SELECT guestName, guestAddress FROM Guest
  WHERE address LIKE '%Delhi%'
  ORDER BY guestName;

- SELECT price, type FROM Room
  WHERE hotelNo =
  (SELECT hotelNo FROM Hotel
  WHERE hotelName = 'GRAND Hotel');

- SELECT * FROM Room r
  WHERE roomNo NOT IN
  (SELECT roomNo FROM Booking b, Hotel h
  WHERE (dateFrom <= CURRENT_DATE AND
  dateTo >= CURRENT_DATE) AND
  b.hotelNo = h.hotelNo AND hotelName = 'GRAND Hotel');
- SELECT hotelNo, COUNT(roomNo) AS count
  FROM Room
  GROUP BY hotelNo;

- SELECT MAX(X)

```
FROM ( SELECT type, COUNT(type) AS X
FROM Booking b, Hotel h, Room r
WHERE r.roomNo = b.roomNo AND b.hotelNo = h.hotelNo AND
        city = 'LONDON'
GROUP BY type);
```

- UPDATE Room SET price = price*1.05;

3) (i) – Give names of employees who are working on all projects.

(ii) - Give names of the projects which are currently not being worked upon.

1.11 FURTHER READINGS

Fundamentals of DatabaseSystems; Almasri and Navathe; Pearson Education Limited; Fourth Edition; 2004.
A Practical Approach to Design, Implementation, and Management; Thomas Connolly and Carolyn Begg; Database Systems, Pearson Education Limited; Third Edition; 2004.
The Complete Reference; Kevin Lonely and George Koch; Oracle 9i, Tata McGraw-Hill; Fourth Edition; 2003.
Jeffrey A. Hoffer, Marry B. Prescott and Fred R. McFadden; Modern Database Management; Pearson Education Limited; Sixth Edition; 2004.

UNIT 2 TRANSACTIONS AND CONCURRENCY MANAGEMENT

Structure **Page Nos.**

2.0 INTRODUCTION

One of the main advantages of storing data in an integrated repository or a database is to allow sharing of it among multiple users. Several users access the database or perform transactions at the same time. What if a user's transactions try to access a data item that is being used /modified by another transaction? This unit attempts to provide details on how concurrent transactions are executed under the control of DBMS. However, in order to explain the concurrent transactions, first we must describe the term transaction.

Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilizing CPU time efficiently, since disk accesses are frequent and are relatively slow in case of DBMS. Also, a user's program may carry out many operations on the data returned from DB, but DBMS is only concerned about what data is being read /written from/ into the database. This unit discusses the issues of concurrent transactions in more detail.

2.1 OBJECTIVES

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

- describe the term CONCURRENCY;
- define the term transaction and concurrent transactions;
- discuss about concurrency control mechanism;
- describe the principles of locking and serialisability, and
- describe concepts of deadlock & its prevention.

2.2 THE TRANSACTIONS

A transaction is defined as the unit of work in a database system. Database systems that deal with a large number of transactions are also termed as transaction processing systems.

What is a transaction? Transaction is a unit of data processing. For example, some of the transactions at a bank may be withdrawal or deposit of money; transfer of money from A's account to B's account etc. A transaction would involve manipulation of one

or more data values in a database. Thus, it may require reading and writing of database value. For example, the withdrawal transactions can be written in pseudo code as:

Example 1:

```
; Assume that we are doing this transaction for person
;  whose account number is X.

TRANSACTION WITHDRAWAL (withdrawal_amount)
Begin transaction
        IF X exist then
                READ  X.balance
                IF        X.balance > withdrawal_amount
                          THEN  SUBTRACT withdrawal_amount
                                  WRITE X.balance
                                   COMMIT
                ELSE
                DISPLAY "TRANSACTION CANNOT BE PROCESSED"
        ELSE  DISPLAY "ACCOUNT X DOES NOT EXIST"
End transaction;
```

Another similar example may be transfer of money from Account no x to account number y. This transaction may be written as:

```
Example 2:
; transfers transfer_amount from x's account to y's account
; assumes x&y both accounts exist

TRANSACTION (x, y, transfer_amount)
Begin transaction
        IF X AND Y exist then
                READ x.balance
                IF x.balance > transfer_amount THEN
                          x.balance = x.balance – transfer_amount
                          READ y.balance
                          y.balance = y.balance + transfer_amount
                          COMMIT
                ELSE DISPLAY ("BALANCE IN X NOT OK")
                        ROLLBACK
        ELSE DISPLAY ("ACCOUNT X OR Y DOES NOT EXIST")
End_transaction
```

Please note the use of two keywords here COMMIT and ROLLBACK. Commit makes sure that all the changes made by transactions are made permanent. ROLLBACK terminates the transactions and rejects any change made by the transaction. Transactions have certain desirable properties. Let us look into those properties of a transaction.

Properties of a Transaction

A transaction has four basic properties. These are:

- Atomicity
- Consistency
- Isolation or Independence
- Durability or Permanence

Atomicity: It defines a transaction to be a single unit of processing. In other words either a transaction will be done *completely* or *not at all*. In the transaction example 1 & 2 please note that transaction 2 is reading and writing more than one data items, the atomicity property requires either operations on both the data item be performed or not at all.

Consistency: This property ensures that a complete transaction execution takes a database from one consistent state to another consistent state. If a transaction fails even then the database should come back to a consistent state.

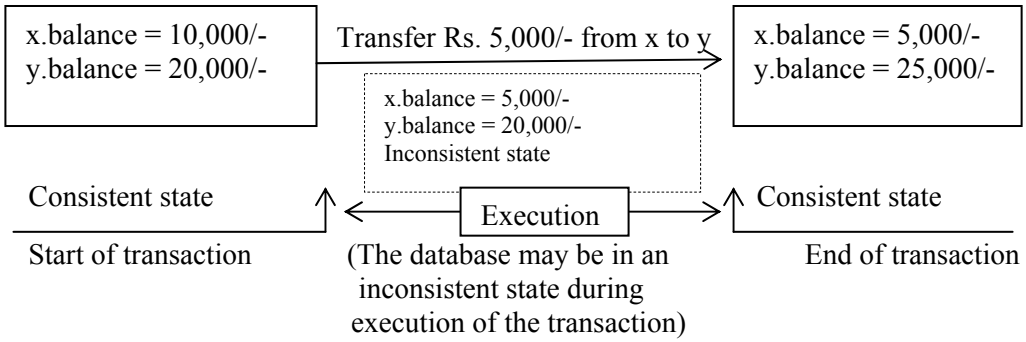


Figure 1: A Transaction execution

Isolation or Independence: The isolation property states that the updates of a transaction should not be visible till they are committed. Isolation guarantees that the progress of other transactions do not affect the outcome of this transaction. For example, if another transaction that is a withdrawal transaction which withdraws an amount of Rs. 5000 from X account is in progress, whether fails or commits, should not affect the outcome of this transaction. Only the state that has been read by the transaction last should determine the outcome of this transaction.

Durability: This property necessitates that once a transaction has committed, the changes made by it be never lost because of subsequent failure. Thus, a transaction is also a basic unit of recovery. The details of transaction-based recovery are discussed in the next unit.

A transaction has many states of execution. These states are displayed in *Figure 2*.

Error!

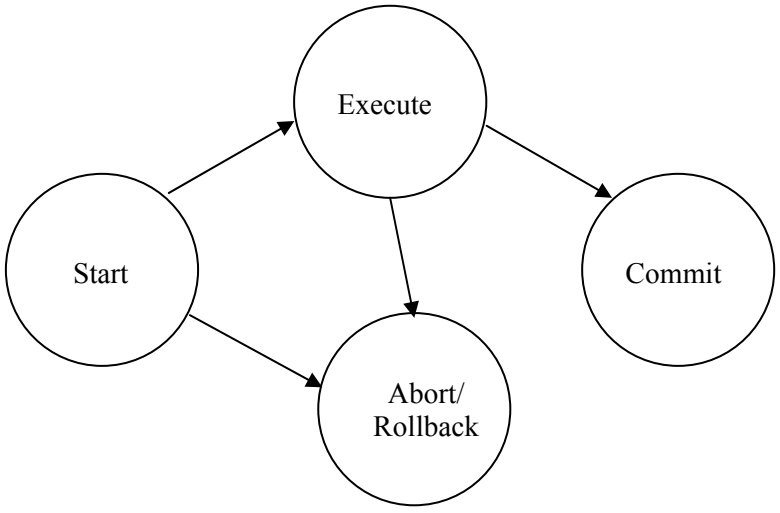


Figure 2: States of transaction execution

A transaction is started as a program. From the start state as the transaction is scheduled by the CPU it moves to the Execute state, however, in case of any system

error at that point it may also be moved into the Abort state. During the execution transaction changes the data values and database moves to an inconsistent state. On successful completion of transaction it moves to the Commit state where the durability feature of transaction ensures that the changes will not be lost. In case of any error the transaction goes to Rollback state where all the changes made by the transaction are undone. Thus, after commit or rollback database is back into consistent state. In case a transaction has been rolled back, it is started as a new transaction. All these states of the transaction are shown in *Figure 2*.

2.3 THE CONCURRENT TRANSACTIONS

Almost all the commercial DBMS support multi-user environment. Thus, allowing multiple transactions to proceed simultaneously. The DBMS must ensure that two or more transactions do not get into each other's way, i.e., transaction of one user doesn't effect the transaction of other or even the transactions issued by the same user should not get into the way of each other. Please note that concurrency related problem may occur in databases only if **two transactions are contending for the same data item and at least one of the concurrent transactions wishes to update a data value in the database**. In case, the concurrent transactions only read same data item and no updates are performed on these values, then it does NOT cause any concurrency related problem. Now, let us first discuss why you need a mechanism to control concurrency.

Consider a banking application dealing with checking and saving accounts. A Banking Transaction T1 for Mr. Sharma moves Rs.100 from his checking account balance X to his savings account balance Y, using the transaction T1:

Transaction T1:

```
A:  Read X
    Subtract 100
    Write X
B:  Read Y
    Add 100
    Write Y
```

Let us suppose an auditor wants to know the total assets of Mr. Sharma. He executes the following transaction:

Transaction T2:

```
Read X
Read Y
Display X+Y
```

Suppose both of these transactions are issued simultaneously, then the execution of these instructions can be mixed in many ways. This is also called the **Schedule**. Let us define this term in more detail.

A schedule S is defined as the sequential ordering of the operations of the 'n' interleaved transactions. A schedule maintains the order of operations within the individual transaction.

Conflicting Operations in Schedule: Two operations of different transactions conflict if they access the same data item AND one of them is a write operation.

For example, the two transactions TA and TB as given below, if executed in parallel, may produce a schedule:

TA	TB

READ X WRITE X

READ X WRITE X

SCHEDULE	TA	TB
READ X	READ X	
READ X		READ X
WRITE X		WRITE X
WRITE X	WRITE X	

One possible schedule for interleaved execution of TA and TB

Let us show you three simple ways of interleaved instruction execution of transactions T1 and T2. Please note that in the following tables the first column defines the sequence of instructions that are getting executed, that is the schedule of operations.

a) T2 is executed completely before T1 starts, then sum X+Y will show the correct assets:

Schedule	**Transaction T1**	**Transaction T2**	**Example Values**
Read X		Read X	X = 50000
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	**150000**
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

b) T1 is executed completely before T2 starts, then sum X+Y will still show the correct assets:

Schedule	**Transaction T1**	**Transaction T2**	**Example Values**
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100100
Display X+Y		Display X+Y	**150000**

c) Block A in transaction T1 is executed, followed by complete execution of T2, followed by the Block B of T1.

Schedule	**Transaction T1**	**Transaction T2**	**Example Values**
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	**149900**
Read Y	Read Y		Y= 100000

Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

In this execution an incorrect value is being displayed. This is because Rs.100 although removed from X, has not been put in Y, and is thus missing. Obviously, if T1 had been written differently, starting with block B and following up with block A, even then such an interleaving would have given a different but incorrect result.

Please note that for the given transaction there are many more ways of this interleaved instruction execution.

Thus, there may be a problem when the transactions T1 and T2 are allowed to execute in parallel. Let us define the problems of concurrent execution of transaction more precisely.

Let us assume the following transactions (assuming there will not be errors in data while execution of transactions)

Transaction T3 and T4: T3 reads the balance of account X and subtracts a withdrawal amount of Rs. 5000, whereas T4 reads the balance of account X and adds an amount of Rs. 3000

T3
READ X
SUB 5000
WRITE X

T4
READ X
ADD 3000
WRITE X

Problems of Concurrent Transactions

1. **Lost Updates:** Suppose the two transactions T3 and T4 run concurrently and they happen to be interleaved in the following way (assume the initial value of X as 10000):

T3	T4	Value of X	
		T3	T4
READ X		10000	
	READ X		10000
SUB 5000		5000	
	ADD 3000		13000
WRITE X		5000	
	WRITE X		13000

After the execution of both the transactions the value X is 13000 while the semantically correct value should be 8000. The problem occurred as the update made by T3 has been overwritten by T4. The root cause of the problem was the fact that both the transactions had read the value of X as 10000. Thus one of the two updates has been lost and we say that a **lost update** has occurred.

There is one more way in which the lost updates can arise. Consider the following part of some transactions:

T5	T6	Value of x originally 2000	
		T5	T6
UPDATE X		3000	
	UPDATE X		4000
ROLLBACK		2000	

Here T5 & T6 updates the same item X. Thereafter T5 decides to undo its action and rolls back causing the value of X to go back to the original value that was 2000. In this case also the update performed by T6 had got lost and a lost update is said to have occurred.

2. **Unrepeatable reads:** Suppose T7 reads X twice during its execution. If it did not update X itself it could be very disturbing to see a different value of X in its next read. But this could occur if, between the two read operations, another transaction modifies X.

T7	T8	Assumed value of X=2000	
		T7	T8
READ X		2000	
	UPDATE X		3000
READ X		3000	

Thus, the inconsistent values are read and results of the transaction may be in error.

3. **Dirty Reads:** T10 reads a value which has been updated by T9. This update has not been committed and T9 aborts.

T9	T10	Value of x old value = 200	
		T9	T10
UPDATE X		500	
	READ X		500
ROLLBACK		200	?

Here T10 reads a value that has been updated by transaction T9 that has been aborted. Thus T10 has read a value that would never exist in the database and hence the problem. Here the problem is primarily of isolation of transaction.

4. **Inconsistent Analysis:** The problem as shown with transactions T1 and T2 where two transactions interleave to produce incorrect result during an analysis by Audit is the example of such a problem. This problem occurs when more than one data items are being used for analysis, while another transaction has modified some of those values and some are yet to be modified. Thus, an analysis transaction reads values from the inconsistent state of the database that results in inconsistent analysis.

Thus, we can conclude that the prime reason of problems of concurrent transactions is that a transaction reads an inconsistent state of the database that has been created by other transaction.

But how do we ensure that execution of two or more transactions have not resulted in a concurrency related problem?

Well one of the commonest techniques used for this purpose is to restrict access to data items that are being read or written by one transaction and is being written by another transaction. This technique is called locking. Let us discuss locking in more detail in the next section.

☞ Check Your Progress 1

1) What is a transaction? What are its properties? Can a transaction update more than on data values? Can a transaction write a value without reading it? Give an example of transaction.

……

……

……

2) What are the problems of concurrent transactions? Can these problems occur in transactions which do not read the same data values?

……

……

……

3) What is a Commit state? Can you rollback after the transaction commits?

……

……

……

2.4 THE LOCKING PROTOCOL

To control concurrency related problems we use locking. A lock is basically a variable that is associated with a data item in the database. A lock can be placed by a transaction on a shared resource that it desires to use. When this is done, the data item is available for the exclusive use for that transaction, i.e., other transactions are locked out of that data item. When a transaction that has locked a data item does not desire to use it any more, it should unlock the data item so that other transactions can use it. If a transaction tries to lock a data item already locked by some other transaction, it cannot do so and waits for the data item to be unlocked. The component of DBMS that controls and stores lock information is called the Lock Manager. The locking mechanism helps us to convert a schedule into a serialisable schedule. We had defined what a schedule is, but what is a serialisable schedule? Let us discuss about it in more detail:

2.4.1 Serialisable Schedules

If the operations of two transactions conflict with each other, how to determine that no concurrency related problems have occurred? For this, serialisability theory has been developed. Serialisability theory attempts to determine the **correctness** of the schedules. The rule of this theory is:

"A schedule S of n transactions is serialisable if it is equivalent to some **serial schedule** of the same 'n' transactions".

A serial schedule is a schedule in which either transaction T_1 is completely done before T_2 or transaction T_2 is completely done before T_1. For example, the following figure shows the two possible serial schedules of transactions T_1 & T_2.

Schedule A: T2 followed by T1			Schedule B: T1 followed by T2		
Schedule	**T1**	**T2**	**Schedule**	**T1**	**T2**
Read X		Read X	Read X	Read X	
Read Y		Read Y	Subtract 100	Subtract 100	
Display X+Y		Display X+Y	Write X	Write X	
Read X	Read X		Read Y	Read Y	
Subtract 100	Subtract 100		Add 100	Add 100	
Write X	Write X		Write Y	Write Y	
Read Y	Read Y		Read X		Read X
Add 100	Add 100		Read Y		Read Y
Write Y	Write Y		Display X+Y		Display X+Y

Figure 3: Serial Schedule of two transactions

Schedule C: An Interleaved Schedule		
Schedule	**T1**	**T2**
Read X	Read X	
Subtract 100	Subtract 100	
Read X		Read X

Write X	Write X	
Read Y		Read Y
Read Y	Read Y	
Add 100	Add 100	
Display X+Y		Display X+Y
Write Y	Write Y	

Figure 4: An Interleaved Schedule

Now, we have to figure out whether this interleaved schedule would be performing read and write in the same order as that of a serial schedule. If it does, then it is equivalent to a serial schedule, otherwise not. In case it is not equivalent to a serial schedule, then it may result in problems due to concurrent transactions.

Serialisability

Any schedule that produces the same results as a serial schedule is called a serialisable schedule. But how can a schedule be determined to be serialisable or not? In other words, other than giving values to various items in a schedule and checking if the results obtained are the same as those from a serial schedule, is there an algorithmic way of determining whether a schedule is serialisable or not?

The basis of the algorithm for serialisability is taken from the notion of a serial schedule. There are two possible serial schedules in case of two transactions (T1- T2 OR T2 - T1). Similarly, in case of three parallel transactions the number of possible serial schedules is 3!, that is, 6. These serial schedules can be:

T1-T2-T3	T1-T3-T2	T2-T1-T3
T2-T3-T1	T3-T1-T2	T3-T2-T1

Using the notion of precedence graph, an algorithm can be devised to determine whether an interleaved schedule is serialisbale or not. In this graph, the transactions of the schedule are represented as the nodes. This graph also has directed edges. An edge from the node representing transactions Ti to node Tj means that there exists a **conflicting operation** between Ti and Tj and Ti precedes Tj in some conflicting operations. It has been proved that a serialisable schedule is the one that contains no cycle in the graph.

Given a graph with no cycles in it, there must be a serial schedule corresponding to it.

The steps of constructing a precedence graph are:

1. Create a node for every transaction in the schedule.
2. Find the precedence relationships in conflicting operations. Conflicting operations are (read-write) or (write-read) or (write–write) on the same data item in two different transactions. But how to find them?

    2.1 For a transaction Ti which *reads* an item A, find a transaction Tj that *writes* A later in the schedule. If such a transaction is found, draw an edge from Ti to Tj.
    2.2 For a transaction Ti which has *written* an item A, find a transaction Tj later in the schedule that *reads* A. If such a transaction is found, draw an edge from Ti to Tj.
    2.3 For a transaction Ti which has *written* an item A, find a transaction Tj that *writes* A later than Ti. If such a transaction is found, draw an edge from Ti to Tj.
3. If there is any cycle in the graph, the schedule is not serialisable, otherwise, find the equivalent serial schedule of the transaction by traversing the transaction nodes starting with the node that has no input edge.

Let us use this algorithm to check whether the schedule as given in Figure 4 is Serialisable. *Figure 5* shows the required graph. Please note as per step 1, we draw the two nodes for T1 and T2. In the schedule given in *Figure 4*, please note that the transaction T2 reads data item X, which is subsequently written by T1, thus there is an edge from T2 to T1 (clause 2.1). Also, T2 reads data item Y, which is subsequently written by T1, thus there is an edge from T2 to T1 (clause 2.1). However, that edge already exists, so we do not need to redo it. Please note that there are no cycles in the graph, thus, the schedule given in *Figure 4* is serialisable. The equivalent serial schedule (as per step 3) would be T2 followed by T1.

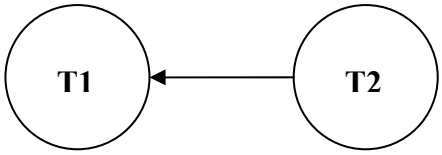


Figure 5: Test of Serialisability for the Schedule of Figure 4

Please note that the schedule given in part (c) of section 2.3 is not serialsable, because in that schedule, the two edges that exist between nodes T1 and T2 are:

- T1 writes X which is later read by T2 (clause 2.2), so there exists an edge from T1 to T2.
- T2 reads X which is later written by T1 (clause 2.1), so there exists an edge from T2 to T1.

Thus the graph for the schedule will be:

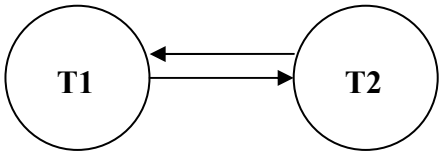


Figure 6: Test of Serialisability for the Schedule (c) of section 2.3

Please note that the graph above has a cycle T1-T2-T1, therefore it is not serialisable.

2.4.2 Locks

Serialisabilty is just a test whether a given interleaved schedule is ok or has a concurrency related problem. However, it does not ensure that the interleaved concurrent transactions do not have any concurrency related problem. This can be done by using locks. So let us discuss about what the different types of locks are, and then how locking ensures serialsability of executing transactions.

Types of Locks

There are two basic types of locks:

- Binary lock: This locking mechanism has two states for to a data item: locked or unlocked
- Multiple-mode locks: In this locking type each data item can be in three states read locked or shared locked, write locked or exclusive locked or unlocked.

Let us first take an example for binary locking and explain how it solves the concurrency related problems. Let us reconsider the transactions T1 and T2 for this purpose; however we will add to required binary locks to them.

Schedule	T1	T2
Lock X	Lock X	
Read X	Read X	
Subtract 100	Subtract 100	
Write X	Write X	
Unlock X	Unlock X	
Lock X		Lock X
Lock Y		Lock Y
Read X		Read X
Read Y		Read Y
Display X+Y		Display X+Y
Unlock X		Unlock X
Unlock Y		Unlock Y
Lock Y	Lock Y	
Read Y	Read Y	
Add 100	Add 100	
Write Y	Write Y	
Unlock Y	Unlock Y	

Figure 7: An incorrect locking implementation

Does the locking as done above solve the problem of concurrent transactions? No the same problems still remains. Try working with the old value. Thus, locking should be done with some logic in order to make sure that locking results in no concurrency related problem. One such solution is given below:

Schedule	T1	T2
Lock X	Lock X	
Lock Y	Lock Y	
Read X	Read X	
Subtract 100	Subtract 100	
Write X	Write X	
Lock X (issued by T2)	Lock X: denied as T1 holds the lock. The transaction T2 Waits and T1 continues.	
Read Y	Read Y	
Add 100	Add 100	
Write Y	Write Y	
Unlock X	Unlock X	
	The lock request of T2 on X can now be granted it can resumes by locking X	
Unlock Y	Unlock Y	
Lock Y		Lock Y
Read X		Read X
Read Y		Read Y
Display X+Y		Display X+Y
Unlock X		Unlock X
Unlock Y		Unlock Y

Figure 8: A correct but restrictive locking implementation

Thus, the locking as above when you obtain all the locks at the beginning of the transaction and release them at the end ensures that transactions are executed with no concurrency related problems. However, such a scheme limits the concurrency. We will discuss a two-phase locking method in the next subsection that provides sufficient concurrency. However, let us first discuss multiple mode locks.

Multiple-mode locks: It offers two locks: shared locks and exclusive locks. But why do we need these two locks? There are many transactions in the database system that never update the data values. These transactions can coexist with other transactions that update the database. In such a situation multiple reads are allowed on a data item, so multiple transactions can lock a data item in the shared or read lock. On the other hand, if a transaction is an updating transaction, that is, it updates the data items, it has to ensure that no other transaction can access (read or write) those data items that it wants to update. In this case, the transaction places an exclusive lock on the data items. Thus, a somewhat higher level of concurrency can be achieved in comparison to the binary locking scheme.

The properties of shared and exclusive locks are summarised below:

a) **Shared lock or Read Lock**

- It is requested by a transaction that wants to just read the value of data item.
- A shared lock on a data item does not allow an exclusive lock to be placed but permits any number of shared locks to be placed on that item.

b) **Exclusive lock**

- It is requested by a transaction on a data item that it needs to update.
- No other transaction can place either a shared lock or an exclusive lock on a data item that has been locked in an exclusive mode.

Let us describe the above two modes with the help of an example. We will once again consider the transactions T1 and T2 but in addition a transaction T11 that finds the total of accounts Y and Z.

Schedule	T1	T2	T11
S_Lock X		S_Lock X	
S_Lock Y		S_Lock Y	
Read X		Read X	
S_Lock Y			S_Lock Y
S_Lock Z			S_Lock Z
			Read Y
			Read Z
X_Lock X	X_Lock X. The exclusive lock request on X is denied as T2 holds the Read lock. The transaction T1 Waits.		
Read Y		Read Y	
Display X+Y		Display X+Y	
Unlock X		Unlock X	
X_Lock Y	X_Lock Y. The previous exclusive lock request on X is granted as X is unlocked. But the new exclusive lock request on Y is not granted as Y is locked by T2 and T11 in read mode. Thus T1 waits till both T2 and T11 will release the read lock on Y.		
Display Y+Z			Display Y+Z
Unlock Y		Unlock Y	
Unlock Y			Unlock Y
Unlock Z			Unlock Z
Read X	Read X		
Subtract 100	Subtract 100		
Write X	Write X		
Read Y	Read Y		
Add 100	Add 100		

Write Y	Write Y		
Unlock X	Unlock X		
Unlock Y	Unlock Y		

Figure 9: Example of Locking in multiple-modes

Thus, the locking as above results in a serialisable schedule. Now the question is can we release locks a bit early and still have no concurrency related problem? Yes, we can do it if we lock using two-phase locking protocol. This protocol is explained in the next sub-section.

2.4.3 Two Phase Locking (2PL)

The two-phase locking protocol consists of two phases:

Phase 1: The lock acquisition phase: If a transaction T wants to read an object, it needs to obtain the S (shared) lock. If T wants to modify an object, it needs to obtain X (exclusive) lock. No conflicting locks are granted to a transaction. **New locks on items can be acquired but no lock can be released till all the locks required by the transaction are obtained.**

Phase 2: Lock Release Phase: The existing locks can be released in any order but no new lock can be acquired **after a lock has been released**. The locks are held only till they are required.

Normally the locks are obtained by the DBMS. Any legal schedule of transactions that follows 2 phase locking protocol is guaranteed to be serialisable. The two phase locking protocol has been proved for it correctness. However, the proof of this protocol is beyond the scope of this Unit. You can refer to further readings for more details on this protocol.

There are two types of 2PL:

(1) The Basic 2PL
(2) Strict 2PL

The basic 2PL allows release of lock at any time after all the locks have been acquired. For example, we can release the locks in schedule of *Figure 8*, after we have Read the values of Y and Z in transaction 11, even before the display of the sum. This will enhance the concurrency level. The basic 2PL is shown graphically in *Figure 10.*

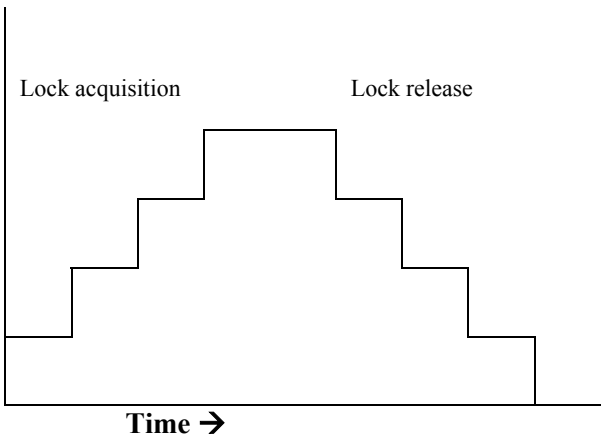


Figure 10: Basic Two Phase Locking

However, this basic 2PL suffers from the problem that it can result into loss of atomic / isolation property of transaction as theoretically speaking once a lock is released on a

data item it can be modified by another transaction before the first transaction commits or aborts.

To avoid such a situation we use strict 2PL. The strict 2PL is graphically depicted in *Figure 11*. However, the basic disadvantage of strict 2PL is that it restricts concurrency as it locks the item beyond the time it is needed by a transaction.

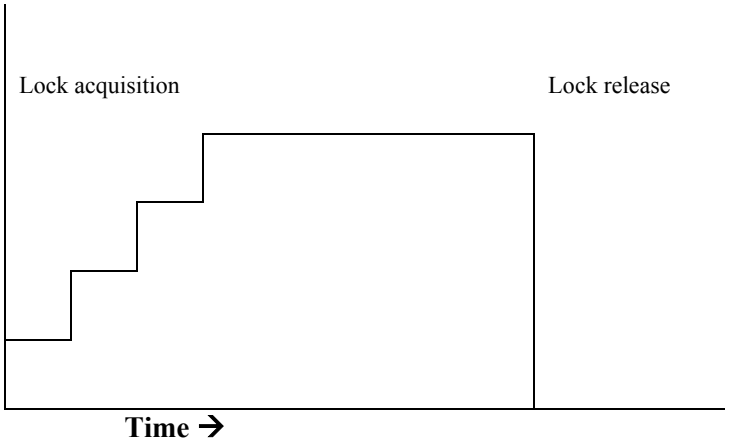


Figure 11: Strict Two Phase Locking

Does the 2PL solve all the problems of concurrent transactions? No, the strict 2PL solves the problem of concurrency and atomicity, however it introduces another problem: "Deadlock". Let us discuss this problem in next section.

☞ Check Your Progress 2

1) Let the transactions T1, T2, T3 be defined to perform the following operations:

   T1: Add one to A
   T2: Double A
   T3: Display A on the screen and set A to one.

   Suppose transactions T1, T2, T3 are allowed to execute concurrently. If A has initial value zero, how many possible correct results are there? Enumerate them.

   ……

   ……

2) Consider the following two transactions, given two bank accounts having a balance A and B.

   Transaction T1: Transfer Rs. 100 from A to B

   Transaction T2: Find the multiple of A and B.

   Create a non-serialisable schedule.

   ……

   ……

3) Add lock and unlock instructions (exclusive or shared) to transactions T1 and T2 so that they observe the serialisable schedule. Make a valid schedule.

   ……

   ……

2.5 DEADLOCK AND ITS PREVENTION

As seen earlier, though 2PL protocol handles the problem of serialisability, but it causes some problems also. For example, consider the following two transactions and a schedule involving these transactions:

TA	TB
X_lock A	X_lock A
X_lock B	X_lock B
:	:
:	:
Unlock A	Unlock A
Unlock B	Unlock B

Schedule

T1: X_lock A
T2: X_lock B
T1: X_lock B
T2: X_lock A

As is clearly seen, the schedule causes a problem. After T1 has locked A, T2 locks B and then T1 tries to lock B, but unable to do so waits for T2 to unlock B. Similarly, T2 tries to lock A but finds that it is held by T1 which has not yet unlocked it and thus waits for T1 to unlock A. At this stage, neither T1 nor T2 can proceed since both of these transactions are waiting for the other to unlock the locked resource.

Clearly the schedule comes to a halt in its execution. The important thing to be seen here is that both T1 and T2 follow the 2PL, which guarantees serialisability. So whenever the above type of situation arises, we say that a deadlock has occurred, since two transactions are **waiting for a condition that will never occur**.

Also, the deadlock can be described in terms of a directed graph called a "*wait for*" graph, which is maintained by the lock manager of the DBMS. This graph G is defined by the pair (V, E). It consists of a set of vertices/nodes V is and a set of edges/arcs E. Each transaction is represented by node and an arc from Ti → Tj, if Tj holds a lock and Ti is waiting for it. When transaction Ti requests a data item currently being held by transaction Tj then the edge Ti → Tj is inserted in the "wait for" graph. This edge is removed only when transaction Tj_j is no longer holding the data item needed by transaction Ti.

A deadlock in the system of transactions occurs, if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, a periodic check for cycles in graph can be done. For example, the "wait-for" for the schedule of transactions TA and TB as above can be made as:

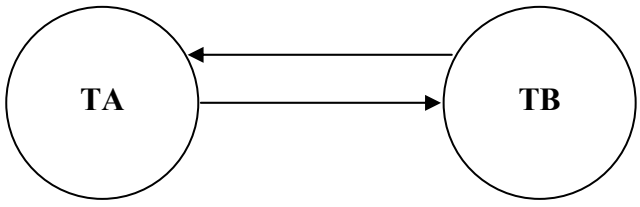


Figure 12: Wait For graph of TA and TB

In the figure above, TA and TB are the two transactions. The two edges are present between nodes TA and TB since each is waiting for the other to unlock a resource held by the other, forming a cycle, causing a deadlock problem. The above case shows a direct cycle. However, in actual situation more than two nodes may be there in a cycle.

A deadlock is thus a situation that can be created because of locks. It causes transactions to wait forever and hence the name deadlock. A deadlock occurs because of the following conditions:

a) Mutual exclusion: A resource can be locked in exclusive mode by only one transaction at a time.
b) Non-preemptive locking: A data item can only be unlocked by the transaction that locked it. No other transaction can unlock it.
c) Partial allocation: A transaction can acquire locks on database in a piecemeal fashion.
d) Circular waiting: Transactions lock part of data resources needed and then wait indefinitely to lock the resource currently locked by other transactions.

In order to prevent a deadlock, one has to ensure that at least one of these conditions does not occur.

A deadlock can be prevented, avoided or controlled. Let us discuss a simple method for deadlock prevention.

Deadlock Prevention

One of the simplest approaches for avoiding a deadlock would be to acquire all the locks at the start of the transaction. However, this approach restricts concurrency greatly, also you may lock some of the items that are not updated by that transaction (the transaction may have if conditions).Thus, better prevention algorithm have been evolved to prevent a deadlock having the basic logic: *not to allow circular wait to occur*. This approach rolls back some of the transactions instead of letting them wait.

There exist two such schemes. These are:

"Wait-die" scheme: The scheme is based on non-preventive technique. It is based on a simple rule:

```
If Ti requests a database resource that is held by Tj
        then if Ti has a smaller timestamp than that of Tj
                it is allowed to wait;
        else Ti aborts.
```

A timestamp may loosely be defined as the system generated sequence number that is unique for each transaction. Thus, a smaller timestamp means an older transaction. For example, assume that three transactions T1, T2 and T3 were generated in that sequence, then if T1requests for a data item which is currently held by transaction T2, it is allowed to wait as it has a smaller time stamping than that of T1. However, if T3 requests for a data item which is currently held by transaction T2, then T3 is rolled back (die).

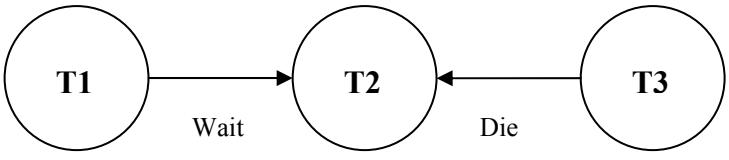


Figure 13: Wait-die Scheme of Deadlock prevention

"Wound-wait" scheme: It is based on a preemptive technique. It is based on a simple rule:

If Ti requests a database resource that is held by Tj
then if Ti has a larger timestamp (Ti is younger) than that of Tj
it is allowed to wait;
else Tj is wounded up by Ti.

For example, assume that three transactions T1, T2 and T3 were generated in that sequence, then if T1requests for a data item which is currently held by transaction T2, then T2 is rolled back and data item is allotted to T1 as T1 has a smaller time stamping than that of T2. However, if T3 requests for a data item which is currently held by transaction T2, then T3 is allowed to wait.

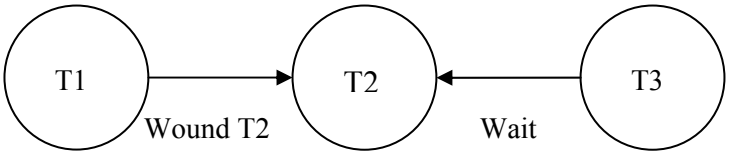


Figure 14: Wound-wait Scheme of Deadlock prevention

It is important to see that whenever any transaction is rolled back, it would not make a starvation condition, that is no transaction gets rolled back repeatedly and is never allowed to make progress. Also both "wait-die" & "wound-wait" scheme avoid starvation. The number of aborts & rollbacks will be higher in wait-die scheme than in the wound-wait scheme. But one major problem with both of these schemes is that these schemes may result in unnecessary rollbacks. You can refer to further readings for more details on deadlock related schemes.

2.6 OPTIMISTIC CONCURRENCY CONTROL

Is locking the only way to prevent concurrency related problems? There exist some other methods too. One such method is called an Optimistic Concurrency control. Let us discuss it in more detail in this section.

The basic logic in optimistic concurrency control is to allow the concurrent transactions to update the data items assuming that the concurrency related problem will not occur. However, we need to reconfirm our view in the validation phase. Therefore, the optimistic concurrency control algorithm has the following phases:

a) READ Phase: A transaction T reads the data items from the database into its private workspace. All the updates of the transaction can only change the local copies of the data in the private workspace.

b) VALIDATE Phase: Checking is performed to confirm whether the read values have changed during the time transaction was updating the local values. This is performed by comparing the current database values to the values that were read in the private workspace. In case, the values have changed the local copies are thrown away and the transaction aborts.

c) WRITE Phase: If validation phase is successful the transaction is committed and updates are applied to the database, otherwise the transaction is rolled back.

Some of the terms defined to explain optimistic concurrency contents are:

- write-set(T): all data items that are written by a transaction T
- read-set(T): all data items that are read by a transaction T

- Timestamps: for each transaction T, the start-time and the end time are kept for all the three phases.

More details on this scheme are available in the further readings. But let us show this scheme here with the help of the following examples:

Consider the set for transaction T1 and T2.

T1		T2	
Phase	**Operation**	**Phase**	**Operation**
-	-	Read	Reads the read set (T2). Let say variables X and Y and performs updating of local values
Read	Reads the read set (T1) lets say variable X and Y and performs updating of local values	-	-
Validate	Validate the values of (T1)	-	-
-	-	Validate	Validate the values of (T2)
Write	Write the updated values in the database and commit	-	-
-	-	Write	Write the updated values in the database and commit

In this example both T1 and T2 get committed. Please note that Read set of T1 and Read Set of T2 are both disjoint, also the Write sets are also disjoint and thus no concurrency related problem can occur.

T1	T2	T3
Operation	Operation	Operation
Read R(A)	--	--
--	Read R(A)	--
--	--	Read (D)
--	--	Update(D)
--	--	Update (A)
--	--	Validate (D,A) finds OK Write (D,A), COMMIT
--	Validate(A):Unsuccessful Value changed by T3	--
Validate(A):Unsuccessful Value changed by T3	--	--
ABORT T1	--	--
--	Abort T2	--

In this case both T1 and T2 get aborted as they fail during validate phase while only T3 is committed. Optimistic concurrency control performs its checking at the transaction commits point in a validation phase. The serialization order is determined by the time of transaction validation phase.

☞ Check Your Progress 3

1) Draw suitable graph for following locking requests, find whether the transactions are deadlocked or not.

T1: S_lock A	--	--
--	T2: X_lock B	--
--	T2: S_lock C	--

--	--	T3: X_lock C
--	T2: S_lock A	--
T1: S_lock B	--	--
T1: S_lock A	--	--
--	--	T3: S_lock A
All the unlocking requests start from here		

……

……

……

……

……

……

……

……

2) What is Optimistic Concurrency Control?

……

……

……

……

……

…….

2.7 SUMMARY

In this unit you have gone through the concepts of transaction and Concurrency Management. A transaction is a sequence of many actions. Concurrency control deals with ensuring that two or more users do not get into each other's way, i.e., updates of transaction one doesn't affect the updates of other transactions.

Serializability is the generally accepted criterion for correctness for the concurrency control. It is a concept related to concurrent schedules. It determines how to analyse whether any schedule is serialisable or not. Any schedule that produces the same results as a serial schedule is a serialisable schedule.

Concurrency Control is usually done via locking. If a transaction tries to lock a resource already locked by some other transaction, it cannot do so and waits for the resource to be unlocked.

Locks are of two type a) shared lock b) Exclusive lock. Then we move on to a method known as Two Phase Locking (2PL). A system is in a deadlock state if there exist a set of transactions such that every transaction in the set is waiting for another transaction in the set. We can use a deadlock prevention protocol to ensure that the system will never enter a deadlock state.

Finally we have discussed the method Optimistic Concurrency Control, another concurrency management mechanism.

2.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1) A transaction is the basic unit of work on a Database management system. It defines the data processing on the database. IT has four basic properties:

   a. Atomicity: transaction is done completely or not at all.
   b. Consistency: Leaves the database in a consistent state
   c. Isolation: Should not see uncommitted values
   d. Durability: Once committed the changes should be reflected.

   A transaction can update more than one data values. Some transactions can do writing of data without reading a data value.

   A simple transaction example may be: Updating the stock inventory of an item that has been issued. Please create a sample pseudo code for it.

2) The basic problems of concurrent transactions are:

   - Lost updates: An update is overwritten
   - Unrepeatable read: On reading a value later again an inconsistent value is found.
   - Dirty read: Reading an uncommitted value
   - Inconsistent analysis: Due to reading partially updated value.

   No these problems cannot occur if the transactions do not read the same data values. The conflict occurs only if one transaction updates a data value while another is reading or writing the data value.

3) Commit state is defined as when transaction has done everything correctly and shows the intent of making all the changes as permanent. No, you cannot rollback after commit.

Check Your Progress 2

1) There are six possible results, corresponding to six possible serial schedules:

Initially:	A = 0
T1-T2-T3:	A = 1
T1-T3-T2:	A = 2
T2-T1-T3:	A = 1
T2-T3-T1:	A = 2
T3-T1-T2:	A = 4
T3-T2-T1:	A = 3

2)

Schedule	T1	T2
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Read A		Read A
Read B		Read B

Read B	Read B	
Result = A * B		Result = A * B
Display Result		Display Result
B = B + 100	B = B + 100	
Write B	Write B	

Please make the precedence graph and find out that the schedule is not serialisable.

3)

Schedule	**T1**	**T2**
Lock A	Lock A	
Lock B	Lock B	
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Unlock A	Unlock A	
Lock A		Lock A: Granted
Lock B		Lock B: Waits
Read B	Read B	
B = B + 100	B = B + 100	
Write B	Write B	
Unlock B	Unlock B	
Read A		Read A
Read B		Read B
Result = A * B		Result = A * B
Display Result		Display Result
Unlock A		Unlock A
Unlock B		Unlock B

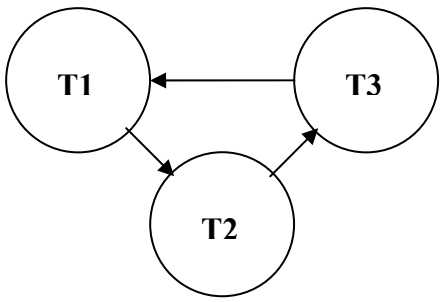
You must make the schedules using read and exclusive lock and a schedule in strict 2PL.

Check Your Progress 3

1) The transaction T1 gets the shared lock on A, T2 gets exclusive lock on B and Shared lock on A, while the transactions T3 gets exclusive lock on C.

- Now T2 requests for shared lock on C which is exclusively locked by T3, so cannot be granted. So T2 waits for T3 on item C.
- T1 now requests for Shared lock on B which is exclusively locked by T2, thus, it waits for T2 for item B. The T1 request for shared lock on C is not processed.
- Next T3 requests for exclusive lock on A which is share locked by T1, so it cannot be granted. Thus, T3 waits for T1 for item A.

The Wait for graph for the transactions for the given schedule is:



Since there exists a cycle, therefore, the schedule is deadlocked.

2) The basic philosophy for optimistic concurrency control is the optimism that nothing will go wrong so let the transaction interleave in any fashion, but to avoid any concurrency related problem we just validate our assumption before we make changes permanent. This is a good model for situations having a low rate of transactions.

UNIT 3 DATABASE RECOVERY AND SECURITY

Structure **Page Nos.**

3.0 INTRODUCTION

In the previous unit of this block, you have gone through the concepts of transactions and Concurrency management. In this unit we will introduce two important issues relating to database management systems.

A computer system suffers from different types of failures. A DBMS controls very critical data of an organisation and therefore must be reliable. However, the reliability of the database system is linked to the reliability of the computer system on which it runs. In this unit we will discuss recovery of the data contained in a database system following failure of various types and present the different approaches to database recovery. The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, software failures, power outages, accidents, unforeseen situations and natural or man-made disasters. Database recovery techniques are methods of making the database consistent till the last possible consistent state. The aim of recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

The second main issue that is being discussed in this unit is Database security. "Database security" is protection of the information contained in the database against unauthorised access, modification or destruction. The first condition for security is to have Database integrity. "Database integrity" is the mechanism that is applied to ensure that the data in the database is consistent.

Let us discuss all these terms in more detail in this unit.

3.1 OBJECTIVES

At the end of this unit, you should be able to:

- describe the terms RECOVERY and INTEGRITY;
- describe Recovery Techniques;
- define Error and Error detection techniques, and
- describe types of Authorisation.

3.2 WHAT IS RECOVERY?

During the life of a transaction, that is, a after the start of a transaction but before the transaction commits, several changes may be made in a database state. The database during such a state is in an inconsistent state. What happens when a failure occurs at this stage? Let us explain this with the help of an example:

Assume that a transaction transfers Rs.2000/- from A's account to B's account. For simplicity we are not showing any error checking in the transaction. The transaction may be written as:

Transaction T1:

READ A
A = A – 2000
WRITE A
——————→ Failure
READ B
B = B + 2000
WRITE B
COMMIT

What would happen if the transaction fails after account A has been written back to database? As far as the holder of account A is concerned s/he has transferred the money but that has never been received by account holder B.

Why did this problem occur? Because although a transaction is considered to be atomic, yet it has a life cycle during which the database gets into an inconsistent state and failure has occurred at that stage.

What is the solution? In this case where the transaction has not yet committed the changes made by it, the partial updates need to be undone.

How can we do that? By remembering information about a transaction such as when did it start, what items it updated etc. All such details are kept in a log file. We will study about log in Section 3.3. But first let us analyse the reasons of failure.

Failures and Recovery

In practice several things might happen to prevent a transaction from completing. Recovery techniques are used to bring database, which does not satisfy consistency requirements, into a consistent state. If a transaction completes normally and commits then all the changes made by the transaction on the database are permanently registered in the database. They should not be lost (please recollect the durability property of transactions given in Unit 2). But, if a transaction does not complete normally and terminates abnormally then all the changes made by it should be discarded. An abnormal termination of a transaction may be due to several reasons, including:

a) user may decide to abort the transaction issued by him/ her
b) there might be a deadlock in the system
c) there might be a system failure.

The recovery mechanisms must ensure that a consistent state of database can be restored under all circumstances. In case of transaction abort or deadlock, the system remains in control and can deal with the failure but in case of a system failure the system loses control because the computer itself has failed. Will the results of such failure be catastrophic? A database contains a huge amount of useful information and

any system failure should be recognised on the restart of the system. The DBMS should recover from any such failures. Let us first discuss the kinds of failure for identifying how to recover.

3.2.1 Kinds of Failures

The kinds of failures that a transaction program during its execution can encounter are:

1) **Software failures:** In such cases, a software error abruptly stops the execution of the current transaction (or all transactions), thus leading to losing the state of program excution and the state/ contents of the buffers. But what is a buffer? A buffer is the portion of RAM that stores the partial contents of database that is currently needed by the transaction. The software failures can further be subdivided as:
    a) Statement or application program failure
    b) Failure due to viruses
    c) DBMS software failure
    d) Operating system failure

    A Statement of program may cause abnormal termination if it does not execute completely. This happens if during the execution of a statement, an integrity constraint gets violated. This leads to abnormal termination of the transaction due to which any prior updates made by the transaction may still get reflected in the database leaving it in an inconsistent state.

    A failure of transaction can occur if some code in a transaction program leads to its
    abnormal termination. For example, a transaction can go into an infinite loop. In such a case the only way to break the loop is to abort the program. Similarly, the failure can be traced to the operating system or DBMS and transactions are aborted abruptly. Thus part of the transaction that was executed before abort may cause some updates in database, and hence the database is updated only partially which leads to an inconsistent state of database.

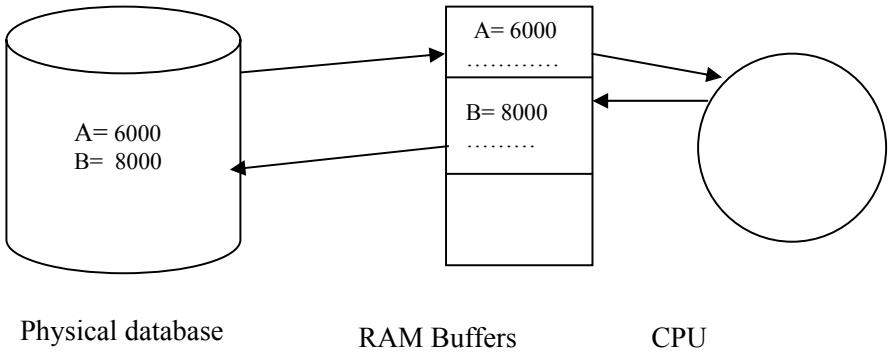
2) **Hardware failure:** Hardware failures are those failures when some hardware chip or disk fails. This may result in loss of data. One such problem can be that a disk gets damaged and cannot be read any more. This may be due to many reasons. For example, a voltage fluctuation in the power supply to the computer makes it go off or some bad sectors may come on disk or there is a disk crash. In all these cases, the database gets into an inconsistent state.

3) **External failure:** A failure can also result due to an external cause, such as fire, earthquakes, floods, etc. The database must be duly backed up to avoid problems occurring due to such failures.

In practice software failures are more common than hardware failures. Fortunately, recovery from software failures is much quicker.

The basic unit of recovery is a transaction. But, how are the transactions handled during recovery? Consider that some transactions are deadlocked, then at least one of these transactions has to be restarted to break the deadlock and thus the partial updates made by such restarted program in the database need to be **undone** so that the database does not go to an inconsistent state. So the transaction may have to be rolled back which makes sure that the transaction does not bring the database to an inconsistent state. This is one form of recovery. Let us consider a case when a transaction has committed but the changes made by the transaction have not been communicated to permanently stored physical database. A software failure now occurs and the contents of the CPU/ RAM are lost. This leaves the database in an inconsistent state. Such failure requires that on restarting the system the database be brought to a consistent state using **redo** operation. The redo operation makes the

changes made by the transaction again to bring the system to a consistent state. The database system then can be made available to the users. The point to be noted here is that the database updates are performed in the buffer in the memory. *Figure 1* shows some cases of undo and redo. You can create more such cases.



	Physical Database	RAM	Activity
Case 1	A=6000 B=8000	A=4000 B=8000	Transaction T1 has just changed the value in RAM. Now it aborts, value in RAM is lost. No problem. But we are not sure that the physical database has been written back, so must undo.
Case 2	A=4000 B=8000	A=4000 B=8000	The value in physical database has got updated due to buffer management, now the transaction aborts. The transaction must be undone.
Case 3	A=6000 B=8000	A=4000 B=10000 Commit	The value B in physical database has not got updated due to buffer management. In case of failure now when the transaction has committed. The changes of transaction must be redone to ensure transfer of correct values to physical database.

Figure 1: Database Updates And Recovery

3.2.2 Failure Controlling Methods

Failures can be handled using different recovery techniques that are discussed later in the unit. But the first question is do we really need recovery techniques as a failure control mechanism? The recovery techniques are somewhat expensive both in terms of time and in memory space for small systems. In such a case it is more beneficial to better avoid the failure by some checks instead of deploying recovery technique to make database consistent. Also, recovery from failure involves manpower that can be used in some other productive work if failure can be avoided. It is, therefore, important to find out some general precautions that help in controlling failure. Some of these precautions may be:

- having a regulated power supply.
- having a better secondary storage system such as RAID.
- taking periodic backup of database states and keeping track of transactions after each recorded state.
- properly testing the transaction programs prior to use.
- setting important integrity checks in the databases as well as user interfaces etc.

However, it may be noted that if the database system is critical it must use a DBMS that is suitably equipped with recovery procedures.

3.2.3 Database Errors

An error is said to have occurred if the execution of a command to manipulate the database cannot be successfully completed either due to inconsistent data or due to state of program. For example, there may be a command in program to store data in database. On the execution of command, it is found that there is no space/place in database to accommodate that additional data. Then it can be said that an error has occurred. This error is due to the physical state of database storage.

Broadly errors are classified into the following categories:

1) **User error:** This includes errors in the program (e.g., Logical errors) as well as errors made by online users of database. These types of errors can be avoided by applying some check conditions in programs or by limiting the access rights of online users e.g., read only. So only updating or insertion operation require appropriate check routines that perform appropriate checks on the data being entered or modified. In case of an error, some prompts can be passed to user to enable him/her to correct that error.

2) **Consistency error:** These errors occur due to the inconsistent state of database caused may be due to wrong execution of commands or in case of a transaction abort. To overcome these errors the database system should include routines that check for the consistency of data entered in the database.

3) **System error:** These include errors in database system or the OS, e.g., deadlocks. Such errors are fairly hard to detect and require reprogramming the erroneous components of the system software.

Database errors can result from failure or can cause failure and thus will require recovery. However, one of the main tasks of database system designers is to make sure that errors minimised. These concepts are also related to database integrity and have also been discusses in a later section.

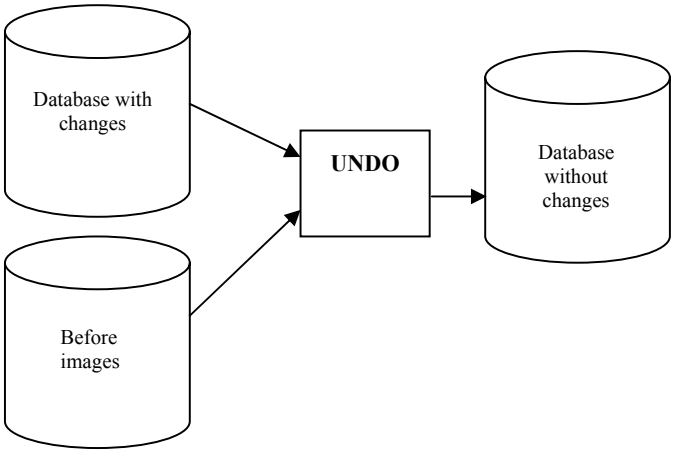
Comment: These are really the normal precautions one would take anyway – nothing specific to controlling database failures.

3.3 RECOVERY TECHNIQUES

After going through the types of failures and database errors, let us discuss how to recover from the failures. Recovery can be done using/restoring the previous consistent state (backward recovery) or by moving forward to the next consistent state as per the committed transactions (forward recovery) recovery. Please note that a system can recover from software and hardware failures using the forward and backward recovery only if the system log is intact. What is system log? We will discuss it in more detail, but first let us define forward and backward recovery.

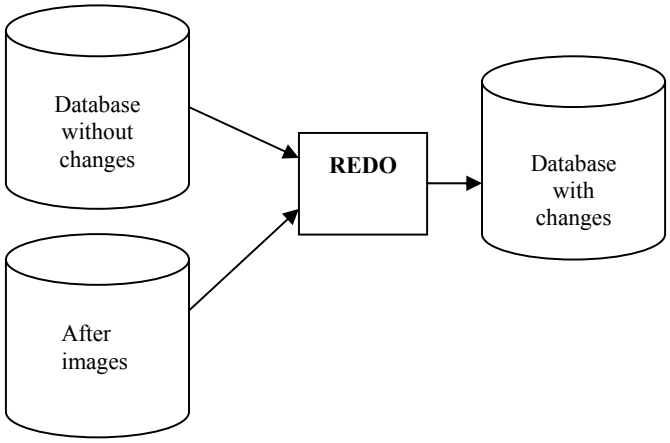
1) **Backward Recovery (UNDO)**

In this scheme the uncommitted changes made by a transaction to a database are undone. Instead the system is reset to the previous consistent state of database that is free from any errors.



2) **Forward Recovery (Redo)**

In this scheme the committed changes made by a transaction are reapplied to an earlier copy of the database.



In simpler words, when a particular error in a system is detected, the recovery system makes an accurate assessment of the state of the system and then makes the appropriate adjustment based on the anticipated results - had the system been error free.

One thing to be noted is that the Undo and Redo operations must be idempotent, i.e., executing them several times must be equivalent to executing them once. This characteristic is required to guarantee correct behaviour of database even if a failure occurs during the recovery process.

Depending on the above discussed recovery scheme, several types of recovery methods have been used. However, we define the most important recovery schemes used in most of the commercial DBMSs

Log based recovery

Let us first define the term transaction log in the context of DBMS. A transaction log is a record in DBMS that keeps track of all the transactions of a database system that update any data values in the database. A log contains the following information about a transaction:

- A transaction begin marker
- The transaction identification: The transaction id, terminal id or user id etc.

- The operations being performed by the transaction such as update, delete, insert.
- The data items or objects that are affected by the transaction including name of the table, row number and column number.
- The before or previous values (also called UNDO values) and after or changed values (also called REDO values) of the data items that have been updated.
- A pointer to the next transaction log record, if needed.
- The COMMIT marker of the transaction.

In a database system several transactions run concurrently. When a transaction commits, the data buffers used by it need not be written back to the physical database stored on the secondary storage as these buffers may be used by several other transactions that have not yet committed. On the other hand, some of the data buffers that may have updates by several uncommitted transactions might be forced back to the physical database, as they are no longer being used by the database. So the transaction log helps in remembering which transaction did which changes. Thus the system knows exactly how to separate the changes made by transactions that have already committed from those changes that are made by the transactions that did not yet commit. Any operation such as begin transaction, insert /delete/update and end transaction (commit), adds information to the log containing the transaction identifier and enough information to undo or redo the changes.

But how do we recover using log? Let us demonstrate this with the help of an example having three concurrent transactions that are active on ACCOUNTS table as:

Transaction T1	**Transaction T2**	**Transaction T3**
Read X	Read A	Read Z
Subtract 100	Add 200	Subtract 500
Write X	Write A	Write Z
Read Y		
Add 100		
Write Y		

Figure 2: The sample transactions

Assume that these transactions have the following log file (hypothetical) at a point:

Transaction Begin Marker	**Transaction Id**	**Operation on ACCOUNTS table**	**UNDO values (assumed)**	**REDO values**	**Transaction Commit Marker**
Y	T1	Sub on X Add on Y	500 800	400 Not done yet	N
Y	T2	Add on A	1000	1200	N
Y	T3	Sub on Z	900	400	Y

Figure 3: A sample (hypothetical) Transaction log

Now assume at this point of time a failure occurs, then how the recovery of the database will be done on restart.

Values	**Initial**	**Just before the failure**	**Operation Required for recovery**	**Recovered Database Values**
X	500	400 (assuming update has been done in physical database also)	UNDO	500
Y	800	800	UNDO	800

A	1000	1000 (assuming update has not been done in physical database)	UNDO	1000
Z	900	900 (assuming update has not been done in physical database)	REDO	400

Figure 4: The database recovery

The selection of REDO or UNDO for a transaction for the recovery is done on the basis of the state of the transactions. This state is determined in two steps:

- Look into the log file and find all the transactions that have started. For example, in *Figure 3*, transactions T1, T2 and T3 are candidates for recovery.
- Find those transactions that have committed. REDO these transactions. All other transactions have not committed so they should be rolled back, so UNDO them. For example, in *Figure 3* UNDO will be performed on T1 and T2; and REDO will be performed on T3.

Please note that in Figure 4 some of the values may not have yet been communicated to database, yet we need to perform UNDO as we are not sure what values have been written back to the database.
But how will the system recover? Once the recovery operation has been specified, the system just takes the required REDO or UNDO values from the transaction log and changes the inconsistent state of database to a consistent state. (Please refer to *Figure 3* and *Figure 4*).

Let us consider several transactions with their respective start & end (commit) times as shown in *Figure 5*.

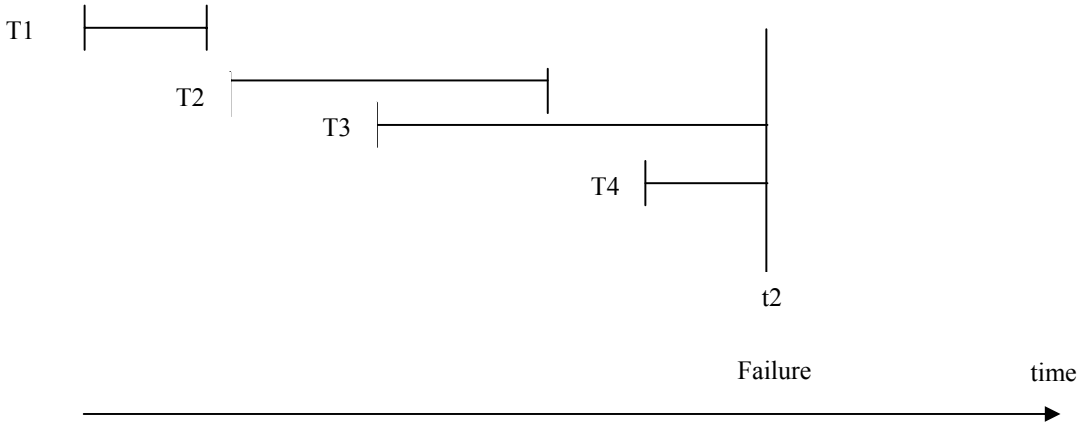


Figure 5: Execution of Concurrent Transactions

In the figure above four transactions are executing concurrently, on encountering a failure at time t2, the transactions T1 and T2 are to be REDONE and T3 and T4 will be UNDONE. But consider a system that has thousands of parallel transactions then all those transactions that have been committed may have to be redone and all uncommitted transactions need to be undone. That is not a very good choice as it requires redoing of even those transactions that might have been committed even hours earlier. So can we improve on this situation? Yes, we can take checkpoints. *Figure 6* shows a checkpoint mechanism:

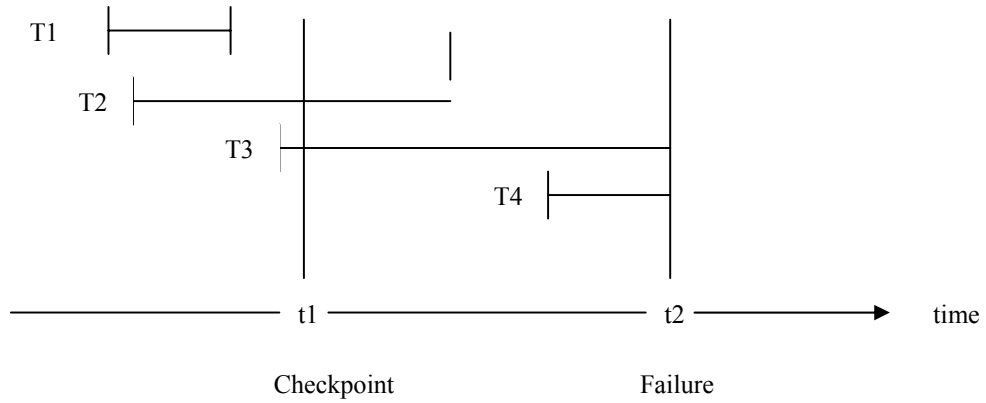


Figure 6: Checkpoint In Transaction Execution

A checkpoint is taken at time t_1 and a failure occurs at time t_2. Checkpoint transfers all the committed changes to database and all the system logs to stable storage (it is defined as the storage that would not be lost). At restart time after the failure the stable check pointed state is restored. Thus, we need to only REDO or UNDO those transactions that have completed or started after the checkpoint has been taken. The only possible disadvantages of this scheme may be that during the time of taking the checkpoint the database would not be available and some of the uncommitted values may be put in the physical database. To overcome the first problem the checkpoints should be taken at times when system load is low. To avoid the second problem some systems allow some time to the ongoing transactions to complete without restarting new transactions.

In the case of *Figure 6* the recovery from failure at t_2 will be as follows:

- The transaction T1 will not be considered for recovery as the changes made by it have already been committed and transferred to physical database at checkpoint t_1.
- The transaction T2 since it has not committed till the checkpoint t_1 but have committed before t_2, will be REDONE.
- T3 must be UNDONE as the changes made by it before checkpoint (we do not know for sure if any such changes were made prior to checkpoint) must have been communicated to the physical database. T3 must be restarted with a new name.
- T4 started after the checkpoint, and if we strictly follow the scheme in which the buffers are written back only on the checkpoint, then nothing needs to be done except restarting the transaction T4 with a new name.

The restart of a transaction requires the log to keep information of the new name of the transaction and maybe give higher priority to this newer transaction.

But one question is still unanswered that is during a failure we lose database information in RAM buffers, we may also lose log as it may also be stored in RAM buffers, so how does log ensure recovery?

The answer to this question lies in the fact that for storing transaction log we follow a **Write Ahead Log Protocol.** As per this protocol, the transaction logs are written to stable storage before any item is updated. Or more specifically it can be stated as; **the undo potion of log is written to stable storage prior to any updates and redo portion of log is written to stable storage prior to commit.**

Log based recovery scheme can be used for any kind of failure provided you have stored the most recent checkpoint state and most recent log as per write ahead log

protocol into the stable storage. Stable storage from the viewpoint of external failure requires more than one copy of such data at more than one location. You can refer to the further readings for more details on recovery and its techniques.

☞ Check Your Progress 1

1) What is the need of recovery? What is it the basic unit of recovery?

……

……

2) What is a checkpoint? Why is it needed? How does a checkpoint help in recovery?

……

……

3) What are the properties that should be taken into consideration while selecting recovery techniques?

……

……..……..……..

3.4 SECURITY AND INTEGRITY

After going through the concepts of database recovery in the previous section, let us now deal with an important concept that helps in minimizing consistency errors in database systems. These are the concepts of database security and integrity.

Information security is the protection of information against unauthorised disclosure, alteration or destruction. Database security is the protection of information that is maintained in a database. It deals with ensuring only the "right people" get the right to access the "right data". By right people we mean those people who have the right to access/update the data that they are requesting to access/update with the database. This should also ensure the confidentiality of the data. For example, in an educational institution, information about a student's grades should be made available only to that student, whereas only the university authorities should be able to update that information. Similarly, personal information of the employees should be accessible only to the authorities concerned and not to everyone. Another example can be the medical records of patients in a hospital. These should be accessible only to health care officials.

Thus, one of the concepts of database security is primarily a specification of access rules about who has what type of access to what information. This is also known as the problem of Authorisation. These access rules are defined at the time database is defined. The person who writes access rules is called the authoriser. The process of ensuring that information and other protected objects are accessed only in authorised ways is called access control. There may be other forms of security relating to physical, operating system, communication aspects of databases. However, in this unit, we will confine ourselves mainly to authorisation and access control using simple commands.

The term integrity is also applied to data and to the mechanism that helps to ensure its consistency. Integrity refers to the avoidance of accidental loss of consistency. Protection of database contents from unauthorised access includes legal and ethical issues, organization policies as well as database management policies. To protect database several levels of security measures are maintained:

1) **Physical:** The site or sites containing the computer system must be physically secured against illegal entry of unauthorised persons.

2) **Human**: An Authorisation is given to a user to reduce the chance of any information leakage and unwanted manipulations.

3) **Operating System:** Even though foolproof security measures are taken to secure database systems, weakness in the operating system security may serve as a means of unauthorised access to the database.

4) **Network:** Since databases allow distributed or remote access through terminals or network, software level security within the network software is an important issue.

5) **Database system:** The data items in a database need a fine level of access control. For example, a user may only be allowed to read a data item and is allowed to issue queries but would not be allowed to deliberately modify the data. It is the responsibility of the database system to ensure that these access restrictions are not violated. Creating database views as discussed in Unit 1 Section 1.6.1 of this block is a very useful mechanism of ensuring database security.

Comment: Can also talk of views to restrict read access.

To ensure database security requires implementation of security at all the levels as above. The Database Administrator (DBA) is responsible for implementing the database security policies in a database system. The organisation or data owners create these policies. DBA creates or cancels the user accounts assigning appropriate security rights to user accounts including power of granting and revoking certain privileges further to other users.

3.4.1 Relationship between Security and Integrity

Database security usually refers to access, whereas database integrity refers to avoidance of accidental loss of consistency. But generally, the turning point or the dividing line between security and integrity is not always clear. *Figure 7* shows the relationship between data security and integrity.

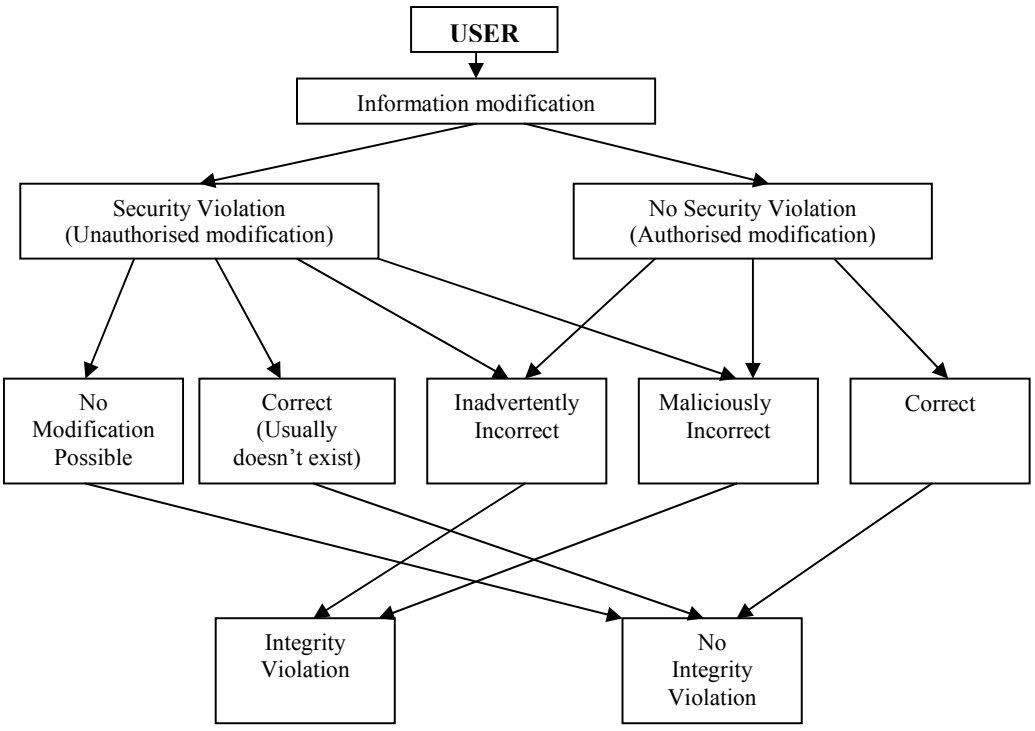


Figure 7: Relationship between security and Integrity

3.4.2 Difference between Operating System and Database Security

Security within the operating system can be implemented at several levels ranging from passwords for access to system, to the isolation of concurrent executing processes with the system. However, there are a few differences between security measures taken at operating system level as compared to those that of database system. These are:

- Database system protects more objects, as the data is persistent in nature. Also database security is concerned with different levels of granularity such as file, tuple, an attribute value or an index. Operating system security is primarily concerned with the management and use of resources.
- Database system objects can be complex logical structures such as views, a number of which can map to the same physical data objects. Moreover different architectural levels viz. internal, conceptual and external levels, have different security requirements. Thus, database security is concerned with the semantics – meaning of data, as well as with its physical representation. Operating system can provide security by not allowing any operation to be performed on the database unless the user is authorized for the operation concerned.

Figure 8 shows the architecture of a database security subsystem that can be found in any commercial DBMS.

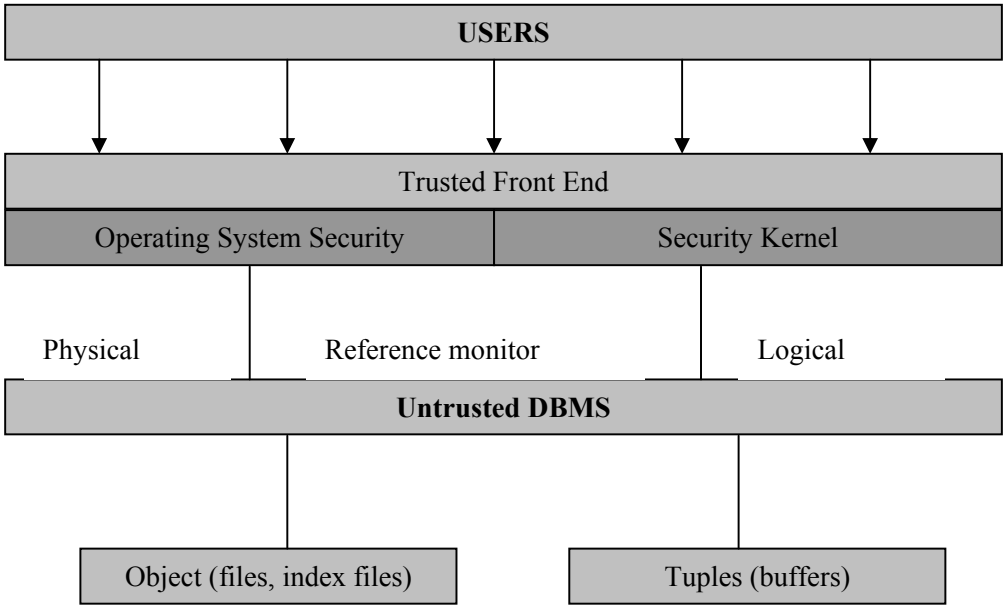


Figure 8: Database Security subsystem

3.5 AUTHORISATION

Authorisation is the culmination of the administrative policies of the organisation. As the name specifies, authorisation is a set of rules that can be used to determine which user has what type of access to which portion of the database. The following forms of authorisation are permitted on database items:

1) **READ**: it allows reading of data object, but not modification, deletion or insertion of data object.
2) **INSERT**: allows insertion of new data, but not the modification of existing data, e.g., insertion of tuple in a relation.
3) **UPDATE**: allows modification of data, but not its deletion. But data items like primary-key attributes may not be modified.

4) **DELETE**: allows deletion of data only.

A user may be assigned all, none or a combination of these types of Authorisation, which are broadly called access authorisations.

In addition to these manipulation operations, a user may be granted control operations like

1) Add: allows adding new objects such as new relations.

2) Drop: allows the deletion of relations in a database.

3) Alter: allows addition of new attributes in a relations or deletion of existing attributes from the database.

4) Propagate Access Control: this is an additional right that allows a user to propagate the access control or access right which s/he already has to some other, i.e., if user A has access right R over a relation S, then if s/he has propagate access control, s/he can propagate her/his access right R over relation S to another user B either fully or part of it. In SQL you can use WITH GRANT OPTION for this right.

You must refer to Section 1.5 of Unit 1 of this block for the SQL commands relating to data and user control.

The ultimate form of authority is given to the database administrator. He is the one who may authorize new users, restructure the database and so on. The process of Authorisation involves supplying information known only to the person the user has claimed to be in the identification procedure.

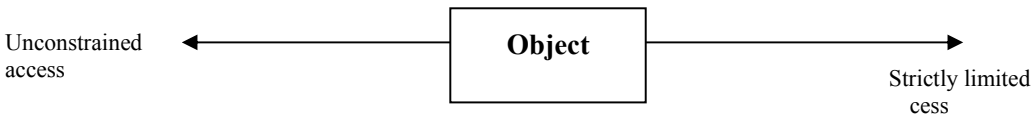
A basic model of Database Access Control

Models of database access control have grown out of earlier work on protection in operating systems. Let us discuss one simple model with the help of the following example:

Security problem: Consider the relation:

Employee (Empno, Name, Address, Deptno, Salary, Assessment)

Assuming there are two users: Personnel manager and general user . What access rights may be granted to each user? One extreme possibility is to grant an unconstrained access or to have a limited access.



One of the most influential protection models was developed by Lampson and extended by Graham and Denning. This model has 3 components:

1) A set of objects: where objects are those entities to which access must be controlled.

2) A set of subjects: where subjects are entities that request access to objects.

3) A set of all access rules: which can be thought of as forming an access (often referred to as authorisation matrix).

Let us create a sample authorisation matrix for the given relation:

Object / Subject	Empno	Name	Address	Deptno	Salary	Assessment
Personnel Manager	Read	All	All	All	All	All
General User	Read	Read	Read	Read	Not accessible	Not accessible

As the above matrix shows, Personnel Manager and general user are the two subjects. Objects of database are Empno, Name, Address, Deptno, Salary and Assessment. As per the access matrix, Personnel Manager can perform any operation on the database of an employee except for updating the Empno that may be self-generated and once given can never be changed. The general user can only read the data but cannot update, delete or insert the data into the database. Also the information about the salary and assessment of the employee is not accessible to the general user.

In summary, it can be said that the basic access matrix is the representation of basic access rules. These rules may be implemented using a view on which various access rights may be given to the users.

☞ Check Your Progress 2

1) What are the different types of data manipulation operations and control operations?

……

……..…….

……...……...……

……...……...……

……...……...……

……

……

2) What is the main difference between data security and data integrity?

……

……

……

……

……

3) What are the various aspects of security problem?

……

……..……

……...……

……...……

4) Name the 3 main components of Database Access Control Model?

……

……

……

……

3.6 SUMMARY

In this unit we have discussed the recovery of the data contained in a database system after failures of various types. The types of failures that the computer system is likely to be subject to include that of components or subsystems, software failures, power outages, accidents, unforeseen situations, and natural or man-made disasters. Database recovery techniques are methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost.

The basic technique to implement database recovery is by using data redundancy in the form of logs, and archival copies of the database. Checkpoint helps the process of recovery.

Security and integrity concepts are crucial since modifications in a database require the replacement of the old values. The DBMS security mechanism restricts users to only those pieces of data that are required for the functions they perform. Security mechanisms restrict the type of actions that these users can perform on the data that is accessible to them. The data must be protected from accidental or intentional (malicious) corruption or destruction. In addition, there is a privacy dimension to data security and integrity.

Security constraints guard against accidental or malicious tampering with data; integrity constraints ensure that any properly authorized access, alteration, deletion, or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data and this correctness has to be preserved in the presence of concurrent operations, error in the user's operation and application programs, and failures in hardware and software.

3.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Recovery is needed to take care of the failures that may be due to software, hardware and external causes. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost. One of the common techniques is log-based recovery. A transaction is the basic unit of recovery.

2) A checkpoint is a point when all the database updates and logs are written to stable storage. A checkpoint ensures that not all the transactions need to be REDONE or UNDONE. Thus, it helps in faster recovery from failure. You should create a sample example, for checkpoint having a sample transaction log.

3) The following properties should be taken into consideration:

- Loss of data should be minimal
- Recovery should be quick
- Recovery should be automatic
- Affect small portion of database.

Check Your Progress 2

1) Data manipulation operations are:

   - Read
   - Insert
   - Delete
   - Update

   Data control operations are:
   - Add
   - Drop
   - Alter
   - Propagate access control

2) Data security is the protection of information that is maintained in database against unauthorised access, modification or destruction. Data integrity is the mechanism that is applied to ensure that data in the database is correct and consistent.

3)
   - Legal, social and ethical aspects
   - Physical controls
   - Policy questions
   - Operational problems
   - Hardware control
   - Operating system security
   - Database administration concerns

4) The three components are:

   - Objects
   - Subjects
   - Access rights

UNIT 4 DISTRIBUTED AND CLIENT SERVER DATABASES

Structure **Page Nos.**

4.0 INTRODUCTION

In the previous units, we have discussed the basic issues relating to Centralised database systems. This unit discusses the distributed database systems which are primarily relational and one important implementation model: the client server model. This unit focuses on the basic issues involved in the design of distributed database designs. The unit also discusses some very basic concepts; in respect of client server computing including the basic structure, advantages/disadvantages etc. It will be worth mentioning here that almost all the commercial database management systems follow such a model of implementation. Although in client server model one of the concepts is the concept of distribution but it is primarily distribution of roles on server computers or rather distribution of work. So a client server system may be a centralised database.

4.1 OBJECTIVES

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

- differentiate DDBMS and conventional DBMS;
- define Network topology for DDBMS;
- define the concepts of horizontal and vertical fragmentation;
- define the concepts and rates of client server computing, and
- identify the needs of all these systems.

4.2 NEED FOR DISTRIBUTED DATABASE SYSTEMS

A *distributed database* is a set of database stored on multiple computers that appears to applications as a single database. As a result, an application can simultaneously access and modify the data in several databases in a network. Each database in the

system is controlled by its local server but cooperates to maintain the consistency of the global distributed database. The computers in a distributed system communicate with each other through various communication media, such as high-speed buses or telephone line. They don't share main memory, nor a clock, although, to work properly many applications on different computers might have to synchronise their clocks. In some cases the absolute time might be important and the clocks might have to be synchronised with atomic clocks.

The processors in a distributed system may vary in size and function such as small microcomputers, workstation, minicomputers, and large general-purpose computer system. These processors are referred to by sites, nodes, computers, and so on, depending on the context in which they are mentioned. We mainly use the term site, in order to emphasise 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 centralised and distributed database systems is that, in the former, the data resides in one single Centralised control, while in the latter, the data resides in several sets under the control of local distributed DBMS components which are under the control of one DBMS. As we shall see, this distribution of data is the cause of many difficulties that will be addressed in this unit.

Independent or decentralised systems were normally used in earlier days. There was duplication of hardware and other facilities. Evolution of computer systems also led to incompatible procedures and lack of management control. A centralised database system was then evolved. In a centralised database the DBMS and data reside at a single database instance. Although for recovery purposes we keep redundant database information yet it is under the control of a single DBMS. A further enhancement of the centralised database system may be to provide access to centralised data from a number of distributed locations through a network. In such a system a site failure except the central site will not result in total system failure. Although, communication technology has greatly improved yet the centralised approach may create problems for an organization that has geographically dispersed operations and data has to be accessed from a centralised database. Some of the problems may be:

a. loss of messages between a local and central site;
b. failure of communication links between local and central site. This would make the system unreliable;
c. excessive load on the central site would delay queries and accesses. A single site would have to bear a large number of transactions and hence would require large computing systems.

The problems as above could be addressed using distributed database systems. It improves the reliability and sharability of data and the efficiency of data access. Distributed Database Systems can be considered a system connected to intelligent remote devices each of which can itself act as a local database repository. All data is accessible from each site. The distributed system increases the efficiency of access because multiple of sites can co-ordinate efficiently to respond to a query and control & processing is limited to this DBMS.

4.3 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 at several sites. The architecture of Distributed Database systems is given in *Figure 1*

Figure 1: **Three different database system architectures. (a) No database sharing architecture. (b) A networked architecture with a centralised database. (c) A distributed database architecture**

The execution of global transactions on the distributed architecture requires communication among the sites. *Figure 2* illustrates a representative distributed database system architecture having transactions. Please note that both the transactions shown are global in nature as they require data from both the sites.

Figure 2: Distributed Database Schema and Transactions

Some of the key issues involving DDBMS are:

- How the data is distributed?
- Is this data distribution hidden from the general users?
- How is the integration of data and its control including security managed locally and globally?
- How are the distributed database connected?

Well we will provide the basic answers to most of these questions during the course of this unit. However, for more details, you may refer to further readings.

The sites in a distributed 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 connection structures are depicted in *Figure 3*. 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 failure of some links or sites.

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 all over India), or over a small geographical area such as a single building or a number of adjacent buildings). The former type of network is based on wide area network, while the latter uses local-area network.

Figure 3: Some interconnection Networks

Since the sites in wide area 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 wide area links are telephone lines, microwave links, and satellite channels. Many newer enhanced communication technologies including fiber optics are also used for such links. The local-area network sites are close to each other, communication links are of higher speed and lower error rate than their counterparts in wide area networks. The most common links are twisted pair, base band coaxial, broadband coaxial, and fiber optics.

A Distributed Transaction

Let us illustrate the concept of a distributed transaction by considering a banking system consisting of three branches located in three 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 other branches of the bank. Suppose that the database systems at the various sites are based on the relational model. Each branch maintains its portion of the relation: DEPOSIT (DEPOSIT-BRANCH) where

DEPOSIT-BRANCH = (branch-name, account-number, customer-name, balance)

A site containing information about the four branches maintains the relation branch-details, which has the schema as:

BRANCH-DETAILS (branch-name, Financial_ health, city)

There are other relations maintained at the various sites which are ignored for the purpose of our example.

A local transaction is a transaction that accesses accounts in one single site, at which the transaction was initiated. A global transaction, on the other hand, is one which either accesses 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 Rs.5000 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 Rs.5000 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. A transaction finding the total financial standing of all branches is global.

What makes the above configuration a distributed database system are the facts that:

- The various sites may be locally controlled yet are aware of each other.
- Each site provides an environment for executing both local and global transactions.

However, a user need not know whether on underlying application is distributed or not.

4.4 ADVANTAGES AND DISADVANTAGES OF DDBMS

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 overheads. In this section, we shall elaborate briefly on each of these.

4.4.1 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

The geographical distribution of an organization can be reflected in the distribution of the data; 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. The main advantage here is that the user need not know the site from which data is being accessed. Data can be placed at the site close to the users who normally use that data. The local control of data allows establishing and enforcement of local policies regarding use of local data. A global database administrator (DBA) is responsible for the entire system. Generally, part of this responsibility is given to the local administrator, so that the local DBA can manage the local DBMS. Thus in the distributed banking system, it is possible for a user to get his/her information from any branch office. This external mechanism would, in effect to a user, look to be a single centralised database.

The primary advantage of accomplishing data sharing by means of data distribution is that each site is able to retain a degree of control over data stored locally. Depending upon the design of the distributed database system, each local administrator may have a different degree of autonomy. This is often a major advantage of distributed databases. In a centralised system, the database administrator of the central site controls the database and, thus, no local control is possible.

Reflects organisational structure

Many organizations are distributed over several locations. If an organisation has many offices in different cities, databases used in such an application are distributed over these locations. Such an organisation may keep a database at each branch office containing details of the staff that work at that location, the local properties that are for rent, etc. The staff at a branch office will make local inquiries to such data of the database. The company headquarters may wish to make global inquiries involving the access of data at all or a number of branches.

Improved Reliability

In a centralised DBMS, a server failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS, or a failure of a communication link making some sites inaccessible, does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. In particular, if data are replicated in several sites, a transaction needing a particular data item may find it at 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 services 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. The recovery from failure in distributed systems is much more complex than in a centralised system.

Improved availability

The data in a distributed system may be replicated so that it exists at more than one site. Thus, the failure of a node or a communication link does not necessarily make the data inaccessible. The ability of most of the systems to continue to operate despite the failure of one site results in increased availability which is crucial for database

systems used for real-time applications. For example, loss of access to data in an airline may result in the loss of potential ticket buyers to competitors.

Improved performance

As the data is located near the site of its demand, and given the inherent parallelism due to multiple copies, speed of database access may be better for distributed databases than that of the speed that is achievable through a remote centralised database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralised DBMS.

Speedup Query Processing

A query that involves data at several sites can be split into sub-queries. These sub-queries can be executed in parallel by several sites. Such parallel sub-query evaluation allows faster processing of a user's query. In those cases in which data is replicated, queries may be sent to the least heavily loaded sites.

Economics

It is now generally accepted that it costs less to create a system of smaller computers with the equivalent power of a single large computer. It is more cost-effective to obtain separate computers. The second potential cost saving may occurs where geographically remote access to distributed data is required. In such cases the economics is to minimise cost due to the data being transmitted across the network for data updates as opposed to the cost of local access. It may be more economical to partition the application and perform the processing locally at application site.

Modular growth

In distributed environments, it is easier to expand. New sites can be added to the network without affecting the operations of other sites, as they are somewhat independent. This flexibility allows an organisation to expand gradually. Adding processing and storage power to the network can generally result in better handling of ever increasing database size. A more powerful system in contrast, a centralised DBMS, would require changes in both the hardware and software with increasing size and more powerful DBMS to be procured.

4.4.2 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:

- **Higher Software development cost:** Distributed database systems are complex to implement and, thus, more costly. Increased complexity implies that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralised DBMS. In addition to software, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional maintenance costs to manage and maintain the local DBMSs and the network.

- **Greater potential for bugs:** Since the sites of a distributed system operate concurrently, it is more difficult to ensure the correctness of algorithms. The art of constructing distributed algorithms is an active and important area of research.

- **Increased processing overhead:** The exchange of messages and the additional computation required to achieve coordination among the sites is an overhead that does not arise in centralised systems.

- **Complexity:** A distributed DBMS that is reliable, available and secure is inherently more complex than a centralised DBMS. Replication of data discussed in the next section, also adds to complexity of the distributed DBMS. However, adequate data replication is necessary to have availability, reliability, and performance.

- **Security:** In a centralised system, access to the data can be easily controlled. However, in a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but also the network needs to be secured. Networks over a period have become more secure, still there is a long way to go.

- **Lack of standards and experience:** The lack of standards has significantly limited the potential of distributed DBMSs. Also, there are no tools or methodologies to help users convert a centralised DBMS into a distributed DBMS.

- **General Integrity control more difficult:** Integrity is usually expressed in terms of constraints or the rules that must be followed by database values. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be very high as the data is stored at various sites. However, with better algorithms we can reduce such costs.

- **Purpose:** General-purpose distributed DBMSs have not been widely accepted. We do not yet have the same level of experience in industry as we have with centralised DBMSs.

- **Database design more complex:** Besides the normal difficulties of designing a centralised database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication. The designer of distributes systems must balance the advantages against the disadvantages of distribution of data.

☞ Check Your Progress 1

1) What are the advantages of distributed databases over centralised Databases?

……

……

……

2) Differntiate between global and local transaction.

……

……

……

4.5 DESIGN OF DISTRIBUTED DATABASES

The distributed databases are primarily relational at local level. So a local database schema is the same as that of a centralised database design. However, a few more dimensions have been added to the design of distributed database. These are:

- **Replication:** It is defined as a copy of a relation. Each replica is stored at a different site. The alternative to replication is to store only one copy of a relation which is not recommended in distributed databases.

- **Fragmentation:** It is defined as partitioning of a relation into several fragments. Each fragment can be stored at a different site.

The distributed database design is a combination of both these concepts. Let us discuss them in more detail in the following subsections.

4.5.1 Data Replication

"If a relation R has its copies stored at two or more sites, then it is considered replicated".

But why do we replicate a relation?

There are various advantages and disadvantages of replication of relation

- **Availability:** A site containing the copy of a replicated relation fails, even then the relation may be found in another site. Thus, the system may continue to process at least the queries involving just read operation despite the failure of one site. Write can also be performed but with suitable recovery algorithm.

- **Increased parallelism:** Since the replicated date has many copies a query can be answered from the least loaded site or can be distributed. Also, with more replicas you have greater chances that the needed data is found on the site where the transaction is executing. Hence, data replication can minimise movement of data between sites.

- **Increased overheads on update:** On the disadvantage side, it will require the system to ensure that all replicas of a relation are consistent. This implies that all the replicas of the relation need to be updated at the same time, resulting in increased overheads. For example, in a banking system, if account information is replicated at various sites, it is necessary that the balance in a particular account should be the same at all sites.

The problem of controlling concurrent updates by several transactions to replicated data is more complex than the centralised approach of concurrency control. The management of replicas of relation can be simplified by choosing one of the replicas as the primary copy. For example, in a banking system, the primary copy of an account may be the site at which the account has been opened. Similarly, in an airline reservation system, the primary copy of the flight may be associated with the site at which the flight originates. The replication can be classified as:

Complete replication: It implies maintaining of a complete copy of the database at each site. Therefore, the reliability and availability and performance for query response are maximized. However, storage costs, communication costs, and updates are expensive. To overcome some of these problems relation, snapshots are sometimes used. A snapshot is defined as the copy of the data at a given time. These copies are updated periodically, such as, hourly or weekly. Thus, snapshots may not always be up to date.

Selective replication: This is a combination of creating small fragments of relation and replicating them rather than a whole relation. The data should be fragmented on need basis of various sites, as per the frequency of use, otherwise data is kept at a centralised site. The objective of this strategy is to have just the advantages of the other approach but none of the disadvantages. This is the most commonly used strategy as it provides flexibility.

4.5.2 Data Fragmentation

"Fragmentation involves decomposing the data in relation to non-overlapping component relations".

Why do we need to fragment a relation? The reasons for fragmenting a relation are:

Use of partial data by applications: In general, applications work with views rather than entire relations. Therefore, it may be more appropriate to work with subsets of relations rather than entire data.

Increases efficiency: Data is stored close to most frequently used site, thus retrieval would be faster. Also, data that is not needed by local applications is not stored, thus the size of data to be looked into is smaller.

Parallelism of transaction execution: A transaction can be divided into several sub-queries that can operate on fragments in parallel. This increases the degree of concurrency in the system, thus allowing transactions to execute efficiently.

Security: Data not required by local applications is not stored at the site, thus no unnecessary security violations may exist.

But how do we carry out fragmentation? Fragmentation may be carried out as per the following rules:

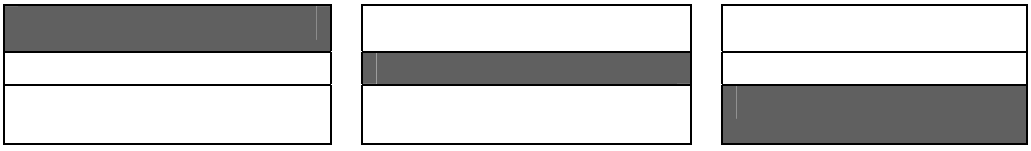
a) **Completeness:** This rule ensures that there is no loss of data during fragmentation. If a relation is decomposed into fragments, then each data item must appear in at least one fragment.

b) **Reconstruction:** This rule ensures preservation of functional dependencies. It should be possible to define a relational operation that will reconstruct the relation from its fragments.

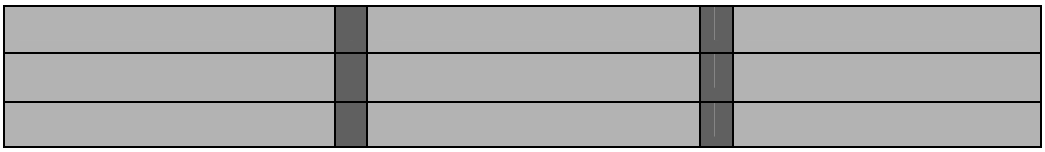
c) **Disjointness:** *A* data item that appears in a fragment should not appear in any other fragment. However, a typical fragmentation called vertical fragmentation is an exception to this rule. In vertical fragmentation the primary key attributes must be repeated to allow reconstruction of original relation. This rule ensures minimization of data redundancy.

What are the different types of fragmentation?

There are two main types of fragmentation: **horizontal** and **vertical**. Horizontal fragments, as the name suggests are subsets of tuples and vertical fragments are subsets of attributes (refer to Figure 4). There are also two other types of fragmentation: mixed, and derived–a type of horizontal fragmentation, are just introduced. A detailed discussion on them is beyond the scope of this unit.



(a) Horizontal Fragmentation



(b) Vertical Fragmentation

Figure 4: Horizontal and Vertical Fragmentation

Horizontal Fragmentation

Horizontal fragmentation groups together the tuples in a relation that are collectively used by the important transactions. A horizontal fragment is produced by specifying a WHERE clause condition that performs a restriction on the tuples in the relation. It can also be defined using the *Selection* operation of the relational algebra.

Example:

Let us illustrate horizontal fragmentation with the help of an example.

DEPOSIT (branch-code, account-number, customer-name, balance)

A sample relation instance of the relation DEPOSIT is shown in *Figure 5*.

Branch-code	Account number	Customer name	Balance
1101	3050	Suresh	5000
1101	2260	Swami	3360
1102	1170	Swami	2050
1102	4020	Khan	10000
1101	1550	Khan	620
1102	4080	Khan	1123
1102	6390	Khan	7500

Figure 5: Sample DEPOSIT relation

Mathematically a fragment may be defined as a selection on the global relation R. The reconstruction of the relation R can be obtained by taking the union of all fragments.

So let us decompose the table in Figure 5 into horizontal fragments. Let us do these fragments on the branch-code as 1101 and 1102

DEPOSIT1 obtained by selection on branch-code as 1101			
Branch-code	**Account number**	**Customer name**	**Balance**
1101	3050	Suresh	5000
1101	2260	Swami	3360
1101	1550	Khan	620
DEPOSIT2 obtained by selection on branch- code as 1102			
Branch-code	**Account number**	**Customer name**	**Balance**
1102	1770	Swami	2050
1102	4020	Khan	10000
1102	4080	Khan	1123
1102	6390	Khan	7500

Figure 6: Horizontal fragmentation of relation DEPOSIT

The two fragments can be defined in relational algebra as:

$$\text{DEPOSIT1} = \sigma_{\text{branch-code= 1101}} (\text{DEPOSIT})$$

$$\text{DEPOSIT2} = \sigma_{\text{branch-code= 1102}} (\text{DEPOSIT})$$

These two fragments are shown in *Figure 6*. Fragment 1 can be stored in the branch whose code is 1101 while the second fragment can be stored at branch 1102.

In our example, the fragments are disjoint. However, by changing the selection predicates used to construct the fragments; we may have overlapping horizontal fragments. This is a form of data replication.

Vertical Fragmentation

Vertical fragmentation groups together only those attributes in a relation that are used jointly by several important transactions. A vertical fragment is defined using the **Projection** operation of the relational algebra. In its most simple form, vertical

fragmentation is the same as that of decomposition. In general, a relation can be constructed on taking Natural join of all vertical fragments.

More generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-number to the scheme R. A tuple-number is a physical or logical address for a tuple. Since each tuple in R must have a unique address, the tuple-number attribute is a key to the new fragments obtained (please refer to *Figure 7*).

Branch-code	Account number	Customer name	Balance	Tuple-number
1101	3050	Suresh	5000	1
1101	2260	Swami	3360	2
1102	1170	Swami	2050	3
1102	4020	Khan	10000	4
1101	1550	Khan	620	5
1102	4080	Khan	1123	6
1102	6390	Khan	7500	7

Figure 7: The relation DEPOSIT of figure 5 with tuple- numbers

This relation now can be decomposed into two fragments as: shows a vertical decomposition of the scheme Deposit-scheme tuple number into:

$$\text{DEPOSIT3} = \prod_{\text{(branch-code. customer-name, tuple-number)}} (\text{DEPOSIT})$$

$$\text{DEPOSIT4} = \prod_{\text{(account-number, balance, tuple-number)}} (\text{DEPOSIT})$$

The example of Figure7 on this basis would become:

DEPOSIT3		
Branch-code	**Customer-name**	**Tuple-number**
1101	Suresh	1
1101	Swami	2
1102	Swami	3
1102	Khan	4
1101	Khan	5
1102	Khan	6
1102	Khan	7
DEPOSIT4		
Account number	**Balance**	**Tuple-number**
3050	5000	1
2260	3360	2
1170	2050	3
4020	10000	4
1550	620	5
4080	1123	6
6390	7500	7

Figure 8: Vertical fragmentation of relation DEPOSIT

How can we reconstruct the original relation from these two fragments? By taking natural join of the two vertical fragments on tuple-number. The tuple number allows direct retrieval of the tuples without the need for an index. Thus, this natural join may be computed much more efficiently than typical natural join.

However, please note that as the tuple numbers are system generated, therefore they should not be visible to general users. If users are given access to tuple-number, it becomes impossible for the system to change tuple addresses.

Mixed fragmentation

Sometimes, horizontal or vertical fragmentation of a database schema by itself is insufficient to adequately distribute the data for some applications. Instead, **mixed** or **hybrid** fragmentation is required. Mixed fragmentation consists of a horizontal fragment that is vertically fragmented, or a vertical fragment that is then horizontally fragmented.

Derived horizontal fragmentation

Some applications may involve a join of two or more relations. If the relations are stored at different locations, there may be a significant overhead in processing the join. In such cases, it may be more appropriate to ensure that the relations, or fragments of relations, that are joined together are at the same location. We can achieve this using derived horizontal fragmentation.

After going through the basis of concepts relating to distributed database systems, let us sketch the process of design of distributed database systems.

A step-wise distributed database design methodology

Following is a step-wise methodology for distributed database design.

(1) Examine the nature of distribution. Find out whether an organisation needs to have a database at each branch office, or in each city, or possibly at a regional level. It has direct implication from the viewpoint of fragmentation. For example, in case database is needed at each branch office, the relations may fragment on the basis of branch number.
(2) Create a detailed global E-R diagram, if so needed and identify relations from entities.
(3) Analyse the most important transactions in the system and identify where horizontal or vertical fragmentation may be desirable and useful.
(4) Identify the relations that are not to be fragmented. Such relations will be replicated everywhere. From the global ER diagram, remove the relations that are not going to be fragmented.
(5) Examine the relations that are on one-side of a relationship and decide a suitable fragmentation schema for these relations. Relations on the many-side of a relationship may be the candidates for derived fragmentation.
(6) During the previous step, check for situations where either vertical or mixed fragmentation would be needed, that is, where the transactions require access to a subset of the attributes of a relation.

☞ Check Your Progress 2

1) What are the rules while fragmenting data?

……

……

……

……

2) What is Data Replication? What are its advantages & disadvantages?

……

……

……

……

……

……

……

4.6 CLIENT SERVER DATABASES

The concept behind the Client/Server systems is concurrent, cooperative processing. It is an approach that presents a single systems view from a user's viewpoint. It involves processing on multiple, interconnected machines. It provides coordination of activities in a manner transparent to end-users. Remember, client-server database is distribution of activities into clients and a server. It may have a centralised or distributed database system at the server backend. It is primarily a very popular commercial database implementation model.

4.6.1 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.

4.6.2 Need for 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 number of users grows, client machines can be added to the network while as the load on the database server increases more servers can be connected through the network. Thus, client-server combination is a scalable combination. Server machines are more powerful machines database services to multiple client requests.

The client-server systems are connected though the network. This network need not only be the Local Area Network (LAN). It can also be the Wide Area Network (WAN) across multiple cities. The client and server machines communicate through standard application program interfaces (called API), and remote procedure calls (RPC). The language through which RDBMS based C/S environment communicate is the structured query language (SQL).

4.6.3 Structure of Client Server Systems

In client/server architecture, clients represent users who need services while servers provide services. Both client and server are a combination of hardware and software. Servers are separate logical objects that communicate with clients over a network to perform tasks together. A client makes a request for a service and receives a reply to that request. A server receives and processes a request, and sends back the required response. The client/server systems may contain two different types of architecture - 2-Tier and 3-Tier Client/Server Architectures

Every client/server application contains three functional units:

- Presentation logic which provides the human/machine interaction (the user interface). The presentation layer handles input from the keyboard, mouse, or

other input devices and provides output in the form of screen displays. For example, the ATM machine of a bank provides such interfaces.

- Business logic is the functionality provided to an application program. For example, software that enables a customer to request to operate his/her balance on his/her account with the bank is business logic. It includes rules for withdrawal, for minimum balance etc. It is often called business logic because it contains the business rules that drive a given enterprise.

- The bottom layer provides the generalized services needed by the other layers including file services, print services, communications services and database services. One example of such a service may be to provide the records of customer accounts.

These functional units can reside on either the client or on one or more servers in the application:

Figure 9: A client-server system

In general two popular client/server systems are:

- 2-Tier client Server Models

- 3-Tier client server Model

2-Tier Client/Server Models

Initial two-tier (client/server) applications were developed to access large databases available on the server side and incorporated the rules used to manipulate the data with the user interface into the client application. The primary task of the server was simply to process as many requests for data storage and retrieval as possible.

Two-tier client/server provides the user system interface usually on the desktop environment to its users. The database management services are usually on the server that is a more powerful machine and services many clients. Thus, 2-Tier client-server architecture splits the processing between the user system interface environment and the database management server environment. The database management server also provides stored procedures and triggers. There are a number of software vendors that provide tools to simplify the development of applications for the two-tier client/server architecture.

In 2-tier client/server applications, the business logic is put inside the user interface on the client or within the database on the server in the form of stored procedures. This results in division of the business logic between the client and server. File servers and database servers with stored procedures are examples of 2-tier architecture.

The two-tier client/server architecture is a good solution for distributed computing. Please note the use of words distributed computing and not distributed databases. A 2-tier client server system may have a centralised database management system or distributed database system at the server or servers. A client group of clients on a LAN can consist of a dozen to 100 people interacting simultaneously. A client server system does have a number of limitations. When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via communication messages with each client, even when no work is being done.

A second limitation of the two-tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications. The implementations of the two-tier architecture provides limited flexibility in moving program functionality from one server to another without manually regenerating procedural code.

Figure 10: 2-tier and 3 tier client server systems

Some of the major functions performed by the client of a two-tier application are: present a user interface, gather and process user input, perform the requested processing, report the status of the request.

This sequence of commands can be repeated as many times as necessary. Because servers provide only access to the data, the client uses its local resources to perform most of the processing. The client application must contain information about where the data resides and how it is organised in the server database. Once the data has been retrieved, the client is responsible for formatting and displaying it to the user.

3-tier architecture

As the number of clients increases the server would be filled with the client requests. Also, because much of the processing logic was tied to applications, changes in business rules lead to expensive and time-consuming alterations to source code. Although the ease and flexibility of two-tier architecture tools continue to drive many small-scale business applications, the need for faster data access and rapid developmental and maintenance timelines has persuaded systems developers to seek out a new way of creating distributed applications.

The three-tier architecture emerged to overcome the limitations of the two tier architecture (also referred to as the multi-tier architecture). In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. The middle tier may consist of transaction processing monitors, message servers, or application servers.

The middle-tier can perform queuing, application execution, and database staging. For example, on a middle tier that provides queuing, the client can deliver its request to the middle layer and simply gets disconnected because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritisation for various tasks that are currently being performed.

The three-tier client/server architecture has improved performance for client groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier approach. Flexibility is in terms of simply moving an application on to different computers in three-tier architecture. It has become as simple as “drag and drop” tool. Recently, mainframes have found a new use as servers in three-tier architectures:

In 3-tier client/server applications, the business logic resides in the middle tier, separate from the data and user interface. In this way, processes can be managed and deployed separately from the user interface and the database. Also, 3-tier systems can integrate data from multiple sources.

4.6.4 Advantages of Client Server Computing

Client/Server systems have following advantages:

- They provide low cost and user-friendly environment
- They offer expandability that ensures that the performance degradation is not so much with increased load.
- They allow connectivity with the heterogeneous machines deals with real time data feeders like ATMs, Numerical machines.
- They allow enforcement of database management activities including security, performance, backup, integrity to be a part of the database server machine. Thus, avoiding the requirement to write a large number of redundant piece of code dealing with database field validation and referential integrity.

- One major advantage of the client/server model is that by allowing multiple users to simultaneously access the same application data, updates from one computer are instantly made available to all computers that had access to the server.

Client/server systems can be used to develop highly complex multi-user database applications being handled by any mainframe computer.

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 on a slightly upgraded machine provided it offers the same version of RDBMSs.

A more detailed discussion on these topics is beyond the scope of this unit. You can refer to further readings for more details.

☞ Check your Progress 3

1) What are the advantages of Client/Server Computing?

……

……

2) Describe the Architecture of Client/Server system.

……

……

4.7 SUMMARY

A distributed database system consists of a collection of sites, each of which maintains a local database system. Each site is able to process local transactions–those transactions that access data only at that single site. In addition, a site may participate in the execution of global transactions–those transactions that access data at 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 overheads. The primary disadvantage of distributed database systems is 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.

Companies that have moved out of the 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.

4.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) The advantages of DDBMS are:

- The primary advantages of distributed database systems is the ability to share and access data in a reliable and efficient manner. A user at one site may be able to access data that is available at another site.
- Each site is able to retain a degree of control over data stored locally.
- It reflects organisational structure
- If distributed over several locations then improved availability of DBMS. A failure of a communication link making some sites inaccessible does not make the entire system inoperable. The ability of most of the systems to continue to operate despite failure of onesite results in increased availability, which is crucial for database systems used for real-time applications.
- Speed of database access may be better than that achievable from a remote centralised database. Parallel computation allows faster processing of a user's query.
- It costs much less to create a system of smaller computers with lower equivalent a single large computer. It is much easier to expand. New sites can be added to the network without affecting the operations of other sites.

2)

Global Transactions	**Local Transactions**
Involves multiple sites	Can be performed locally
Uses data communication links	Performed locally
Takes longer to execute, in general	Faster
Performance depends on global schema objects available	Performance is governed by local schema objects like local indexes
Just 20-25 % of total	Most transactions are local

Check Your Progress 2

1) Fragmentation cannot be carried out haphazardly. There are three rules that must be followed during fragmentation.

a) Completeness: If a relation R is decomposed into fragments R1, R2,.. Rn, each data item that can be found in R must appear in at least one fragment.

b) Reconstruction: It must be possible to define a relational operational that will reconstruct the relation R from the fragments.

c) Disjointness: If a data item d appears in fragment R1 then it should not appear in any other fragment. Vertical fragmentation is the exception to this rule, where primary key attributes must be repeated to allow reconstruction.

2) If relation R is replicated a copy of relation R is stored in two or more sites. 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 found in another site.

Increased parallelism: In cases where the majority of access to the relation R results in only the reading of the relation, several sites can process queries involving R in parallel.

Increasing overhead on update: The system must ensure that all replicas of a relation R are consistent, 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 overheads.

Check Your Progress 3

1) Advantages of client/server computing are as follows:

- C/S computing caters to low-cost and user-friendly environment.
- 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 data feeders.
- It allows server enforced security, performance, backup, integrity to be a part of the database machine avoiding the requirement to write a large number of redundant piece of code dealing with database field validation and referential integrity, allows multiple users to simultaneously access the same application data.
- Updates from one computer are instantly made available to all computers that had access to the server.
- It can be used to develop highly complex multi-user database applications being handled by any mainframe computer. If the load on database machine grows, the same application can be run on a slightly upgraded machine.

2) **2-Tier client/Server Models**

With two-tier client/server, the user system interface is usually located in the user's desktop environment and the database management services are usually in a server. Processing management is spilt between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers. In 2-tier client/server applications, the business logic is buried inside the user interface on the client or
within the database on the server in the form of stored procedures.
Alternatively, the business logic can be divided between client and server.

3-tier architecture

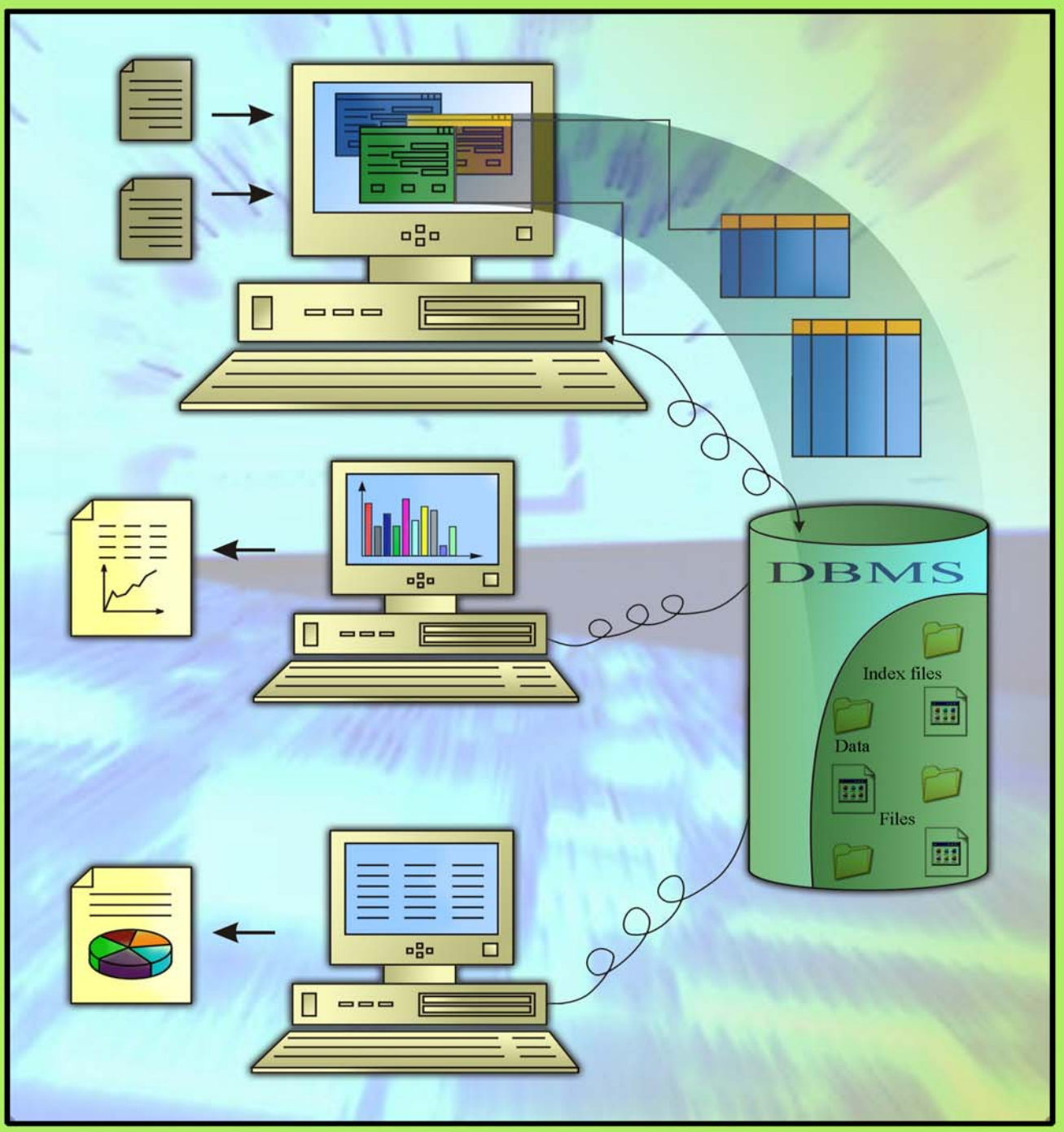
The three-tier architecture (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two-tier architecture. In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. In addition the middle layer adds scheduling and prioritization for work in progress. The three-tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier approach.

In 3-tier client/server application, the business logic resides in the middle tier, separate from the data and user interface. In this way, processes can be managed and deployed separately from the user interface and the database. Also, 3-tier systems can integrate data from multiple sources.

SOCIAS-IGNOU/P.O.10.5T/MAY, 2004

Indira Gandhi National Open University
School of Computer and Information Sciences

MCS - 023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS



CASE STUDY

3

Indira Gandhi
National Open University
School of Computer &
Information Sciences

MCS-023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

Block

3

APPLICATION DEVELOPMENT: DEVELOPMENT OF A HOSPITAL MANAGEMENT SYSTEM (An HMS)

Programme / Course Design Committee

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Prof. M. Balakrishnan, IIT, Delhi
Prof. Harish Karnick, IIT, Kanpur
Prof. C. Pandurangan, IIT, Madras
Dr Om Vikas, Sr. Director,
Ministry of CIT, Delhi
Prof. P. S. Grover, Sr. Consultant,
SOCIS, IGNOU

Faculty of School of Computer and Information Sciences
Shri Shashi Bhushan
Shri Akshay Kumar
Prof. Manohar Lal
Shri V.V. Subrahmanyam
Shri P.Venkata Suresh

Block Preparation Team

Mr. Milind Mahajani (Content Editor)
Software Consultant
New Delhi

Ms. Ranjana Sharma
Ansal Institute of Technology
Gurgaon

Prof. (Retd) A.K.Verma (Language Editor)
New Delhi

Course Coordinator: Shri Akshay Kumar

Block Production Team

Shri T.R. Manoj, Section Officer (Pub.) and Shri H.K. Som, Consultant
CRC prepared by Shri A N Kispotta

Acknowledgement

To Prof. Parvin Sinclair for help in the coordination of this block.

May, 2005

ISBN —

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Director, SOCIS.

DEVELOPMENT OF A HOSPITAL MANAGEMENT SYSTEM (An HMS)

1.0 INTRODUCTION

In the two previous blocks, we have provided the details of the concepts of Relational Database Management. This block provides an example of the use of a Database Management System and user interface development tool to create a Hospital Management System (HMS). The focus in this block is not on developing the Hospital Management System but on providing practical oriented details about how to work with the tools available in MS Access and Visual Basic to create the basic elements of any application that are the tables, forms, reports and queries. Thus, in this block, we will just show the steps about how to create forms, create tables and create reports and queries. You, in turn, have to develop this system by practicing the steps for creating various sample forms, reports, tables and queries.

Please note that although this application in the block is expected to be developed using Microsoft Access and Visual Basic, you may like to use open source tools **MYSQL** and **PHP.**

Each hospital is unique in its requirements and priorities. Hence the HOSPITAL MANAGEMENT SYSTEM (HMS) covers many hospital activities. It also allows hospitals to choose from the various modules to match their specific needs. Please note that this is just a very simple prototype and should not be used for commercial deployment. Also, you must discuss the specification and design errors of this application for better description of the domain of Hospital administration.

This block pays attention to various steps involved in using the tools to evolve the system. Less emphasis is given to analysis and design of the system, which is dealt in more detail in Block 4.

Objectives

After going through this block you should be able to:

- brief the very basic needs for an Information System in a Hospital;
- use the tools needed to develop tables, forms, reports and queries;
- work with various tools to create a sample system, and
- identify the problems encountered during the implementation of an Information System.

2.0 NEED TO DEVELOP THE HOSPITAL MANAGEMENT SYSTEM (An HMS)

Every hospital big or small keeps the records of its patients including the registration details of the patient and the fee payments. The entry of patients is determined whether s/he has arrived in emergency, OPD or for a routine check-up. The patient who gets admitted is provided with a room according to his/her choice. The patient is allotted a doctor according to his illness. The doctor may refer the patient to another doctor with expertise of the illness. On discharge, the patient is required to settle the bills sent by the accounts department of the hospital.

The hospital also keeps the record of the doctors visiting the hospital, plus the permanent employees of the hospital. Each doctor has a few days associated with his/her visit to the hospital and also the timings when s/he is available in the hospital. The employees draw their salary from the accounts department.

The hospital maintains the record of the inventory of the hospital including the equipment and the medicines, blood bank, etc. A limit for inventory is maintained for every item. When the limit is reached, an order for the purchase of the concerned item is placed. The database is updated after the supplier supplies the product.

2.1 Advantages of Hospital Management System

HMS provides manifold benefits to hospitals whether large or medium sized.
Streamlined Operations:

- Minimized documentation and no duplication of records.
- Reduced paper work.

Improved Patient Care:

- Procedures for timely and effective patient care
- Faster information flow between various departments
- Easy access to reference records.

Better Administration Control:

- Availability of timely and accurate information
- Access to updated management information.

Smart Revenue Management:

- Optimized bed occupancy checks
- Effective billing of various services
- Exact stock information.

2.2 ER Diagram of HMS

The following is a very simplified ER Diagram of a much mutated toned down version of HMS that does not fulfill all the requirements above. You must enhance it once you are through with the system.

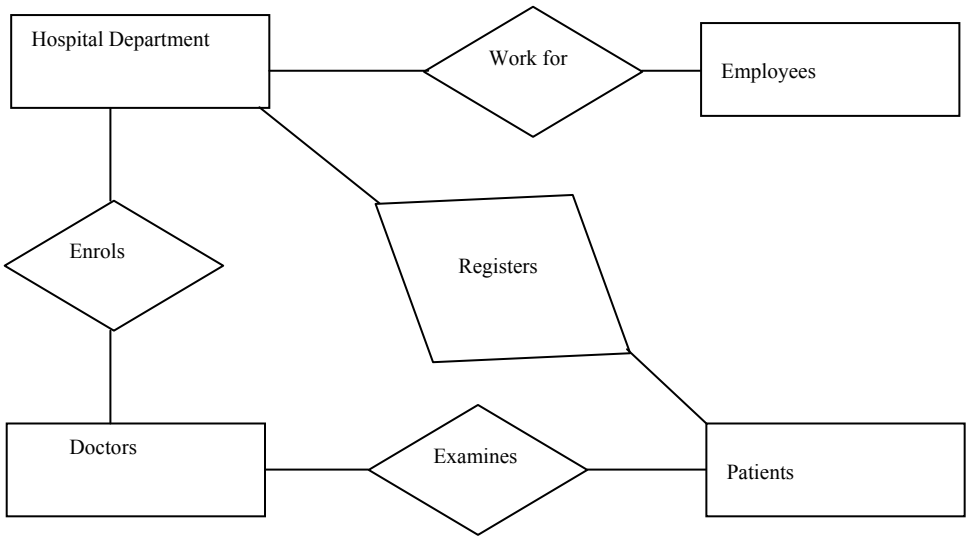


Figure 1: Entity relationship diagram for HMS

2.3 The Software Development tools required

We have selected the combination of MS-Access and VISUAL Basic although you can also use MS-Access forms instead of Visual Basic. The selection of Visual Basic is governed by the reason that this is one of the most simple front end tools. As this application is the first application in any database management system, it may be a good starting point to learn a bit about it. However, you may like to use MySQL and PHP also, a very popular combination of tools for such type of system development. Let us now do some basic development.

Exercise 1: Find out the requirements that can be met considering the Data Model that is being reflected by the ER diagram.

3.0 CREATING A DATABASE FOR HMS

The database has to be created for the Hospital Management System. This is done by creating tables and relations in MS-Access. A Database is a collection of related data describing one or more related activities. A database Management System is a collection of programs that enable the user to create and maintain the database. The following relations are created in MS-Access.

Docdetails – Gives the details of the doctors affiliated to the Hospital.

Patient Registration – Provides the details of a patient as well as arrival status such as OPD or emergency and calculates charges incurred. If a patient is already registered, updation of previous records is made.

Employee Details – This is used to provide the details of employees working in the hospital irrespective of the designation and the department.

Department Details – This provides the details of any department in the hospital including the department heads.

Exercise 2 : Identify the limitations of such a table design. Propose a modified table design for the above.

3.1 Creating a database using MS-Access

1. Open MS-Access
2. Dialog Box – Create a new database using
   Blank database
   Access database wizards, pages and projects
3. Select Create a new database using blank database
4. db1 – create

Now the database db1 has been created. The following screen will appear. You can now create tables, forms, reports, queries etc. in the database db1.

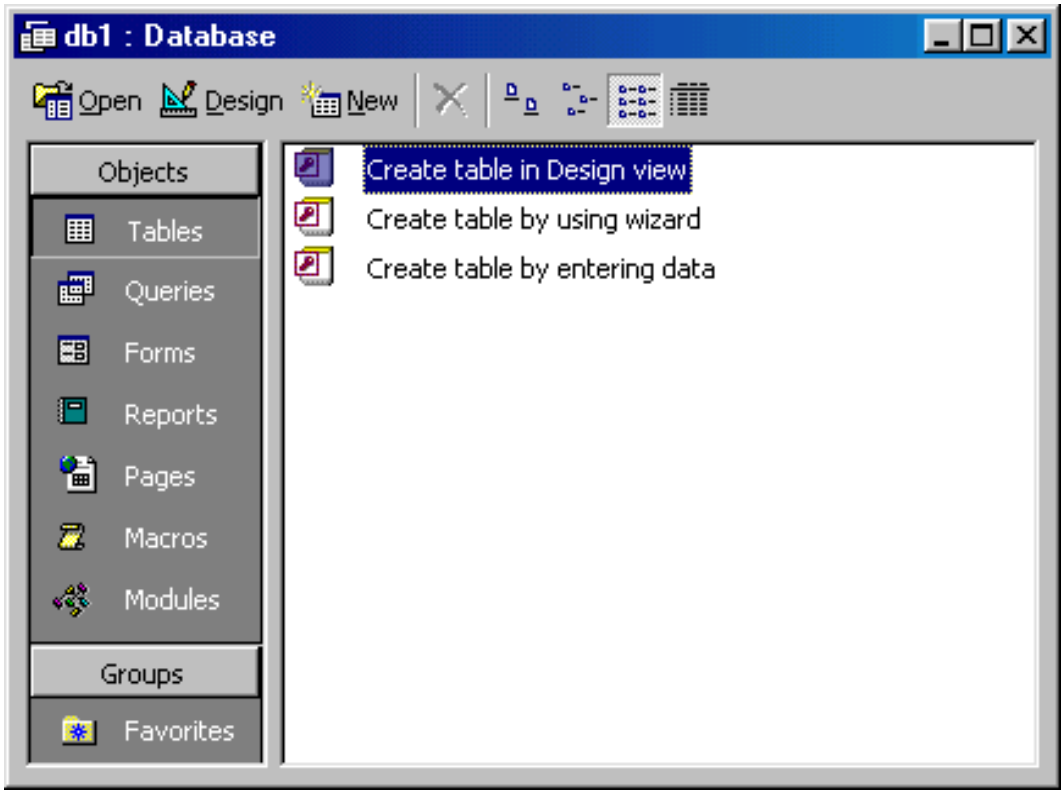


Figure 2: Creating database tables using MS-Access

- **Create table in Design view** will allow you to create the fields of the table. This is the most common way of creating a table and is explained in detail below :
- **Create table using wizard** will step you through the creation of a table.
- **Create table by entering data** will give you a blank datasheet with unlabelled columns that looks much like an Excel worksheet. Enter data into the cells and click the **Save** button. You will be prompted to add a primary key field. After the table is saved, the empty cells of the datasheet are trimmed. The fields are given generic names. Hence create a table “Docdetails” with fields such as “name”, “address”, “qualifications” etc. To rename them with more descriptive titles that reflect the content of the fields, select **Format Rename Column** from the menu bar or highlight the column, right-click on it with the mouse, and select **Rename Column** from the shortcut menu.

Create a Table in Design View

Design View will allow you to define the fields in the table before adding any data to the datasheet. The window is divided into two parts: a top pane for entering the field name, data type, and an option description of the field, and a bottom pane for specifying field properties (*Figure 2*).

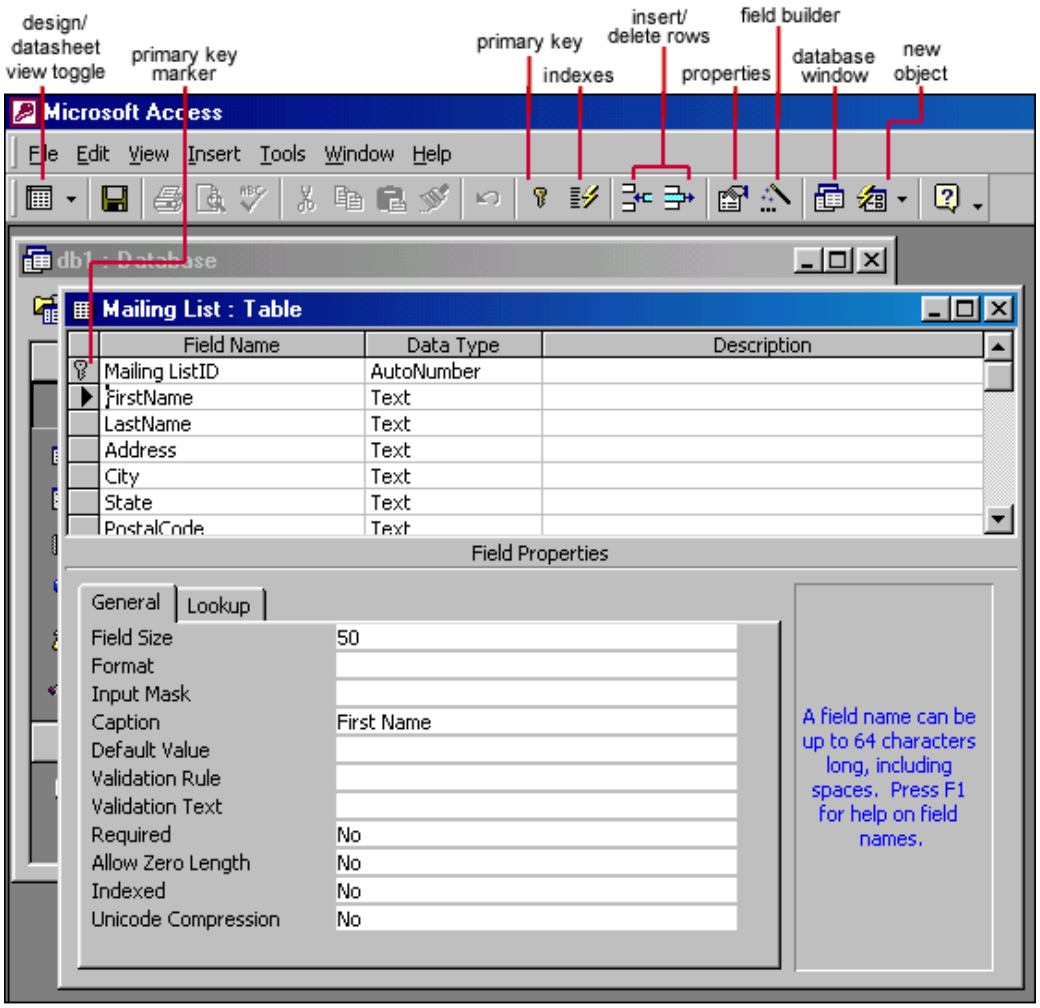


Figure 3: A sample table creation screen

- **Field Name:** This is the name of the field and should represent the contents of the field such as "Name", "Address", "Qualifications", etc. The name cannot exceed 64 characters in length and may include spaces.
- **Data Type** is the type of value that will be entered into the fields.

Primary Key

Every record in a table must have a primary key that differentiates it from every other record in the table. In some cases, it is only necessary to designate an existing field as the primary key if you are certain that every record in the table will have a different value for that particular field. An enrolment number is an example of a field that can be used as primary key to table student.

Designate the primary key field by first selecting them (you may select a composite primary key also) and then by right clicking on it and selecting **Primary Key** from the shortcut menu or select **Edit | Primary Key** from the menu bar. The primary key field will be noted with a key image to the left. To remove a primary key, repeat one of these steps.

Hence in HMS the table Docdetails can be created using the above method. The following tables should be created by you.

DocDetails

DocCode	Text	PK
DeptCode	Text	FK
DocName	Text	
Qualifications	Text	
Designation	Text	
Address	Text	

Patients' Registration

Patient Regn No.	Text	PK
Date of Regn.	Text	
Name	Text	
Father's/husband's name	Text	
Age	Number	
Address	Text	
DocCode	Text	FK
Arrived in	Text	
Charges	Number	

Employee Details

EmpCode	Text	PK
Ename	Text	
Address	Text	
Gender	Text	
Date of Joining	Date/Time	
Salary	Number	
DeptCode	Text	FK

DeptDetails

DeptCode	Text	PK
DeptName	Text	
HOD	Text	
HOD Code	Text	

If none of the existing fields in the table will produce unique values for every record, a separate field must be added. Access will prompt you to create this type of field at the beginning of the table the first time you save the table and a primary key field has not been assigned. The field is named "ID" and the data type is "autonumber". Since this extra field serves no purpose to you as the user, the autonumber type automatically updates whenever a record is added so there is no extra work on your part.

3.2 Table Relationships

To prevent the duplication of information in a database by repeating fields in more than one table, table relationships can be established to link fields of tables together. Follow the steps below to set up a relational database:

- Click the **Relationships** button on the toolbar. 
- From the **Show Table** window (click the **Show Table** button on the toolbar to make it appear), double click on the names of the tables you would like to include in the relationships. When you have finished adding tables, click **Close**.

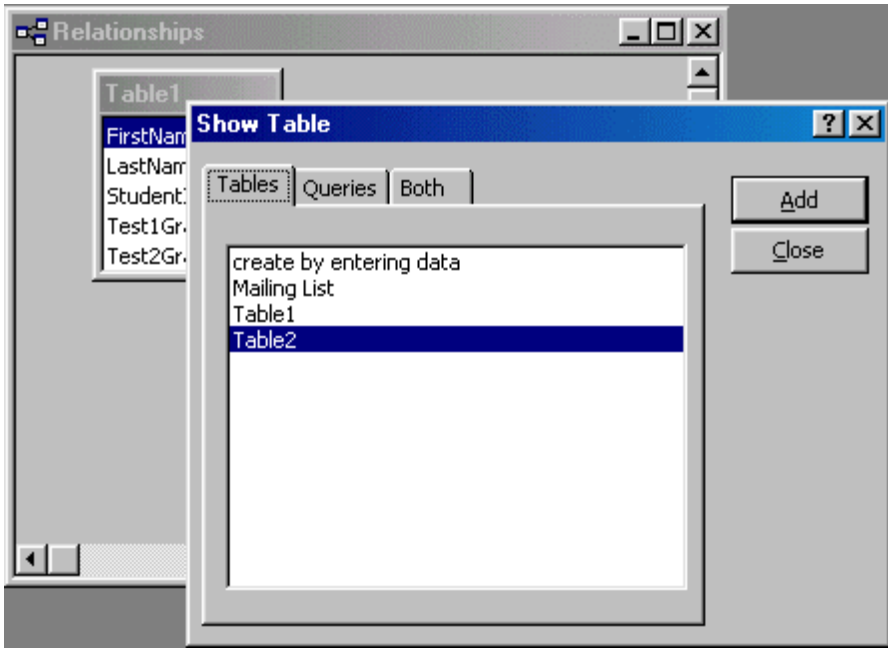


Figure 4: A sample screen for relationships

- To link fields in two different tables, click and drag a field from one table to the corresponding field on the other table and release the mouse button. The **Edit Relationships** window will appear. From this window, select different fields if necessary and select an option from Enforce Referential Integrity if necessary. These options give Access permission to automatically make changes to referential tables if key records in one of the tables is deleted. Check the **Enforce Referential Integrity** box to ensure that the relationships are valid and that the data is not accidentally deleted when data is added, edited, or deleted. Click **Create** to create the link.

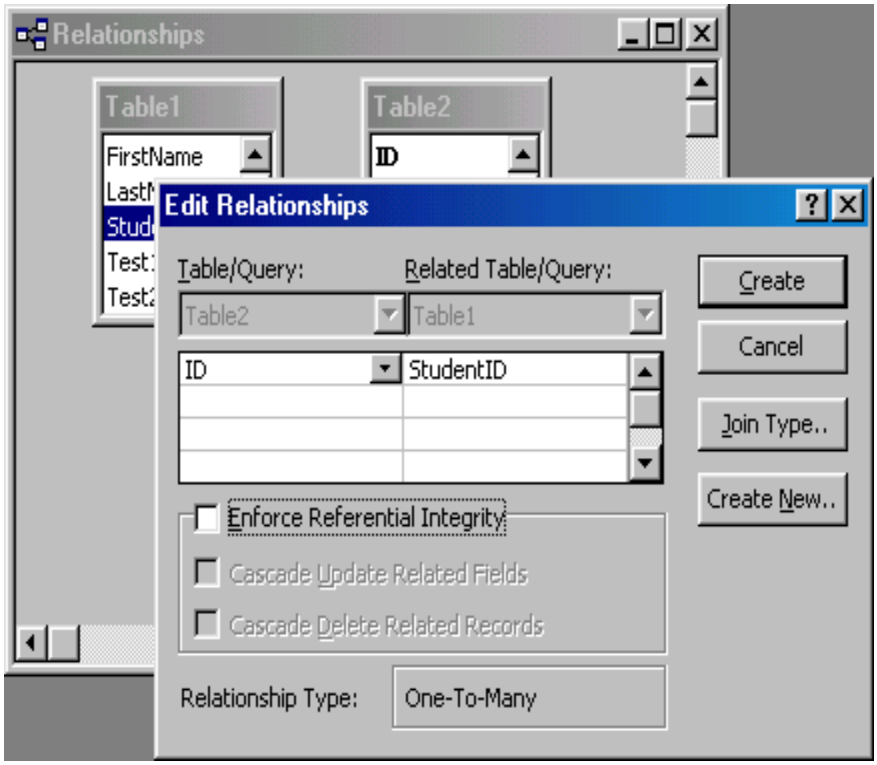


Figure 5: Editing a sample relationship

- A line now connects the two fields in the Relationships window.

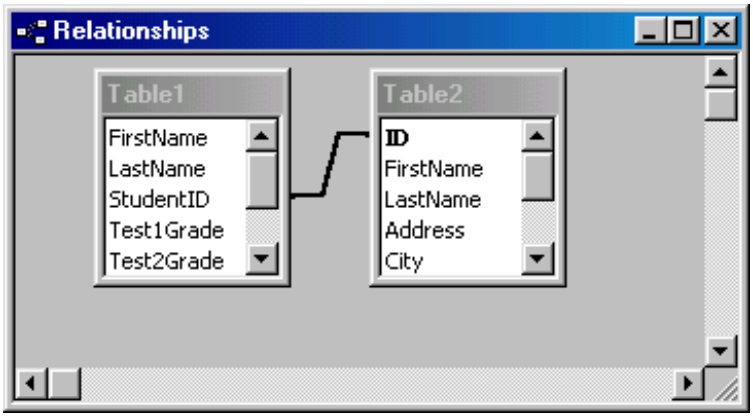


Figure 6: A sample relationship window

- The datasheet of a relational table will provide expand and collapse indicators to view subdatasheets containing matching information from the other table. In the example, the student address database and student grade database are related.

Hence the relationship for the above is created using the process above. Now, use the tables created by you about HMS and develop the following relationships.

DocDetails

DocCode	Text	PK
DeptCode	Text	FK
DocName	Text	
Qualifications	Text	
Designation	Text	
Address	Text	

Patients' Registration

Patient Regn No.	Text	PK
Date of Regn.	Text	
Name	Text	
Father's/husband's name	Text	
Age	Number	
Address	Text	
DocCode	Text	FK
Arrived in	Text	
Charges	Number	

Employee Details

EmpCode	Text	PK
Ename	Text	
Address	Text	
Gender	Text	
Date of Joining	Date/Time	
Salary	Number	
DeptCode	Text	FK

DeptDetails

DeptCode	Text	PK
DeptName	Text	
HOD	Text	
HOD Code	Text	

4.0 DEVELOPING FRONT END FORMS

Forms can be developed using Microsoft Access form wizard and interfaces. But we have selected Visual Basic here as it is one of the useful tools for form design. You are however free to develop these forms using any other platform. We are providing a very brief account of Visual Basic (VB) here, for more details you may refer for help to VB or further readings.

Using Visual Basic -A brief description

VISUAL BASIC is a high level programming language evolved from the earlier DOS version called BASIC. BASIC means **B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. It is a fairly easy programming language to learn. The codes look a bit like English Language. Different software companies produced different versions of BASIC, such as Microsoft QBASIC, QUICKBASIC, GWBASIC, IBM BASIC and so on.

VISUAL BASIC is a VISUAL and events driven Programming Language. These are the main divergences from the old BASIC. In BASIC, programming is done in a text-only environment and the program is executed sequentially. In VISUAL BASIC, programming is done in a graphical environment. Because users may click on a certain object randomly, so each object has to be programmed independently to be able to respond to those actions (events). Therefore, a VISUAL BASIC Program is made up of many subprograms, each has its own program codes, and each can be executed independently and at the same time each can be linked together in one way or another.

The Visual Basic Environment

On start up, Visual Basic 6.0 will display the following dialog box as shown in *Figure7*. Choose to start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that makes up an application. There are various types of applications we could create. However, we shall concentrate on creating Standard EXE programs (EXE means executable program).

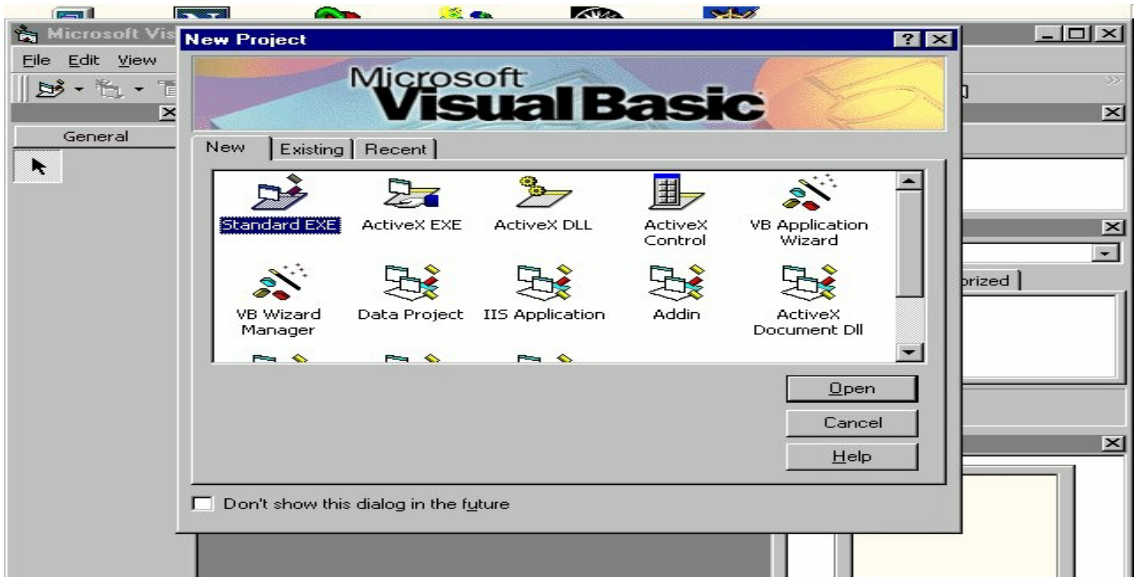


Figure 7: The visual basic start-up dialog box

Now, click on the standard EXE icon to go into the actual VB programming environment.

In *Figure 7*, the Visual Basic Environment consists of –

- The **Blank Form** window which you can design your application's interface.
- The **Project** window displays the files that are created in the application.
- The **Properties** window which displays the properties of various controls and objects that are created in your applications.

It also includes a **Toolbox** that consists of all the controls essential for developing a VB Application. Controls are tools such as boxes, buttons, labels and other objects drawn on a form to get **input** or **display output**. They also add visual appeal.

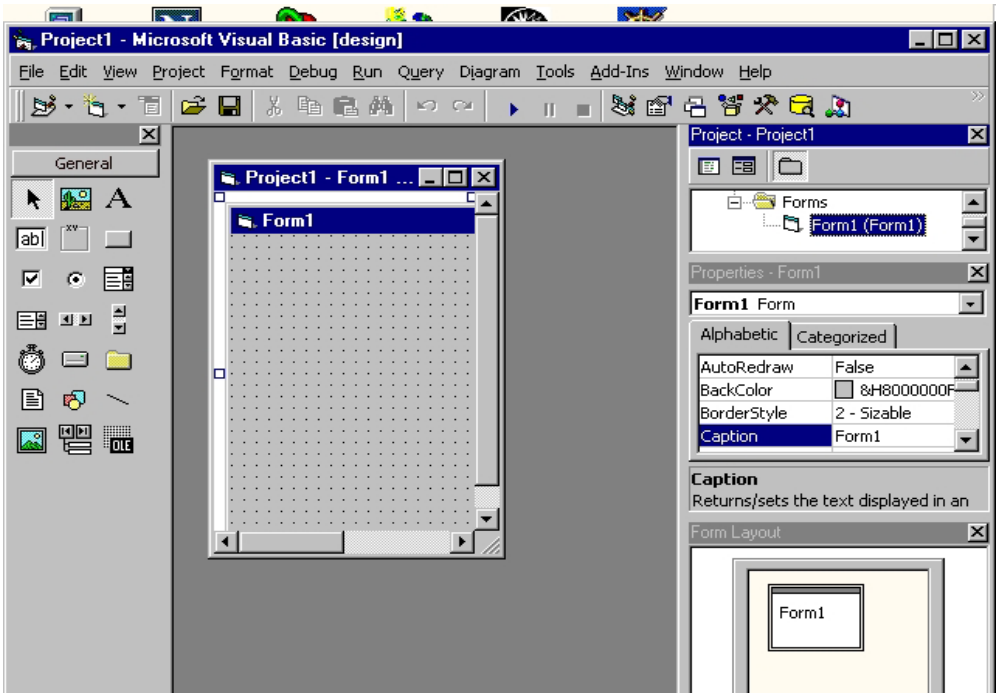


Figure 8: The visual basic environment

To the left of the screen there are a lot of little icons under the heading 'General'. These are called components - ActiveX controls. Basically, these can be added to your program (technical description, that) - such as a button or picture. Let's see one in action.

- Double-click on the component represented by a black 'A'

An item called 'Label1' should appear in the centre of Form1.

To the bottom-right of the screen there is a box entitled "Properties - Label1". If it doesn't say 'Label1', click on the label in the middle of your form.

These are the "properties" of the label. They tell it how to work - what text to display, what colour it should be, whether it should have a border, and so on. These can be easily changed by typing in or selecting new values from the list.

- Click on the box to the right of the 'Caption' property.
- Replace the existing "Label1" value with "My First Program".
- Press Enter.

To move the label –

- Click on the label and hold your mouse down.
- Drag your mouse to the upper left hand corner of "Form1" and release the mouse button.

The screen looks something like this right now –

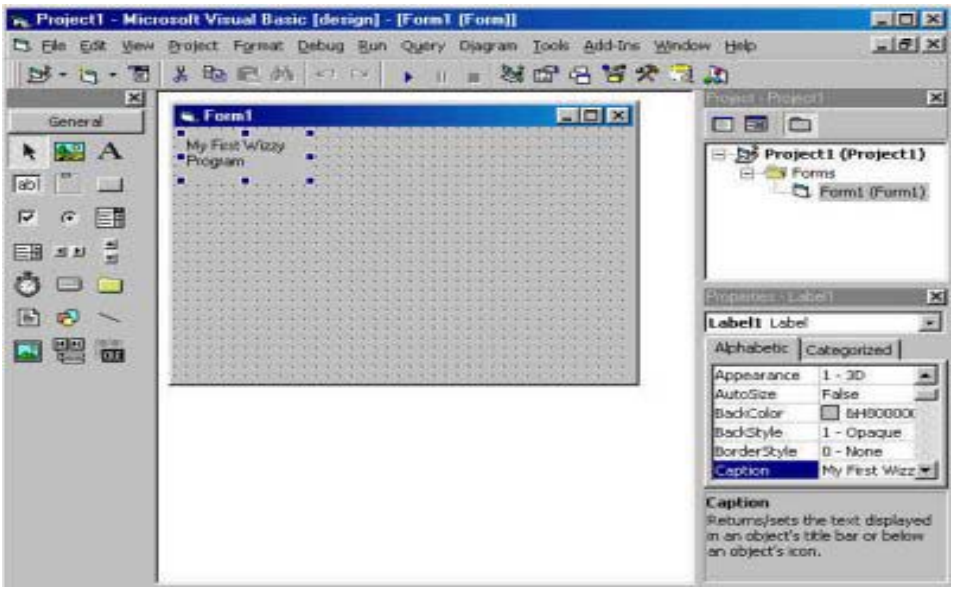


Figure 9: The text label

Double click on the  button in the General toolbar. This is a Command Button – you'll find them in virtually all Windows programs, usually in the form of "OK", "Apply" or "Cancel" buttons.

To change the property of Command Button –

- Change its Caption property to "Click Me!"
- Move down in the Properties list and change the Font.
- Resize the Command Button using the eight blue squares surrounding it.

The Form looks like this right now –

Figure 10: A sample form

Finally, let us add our first line of code: For this double click on the Command Button.

The screen should look like this...

Private Sub Command1_Click()

<Cursor flashing here>

End Sub

Now type in.

MsgBox “Welcome to my First program!”

The “Private Sub Command1_Click ()” tells Visual Basic to run this line of code when someone **clicks** on the Command Button called **Command1**. When they do, the line of code runs - telling Visual Basic to display the designated sentence in a message box (**MsgBox**). The “End Sub” merely tells VB to stop running the code.

Control: A control is just an item that can be added to the Form.

We could then change its properties - perhaps alter its Caption to “Click Me You Annoying Person!” or its Font property to Arial, Bold, Size 18. Open Visual Basic and create a new Standard Exe. To do that:

- Click on the Standard Exe icon.
- Click on the Open button

To the left of your screen should be the control toolbox. Right now it’s displaying a standard set of Visual Basic controls.

 This is the Check Box control. Such boxes are checked and unchecked to set various options. Functionality can be added to the applications. The Check Box properties can be changed in the properties window. For instance, alter its Caption, Forecolor or Font property with a few clicks of the mouse! Some of the properties are hardly ever used or only required in very specific circumstances.

Using more built in VB controls

Image -  - This control allows us to display a picture on the Form.

Label -  - This control is often used to display information to the user, such as an explanation of what they should do.

TextBox -  - The TextBox control is a little like McDonalds; common, usually trouble-free and definitely overused. You often utilise this control to collect information from the user or display data from a database.

CommandButton -  - This is a pretty boring yet very common control. It simply creates a button the user may click to run a slot of code.

DriveListBox -  - Whilst this control doesn’t have many properties you can fiddle with, it’s great to observe. You’ve seen this famous drop-down list many times before and now it can be incorporated into your own cool program!

Shape -  - This control draws a shape on your Form.

To run an application:

-  Press F5 (or click the button)

Every single control you add to a Form has a Name property – it should appear at the very top of the Properties window. Every time a control is added, an automatic name is assigned – such as Command1. It can be renamed to give it a meaningful name.

The Visual Basic community also has their own "naming conventions" where, for instance, Text Box names are preceded by "txt". Therefore if a Text Box is added that will hold a date, it is advisable to change its name to "txtDate". Other prefixes include:

ComboBox - cbo (for example, cboProductCodes)

CommandButton - cmd (e.g., cmdOK)

Form - frm (e.g., frmOrders)

CheckBox - chk (eg chkExtraCheese)

OptionButton - opt (e.g., optPlanTwo)

It is advisable to keep names pretty unique.

Code Writing

Writing creative code is easy! Firstly, programmers often refer to "design-time" and "runtime". Design-time is the period when designing of Form or writing code is done – basically any time your program isn't running. And runtime is exactly the opposite – the time when your program *is* running.

The properties of controls can be changed whilst you are in design-time. The properties can also be whilst the program is running. For instance, perhaps you want the Caption of your Form to change depending on which button the user clicks.

- Open Visual Basic and create a Standard Exe
- Change the Name of Form1 to "frmTest" and the Caption to "My Test App!"
- Add two Command Buttons and change the Caption of the first to "Hayley" and the second to "Roy".

Any control can be added to the Form, including Command Buttons. You can easily change the properties of such controls by altering values in the properties window.

How can variables be used in your Visual Basic applications?

A variable can be used in VB. This is explained with the help of the following example.

Example: In order to display a number and increment it, every time the user clicks a Command Button variables can be used.

- Launch Visual Basic and create a new Standard Exe.
- Add a Command Button to your Form.
- Change the button's Name property to "cmdVariableCount" and its Caption property to "Click Me Quick!"
- Double click on a blank area of the Form to open the code window.

Near the top of the code window, there are two drop–down boxes. The first specifies an *object*, the second a *procedure*. In English, these are defined as:

Object the thing you want to work with, say cmdVariableCount.

Procedure denotes the 'event' for which the code is written. For instance, we may write a code in the Click event to have it respond when the user clicks on a cmdVariableCount. The Procedure drop-down can also display user-defined Subs and Functions.

Click on the object drop-down box, the first one.

You should see a list of objects – probably Form, cmdVariableCount and (General). The first two are objects – and you can add code to respond to their events. Select the (General) section from the drop-down list.

The right-hand combo box should change to (Declarations).

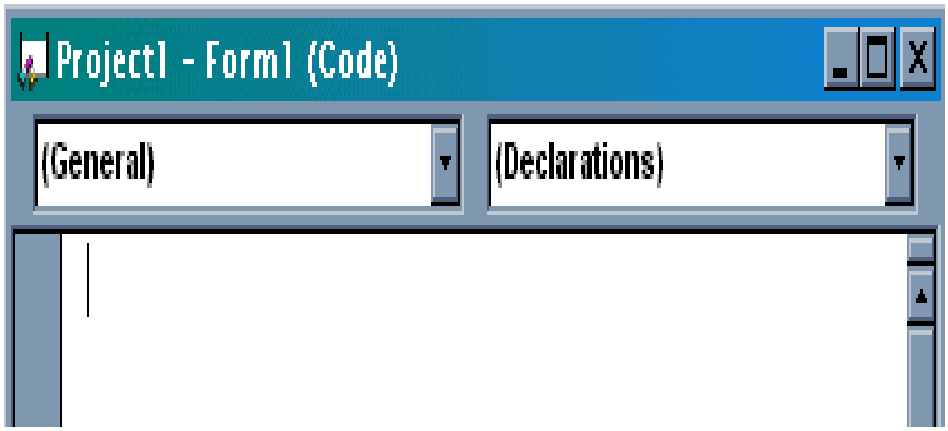


Figure 11: The declarations

You will notice that your cursor is not surrounded by any code as with usual event procedures, such as "Public Sub cmdVariableCount_Click()". That is because you have reached a special place in your code window–the top–where you can create, or "declare" variables.

To declare a variable you use the syntax:

Dim MyVariable as DataType

...where MyVariable is the name of your variable and DataType is the type of information it should hold. Here is a short list of the most common DataType values:

String - If you are going to store text in the variable.

Integer - Used when storing a number.

Boolean - Used when storing a True/False value.

Date - Used when story a date value.

Decoding declarations...

A variable is just something in which we can store a certain type of information.

Loops

This Loop can be used to display a message box for every single doctor in HMS. Two ways can be used – either manually code every single message box or use a loop that says, in English, "display a message box for every Doc".

Creating a loop is simple. Let us look at a sample piece of code.

For i = 1 to 5

```
Msgbox  "I have looped round  " & i & " times"

Next i
```

The first line tells Visual Basic how many times it should loop around. It is saying – loop around this bit of code five times and store the number of times it has looped around in "i". The second line is just a simple message box that shows "i" – i.e., just the number of times it has looped around. The third line, "Next I" just tells it to go back to the beginning until it is gone from one to five.

Do Until

"Do Until" loop is used when you may want to keep asking a user for a student name until they type in "END" or something.

```
Dim Ename as String

Do Until Ename = "END"

Ename = InputBox ("Enter the Employee name, or type END to finish:")

Msgbox "New employee is entered - " & Ename

Loop
```

The first line merely declares a variable called "Ename" to hold a string (that is, plain text). The second line tells Visual Basic to loop around until the "Ename" variable is equal to "END".

The third line introduces another function called the InputBox and is used to get information from the user. On running this, a little box will appear for the user to enter some information–in this case, an Employee name. Whatever they enter will be assigned to "Ename".

And the fourth line just says – loop back to the start again. And at the start, Visual Basic asks the question once more – does "Ename" equal "END"? If it does, then looping stops. If not – looping goes on!

Subs

What are subs, and how can they help VB coders simplify their source?

A sub is basically a piece of code separate from the rest. It is often used when you have a piece of common code that you do not want to rewrite each time you use it.

You can get around these using subs. You can create a singular, generic chunk of code that displays the message and then ends your program.

The ability to create such subs can be a real time saver! It avoids unnecessary code and therefore cuts down on the size of your final program. It also saves on boring maintenance time.

Exercise 3: Create forms for HMS. With this basic information you should try to work with some of the components of Visual Basic by adding various components and create the necessary forms. If you are unable to do so then you may either use MS-Access forms or read through further readings. The forms for HMS that should be designed by you can include:

- Form for data entry information of Doctors, patients, employees and departments.
- Forms for display of detailed information.
- Forms for editing this information.
- Any other form that may be required for the modified HMS by you.

5.0 REPORTS

A report is an effective way to present data in a printed format. Because you have control over the size and appearance of everything on a report, the information can be displayed the way you want to see it. We are demonstrating report creation through MS-Access report facility.

5.1 Creating a Report using the Wizard

As with the Form, a report can be created using the Wizard, unless you are an advanced user.

1. Switch to the Database Window. This can also be done by pressing F11 on the keyboard.
2. Click on the **Reports** button under **Objects** on the left side of screen.
3. Double click on **Create Report Using Wizard.**
4. On the next screen select the fields you want to view on your form. All of them could be selected.
5. Click Next
6. Select if you would like to group your files. Keep repeating this step for as many groupings as you would like.
7. Click Next
8. Select the layout and the paper orientation desired
9. Click Next
10. Select the style desired.
11. Click Next
12. Give a name to the report, and select **Preview the Report.**
13. Select **Finish**
14. The report is displayed. To adjust the design of report, press the design button and adjust the report accordingly.

5.2 Creating Mail Merge Labels using a Wizard

Mailing Labels can be created using Microsoft Access for the database. To do this:

1. Switch to the Database Window. This can be done by pressing F11 on the keyboard.
2. Click on the **Reports** button under **Objects** on the left side of screen.
3. Click on **New.**

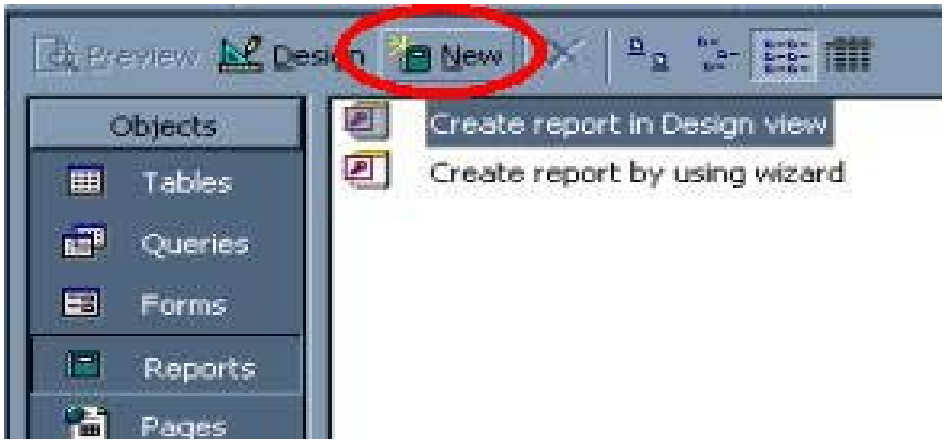


Figure 12: Creating reports

Select **Label Wizard** and the table from which the information is taken.

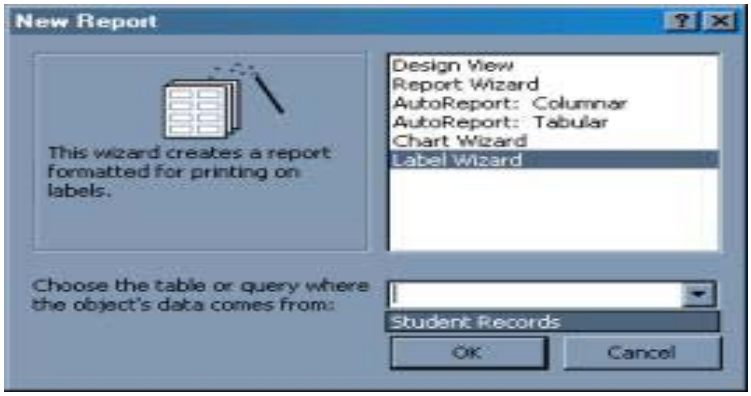


Figure 13 : Creating report using label wizard

Click OK
Select the layout of the labels
Click Next
Select the font size and required color on each label
Click Next
Select the layout of the label
Click Next
Select how you want your labels sorted
Name the label report and preview it

5.3 Creating a Report using the Access Report Wizard

Microsoft Access provides many built-in wizards which allow one to create database objects, such as reports, quickly and easily. The following steps demonstrate the process of creating a very simple report.

1. **Choose the Reports Menu:** Once a database is opened, the main database menu as shown below is displayed. Click on the "Reports" selection and a list of the various reports Microsoft included in the sample database is displayed. Double-click on a few of these and get a feel for what reports look like and the various types of information that they contain.

2. **Create a new Report:** Click on the "New" button and the process of creating a report from scratch will start.

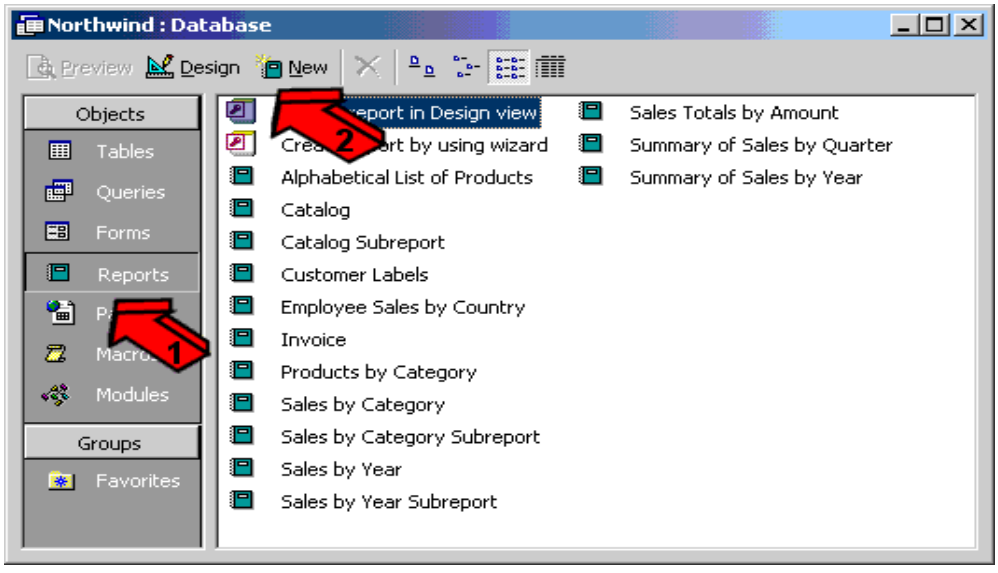


Figure 14: Creating a new report

3. **Select the Report Wizard:** The next screen that appears will require the selection of the method to be used to create the report. Select the Report Wizard, which will help through the creation process step-by-step.

4. **Choose a Table or query:** Choose the source of data for the report. If the information is to be retrieved from a single table, select it from the drop-down box below. Alternatively, for more complex reports, the report can be based on the output of a query previously designed. For example, all of the data we need is contained within the Employees table, so choose this table and click on OK.

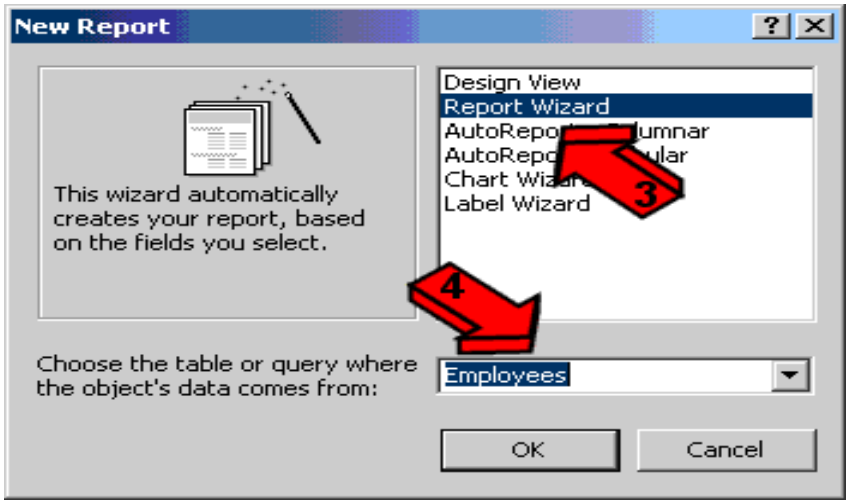


Figure 15: Selecting creation method

5. **Select the fields to include:** Use the '>' button to move over the desired fields. Note that the order in which the fields are placed in the right column determines the default order they will appear in the report. For creating a simple report on home phones of Hospital Employees, even the information contained in it is kept simple -- the first and last name of each employee, the title and the home telephone number. Select the fields and click the Next button.

5. **Select the grouping levels:** At this stage, you can select one or more grouping levels to refine the order in which our report data is presented. For example, we may wish to list home phone numbers of the employees by department so that all of the members of each department are listed separately. Click on the Next button to bypass this step and return here later.

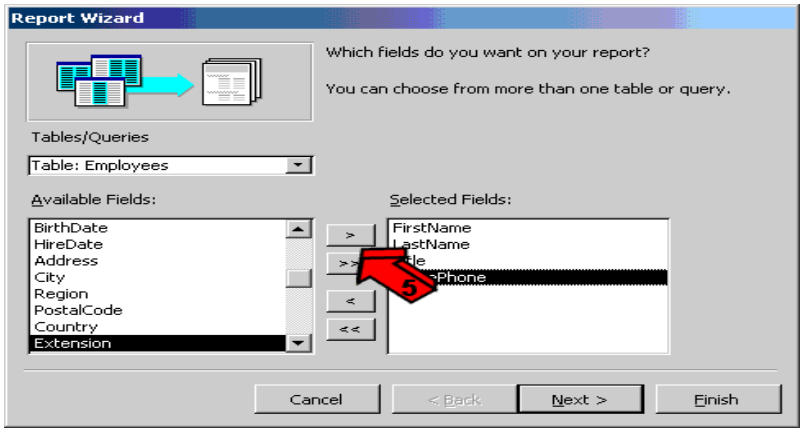


Figure 16: Selecting the fields to include

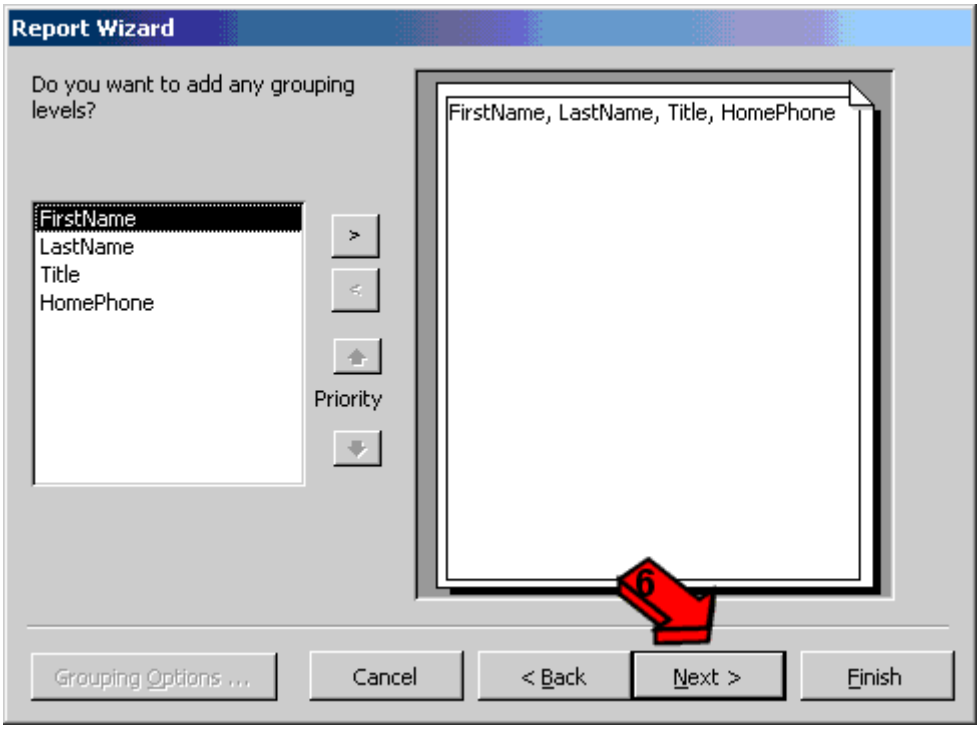


Figure 17: Choosing the grouping levels

7. **Choose the sorting options:** In order to make reports useful, results can be sorted by one or more attributes. A list of Doctors or employees can be sorted by the last name of each employee. Select this attribute from the first drop-down box and then click the Next button to continue.

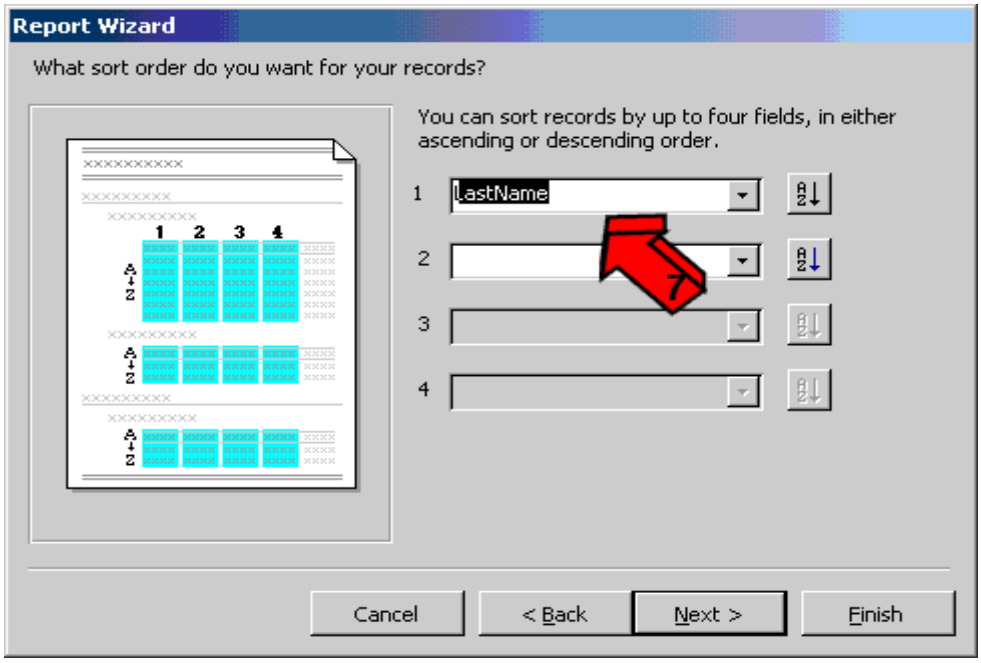


Figure 18: Choosing the sorting options

8. **Choose the formatting options:** In the next screen, some formatting options are given. The default tabular layout can be accepted but can be changed according to needs. Click the Next button to continue.

9. **Select a report style:** The next screen is to select a style for the report. Click on the various options and a preview of the report in that style can be seen in the left portion of the screen. Select the option and then click the Next button to go to the next screen as given below.

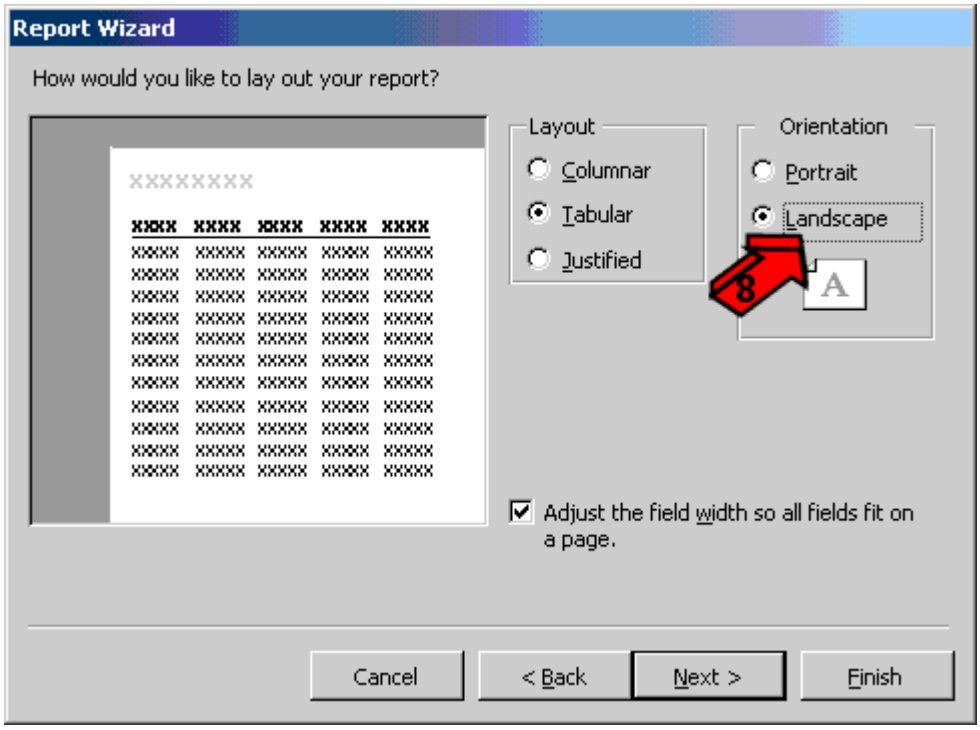


Figure 19: Choosing formatting options

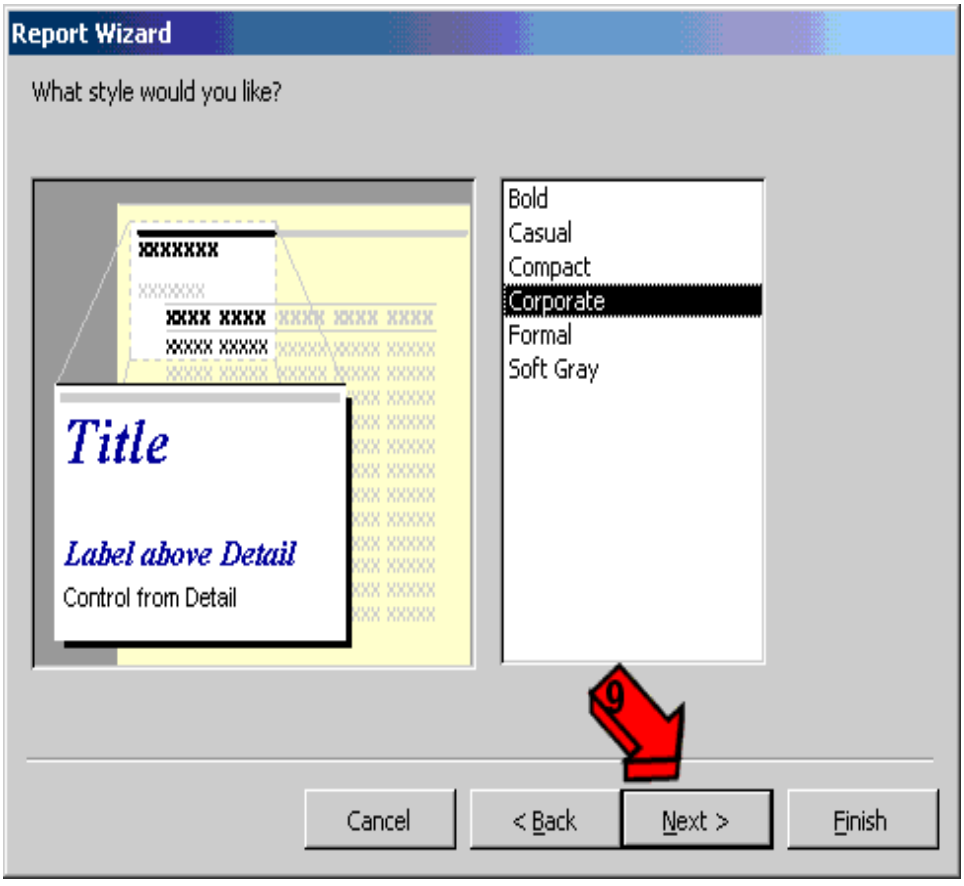


Figure 20: Selecting a report style

10. **Add the title:** Finally, a title is to be given to the report. Access will automatically provide a nicely formatted title at the top of the screen, with the appearance shown in the report style selected during the previous step. Enter a title for the report, select the "Preview the report" option and then click on Finish to see the report.

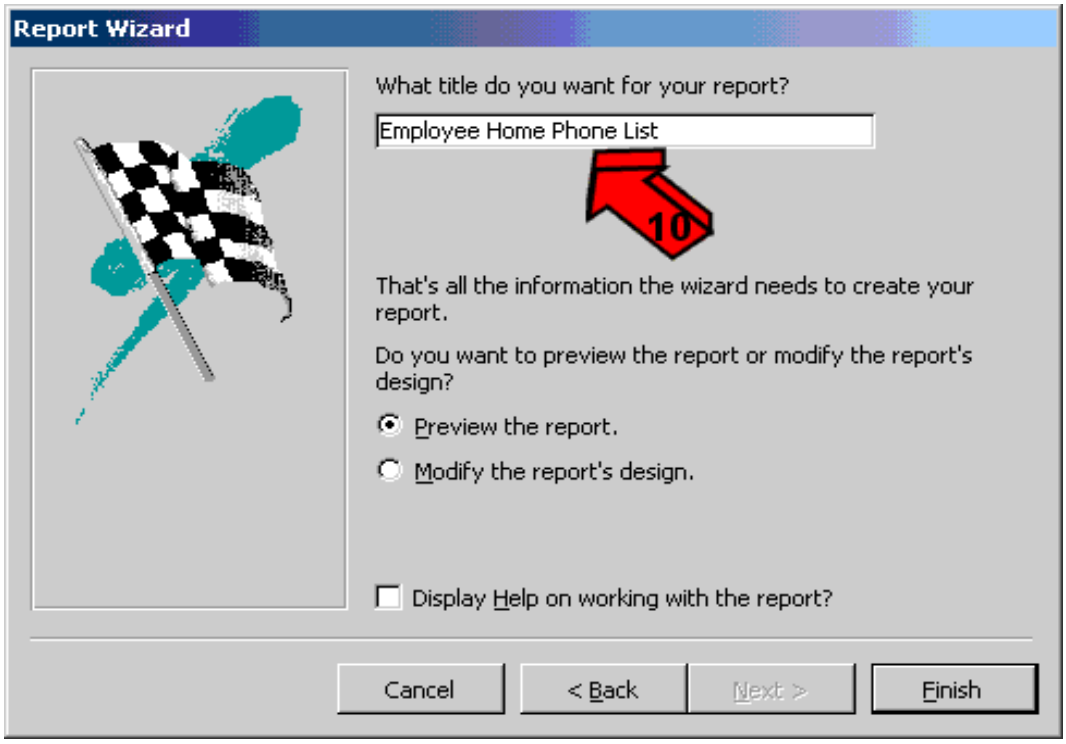


Figure 21: Adding a title

Employee Home Phone List

Last Name	First Name	Title	Home Phone
Buchanan	Steven	Sales Manager	(71) 555-4848
Callahan	Laura	Inside Sales Coordinator	(206) 555-1189
Davolio	Nancy	Sales Representative	(206) 555-9857
Dodsworth	Anne	Sales Representative	(71) 555-4444
Fuller	Andrew	Vice President, Sales	(206) 555-9482
King	Robert	Sales Representative	(71) 555-5598
Leverling	Janet	Sales Representative	(206) 555-3412
Peacock	Margaret	Sales Representative	(206) 555-8122
Suyama	Michael	Sales Representative	(71) 555-7773

Figure 22: A sample report

The final report should appear similar to the one presented above but having the data you might input.

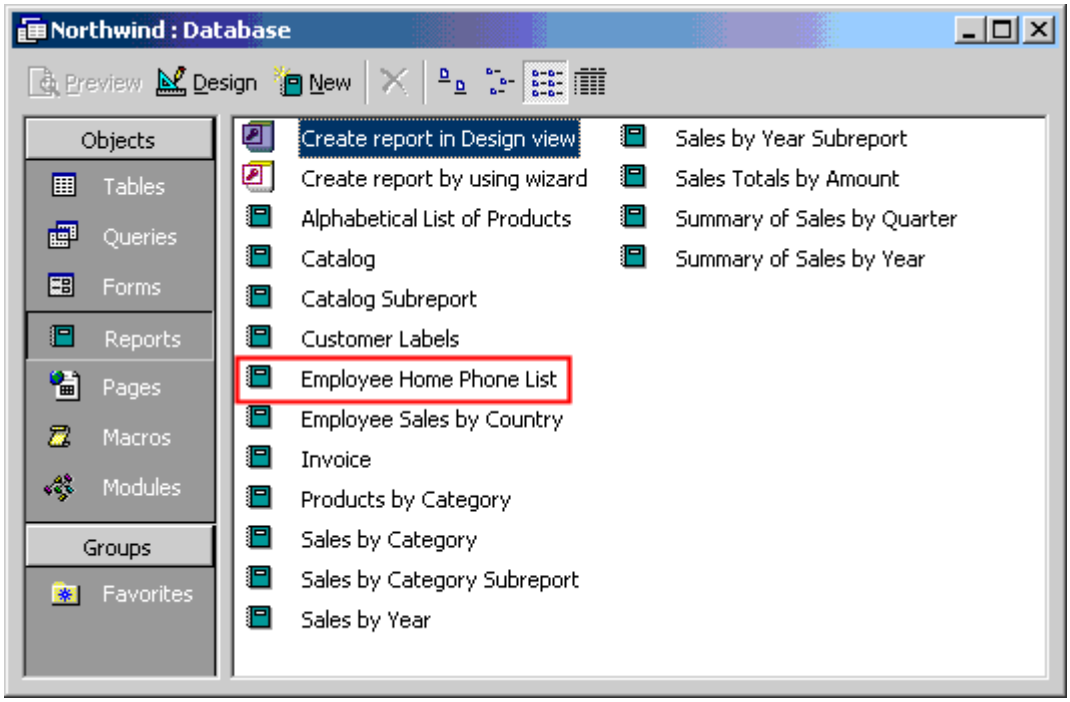


Figure 23: Updated reports menu

5.4 Creating Report using Design view

Access contains wizards that make creating reports very easy. However, sometimes wizards do not create the report the way you want it, or you may need to make modifications. With this in mind, it is helpful to know a little bit about creating a report from scratch.

1. Click on Reports, New, Design View. Click on the list arrow and choose the First Period query. Click on OK.

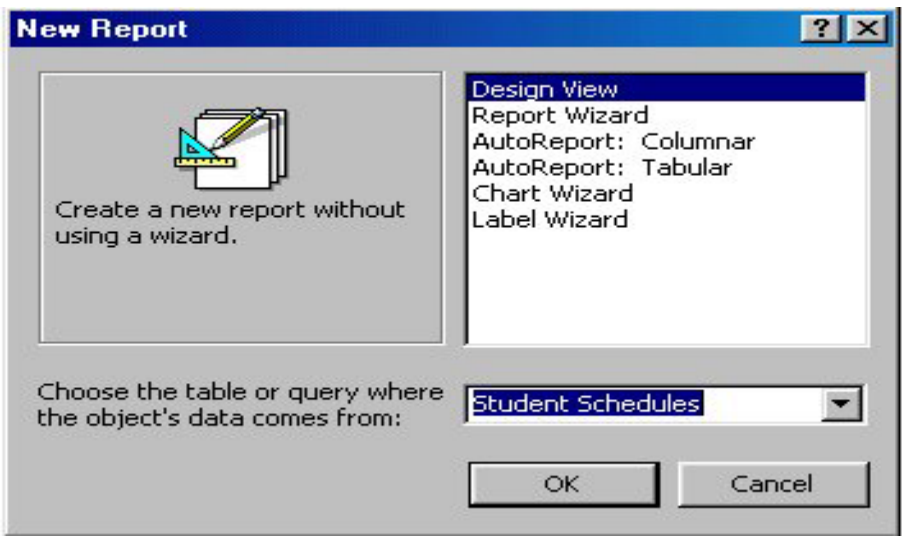


Figure 24: An initial screen of design view

2. The report will open in Design view. If the Toolbox does not appear, click on the Toolbox icon (wrench and hammer).

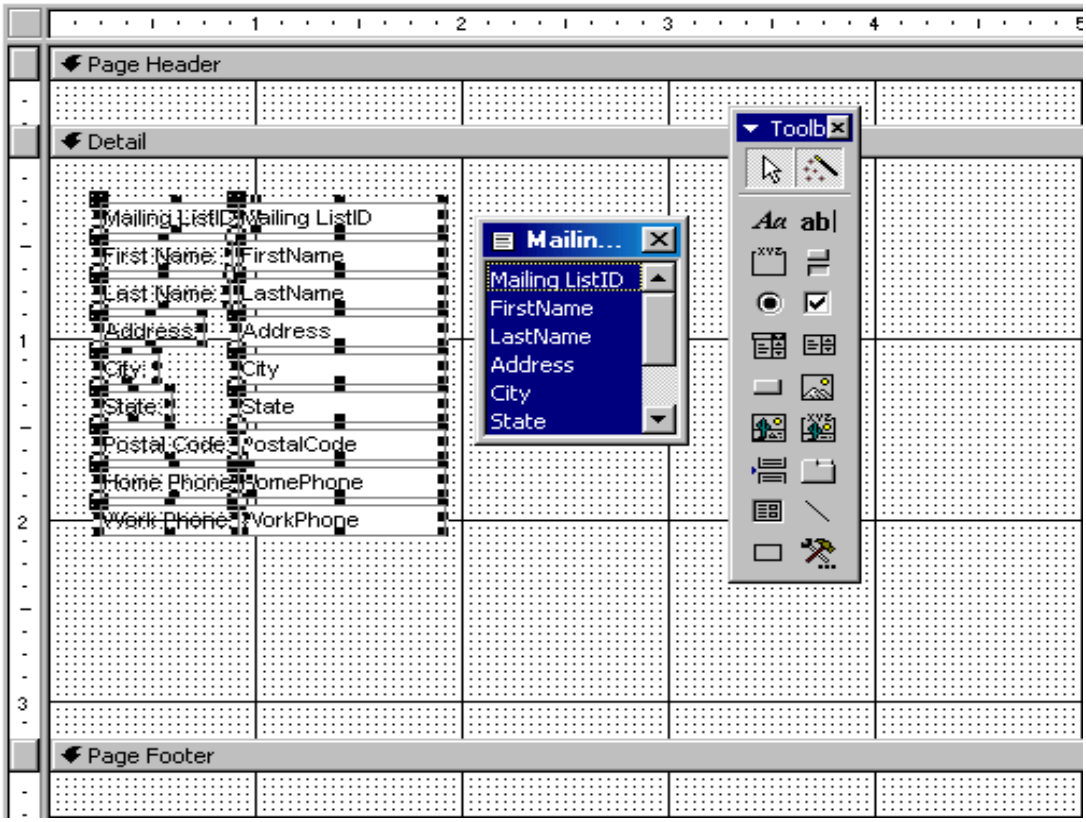


Figure 25: A sample report creation

3. At this point, the report has 3 sections: Page Header, Detail, and Page Footer. Other sections that can be added include Report Header, Report Footer, Grouping Header and Grouping Footer. The characteristics of each section vary:

Report Header

Appears at the top of the ***first*** page of the report. The title of the report usually goes in the report header.

Page Header

Appears at the top of ***every*** page of the report. Column headings usually go in the page header.

Grouping Header

If grouping has been selected, the grouping header may appear at the top of each new group. Grouping headers are optional.

Detail Section

This section contains the field values from the table or query that the report is based on.

Grouping Footer

If grouping has been selected, the grouping footer may appear at the bottom of each new group. Grouping footers are optional. Subtotals might be placed in the grouping footer.

Page Footer

Appear at the bottom of ***every*** page of the report. Page numbers usually go in the page footer section.

Report Footer

Appear at the bottom of the ***last*** page of the report. A grand total should be placed in the report footer.

Note that the sections at the far right side of this screen capture of a report. Grouping footers and page footers were not used.

4. Click on View in the menu bar, then Report Header/Footer to add these sections.

We will not use the page footer, so move the mouse to the top of the report footer bar; the mouse will change to a double-headed arrow.

Click and drag the report footer bar up until it meets the bottom of the page footer bar. This closes the page footer bar so that it will not take up unnecessary space. Your screen should look like this:

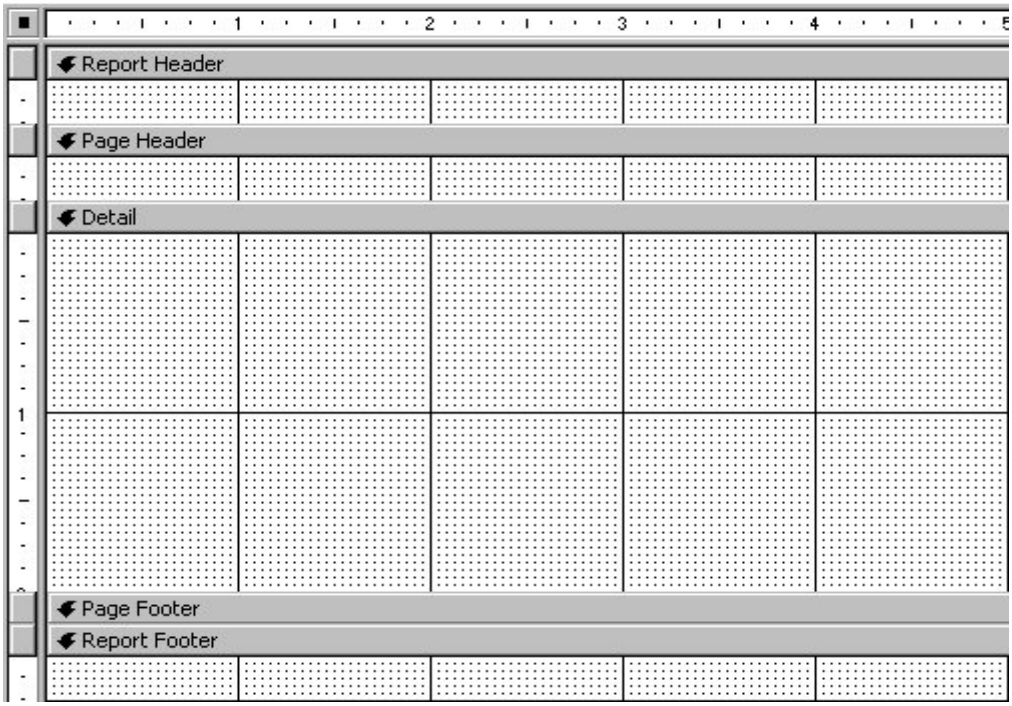


Figure 26: Report with header and footer

5. Next, we need to add the Sorting/Grouping Header. Click on the Sorting/Grouping icon on the toolbar. We want to group on the First Period class, so click on the list arrow at the right side of the field expression box and select First Period. Click in the group Header area, click on the list arrow, and select Yes. This adds the group header.

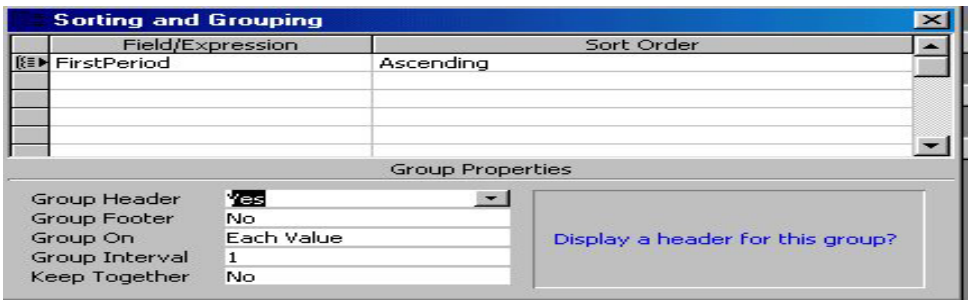


Figure 27: Adding sorting for date in report

6. Close the Sorting and Grouping window. You should now have the following sections in your report: report header, page header, first period header (this is the grouping header - it takes its name from the grouping field we selected), detail, page footer (which we have closed up), and report footer.

7. Now we can begin adding the fields into the report. Click on the Field List icon, and the list of available fields from our First Period query should appear. Click on the first field, SSN, and drag it down into the detail section. Repeat for FirstName, LastName and FirstPeriod. It really doesn't matter where you put them, but try to line them up something like this:

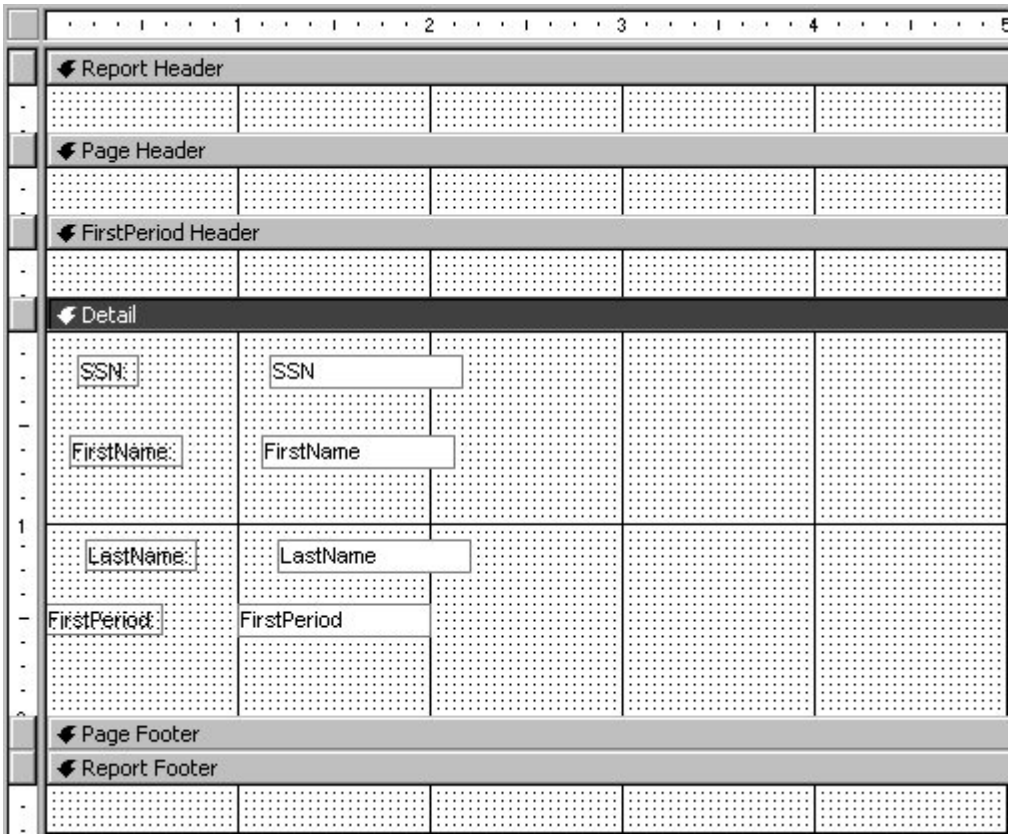


Figure 28: A report after adding fields

8. Close the Field List window. We have all of the fields we need.

9. The items in the detail section are called ***controls***, and in this case, they are ***bound*** controls. This means that they are directly linked to the underlying table or query. Bound controls consist of two parts, the label box, which in our example is on the left, and the text box which is on the right. You can change or delete the label box and the report will still run. However, you cannot change the contents of the text box. If you do this, you have changed the name of the field that the report is searching for in the underlying table or query, and the report will not work.

10. Each set of controls can be moved separately or together. Click on the SSN text box control (the one on the right). A set of handles should appear on that control, and a single handle should appear on its associated label control. Notice that the first item on the left of the formatting toolbar says SSN. This is the object list. Click on the list arrow on the object list, and you'll see all the other objects and their labels. Click on some of the other objects and watch the handles move to select those objects. You can use this object list as a visual clue to determine if you have a text box or a label selected. Text boxes will show the field name in the object list; labels will show label x (where x is some number). Hopefully, this will help to prevent you from making changes to contents of the text box controls.

11. Click back on the SSN text box control, then move your mouse to the top of the text box control. Your mouse pointer should change to a hand. If you click and drag. the hand allows both the text box and the label to move ***together***, keeping their relationship to each other. Now move the mouse to the larger handle located at the top left of either the text box or the label. Your mouse should change to a pointing finger. Clicking and dragging with the finger pointer will move ***only that control***.

All of the labels in our report are at the top of the page in the page header, so we will need to move them there. You can not click and drag across the boundaries of a section, so we will cut and paste instead. You can select more than one control at a time by clicking on the first control, then holding the Shift key down and clicking on the other controls.

12. Click on the SSN label box, hold Shift down, and click on the other label boxes to select them. When you finish, there should be handles on all of the label boxes.

Click on the cut icon.

Click anywhere in the Page Header section, then click on paste. The labels should move to the Page Header section. Your screen should look something like this:

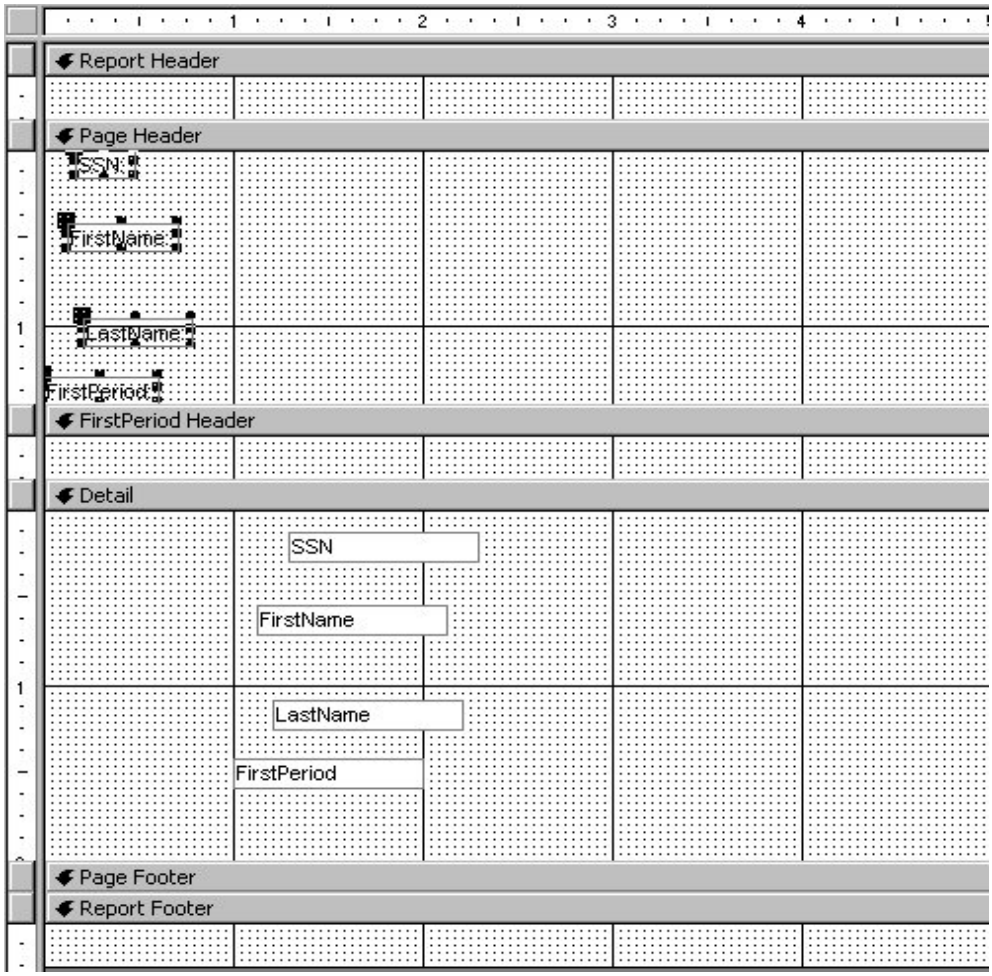


Figure 29: Reports with labels on page header section

Click anywhere in the Page Header section to deselect the labels.

13. By clicking and dragging, begin aligning the labels in the Page Header section with the text boxes in the Detail section. You can check your progress by changing to Print Preview and back to Design View. Use the first icon on the left to change views.

This is a tedious process - sometimes you have to go back and forth many times to get things aligned just right. Using the Grid dots is helpful, but the Align tool can help speed up the process.

Select the SSN label in the Page Header, hold Shift down, then select the SSN text box in the Detail section.

Click on Format, Align, and choose which side to align on. This will align both controls at whatever side you choose.

Try to make your report look like this:

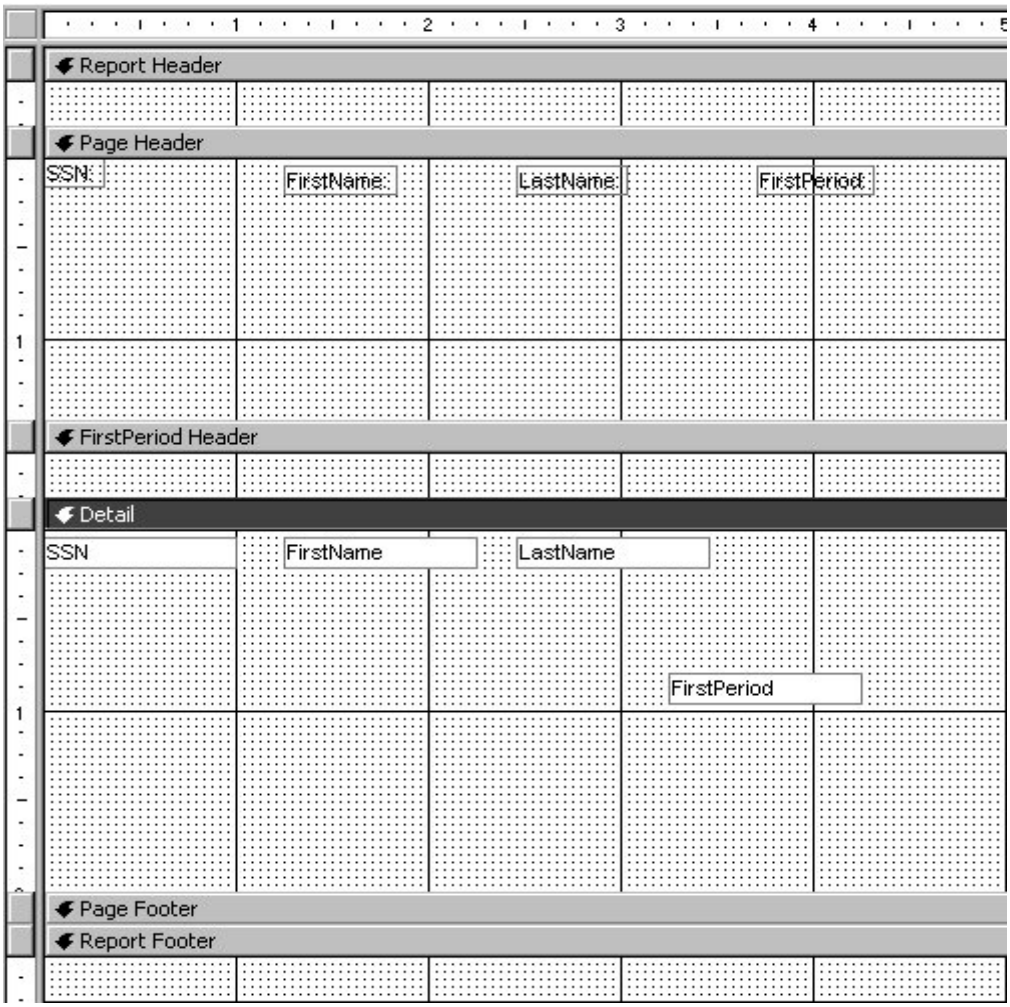


Figure 30: An aligned report

14. You must leave enough space between the controls, especially in the detail section, for the student's entire SSN, first name and last name to show. This may mean that you need to increase the size of the controls, which you can do by clicking and dragging on the handle on the right side of a control. Change to the Print Preview and scroll through the pages of the report. You can change from page to page by clicking on the navigation buttons in the bottom left corner of Print Preview.

Notice that neither of the FirstPeriod controls is aligned. These will be moved into the FirstPeriod Header in the next step.

Select the FirstPeriod label, which is now in the Page Header section. Cut and paste it into the FirstPeriod Header section. Do the same for the FirstPeriod text box that is in the Detail section. Position them at the far left of the grid, using the Align command to adjust them vertically. Chances are they will paste right on top of each other, so try to use the finger pointer to get them separated. Now the Report Design looks something like this:

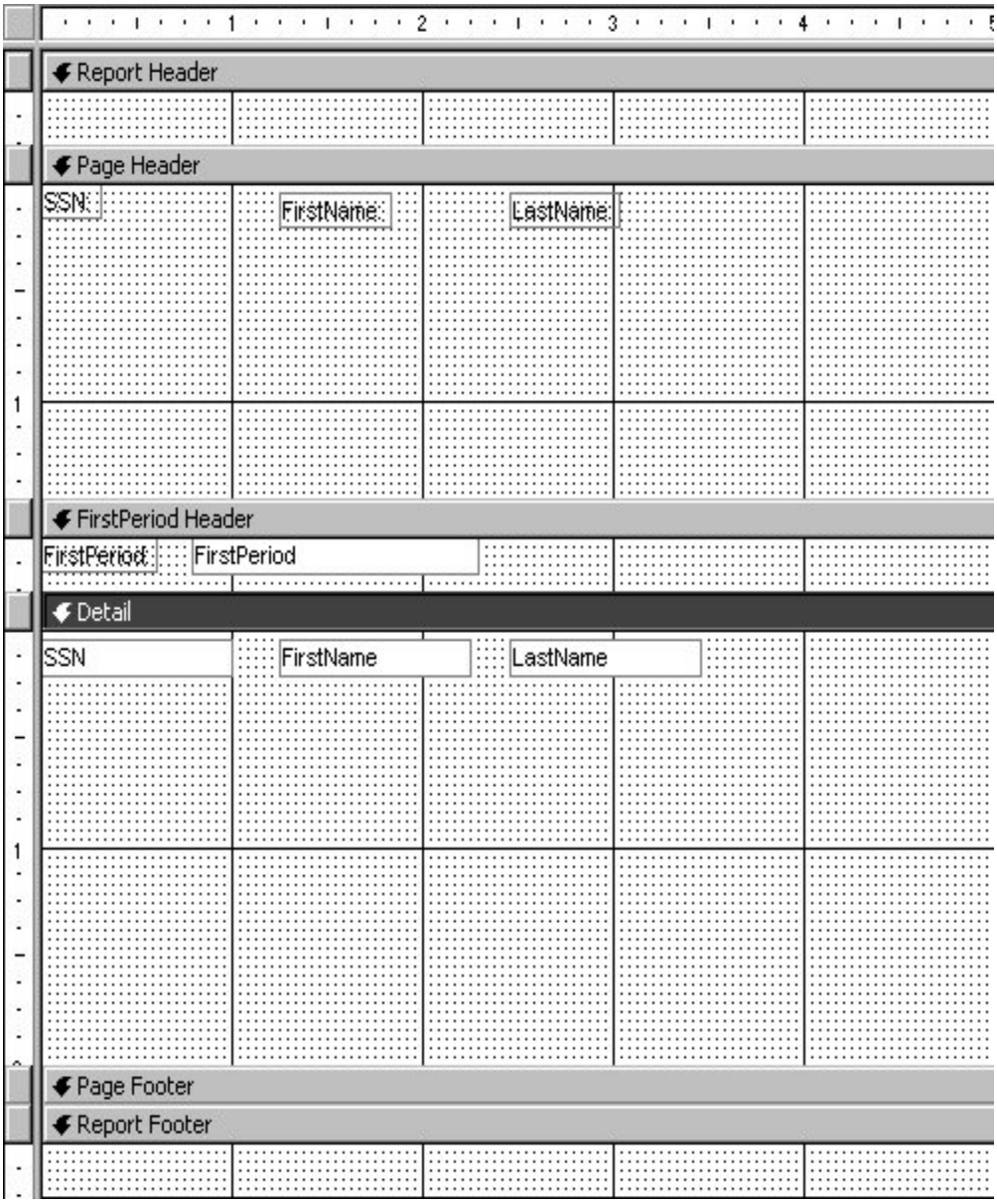


Figure 31: A report having periods

15. If you change to the Print Preview, alignment is probably close, but there is a great deal of blank space in the report. In the next step, we will close up the sections to get rid of the white space in the report.

Move your mouse to the top of the Page Footer bar; it should change to a double-headed arrow. Drag the Page Footer bar up until it almost touches the bottom of the text boxes in the Detail section. Repeat for at the top of the FirstPeriod Header bar - drag it up until it almost touches the bottom of the label boxes in the Page Header section. Your design should look something like as follows.

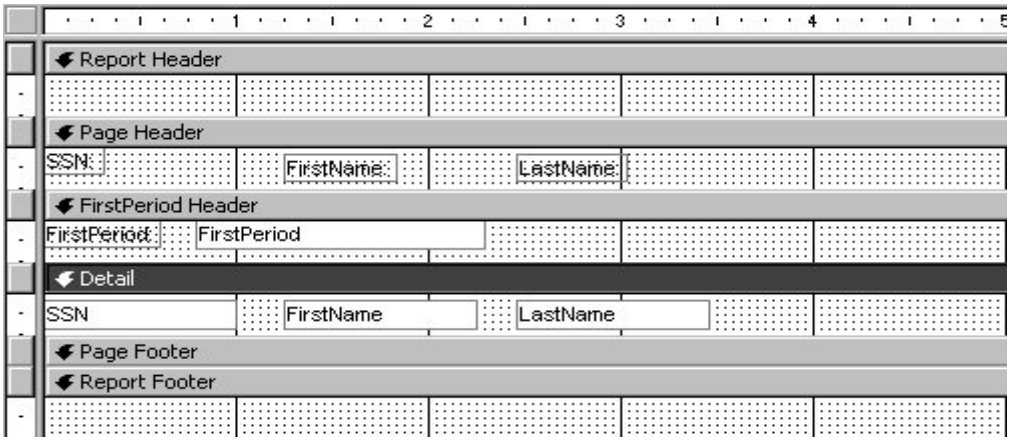


Figure 32: A final report layout

When you change to Print Preview, your report should have shrunk down to one page. However, it is hard to distinguish where you change from one class to another, so in the next step we will add a line and some formatting to make the report easier to read.

16. If your toolbox is not visible on your screen, click on the toolbox icon. The toolbox should appear as a floating toolbar.

The toolbox contains icons for many things, including some ***unbound*** controls such as lines and labels. Unbound controls are in no way connected to the underlying table or query. You can move them or change them to suit you without worrying.

Click on the line icon on the toolbar. Drag your mouse just under the controls in the FirstPeriod Header to draw the line. If you hold Shift down while you drag, you will be assured of a straight line.

Preview the report. The line helps, but it would be better if it were thicker, if the controls were bold and if there was a space between First and Period. Go back to the Design View to make these changes.

17. Point to the line and click to select it. If this proves too difficult, remember you can use the Object list at the left side of the formatting toolbar. Click on the list arrow and select the line (it should be the only one, so it doesn't matter what number it is). Once the line is selected, the handles should appear on the line.

Each control has ***properties***. Click on the Properties icon on the Report Design toolbar. Since the line was selected, the Line Properties window should appear.

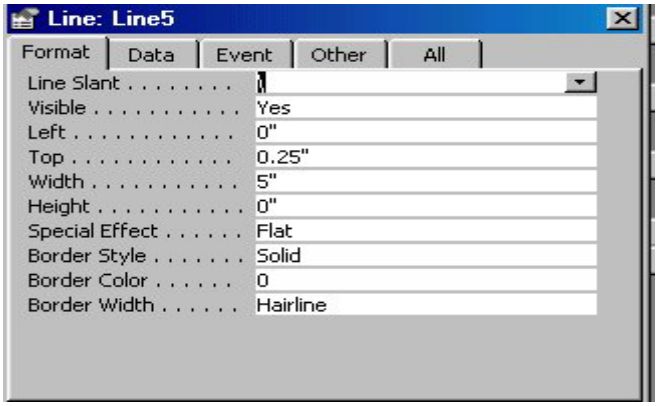


Figure 33: Setting properties in report

Click on the Border Width box, then click on the list arrow that appears to the right, and select 2 pt. This will make the line fatter and easier to see. Preview the report.

18. Change back to the Design View. Select both the FirstPeriod label and text box (remember to hold shift down), then click on the Bold icon. Preview the report to see the difference.

Change back to the Design View. If both the label and text box are still selected, click somewhere to deselect. Then select only the label box (be sure about this!) We need to put a space between First and Period. There are two ways to do this.
Either select the label box, click on Properties, then change the caption to read First Period, or Click once on the label box, then click again (2 single clicks, not a double-click). This should put you into the Edit mode for the label. Put a space between First and Period.

Either method will work. It is important to remember, though, you are doing this on the label, **NOT** the text box. If you ever should change a text box beyond repair, delete the text box and its corresponding label, click on the Field List icon, and drag the field back into the Report. You will then have to move and edit it back to its proper location and formatting.

A good clue that you have changed a text box is this: when you preview the Report, you get a Parameter message. This is usually what happens when Access cannot find a bound control that is in the report, but not in the underlying table or query.

At this point, the report should look something like this in Print Preview:

SSN:	FirstName:	LastName:
First Period:		
746-64-1354	Sam	Smith
583-17-5828	Kip	Johnson
572-47-2819	Kim	Carruthers
First Period: Chem101		
262-62-1854	Chris	Jackson
First Period: Eng101		
575-23-2990	Kyle	Baker
472-17-1718	Steve	Baker
456-27-7888	Kristy	Cooper
171-77-1828	Frieda	Little
First Period: Math 157		
573-28-1739	Susan	Smith
462-78-2739	Beth	Porterson

If the lack of space between Chem and 101 or Eng and 101 bothers you, you would need to go back to the Student Schedule table and change all the records, adding spaces appropriately.

19. The report would look better with a title. We will put the title in the Report Header, so that it will appear only once, at the beginning of the report.

Click on the Label tool on the Toolbox.

Drag and draw a box in the Report Header section. The size of the box really does not matter – we can adjust it later.

As soon as you leave your finger off the mouse, you will go right into editing the contents of the label. Key in First Period Student Roster, then click some place outside of the box to get out of the editing mode. Preview the report. The title needs to be bigger and bolder. Go back to the Design View.

Select the label box; handles should appear around it. Select 18 pts and Bold on the Formatting toolbar. However, as soon as you do this, the text becomes too big for the box.

Click on Format, Size, To fit. This should adjust the box size, although you may have to drag the handle at the far right to increase the length of the box.

While the box is selected, you could also change the font face, colour, fill or back colour and alignment by using other icons on the formatting toolbar.

Move the label so that it aligns at the left side of the report. Check it with Print Preview.

At this point, your report should look something like this:

First Period Student Roster

SSN:	FirstName:	LastName:
First Period:		
746-64-1354	Sam	Smith
583-17-5828	Kip	Johnson
572-47-2819	Kim	Carruthers
First Period:	**Chem101**	
262-62-1854	Chris	Jackson
First Period:	**Eng101**	
575-23-2990	Kyle	Baker
472-17-1718	Steve	Baker
456-27-7888	Kristy	Cooper
171-77-1828	Frieda	Little
First Period:	**Math 157**	
573-28-1739	Susan	Smith
462-78-2739	Beth	Porterson

20. The report fits all on one page and it is readable. Note how the students in each class are grouped by the name of the class they are taking. This is the Grouping Header (FirstPeriod Header) at work.

21. Our example report also had a counter in the Report Footer. This is a ***calculated*** control. Calculated controls display values calculated from one or more fields in the underlying table or query. To create a calculated control, use the text box icon on the Toolbox, then key in the formula.

The section that a calculated control is placed in plays an important role. For instance, if you wanted to find the sum of all invoices for companies in a particular state, where the field is called Invoice, the formula = sum ([Invoice]) would be placed in a text

box in the Grouping header for State. Think of this as a subtotal. The **exact same formula** placed in the Report Footer (the **last** thing in the Report) would produce a grand total**.**

Formulas used for calculated controls are very similar to those used in Excel.

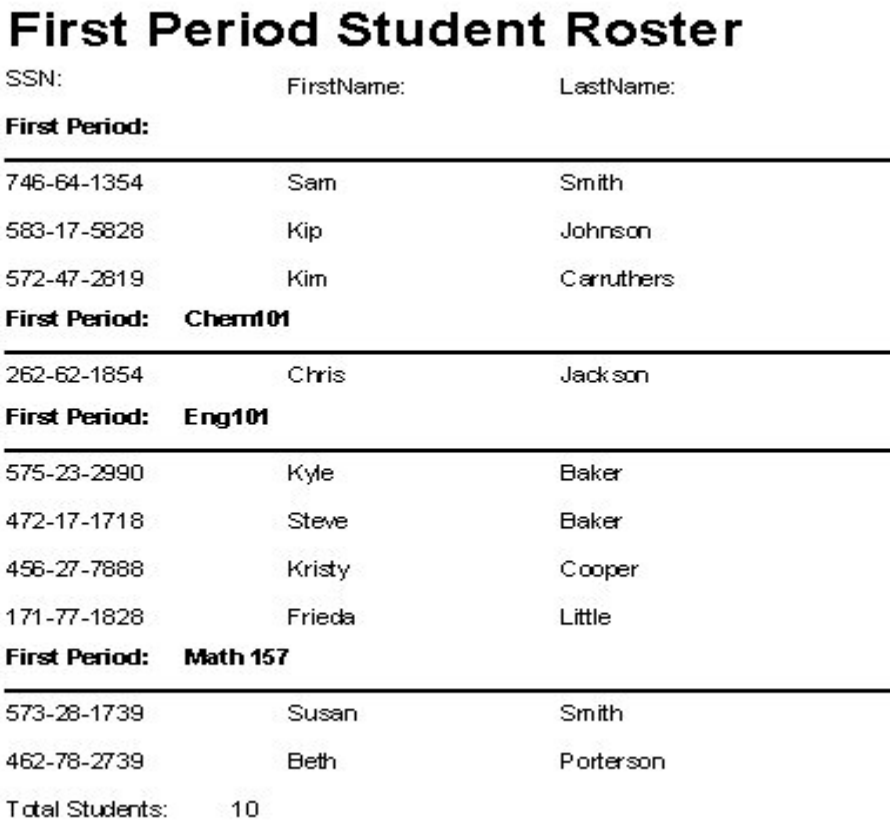
Our example has a calculated control in the Report Footer that counts the number of students in our roster. To create this:

Click on the text box icon on the Toolbox, then click anywhere in the Report Footer section. The text box and its accompanying label will appear. The text box has "Unbound" in it.

Click on Unbound, then key in this formula = count ([LastName]). This will count the number of last names in the detail section. Note the use of both the parentheses and the brackets and their position. Also the name of the field must be spelled identically to the way it is in the underlying table or query.

Select the text box (the one containing the formula), then click on the left align icon on the Formatting toolbar. Numbers align to the right by default. By changing the alignment to left, there will not be as big a gap between the label and the number. Click in the label box, delete the contents, then key in Total Students**.**

Align the boxes at the left side of the report, then switch to Print Preview. The total should be 10 and your finished report should look something like this:



First Period Student Roster

SSN:	FirstName:	LastName:
First Period:		
746-64-1354	Sam	Smith
583-17-5828	Kip	Johnson
572-47-2819	Kim	Carruthers
First Period: Chem101		
262-62-1854	Chris	Jackson
First Period: Eng101		
575-23-2990	Kyle	Baker
472-17-1718	Steve	Baker
456-27-7888	Kristy	Cooper
171-77-1828	Frieda	Little
First Period: Math 157		
573-28-1739	Susan	Smith
462-78-2739	Beth	Porterson
Total Students: 10		

Figure 34: A final report printout

Close the report. Save it as First Period Student Roster**.**

22. Things that may cause problems in reports:

If you get a message that some of the area of the report is outside the printable area, take a look at the width of the grid. It can be adjusted by dragging the right side of the grid. You can also change the paper to landscape by clicking on File, Page Setup.

Watch out for typos in field names. Again, a big clue is the Parameter query message. If that happens, go back and start looking at field names in the report.

If you try to open a report and you get a message that tells you the table or query that the report is based on no longer exists or the name may have changed, stop and think. Did you change the name of a table or query lately, or delete a table or query? If that's the case, the report will no longer function.

A lot of times, users see a typo in a table or query name, so they rename it, not thinking about the consequences. If the table or query still exists in the database, try renaming it to the previous name and see if that makes your report work.

23. Take time to further investigate the properties window. Each control has properties, and there are many properties that can help you customize your report.

Exercise 4

Now you must create the reports for HMS. The sample reports that you may create can be:

- Report on Patients of OPD and also a report on other patients
- Report on Doctor's details
- Reports on Departments & employees
- Report giving combination of Doctors and Patients
- Report providing details on Patients attended by a Doctor and so on.
- Reports of total charges obtained.

6.0 USING QUERIES AND RECORD SET

A query is a question about information in a table or tables. Queries can be used to view and analyze data or as a basis for forms and reports. Queries are commonly used to display fields from related tables and enable us to control not only what records display, but also what fields display.

A query does not contain data. Rather, it is a set of instructions. Access uses these instructions to select and display the appropriate records in a table. The query always considers all the data in the table. If the new records meet the conditions of the query, they will be included when the query results appear.

When a query is opened or run, a RecordSet appears. A RecordSet contains all the fields and records that meet the conditions of the query. While the RecordSet is not a table, it can be used under certain conditions to add and edit records in tables.

6.1 Using the Simple Query Wizard

Access provides a Simple Query Wizard that guides through the steps to create a basic select query. When the Simple Query Wizard is used, the table from which the data is to be taken is selected as well as the fields to be displayed in the query. In the last step, the query is named and then whether or not to display the results (the RecordSet) of the query is chosen. To change the design of the query Design View can be used.

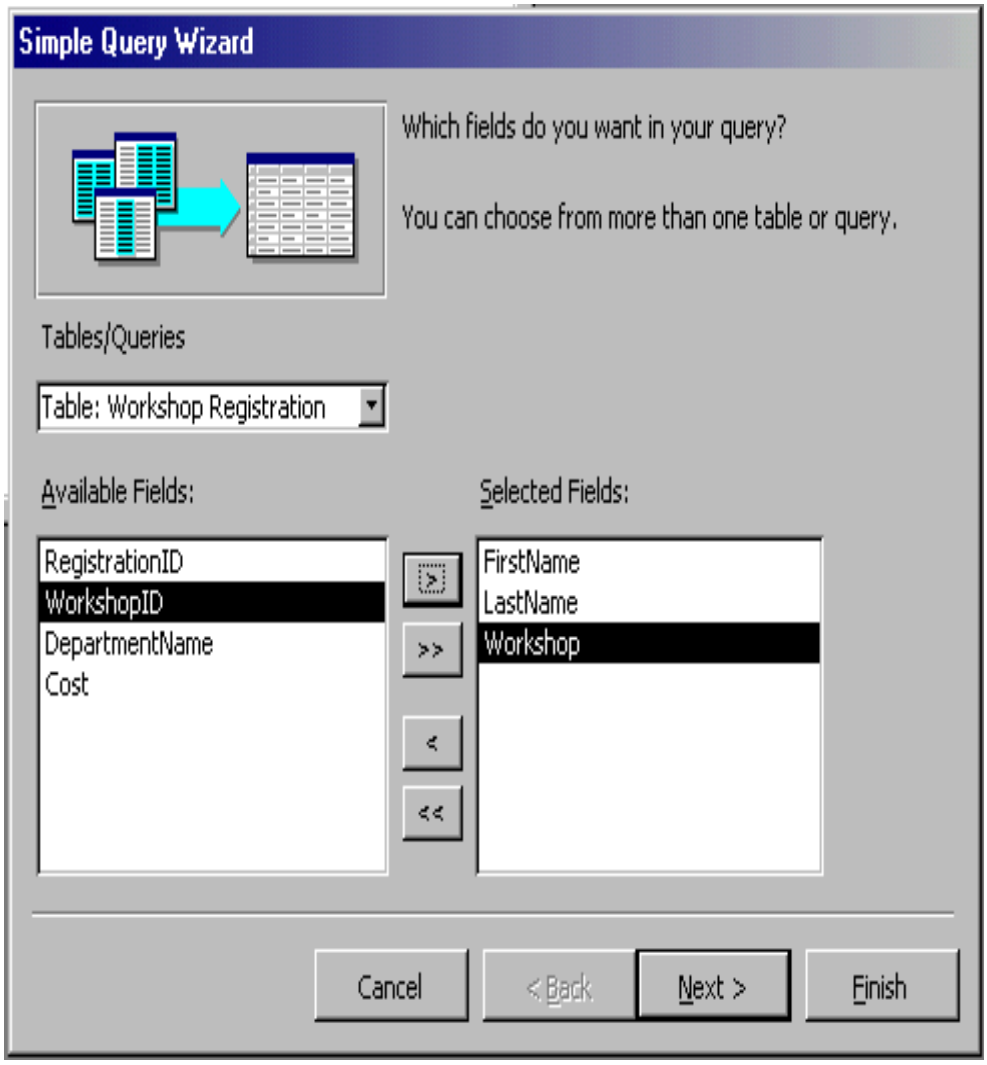


Figure 35: A query form

Create a Query that shows the First Name, Last Name, and Workshop for workshop participants.

1. Select the **New** button on the Database window toolbar. Select **Simple Query Wizard**.
2. Select **OK**.
3. Select the **Tables/Queries** list.
4. Select the table or query you want to use as the basis for the query.
5. Use the single right arrow to add the field(s) you want to display in the query from the **Available Fields** list box.
6. Select **Next**.
7. Type a name for the query.
8. Select **Finish**.

6.2 Creating a Query in Design View

A query can be created in Design view. This option gives most flexibility in designing a select query. It allows addition of criteria to select records and sort the resulting RecordSet.

To create a query in Design view, the design grid is used to set up the query. The field list of the required table appears in the top pane of Design view. The required fields can be added to the design grid in the bottom pane of Design view, along with any sort orders or criteria for selecting records.

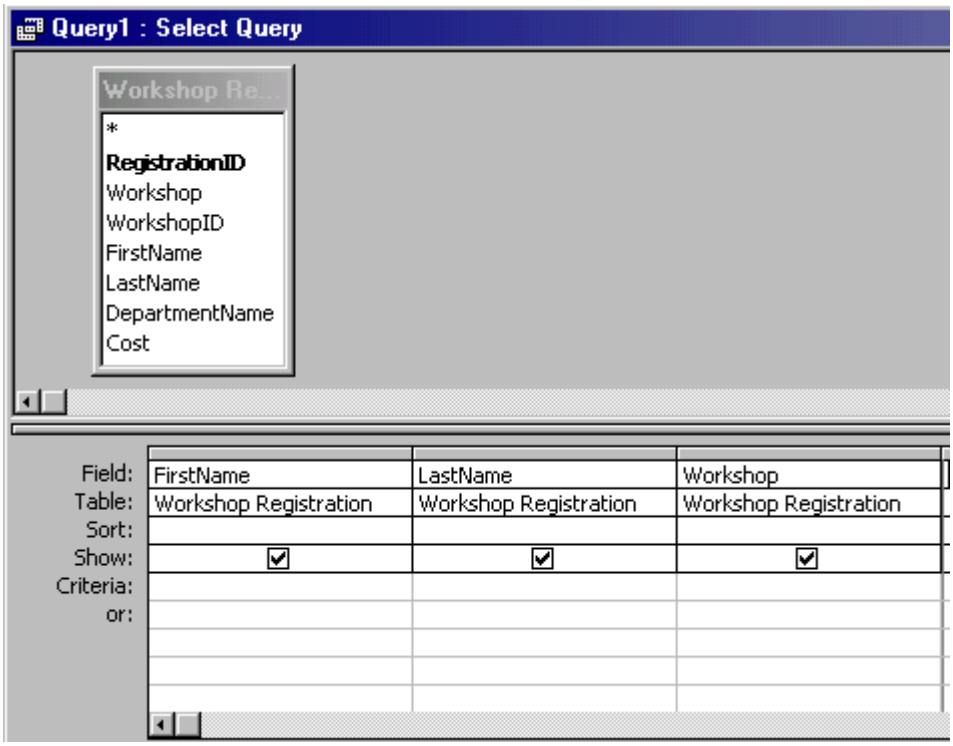


Figure 36: A query making

Create a Query in Design view that shows the First Name, Last Name, and Workshop for workshop participants.

1. Click the **New** button on the Database window toolbar.
2. Select **Design View**.
3. Select **OK**.
4. Select a table and add it to the query.
5. Select **Close** to close the Show Table dialog box.
6. Select the first field you want to add to the query.
7. Drag the field to the desired column in the **Field** row in the design grid.
8. Add other fields to the query as desired.
9. Click the **Save** button.
10. Type a name for the query.
11. Select **OK**.

6.3 Opening a Query

To open a query, Access runs the instructions in the query and displays the resulting RecordSet in **Datasheet** view. If new records have added, since the last execution of a particular query, the new records will appear only if they meet the query criteria.

1. Select the name of the query you want to open.
2. Select the **Open** button on the Database window toolbar.

More than one table can be used in a query. The tables must be joined in order for the query to give accurate results. If they are not joined, a join is created in the top pane of Design view.

On adding more than one table, the field lists appear in the top pane of Design view. If the tables are already related, join lines appear automatically.

Once a table is added to the query, the fields can then be added from the field list to the design grid. The second row in the design grid is the Table row, which indicates from which table the field originates.

When Design view is opened to design a new query, the Show Table dialog box opens automatically so that multiple tables can be added. However, when Design view is opened to modify an existing query design (i.e., to add a table), the Show Table dialog box is to be opened manually.

1. Select the name of the query to which a base window toolbar.
2. Click the **Show Table** button on the **Query Design** toolbar.
3. Select the name of the table to be added to the query.
4. Select **Close** to close the Show Table dialog box.

In case of multiple tables query, the tables must be joined in order for the query to give accurate and meaningful results. If Access does not know how to relate the data between tables, it displays every combination of data between the two tables. For example, if one table has 20 records and the other has 5, then the RecordSet will contain 100 records and the results are virtually meaningless.

If the relationships have already been defined in the Relationships window, join lines between the field lists display automatically in Design view. Therefore, if a related table is added to a query, a join line appears automatically. Access also automatically creates a join if there is a field with the same name in both tables. If there is no predefined relationship or fields with the same name, such a relationship must be created.

The join type defined in the relationship is particularly important to queries. The default type is an inner join in which records are only included in the RecordSet if there is matching data in the join fields for both tables. An outer join can also be created in which all the records from the "one" table appear, even if there is no matching data in the "many" table.

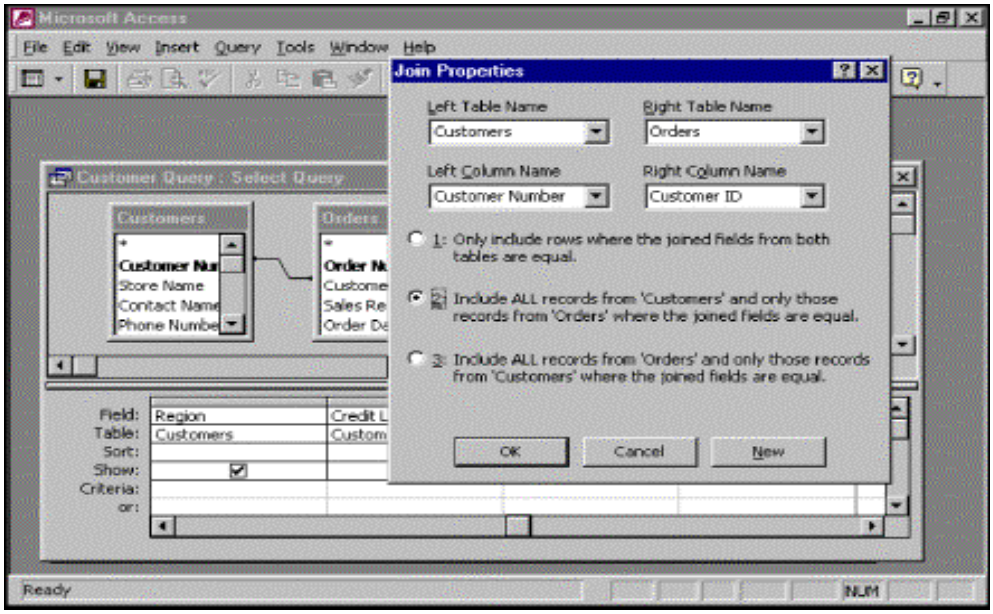


Figure 37: Query on joined tables

1. Select the join field in the field list of the first table.
2. Drag the field on top of the matching field in the field list of the second table.
3. Double-click the join line.
4. Select the desired join type.
5. Select **OK**.

Running a Query

A query can be run directly from Design view to display the RecordSet. This option is useful to test the design of the query to see if the resulting RecordSet contains the required information.

Running a query does not save the design of the query. If the RecordSet is closed after running a query, a message box opens, asking to save the changes.

Run Query button

1. Click the **Run** button on the **Query Design** toolbar.

6.4 Adding Features in Query

Sorting a Query

When a query is run, the records in the RecordSet appear in the same order in which they appear in the table. The records can be ordered by either sorting the RecordSet or assigning a sort order in the query design. The RecordSet can be sorted just as a table. However, this must be done every time the query is executed. If the sort order is assigned in the query design, Access performs the sort automatically each time the query is executed.

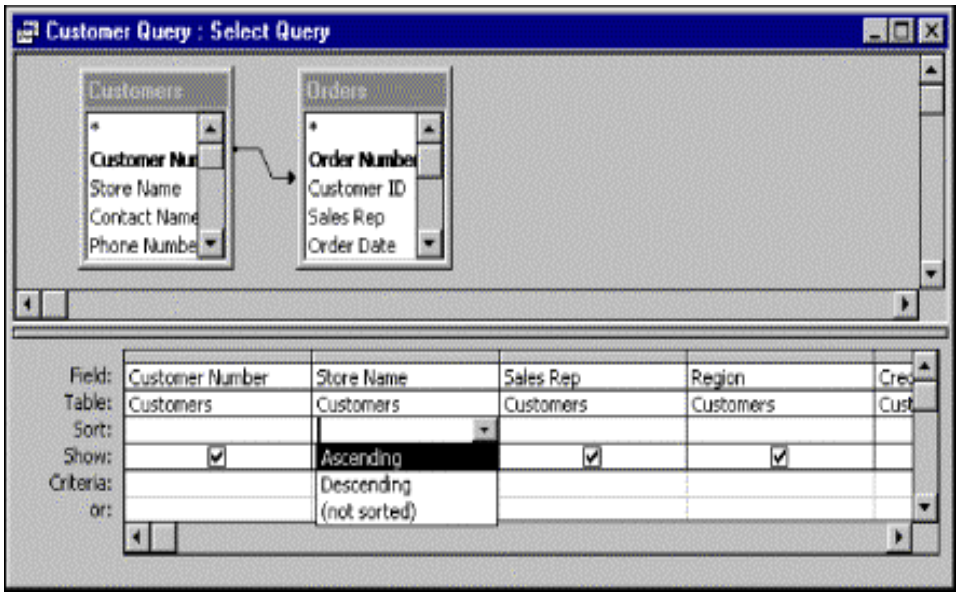


Figure 38: Sorting in a query

1. Select the **Sort** row under the field by which you want to sort.
2. Select the **Sort** list.
3. Select **Ascending** or **Descending**.

Data can be entered in the **Criteria** row of the query design grid to restrict the number of records that are returned in a query.

*To select records that meet a single value, enter the value in the **Criteria** row under the appropriate field. Access automatically inserts quotation marks (" ") around alphanumeric entries and number symbols (#) around date entries. If the entry is numeric, the number appears without quotation marks. When the query is run, only those records with values that match the criteria appear in the RecordSet.*

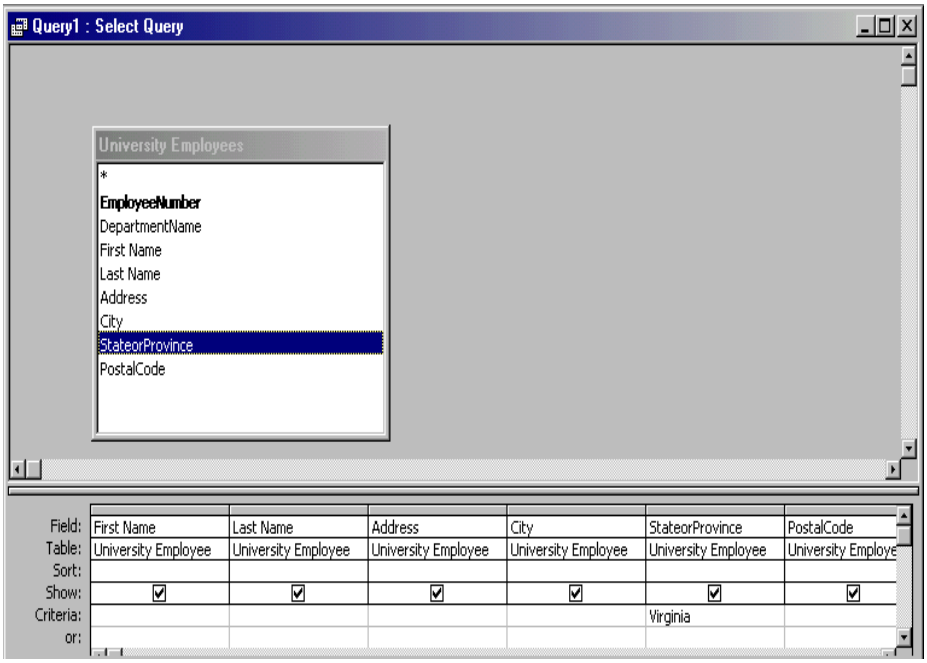


Figure 39: A sample query

For Example, Create a Query that filters all University Employees from the state of Virginia.

Write a new Query that includes the following fields: First Name, Last Name, Address, City, StateorProvince, PostalCode.

1. Select the **Criteria** row under the **StateorProvince** field.
2. Type the value for the criteria.
3. Press **[Enter]**.

Hiding a Field in a Query

A field can be used to select records that meet a certain criteria without displaying the field in the RecordSet. This option is useful when all the records meet the same specified criteria.

For example, you may want to create a query to display customers in the **Southeast** region. Therefore, you must add the **Region** field to the query in order to enter the criteria to select only the **Southeast** region. However, you might not want the field to appear in the RecordSet because you know that all the data in the **Region** field is the same (**Southeast**). In this case, you may want to hide the field you used for the criteria.

The design grid includes a **Show** row with a check box for each field. If the check box is selected, the field will appear in the RecordSet. If the check box is deselected, the field will be hidden in the RecordSet. All the **Show** field check boxes are selected by default.

Adding a Record Using a Query

A query can be used to update records in related tables. When information is entered into the join field for a primary table, Access automatically locates the corresponding information in the related table and enters it into the record.

For example, the **Orders** and **Customers** tables are related. When you enter the customer ID number in the appropriate field in the **Orders** table, Access completes the customer name and address information automatically through the relationship with the **Customers** table.

You can always edit information in queries based on a single table. In queries based on related tables, Access must be able to determine the relationship type (i.e., one-to-one or one-to-many), or you cannot edit information. In queries with tables that have a one-to-one relationship, you can always edit data. In queries with tables that have a one-to-many relationship, you may not be able to edit data at times. For example, this can happen if the join field from the "many" table is not in the query, since Access cannot create new records in the "many" table. For example, Access cannot create new orders for a customer if the query is based on the **Customers** table unless the **Customer ID** field from the **Orders** table is included in the query.

New Record button

Click the **New Record** button on the **Query Datasheet** toolbar.

1. Type the necessary data in the first field.
2. Move to the next field.
3. Continue to add data as necessary.
4. Save the current record.
5. Printing a Query

The RecordSet resulting from a query can be printed. It can be printed after running a query, while it appears on the screen, or by printing it directly from the Database window. If the RecordSet is printed from the Database window, Access runs the query and sends the results to the printer, rather than to the screen.

6.5 Conditional Queries

Using Comparison Operators

In order to select specific records the criteria is entered in the **Criteria** row of the query design grid. The simplest criterion requires that records match a single value to be included in the RecordSet.

Comparison operators can also be used to select a specific group of records in a table. For example, if you want to find all customers with credit limits less than 1000, or all customers with a contract date on or before January 2004, you can write an expression that defines the criteria using a combination of comparison operators and field values, such as <1000 or <=1/1/04. Comparison operators are symbols that represent conditions recognised by Access. The available comparison operators are

Operator	***Description***
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
=	equal to
<>	not equal to
Not	reverse logic

A comparison operator can be used to compare a specified value with all the values in a field. When the query is executed, only the records with values meeting the specified criteria appear in the RecordSet.

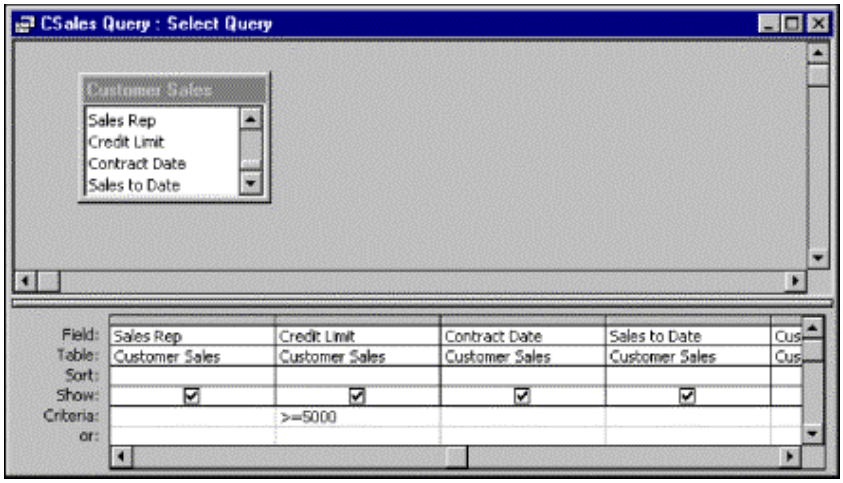


Figure 40: A sample conditional query

1. Select the **Criteria** row under the desired field.
2. Type a comparison operator and the value for the criteria.
3. Press **[Enter]**.

Using an AND Condition

Many times, a query requires more than one condition to obtain the desired result. For example, if you want to find all customers in PA with sales to date over 10,000, you would need two conditions: State=PA and Sales to Date>10000. The records must meet both conditions in order to be included in the RecordSet. To combine two criteria in this way, you use the **And** logical operator.

You can use the **And** operator in a single field or in different fields. In a single field, you can use the **And** operator to find records that fall into a range. For example, to find customers whose contract dates fall between 9/1/04 and 9/30/04, you type both criteria on a single line in the **Criteria** row under the appropriate field (i.e., **>=9/1/04 And <=9/30/04** in the **Contract Date** field).

The **And** operator also allows you to impose conditions in two different fields. For example, to find customers in PA with sales to date over 10,000, you type each criterion on a single line in the **Criteria** row under the appropriate fields (i.e., **=PA** in the **State/Province** field and **>10000** in the **Sales to Date** field).

Field:	First Name	Last Name	DepartmentName	StateorProvince
Table:	University Employee	University Employee	University Employee	University Employee
Sort:				
Show:	☑	☑	☑	☑
Criteria:			"Education"	"Maryland"
or:				

Create a Query that lists University Employees from the Education Department **AND** from the state of Maryland.

Using the University Employees database, create a new Query in Design view using the University Employees Table. Add the following fields: ***First Name, Last Name, DepartmentName, and StateorProvince.***

1. Select the **Criteria** row under the **DepartmentName** field.
2. Select the **Criteria** row under the **StateorProvince** field.
3. Switch to **Datasheet** view to view the Query results.

Using an OR Condition

Many times, a query requires more than one condition to obtain the desired result. For example, if you want to find all customers in PA or all customers with sales to date over $10,000, you would need two conditions: State=PA as well as Sales to Date>10000. The records only need to meet one of the conditions in order to be included in the RecordSet. To combine two criteria in this way, you use the **Or** logical operator.

*You can use the **Or** operator in a single field or in different fields. In a single field, you type the criteria on two separate lines under the same field. In different fields, you type the criteria on two separate lines under the appropriate fields. For example, to find all customers with contract dates on or before 1/1/04 or credit limits above 3,000, you type <=**1/1/04** in the **Criteria** row under the **Contract Date** field and >**3000** in the or row under the **Credit Limit** field.*

Field:	First Name	Last Name	DepartmentName
Table:	University Employee	University Employee	University Employee
Sort:			
Show:	☑	☑	☑
Criteria:			"Psychology"
or:			"Music"

*Create a Query that lists University Employees from the Psychology **OR** Music department*

Using the University Employees database, create a new Query in Design view using the University Employees Table. Add the following fields: **First Name, Last Name, and DepartmentName.**

1. Select the Criteria row under the DepartmentName **field.**
2. Select the **or** row under the **DepartmentName** field.
3. Switch to **Datasheet** view to view the Query results.

Using the BETWEEN AND Operator

To find data that is between two values, the **Between** And operator can be used in a query. It can be used with a text, numeric, or date field. For example, to find all records of customers with credit limits between 1000 and 2000, enter **Between 1000 And 2000** in the **Criteria** row under the **Credit Limit** field.

1. Select the **Criteria** row under the desired field.
2. Type the **Between And** operator and the criteria.
3. Press **[Enter]**.

6.6 Advanced Query Features

Setting Top Values in a Query

The results of a query can be limited so that only the highest or lowest values for a field appear in a RecordSet. For example, you can set the top values of a **Quantity Sold** field to 10 to find the top ten best selling products. You can limit the number of records to a specific number or a percentage of all records being queried (i.e., top 25%). The field for which you are setting the top or bottom values must be sorted. If the field is sorted in descending order (Z to A, 9 to 0), the top values will be found. If

the field is sorted in ascending order (A to Z, 0 to 9), the bottom values will be found.

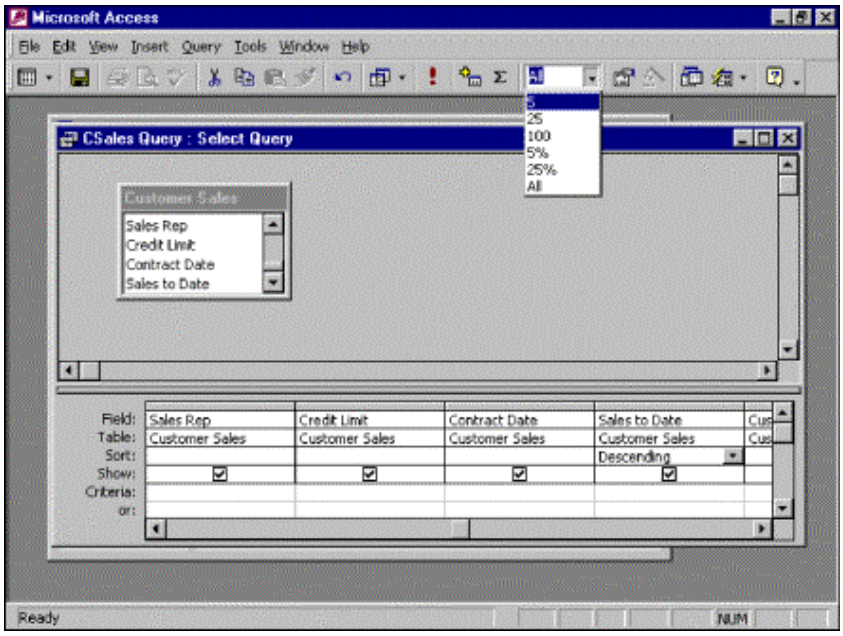


Figure 41: A query with advanced features

1. Select the **Sort** row under the desired field.
2. Select the **Sort** list.
3. Select the desired sort order.
4. Select the **Top Values** list on the **Query Design** toolbar.
5 Select the desired top value.

Creating a Function Query

Access allows creating a query that groups records by a selected field and applies a function that calculates values on other fields in the query as needed. ==For example, you can group records in a table by state and then select the **Count** function to find out how many customers (records) are in each state (field). You can also group by customer name (field) and calculate the **Sum** of each customer's orders (record values).

There are several types of functions, the most common of which are listed in the following table:

Function	***Description***
Sum	Sums the values in the calculated field
Average	Finds the average value of the calculated field
Count	Counts the number of records in the calculated field
Max	Finds the highest value in the calculated field
Min	Finds the lowest value in the calculated field

1. Select the **New** button on the Database window toolbar.
2. Select **Design View**.
3. Select **OK**.
4. Select the name of the table you want to add to the query.
5. Select **Add**.
6. Select **Close** to close the Show Table dialog box.

7. Select the field by which you want to group.
8. Select the field you want to calculate.
9. Select the **View** menu.
10. Select the **Totals** command.
11. Select the **Total** row under the field you want to calculate.
12. Select the **Total** list.
13. Select the desired function.

Creating a Parameter

A query can be executed with different criteria each time. To do this a parameter query can be created. A parameter query is a query that prompts the user for information when the query is run. Access then uses the information as the criteria and runs the query. The resulting RecordSet only includes those records that meet the criteria. This option allows one to avoid displaying the query in **Design** view each time the criteria is changed.

*You enter the text that will display in the prompt in the **Criteria** row under the appropriate field in the design grid, followed by a colon (**:**) and enclosed in square brackets (**[ ]**). You can set up a parameter query to prompt the user for more than one piece of information as well.*

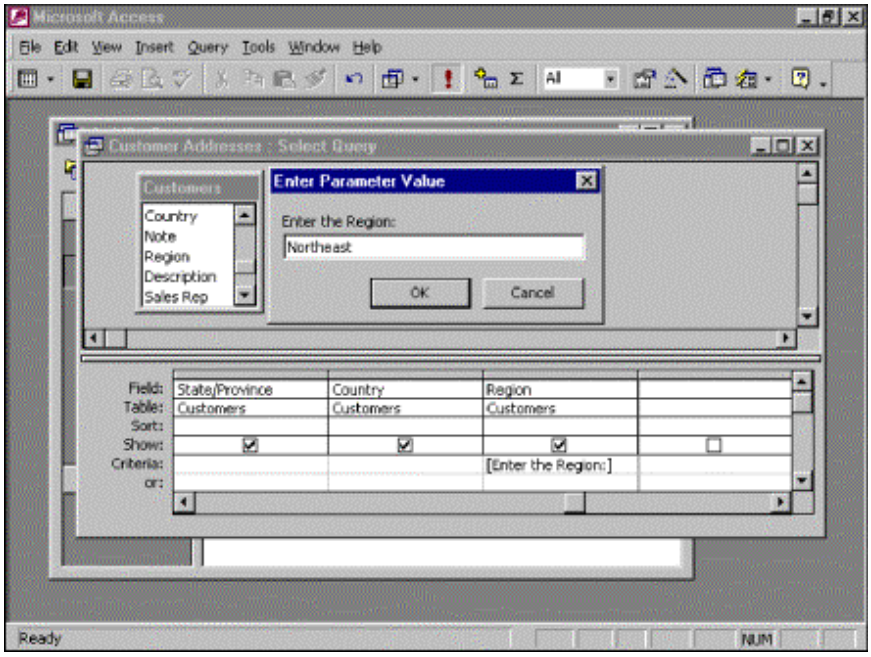


Figure 42: A query with parameter

*1. Select the **Criteria** row under the desired field.*

The insertion point appears in the **Criteria** row under the appropriate field.

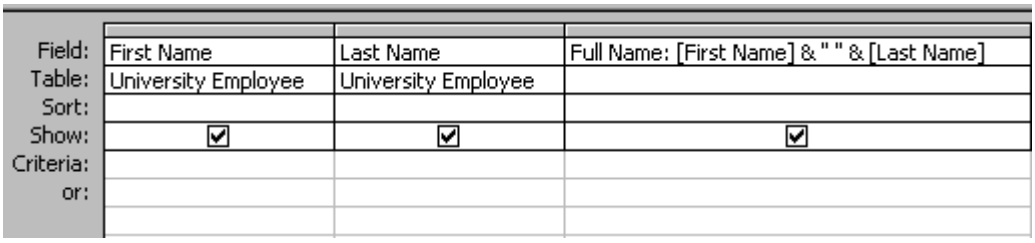
1. Type the desired text for the prompt, followed by a colon (**:**) and enclosed in square brackets (**[ ]**).
2. Press **[Enter]**.

Creating a Concatenation in a Query

Access allows combining two or more fields into one field. This process, known as concatenation, enables us to add field name text strings to one another. The text strings can follow each other with or without spaces. You can add other characters between the text strings if needed. For example, you can combine the individual **City**, **State**, and **Postal Code** fields into one field called **Address**. You can have the comma

and space characters appear between the field text strings. This concatenation process can be performed by creating a query to combine two or more fields.

When typing expressions for concatenation, the first part of the expression defines the name of the new field. The second part of the expression defines the fields which are to be concatenated. These field names must be surrounded by brackets. The ampersand (&) appears between the field name brackets. Any additional characters that are to appear between the fields are surrounded by double quotes. For example, the expression *Names: [First Name] &" - "&[Last Name]* ***creates a new field called*** *Names****. The new field contains the first name and last name separated by a space, a hyphen, and a space.***



Field:	First Name	Last Name	Full Name: [First Name] & " " & [Last Name]
Table:	University Employee	University Employee	
Sort:			
Show:	☑	☑	☑
Criteria:			
or:			

Using the **University Employees** database, create a **Concatenation** Query to combine **First Name** and **Last Name** into one field called **Full Name**.

1. Open the **Combine Names** Query from the **University Employees** database in Design View.
2. Select the next available field in Query design view and type the concatenation.

Using Multiple Tables in a Query

There may be instances when more than one table or query are added to a query. In order to do this, ensure that the field lists are joined to each other with a join line so that Access knows how to connect the information.

If tables in a query are not joined to one another, Access does not know which records are associated with which, so every combination of records between the two tables would appear in the query. Therefore, if each table had 50 records, the query's results would contain 2500 records (50×50), thereby rendering useless results.

If a relationship has been previously created between tables in the Relationship window, Access automatically displays join lines when related tables are added in query **Design** view. If the relationships have not been previously created, Access automatically creates joins if tables were added to a query as long as the tables each have a field with the same or compatible data type and one of the join fields is a primary key. If tables added to the query do not include any fields that can be joined, one or more extra tables or queries have to be added to serve solely as a bridge between the tables containing the required data.

Exercise 5

After doing so much about queries, you now create at least five queries relating to HMS. Some sample queries are listed below:

- Find the doctors who have not seen any patient.
- Create more relationship tables and track the history of the patients.
- Create more relationship tastes and track the payment to employees over last year. Also calculate the income tax return for each employee using a simple formula.

7.0 SUMMARY

This block is an attempt to initiate you into the practical world of Database Management System. We started the block with a brief but not sufficient description of a Hospital Management System and have identified a few tables. You can undertake a thorough study on this system such that you are able to design better tables for this system. We have tried to explain the process of creating various tables, their relationships, forms, reports and queries. However, you must use this information to develop the required exercises for hospital management. Thus, the block provides a flavor of various tools that are essential to build up a database application. Some of the basic tools which have been introduced in this block are:

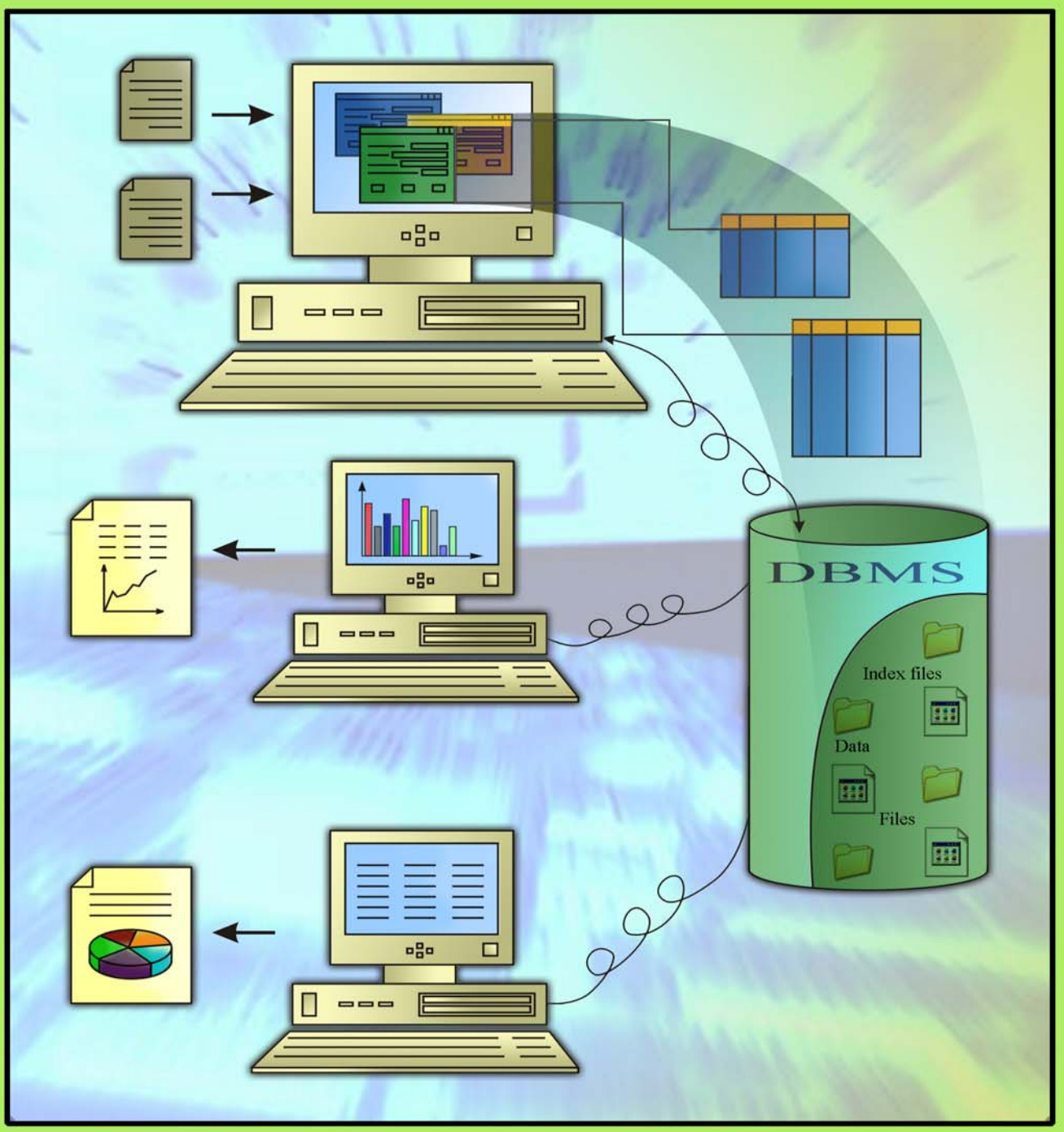
- MS-Access Reports
- MS-Access Query Generator
- Visual Basic – a very brief introduction

You must keep working on these tools to obtain expertise over a period of time. However, it is strongly recommended that you also use certain open source tools like MySQL and PHP.

SOCIAS-IGNOU/P.O.10.5T/MAY, 2004

Indira Gandhi National Open University
School of Computer and Information Sciences

MCS - 023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS



STUDY CENTRE MANAGEMENT SYSTEM : A CASE STUDY

4

Indira Gandhi
National Open University
School of Computer and
Information Sciences

MCS-023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

Block

4

STUDY CENTRE MANAGEMENT SYSTEM: A CASE STUDY

Programme / Course Design Committee

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Prof. M. Balakrishnan, IIT, Delhi
Prof. Harish Karnick, IIT, Kanpur
Prof. C. Pandurangan, IIT, Madras
Dr. Om Vikas, Sr. Director,
Ministry of CIT, Delhi
Prof. P. S. Grover, Sr. Consultant,
SOCIS, IGNOU

Faculty of School of Computer and Information Sciences

Shri Shashi Bhushan
Shri Akshay Kumar
Prof. Manohar Lal
Shri V.V. Subrahmanyam
Shri P.Venkata Suresh

Block Preparation Team

Mr Milind Mahajani (Content Editor)
Software Consultant
New Delhi

Prof. (Retd.) A.K.Verma
(Language Editor)
New Delhi

Ms. Suman Madan
Delhi University
New Delhi

Course Coordinator: Shri Akshay Kumar

Block Production Team

Shri T.R. Manoj, Section Officer (Pub.) and Shri H.K. Som, Consultant

Acknowledgement

To Prof. Parvin Sinclair for help in the coordination of this block.

May, 2005

ISBN—

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Director, SOCIS.

STUDY CENTRE MANAGEMENT SYSTEM: A CASE STUDY

1.0 INTRODUCTION

In the first two blocks of this course, we have mainly focused on the basic concepts of Database Management System, including Relational Model, ER Model, integrity constraints, normalization, SQL, transactions and many more concepts. These blocks are an attempt to enhance your basic information with some practical inputs.

Block 3 was devoted mainly to the process of an application development without giving much emphasis to systems analysis and design.
This block, Block 4, tries to covers an almost complete application development process along with the required security, based on your familiarity by now with all the

basic DBMS concepts. However, the implementation details in this block are needed to be worked out by you.

Please note that the case study here is small in nature. However, it opens up several development related issues for us. The strength of DBMS as one of the key ingredients of Industry can broadly be visualized from this case study. This case study is an attempt to bring you closer to industry as far as a student projects are concerned. The project/ software development provides you some theoretical and practical aspects of a software project development life cycle.

This block has five sections. Section 1 introduces the case study; Section 2 introduces the software and various models in details and provides an executive summary of the software being developed. Section 3 provides complete details on the Analysis of the proposed software development process. Section 4 lists the design of the software being developed and finally Section 5 provides details on implementation, validation checks and maintenance related issues.

Please note that we have not followed any standard methodology used in the industry. However, this project provides a first good point to move towards the industry standards.

Some of the main learning points of this block are the detailed description of a typical process. However, it should be noted that this is not an ideal process in any way. The industry follows many different standards, which you may study in your 4th Semester.

The tools used here are VB and SQL Server and only a sample code of VB is shown.

Ideally this case study is suited for finding errors, process related problems. But you get a detailed view of how DBMS forms part of an overall system.

Some suggestions about this case study:

- This case study does not attempt industry oriented process.
- There may be analysis/ design/ implementation errors. Discuss these in the group counselling sessions.
- You can improve and implement in software of your choice.

OBJECTIVES

After going through the case study, you should be able to:

- define basic software development issues;
- make simple table and graphical designs;
- describe models like DFDs/ ERDs, and
- revise your DBMS and System Analysis Design Concepts.

2.0 INTRODUCTION TO SOFTWARE

In this section we will discuss some of the basic thoughts about the software that is being developed. The section starts with an executive summary and provides overview of the study centre and the existing system.

2.1 Executive Summary

The software **Study Centre Management System** for IGNOU's Study Centre is a well-organised software developed to provide the user to get relief off from the manual system as it is basically automatized version of management. The software with its capabilities has ability to computerize the whole system. The system has features that are vital to the user such as Report Generation for the Students Marks, fees, attendance, etc. Manually creating a report takes hours but the built-in

functionality in the software allows user to create different reports in a few clicks and types. Of course it requires data entry to computer, yet this data if structured properly can be reused for many different reports.

Some of the key features of this system are:

Auto-complete feature allows user to enter the data in a quicker and faster way with few letter types as well as enhanced validation checks, and maintains the data integrity and consistency.

Backup and Recovery System allows user to take the backup copies of the database and recovery facility and allows recovering the system in case of system failure.

Color Themes and Background Pictures allow user to totally customize the look and feel of the software according to their demand in the same way as there are color themes in Windows.

Help to the operator is provided in terms of Tips Box and user manual so that user could get immediately help support from the software. Also the on-line help, i.e., mini-help on status bar, allows user to quickly know, the use of various forms.

The software has enough facilities for which it can be implemented and used. It provides various advance functionalities, which attracts user to use it. Its look is very attractive which makes the user comfortable to work on it.

That's all of the software which we have built. Now let us have a very brief diagrammatic representation of role of study centre in IGNOU

The figure describes the basic process of delivery of IGNOU courses to the students. The role of study centre is primarily in the student interface input, where they come in contact with the centre of their choice. All of you by now must be familiar with the system.

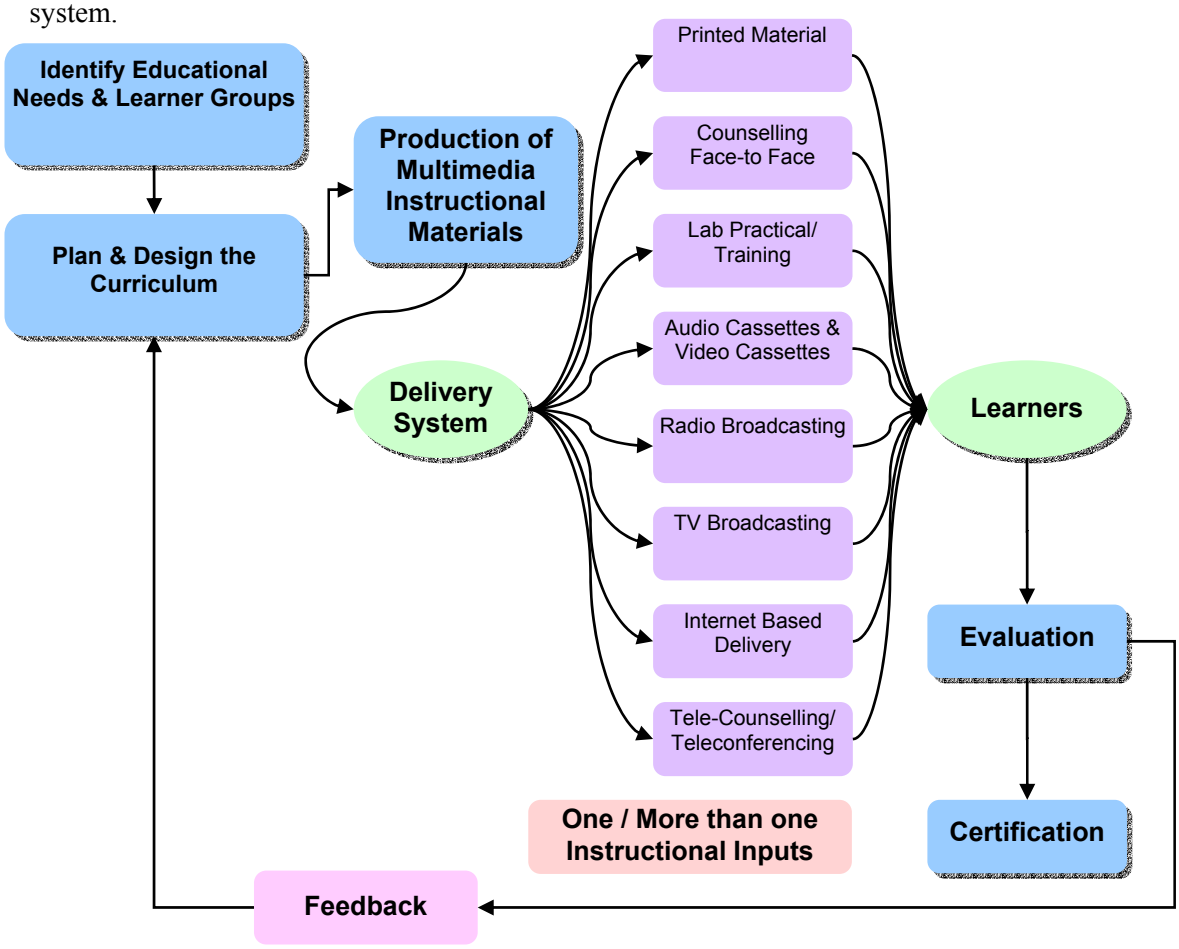


Figure 1: Diagrammatic representation of the instructional system of IGNOU

2.2 Brief Overview of Study Centre

In this system we are stressing more on Study centres because software is to be developed for the Administrative Management of Study Centres. We will discuss in detail its aim, goals and visions in later sections. At present let us have a brief idea of the Study Centres.

The study centres are the student support centres of IGNOU and are controlled by the Regional Centres. The students are serviced by effective networks of 48 Regional Centres and 1068 Study Centres located all over the country. With this large cross network, IGNOU endeavours to provide seamless access to higher education using innovative technologies and methodologies.

Now, let us see the various study centres categorised according to their working:

- Regular Study Centres:
- Programme Study Centres
- Recognised Study Centres
- Sub Study Centres
- Special Study Centres
- Study Centres (North Bihar Project)
- CWDL Down Link Centres
- DLF (Distance Learning Facilitator)
- Army Recognised Centres
- Navy Recognised Study Centres.

Now, let us point out the various functionalities and dimensions of a Study Centre.

Some of the major functions are as follows:

- Appointment of Local Tutors / Counsellors.
- Academic Support through face-to-face services
- Conduct of Practical / Hands on Experience.
- Library Facilities
- Teaching through Media
- Individual Teaching / Counselling
- Evaluation of TMAs
- Maintenance of Learner Records
- Venue for Peer-Group Interaction
- Conduction of Term End Examinations
- Co-ordination of Staff Members.

The main entities of a Study Centre are as follows:

- **Coordinator of study centre**: S/he is one of the executives who will look after the working of the Study Centre and its functionality.
- **Counsellors/ Teachers or Faculty Members:** The teachers are also an important part of the SC as they will use the proposed software and would evaluate students' performance.
- **Different Programmes and Courses:** Programmes and their courses are also an important part of SC. As there are a large number of courses, we need proper handling, so we need the proposed system.
- **Students:** They are also an important part of a Study Centre.

Programmes/ Courses on offer:

A study centre may offer many UG and PG Programmes/ courses as offered by IGNOU.

2.3 Brief overview of The Software

In this section we like to provide a brief information on our software which is named as Study Centre Management System. As the name suggests, the software is for the administration of the Study Centre. We hope that you know about the study centre functioning. So let us have a look at the functionalities, financial planning, courses offered, and administrative support provided by the study centre, which we are going to computerize through this software project. Now let us study each aspect in more detail.

Please note that our attempt is not to replicate the functioning of a single type of study centre of IGNOU. We have added some features, which may be desirable in many different study centres being considered for this software development project.

The main functionalities of our project are the maintenance of all records of Courses, Teachers, Faculty and Students. These are:

- Study Centre Details
- Teacher Details
- Students Affiliated and their Marks, Fees, Attendance Records
- Course Structure
- Schedules Maintenance
- Assignment Submission
- Study Centre Allotment Facility
- HTML Reports Creation of Marks, Fees, Attendance, etc.
- Query about all the Resources available in Study Centre.

A detailed cost benefit analysis may be difficult to work out as many of the benefits are more on the side of providing better service to the students. In this matter this system would definitely be more advantageous. Thus, a study centre person's work now can be focused on intellectual abilities rather than wasting time on doing trivial work. This provides great benefits to the students and staff.

Figure 2 shows a very brief plan for the development of the software project. It is a Gantt chart showing various important activities of the Life cycle for the software project development.

2.3.1 Long-term model for study centre

The student at an Open University undertakes a methodology of instruction which is different from that of conventional universities. The University follows a multimedia approach for instruction. It comprises:

- Self - instructional written material
- Audio-Visual Material Aids
- Counselling Sessions at the Study Centres
- Live one-way video two-way audio Teleconferences
- Practical / Project Work at the Study Centres
- Repeated telecast on Gyan Darshan Education Channel
- Interactive Radio-Counselling through Gyan Vani the Radio Channel.

A student primarily visits the Study Centre for watching audio-video programmes, counselling sessions, live teleconferences (as they are on specific channel at present not widely telecast by cable operators), and practical work. However, in the long run it is expected that IGNOU would reach the home of the students through its Teleconferencing, Radio sessions, and using EDUSAT. Then the role of the Study

Centre in such cases may primarily be to provide on-line Administrative support and practical work.

Figure 2: Gantt Chart of Project

2.3.2 Aim of the Software Development Project

Whenever a Project is made its Scope is determined − whether it will last for a long period of time or short. So, in this section we are going to explain the aim of our project keeping in mind that our software will have adequate life.

The aim of our project can be categorised by the computerization of the following tasks:

- **Record Maintenance of Study Centre Details:** Maintenance of Study Centre Details means that we are going to keep the details of the Study Centre in which our project is currently implemented.
- **Record Maintenance of Teacher Details, Study Centre Resources:** In this section we are going to keep the teacher's complete bio-data which will enable students to find out about their teacher's capabilities, mastery, experiences and knowledge.
- **Record Maintenance of Students Affiliated:** In this we are going to keep complete details of the students of the Study Centre. Using these details students will be frequently asked for suggestions.
- **Programme and its Course Structure:** Complete details about programme structure, its fees, eligibility, duration and many other details which would enable students to select the programme of their choice.
- **Maintenance of Fees, Marks and Attendance details of Students:** Although the fee at present is not being deposited at the study centre, in the long term it may be the case, if every data transfer is sound and secure.

- **Schedule Creation for Classes**
- **FAQ (Frequently Asked Questions) Facility:** This section will enable students to find out answers to their queries. It would contain Frequently Asked Questions with their answers. Using this facility, students can find out the solution of their most common problems during admission and study phase.
- **Report Generation of all the Resources available in Study Centre and Student Support Services like marks, fees, attendance report.**

Some of the standard features of this software project are as follows:

- *Backup/Recovery System.*
- *Automatic creation of Database on a new System.*
- *Multi-user Environment Facility: Support for concurrent transactions.*
- *Password Facility for Administrator and Teachers: Support for security.*

2.3.3 Present working system

At present our system has lots of manual work which needs to be computerized. So, lots of paper work is required. In order to minimize this we need to computerize all those departments which require record maintenance. The different points which illustrate the present working of the System are:

- All the different administrative department works require paper work which includes the record maintenance of current Study Centre, Students Affiliated, Teacher Details or Faculty Details.
- Fees of the students include submission of the form which is duly filled by the student (This feature is not there in IGNOU system but we are assuming it from the long-term plan point of view).
- Records of Marks of Assignment, Theory and Practical are maintained on the paper.
- No FAQ facility, so students who have some confusions and problems regarding courses structure and fees need to talk to specific functionaries who are normally not available on weekdays at study centres as most of the Study Centres operate on Saturdays and Sundays only.
- They are not able to find out their teacher qualifications and experiences.

2.3.4 Proposed working

After the implementation of this project, there would be lots of benefits as there would be less manual work, more computerized working, resulting in faster resolution of students' queries.

It would provide all the goals and aims which we have discussed in the earlier sections. So, making the present system faster, quicker, more active and user friendly to the students, teachers and study centre staff members. The system will become more responsive and fast.

2.4 Software life cycle

In this sub-section, we are going to list the procedure we are going to use in order to build our project. The details of this methodology are defined in the System Analysis Courses and Software Engineering Courses. The term Project Methodology suggests the processes, structures, method or style for building our Project.

Software Life Cycle consists of the set of activities which the analysts, designers and users carry on to develop an information system. Let us have a brief overview of the stages. It consists of following stages:

Preliminary investigation: It consists of request clarification, feasibility study

(software being technical, economical and operational) and approval of request.

Determination of requirements: It consists of the following questions which are needed to be answered:

- What is being done?
- How is it being done?
- How frequently does it occur?
- How great is the volume of transactions?
- How well is the task being performed?
- Does a problem exist?
- If problems exist, the level of seriousness of the problem.
- If problems exist, the underlying cause.

In order to answer these questions the System Analyst will do detailed investigation by study and observation of data collected using fact finding techniques like interviews, questionnaires, etc.

Development of Prototype System: It consists of developing models of the requested task, object oriented methods for easier programming and 4th Generation Language Technologies.

Design of System: It consists of elements of the task to be designed, the design of the report to be printed, the design of input to be provided and design of files.

Development of Software: It consists of the language used, the file description and the user friendliness of the software.

System Testing: It consists of different types of testing of the software including unit testing, system testing and validation testing and user acceptance.

System Implementation: It consists of the training needed by the user, conversion methods, system review and maintenance.

System Maintenance: It requires trained, skilled persons who have good knowledge of system working and of handling the system in case of any failures or adaptations.

But what process Model do we follow to develop this Project? Before answering this question let us recapitulate the different process Models, their advantages and disadvantages and then select one of the most suitable models for our domain.

2.4.1 Different models

Waterfall Model

It consists of a linear set of distinct phases including requirement analysis, specification, design, coding, testing and implementation. It can be represented as in *Figure 3*. Please note the use of two terms, verification and validation, in the figure. Verification is defined as the question "Are we building the product right?" Validation is defined as the question "Are we building the right product?"

Features of the waterfall model:

- Systematic and linear approach towards software development.
- Each phase is distinct.
- Design and implementation phase only after analysis is over.
- Proper feedback, to minimize the rework.

Drawbacks of the waterfall model:

- Difficult for the customer to state all the requirements in advance.
- Difficult to estimate the resources, with limited information.
- Actual feedback is always after the system is delivered. Thus, it is expensive to make changes during the later stages of software development.
- Changes are not anticipated.

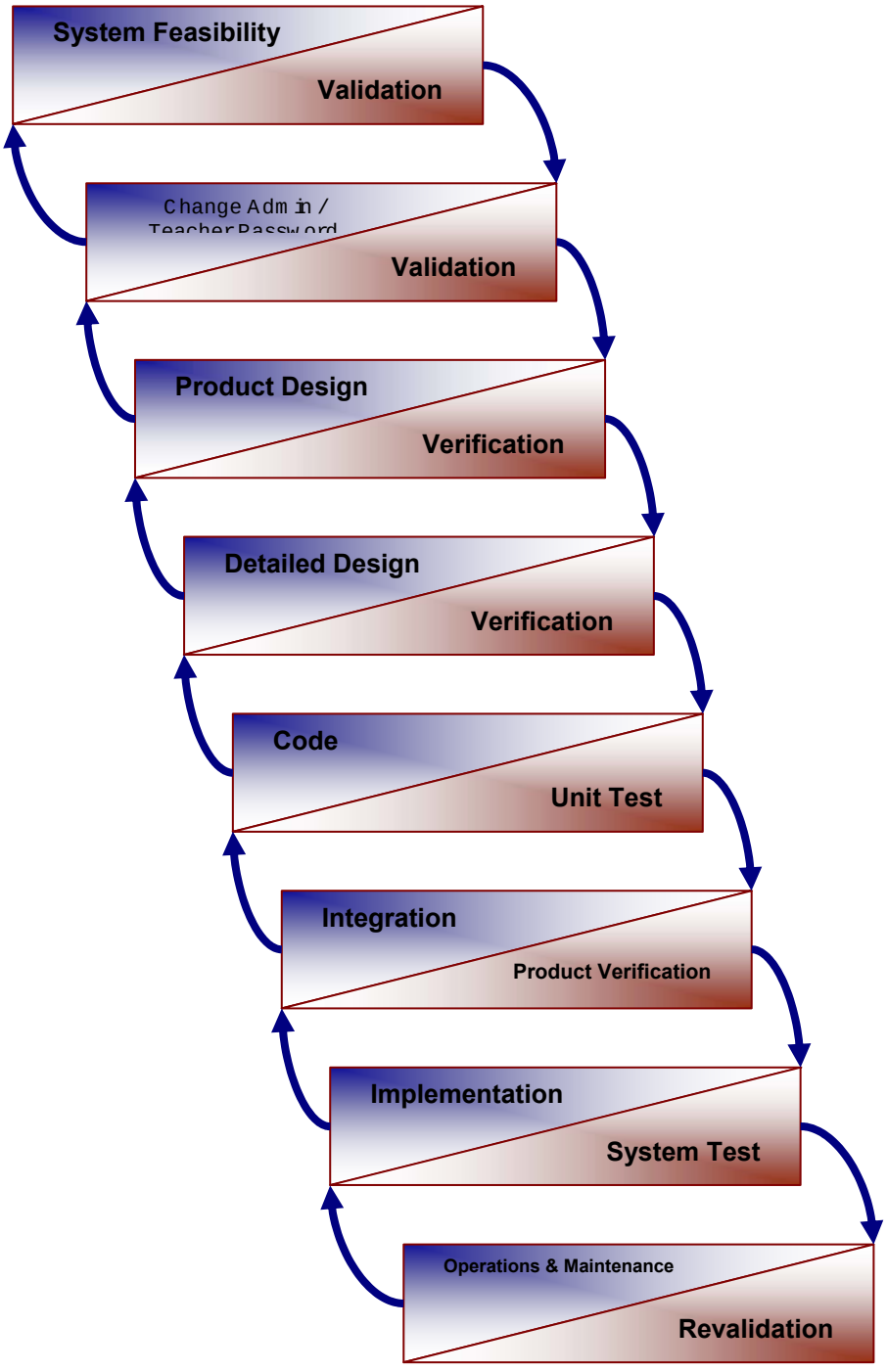


Figure 3: The Waterfall Model

Rapid Application Development Model

RAD is more like a set of parallel waterfalls where the system can be decomposed into largely independent subsystems. It includes a broad array of software tools that automatically generate source code as per the developer's specification.

Prototype Model

It consists of distinct phases including its prototype, for example, demonstration that enables the customer to get a feel of the system. It can be represented as in *Figure 4.*

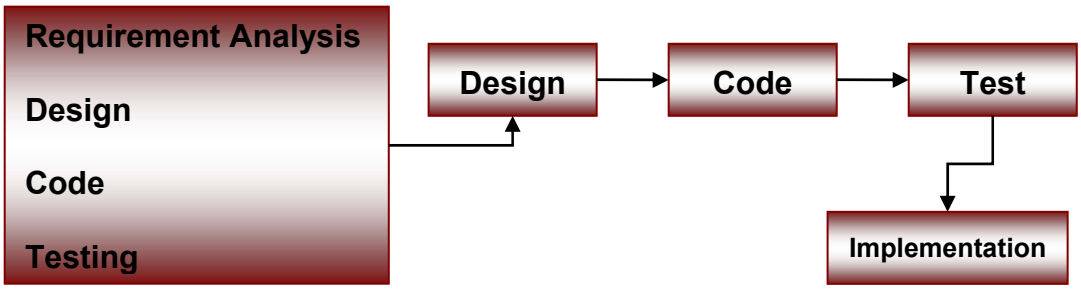


Figure 4: The Prototype Model

Features of the prototype model:

- Reduced functionality.
- Begins with requirements gathering.
- Used in large systems.
- Makes use of existing programs.
- Reduces the risk of project failure.

Drawbacks of the prototype model:

- Customers feel that the cost incurred or provided is too high for the model.
- Developer might use inefficient algorithm during prototype development, which may not ultimately be changed due to customer's non-cooperation.

Iterative Enhancement Model

It consists of distinct phases in which the steps after the designing steps are repeated if the customer is not satisfied. The demerit of this model is that the iteration may never end thus the user can never get the "final" product. It can be represented as *Figure 5.*

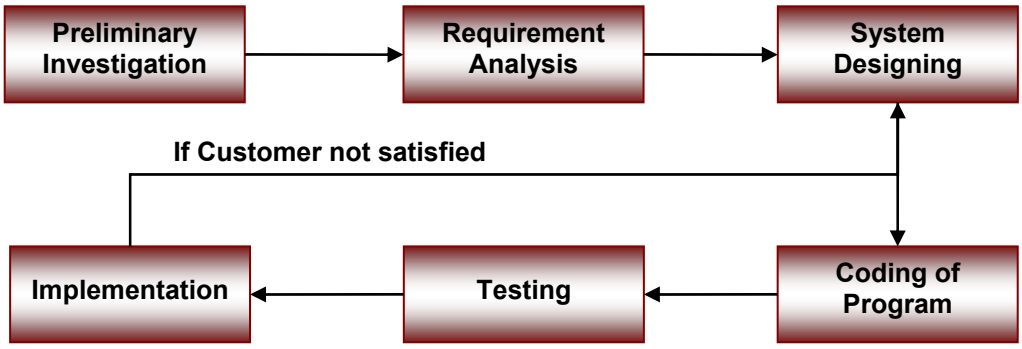


Figure 5: Iterative Enhancement Model

Spiral Model

It consists of distinct phases including both the prototype model and the classic life cycle model. It includes six major phases which are customer communication activities named planning, construction risk analysis, engineering and release customer evaluation. This model can be represented as *Figure 6.*

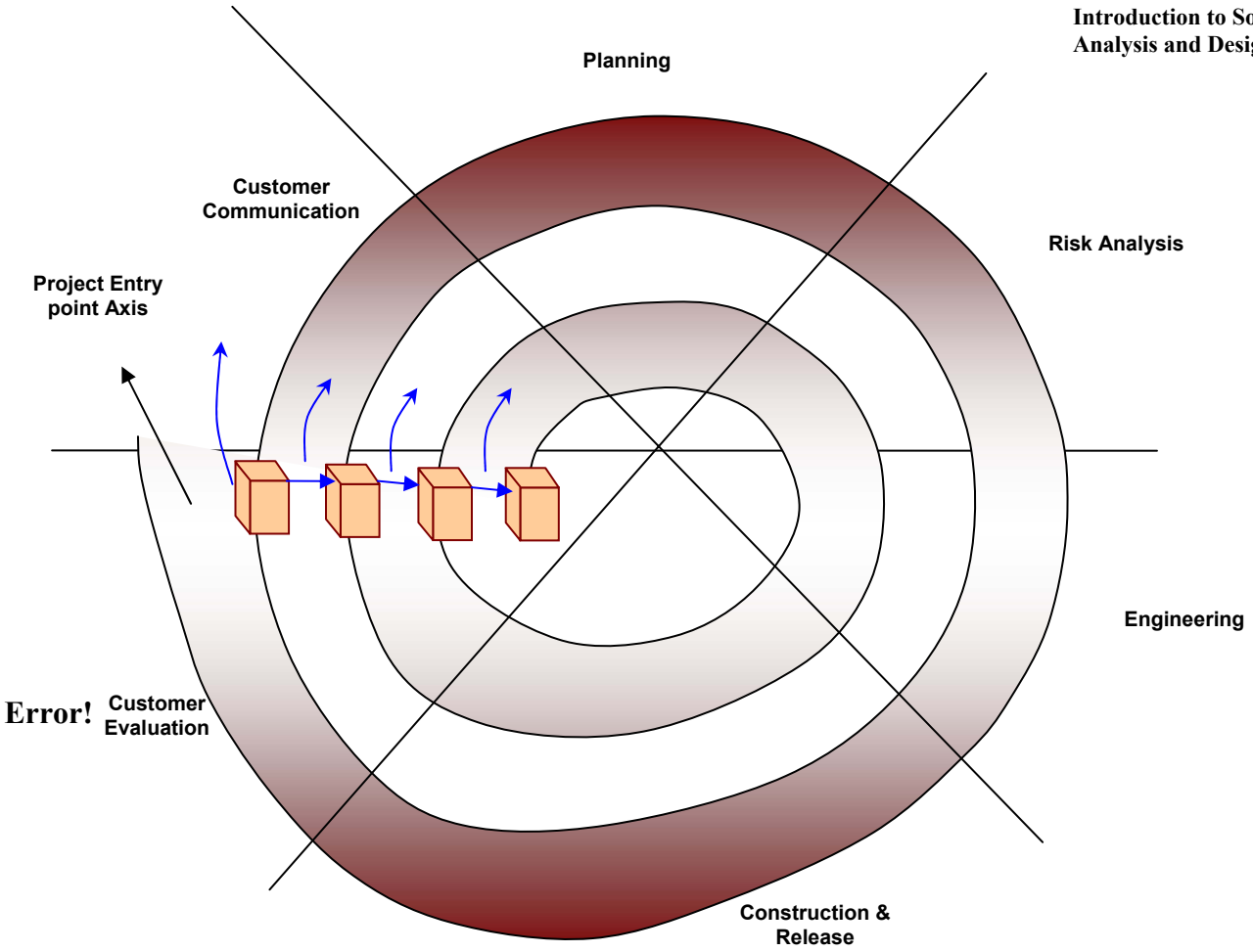


Figure 6: The Spiral Model

Features of the Spiral model:

- Risk assessment is involved at every activity.
- Most realistic approach to development for large systems.

Drawbacks of the Spiral model:

- Risk assessment involves great expertise.
- Comparatively a new approach.

Fourth Generation Techniques

It covers a broad array of software tools that have one thing in common. They are considered to be the high level languages, as they serve the users in a very friendly way. Based upon the specifications given by the customer, the tools available automatically generate the source code. 4GT's include Non-procedural languages for database query, report generators, menu generation, screen generation, code generation, graphics ability, and spreadsheet capability and even more. These tools exist but for very specific application domains.

Features of the 4GT's model:

- Non-procedural language.
- Build application fourth generation language (4GL's).
- Tools for making report and menu and screen generation of the required program.

- Faster query handling.
- Graphics ability and spreadsheet ability in the required program.
- Conversion of requirements gathering to implementation using fourth generation language (4GL's).
- Good for business information system.

Drawbacks of the 4GT's model:

- Customer is not assured of what is required in advance.
- Customer might not be able to specify the facts properly.
- Not good for big systems.
- Poor maintainability, quality, customer acceptance.

2.4.2 Used model and its justification

Spiral Model

As we know, in the **waterfall model** we cannot perform the next step without doing the previous steps because each step of the waterfall model is **dependent on each other**. However, the spiral model allows use of the features of prototype and incremental model which allow us to divide the project into increments. So we can increment the on-going project after some time, i.e., we can add some new features to the project or we can add some new division or some new functionality in our project. So the chances of failure of the project by using this model is minimal because some result is obtained after the completion of the 1st increment which is generally not possible in the waterfall model. If we want to change some feature or functionality or add some functionality, it can be done by this model. In the waterfall model we cannot do this because repetitive changing may cause the process to restart again and again.

The next best feature of this model is that it is an ongoing process for the lifetime of the project. This means that the project cannot die till the user is satisfied.

3.0 SOFTWARE DEVELOPMENT PROCESS : ANALYSIS

Building an information system or software system is not merely putting the computer instructions together but it involves uses of software engineering process methods and tools. Until software system is implemented and delivered to a customer, and even after that, the system undergoes gradual development and evolution.

We have already discussed the stages related to our project in the previous sections. Let us first discuss the type of this system. The system which is being developed by us involves transaction processing and Management decision system. The system includes processing data. **Transaction Processing System** aims at improving the day to day business activities operations on which the organisation depends. Some of the transactions for the proposed system may involve online updating of fee records, marks, attendance, etc. The sole objective behind the system is to increase the efficiency, speed and accuracy of processing on the voluminous data. So, all these facilities result in greater efficiency and quicker response time of the system.

The users of the system can be categorized into administrator, teachers and students. The following *Figure7* shows the type of users and their rights. The classifications of users of a system are best explained by the Neumann Haddass Model which illustrates the various levels of management which looks after the planning and controls of the system. Let us have a brief overview first of the Neumann Haddass model and then of its application to this system. *Figure 7* shows various users of the system.

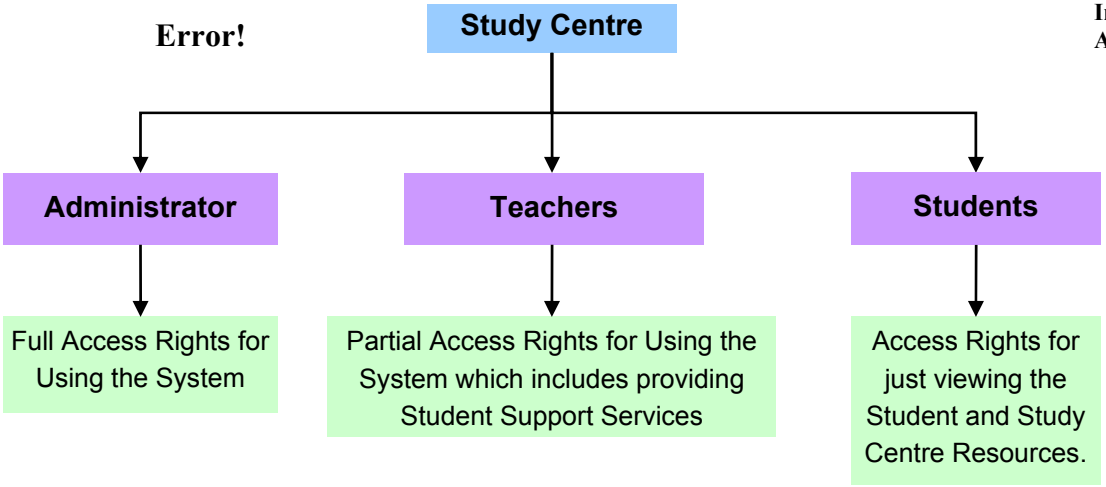


Figure 7: Users of the Proposed System

Neumann Haddass Model

It states that the organisational chart can be divided into three levels of management who look after three different types of planning and control. It is represented by the tree or triangular interdependent systems at different levels. Refer to Figure 8.

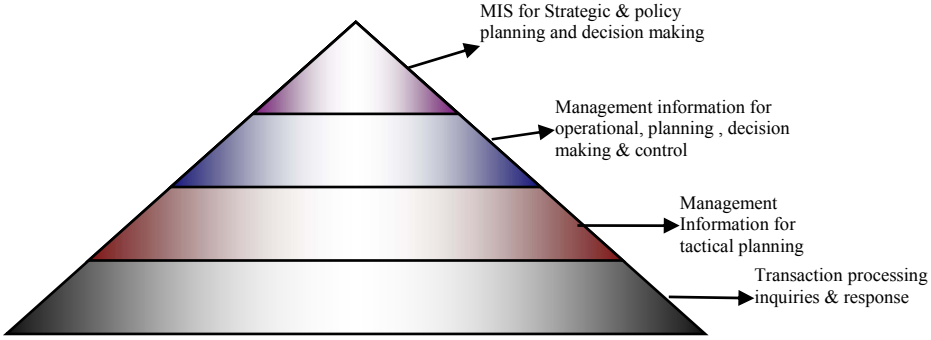


Figure 8: Neumann Haddass Model

The top-level management looks after strategic planning which determines long-term goals and the purpose of the organisation. It also identifies the resource needed to achieve the goal, the government rules, etc.

The second one is the middle level management which monitors the use of resources identified by the top level management. They look after the people, money, equipments, etc. to achieve the goal. They are very important and the progress and speed of working depends on them, not completely but substantially.

The third one is the lower level management. Its working is similar to that of middle level but at a lower level. They look after the day-to-day working and progress of the individual.

In our case, the Top level Management is under the administrator who is either the Study Centre Coordinator or Principal. Next, in the middle are the teachers, or faculty members. However, they sometimes work at lower levels also. The students come at the bottom level or lower level. They can avail of the facilities provided by the system.

Now, what is the use of this analysis phase? It allows you to examine the eight basic characteristics of the system which are:

1. **Goal**: Each system is developed for a reason. It has some purpose for its existence.

2. **Inputs**: The inputs to the system are the resources like manpower, capital, materials, equipment, etc.

3. **Output**: These follow from the system. They are usually goods (may be furniture from a factory or leather handbags from a workshop) or services (legal consultation from a lawyer's office or a project feasibility report from a management consultancy firm).

4. **Boundaries**: Each system has boundaries. In other words, each system has a definite scope. We may wish to include everything in one system but that is practically impossible. Each system has some definable limits. Remember the user's wish list? Some items in the wish list may be outside the scope of the system.

5. **Environment**: Each system exists in an environment that can influence the system in a number of ways, such as making it more or less difficult to perform.

6. **Components**: The components of a system transform the inputs into outputs and the linking together of the components is accomplished by interrelations.

7. **Interrelations**: These describe how resources pass from one system component to another.

8. **Constraints**: These represent the environmental, social, technical or behavioral factors that typically limit system performance.

You need to find out all about the system's environment, boundaries, inputs, outputs, goals, components, resources and interrelations.

Let us now list the important characteristics of the system being developed by us:

Goal – Printing of Schedules, Fees, Marks, Attendance and all Miscellaneous reports that will help better planning at study centre for student support.
Inputs – Attendance, Fees, Marks, Schedules details and all Miscellaneous details
Outputs – Schedule Creation, Student Support Services
Boundaries – Study Centre Data which includes its Resources and Students Record.
Environment – IGNOU Terms and Conditions as well as Study Centre Rules and Regulations.
Components – Various modules that we will develop.
Interrelations – Computer programs
Constraints – Data available to administrator and teachers.

3.1 Preliminary Investigation

After having a brief idea of the project, let us undertake the preliminary investigation to ascertain the feasibility of the project. Ideally this stage of analysis should be performed prior to project approval. The three most important stages of preliminary investigation are:

1. Request Clarification
2. Feasibility Study
3. Approval of Request.

3.1.1 Request Clarification

This step is a one of the component of preliminary investigation stages. Many requests from employees and users in the organisation are not clearly stated. So, it should be clear in advance what the originator wants. Problem clarification is a difficult task

because requester is making a request but how to state it or where actually the problem is or what is the problem is not clearly stated. For instance in this project the Coordinator has made the request for adding web-site facility for the study centre but what would be the contents was not stated. Thus, the cost estimation for website could not be done.

3.1.2 Feasibility study

It is an important part of the **Preliminary Investigation** because only feasible projects go to development stages. Let us do a very basic feasibility study for the current project.

1. **Technical feasibility**: Technical feasibility raises questions like, is it possible that the work can be done with the current equipment, software technology and available personnel? And if new technology is required what is the possibility that it can be developed?

   In case of our project, the software which we have built up fully supports current Windows OS but it lacks the support for other OS environment like Apple Mac, Unix and Linux. Next, it is not dependent on the number of users so it can handle a very large number of user environments (in principle, but does not happen in practice). Next, the Support for Hardware, it has full support for new hardware, so no hardware compatibilities issues arise as it requires minimum configuration but only non-Mac environment.

Minimum Hardware Support Required

Computer	Intel® or compatible Pentium 166 MHz or higher.
Memory (RAM)[i]	64 MB minimum on Windows 2000, 32 MB minimum on all other operating systems 128 MB or more recommended
Hard disk space[ii]	Minimum Space for Software 10 MB
Monitor	VGA or higher resolution 800×600 or higher resolution required for the Software Project
Pointing device	Microsoft Mouse or compatible
CD-ROM drive	Required for Installation

(i) Additional memory may be required, depending on operating system requirements.
(ii) Actual requirements will vary based on your system configuration and the applications and features you choose to install as well as SQL Server Space Allocation.

2. **Economic Feasibility**: It deals with economical impacts of the system on the environment it is used, i.e., benefits in creating the system.

   In case of our project we are assuming an economically feasible solution.

3. **Operational Feasibility**: It deals with the user friendliness of the system, i.e., will the system be used if it is developed and implemented? or will there be resistance from the users?

   In case of our Project we have done the Operational Feasibility Study with Centre Coordination and Incharge about the usage of the project, and regarding user friendliness we have tried our best to make the software highly user friendly so that a person having only a little knowledge of English can handle it. By the way we have also built on-line as well as special help programs which help in training the user. Also one of the interesting parts of the Project is **Tips of the Day** which

gives some special Tips to the user for proper functioning.

4. **Time Feasibility**: In this type of feasibility study, we examine whether our proposed project can be completed in the specified time frame or not. In our case, our project is finished in the targeted time-frame. So, it is feasible regarding time scope.

5. **Legal Feasibility**: This type of feasibility evaluates whether our project breaks any law or not. According to our analysis, our project doesn't break any laws. So, it is legally feasible too.

3.1.3 Request approval

Not all the requested projects are desirable or feasible. And only those which are desirable or feasible should be put into the schedule. In some of the cases the development can start immediately but in some of the other cases, the staff members are busy on their ongoing projects. Many business organisations develop information systems as carefully as they plan for new products, or new manufacturing programs. After a request is approved its cost, priority, completion time and personnel requirements are estimated.

For example, in our case the management who will be investing the money might feel that there is no requirement of such a system, so unless they approve it work cannot start. However, this project does not have such constraints, as it is just a sample project.

3.2 Requirements Determination

Any software project can succeed only if it has been fully analysed in this stage of Requirements Determination, since Requirement Determination is the heart of the systems analysis, aimed at acquiring a detailed description of all important areas of business that is under investigation. Analysts working closely with employees and managers must study the complete business process for the designers of the system.

In this stage, the system analyst will talk to a variety of people in detail and will take their ideas and opinions. We use fact-finding techniques like Questionnaires to collect the information from people who cannot be interviewed individually and interviews with special appointments for discussion on the project. The detailed investigation includes:

1. Study of manuals and reports.
2. Actual observations of work activities.
3. Collection of sample forms and documents.

In our case we have several interviews with Mr. Goel, centre incharge, about the project discussions. He helped us regarding this project on what to build in it, what are user requirements, etc. We also got ideas from the students who may be users of the system.

Now let us have a brief overview of the fact-finding techniques of the project which we used.

The most commonly used techniques of fact finding by us are as follows:

- Existing documentation, forms, file and records
- Research and site visits
- Observation of the work environment
- Questionnaires
- Interviews and group work sessions.

All these techniques helped us a very great deal in finding out the information related to the project. Let us first have a glance at all these techniques.

3.3 Study of Existing Documentation

You may feel that getting existing information may not be right since you are there to create a new computerized system. So why bother about what happens at present? Think back to the earlier session. What did we say? To build a computer-based system, it is essential to first clearly understand what happens right now. You will be able to build a system only after a correct understanding of the current system. You may even improve it.

So, existing information is absolutely essential. It will give you a clear picture of what actually happens in the department or organisation being studied.

For each document, it is important to know its use in the system. To understand this, it is useful to prepare the document-department routing grid.

So talking about getting existing information, what are the advantages of this technique?

- You don't have to disturb any person while they are working. You can take the files to an empty room and study them at your leisure.
- There is no time constraint. This does not mean that you take 3 months just to study the existing documentation. What it means is that there is no time deadline like an interview. If your interview carries on for 2-3 hours, you are going to have a very unhappy user. Whereas if you spend 2-3 hours studying the document, you will have bothered no one.
- You will know exactly what happens and not what the user wants you to know. You will be able to study the processes free from the bias and personal viewpoint of the user. You will be unaffected by the users' feelings or personal office politics and get an impartial view of the system.

We are talking about getting existing documentation but what are the documents that you should seek? Some of the documents looked into by us were:

After this you can collect documents like:

- Class schedules, faculty schedules.
- Minutes of important meetings.
- Accounting records of the students
- Assignments submission of return records
- Making system and results communication
- Attendance record, student batches record, infrastructure etc.

Let us look at some of the above documents more closely and see how they can help an analyst.

3.4 Completed Forms

The most important documents are the **completed forms** representing actual transactions. Remember that blank forms should not be accepted. This is emphasized because commonly, users prefer to tear out a blank sheet from their file or register when they are asked for a form. A blank form will tell you nothing about the kind of information that is actually filled in the columns. The form design may be obsolete and the users may have devised their own forms within the framework of the old one. The heading to the column may be totally misleading and the user may actually be entering something different out there.

We have said that we should study only completed forms. All right, but how many should we study? Just 1 or 10 or 100 or 1000? What is the correct sample to study?

There exists an empirical formula by which you can find the sample size to be certain

of the quality of data. In our case a sample size of 25-50 forms is ok.

Then comes the question – how do we choose the sample?

There are 2 ways.

The first is **randomization,** i.e., you pick up any 50 documents and use them as a sample.

The other approach is **stratification,** that is, a systematic method for choosing the sample. Let us take an example. Suppose the total number of student documents we have is 5,000. We have to choose 50 documents in our sample. So we choose every 100th document which is 5000/50.

3.5 Reports (Manual and Computerized)

All **manual and computerized reports** must be collected. This may sound easier than it actually is because many users devise their own reports for follow-up or for statistical analysis. These reports are normally prepared by hand or word-processed from information available in the system. These are considered by the users as 'their' property and normally not shown to the analyst. Remember in our last section we talked about some notebook or register maintained by the user privately? Well, these are reports. Therefore, when you ask for all reports, make sure that you really get all the reports. The reports prepared manually by the user are actually the most important because they are the ones being used regularly. You may wish to modify the existing computer reports to absorb the manual report.

If the organisation has **standard operating procedures (SOPs)** in place, it makes life easier for the analyst. The analyst will know how things ought to be done and then, when the procedure is being reviewed with user, the gaps or inaccuracies will be easily identified. In case of missing SOPs, the analyst will have to write the procedures himself after identifying all responsible parties and determining the tasks appropriate to each.

It is very important to review the SOPs very carefully to verify that they are complete and accurate. Any mismatch between the actual procedures carried out and the SOPs should be identified and resolved. You may find a number of duplicate information flowing from one department to the other or a particular procedure being eliminated in actual practice or different information recorded on the same form.

Let us now see how it helped us in our project.

"You may find that a different type of form was used to record the details of Students Marks, Fees and Attendance Schedules, etc. We questioned about the formats, the conclusion may well be that no one really knew why those formats were used but they were doing it because it was always done that way."

Therefore, the point to be made is that collection of existing documentation is important because it clarifies your understanding of the current system. It also throws up gaps, missing parts and other questions about the current system which you may need to clarify with the users.

3.6 Research and Site Visits

Let us consider the next fact-finding technique, that is, research and site visits. This particular technique is not used very often because it is not relevant in many projects. Here research means **studying the application and the problem areas**. You could do this by studying the trade journals or you can visit reference rooms in libraries to find out more about the application and the problems. You can discover how others have solved similar problems and whether any tools (whether mechanical or manual) exist for solving the problems. You can visit other companies or departments in other organisations who have already gone through the analysis exercise and get some tips from them. This technique of fact finding was not employed in this project.

3.7 Observation of Work Area

This technique involves observation of place of work and work procedure. In this technique, the analyst watches a person perform certain tasks and tries to understand the system. This technique is not very commonly used.

Can you think of the reasons?

- To create a 'good impression' on the analyst, very simple or routine work may be shown to the analyst. In which case the analyst will not be able to observe the special or irregular cases.
- The tasks may be carried out at odd times, e.g., there may be a graveyard shift (night shift) which may be difficult for the analyst to observe or there may be too many interruptions.
- In case persons have been performing a task contrary to the standard operating procedures, s/he may perform it correctly temporarily while under observation.

Hawthorne Effect

People perform differently on stage as compared to real life, i.e., if someone knows that s/he is being observed, s/he becomes aware of it and subconsciously performs better. The Hawthorne Effect may prove this point.

What is the Hawthorne effect?

It has been described as the rewards you reap when you pay attention to people. The mere act of showing people that you are concerned about them usually spurs them to better job performance. That is the Hawthorne Effect.

Suppose you have taken a management trainee and given her specialised training in management skills she doesn't now possess. Without saying a word, you have given the trainee the feeling that she is so valuable to the organisation that you will spend time and money to develop her skills. She feels she is on track to the top, and that motivates her to work harder and better. The motivation is independent of any particular skills or knowledge s/he may have gained from the training session. That is the Hawthorne Effect at work.

When people spend a large portion of their time at work, they must have a sense of belonging, of being part of a team. When they do, they produce better. That is the Hawthorne Effect.

Occasionally, managers object saying that observation is not a valid test. Of course they will do a good job if you are watching them. Isn't that the Hawthorne Effect? Which is why some people do not count observation as a useful fact-finding technique.

On the other hand, there are many points in favour of observation

The main point in its favour is that the analyst is able to see exactly what is being done. Sometimes, users may find it difficult to explain complex procedures. In such cases, the analyst may get a clear idea if s/he observes the person at work.

- Observation also reveals the physical conditions (like excessive noise, poor lighting, physical layout and place of movement) which may have a bearing on the work being done and which may have been missed or inaccurately described during discussions.

To obtain maximum benefits from observation, you should be fully prepared. Ideally, the analyst should observe the work during normal hours and during peak times. The analyst should prepare proper data collection forms in which he can record the data

observed. Your findings should be documented during or immediately following the observation.

In this project the efforts were made to watch various activities of the students/teacher etc. to determine the environment of working.

3.8 Questionnaires

All of you would have received and even responded to some questionnaire.

So what is a questionnaire?

It is a document prepared for a special purpose that allows the analysts to collect information and opinions from a number of respondents.

It contains a list of questions. The questionnaire is distributed to the selected respondents; the respondents answer the questions in their own time and return the questionnaire with the answers to the analyst. The analyst can then analyze the responses and reach conclusions.

Sample Questionnaires Used by Us:

- What are the services provided by the Study Centre?
- What are the specific data of Student?
- How Schedule is created for a Teacher?
- How are Marks, Fees and Attendance recorded?
- Some of the Sample Layouts of Reports
- What are the various outputs of Systems?

3.9 Personal Interview

There are always 2 roles in the personal interview.

The analyst is the *interviewer* who is responsible for organising and conducting the interview. The other role is that of the *interviewee* who is the end-user or the manager or the decision-maker. The interviewee is asked a number of questions by the interviewer.

In this project we have done interviews of centre incharge, co-ordinator and other main functionaries to ascertain their expectations of the system.

4.0 SYSTEM DESIGNING

It is the process which starts after the completion of Analysis Phase. In this phase the planning of activity starts whereas in the Analysis phase the important data are gathered.

System Designing consists of various activities. The main activities that are being conducted here:

- Modeling Data Flow Diagram of System
- Creating Entity Relationship Diagram of System
- Creation of Database Dictionary Designing
- Database Design
- Input Form Design
- Output Forms or Reports Design.

Finally, after the completion of this phase, the development phase is performed where these designs are used to achieve the proper look and feel of the software.

4.1 Data Flow Diagram for System

It is one of the important tools of System Designing which enables software engineer to develop models of the information domain and functional domain at the same time. It serves two purposes:

1. Provides an indication of how data are transformed as they move through the system
2. Depicts the Functions (and sub-functions) that transform the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modelling of function. A description of each function presented in the DFD is contained in a **Process Specification.** As the DFD is refined into greater levels of details, the analyst performs an implicit functional decomposition of the system.

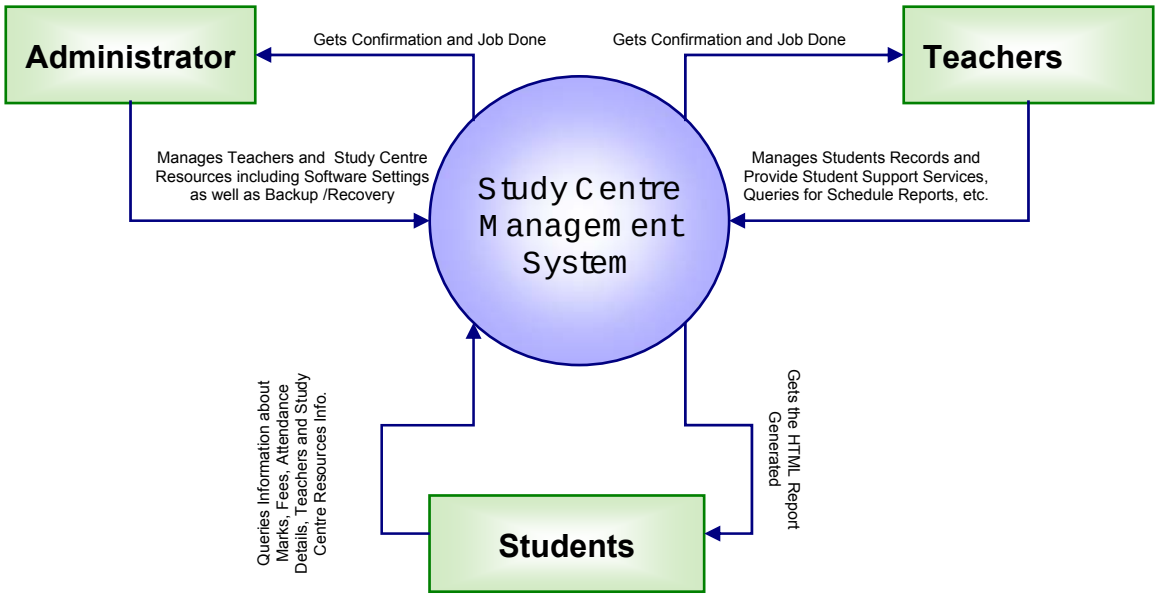
One of the faults in our process while developing the system was that the system was analysed, designed and implemented but the DFDs were developed as an afterthought. Thus, you will find the following problems in the DFDs:

- The dataflow have not been labeled in many cases.
- The processes are shown as linear processes especially in the second level DFDs
- The verification activities are being performed in sequence, although they are not practically done that way in the actual situation.

Exercise

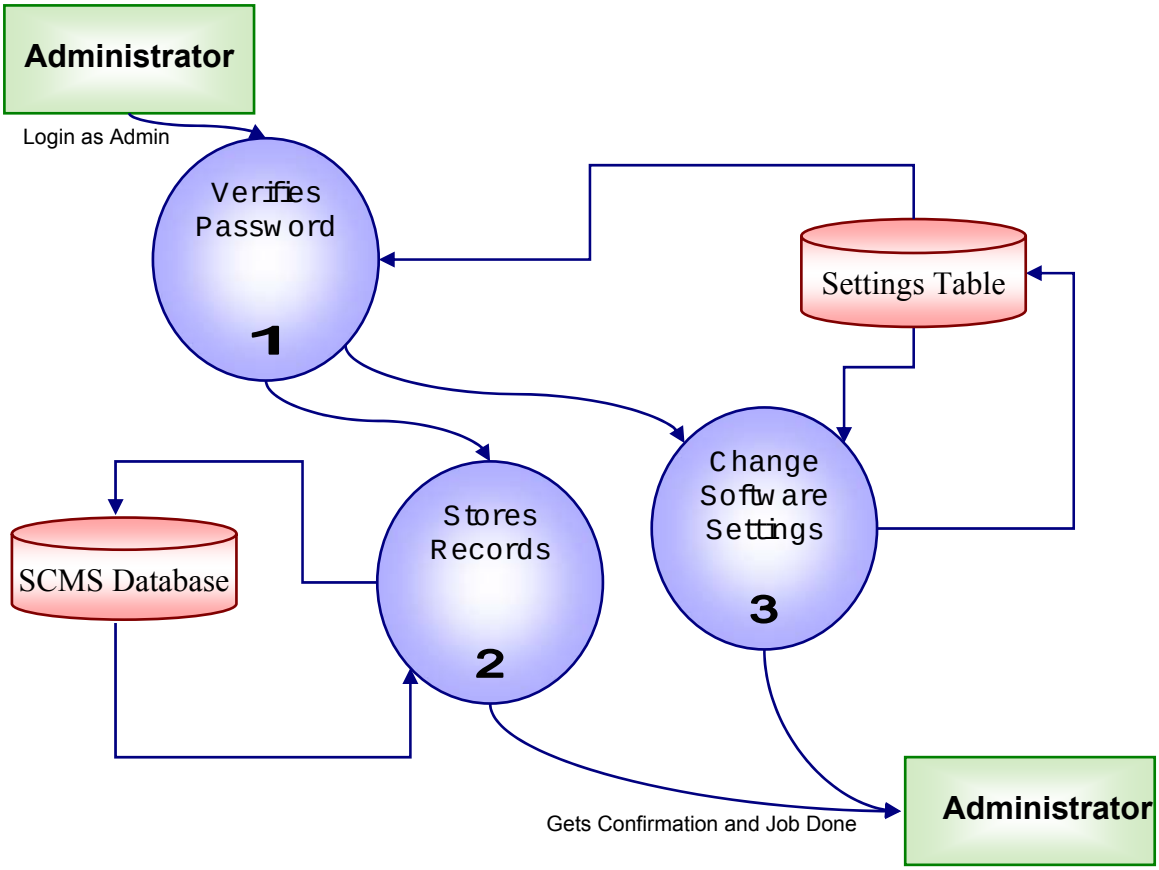
You must discuss the DFDs thoroughly in your sessions and come up with the most optimum DFDs.

0^{th} Level (Context Flow Diagram)

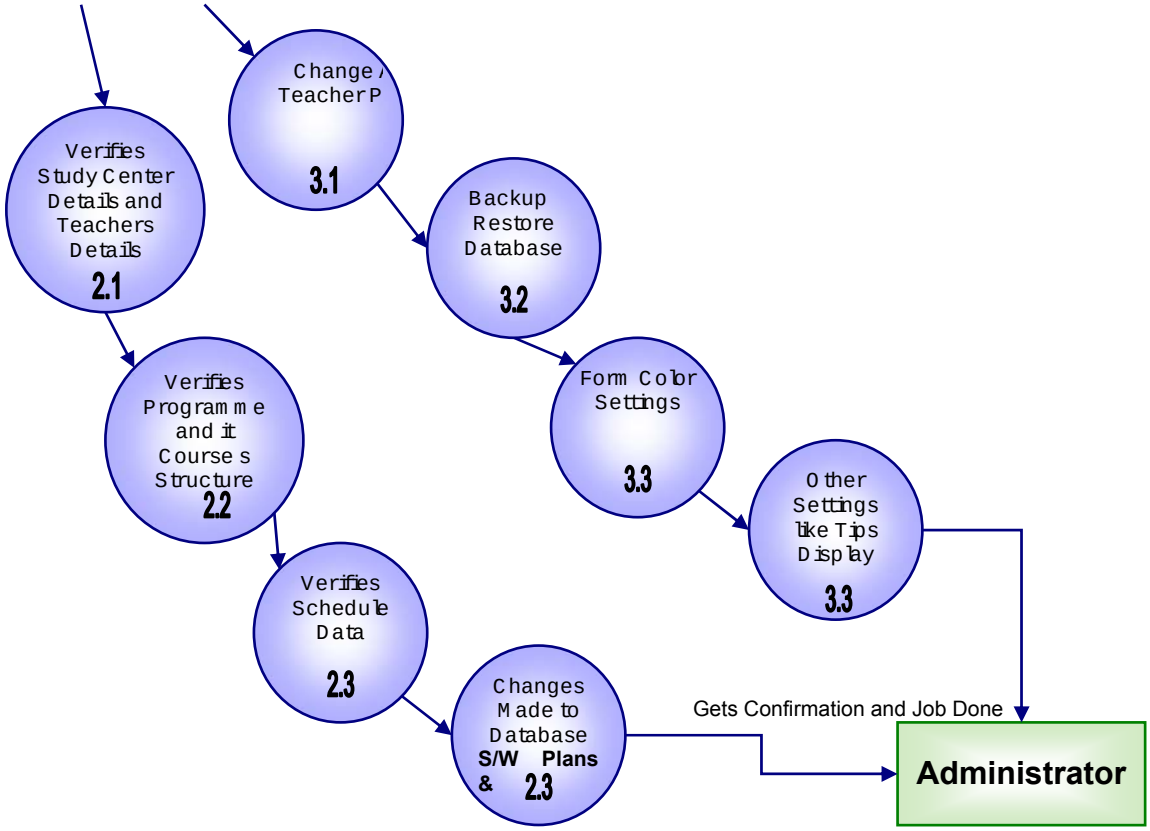


DFD For Administrator

1st Level

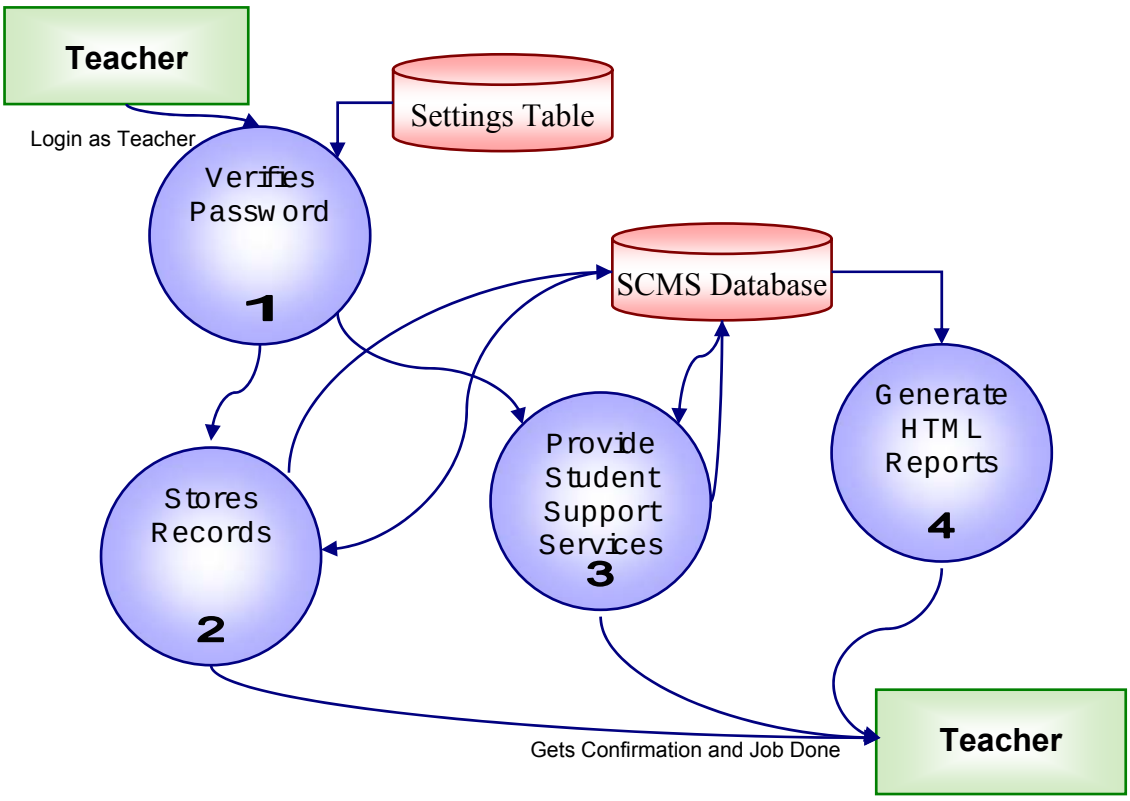


2nd Level

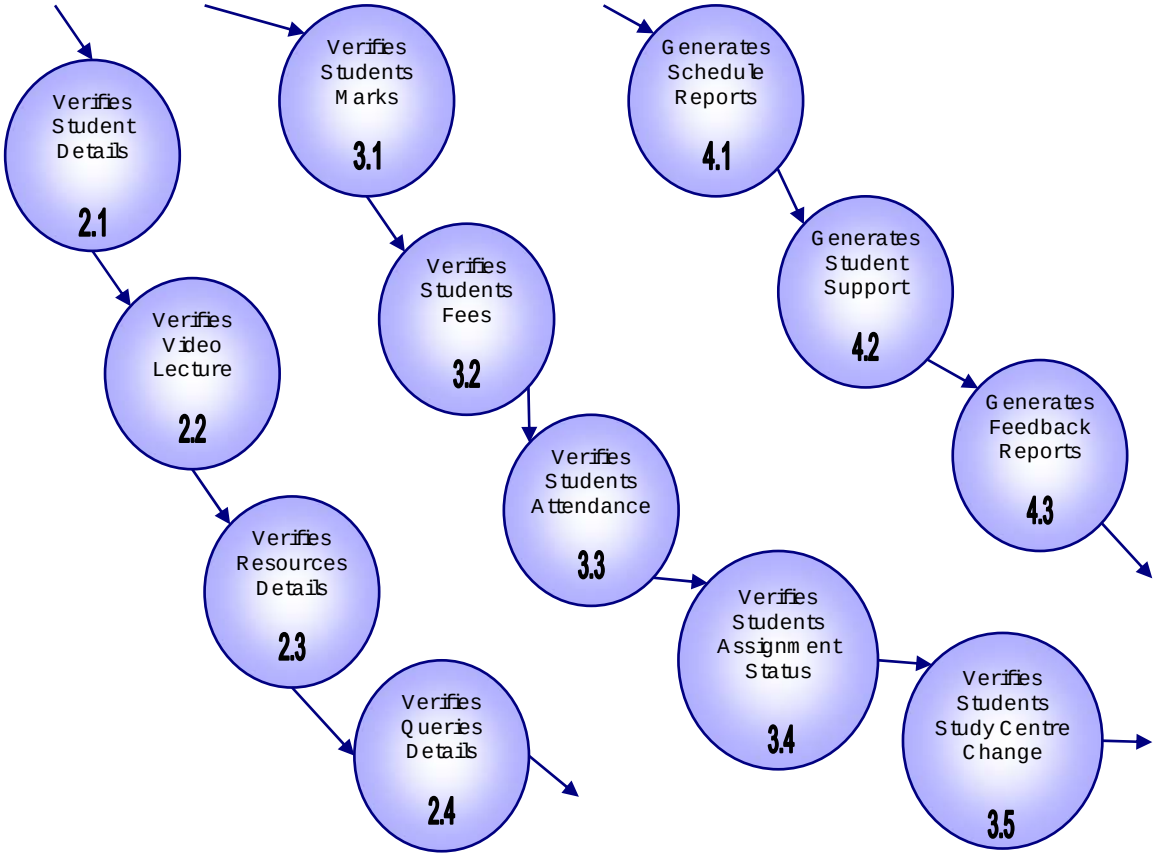


DFD for the Teacher

1st Level

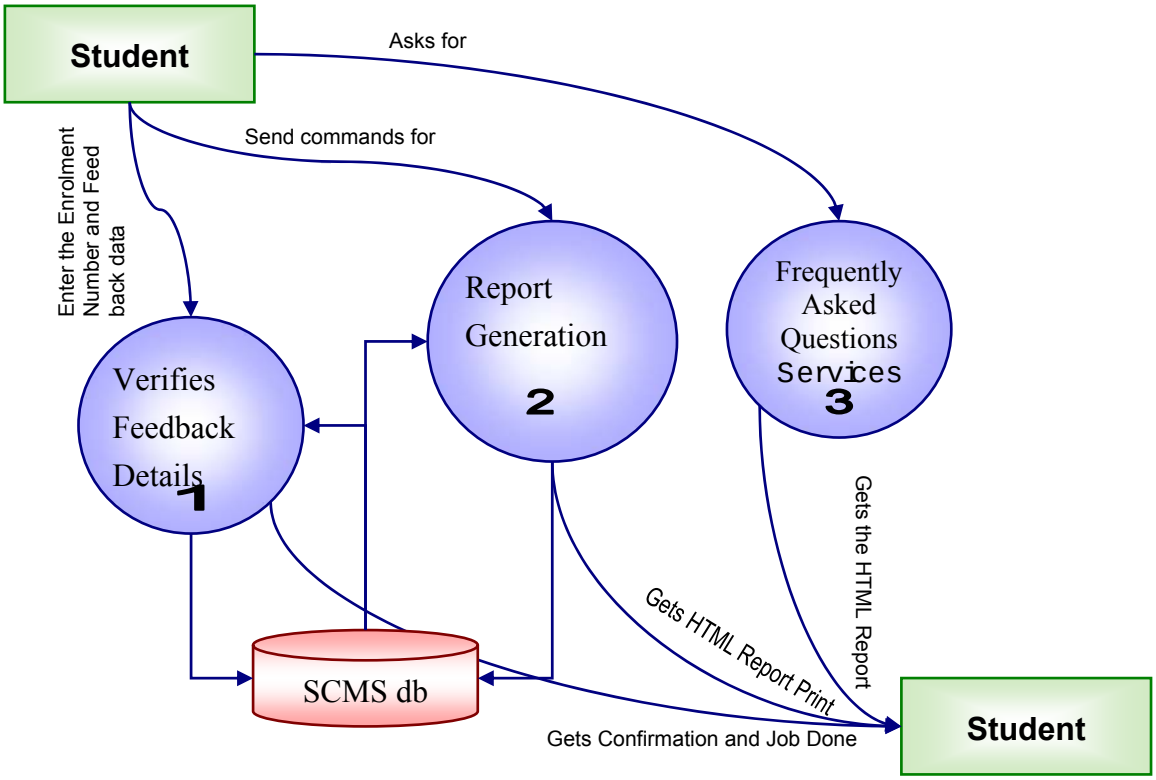


2nd Level



DFD for the Students

1st Level



2nd Level

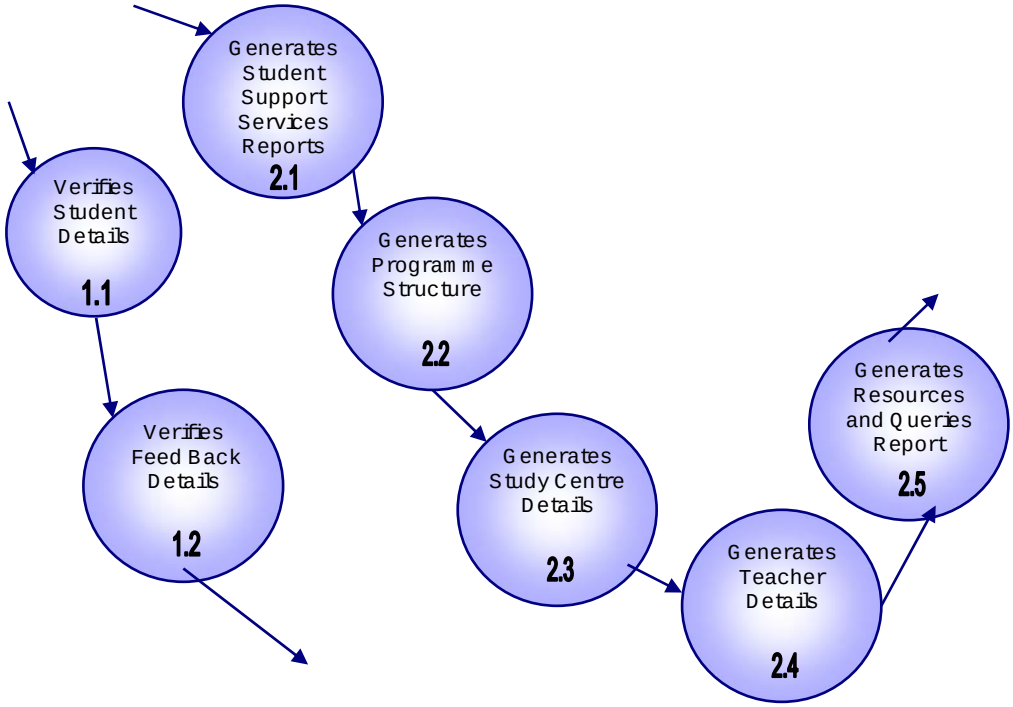


Figure 9: All the DFDs for the System

The **Data Flow Diagrams** as above give a brief idea on how our system is working. It also suggests to us the type of users who are using this system and the processes involved in the system.

After having a brief idea we would like to now explain the relationships among the users and data records available in the system in the next section.

4.2 Entity-Relationship Diagram for the system

The ER Diagram of the system is shown in *Figure 10.*

Figure 10: Entity Relationship Diagram for System

SOCIAS-IGNOU/P.O.10.5T/MAY, 2004