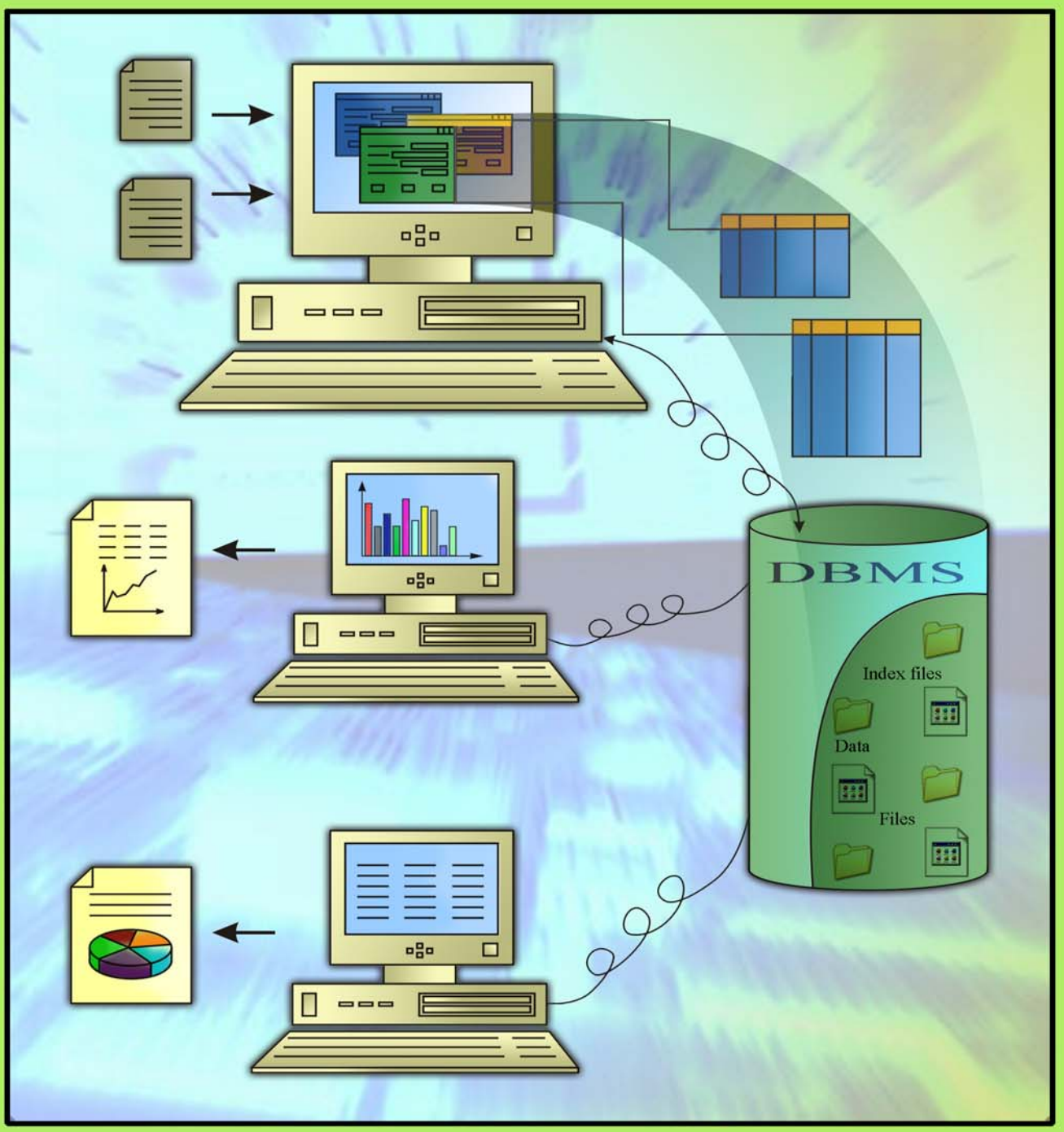
Indira Gandhi National Open University
School of Computer and Information Sciences

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.

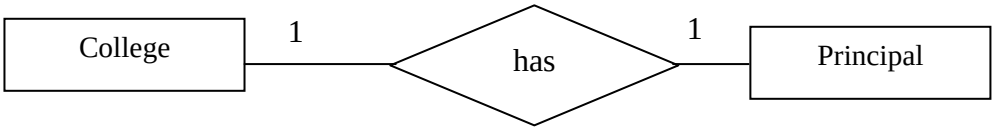
Similarly relationship among n entities is called n-ry relationship.

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

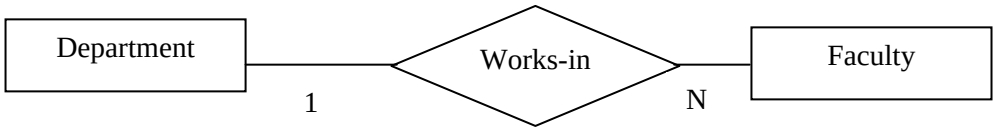


One college can have at the most one principal and one principal can be assigned to only one college.

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.

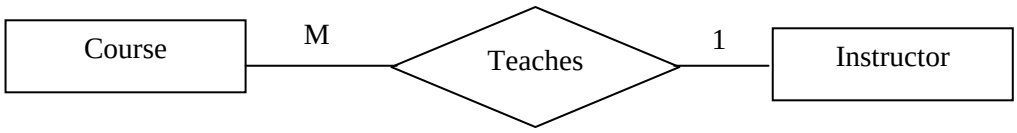
Example 18: Relationship between department and faculty.



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.

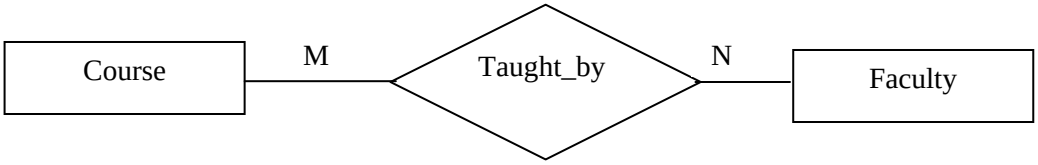
Example 19: Relationship between course and instructor. An instructor can teach various courses but a course can be taught only by one instructor. Please note this is an assumption.



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

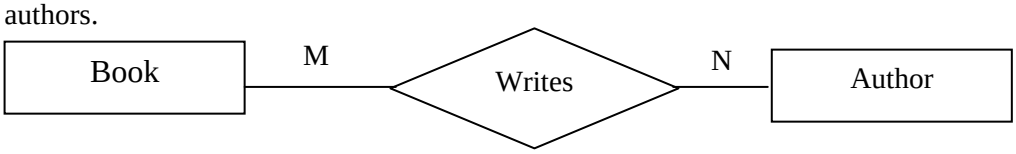
Example 20:

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:

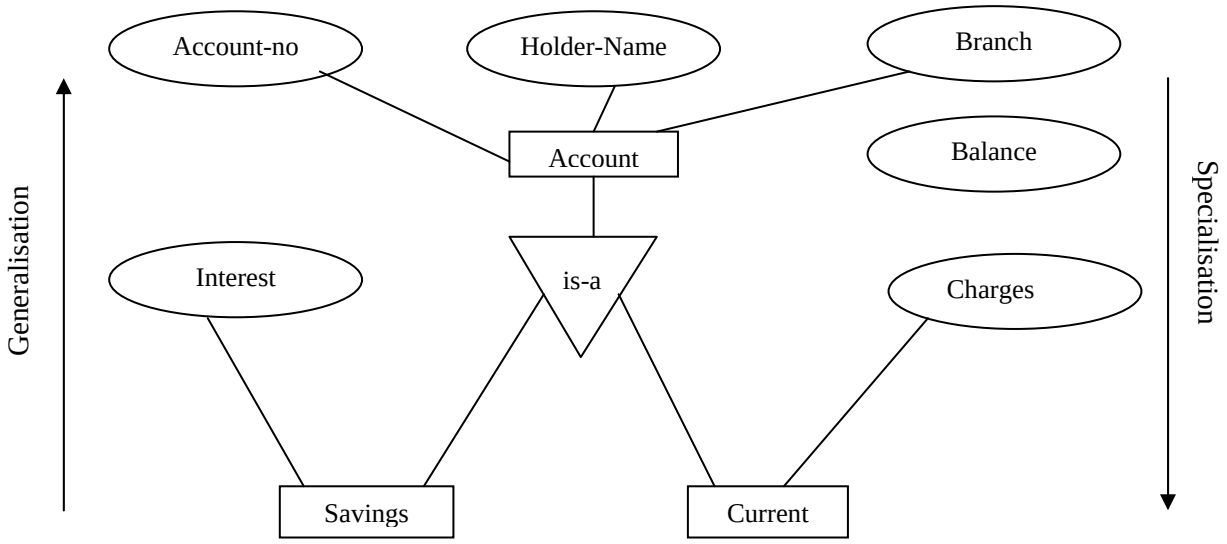


Figure 4: Generalisation and Specialisation hierarchy

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.

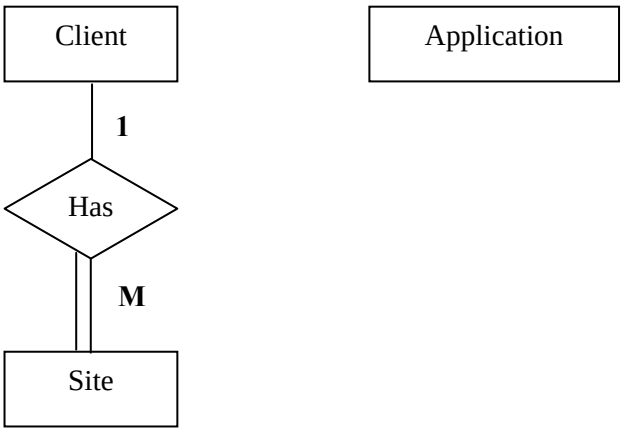


Figure 12: A One-to-Many Relationship

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:

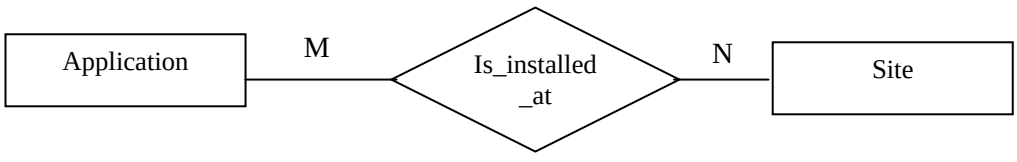


Figure 13: A Many-to-Many Relationship

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.

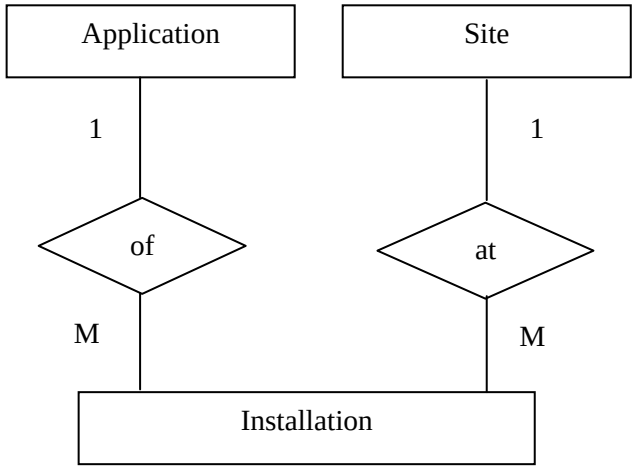


Figure 14: The INSTALLATION ENTITY

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.

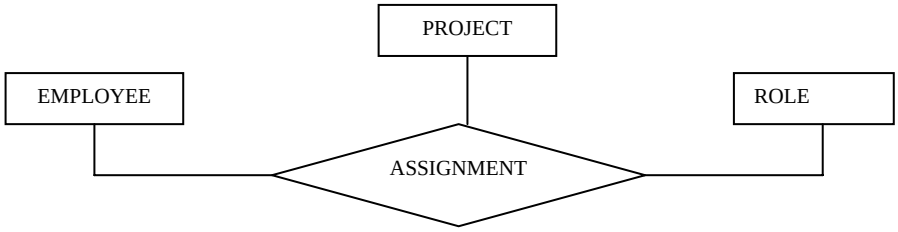


Figure 1: E-R diagram for employee role in development team

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*:

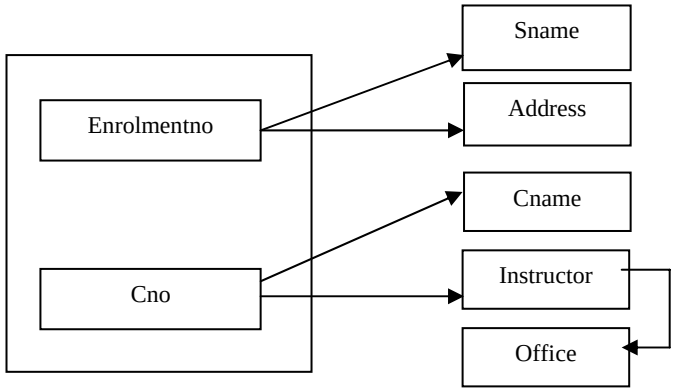


Figure 4: The dependencies of relation in Figure 3

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:

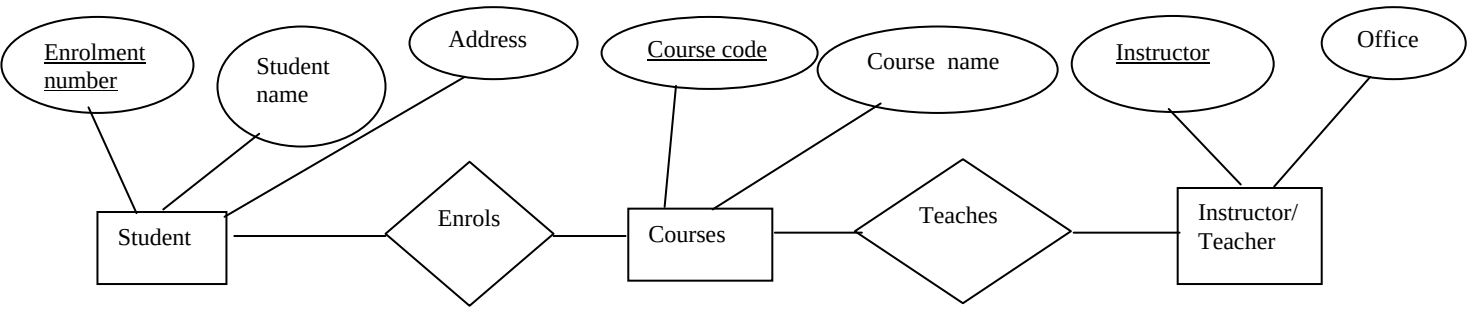


Figure 5: E-R diagram for student course Teacher

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!

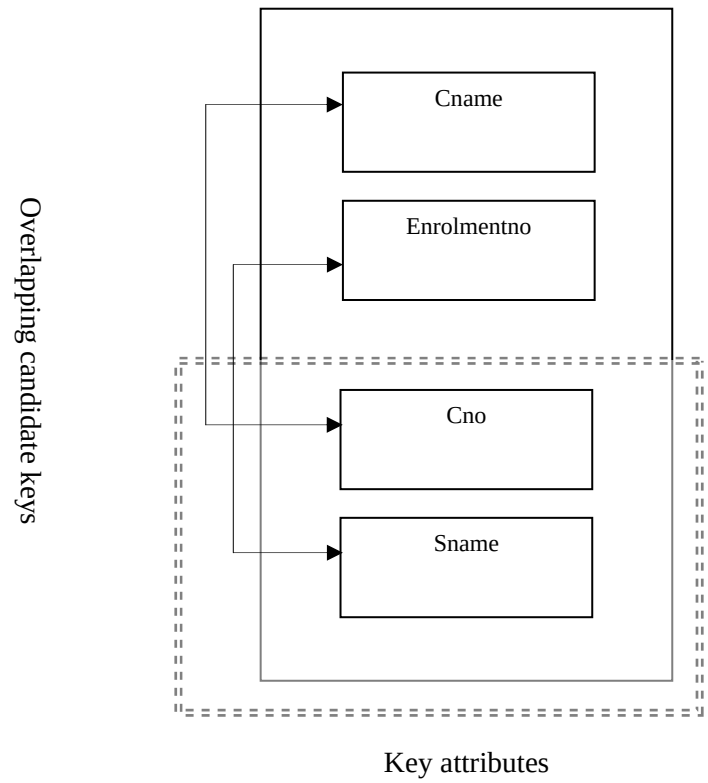


Figure 6: Functional Diagram for NEWSTUDENT relation

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)

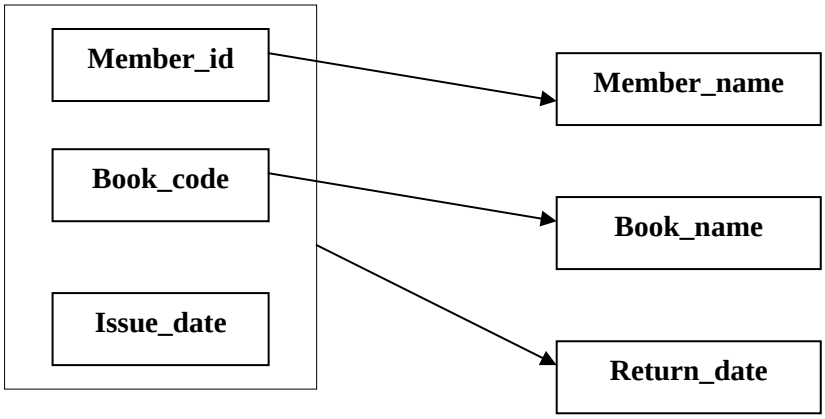
Why is the attribute issue_date on the left hand of the FD above?
Because a student for example Abishek can be issued the book having the book_code 0050 again on a later date, let say in February after Raman has returned it. Also note that return_date may have NULL value, but that is allowed in the FD.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