UG CA S3-01
(DSE)

FOUR YEAR BACHELOR'S DEGREE COURSE
(CBCS in Accordance with NEP 2020)

SEMESTER - III

FUNDAMENTALS OF DATABASE MANAGEMENT SYSTEM

COMPUTER SCIENCE

Hiranya Chandra Bhuyan School of Science and Technology
KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Shikhar Kumar Sarma, Department of Information Technology, Gauhati University
Prof. Hiren Kumar Deva Sarma, Department of Information Technology, Gauhati University
Dr. Sanjib Kr. Kalita, Department of Computer Science, Gauhati University

Course Co-ordinator : **Dr. Ridip Dev Choudhury,** KKHSOU
Dr. Tapashi Kashyap Das, KKHSOU
Dr. Sruti Sruba Bharali, KKHSOU

SLM Preparation Team

Units	Contributors
1, 5, 6	Dr. Tapashi Kashyap Das, KKHSOU
2, 7	Arabinda Saikia, KKHSOU
3, 4	Dr. Sangeeta Kakoty, KKHSOU
8, 9, 10, 11	Dr. Sruti Sruba Bharali, KKHSOU
12, 13	Dr. Ridip Dev Choudhury, KKHSOU
14	Dr. Chandan Jyoti Kumar, Cotton University

Editors

Content : **Dr. Tapashi Kashyap Das,** KKHSOU (Unit 2, 3, 4, 7 to 14)
Dr. Sanjib Kr. Kalita, Gauhati University (Former Faculty, KKHSOU) (Unit 1, 5 and 6)
Language: Prof. (Retd.) Robin Goswami, Cotton College

Structure & Format : Dr. Sruti Sruba Bharali, KKHSOU

October, 2024

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international) http://creativecommons.org/licenses/by-nc-sa/4.0/
Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

Headquarters: Patgaon, Rani Gate, Guwahati - 781 017
City Campus: NH 37, Resham Nagar, Khanapara, Guwahati - 781 022; www.kkhsou.ac.in

DETAILED SYLLABUS

UNIT 1 :	**FILE STRUCTURE** Data and Information, Concept of Field, Key field; Records and its types, Fixed length records and Variable length records; Files, Operation on files, Primary file organization	7-12
UNIT 2 :	**DATABASE SYSTEM** Traditional file approach versus Database approach; Data independence, Database System, Database Architecture, The three level of architecture, Mapping, Database administrator, Database Management system, Types of DBMS, Merits and demerits of DBMS	13-39
UNIT 3 :	**DATA MODEL** Conceptual model, Logical model, Physical model, Entity-Relationship (ER) model as a tool for conceptual design: Entities, Attributes and Relationships, Weak and Strong entities, Conversion of ER model into Relational schema, ER modelling symbols.	40-60
UNIT 4 :	**RELATIONAL MODEL** Relational data model concepts, Integrity constraints: Entity integrity, Referential integrity, Domain Constraints.	61-70
UNIT 5 :	**KEYS** Concept of keys, Composite key, Candidate key, Primary key, Alternate key, Foreign key, Defining Primary and Foreign keys in Database.	71-85
UNIT 6 :	**RELATIONAL DATABASE DESIGN** Database Design, Decomposition, Universal Relation, Functional Dependencies, Prime and Non-prime attributes	86-101
UNIT 7 :	**NORMALIZATION** Normalization, First Normal form (1NF), Second Normal form (2NF), Third Normal form (3NF), Boyce-Codd Normal form (BCNF)	102-120
UNIT 8 :	**INTRODUCTION TO SQL** Structured Query Language (SQL), Characteristics of SQL, Advantages of SQL, SQL Data types, Types of SQL commands, DDL, DML.	121-137
UNIT 9 :	**WORKING WITH SQL PART I** Creating a Database, Creating, Altering, and Deleting Tables Inserting new data, Updating data (the WHERE Clause, The Logical AND and OR operator), Deleting Data.	138-159

UNIT 10 : **WORKING WITH SQL PART II** 160-175
The SELECT statement, Filtering results with the WHERE Clause; Logical Operators and Operator Precedence (Using logical operator NOT, BETWEEN, LIKE, IN); Ordering Results with ORDER BY (Joining columns-concatenation); Selecting data from more than one table.

UNIT 11 : **WORKING WITH SQL PART III** 176-196
Join, Types of Join, SQL commands: Select… From…Where… Group by ….. Having… Order by…, Tables, Queries, Sub Queries, Insert, Update and Delete operations, Constraints considers (NOT NULL, UNIQUE, Check Primary key, Foreign key)

UNIT 12 : **WORKING WITH GUI** 197-220
Introduction to WAMP, Installation of WAMP, Working with phpMyAdmin.

UNIT 13 : **DATABASE RECOVERY** 221-236
Concept of database recovery, Backup of database, Types of database failure, Types of database recovery.

UNIT 14 : **DATABASE SECURITY** 237-343
Goals of database security, Discretionary Access Control, Mandatory Access control, Statistical Databases, Data Encryption.

COURSE INTRODUCTION

Welcome to the "Fundamentals of Database Management System" course, a comprehensive journey into the core concepts and practical applications of database technology. The traditional method for storing computer data was data files. Data files have been very popular since 1960s. In fact, even today, many major computer applications run on file-based computer systems. In the last few decades, Database Management Systems has become a subject of great significance in the IT industry. In today's competitive environment, Database and Database Management Systems (DBMS) have become essential for managing our business, governments, banks, universities and every other kind of human endeavour. This course is designed to equip you with a robust understanding of how data is structured, managed, and manipulated in modern computing environments. We will start with the basics of file structure and progressively delve into more complex topics such as database design, SQL, and database recovery and security. By the end of this course, you will have a comprehensive understanding of database systems, equipped with the skills to design, manage, and secure databases effectively.

The unit of these course includes some along-side boxes to help you know some of the difficult, unseen terms. Some "EXERCISES" have been included to help you apply your own thoughts. You may find some boxes marked with: "LET US KNOW". These boxes will provide you with some interesting and relevant additional information. Again, you will get "CHECK YOUR PROGRESS" questions. These have been designed for you to self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with "ANSWERS TO CHECK YOUR PROGRESS" given at the end of each unit.

UNIT 1: FILE STRUCTURE

UNIT STRUCTURE

1.1 Learning Objectives
1.2 Introduction
1.3 Data and Information
1.4 Fields and Records
1.5 Files
    1.5.1 Operation on Files
1.6 Primary File Organization
    1.6.1 Sequential Access Organization
    1.6.2 Direct Access Organization
    1.6.3 Indexed-Sequential Access Organization
1.7 Let Us Sum Up
1.8 Answers to Check Your Progress
1.9 Further Reading
1.10 Model Questions

1.1 LEARNING OBJECTIVES

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

- define Data and Information
- define Fields, Records and Files
- describe operation on Files
- describe different File organization techniques like Sequential access, Direct access, Index-Sequential access etc.

1.2 INTRODUCTION

Computer files and file systems have great similarities with traditional files and file systems. All basic operations like insertion, deletion, updation, searching, etc. are possible in both the systems. For the formation of computerised record keeping systems we are required to introduce Database and Database Management Systems (DBMS).

Some basic terms like data, information, fields, records, files etc. and their organization are prerequisite to understand database and DBMS.

This unit is an introductory unit and gives you an understanding of those basic terms and concepts.

1.3 DATA AND INFORMATION

Data and its efficient management have become an important issue with the growing use of information technology and computers. Proper organization and management of data is necessary to run the organization efficiently.

Data are raw or isolated facts from which required information can be produced. There are uncountable examples of data such as name of a person, name of a place, the roll number of a student, the ticket number of a passenger, passport number of a person, PAN card number, credit card number, identification number for an employee etc.

Data are processed to generate information which is meaningful for the recipient. Information helps human beings in their decision making process. Some common examples of information are: time table of a class , report card, printed documents, pay slips, invoices, bank statements, receipts, reports etc. The information is obtained by assembling items of data into a meaningful form. For example, marks obtained by students and their roll numbers are *data*, but the the report card/mark sheet generated from these data are *information*. Thus, ***information*** may be defined as the collection of related data that when put together gives meaningful and usefull message to a recipient who uses it for different purposes like decision making. Data and infomation are closely related and are often used interchangeably.

1.4 FIELDS AND RECORDS

Data item or Fields

A data item or field is the smallest unit of information that has meaning to its user. For example, in a Telephone bill, *Name*, *Telephone_no*, *Bill_amount*, *Address* are Fields. Similarly, in an employee salary slip, basic

pay, gross pay, HRA, DA, deduction etc. are examples of Fields.

Records

A ***record*** is a collection of logically related fields. Each record contains unique and uniform information that is divided into fields. This uniformity allows for consistent access of information. An example of a student record is shown in figure 1.1. A record consists of values for each field.

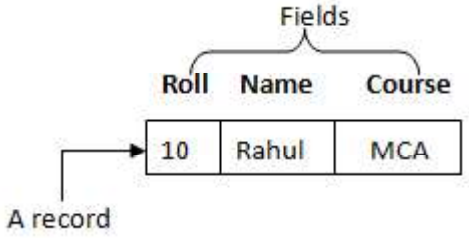


Figure.1.1: An example of record

A file can contain ***fixed length record*** and ***variable length record***. Each field in a *fixed length record* is given a fixed length. The length of the record should be large enough to hold even the largest value anticipated for that field. In a student's record we may provide 15 spaces for the name field, 3 for roll number, 2 for each percentage field, 4 for division etc. Here the length of each field is the same for all records irrespective of the values contained in them. It may result in wastage of storage space in the file if all names are not of 15 characters.

In case of *variable length record*, the size of the data item are not fixed. The value in every record is allowed to take up as much space as is required by its size which enables us to save disk space. The end of each item is recognized by a delimiter such as a comma, or a colon. The following example will make the definition clear. The gap between each data item is having a space which is counted as a character.

4, Karabi Roy, 72, 1st (22 characters)

In the above record, the respective fields are Roll_No, Name, Percentage and Disvision. This record occupies 19 characters including the blank spaces and commas.

When we use fixed-length records, we need to make the record length equal to the length of the longest record. If we need to store many short records with occasional long ones, using fixed-length records wastes a lot of disk space, so variable-length records would be a better choice.

1.5 FILES

A file is a collection of related sequence of records. Records with identical formats/types are generally kept in a file. If every record in the file has exactly the same size in bytes, the file is said to be made up of fixed-length records. If the records in the file are of different sizes, the file is said to be made up of *variable-length records*.

For example, in Figure 1.2, a student result file is shown in tabular format. It contains ten records of students and each record has four related fields namely ***Roll_No***, ***Name***, ***Percentage***, ***Division***. In the file, the information about a particular student in one line or row of the table is an example of record. The collection of result information for all the students (all columns and rows), i.e., the entire table is an example of ***file***.

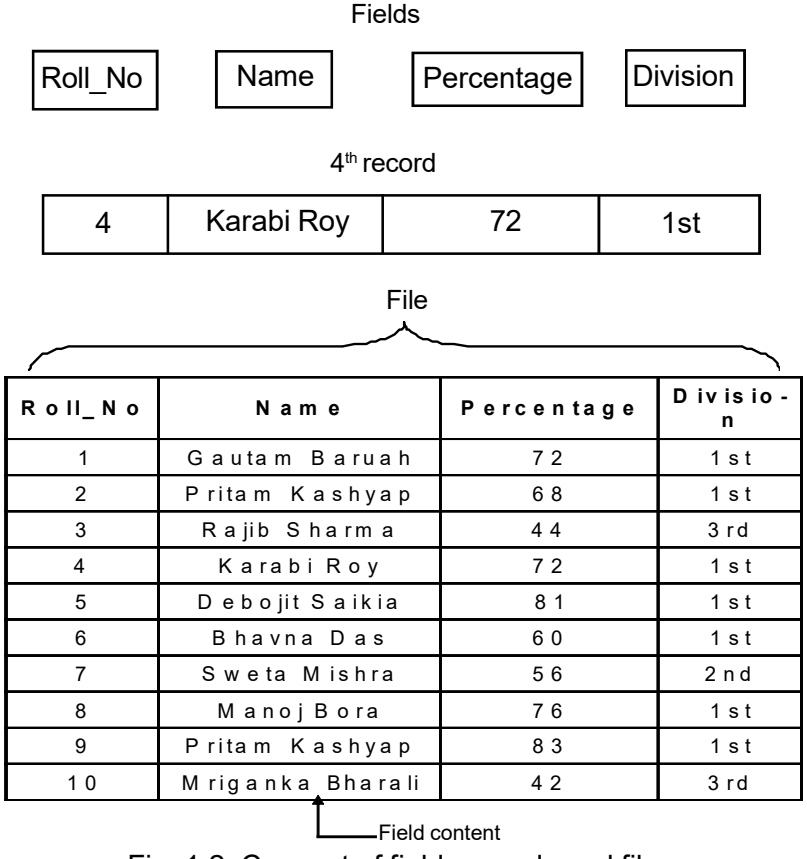


Fig. 1.2: Concept of field, records and files

Each record in a file may contain many fields, but the value in a certain field may uniquely determine the record in the file. Such a field is known as ***key field***. In case of student's record in Figure 1.2, *Roll_No* is the key

field because it is unique for each student. Similarly, *Part_no* in a stock file, *Account_no* in a bank customer's file are all examples of key fields.

A file can be of two types:

Master File and ***Transaction File***

A *master file* contains records of relatively permanent data. For example, the name, roll number, sex, date of birth of a student would appear in a student master file as these records are.

A *transaction file* is a file in which day to day operational data are stored. Transaction file contains the records that are used to update the records of the master file. For example, a transaction file may contain the information of students who pay their monthly fees. This file will update the record of the student whose roll number matches with the transaction file record. The record of the master file will thus be updated as and when necessary.

1.5.1 Operation on Files

There are various operations associted with files. Let us take a real life example where files are being maintained to keep records. In a school, class wise student files are maintained where information about each student is recorded and all these records are placed together within a file cover with a name on it. For different classes different student files are normally maintained. Again, all such files can be stored together in a shelf or a file-cabinet (like a folder in a computer). To retrieve any information about a particular student of a specified class, the concerned class file (identified by the filename) can be taken out and the records within that file can be searched out to get the details of the student. If the file records remain sorted with respect to roll numbers, then the roll numbers of the students can be treated as a search key.

A computer can also work in the same way. Computer files also facilitate easy storage, retrieval, and manipulation of data. Instead of paper files, computer store electronic files in a hard disk or removable

disk in the form of bits and bytes. Therefore, a disk file is nothing but a collection of records. The record can be entered through a keyboard and saved as a file in the hard disk. Various operations are associated with computerised files. There are three major operations on files as given below :

(a) **Insertion of new records**:

This is the process of adding data or record in a file at the indicated location. A record is inserted in a sequential file at the end of the file.

(b) **Modifications of some existing records**:

This operation is done to modify old data or record with a new data or record in a file at the indicated location.

(c) **Deletion of records**:

This is the process of removal of records or data at the specified location.

Maintaining records and files in proper format is very essential. It enables the users to access information easily and quickly. In any organization the maintenance is basically carried out by the system administrator of the system. It involves integrity checking, missing of null value checking, corrupted data checking etc. We will discuss the concept of data integrity and null value later in this block.

1.6 PRIMARY FILE ORGANIZATION

A file containing records may have the organization depending upon the way these are arranged in the file. A file is organized to ensure that records are available for processing. Following are the tree types of file organization:

- Sequential Access organization.
- Direct Access organization.
- Indexed-Sequential Access organization.

1.6.1 Sequential Access Organization

In case of sequential file, records are arranged in some order. For example, a student's file may be kept in ascending order of roll numbers. It is not necessary that the records of a sequential file should be physically in adjacent positions. However, on a magnetic tape for sequential organizations, the records are written one after the other along the length of the tape. In case of disks, the records of a sequential file may not be in contiguous locations. The sequential order may be maintained with the help of pointer in each record. To access a record, previous records within the block are needed to be scanned. Thus, sequential record design is suitable for reading one record after another without a search delay. In a sequential organization, records can be added only at the end of the file. It is not possible to insert a record in the middle of the file without rewriting the file. However, in a database system a record may be inserted anywhere in the file, which would automatically re-sequence the records following the inserted record. Another approach is to add all new records at the end of the file and later sort the file on a key (name, number, etc). Information on a sequential-access device can only be retrieved in the same sequence in which it is sorted.

Sequential processing is quite suitable for such applications like preparation of monthly pay slips, or monthly electricity bills etc., where most, if not all, of the data records need to be processed one after another. In these applications, data records for every employee or customer needs to be processed at scheduled intervals (in this case monthly). However, while working with a sequential-access device, if an address is required out of order, it can only be reached by searching through all those addresses, which are stored before it. For instance, data stored at the last locations cannot be accessed, until all preceding locations in the sequence have been traversed. This is analogous to a music tape cassette. Suppose you like to hear a particular song which is in 8th position in the cassette. For that you can "fast forward" the first seven songs. Although, not fully played, the 7 songs are still accessed. Magnetic tape is an example of sequential access storage

device. Following are the main drawbacks and advantages of sequential file organization:

Drawbacks :

- Sequential files are not suitable for on-line enquiry where up-to-date information is required.
- Information on the file is not always current.
- Addition and deletion of records are not a simple task.
- Searching information in a sequential file can be a very slow process. For any search operation, we need to start reading a sequential file from the beginning and continue till the end, or untill the desired record is found, whichever is earlier. This is both time-consuming and cumbersome.

Advantages:

- File design is simple.
- Low-cost file medium. Tape can be used.

1.6.2 Direct Access Organization

There is a popular type of file, called *direct files* which permit random access or direct access. In case of direct file organization records are placed randomly throughout the file. Records need not be in sequence because they are updated directly and rewritten back in the same location. New records are added at the end of the file or inserted in specific locations based on software commands. Records are accessed by addresses that specify their disk locations. An address is required for locating a record, for linking records, or for establishing relationships. Addresses are of two types: *absolute* and *relative*.

An ***absolute*** address represents the physical location of the record. It is usually stated in the format of sector/track/record number. For example, *3/16/4* means *go to sector 3*, *track 16 of that sector*, and the *fourth record of the track*. One problem with absolute address is that they become invalid when the file that contains the records is relocated on the disk.

A ***relative*** address gives record locations relative to beginning of the file. There must be fixed-length records for reference. Another way of locating a record is by the number of bytes it is from the beginning of the file.

1.6.3 Indexed-Sequential Access Organization

An *indexed-sequential* file is basically a file organized serially on a key field. In addition, an *index* is maintained which speeds up the access of isolated records. The index provides random access to records, while the sequential nature of the file provides easy access to the subsequent records as well as sequential processing. Indexed-sequential access organization reduces the magnitude of the sequential search and provides quick access for sequential and direct processing. The primary drawback is the extra storage space required for the index. It also takes longer to search the index for data access or retrievals. The retrieval of a record from a sequential file, on average, requires access to half the records in the file. To improve the query response time of a sequential file, a type of indexing technique can be added. The purpose of indexing is to expedite the search process. Indexes created from a sequential set of primary keys are referred to as indexed-sequential. Just as we use the index to locate information in a book, an index is provided for the file. Some advantages and disadvantages of this organization are given below:



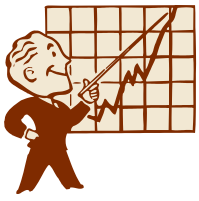
Index:
An index is a table of records arranged in a particular fashion for quick access to data.

Advantages:

- Up-to-date information will always be available on the file
- It is suitable for on-line or derect access processing.

Disadvantages:

- Less efficient in the use of storage space.
- Relatively an expensive medium.



CHECK YOUR PROGRESS

1. State whether the following statements are true (T) or false (F) :
   (i) A record is inserted in a sequential file at the end of the file.
   (ii) Information is data that has been processed into a more useful form.
   (iii) A collection of fields constitutes a file.
   (iv) A file needed for updating a master file is called transaction file.
   (v) Records of transaction files are permanent in nature.
   (vi) Magnetic tape is an example of sequential access storage device.
   (vii) In direct access file organization records are placed randomly.
   (viii) An index is maintained which speeds up the access of isolated records in case of sequential file organization.
   (ix) The smallest piece of meaningful information is called data item.
   (x) Sequential files are suitated for on-line enquiry where up-to-date information is required.



1.7 LET US SUM UP

- Data may be defined as a known fact that can be recorded and that have implicit meaning.
- Information is processed and organised data. It can be defined as collection of related data that, when put together, communicate meaningful and useful message to a recipient who uses it.
- The smallest piece of meaningful information is called a field.
- A record is collection of field values or data items of a given entity.

- A file is organized to ensure that records are available for processing. There are three types of organization used in computer to store records. They are: Sequential access, Direct access, and Index-sequential access organization.
  (a) Sequential organization means storing records in contiguous blocks according to a key field. Magnetic tape is an example of sequential access storage device.
  (b) Indexed-sequential organization stores records sequentially but uses an index to locate records. Records are related through chaining using pointers.
  (c) Direct-access organization has records placed randomly through-out the file. Records are updated directly and independently of other records.
- In direct addressing two types of addressing may be used: absolute and relative address.
- An absolute address represents the physical location of the record. It is usually stated in the formal of sector/track/record number. A relative address gives record locations relative to beginning of the file. There must be fixed-length records for reference.
- A sequential that is indexed is called an indexed-sequential file.



1.8 ANSWERS TO CHECK YOUR PROGRESS

1. (i) True (ii) True (iii) False (iv) True (v) False
   (vi) True (vii) True (viii) False (ix) True (x) False



1.9 FURTHER READING

1. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems. Pearson.

2. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



1.9 MODEL QUESTIONS

Q1. Explain Data and Information?

Q2. Explain the following terms with example: Field, Record, File.

Q3. What are the different types of file organization techniques?

Q4. Write short notes on :

(a) Direct access organization

(b) Sequential access organization

(c) Index-Sequential organization

Q5. What are the advantages and disadvantages of sequential access organization.

Q6. What are the advantages and disadvantages of indexed-sequential access organization.

*****

UNIT 2 : DATABASE SYSTEM

UNIT STRUCTURE

2.1 Learning Objectives
2.2 Introduction
2.3 Traditional File Approach
2.4 Database Approach
    2.4.1 Advantages of Database
2.5 Database Management System
    2.5.1 Merits and Demerits of DBMS
2.6 Database Architecture
2.7 Data Independence
    2.7.1 Logical Data Independence
    2.7.2 Physical Data Independence
2.8 DBMS Language
2.9 Types of DBMS
    2.9.1 Centralised DBMS
    2.9.2 Parallel DBMS
    2.9.3 Distributed DBMS
    2.9.4 Client-Server DBMS
2.10 Database Administrator
2.11 Let Us Sum Up
2.12 Further Reading
2.13 Answers to Check Your Progress
2.14 Model Questions

2.1 LEARNING OBJECTIVES

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

- define a database and a Database Management System (DBMS)
- describe DBMS architecture
- illustrate data independence and data dictionary
- explain DBMS language

- identify the components of database system environment
- describe the role of Database Administrator (DBA)

2.2 INTRODUCTION

In the previous unit, we have learnt the basic idea of data, information, fields and records. In addition, concepts of files and basic file organization techniques are also discussed. All these are the elementary concepts relating to the database system. A *database system* is a tool that simplifies the managing of data and provides necessary manipulation on data, on the basis of the users demand. It is the central repository of data in the organization's information system. It provides an array of features which can be used to ensure optimal utilization of data for enhancing effective decision making in an organization.

In this unit, we will learn a some basic concept of database and DBMS. We will also discuss the ANSI/SPARC three-tier database architecture and different types of DBMS. Besides, we will discuss the responsibility of a database administrator.

2.3 TRADITIONAL FILE APPROACH

In earlier days, an organization's information was stored as group of records in separate files. These file processing systems consisted of a few data files and many application programs as shown in the figure below. Each file, called a *flat file,* contained and processed information for one specific function, such as, accounting or inventory. At that time programmers used programming languages such as COBOL to write application programs that can directly access flat files to perform data management services and provide information for the users. Each application files and programs were created and maintained independent of other applications.

For example, if we consider the students data in a universitiy then

– student address may be needed for *the* applications like *registering, library management, financial office, grade reporting* etc.

– each application separately maintains its data files and programs to manipulate those files
– possibly the same data (e.g., length of names, address etc.) are stored in different formats in the above applications
– whenever some information regarding a group of students is updated in one application that updation may not be done simultaneously in the different applications where students records are stored.

As a result, the system will provide wrong information about the students. In addition, potentially different values and/or different formats for the same data are stored in different files which lead to not only wastage of space but also cause the redundancy.

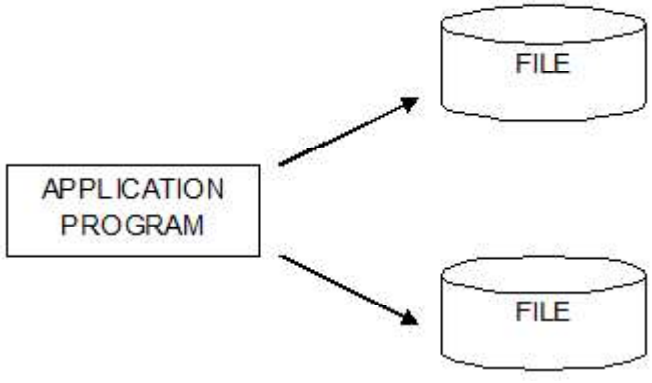


Figure 2.1: Traditional File Approach

On creating the files and programs for the file oriented system, the developers focused on business processes, or how business was transacted, and their interaction. However, business processes are dynamic, requiring continuous changes in files and applications. Moreover, programmers design the codes in accordance with the physical storage structure of data and access procedures also depends on it. Therefore, any physical changes result in the programmers rewriting the code to adjust the change.

The file-based approaches, which came into being as the first commercial applications of computers, suffered from the following significant disadvantages :

Data redundancy :

In a file system if an information is needed by two distinct applications, then it may be stored in two or more files. Repetition of the same data item in more than one file is known as *data redundancy*. This leads to an increase in the cost of data entry and data storage.

Data integrity problem :

Data redundancy also leads to data inconsistency or loss of data integrity. Data integrity refers to the consistency of data in all files. That is, any change in a data item must be carried out in every file containing that field for consistency.

Lack of data independence :

In file processing systems, files and records are described by specific physical formats that are coded into the application program by programmer. If the format of a certain record is changed, the code in each file containing that format must be updated.

Poor data control :

A file-oriented system is decentralised in nature. It means there was no centralised control at the data element level.

Incompatible file formats :

As the structure of files is embedded in the application programs, the structures are dependent on the application programming language. For example, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a 'C' program. The direct incompatibility of such files makes them difficult to process jointly.

2.4 DATABASE APPROACH

An alternative approach to the traditional file processing system is the modern concept, known as the database approach. A database is an organised collection of records and files which are related to each other. In

a database system, a common pool of data can be shared by a number of applications as it is data and program independant. Thus, unlike a file processing system, data redundancy and data inconsistency in the database system approach are minimised. In this approach, the user is free from the detailed and complicated task of keeping up with the physical structure of data. A clelar-cut distinction between traditional file system and database system is depicted by the following diagram.

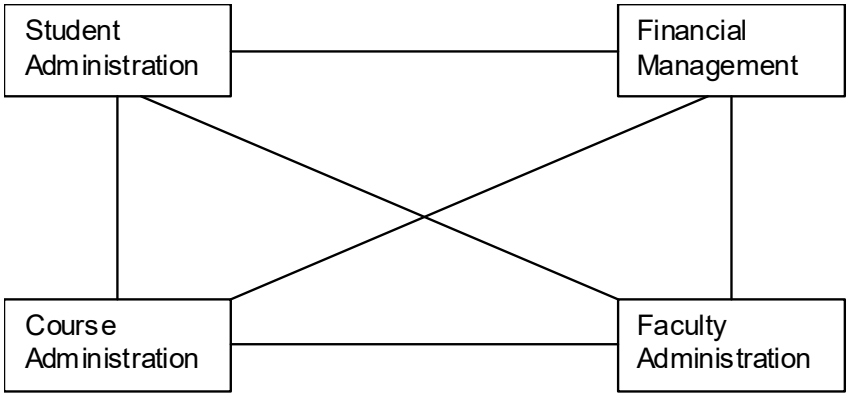


Figure 2.2: Traditional file approach

Here in this figure, the traditional record keeping system in a university is shown, where all applications are interrelated and cause repetation of the same data in different files leading to the problem of data redundancy and inconsistency.

In the figure below, indicates how several applications share common data in a database approach.

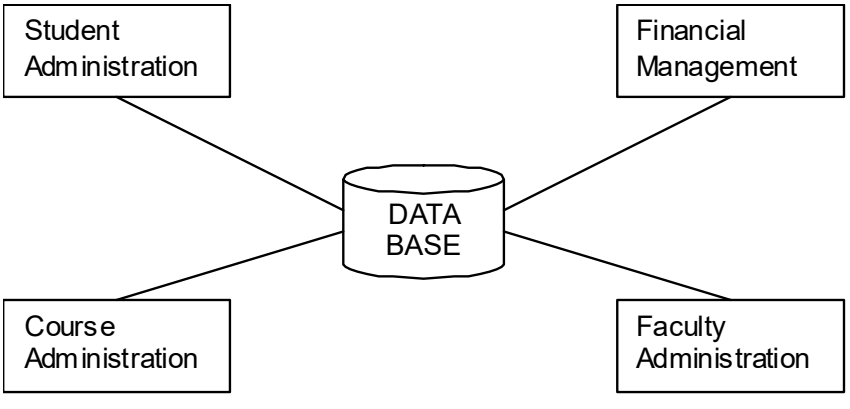


Figure 2.3: Database approach

Always remember - a database is organised in such a way that a computer program can quickly select the desired piece of data. A database can further be defined as under. It -

- is a collection of interrelated data stored together without harmflul or unnecessakry redundancy.
- stores data indepedent of programs, and any changes in data storage structure or access strategy donot require changes in accessiing programs or queries.
- serves multiple applications in which each user has its own view of data. The data is protected from unauthorised access by security mechanism and concurrent access to data is provided with recovery mechanism.

Broadly, the objectives of the database approach are to make informaltion access easy, fast, relatively inexpensive and flexible for the user. The specific objectives may be listed as follows :

- controlled data redundancy
- enhanced data consistency
- data independence
- application independence
- ease of use
- econommical, and
- recovery from failure

2.4.1 Advantages of Database

Database approach provides the following benefits over the traditional file processing system :

Redundancy control :

In a file processing system, each application has its own data, which causes duplication of common data item in more than one file. This data duplication needs more storage space as well as multiple updation for a single transaction. This problem is overcome iin database approach where data is stored only once.

Data consistency :

The problem of updating multiple files in file processing system leads to inaccurate data as different files may contain different information of the same data item at a given point of time. In database approach, this problem of inconsistent data is automatically solved wiith the control of redundancy.

Thus, in a database, data accuracy or integrity or accessibility of data is enhanced to a great extent.

Data Independence :

This means that data and programs are independent. Most of the file processing systems are data dependent, which implies that the file structures and accessing programs are interrelated. However, the database approach provides an independence between the file structure and program structure.

Sharing of data and security :

Data in a database are shared among the users and the applications. In database approach data are protected from unauthorised access.

2.5 DATABASE MANAGEMENT SYSTEM

The *database management system* is the interface between the users (application programmers) and the database(the data). A database management system(DBMS) is a program that allows the user to define, manipulalte and process the data in a database, in order to produce meaningful information.

A DBMS is a set of *software programs* that controls the *organization, storage, management, and retrieval of data* in a database. It is a set of pre-written programs that are used to store, update and retrieve a database. The DBMS accepts requests for data from the application program and instructs the *operating system* to transfer the appropriate data. The follow-ings are the examples of DBMS software :

Microsoft Visual FoxPro, MonetDB, MySQL , Oracle Database, PostgreSQL, SQL Anywhere, SQLite, FileMaker, IBM DB2, IBM UniVerse, Firebird, Microsoft Access, Microsoft SQL Server etc.

The following are examples of database applications :

- reservation systems, banking systems
- record/book keeping (corporate, university, medical), statistics
- bioinformatics, e.g., gene databases
- criminal justice
  - fingerprint matching
- multimedia systems
  - image/audio/video retrieval
- satellite imaging; require petabytes (10^{15} bytes) of storage
- the web
  - almost all data-intensive websites are database-driven;
- data mining (Knowledge Discovery in Databases) etc.

To complete our initial definitions, we will call the database and DBMS software together as a *database system*. Figure 2.4 depicts a database system.

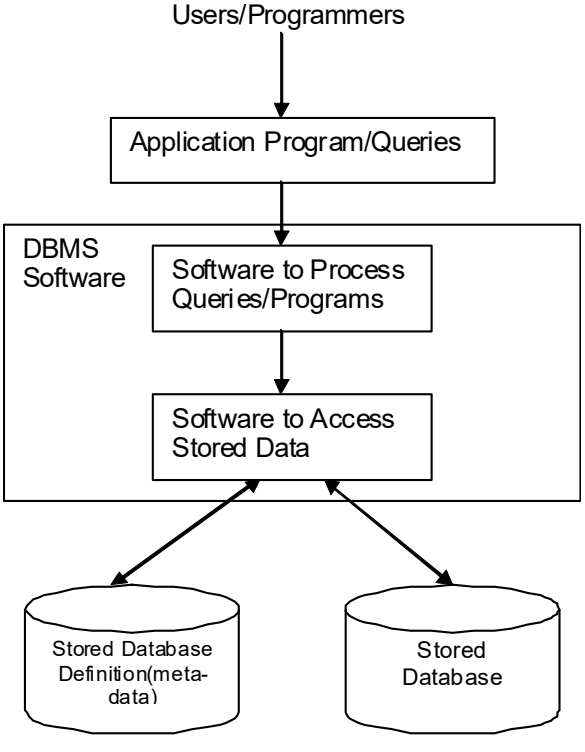


Figure: 2.4: A simplified database system

2.5.1 Merits and Demerits of DBMS

Due to the centralised management and control, the database management system has numerous advantages, some of which are explained below :

a) Merits :

i) Minimal data redundancy :

Centralized control of data avoids the unnecessary duplication of the data and effectively reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large storage of data. Another advantage of avoiding duplication is the elimination of the inconsistencies that tend to be present in redundant data files.

ii) Program-data independence :

The separation of metadata(data description) from the application programs that use the data is called data independence. In the database environment, it allows for changes at one level of the database without affecting the other levels. With the database approach, metadata are stored in a central location called repository. This property of data systems allows an organizations data to change and develop without changing the application programs that process the data.

iii) Efficient data access :

DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

iv) Improved data sharing :

Since, database system is a centralised repository of the data belonging to the entire organization(all departments), it can be shared by all authorised users. Existing application programs as well as new application programs(which are designed on the basis of the existing data) can share the data in the database.

v) Data Integrity :

Integrity of data means that the data in a database is always accurate, such that incorrect information cannot be stored in a database. In order to maintain the integrity of data, some integrity constraints are enforced on the database.

vi) Data security :

Database security is the protection of database from unauthorised users. The DBA (Database Administrator), can define security rules to check unauthorised access to data. Some users may be given rights to retrieve the data only, whereas others may be permitted to retrieve and edit the data. The DBA can formulate different rules for each type of access (retrieve, modify, delete, etc) to each piece of information in the database.

vii) Enforcement of standards :

With the central control of the database, a DBA can define and enforce the necessary standards. Applicable standards might include any or all of the following : departmental, organizational, industry, corporate, national or international. Standards can be defined for data formats to facilitate the exchange of data between systems, naming conventions, display formats, terminology, report structure etc. The data repository provides DBAs with a powerful set of tools for developing and enforcing these standards.

viii) Providing Backup and Recovery :

A DBMS must provide the facilities for recovering from hardware or software failures. The ba ckup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the program started executing.

b) Demerits :

The **demerits** of the database approach are summarized below :

i) Complexity:

The provision of the functionality that is expected of a good DBMS makes the DBMS an extremely complex piece of software. Database designers, developers, database administrators and the end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization.

ii) Size :

The complexity and breadth of functionality makes the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently.

iii) Performance :

Typically, a file-based system is written for a specific application, such as invoicing. As result, performance is generally very good. However, the DBMS is written to be more general, to cater for many applications rather than just one. The effect is that some applications may not run as fast as they used to.

iv) Higher impact of a failure :

The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the availability of the DBMS, the failure of any component can bring operations to a halt.

v) Cost of DBMS :

The cost of DBMS varies significantly, depending on the environment and functionality provided. There is also the recurrent annual maintenance cost.

vi) Additional Hardware cost :

The disk storage requirements for the DBMS and the database necessitate the purchase of additional storage space. Furthermore,

to achieve the required performance it may be necessary to purchase a large machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.

vii) Cost of Conversion:

The cost of the DBMS and extra hardware may be significant compared to the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems and possibly the employment of specialist staff to help with conversion and running of the system. This cost is one of the main reasons why some organizations feel tied to their current systems and cannot switch to modern database technology.



CHECK YOUR PROGRESS

1. Select the correct answer :

(a) Which is not a DBMS packages?

(i) Unify (ii) Ingress

(iii) IDMS (iv) All are DBMS packages

(b) Find the wrong statement

Database software

(i) provides facilities to create, use and maintain database.

(ii) supports report generation, statistical output, graphical output.

(iii) provides routine for backup and recovery.

(iv) all are correct.

(c) Which one of the following is not a valid relational database?

(i) SYBASE (ii) IMS

(iii) ORACLE (iv) UNIFY

(d) Centralized control is

(i) advantage of a DBMS (ii) disadvantage of a DBMS

(iii) Both (i) and (ii) (iv) None of the above

(e) Data are

(i) Raw facts and figures (ii) Information

(iii) Electronic representation of facts

(iv) None of these

2.6 DATABASE ARCHITECTURE

So far, we have come to know that, a DBMS is a collection of interrelated files and a set of programs that allow several users to access and modify these files. A major purpose of a database system is to provide the users with an abstract view of the data. That is the system hides certain details of how the data is stored and maintained. We can imagine that the whole database system is divided into levels. The generalised architecture of a database system is called the ANSI/SPARC (American National Standards Institute - Standards Planning and Requirements Committee) model.

ANSI/SPARC three-tier database architecture is shown in the Fig. 2.5

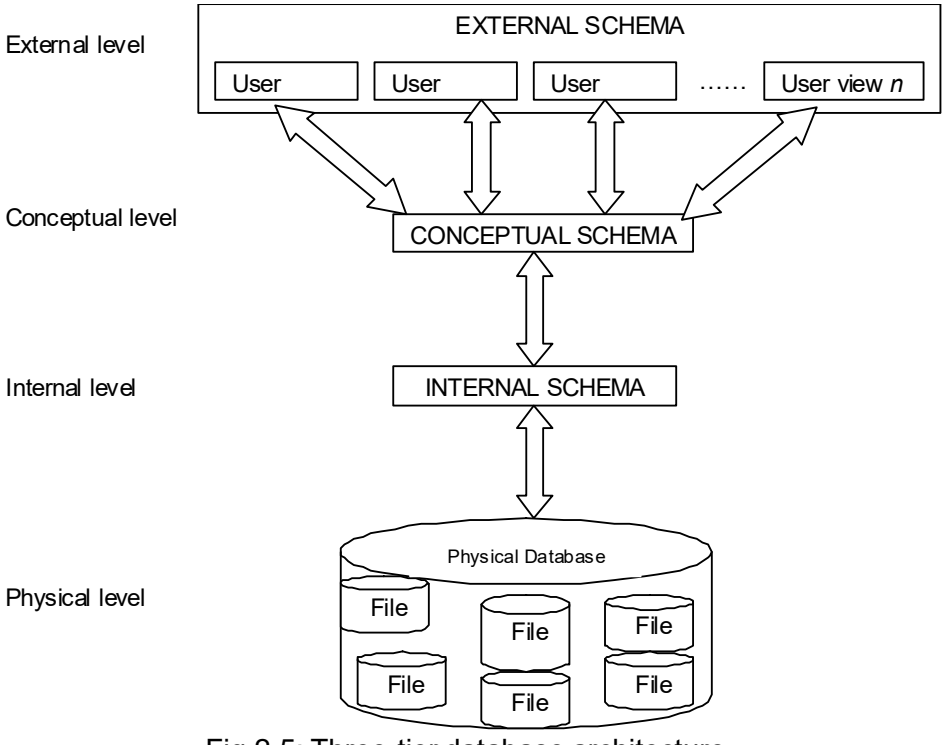


Fig.2.5: Three-tier database architecture

It consists of the following three levels :

- External level or view level,
- Conceptual level,
- Internal level or physical level.

External level :

The external level is the user's view of the database and closest to the users. This level describes that part of the database that is relevant to the user. Most of the users of database are not concerned with all the information contained in the database. Instead, they need only a part of the database relevant to them. For example, even though the bank database stores a lot more information, an *account holder* would be interested only in the account details such as the current balance and the transactions made. They may not need the rest of the information stored in the account holders database. An e*xternal schema* describes each external view. The external schema consists of the definition of the logical records and the relationships in the external view.

In the external level, the different views may have different representations of the same data. The figure below (Fig. 2.6) describes the different views of the database related to different users.

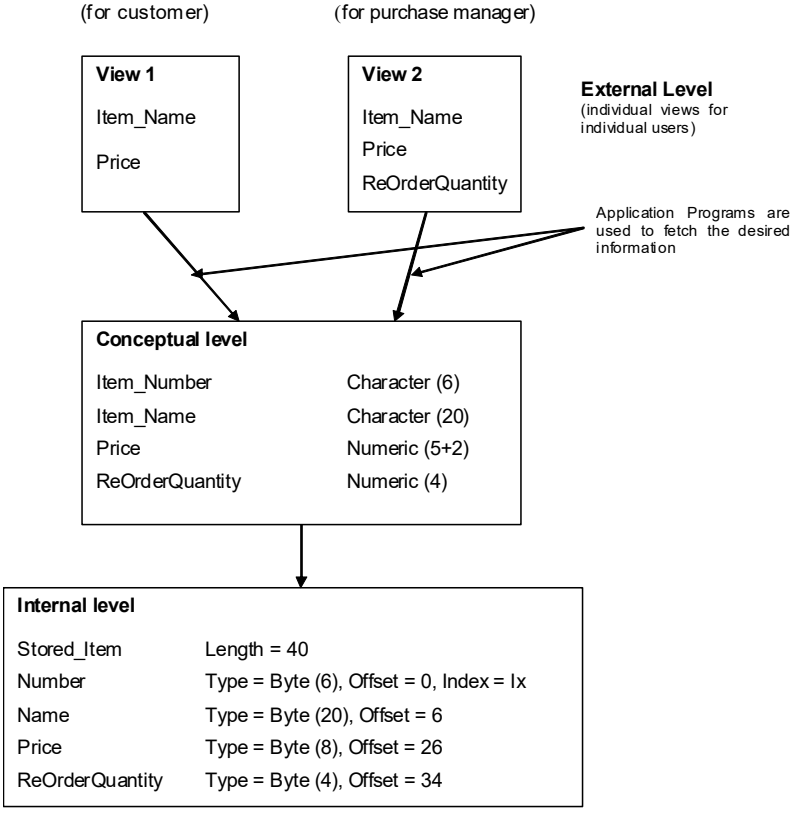


Fig.2.6: View of data at three-tier database architecture

Conceptual level :

Conceptual level is the middle level of the three-tier architecture. At this level of database abstraction, all the database entities and relationships among them are included. Conceptual level provides the community view of the database and describes what data is stored in the database and the relationships among the data. One conceptual view represents the entire database of an organization. It is a complete view of the data requirements of the organization that is independent of any storage consideration. The *conceptual schema* defines conceptual view. It is also called the logical schema. There is only one conceptual schema per database.

Internal level or physical level :

The lowest level of abstraction is the internal level. It is the one closest to the physical storage device. This level is also termed as physical level, because it describes how data are actually stored on the storage medium such as hard disk, magnetic tape etc. This level indicates how the data will be stored in the database and it also describes the data structures, file structures and access methods to be used by the database. The *internal schema* defines the internal level. The internal schema contains the definition of the stored record, the methods of representing the data fields and accessed methods used. The figure shows the internal view record of a database.

2.7 DATA INDEPENDENCE

Data independence is the characteristic of a database system to change the schema at one level without having to change the schema at the next higher level. This characteristic of DBMS insulates the application programs from changing the data. The data independence is achieved by DBMS through the use of the three-tier architecture of data abstraction. There are two types of data independence -

(i) Logical data independence

(ii) Physical data independence

2.7.1 Logical Data Independence

Logical data independence is the ability to change the conceptual schema without having to change the external schema or application programs. We may change the conceptual schema to expand the database(by adding a record type or data item) or to reduce the database(by removing a record type or data item). Only the view definition and the mapping need to be changed in a DBMS that supports logical data independence. After a logical change in the conceptual schema, the application programs that refer to the external schema construct must work as before.

2.7.2 Physical Data Independence

Physical data independence implies the ability to change the internal schema without changinig the conceptual(or external) schemas. Changes to the internal schema may be required for improving the performance of the retrieval or updation operations. In other words, physical data independence indicates that the physical storage structures or devices used for storing the data could be changed without changing the conceptual view or any of the external views.

2.8 DBMS LANGUAGE

A DBMS must provide appropriate languages and interfaces for each category of users to express database queries and updates. After completing the design of a data base, a DBMS is chosen to implement the database. It is important to specify first the conceptual and internal schemas for the database. Following laguages are used for specifying database schemas :

i. Data definition language (DDL)
ii. Storage definition language (SDL)
iii. View definition language (VDL)
iv. Data manipulation language (DML)

i) Data definition language (DDL)

DDL is a special language which species the database conceptual schema using set of definitions. DDL allows the DBA or the user to describe and name the entities, attributes and relationships required for the application, together with any associated integrity and security constraints. The DBMS has a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs.

For example, look at the following DDL statements :

```
CREATE TABLE EMPLOYEE
    (
    Fname          varchar(50     NOT NULL,
    Lastname       varchar(50)    NOT NULL,
    Eno            char(9)        NOT NULL,
    DOB            date,
    Address        varchar(60),

    PRIMARY KEY (Eno),
    );
```

The execution of the above DDL statements will create an EMPLOYEE table as shown below :

EMPLOYEE				
Fname	Lastname	Eno	DOB	Address

ii) Storage definition language (SDL)

Storage definition language is used to specify the internal schema in the database. In SDL, the storage structure and access memthods used by the database system is specified by set of statements.

iii) View definition language (VDL)

View definition language is used to specify user's views(external schema) and their mappings to the conceptual schema. There are two views of data - *logical view (*refers to the programmers view) and *physical view* (reflects the way how the data are stored on disk).

iv) Data manipulation language (DML)

DML provides a set of operations to support the basic data manipulation operations on data in a database. Data manipulation is applied to all the three(conceptual, internal, external)l levels of schema. The part of DML that provides data retrieval is called *query language*. DML provides the following data manipulation operations on a database :

- retrive data or records from database
- insert (or add) records to database
- delete records from database
- retrieve records sequentially in the key sequence
- retrieve records in the physically recorded sequence
- retrieve records that have been updated
- modify data or record in the database file

In other words, we can say that DML helps in communicating with the DBMS.

2.9 TYPES OF DBMS

The modern business environment revolves around the accuracy and integrity of information. The advancements in computer technology and rapid development of graphical user interface (GUI)-based applications, networking and communications have resulted in new dimensions in database management systems. A DBMS can be classified according to the number of users, the database site locations and the type and extent of use.

On the basis of number of users **:**

- Single-user DBMS
- Multi-user DBMS

Onthe basis of site locations **:**

- Centralised DBMS
- Parallel DBMS
- Distributed DBMS
- Client/Server DBMS

On the basis of the type and extent of use **:**

- Transactional DBMS
- Decision support DBMS
- Data Warehouse

We will mainly discass centralised, parallel, distaibuted and client/server DBMS in the following section.

2.9.1 Centralised DBMS

The centralised database system consists of a single computer system associated with its peripherals, physically located in a single location. The computer system offers data processing facilities to the users located either at the same site, or, at geographically dispersed sites, through remote terminials. The management of the system and its data are controlled centrally from any one or central site. The following figure shows a centralised database system.

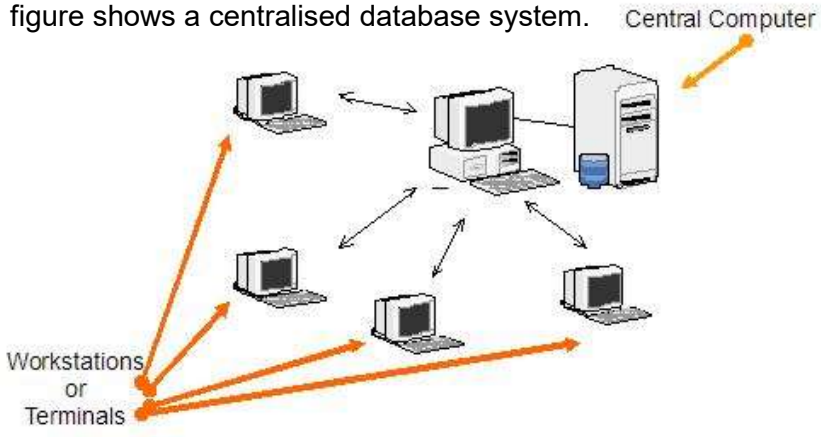


Fig.2.7: Centralised DBMS

Advantage of such sytem are given below :

- Most of the functions such as update, backup, query, control access etc. are easier with this system.

- Single database is shared accross the several different users.

Ofcourse, when the central site computer goes down, then every user is blocked from using the system untill it recover.

2.9.2 Parallel Database System

Parallel database systems architecture consists of a multiple central processing units (CPU) and data storage disk in parallel as shown in the figure. Hence, in such a system data processing speed is fast as well as input/output speed is also fast. The system, in which it needs to process an extremely large number of transactions per second, in such a system parallel database system is used.

Advantages of such sytem are given below :

- Useful in the applications, which have to process an extremely large number of transactions per seconds (of the order of thousands of transactions per seconds)
- Performance of such database system is very high.

The following figure shows a parallel database system.

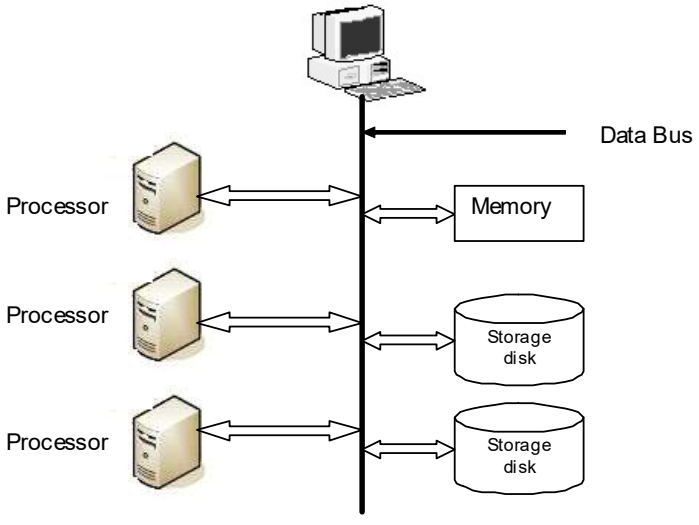


Fig.2.8: Parallel database system

2.9.3 Distributed DBMS

In a distributed database system, data are spread across a variety of different databases. These are managed by a variety of different DBMS

softwares running on a variety of different computing machines having different operating systems. These machines are actlually located on different sites and connected with some kind of communication networks as shown in the figure below. Thus, in a distributed database system, the organizations data might be distributed on different computers in such a way that data for one portion (or department) of the organization is stored in one computer and data for another department is stored in another computer. Each machine can have own data and applications and the users of one machine can access the data of several other computers.

The following figure shows a distributed database system.

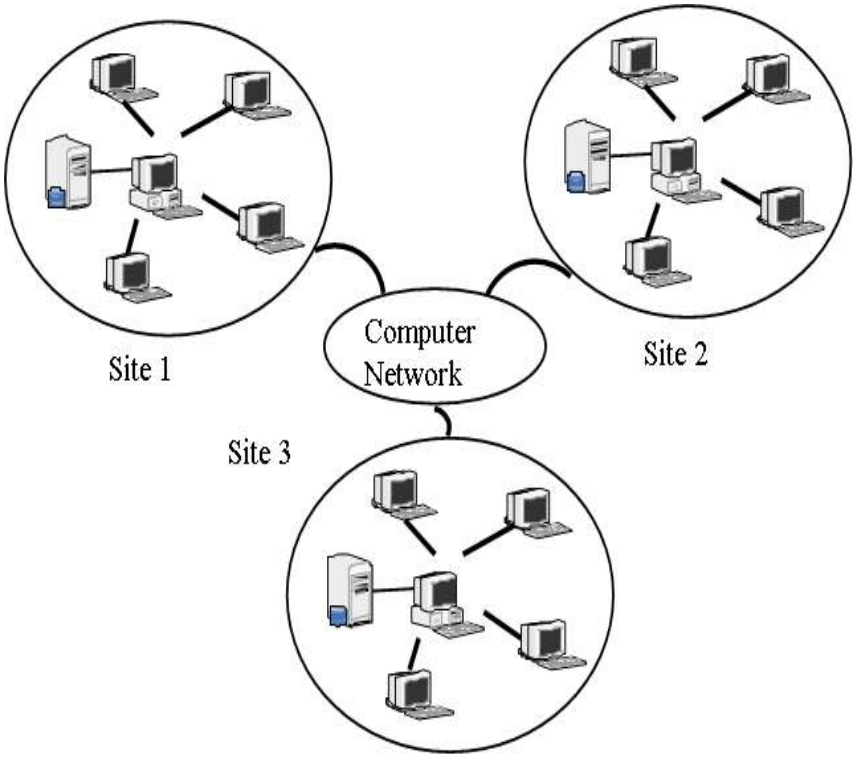


Fig.2.9: Distributed database

Advantages of such sytem are given below :

- efficiency and performancce of this system is high.
- a single database can be shared across several distinct client systems.
- As data volumes and transaction rates increase, the users can grow the system incrementally.

2.9.4 Client-Server DBMS

The client-server database system has two logical components namely - *client* and *server*. Clients are generally personal computers or workstations and the servers are the large workstations or mainframe computer system. The applications and tools of DBMS run on one or more client plateforms, while the DBMS softwares reside on the server. The server computer is called *backend* and the client computer is called *front-end*. The server and the clients are connected through networks. The clients send request to the server for performing some special tasks. The DBMS in the server side, in turn, process these requests and returns the results to the clients. The server handles parts of the job that are common to many clients, for example, database access and updates.

The following figure shows a client-server database model.

Advantages of such system are given below :

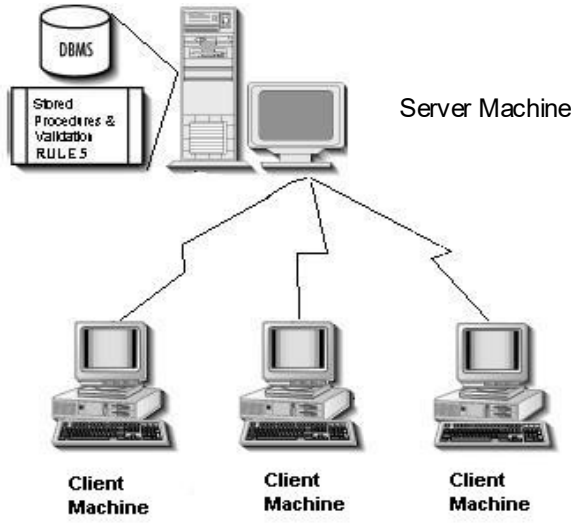


Fig 2.10 : Client-server database model

The advantages of such system are given below **:**

- Performance is high.
- A single database can be shared accross several distinct client system.

- More flexible as compared to the centralised system.
- Facilitates in more productive work by the users and making better use of existing data.

2.10 DATABASE ADMINISTRATOR

A database administrator (DBA) is a person or a group of person who is responsible for the environmental aspects of a database. A DBA is the central controller of the database system who designs database, controls and manages all the resources of database as well as provides necessary technical support for implementing policy decisions of database.

The role of a database administrator has changed according to the technology of database management systems (DBMSs) as well as the needs of the owners of the databases.

Some of the roles of the DBA may include

- Installation of new software — It is primarily the job of the DBA to install new versions of DBMS software, application software, and other software related to DBMS administration.
- Configuration of hardware and software with the system administrator — In many cases the system software can only be accessed by the system administrator. In this case, the DBA must work closely with the system administrator to perform software installations, and to configure hardware and software so that it functions optimally with the DBMS.
- Security administration — One of the main duties of the DBA is to monitor and administer DBMS security. This involves adding and removing users, administering quotas, auditing, and checking for security problems.
- Data analysis — The DBA will frequently be called on to analyze the data stored in the database and to make recommendations relating to performance and efficiency of that data storage.
- Database design (preliminary) — The DBA is often involved at the

preliminary database-design stages. Through the involvement of the DBA, many problems that might occur can be eliminated. The DBA knows the DBMS and system, can point out potential problems, and can help the development team with special performance considerations.

- Data modeling and optimization — By modeling the data, it is possible to optimize the system layouts to take the best advantage of the I/O subsystem.
- Responsible for the administration of existing enterprise databases and the analysis, design, and creation of new databases.



CHECK YOUR PPROGRESS

2. Select TRUE or FALSE in the following statements:
   (i) The conceptual view is a view of the totaldatabase content.
   (ii) User's view is also called external view.
   (iii) The database schema and an instance of the database are the same thing.
   (iv) A view of a database that appears to an application program is known as schema.
   (v) Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemes.
   (vi) A database is a computer-based record keeping system whose over all purpose is to record and maintain information.
3. Multiple Choice
   (a) A view of database that appear to an application program is known as –
      (i) schema (ii) subschema
      (iii) virtual table (iv) none of these
   (b) User's view is also called
      (i) external view (ii) conceptual view

(iii) internal view (iv) none of these

(c) Which of the following schemas defines the stored data structures in terms of the database model used -

(i) external (ii) conceptual

(iii) internal (iv) none of these

(d) Data is processed by using

(i) DDL (ii) DML

(iii) DCL (iv) DPL

(e) Immunity of the conceptual (or external) schemas to changes the internal schemas is referred to as

(i) physical data independance

(ii) logical data independence

(iii) both (i) and (ii)

(iv) none of these



2.11 LET US SUM UP

- The traditional file approach to information processing has for each application a separate master file and its own set of application programs, COBOL language used to write these application programs.
- A database is a single organized collection of instructed data, stored with a minimum of duplication of data items so as to providea consistent and controlled pool of data.
- A database management system (DBMS) is a collection of programs that enables the users to store, modify and extract information from a database as per the requirements. DBMS is an intermediate layer between programs and the data. Programs access the DBMS, which then accesses the data.
- According to the ANSI/SPARC architecture of a database system the whole database is divided into the following three levels :
  - ❍ External level or view level
  - ❍ Conceptual level
  - ❍ Internal level or physical level
- Logical data independence indicates that the conceptual schema can

be changed without affecting the existing external schema.

- Physical data independence indicates that the physical storage structures or devices used for storing the data could be changed without necessitating a change in the conceptual view or any of the external views.
- DBMS provide appropriate languages and interfaces for each category of users to express database queries and updates.
- Following languages are used for specifying database schemas :
  - ❍ Data definition language (DDL)
  - ❍ Storage definition language (SDL)
  - ❍ View definition language (VDL)
  - ❍ Data manipulation language (DML)
- DBMS can be classified according to the number of users, the database site locations and the type and extent of use.

  On the basis of site locations, the followings are types of DBMS :
  - ❍ Centralised DBMS
  - ❍ Parallel DBMS
  - ❍ Distributed DBMS
  - ❍ Client/Server DBMS

  On the basis of the type and extent of use DBMS are of following types :
  - ❍ Transactional DBMS
  - ❍ Decision support DBMS
  - ❍ Data Warehouse
- DBA is database administrator who is responsible for maintaining database. A DBA provides the necessary technical support for implementing policy decisions of database.



2.12 FURTHER READING

1. Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of database systems*. Pearson.
2. Date, C. J. (2006). *An introduction to database systems*. Pearson Education India.

2.13 ANSWER TO CHECK YOUR PROGRESS

1. a. (iv) b. (iv) c. (ii) d. (i) e. (i)
2. (i) False (ii) True (iii) False (iv) False (v) True (vi) True
3. a. (ii) b. (i) c. (ii) d. (ii) e. (i)

2.14 MODEL QUESTIONS

1. What is the file based approach of database? Explain its limitations?
2. Explain the three levels of database architecture. What are its objectives?
3. Define data independence and its types? How is data independence achieved?
4. State the advantages of DBMS?
5. Discuss the main disadvantages of a Traditional file approach?
6. Discuss the main disadvantages of DBMS?
7. Differenciate between DBMS approach and the traditional file approach.
8. Mention the differences between text files and database files. Why are database files preferred in a commercial organization?
9. Write short notes on :
   (i) Data independence
   (ii) Database
   (iii) DBMS
   (iv) DBMS Architecture
   (v) Client-server database model
   (vii) Distributed database system
   (vii) Physical Data Independence
   (viii) traditional File Approach
   (ix) Centralised database system
   (x) DBMS language
10. What is logical data independence and why is it important?
11. Explain the difference between logical and physical data independence.
12. Describe the three levels of data abstraction?
13. What do you mean by Database Language? What are the different types of data base language?
14. Explain the role of a Database Administrator.

***

UNIT 3: DATA MODEL

UNIT STRUCTURE

3.1 Learning Objectives
3.2 Introduction
3.3 Data Models
  3.3.1 Conceptual Model
  3.3.2 Logical Model
  3.3.3 Physical Model
3.4 Entity Relationship Model
  3.4.1 Entity, Attribute and Relationship type
  3.4.2 Degree of Relationship type
  3.4.3 Constraints on Relationship type
  3.4.4 Weak and Strong Entities
3.5 Conversion of ER model into Relational Schema
3.6 ER Modelling Symbols
3.7 Let Us Sum Up
3.8 Answers to Check Your Progress
3.9 Further Reading
3.10 Model Questions

3.1 LEARNING OBJECTIVES

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

- describe different data models
- identify the categories of data model used in database management system
- identify the symbols to describe the ER model
- design the ER diagram with its constraints
- describe conversion procedure from ER diagram to relational databases and vice versa

3.2 INTRODUCTION

In our previous units we have discussed the various approaches of database system and their architecture. We have come across the schema diagram of DBMS architecture, their types and also their merits and demerits.

Now in this unit, we will be able to learn a brief description of all the three approaches and the various types of data model that are categorised on the basis of these concepts. We will also discuss in detail the design approach of Entity Relationship (ER) model with all their constraints. ER model is primarily a semantic model and is very useful in creating raw database design that can be further normalised. This is usually used for real world application development process. Along with this we will discuss the basic conversion procedure of ER model into relational schema with suitable example. We will conclude this unit with the modelling symbol used in ER diagram.

3.3 DATA MODELS

There are many basic structures that exist in a database system to organise the data. One of the fundamental characteristics of the database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by most database users. Data model is the main tool to provide this abstraction. A data model is a set of concept that can be used to describe the structure of a database. Each structure of a database has its own data type, relationships of one data structure to another and constraints that should hold on the data. Most data models also include a set of operation for specifying retrievals and updates on the database. So, a data model defines

- The logical data structure
- Data relationships
- Data consistency constraints.

Data models are categorised in different ways. The most general category is on the basis of the concepts they provide to describe the database structure. They are: *Conceptual Model, Logical Model, and Physical Model.*

3.3.1 Conceptual Model

Conceptual model is also called **High Level Data Model**. It provides the concepts that are close to the way by which many users perceive data. It uses concepts such as entities, attributes and relationship types. An *entity* is an object which is used to represent a database. An *attribute* is a property that describes some aspects of an object. *Relationship* is the relation between entities that is easily represented in high level data models. It is sometimes called **Object-Based Model** because it uses objects as key data representation components that contain 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 interrelationships are complex, for example, Computer Assisted Design based components. The popular high level data model is Entity-Relationship Model.

3.3.2 Logical Model

Logical model is also called Implementation data model. It provides concepts understandable by the end users but is not too far removed from the way the data is organised within the computer. It hides some details of data storage but can be implemented on a computer system in a direct way. This model is sometimes called **Record-based data model** because it uses records as the key data representation components. It is used most frequently in current commercial DBMSs and includes the three most widely used data models - *Relational*, *Network* and *Hierarchical data model*.

3.3.3 Physical Model

It is the low-level data model. It provides concepts that describe the details of how data is stored in the computer by representing information such as record formats, record orderings and access paths. An access

path is a structure that makes the search for particular database records much faster.



CHECK YOUR PROGRESS

1. State true or false (T/F) of the following:

a) Data model gives logical as well as physical data structure.

b) Conceptual data model is also called object-based model.

c) High level data model uses records as the key data representation component.

d) Low level data model describes the details of how to store data in computer.

e) Logical data model provide concept which is complicated for end users.

f) Implementation data model is also a record-based data model.

g) Access path can search database record much faster.

h) Three most widely used data models belongs to the physical data model

3.4 ENTITY RELATIONSHIP MODEL

Entity-Relationship (ER) model is a high-level conceptual data model developed to facilitate database design. The ER model concepts are designed to be closer to users' perception of data and are not meant to describe the way in which data will be stored in the computer. It is the generalised data model. Although ER model has some means of describing the physical database model, it is basically useful in the design of logical database model. This analysis is then used to organise data as a relation, normalizing relation and finally obtaining a relational database model.

Some of the main features of ER model are:

- It is a high level conceptual data model.
- It allows describing the data involved in a real-world enterprise in

terms of objects and their relationships.

- This is widely used to develop an initial design of a database.
- Describes data as a collection of entities, relationships and attributes.

3.4.1 Entity, Attribute and Relationship type

ER model uses three features to describe data. These are: *entity*, *attribute* and *relationship*.

Entity:

- An entity is an object which specifies distinct things in the real world items, e.g., car, book, table, student, employee etc.
- It need not be a physical entity; it can also represent a concept in real world like project, loan, account etc.
- It represents class of things, not any one instance. For example, 'STUDENT', 'EMPLOYEE' etc.

Each entity has attributes. For example, an employee may be described by the employee's name, employee id, age, address, designation and salary. In figure 3.1(a) and 3.1(b), two entity types EMPLOYEE and STUDENT, are shown with their different attributes.

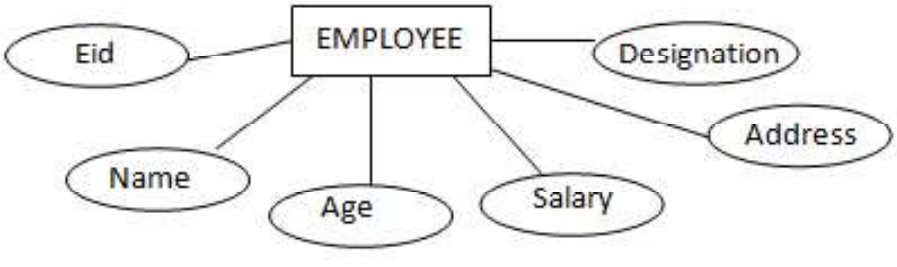


Figure 3.1(a): EMPLOYEE entity type with six different attributes.

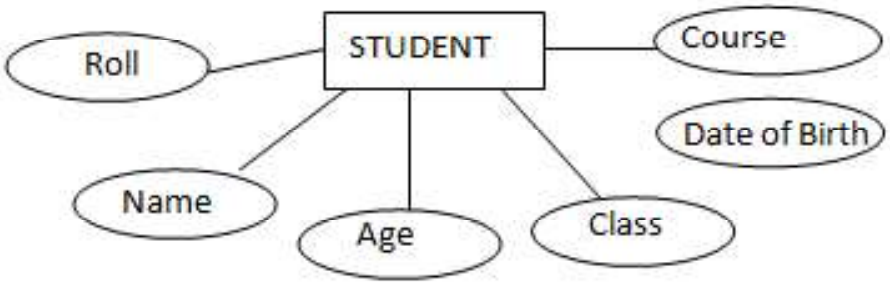


Figure 3.1(b): STUDENT entity type with six different attributes.

Entity Type:

A collection of a similar kind of entities having same attribute is called an ***Entity Set*** or ***Entity Type***. For most databases there may be many entity types. An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name. Attribute names are enclosed in ovals and are attached to their entity type by straight lines. [Fig. 3.1 (a) and Fig. 3.1 (b)]

Key Attributes of an Entity Type: 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. For example, Roll attribute is a key of the STUDENT entity type.

Attributes:

The properties of an entity that describes the specific feature of the entity is called *attribute*. In case of a student entity, 'Roll No', 'Student's name', 'Age', 'Marks' etc. are attributes.

Each entity posseses values for their attributes. For example, in case of STUDENT entity type, the attributes are Roll, Name, Age, Class, Date of Birth and Course. The values for those attributes may be like :

Roll =1, Name = Rishi , Age =17, Class = Twelve, Date of Birth = 21/07/2002 , Course= Science.

The attribute values are different for differnet students. There may be some common attribute values for two different students.

Value Sets (Domains) of attribute:

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.

For example, in the Figure 3.1 (a), if the range of attribute 'age' allowed is in between 20 and 50, we have to specify the value set of the age of EMPLOYEE to be the set of integer numbers from 20 to 50. Similarly, we can set the value set for 'Name' attribute as the set of strings of alphabetic characters separated by blank characters and so on.

Types of attributes:

Attributes attached to an entity can be of various types.

i) **Simple (atomic) attribute:** The attribute that cannot be further divided into smaller parts and represents the basic meaning is called the simple attribute. For example, pin code of a person.

ii) **Composite attribute:** Attribute that can be further divided into smaller unit and each individual unit contains a specific meaning. It is composed of several more basic attributes. For example, address of a person is composed of zip code, street, district etc.

iii) **Single valued attribute:** Attribute having a single value for a particular entity. For example, a person entity having one value for age. So age is a single-valued attribute.

iv) **Multi valued attribute:** Attribute that has a set of values for the same entity is called a multi valued attribute. Different entities may have different number of values for this kind of attribute. For multi valued we much also specify the minimum and maximum number of values that can be attached. For example, college degree attribute for a person.

v) **Stored attribute:** Attributes that are directly stored in the database. For example, birth date attribute of a person.

vi) **Derived attribute:** attributes that are not stored directly but can be derived with the help of some other attribute. For example, a person's age is derived from date of birth of that person.

Relationship:

In most databases there will be more entities and entities are joined with each other by some relations called relationships between entities. For example, entity EMPLOYEE has the relationship with entity DEPARTMENT. A relationship can be defined as:

- A connection or set of associations, or
- A rule for communication among entities.

Relationship sets:

A relationship set is a set of relationships of the same type. Collection of all the instances of relationship forms a relationship set called *relationship type.*

For example, consider a relationship type WORKS_FOR between the two entities EMPLOYEE and DEPARTMENT, which associates each employee with the department the employee works for. Each relationship instance in WORKS_FOR associates one employee entity and one department entity.

3.4.2 Degree of Relationship Type

The degree of a relationship type is the number of participating entity type. A relationship type of degree two is called binary relation and of degree three is called ternary relation. An example of binary and ternary relationship type are explained in Figure 3.2, where the degree of WORKS_FOR relationship type is two and for SUPPLY the degree is three.

(a)

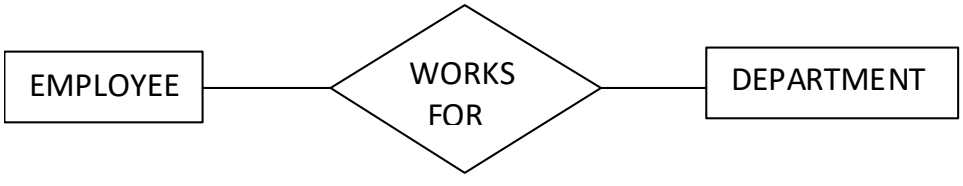


Figure 3.2: (a) binary relationship WORKS_FOR

(b)

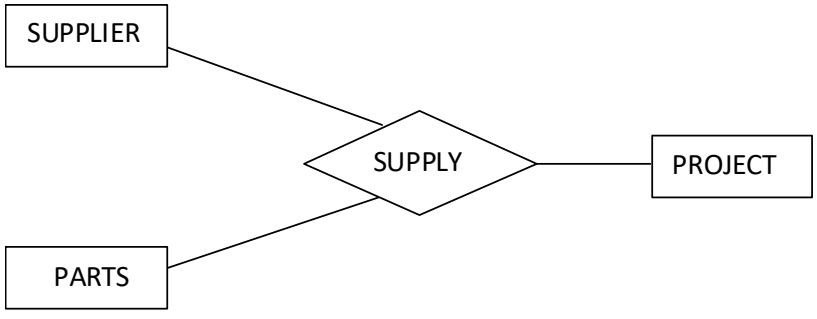


Figure 3.2 (b) ternary relationship SUPPLY

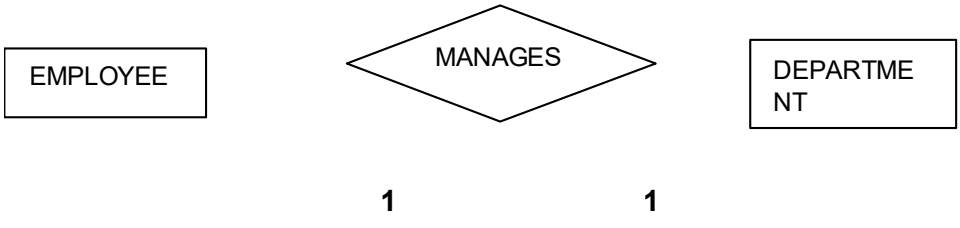
3.4.3 Constraints on Relationship type

Relationship types usually have certain constraints that limit the possible combinations of entities participating in relationship instances. These constraints are determined from the mini world situation that the relationships represent. The two main types of relationship constraints that occur relatively frequently are cardinality ratio and participation constraints.

a) Cardinality Ratio:

This constraint specifies the number of relationship instances that an entity can participate in. Common cardinality ratios for binary relationship types are **1:1, 1:N** and **M:N**.

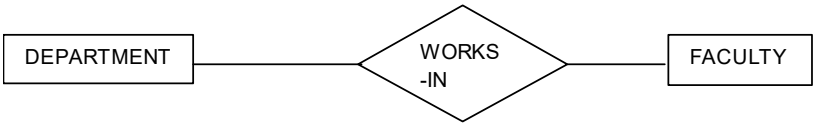
- **One-to-one (1:1):** This is the relationship where an entity in an entity type 'A' is associated with at most one entity of the entity type 'B' and an entity in entity type 'B' is associated with at most one entity in entity type 'A'. For example, the relationship between employee and department.



It is 1:1 relationship type, because it is assumed that at least one employee should be there to manage one department and one department should have one manager.

Similarly, we can define the relationship between college and principal because we know that one college can have at most one principal and one principal can be assigned to only one college.

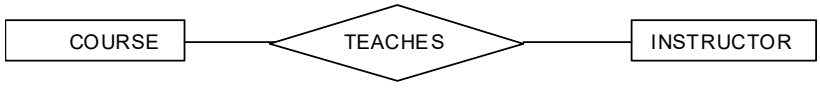
- **One-to-many (1:n):** If an entity in entity type 'A' is associated with any number of entities in entity type 'B' and an entity in entity type 'B' is associated with at the most one entity in entity type 'A', then this type of ratio is called 1:n cardinality ratio. For example, the relationship between department and faculty.



1 **N**

Here, in the example, it is considered that a particular department may have more than one faculty members but a faculty member can be an employee of only one department.

- **Many-to-one (m:1):** In this type of ratio, an entity in entity type 'A' is associated with at most one entity in entity type 'B' or an entity in entity type 'B' is associated with any number of entity in entity type 'A'. For example, the relationship between course and instructor.

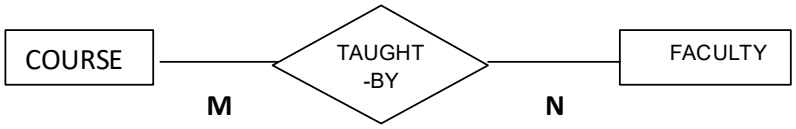


M **1**

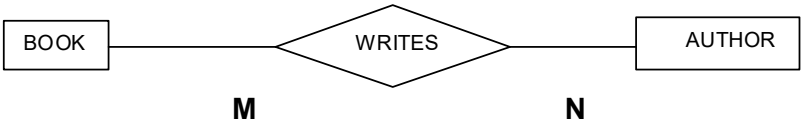
It is assumed that an instructor can teach various courses but a course can be taught only by one instructor.

- **Many-to-many (m:n):** Entities in entity type 'A' and 'B' are associated with any number of entities from each other. For example, if we assume that one faculty member can be assigned to teach many courses and one course may be taught by many faculty

members, then the relationship taught-by will be a many-to-many relationship.



Like relationship between book and author, if we assume that one author can write many books and one book can be written by more than one authors, then it will be many-to-many relationship.



b) 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 two types of participation constraints: **total** and **partial**.

Total participation constraints: When all the entities from an entity set participate in a relationship type, is called **total participation**. It is denoted by double line ' ═══ ' sign.

Partial participation constraints: When it is not necessary for all the entities from an entity set to participate in a relationship type, it is called partial participation and is denoted by '▬' sign.

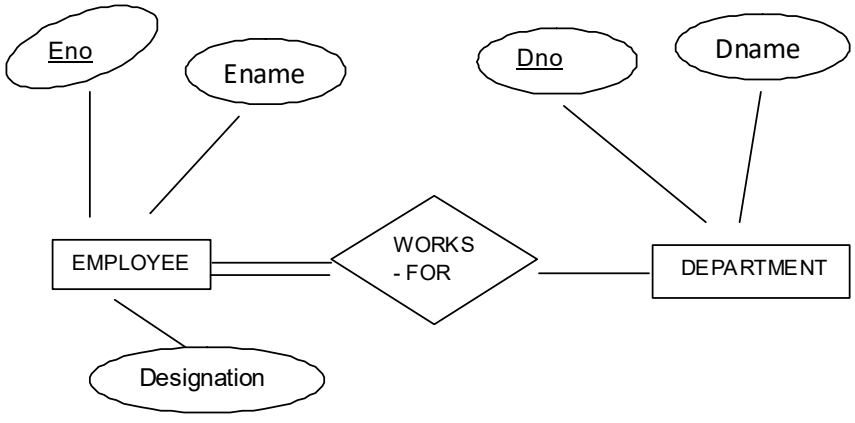


Figure 3.3: Total and Partial participation constraints

In the Figure.3.3, the relationship WORKS-FOR has two entities EMPLOYEE and DEPARTMENT. The participation constraints of entity EMPLOYEE is total and of entity DEPARTMENT is partial with relationship type WORKS-FOR, if we assume that,

i) All employees works for any one or more department, then if it is a total participation constraint.

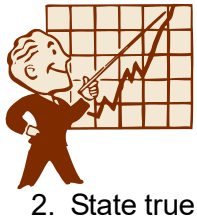
ii) There are some departments which do not have any employee. It means that all departments of DEPARTMENT entity are not dependent on the entity EMPLOYEE via relationship WORKS-FOR. So, it is a partial participation constraint.

3.4.4 Weak and Strong Entities

Entities are of two types: *strong entities* and *weak entities*.

i) Strong entity type: The entity type containing a key attribute is called a strong entity type or a regular entity type. For example, if the entity 'student' has a key attribute 'Roll No' which is used to uniquely identify the entity 'student', then the entity 'student' is called a strong entity.

ii) Weak entity type: The entity type without having any key attribute is called weak entity type. For example, an employee wants to keep track of the information of their dependents and the attributes of dependent entity is name, sex, and relationship with employee. Here, the entity dependent does not have any key attribute; hence this type of entity is called weak entity.



CHECK YOUR PROGRESS

2. State true or false (T/F) for the following

a) An entity is an object used to represent things.

b) Entities having key attributes are called weak entity.

c) Strong entity is also called regular entity.

d) Domain may be the range of values used against attribute.

e) Composite attribute cannot be divided into smaller parts.

3. Fill in the blanks in of the following studences with appropriate word given below:

attributes, many-to-many, diamond, association, ER.

a) An _____ of several entities in an Entity-Relation model is called relationship.

b) The various kinds of data that describes an entity are known as its _____.

c) A weak entity set is represented by a doubly outlined rectangle in the _____ diagram.

d) In an ER diagram a _____ represents a relationship

e) A _____ relationship describes entities that may have many relationships in both the directions



ACTIVITY -1

1. Consider a COLLEGE database that keeps track of students, faculty, department and courses organised by various departments. College contains departments and 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.

a) Find out the possible entities and their relationship types.

b) List out the possible key attributes of each entities.

c) Give the cardinality ratio between faculty and department entities.

d) Show the participation constraints of entities students and courses needed by the departments

3.5 CONVERSION OF ER MODEL INTO RELATIONAL SCHEMA

For every ER diagram we can construct a relational database which is a collection of tables called schema diagram. Following are the set of

steps to convert ER diagram to a relational schema.

Conversion of entity sets:

a) For each strong entity type E in the ER diagram, we create a relation R containing all the simple attributes of E. The key attribute of the relation R will be one of the key attribute of R. For example, the STUDENT entity having Roll No as a key attribute, the relational schema will be as shown below

Roll No : primary key	**name**	**address**

Similarly, we can convert other entities also into corresponding relational schema.

b) For each weak entity type W in the ER diagram, we create another relation R that contains all the simple attributes of W. If E is an owner entity of W then the key attribute of E is also included in R. This key attribute of R is a set as a foreign key attribute of R. Now the combination of primary key attribute of owner entity type and partial key of the weak entity type will form the key of the weak entity type. For example, there is a weak entity GUARDIAN, where any key field of student entity RollNo has been added.

RollNo	Name	Address	Relationship
(Primary Key)			

Conversion of relationship sets:

Binary relationships:

a) 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 the entities (say E1) preferably with total participation and add primary key attribute of another entity E2 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, considering DEPARTMENT entity having attributes DNO, DNAME and 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. Then the relational schema will be as:

DEPARTMENT

Dno	Dname	Head_Id	Date_from

b) **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, consider a relationship WORKS_ON between the DEPARTMENT and FACULTY entities. 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. The relational schema for this relationship will be as:

FACULTY (CONTAINS WORKS_ON RELATIONSHIP)

Id	Name Address	Basic_pay	DNO

c) **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 rela-

tionship type or simple components of a composite attribute is also included as attributes of S. For example, consider a 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. The relational schema for TAUGHT_BY relationship will be a new table as:

TAUGHT_BY

ID (Primary key of FACULTY table)	Course_ID (Primary key of COURSE table)

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.

Multi valued attributes:

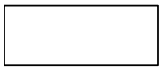
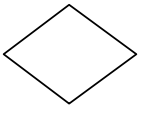
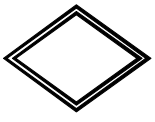
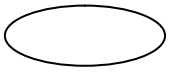
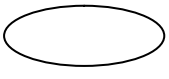
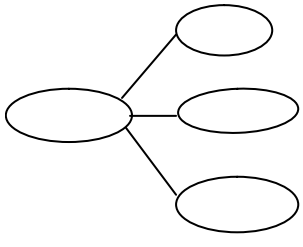
For each multi valued attribute 'A', we create a new relation R that includes an attribute corresponding to the primary key attribute k of the relation that represents the entity type or relationship type that has as an attribute. The primary key of R is then combination of A and k.

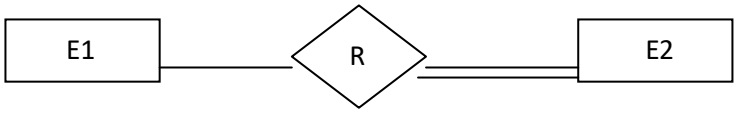
For example, if the STUDENT entity has attributes Roll No, Name and Phone No with Roll No as a primary key and Phone No as a multi valued attribute, then we will create a table PHONE(Roll No, Phone No) where primary key is the combination. In this

case the STUDENT table need not have the attribute Phone No; instead it can be simply Roll No and Name.

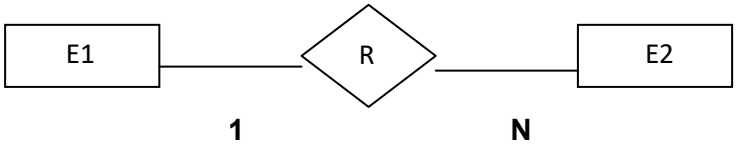
3.6 ER Modelling Symbols

We can express the overall logical structure of a database graphically by using ER diagram. ER diagram is composed according to the following modelling symbols:

Symbol	Meaning	Symbol	Meaning
	Entity		Relationship
	Weak Entity		Identifying Relationship
	Attribute		Multi Valued Attribute
	Key Attribute		Derived Attribute
			Composite Attribute



Total participation of E2 in R



1 **N**

Cardinality ratio 1:N for E1:E2



CHECK YOUR PROGRESS

4. State whether the following are true or false (T/F)

a) For weak entity, only the key attributes of owner entity will form relational table.

b) For each and every ER diagram, we cannot construct a relational schema.

c) For strong entity type, all the keys of those entities will form the relational schema.

d) We cannot form a relational schema for relationships of entities.

e) For 1:1 relationship, only the key attributes of entities of both sides will form relational schema.

5. Fill in the blanks:

a) We include all simple attributes in relational schema for _____ relationship type.

b) For each _____ in the ER diagram, we create another relation that contains all simple attributes.

c) For _____ relation, the primary keys only form a new relation.

d) Relational database is a collection of tables called



ACTIVITY 2

Consider the ER diagram of COLLEGE given in the Activity 1, and construct the corresponding relational schema for that.



3.7 LET US SUM UP

The ER model explained in this unit covers the basic aspects of ER modelling, database designing procedure to store data in a database with their basic symbols to represent.

- A data model is a set of concept that can be used to describe the structure of a database.
- Data models are of three types: Logical, Physical and Conceptual.
- The most often used data models in current commercial DBMSs are Relational, Network and Hierarchical models.
- E-R is a high-level conceptual data model. It is basically useful in the design of logical data base model. It describes data as a collection of entities, relationships and attributes.
- An entity is an object which specifies distinct things in the real world items, e.g., car, book, table etc. A collection of a similar kind of entities having same attributes is called an Entity Set or entity type.
- An entity having key attribute is called strong entity or a regular entity and without key attribute is called weak entity.
- The properties of entity that describes specific feature of entity is called attribute. These are of different types that described in this unit
- The relations by which entities are joined each other is called relationships between entities.
- Representation of entities with tabular form is called schema diagram. For every ER diagram we can construct a relational database which is a collection of tables. The attributes of an entity will be also attributes of the schema diagram.
- For each relationship type there will be one schema diagram.



3.8 ANSWERS TO CHECK YOUR PROGRESS

1. a) False b) True c) False d) True
   e) False f) True g) True h) False

2. a) True b) False c) True d) True
   e) False

3. a) association b) attributes c) E-R d)diamond
   e) many-to-many

4. a) True b) False c) True d) False
   e) False

5. a) 1:N b) Weak entity type c) M:N
   d) schema diagram



3.9 FURTHER READING

1. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems. Pearson.
2. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



3.10 MODEL QUESTIONS

Q1. What do you mean by Data Model? What does it define?
Q2. Describe conceptual, logical and physical data model.
Q3. Design an E-R diagram for airline reservation system consisting of flights, aircrafts, airports, fares, reservations, tickets, pilot, crew and passengers. Clearly highlight the entities, the relationships, the primary keys and the mapping constraints.

4. What do you mean by Constraints of Relationship type? Explain with example the constraint of different cardinality ratio.
5. Discuss about participation constraints used in E-R diagram design. Distinguish between Total participation constraints and Partial participation constraints.
6. What is Cardinality Ratio in ER diagram?
7. Discuss the procedure of converting E-R model into relational schema.

9 Can you convert a relationship type of an E-R model into corresponding relational schema? If Yes, how?

10. Explain E-R modelling symbols.

****

UNIT 4: RELATIONAL MODEL

UNIT STRUCTURE

4.1 Learning Objectives
4.2 Introduction
4.3 Relational Data Model Concept
    4.3.1 Relational Schema and Instances
4.4 Integrity Constraints
    4.4.1 Entity Integrity Constraints
    4.4.2 Referential Integrity Constraints
    4.4.3 Domain Constraints
4.5 Let Us Sum Up
4.6 Further Reading
4.7 Answers to Check Your Progress
4.8 Model Questions

4.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- define Relational model
- define the terminologies used in Relational model
- describe the Schema design of a database
- define various Integrity Constraints
- describe how data are organised in the form of tables

4.2 INTRODUCTION

In the previous unit, we have discussed the properties of data models and the designing procedure of Entity-Relationship model with symbols. We also discussed the conversion procedure of ER model to its Relational data model.

This unit focuses on Relational model. Most of the commercial DBMS products available in the industry are relational at core. In this unit we will discuss the terminology and operations used in relational model. The restrictions associated with the formulation of relation table will also will be discussed in this unit.

4.3 RELATIONAL DATA MODEL CONCEPT

We have already come to know across that a data model in a database system is basically a structure or an organisation of data and a set of operations on that data. Relational data model is one of the traditional data model. This is the most widely used database model that represents data as well as relationship among the data in the form of tables. This is a very simple model and is based on a proven mathematical theory. In this model, a database is represented as a collection of "Relations", where each relation is represented by a two dimensional table. This model is most commonly used in real world due to its simplicity. Following figure (4.1) represents a simple "STUDENT" relation:

R_NO	S_NAME	ADDRESS	MARKS
10	Sanjib Kaur	Block -4, Noonmati	69
12	Padip Sen	Ganeshguri	75
15	Bipul Prasad	Bamunimaidan	58

Fig. 4.1: A sample STUDENT relation

Properties of a table

A table should contain the following properties:

a) Each entry in a table represents one data item and two column headings with the same name are not allowed.
b) In each column, the data items are of the same data type.
c) Each column is assigned with a distinct heading.
d) All rows are distinct; duplicate rows are not allowed.

e) Both the rows and the columns can be viewed in any sequence at any time without affecting the information.

A relational database model uses a collection of tables to represent both the data and the relationships among those data items. Each table has multiple columns and each column has a unique name.

Properties of RDBMS

A relational Database Management System (RDBMS) has the following properties:

a) Stores data in the form of tables.
b) Does not require the user to understand its physical implementation.
c) Provides information about its contents and structure in the system table.
d) Supports the concept of NULL values.

Advantages of Relational Data Model:

Following are some of the **advantages** of relational model:

a) **Ease to use:** Convenient to define and query the database as tables contain rows and columns. It is quite natural even for first time users.
b) **Flexibility:** Data can manipulate easily with the help of relational operations.
c) **Precise:** Since it uses mathematical operations, it can ensure accuracy and less ambiguity as compared to other models.
d) **Security:** security control and authorization can also be implemented more easily. It has the user's own authorization controls.
e) **Data Independence:** Data independence is achieved more easily with normalization structure used in a relational database.
f) **Data Manipulation Language:** The possibility of responding to query on relational algebra and relational calculus is easy in the relational database approach.

The basic terminologies used in relational database model are:

- **Tuple (record) and attribute:**

Each row in a table is called a tuple and a column name is called an attribute. For example, Figure 4.1 represents a STUDENT relation where ROLL NO, NAME, ADDRESS and MARKS are attributes and each entry against these attributes is called tuple of relation STUDENT.

- **Domain:**

A domain is a collection of all possible values from which the values for a given column or attribute is drawn. So, every attribute in a table has a specific domain. Values to these attributes cannot be assigned outside their domains. For example, the domain of attribute NAME is the set of all alphabetic string of finite length and the domain of a MARKS attribute should not be greater than 100 for the relation STUDENT in Figure 4.1.

- **Relation:**

The table with all tuples and attributes is called relation. It has three components: **Name** that is represented by the title of the relation, **Degree**, the number of column associated with the table and the **Cardinality**, the number of rows in the table. For example, figure 4.1 represents a relation named STUDENT of degree 4, because it has total four attributes, and the cardinality for this relation is 3(number of rows).

4.3.1 Relational Schema and Instances

A relation consists of Relational Schema and Relation Instances.

Relational Schema: A schema specifies the relation name, its attributes and the domain of each attribute. The description of a database is called database schema. It is specified during database design and is not expected to change frequently. If R is the name of a relation and A1, A2, A3..... An are the attributes of R, then R (A1, A2,.....An) is called a relational schema. Each attribute takes values from a specific domain D. A displayed schema is called a schema diagram. It shows only the specified

data items, not the data types or relationships among various other files. For example, the relational schema diagram for the relation "STUDENT" is given below:

STUDENT

R_NO	S_NAME	ADDRESS	MARKS

where, STUDENT is the name of the Relation and R_NO, S_NAME, ADDRESS, and MARKS are four different attributes that represent the relational schema.

Relation Instance: The data in the database at a particular moment in time is called a database instance. It is also called an occurrence or state. Many database instances are corresponding to a particular database schema.

We can define the relational instances as,

A relation instance denoted as 'r' is a collection of tuples for a given relational schema. The relation state of the relational schema R (A1, A2,.....An) is denoted as r(R) is a set of n-tuples.

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



CHECK YOUR PROGRESS

1. Fill in the blanks for the following.

i. In a relational database model, the columns of a table are called _____.

ii. _____ is the rows in a table.

iii. _____ is a collection of related files.

iv. The number of columns in a table is the _____ of the relation.

v. _____ is the number of rows in a table.

vi. An entity is an object that is distinguishable from other objects by

a specific set of _____

vii. Collection of tuples for a relational schema at a particular time is called _____

viii. A relation state of a relational schema R denotes as _____ if r is the collection of tuples.

ix. A relation state is also called _____

x. Relational schema contains name and _____of that relation.

4.4 INTEGRITY CONSTRAINTS

The term ***integrity*** refers to the accuracy or correctness of data in the database schema and is expected to hold on every database instance of that schema. Relational model includes two general integrity constraints. They are: Entity Integrety Constraints (Rule 1) and Referential Integrity Constraints (Rule 2)

4.4.1 Entity Integrity Constraints

Entity Integrity Constraints state that no primary key value can be NULL. This is because we use the primary key value to identify individual tuples in a relation. It ensures that instances of the entities are distinguishable i.e., they must have a unique identification of some kind. Primary keys perform that unique identification function in a relational database.

4.4.2 Referential Integrity Constraints

Referential Integrity Constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations (not necessarily be distinct). It uses a concept of foreign key which will be explained more details in the subsequent unit. Informally, it states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. Considering the following relations,

EMPLOYEE

p.k **f.k**

ENO	ENAME	DNO
101	Robert	10
102	Smith	12
103	Robindra	12
104	John	10

DEPARMENT

p.k

DNO	DNAME	LOCATION
10	Comp. Sc.	Jalukbari
12	Electronic Sc.	Guwahati

Fig. 4.2 relational database table showing referential integrity

In figure 4.2, EMPLOYEE and DEPARTMENT are two relations where ENO and DNO are primary keys respectively. Here the attribute DNO of EMPLOYEE table is a foreign key that gives the department number for which each employee works. Hence its value in each EMPLOYEE tuple must match the DNO value of some tuple in the DEPARTMENT relation.
To specify the Referential Integrity constraints between two relations R1 and R2, the following properties have to be satisfied:

a) The Foreign key attribute of a relation have to be same domain with the primary key of the other relation.
b) The value of foreign key in a tuple t1 of relation R1 occur either a value of primary key of tuple t2 in R2 or to be NULL.

4.4.3 DOMAIN CONSTRAINTS

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

- Standard numeric data types for integer
- Real numbers
- Characters
- Fixed length 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.



CHECK YOUR PROGRESS

2. Fill in the blanks for the following

i. In a relational database, _____ are not allowed to have null values.

ii. In a relational database, a referential integrity constraint is specified with the help of _____.

iii. The _____ in a record is a unique data item.

iv. In a relation, each _____ name must have different meaning.

v. According to integrity rule 1, two _____ should be distinguishable from each other.

3. Write True or False.

i. Prime attribute have unique identifier.

ii. The key field may be NULL in relational database.

iii. Foreign key concept is used to explain Integrity Rule 1.

iv. Domain constraint specifies the values related to

a. instances.

v. According to referential integrity constraints, values of foreign key field of that table are dependent on the values of primary key field of another table



4.5 LET US SUM UP

- 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. Each table has multiple columns, and each column has a unique name.
- In Relational Database Model each row in a table is called a tuple and a column name is called an attribute. The table with all tuples and attributes is called relation.
- A domain is a collection of all possible values from which the values for a given column or attribute is drawn.
- A Relational Table has three components: Name that represent by the title of the relation, Degree, the number of column associated with the table and the Cardinality, the number of rows in the table.
- A Relational Schema of Relational Model specifies the relation name, its attributes and domain of each attribute and the Instances are collection of tuples for a given relational schema.
- The relation schema is also called 'intension' and relation state is called 'extension'.
- Integrity is the accuracy or correctness of data in the database schema and is expected to hold on every database instance of that schema. Relational model includes two general.
- Entity Integrity Constraints states that no primary key value can be NULL.
- Referential Integrity Constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations.
- Domain Constraint specifies that each attribute in a relation must contain an atomic value only from the corresponding domains.



4.6 FURTHER READING

1. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems. Pearson.
2. Date, C. J. (2006). An introduction to database systems. Pearson Education India.
3. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.



4.7 ANSWERS TO CHECK YOUR PROGRESS

1.	i) Attribute	ii) Tuple	iii) Database	iv) Degree
	v) Cardinality	vi) Attribute	vii) Relation instance	
	viii) r(R)	ix) Extension	x) attribute	
2.	i) primary key	ii) foreign key	iii) key field	
	iv) attribute	v) entities		
3.	i) True	ii) False	ii) False	
	iv) True	v) True		



4.8 MODEL QUESTIONS

1. Define relation and domain in the context of RDBMS.
2. Explain the properties adopted by a Relational Database Management System?
3. What are the advantages of Relational Model?
4. State and explain the basic components of a Relation.
5. Give a brief description of integrity constrains?
6. What is Domain Constraint?

*****

UNIT 5 : KEYS

UNIT STRUCTURE

5.1 Learning Objectives
5.2 Introduction
5.3 Keys
5.4 Types of Keys
    5.4.1 Super Key
    5.4.2 Candidate Key
    5.4.3 Primary Key
    5.4.4 Alternate Key
    5.4.5 Composite Key
    5.4.6 Foreign Key
5.5 Let Us Sum Up
5.6 Further Reading
5.7 Answers to Check Your Progress
5.8 Model Questions

5.1 LEARNING OBJECTIVES

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

- learn about the concept of keys and its uses in database
- learn the different types of keys like super key, candidate key, alternate key, primary key, foriegn key etc.
- identify primary and foreign key in a relation
- learn to use composite key

5.2 INTRODUCTION

In our previous unit, we have seen that in case of relational model, the database is logically represented in the form of tables so that it can be easily understood and visualized by everyone.The role of keys are very important in case of relational databases. In fact, without keys relational database will not be usuable at all.

In this unit, we will discuss the concept of keys in a database. The use of different types of keys will be covered in this unit.

5.3 KEYS

In a relational model, a database consists of relations (tables), which consists of tuples (or records/rows), and attributes (or fields/columns). We must have a way to distinguise each tuple within a relation. A relation in a relational database must have an attribute or a combination of attributes such that they can uniquely identify the tuple. This unique identifier is called ***key***. A key is a data item that exclusively identifies a record or tuple. It may consists of one or more attributes. We can split related data into different relations or tables and logically link them together with the help of keys. Without this unique identifier, there is no way to retrieve the unique tuple from a relation.

In this unit, we may use the terminologies *table*, *row* or *record* and *field* in place of *relation*, *tuple* and *attribute* respecively. For example, let us consider the following table “STUDENT”.

Table 5.1: **STUDENT**

Roll_no	Name	Marks	Grade
1	Monirupa Misra	360	A
2	Ranjita Dutta	180	C
3	Rajib Sharma	310	A
4	Kaustab Baruah	265	B
5	Apurba Bora	310	A
6	Rajib Sharma	210	B

Table 5.1 gives us marks and grades of students of a particular class. There are six records in the table “STUDENT”. Each record has the following four fields: ***Roll_no***, ***Name***, ***Marks*** and ***Grade***. As we can see, among the fields *Name*, *Marks* and *Grade*, no one field can identify a record in the table uniquely. The *Name* field, cannot be used as key because several

students might have the same name. *Marks* field may contain same marks for more than one student. Similarly, more than one student may have the the same *Grade*. So these three fields cannot be used as key. However, the field ***Roll_no*** can easily identify any row in the table uniquely. Roll numbers of students in a particular class are different. So such fields can be used as key.

5.4 DIFFERENT TYPES OF KEYS

Every key which has the property of uniqueness can be distinguished as follows :

- Super Key
- Candidate Key
- Primary Key
- Alternate Key
- Composite Key
- Foreign Key

5.4.1 Super Key

A superkey is a set of columns that uniquely identifies every row in a table. For example, if there is a table “STUDENT” with only two columns **Roll_no** and ***Name***, then the super key will be if we assume that there are no two students in the class with the same Roll_no as well as Name.

{ Roll_no, Name}

Similarly, let us consider an EMPLOYEE table as in Table 5.2 consisting of the columns ***Emp_ID***, ***Name*** and ***Post***. We could use the ***Emp_ID*** in combination with any or all other columns of this table to uniquely identify a row in the table. Examples of superkeys in this table would be ***{Emp_ID}***, ***{Emp_ID, Name}*** and ***{Emp_ID, Name, Post}***.

Table 5.2: **EMPLOYEE**

Emp_ID	Name	Post
001	Goutam Das	Accountant
011	Prakash Bora	Account Asstt.
023	Himangshu Das	Superitendent
033	Niharika Sarma	Financial Officer

In a real database we donot need values for all of those columns to identify a row. We only need a minimal set of columns that can be used to identify a single row. In our example, the set ***{Emp_ID}*** is the minimal superkey.

5.4.2 Candidate Key

A table can have more than one column that could be chosen as the key because they individually have the capability to identify a record uniquely. These fields are termed ***candidate keys***. In other words, a candidate key is any set of one or more columns whose combined values are unique among all occurrences (i.e., tuples or rows or record). Since a NULL value is not guaranteed to be unique, no component of a candidate key is allowed to be NULL. Candidate keys are those attributes of a relation, which have the properties of *uniqueness* and *irreducibility*. Now, let us explain these two properties:

Let K be a set of attributes of relation R. Then K is a candidate key for R if and only if it possesses both of the following properties:

Uniqueness: No legal value of R ever contains two distinct tuples with the same value for K.
Irreducibility: No proper subset of K has the uniqueness property.

Let us consider the following relation ***EMP_INFO*** containing some personal information of employees working in an office. Suppose all of them have passport.

Table: 5.3: **EMP_INFO**

Emp_ID	Name	Passport_no	Blood Group
12341	Kunal Kashyap	M 9523421	A+
12342	Rajib Sharma	M 9515212	O+
12343	Ankur Chakraborty	M 9523123	O+
12344	Niharika Bora	F 9515456	B+
12345	Antara Dutta	F 9643521	AB+

The attribute ***Emp_ID*** and ***Passport_no*** posseses unique data item for each employee. Therefore, any of these two attribute can be chosen as the key. These two are examples of *candidate keys* in Table 5.3. The attribute ***Name*** cannot be a candidate key as more than one employee might have similar name. Similary, several employees might have the same blood group. So ***Blood Group*** cannot be chosen as key.

5.4.3 Primary Key

Every database table should have one or more columns designated as the *primary key*. The value of primary key must be unique for each record in the database. In a database, there can be multiple candidate keys. Out of all the available candidate keys, a database designer decides a primary key. The primary key should be chosen in such a way that its attribute values are never or very rarely changed.

A ***primary key*** is a field or combination of fields that uniquely identify a record in a table, so that an individual record can be located without confusion. Depending on its design, a table or relation may have arbitrarily many unique keys but must have one primary key. For example, let us assume we have a table called ***EMPLOYEE_ADDRESS*** that contains some information for every employee in an organization. We should need to select an appropriate primary key that would uniquely identify each employee. Our first thought might be to use the employee's name i.e, ***Emp_Name***. But this would not work properly because two or more employees with the same

name might be possible in the organization. The ***Location*** field of a person cannot be chosen as primary key since it is likely to change. A better choice might be to use a unique ***Emp_ID*** number that the organization assign to each employee when they have appointed. ***Emp_ID*** can be a primary key as it does not change till the person is working in the same organization.

Table: 5.4: **EMPLOYEE_ADDRESS**

PK

Emp_ID	Emp_Name	Location
1231	Gautam Baruah	GS Road Guwahati
1232	Arindam Dutta	RG Barua Road Guwahati
1233	Meghali Gogoi	Chandmari Guwahati
1234	Bornalee Sharma	Jalukbari Guwahati
1235	Arindam Dutta	Chandmari Guwahati

In the Table 5.1., student's ***Roll_no*** would be a good choice for a primary key. The student's name would not be a good choice, as there is always the chance that more than there may be one student with same name. Some other examples of primary keys are ***Social Security Numbers*** (associated with a specific person) , ***ISBN_no*** (associated with a specific book).

A primary key is a special case of unique keys. *Unique key constraint* is used to prevent the duplication of key values within the rows of a table and allow NULL values. Primary key allows each row in a table to be uniquely identified and ensures that no duplicate rows exist and no NULL values are entered. Thus ***primary key constraint*** can be defined as a rule that says that the primary key fields cannot be NULL and cannot contain duplicate data.

Once we decide upon a primary key and set it up in the database, the database management system (DBMS) will enforce the uniqueness of the key. If we try to insert a record into a table with a primary key that duplicates an existing record, the insert will fail.

Sometimes, a table may have a primary key. In such cases, duplicate records may occur in the table. Most databases are also capable of generating their own primary keys. *Microsoft Access*, for example, may be configured to use the *AutoNumber* data type to assign an unique ID to each record in the table. This is a bad design practice, because it leaves us with a meaningless value in each record in the table. It is better to use that space by storing some useful data.



NULL :
The NULL value indicates that the value does not exist or is not known.

Properties of Primary Key

To qualify as a primary key for an entity, an attribute must have the following properties:

i) **Stable:**

The value of a primary key must not change or should not become NULL throughout the file of an entity. A stable primary key helps to keep the model stable. For example, if we consider a patient record, the value for the primary key (*Patient number*) must not change with time as would happen with the age field.



Constraint :
A constraint is a rule that defines what data is valid for a given field.

ii) **Minimal:**

The primary key should be composed of the minimum number of fields that ensures the occurrences are unique.

iii) **Definitive:**

A value must exist for every record at creation time because an entity occurrence cannot be substantiated unless the primary key value also exists.

iv) **Accessible:**

Anyone who wants to create, read or delete a record must be able to see the primary key value.

5.4.4 Alternate Key

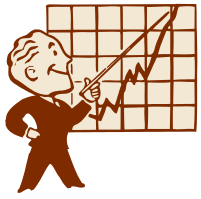
As we have seen, it is possible for a table or relation to have two or more candidate keys. If we choose any one of them as primary key, then the remaining keys will be termerd as alternate key. The alternate

key (or secondary key) is any candidate key which is not selected to be the primary key. For the illustration of alternate key, let us consider the following table ELEMENT which stores some information like element name, symbol, atomic number of the elements of periodic table.

Table: 5.5 : **ELEMENT**

Name	Symbol	Atomic_no
Hydrogen	H	1
Helium	He	2
Lithium	Li	3
Berylium	Be	4
Boron	B	5
Carbon	C	6
Nitrogen	N	7
Oxygen	O	8
Fluorine	F	9
Neon	Ne	10

All the three fields can individually identify each element in the table. So any of these three fields can be chosen as the primary key . If we choose ***Symbol*** as the primary key; ***Name*** and ***Atomic_no*** would then be alternate keys. Similarly, in the ***EMP_INFO*** (Table 5.3), if we consider ***Emp_ID*** as the primary key then ***Passport_no*** will be the alternate key.



CHECK YOUR PROGRESS

1. State whether the following statements are true or false:

   (a) A key is that data item that exclusively identifies a record.

   (b) A table or relation may have arbitrarily many unique keys but at most one primary key.

   (c) The alternate key is any candidate key which is not selected to be the primary key.

   (d) Unique key constraint is used to allow the duplication of key values within the rows of a table and allow NULL values.

   (e) The primary key fields cannot be NULL and cannot contain duplicate data.

5.4.5 Composite Key

In some situations, while designing a database, there may not be a particular column or field that can individually identify a record uniquely in a table. In such cases, we may require to select two or more fields so that combination of those can identify each record uniquely. This combination of fields is known as ***composite key***. It is used when a record cannot be uniquely identified by a single field.

For the illustration of composite key, let us consider Table 5.6. ***ITEM*** with the fields ***Supplier_ID, Item_ID, Item_Name*** and ***Quantity***. This table gives us the information that a supplier supply some items. As we can see, any of these fields indivisually cannot identify a row in the table uniquely. But if we combine ***Supplier_ID*** and ***Item_ID***, then these together can easily identify any row in the table uniquely. Thus, ***Supplier_ID*** and ***Item_ID*** together becomes a composite key.

Table: 5.6 : **ITEM**

Composite Key

Supplier_ID	Item_ID	Item_Name	Quantity
S_1	I_1	AC	5
S_1	I_2	Inverter	8
S_2	I_2	Inverter	4
S_2	I_3	UPS	15
S_2	I_4	Generator	5
S_3	I_3	UPS	10



Entity Integrity:
According to entity integrity, no key column of any row in a table can have a NULL value. In other words, if an attribute of a table is prime attribute (unique identifier), it cannot accept NULL values.

5.4.6 Foreign Keys

In this section we will discuss the concept of ***foreign key***. These keys are used to create relationships between tables.

A foreign key is a field in one relational table that matches the primary key column of another table. It identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that donot exist in the referenced table. This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table.

For example in an employees database, let us imagine that we wanted to add a table ***DEPARTMENT*** containing departmental information to the database. We would also want to include information about the employees in the department, but it would be redundant to have the same information in two tables (***EMPLOYEE*** and ***DEPARTMENT***). Instead, we can create a relationship between the two tables. Let us assume that the ***DEPARTMENT*** table uses the ***Department_Name***



Referential Integrity:
To ensure that a value which appears in one relation or table for a given set of attributes will also appear for certain set of attributes in another relation. This condition is known as referential integrity.

column as the primary key. To create a relationship between the two tables, we add a new column to the ***EMPLOYEE*** table called ***Department_Name***. We then fill in the name of the department to which each employee belongs. The ***Department_Name*** column in the ***EMPLOYEE*** table is a *foreign key (FK)* that references the ***DEPARTMENT*** table. The database will then enforce *referential integrity* by ensuring that all of the values in the ***Department*** column of the ***EMPLOYEES*** table have corresponding entries in the ***DEPARTMENT*** table.

Table: 5.7: **EMPLOYEE**

PK(Primary key) FK(Foreign key)

Emp_ID	**Name**	**Post**	**Department_Name**
A01	Goutam Bora	Accountant	Sales
A02	Manash Saikia	Manager	Marketing
A03	Himangshu Das	Financial Officer	Human Resource
A04	Niharika Baruah	Accountant	Production
A05	Pranjal Hazarika	Law Officer	Human Resource
A06	Meghna Roy	Receptionist	Executive Sales
A07	Diganata Bora	Manager	Production
A08	Smita Roy	Law Officer	Production
A09	Barnalee Sharma	Receptionist	Marketing
A10	Dhrub Sharma	Manager	Human Resource
A11	Kamalesh Das	Manager	Sales

Table: 5.8: **DEPARTMENT**

PK

Department_Name	**Manager**	**Dept_Code**
Human Resource	Dhruba Sharma	D1
Marketing	Manash Saikia	D2
Production	Diganata Bora	D3
Sales	Kamalesh Das	D4

Again, let us consider a book database. The ***BOOKS*** table has a link to the ***PUBLISHERS*** table. The ***Pub_ID*** column is the primary key for the ***PUBLISHERS*** table and ***ISBN_no*** is the primary key for the ***BOOKS*** table. The ***BOOKS*** table also contains a ***Pub_ID*** column which matches the primary key column of the ***PUBLISHERS*** table. This ***Pub_ID*** is the foreign key in the ***BOOKS*** table. The ***Pub_ID*** field in the ***BOOKS*** table indicates which publisher a book belongs to.

Table: 5.9: **PUBLISHERS**

PK

Pub_ID	Pub_Name	City	State

Table:5.10: **BOOKS**

PK FK

ISBN_no	Book_name	Author_name	Pub_ID	Price	Pub_date

Although the primary purpose of a ***foreign key constraint*** is to control the data that can be stored in the foreign key table, it also controls changes to data in the primary key table. For example, if the row for a publisher is deleted from the ***PUBLISHERS*** table, and the publisher's ID is used for books in the ***BOOKS*** table, the relational integrity between the two tables is broken; the deleted publisher's books are orphaned in the ***BOOKS*** table without a link to the data in the ***PUBLISHERS*** table. A foreign key constraint prevents this situation. The constraint enforces referential integrity by ensuring that changes cannot be made to data in the primary key table if those changes invalidate the link to data in the foreign key table. If an attempt is made to delete the row in a primary key table or to change a primary key value, the action will fail if the deleted or changed primary key value corresponds to a value in the foriegn key constraint of another table. To change or delete a row in a foreign key constraint successfully, we must first either delete the foreign key data in the foreign key table or change the foreign key data in the foreign key table, thereby linking the

foreign key to different primary key data. i.e., a primary key constraint cannot be deleted if referenced by a foreign key constraint in another table; the foreign key constraint must be deleted first.



CHECK YOUR PROGRESS

2. State whether the following statements are True or False:

(a) In a relational database, the foreign key of a relation would be the primary key of an another relation.

(b) Foreign Key represents relationship between the tables.

(c) There may be more than one foreign key in a table.



5.5 LET US SUM UP

- A key is that data item that exclusively identifies a record. A database consists of tables, which consists of records, which further consist of fields.
- Database keys can be classified into *super key*, *candidate key, primary key, alternate key, composite key* and *foreign key*.
- A candidate key is any set of one or more columns whose combined values are unique among all occurrences (i.e., tuples or rows).
- A super key for an entity is a set of one or more attributes whose combined value uniquely identifies the entity in the entity set. A super key has the uniqueness property but not necessarily the irreducibility property .
- A primary key is a special case of unique keys. The major difference is that for unique keys the NOT NULL constraint is not automatically enforced, while for primary keys it is. Thus, the values in a unique key columns may or may not be NULL.
- The primary key of any table is any candidate key of that table which

the database designer arbitrarily designates as “primary”. The primary key may be selected for convenience, comprehension, performance, or any other reasons.

- The alternate key is any candidate key which is not selected to be the primary key.
- A composite key is made up of two or more columns.
- Foreign keys are very important in the context of table inter-relationships. It is a field in a relational table that matches the primary key column of another table.



5.6 FURTHER READING

1. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems. Pearson.
2. Date, C. J. (2006). An introduction to database systems. Pearson Education India.
3. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.



5.7 ANSWERS TO CHECK YOUR PROGRESS

1. (a) True (b)True (c) True (d) False (e) True
2. (a) True (b) True (c) False



5.8 MODEL QUESTIONS

1. Explain the following keys with examples:
   (i) Primary Key
   (ii) Candidate Key
   (iii) Super Key

(iv) Alternate Key

(v) Foreign Key.

2. Create a relation student and mark the different types of keys in it.
3. Explain the significance of primary key and foreign key in employee database.
4. Define key?
5. What is the difference between candidate key and super key? Explain.
6. Discuss the properties of primary key?
7. Define candidate key with appropriate example?
8. Define primary key constraint.
9. Define the uniqueness and irreducibility properties of candidate key.
10. Design two database table showing primary key and foreign key.

*****

UNIT 6 : RELATIONAL DATABASE DESIGN

UNIT STRUCTURE

6.1 Learning Objectives
6.2 Introduction
6.3 Design Guidelines
6.4 Anomalies in a Database
6.5 Functional Dependency
    6.5.1 Full Functional Dependency
    6.5.2 Partial Dependency
    6.5.3 Transitive Dependency
    6.5.4 Multi-Valued Dependency
    6.5.5 Join Dependency
6.6 Decomposition
    6.6.1 Decomposition and Dependency Preservation
    6.6.2 Decomposition and Lossless Join
6.7 Prime and Non Prime Attribute
6.8 Universal Relation
6.9 Let Us Sum Up
6.10 Further Reading
6.11 Answer to Check Your Progress
6.12 Model Questions

6.1 LEARNING OBJECTIVES

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

- learn how to design a good database
- learn about functional dependency
- discuss different types of anomalies in a database
- define the decomposition nature of a database

6.2 INTRODUCTION

We are familiar with the concept of database and DBMS from the previous units. We have acquainted with relational model and their constraints and keys.

This unit focuses on relational database design. The main goal of relational database design is to generate a set of schemas that allow us to store data without unnecessary redundancy as well as retrieve information easily and accurately. Here, we will discuss some important concept like anomalies in a database, functional dependency, decomposition etc. which are related with database design process.

6.3 DESIGN GUIDELINES

Designing a database is a very important task. If we design a database correctly, storing, updating and retrieving of data will be efficient. Some important features of a good database design are as follows:

Semantics of the Relation Attributes :

Semantics specifies how the attribute values in a tuple are related to one another. To explain the semantics, it is better to design relation schema. So, we have to design a relation schema in such a way that it is easy to explain its meaning. We should not combine the attributes from multiple entry types and relationship types into a single relation.

Reducing the redundant values in tuples :

One goal of schema design is to minimize the storage space. Grouping of attributes into relation schemas has a significant effect on storage space. We have to design the base relation schemas so that no insertion, deletion and modification anomalies occur in the relations.

Null values in tuples:

A null value in a relation is the wasteful of space at the storage level that may lead problems in understanding the meaning of the attributes. So it is better to avoid null values in base relation. If nulls are unavoidable, it is necessary to make sure that they apply in exceptional cases only and do not apply to a majority of tuples in a relation.

Disallowing spurious tuples:

We should design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

6.4 ANOMALIES IN A DATABASE

The aim of the database system is to reduce redundancy, i.e., unnecessary repetation of data should be eliminated. Updates to the database with such redundancies have the potential of becoming inconsistent. The inconsistencies are explained below.

Update anomalies:

Multiple copies of the same fact may lead to update anomalies or inconsistencies when an update is made and only some of the multiple copies are updated. Consider the STUDENT relation of Figure 6.1.

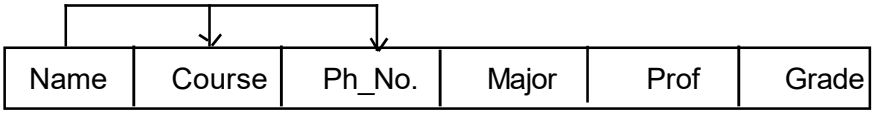


Fig. 6.1: Functional dependencies of STUDENT relation

In the above figure 6.1, it is assumed that one student can be enrolled for more than one courses. Here we have considered that there are three entries with the name “Rahul” with different courses. The phone numbers of three tuples are same because the entries are for one student Rahul. Suppose Rahul has taken a new phone number. Now a change in the phone number (Ph_No.) of Rahul must be made in all tuples pertaining to the student Rahul for consistency. If one of these three tuples is not changed to reflect the new phone no. of Rahul, there will be an inconsistency in the data.

Insertion anomalies:

If above is the only relation in the database showing the association between

a faculty member and the course he or she teaches, the fact that a given professor is teaching a given course cannot be entered in the database unless a student is registered in the course. Also, if another relation also establishes a relationship between a course and a professor who teaches that course the information stored in these relations has to be consistent.

Deletion Anomalies:

If the only student registered in a given course discontinues the course, the information as to which professor is offering the course will be lost if this is the only relation in the database showing the association between a faculty member and the course she or he teaches. If another relation in the database also establishes the relationship between a course and a professor who teaches that course, the deletion of the last tuple in STUDENT for a given course will not cause the information about the course's teacher to be lost.



CHECK YOUR PROGRESS

1. Fill in the blanks.

i) Redundancy leads to _____.

ii) Avoiding redundancy is important because it _____.

iii) Faithfulness is one _____ of a good database design.

iv) Multiple copies of the same fact may lead to _____ anomalies.

v) A null value in a relation is the wasteful of space at the _____.

vi) It is always better to avoid null values in _____.

6.5 FUNCTIONAL DEPENDENCY

A ***functional dependency*** is a constraint between two sets of attributes from a database. A functional dependency denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of relation R. It specifies a constraint on the possible tuples that can form a relation instance r of R.

The constraint states that for any two tuples t1 and t2 in r such that t1[X] = t2[Y], it must also have t1[X] = t2[Y]. This means that values of the X component of a tuple uniquely or functionally determine the values of the Y component.

For example, let us consider the following **ITEM** table, shown in Table 6.1. Let us consider a combination of the ***Item_code*** and the ***Item_name*** columns. Item_code is the primary key of the table and therefore always unique. If Item_code given, we can determine Item_name. Given the value of *Item_code*, there is only one value of *Item _name* for it. Thus, *Item_name* is functionally dependent on *Item_code*. Symbolically, this is written as:

Item_code → Item_name

This should be read as ***Item_code*** functionally determines ***Item_name***.

Table 6.1: ITEM table

ITEM

Item_code	Item_name	Quantity
P01	Pen	200
P02	Marker	150
P03	Pencil	300
P04	Eraser	250

Functional dependency may also be based on a composite attribute. For example, if we write

{X,Y} → Z

It means that Z is functionally dependent on composite attribute X and Y.

A functional dependency is a property of the meaning or semantics of the attributes in a relation schema R. If we consider the Figure 6.1, the functional dependencies are:

Name → {Ph.No., Major}
Course → Prof
{Name, Course} → Grade

Here, in the first relation, the attributes ***Ph.No*** and ***Major*** are functionally dependent on the prime attribute ***Name***. Alternatively, we can say that the prime attribute ***Name*** determined the non prime attributes ***Ph.No*** and ***Major***. Similarly, we can now explain the other two relational schemas also. Concept of prime and non prime attribute will be discussed at the end of this unit.

6.5.1 Full Functional Dependency

When all non-key attributes are dependent on the key attribute, it is called full functional dependency. For example, consider the relation schema of an employee,

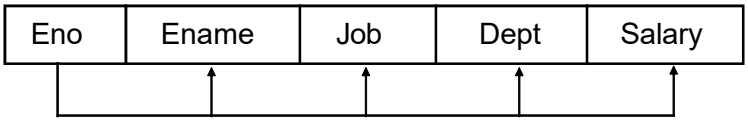


Fig. 6.2: Functional dependency of relation EMPLOYEE

Here all non-key attributes (Ename, Job, Dept, Salary) are dependent on key attribute Eno, which is the Employee No. in the relation EMPLOYEE.

6.5.2 Partial Dependency

A functional dependency X $\rightarrow$ Y is a partial dependency if some attributes A $\rightarrow$ X can be removed from X and the dependency still holds. It means that all non-key attributes are not dependent on the key attribute. There is a partial dependency of non-key attributes either on the key attribute or on the non-key attribute.

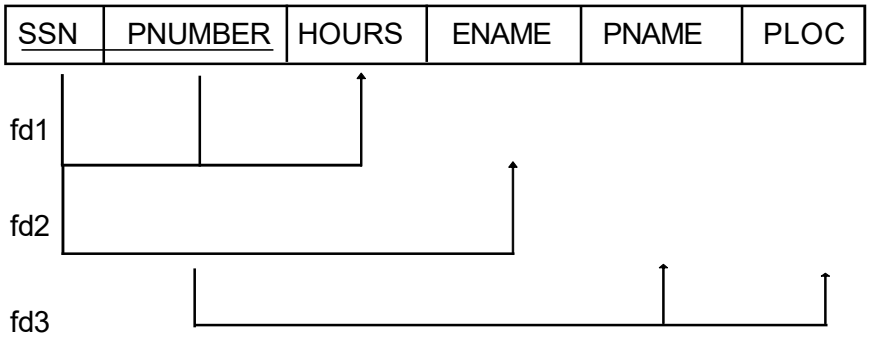


Fig. 6.3: A relation schema of Emp-Project relation

In figure 6.3, the dependency
{SSN, PNUMBER} → ENAME is partial because SSN → ENAME holds.

6.5.3 Transitive Dependency

A functional dependency X → Y in a relation R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R and both X → Z and Z → Y hold. In general sense we can say that if A, B, C are three attributes in a table, and if A is related to B and B is related to C, then A is indirectly related to C.

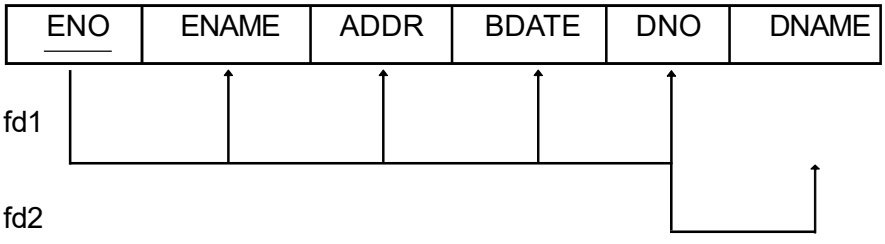


Fig. 6.4: A relation schema of Emp-Dept relation

In the above figure, DNO is dependent on key attribute ENO and DNAME is dependent on DNO which we can denote as,

ENO → {ENAME, ADDR, BDATE, DNO}
DNO → DNAME

From the above dependencies we can say that DNAME is indirectly related to key attribute ENO. So, DNAME is transitively dependent on ENO.

6.5.4 Multi-Valued Dependency

Let R be a relational variable and A, B and C are subsets of the attributes of R. Then we say that B is multi-dependent on A. Symbolically, it can be denoted as,

$A \rightarrow\rightarrow B$ and read as A multi-determines B

Multi-valued dependencies are a generalization of functional dependencies, in the sense that every functional dependency is an multi-valued dependency. But the converse is not true.

COURSE	TEACHER	TEXT
Data structure	Prof. Baruah	Basic data structure
Data structure	Prof. Sarma	Data structure through C
Digital Logic	Prof. Reddy	Principle of Logic gates
Digital Logic	Prof. Talukdar	Basic logic structure
Digital Logic	Prof. Sarma	Principle of Digital Circuit
Digital Logic	Prof. Baruah	Fundamental of Logic gates

Fig. 6.5 : A relation table of relation COURSE

In the above table there are two multi-valued dependencies that hold:

$COURSE \rightarrow\rightarrow TEACHER$

$COURSE \rightarrow\rightarrow TEXT$

The first multi-valued dependency means that although a course does not have a single corresponding teacher, each course does have a well-defined set of corresponding teaches. Multi-valued dependency does not rule out the existence of certain tuples, which have multiple dependencies. Instead, they require that other tuples of a certain form be present in the relation.

6.5.5 Join Dependency

Join dependency is a constraint, similar to a Functional dependency or a Multi-valued dependency. It is satisfies if and only if the relation concerned is the join of certain number of projections. The definition is if R be a relational variable, and A, B, ..., Z are subsets of attributes

of R, then R satisfies the Join Dependency that is denoted a,

* {A, B, ..., Z} and read as star A, B, ..., Z

6.6 DECOMPOSITION

The decomposition of a relation scheme $R = (A_1, A_2, ..., A_n)$ is its replacement by a set of relation schemas $\{ R_1, R_2, ..., R_m\}$, such that $R_1 \leq R$ for $1 \leq i \leq m$ and $R_1 \cup R_2 \cup R_m = R$. A relation schema R can be decomposed into a collection of relation schemas $\{R_1, R_2, ..., R_m\} = D$ to eliminate some of the anomalies contained in the original relation R. Here the relation schemas $R_1 (1 \leq i \leq m)$ are subsets of R and intersection of $R1 \cap Rj$ for $i \neq j$ need not be empty. Furthermore the union of Rj $(1 \leq i \leq m)$ is equal to R, i.e $R = R_1 R_2 ... R_m$.

6.6.1 Decomposition and Dependency Preservation

We need to preserve the dependencies because each dependency in functional dependency F represents a constraint on the database. If one of the dependencies is not represented by the dependencies on some individual relation R_i of the decomposition, it will not be able to enforce this constraint by looking only at an individual relation. Instead to enforce the constraint, we will have to join two or more of the relations in the decomposition and then check that the functional dependency holds in the result of the join operation. It is useful that each functional dependency $X \rightarrow Y$ specified in F either appear directly in one of the relation schemas Ri in the decomposition D.

We say that a decomposition D = {R1, R2, ..., Rm} of R is dependency preserving with respect to F if the union of the projections of F on each Ri in D is equivalent to Fj, that is

$$((\pi_F (R_1)) \ ... ... \ (\pi_F (R_m)))^+ = F^+$$

If any decomposition is not dependency preserving, some dependency is lost in the decomposition. We can check by joining several relations in the decomposition. Let us consider the relation of STUDENT with their functional dependencies.

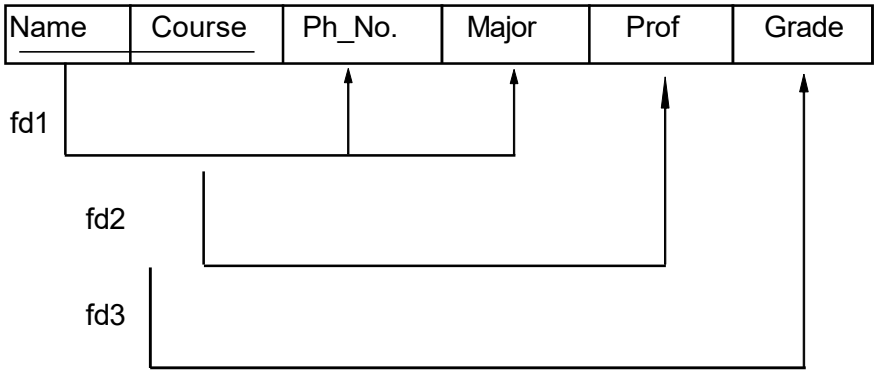


Fig. 6.6 Functional dependency of relation STUDENT

The dependency will resolve if we decompose the above relation with the following relations.

STD_INFO (Name, Ph_no, Major)
STD_RES (Name, Course, Grade)
TEACHER (Course, Prof)

The **first relation** schema gives the phone number and major of each student and such information will be stored only once for each student. Any change in the phone number will thus require a change in only one tuple of this relation. The **second relation** schema stores the grade of each student in each course that the student is or was controlled in. The **third relation** schema records the teacher of each course.

6.6.2 Decomposition and Lossless Join

This property ensures that no spurious tuples are generated when a **natural join** operation is applied to the relations in the decomposition.

It is the property of a decomposition of relation schemas that no spurious tuples should hold on every legal relation instances. The lossless join property always defined with respect to a specific set F of dependencies. The word “lossless” refers to loss of information, not loss of tuples.

Formally, a decomposition D = {R1, R2, ... ,Rm} of R has the lossless join property with respect to the set of dependencies F on R if for every relation instance r of R that satisfies F, the following holds,

$$*(\pi_{<R1>}(r),.....,\pi_{<Rm>}(r)) = r$$

If decomposition does not have lossless join property, then it may get an additional spurious tuples after the projection or natural join operation applied. These additional tuples represent erroneous information and hence add wrong information. It is also called *non-additive join* because it describes the situation more accurately.



CHECK YOUR PROGRESS

2. Fill in the blanks

a) When all non key attributes are dependent on the key attribute, it is called _____ dependency.

b) When one non key attribute depends on other non key attribute, it is called a _____ dependency.

c) A functional dependency is denoted by, _____ between two sets of attributes A, B.

d) Multi-valued dependencies are also called as _____.

e) A multi-valued dependency is denoted by, _____ between two sets of attributes A,B.

f) The _____ of a relation scheme is its replacement by a set of relation schemas.

g) A relation schema can be decomposed into a collection of relation schemas to eliminate some of the anomalies contained in the original relation.

h)The _____ property guarantees the spurious tuple problems.

i) The lossless join property is also known as _____.

j) A _____ attribute is a member of any key of a relational schema.

3. Write whether the following statements are true or false:

a) A functional dependency is a constraint between two sets of attributes from the database.

b) The lossless join property cannot give the guarantee of the spurious tuple problem.

c) Dependency preservation property ensures that all FDs are represented in some of the individual resulting relations.

d) Join dependency is a join of certain number of projection of relation schemas.

e) Multi-valued dependency is also referred as equality generating dependency.

6.7 PRIME AND NON PRIME ATTRIBUTE

An attribute of relation schema R is called a prime attribute of R if it is a member of any key of R. An attribute is called non prime attribute, if it is not a prime attribute, i.e. it is not a member of any candidate key of R.

SSN	PNUMBER	HOURS

Fig. 6.7 : a relation schema of relation WORKS_ON

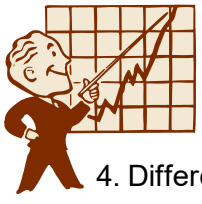
In the figure both SSN and PNUMBER are prime attributes and HOURS is non prime attribute.

6.8 UNIVERSAL RELATION

The concept of the universal relation manifests itself in several different ways. First of all it includes all of the attributes relevant to the database

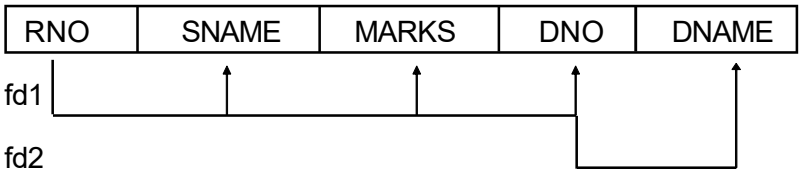
under consideration and then showed how those relational variables can be replaced by successively smaller projections until some good structure is reached.

The second, and more pragmatically significant, manifestation of the universal relation is the ***user interface***. The basic idea here is quite straightforward, and indeed quite appealing: Users should be able to frame their database requests, not in terms of relational variables and joins among those relational variables, but rather in terms of attributes alone.



CHECK YOUR PROGRESS

4. Differentiate prime and non prime attributes for the following STUDENT relation.



6.9 LET US SUM UP

- The design and implementation of a database relation should be faithful to the requirements. Simplicity requires that the design and implementation avoid introducing more elements than are absolutely necessary – Keep it Simple.
- Attributes are easier to implement, but entity sets and relationships are necessary to ensure that the right kind of element is introduced.
- The aim of the database system is to reduce redundancy, meaning that information is to be stored only once. Storing information several times leads to the waste of storage space and an increase in the total size of the data stored. Updates to the database with such redundancies have the potential of becoming inconsistent.
- Functional Dependencies are the consequence of the interrelation-

ships among attributes of an entity represented by a relation. It is denoted by X $\rightarrow$ Y, between two sets of attributes X and Y.

- When all non-key attributes are dependent on key attribute, it is called full functional dependency.
- In partial dependency, all non-key attributes are not dependent on the key attribute.
- Functional Dependencies rule out certain tuples from being in a relation, but multi-valued dependency do not rule out the existence of certain tuples, which have multiple dependencies. Functional dependencies are ***equality generating dependencies*** and Multi-valued dependencies are ***tuple generating dependencies***.
- The reduction process of a relation consists of replacing the given relational variable by certain projections, in such a way that joining those projections back together again gives us back the original relational variable. The process is reversible.
- Decomposing a universal relation schema satisfies the lossless join property, dependency preservation property or both properties. All these properties are based on the functional dependencies specified on the attributes of the universal relation.



6.10 FURTHER READING

1.Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems. Pearson.

2. Date, C. J. (2006). An introduction to database systems. Pearson Education India.

3.Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.

6.11 ANSWER TO CHECK YOUR PROGRESS

1. i) Inconsistency ii) Waste space iii) Feature
   iv) Update v) Storage level vi) Base relation

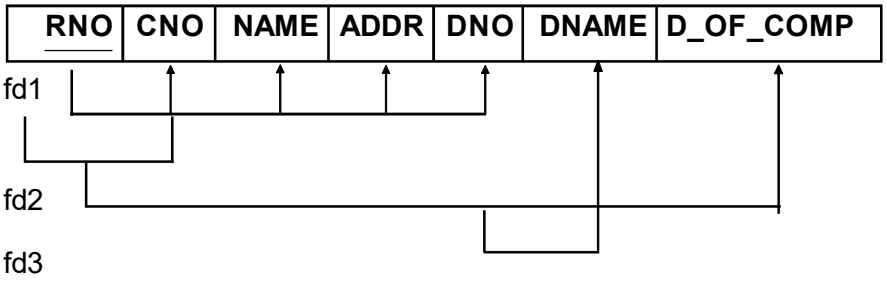
2. (a) full functional dependency (b) transitive dependency
   (c) A → B (d) tuple generating dependency
   (e) A →→ B (f) decomposition
   (g) anomalies (h) lossless join
   (i) non adaptive join (j) prime

3. (a) True (b) False (c) True (d) True (e) False

4. Prime: RNO, DNO
   Non prime: SNAME, MARKS, DNAME

6.12 MODEL QUESTIONS

1. Define Functional Dependency? Why it is required?
2. Explain Partial Dependency with a suitable example.
3. What is Multi-valued and Join Dependency?
4. Consider the following relational schema of STUDENT relation where RNO is the Roll no of student, CNO is the Course Enrolment no for the student and DNO is the Department no. Relation shows their functional dependencies also. Decompose it, so that it preserves the dependencies.



5. Write short notes on:
   a) Transitive dependency
   b) Multi-valued Dependency

c) Decomposition

d) Universal Relation

e) Dependency Preservation

f) Decomposition with Lossless Join

6. Differentiate Prime attribute and Non-prime attribute with example.
7. State the features of a good database design
8. Explain the inconsistency arise at the designing time of databases.

****

UNIT 7 : NORMALIZATION

UNIT STRUCTURE

7.1 Learning Objectives
7.2 Introduction
7.3 Normalization and Its Objectives
7.4 Normal Forms
    7.4.1 First Normal Form (1NF)
    7.4.2 Second Normal Form (2NF)
    7.4.3 Third Normal Form (3NF)
7.5 Boyce-Codd Normal Form (BCNF)
7.6 Fourth Normal Form (4NF)
7.7 Fifth Normal Form (5NF)
7.8 Let Us Sum Up
7.9 Further Reading
7.10 Answers to Check Your Progress
7.11 Model Questions

7.1 LEARNING OBJECTIVES

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

- understand what normalization is.
- list out the objectives of Normalizations
- describe 1st, 2nd and 3rd Normal forms
- illustrate fourth and fifth Normal forms

7.2 INTRODUCTION

In the previous unit, we have learnt two important concepts in database design i.e., Functional Dependency and Decomposition which usually help in minimizing the redundancy in database design. In this unit we will discuss Normalization which is one of the most important concept in database.

Normalization is the process of efficiently organizing data in a database. Normalization is the name given to the process of simplifying the relationship among data elements in a record. In this unit, we will discuss

different normal forms. The learners will be able to normalize database tables into different normal forms after completing this unit.

7.3 NORMALIZATION AND ITS OBJECTIVES

Normalization is a process of decomposing a set of relations(table) with anomalies to produce smaller and well structured relations that contain minimum or no redundancy. The basic objectives of normalization are to reduce redundancy, which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Normalization provides the designer with a systematic and scientific process of grouping of attributes in a relation.

The normalization process can be defined as a procedure of analyzing and successive reduction of the given relational schemas based on their FDs and primary keys to achieve the desirable properties of -

(i) minimizing redundancy,

(ii) minimizing insertion, deletion and update anomalies.

The process of normalization was first proposed by ***E.F. Codd***. Normalization is a bottom up design technique for relational database.

The objectives of the normalization process are :

- To create a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- To free relations from undesirable insertion, update and deletion anomalies.
- To reduce the need for restructuring the relations as new data types are introduced.
- To carry out series of tests on individual relation schema so that the relational database can be normalized to some degree. When a test

fails, the relation violating that test must be decomposed into relations that individually meet the normalization tests.

7.4 NORMAL FORMS

Whenever the simple rules of Functional Dependencies are applied to a relation, it transforms the relation into a state which called *normal form.* The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. Various types of normal forms are used in relational database. They are :

- First normal form (1 NF)
- Second normal form (2 NF)
- Third normal form (3 NF)
- Boyce/Codd normal form (BCNF)
- Fourth normal form (4 NF)
- Fifth normal form (5NF)

7.4.1 First Normal Form (1NF)

A relation is said to be in First normal form if the values in the domain of each attribute of the relation are atomic. The first normal form prohibits multivalued attributes, composite attributes and their combinations. It means that, 1NF disallows having a set of values, a tuple of values, or combination of both as an attribute value for a single tuple.

Let us consider the relation TRAVEL_INFO as shown in Table 7.1.

Table 7.1: Un-normalized relation TRAVEL_INFO

PERSON	**VISITED_CITY**	
HEMANGA	CITY	JOURNEY_DATE
	DELHI	04-07-02
	MUMBAI	15-09-03
	AGRA	02-03-04
BIKASH	CITY	JOURNEY_DATE
	CHENNAI	05-01-02
	KOLKATA	13-02-03
	PUNE	21-08-04

Here, in the relation the domain VISITED_CITY is not very simple. and in un-normalized form. Now, let us combine the respective rows in VISITED_CITY with the value of the attribue PERSON and the resultant relation will be as follows:

Table 7.2 Modified relation TRAVEL_INFO

PERSON	**CITY**	**JOURNEY_DATE**
HEMANGA	DELHI	04-07-02
HEMANGA	MUMBAI	15-09-03
HEMANGA	AGRA	02-03-04
BIKASH	CHENNAI	05-01-02
BIKASH	KOLKATA	13-02-03
BIKASH	PUNE	21-08-04

Let us consider another relation PATIENT_DOCTOR, which keeps the records of appointment details of patient to doctors. This relation is in first normal form (1NF). The relational table can be depicted as:

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, PHONE, DURATION)

Table 7.3 Relation PATIENT_DOCTOR

P__NAME	DOB	D__NAME	DATE__TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	06-02-05 16:00	15

From the relational table we have observed that a doctor cannot have two simultaneous appointments so D_NAME and DATE_TIME is a compolsite key. Similarly, a patient cannot have same time from two different doctors. Therefore, P_NAME and DATE_TIME attributes are also a candidate key.

Problems in 1 NF :

1NF contains redundant information. In our example, PATIENT_DOCTOR relation has the following errors :

a) There exists redundant information in patients date of birth and phone number.

b) A doctor, who does not currently have an appointment with a patient, cannot be represented.

c) A patient, who does not currently have an appointment with a doctor, cannot be represented.

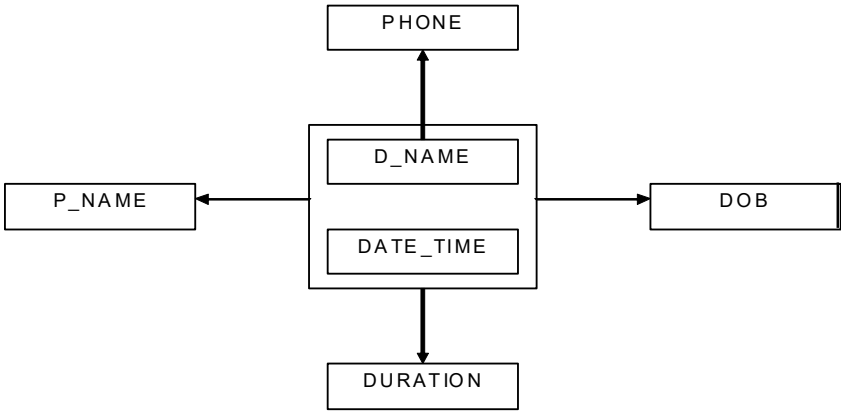


Fig. 7.1: FDD of relation PATIENT_DOCTOR

Functional dependancy diagram(FDD) of the relation PATIENT_DOCTOR is shown. Here, P_NAME, DURATION, DOB are dependent on the key D_NAME & DATE_TIME and doctor's contact number i.e., PHONE is only dependent on the D_NAME.

7.4.2 Second Normal Form (2NF)

A relation or table is said to be in Second normal form (2NF) if and only if -

a) it is in 1NF,

b) each non-primary key attribute is fully functionally dependent on primary key of that relation.

2NF is an intermediate step towards higher normal forms. 2NF is based on the concept of *full functional dependency*. It eliminates the problems of 1NF.

So, we come to know that no attributes of the relation (or table) should be functionally dependent on only one part of a concatenated primary key. In our example, we have seen from the functional dependancy diagram that PHONE is partiallly dependent only on D_NAME for which the relation is not in 2NF.

Therefore, to bring the relation into 2NF, the information about doctor and their contact numbers have to be separated from the information about patient and their appointments with doctors. Thus, the relation is decomposed into two tables, namely PATIENT_DOCTOR and DOCTOR as shown below. The relational table can be depicted as :

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, DURATION)

DOCTOR (D_NAME, PHONE)

Table 7.4: Relation PATIENT_DOCTOR

P__NAME	DOB	D__NAME	PHONE	DATE__TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	0361-223470	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	0361-270512	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	0361-223470	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	0361-270615	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	0361-270512	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	0361-223470	06-02-05 16:00	15

Table 7.5 Relation DOCTOR

D__NAME	PHONE
Dr Deepak	0361-223470
Dr Kalyan	0361-270512
Dr Dhiraj	0361-270615
Dr Dhiraj	0361-270615
Dr Kalyan	0361-223470
Dr Dhiraj	0361-270615
Dr Kalyan	0361-270512
Dr Deepak	0361-223470

The functional dependency diagram of the above two relations are

shown below :

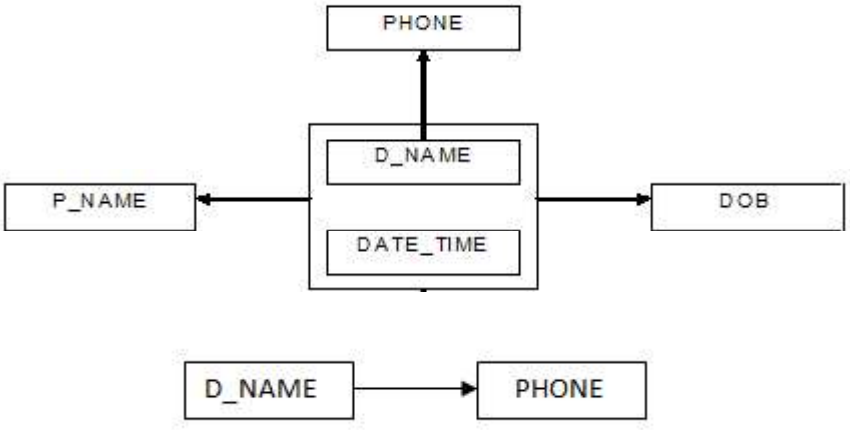


Fig. 7.2: FFD of relation PATIENT_DOCTOR

Problems in 2 NF :

a) In the relation (or table) PATIENT_DOCTOR, deleting a record from it may lose the patient's details.

b) Any changes made on the PATIENT_DOCTOR table may involve in changing multiple records because the information in the table is still redundant.

7.4.3 Third Normal Form (3NF)

A relation or table is said to be in Third normal form (3NF) if the relation is in 2NF and the non-prime attributes are -

a) mutually independent,

b) functionally dependent on the primary key.

It means that no attributes of the relation should be transitively functionally dependent on the primary key. Thus, in 3NF, no non-prime attribue is functionally dependent on another non-prime attribute. This means that *a relation in 3NF consists of the primary key and a set of independent nonprime attributes*. 3NF is based on the problem of transitive dependency. The 3NF eliminates the problem of 2NF.

In our example, in the relation PATIENT_DOCTOR (Table 7.4) , there is no dependency between the attributes P_NAME and DURATION. Again, P_NAME and DOB are mutually dependent. So, the relation is not in 3NF.

To bring the relation PATIENT_DOCTOR into 3NF, it has to be decomposed and has to remove the attributes that are not directly dependent on the primary key. Now, using the *transitive dependency,* DOB can be linked to the primary key, through its dependency on the P_NAME. The functional dependency diagram is as follows. Now, the new relations will be:

PATIENT_DOCTOR(P_NAME,D_NAME, DATE_TIME,DURATION)

DOCTOR (D_NAME, PHONE)

PATIENT (P_NAME, DOB)

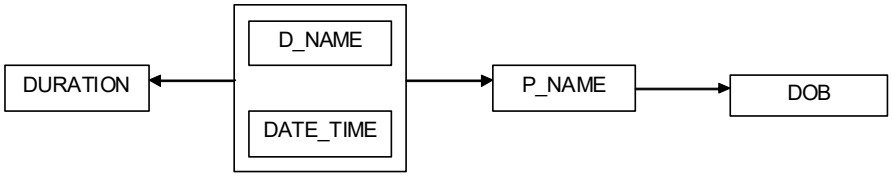


Fig. 7.3: FDD for relation PATIENT_DOCTOR

Now, let us consider another relation STUDENT as shown in Table 7.6

Table 7.6: STUDENT relation

STUDENT (Roll_No, Name, Dept, Year, Hostel_Name)

Roll_No	Name	Dept	Year	Hostel_Name
1784	Ruhit	Physics	1	Ganga
1648	Keshab	Chemistry	1	Ganga
1768	Gopal	Maths	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Mrinal	Geology	3	Krishna
1485	Sumit	Zology	4	Godavari

Here, the dependency **Roll_No** $\rightarrow$ **Hostal_Name** is transitive through the following two dependencies :

Roll_No $\rightarrow$ **Year**

Year $\rightarrow$ **Hostal_Name**

Thus, the STUDENT relation is not in 3NF. To bring the relation into 3NF we can decompose the relation into two relation STUD1 and STUD2, as shown below:

STUD1 (Roll_No, Name, Dept)

STUD2 (Year, Hostel_Name)

Table 7.7 Relation STUD1

Roll_No	Name	Dept	Year
1784	Ruhit	Physics	1
1648	Keshab	Chemistry	1
1768	Gopal	Maths	2
1848	Raja		2
16	rinal		
1485	Sumit		4

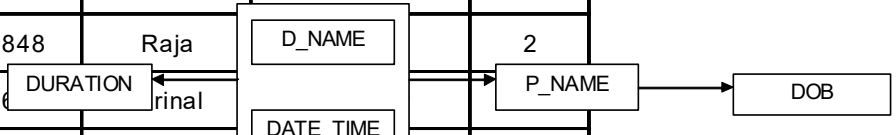


Table 7.8 Relation STUD2

Year	Hostel_Name
1	Ganga
1	Ganga
2	Kaveri
2	Kaveri
3	Krishna
4	Godavari

In the above examples, the conversion into 3NF is not hard, but whenever a relation has more than one combination of attributes that

may be considered as primary key then the conversion takes a number of steps. Let us consider the following relation UTILIZE as shown in Table 7.9.

Table 7.9: Relation UTILIZE

Project	Proj_Manager	Machine	Qty_Used
P1	Thomas	Excavator	5
P3	John	Shovel	2
P2	Abhishek	Drilling	5
P4	Avinash	Dumper	10
P3	John	Welding	3
P1	Thomas	Drilling	4

The relation UTILIZE stores the machines information used by both projects and project managers. Each project has one project manager and each project manager manages one project. Now, it is obvious from the table that we can consider any one of the combination of attributes as primary key, namely, {Project, Machine} or {Proj_Manager, Machine}. The FDD for relation UTILIZE is shown Figure 7.4.

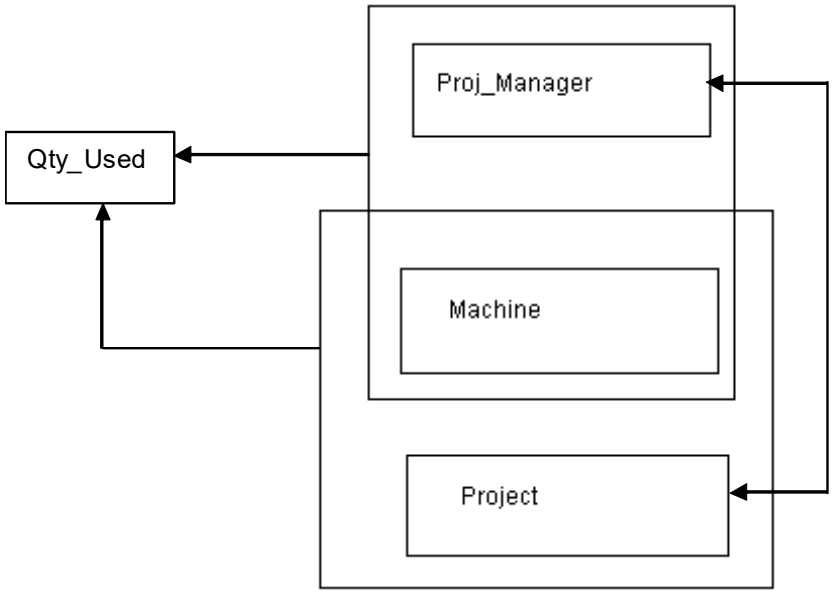


Fig.7.4: FDD of UTILIZE

In the relation UTILIZE, there is only one non-prime attribute called, **QTY_Used**, which is fully functionally dependent on each of the two relations. Thus, the relation UTILIZE is in 2NF. Moreover, there is only one non-prime attribute Qty_Used. There can be no dependencies between the non-prime attributes. Thus the relation is also in 3NF.

Problems in 3NF :

If we consider the above relation i.e., UTILIZE, which is in 3NF, it has the following undesirable properties :

a) The project manager of each project is stored more than once.

b) A project manager cannot be stored until the project has ordered some machines.

c) A project cannot be entered unless that project's manager is known.

d) If a project's manager changes, some rows must also be changed.

7.5 BOYCE-CODD NORMAL FORM (BCNF)

The redundancy and the problems of 3NF can be eliminated by the use of the **Boyce-Codd Normal Form** (**BCNF**) which was proposed by **R.F. Boyce**.

We know that if Y is funcionally dependent on X, that is, X funcionally determines Y, then the following relation holds good:

$$X \rightarrow Y$$

Here, we call X the determinant. From this, we can define BCNF as follows: A table is in Boyce-Codd normal form (BCNF) if the only determinants in the table are the candidate keys.

A relation (or table) R is said to be in BCNF if for every nontrivial FD : $X \rightarrow Y$ between attributes X and Y holds in R. It means :

- X is super key of R,
- $X \rightarrow Y$ is a trivial FD, that is $Y \subset X$.

From the above conditions we have come to know that a relation must only

have candidate keys as determinants. Any relation in BCNF is also in 3NF and consequently in 2NF. However, a relation in 3NF is not necessarily in BCNF.

The difference between 3NF and BCNF is that if the functional dependency $A \rightarrow B$ satisfy that B is a *primary key* attribute and A is not a candidate key, then 3NF will allow this dependency in a relation. Otherwise, if the functional dependency $A \rightarrow B$ satisfy that A must be a candidate key, then this dependency will belong to BCNF.

7.6 FOURTH NORMAL FORM (4NF)

A table is in the Fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.

The fourth normal form is related to the concept of a ***multi-valued dependancy*** (MVD). In simple terms, if there are two columns - A and B - and if for a given A, there can be multiple values of B, then we say that an MVD exists between A and B.

The fourth normal form is theoretical in nature. In practice, normalization up to the third normal form are generally adequate. In certain situations, the designers may also have to look at the BCNF.

Let us consider the following STUDENT table (or relation) :

Table 7.10 STUDENT relation

Sudent	Subject	Language
Geeta	Mythology	English
Geeta	Psycology	English
Geeta	Mythology	Hindi
Geeta	Psycology	Hindi
Sikhar	Gardening	English

We can see that there are two independent MVD facts in this relationship :

a) A student can study many subjects (i.e., Student $\rightarrow \rightarrow$ *Subject)*

b) A student can learn many languages (i.e., Student $\rightarrow \rightarrow$ *Language)*

The primary key for the STUDENT table is currently a composite key made up of all the three columns in the table - *Student*, *Subject* and *Language*. ,i.e., here the primary key is a combination of the three fields *Student, Subject* and *Language.*

The process of bringing STUDENT table into 4NF is by spliting the independent multi-valued components of the primary key into two tables. Now, let us split these two independent multi-valued dependencies into two separate tables namely *Student_Subject* and *Student_Language*. The resulting tables are shown below :

Table 7.11: Relation Student_Subject

Student	Subject
Geeta	Mythology
Geeta	Psycology
Sikhar	Gardening

Table 7.16 Relation Student_Language

Student	Language
Geeta	English
Geeta	Hindi
Sikhar	English

We have seen that, this decomposition reduces redundancy with respect to both the independent multivalued dependency relationships, that is *Subject* and *Language*.

7.7 FIFTH NORMAL FORM (5NF)

A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.

There are some relations which cannot be decomposed into two or higher normal form relations by means of projection methods discussed in 1NF, 2NF, 3NF and BCNF. Such relations are decomposed into three or more relations, which can be reconstructed by means of a three-way or more join operation. This is called fifth normal form (5NF). The 5NF eliminates the problems of 4NF. 5NF allows for relations with join dependencies. Any relation that is in 5NF, is also in other normal forms namely 2NF, 3NF and 4NF. 5NF is mainly used from theoretical point of view and not for practical database design.



CHECK YOUR PROGRESS

1. Select the correct answer from the following :

a) Normalization is a process of
   (i) decomposing of a set of relation.
   (ii) successive reduction of relation schema.
   (iii) deciding which attributes in a relation are to be grouped together.
   (iv) all of these.

b) A normal form is
   (i) a state of a relation that results from applying simple rules regarding FD.
   (ii) the highest normal form condition that it meets.
   (iii) an indication of the degree to which it has been normalized.
   (iv) all of these.

c) In 1NF,
   (i) all domains are simple.
   (ii) in a simple domain, all elements are atomic
   (iii) both (i) & (ii)
   (iv) none of these

d) 2NF is always in
   (i) 1NF (ii) BCNF

(iii) MVD (iv) none of these

e) A relation R is said to be in 2NF
   (i) if it is in 1NF.
   (ii) every non-prime key attributes of R is fully functionally dependent on each relation key of R.
   (iii) if it is in BCNF.
   (iv) both (i) & (ii).

f) A relation R is said to be in 3NF if the
   (i) relation R is in 2NF
   (ii) non prime attributes are mutually independent.
   (iii) functionally dependent on the primary key.
   (iv) all of these.

g) 4NF is concerned with dependencies between the elements of compound keys composed of
   (i) one attributes
   (ii) two attributes
   (iii) three or more attributes
   (iv) none of these

2. Fill in the blanks of the following :

a) _____ first developed the process of normalization.

b) A relation is said to be in 1NF if the values in the domain of each attribute of the relation are _____.

c) 2NF can be violated only when a key is a _____ key or one that consists of more than one _____.

d) In 3NF, no non-prime attribute is functionally dependent on _____.

e) Relation R is said to be in BCNF if for every non-trivial FD : _____ between attributes Xand Y holds in R.

f) A relation is in BCNF if and only if every determinant is a _____.

g) Any relation in BCNF is also in _____ and consequently in _____.

h) 4NF is violated when a relation has undesirable _____.



7.8 LET US SUM UP

- Normalization is a design technique that is widely used in designing relational model . There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table).
- Functional dependencies play an important role in differencing good database design from not so good database design.
- A relation is said to be in First Normal Form (1NF) if only every entry of the relation (the insertion of a tuple and a column) has at most a single value. In other words, “a relation is in First Normal Form if and only if all underlying domains contain atomic values or single value only”.
- A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.
- A relation or table is said to be in third normal form (3NF) if the relation is in 2NF and the non-prime attributes are mutually independent and functionally dependent on the primary key.
- A relation R is in 4NF if and only if the following conditions are satisfied simultaneously: (i) R is already in 3NF or BCNF. (ii) If it contains no multi-valued dependencies.
- A relation (or table) R is said to be in BCNF if for every nontrivial FD : X $\rightarrow$ Y between attributes X and Y holds in R.

- A table is in the fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.

- A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.



7.9 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



7.10 ANSWERS TO CHECK YOUR PROGRESS

1. a. (iv), b. (iv), c. (ii), d. (i), e. (iv), f. (iv), g. (iii)
2. a) E.F.Codd b) atomic c) composite, attribute
   d) another non prime attributes e) $X \rightarrow Y$
   f) candidate key g) 3NF, 2NF h) multi-valued dependencies.



7.11 MODEL QUESTIONS

1. What is meant by normalization? Why do we normalize the database? Describe the data normalization process.
2. What are the different normal forms ?
3. Define 1NF, 2NF and
5. How do we convert a database in 1NF to 2NF?
6. How we convert a database in 2NF to 3NF?
7. Explain 3NFwith example.
8. Define BCNF. How does it differ from 3NF ? Why is BCNF stronger than 3NF? Illustrate with an example.

9. Consider a relation-
   Student(Student_Id, Student_Name, Student_Address, Date_of_Birth, Course_Id, Course_Name, Duration, No_of_Papers)
   The functional dependencies are-
   Student_Id-> Student_Name
   Student_Id->Student_Address
   Student_Id->Course_Id
   Student_Id->Date _of_Admission
   Student_Id->Date_of_Birth
   Course_Id->Course_Name
   Course_Id->Duration
   Course_Id->No_of_Papers
   Decompose the above relation into equivalent Third Normal Form (3NF) relations.

10. Following relation for published books is given :
    BOOK(BOO-TITLE, AUTH-NAME, BOOK-TYPE, LIST-PRICE, AUTH-AFFL, PUBLISHER)
    here AUTH-AFFL refers to the affiliation of author. Suppose that the following FDs exists
    BOOK-TITLE $\rightarrow$ PUBLISHER, BOOK-TYPE
    BOOK-TYPE $\rightarrow$ LIST-PRICE
    AUTH-NAME $\rightarrow$ AUTH-AFFL
    a) What normal form is the relation in ? Explain your answer.
    b) Apply normalization until the relations cannot be decomposed ant further. State the reason behind each decomposition.

11. Explain the following terms with an example each :
    (a) Normalization (b) 1NF (c) BCNF (d) 3NF
    (e) 4NF (f) 5NF (g) MVD
    (h) Candidate key (i) Primary Key (j) Super Key

*****

UNIT 8: INTRODUCTION TO SQL

UNIT STRUCTURE

8.1 Learning Objectives

8.2 Introduction

8.3 Structured Query Language (SQL)

- 8.3.1 Characteristics of SQL
- 8.3.2 Advantages of SQL
- 8.3.3 Disadvantages of SQL

8.4 SQL Data Types

- 8.4.1 Numeric Data Type
- 8.4.2 Character Data Type
- 8.4.3 Bit String Data Type
- 8.4.4 Boolean Data Type
- 8.4.5 Date and Time Data Type

8.5 Types of SQL Commands

- 8.5.1 Data Definition Language (DDL)
- 8.5.2 Data Manipulation Language (DML)
- 8.5.3Transaction Control Statements (TCS)

8.6 Let Us Sum Up

8.7 Further Reading

8.8 Answers to Check Your Progress

8.9 Model Questions

8.1 LEARNING OBJECTIVES

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

- define SQL
- describe the different data types used in SQL
- describe DDL and DML commands

8.2 INTRODUCTION

In the previous unit you have been introduced to the concepts of relational databases and different forms of normalization. The first, second and third stages of normalization have been covered in the previous unit. In addition to these three stages, BCNF has also been covered in the earlier unit.

In the present unit you will be introduced to the concepts of Structured Query Language (SQL). The different data types in SQL are discussed in detail in this unit. Data Definition Language (DDL) and Data Manipulation Language (DML) SQL commands are also introduced in this unit along with a brief introduction to the Create and Delete SQL tables.

Besides, you will also be made familiar with the basics of SQL tables here while in the next unit we will discuss Insertion and Updating of data into the SQL tables.

8.3 STRUCTURED QUERY LANGUAGE (SQL)

Structured Query Language (SQL) is the standard language for relational databases. SQL provides a common standard and defines the methods to create and maintain databases across different platforms. It was initially developed as "SEQUEL (Structured English Query Language)" at IBM and was designed to manipulate and retrieve the data stored in the database management system at IBM. SQL became a standard later with the joint effort of American National Standards Institute (ANSI) and International Organization for Standardization (ISO).

SQL is based on relational algebra and tuple relational calculus. It contains both data definition and data manipulation commands needed to create and provide maintenance of the databases. In SQL the relational database terms relation, tuple and attribute are described as table, row and column. These terms may be interchangeably used. A relation of a relational database is implemented as a table in SQL; the tuple is implemented as a row and the attributes are implemented as columns in SQL. You need to be familiar with these terms, since they will be used extensively in this unit and in the next few units.

Table 8.1: Terms in relational algebra and SQL

Relational Algebra	**SQL**
Relation	Table
Tuple	Row
Attribute	Column

Why do we need SQL?

We may have come across the many different databases that are in use today, namely - Microsoft SQL Server, Oracle, MySQL etc. among many. The basic commands used in all of these databases uses the same syntax of structured query language. This helps in maintaining uniformity among the databases. Let us consider the scenario when all these database companies did not have a common standard and used different query languages. In this case, we would have to make different database management software based on the different query language adopted by the database platform. It would also be very difficult for the users to remember the different set of commands needed to perform database operations in various database platforms. SQL, being a common query language standard, saves us all this trouble. The database software now only needs to be based on SQL.

8.3.1 Characteristics of SQL

Some of the characteristics of SQL are listed below.

- SQL is the standard database query language for creating and maintaining database
- SQL is based on relational algebra
- Database system softwares based on SQL are portable across different databases platforms
- SQL commands are independent of the way the data is stored internally
- SQL is simple and easy to learn
- SQL allows the users to create, update, delete and retrieve data objects from the database

8.3.2 Advantages of SQL

Some of the advantages of SQL are listed below.

- SQL provides a well-defined standard for database query languages
- SQL is easy to learn and understand
- It is portable across various database platforms
- Different views can be given to different users
- SQL queries can be used to retrieve large amount of data efficiently
- Complex queries can be efficiently written using SQL
- SQL is used by all major database vendors like IBM, Oracle and Microsoft
- Many open source databases like MySQL use SQL

8.3.3 Disadvantages of SQL

Some of the disadvantages of SQL are listed below.

- Interfacing an SQL database is complex
- Although SQL databases conform to ANSI & ISO standards, some databases may go for proprietary extensions



CHECK YOUR PROGRESS

Q1. Mention two advantages of SQL.

Q2. Give two characteristics of SQL.

Q3. Mention two disadvantages of SQL.

8.4 SQL DATA TYPES

Data types describe the type of data an attribute can have. We are familiar with the different data types in different programming languages like C, C++ etc. Similarly, there are different data types in SQL also. The basic data types in SQL fall into *numeric*, *character string*, *bit string*, *Boolean*, *date and time* category. Let us discuss each of these categories of data types in detail now.

8.4.1 Numeric Data Type

Numeric data type includes integer numbers of different sizes and also floating point numbers of different precision. Formatted numbers can also be declared in SQL. Some of the numeric data types are listed below.

- **Integer (INT) Data Type:** Integer data type is used to describe integer type data in SQL. There is a separate SMALLINT data type to contain integers of small size. The range of values for Integer (INT) data type is from - 2,147,483,648 to + 2,147,483,647. Examples of integers are 45, 34,567, and -672345 among many.
- **Small Integer (SMALLINT) Data Type:** Small integer data type is used to describe integer values of small sizes. The range of values for SMALLINT data type is -32,768 to + 32,767. Examples of small integers are 45, 31,567, and 28973 among many.
- **Formatted Data Types:** Formatted numbers can be used for both numeric (NUMERIC(i,j))and decimal values (DECIMAL(i,j)). For both the formatted data types DECIMAL(i,j) and NUMERIC(i,j), the i stands for precision, that is the number of total digits in the number and j stands for scale, that is the number of digits before the decimal point. For example, DECIMAL(7,3) is a number that has 4 digits before the decimal and 3 digits after the decimal point.

8.4.2 Character-string Data Type

Character-string data types are used to represent characters of fixed and variable lengths. Some of the character-string data types are listed below.

- **Character (CHAR(n)) data type:** Character (CHAR) data type is used to represent character data in SQL whose size is fixed, meaning that we know the exact number of characters to be put in that column. The length of the string must be specified when we declare a character type data. For example, CHAR (6) means that a character of length 6 is expected. The default value for the length (n) is 1. Examples of characters are 'database', 'DATABASE', 'Query Language', and '45', among many.

  If we assign a character value that contains less number of characters than what was defined in the character column, than the rest of the column is filled with spaces. For example, if we define a character of size 10, and then put a value of 'SQL' in that column, then the first three positions contains 'SQL' but the rest of the column length will be filled with spaces.

- **Variable Character (VARCHAR(n)) data type:** Variable character (VARCHAR) data type is used to represent character data in SQL whose length may vary from 1 up to the maximum length that is specified in the declaration. The length of the string must be specified when we declare a variable character type data and the length starts from 1. For example, VARCHAR (6) means that a character of length six or less than six is expected. Examples of variable characters are 'data', 'SQL', 'Query Language', and '45', among many.
  A CHAR field is a fixed length, and VARCHAR is a variable length field. This means that the storage requirements are different - a CHAR always takes the same amount of space regardless of what you store, whereas the storage requirements for a VARCHAR vary depending on the specific string stored. The character (CHAR) data

type fields have a fixed length and take the same amount of space in computer memory irrespective of the string we store in that field. Whereas for variable character (VARCHAR) data type, the amount of space in computer memory depends on the length of the string we store in that field. For example, let us make two declarations for a field name once using character and once using variable character. In both the declarations, let the maximum size be declared as 30 to store a maximum of 30 characters.

name CHAR(30)

name VARCHAR (30)

Now, if we try to store the string "INDIA" in the field declared using CHAR, even though the string requires space for only 5 characters still it will be stored using space for 30 characters. But in case of VARCHAR, the string "INDIA" will be stored in memory using space required for only 5 characters.

8.4.3 Bit-string Data Type

Bit-string data type is used to represent strings of values 0's and 1's. Some of the bit string data types are listed below:

- **Bit Strings (BIT (n))**: Bit string data type is used to represent strings containing bit values in SQL whose size is fixed meaning that we know the exact number of bits to be put in that column. The length of the string must be specified when we declare a bit string type data. For example, BIT (4) means that a bit string of length 4 is expected. The default value for the length (n) is 1. Examples of bit strings are '0100', '111000' and '1010100' among many.

- **Varying Bit Strings (BIT VARYING(n))**: Varying bit string data type is used to represent bit strings in SQL whose length may vary from 1 up to the maximum length that is specified in the declaration. The length of the string must be specified when we declare a variable bit string type data and the length starts from 1. For example, BIT

VARYING (4) means that a bit string of length four or less than four is expected. Some examples for variable bit strings among many are '010', '1111' and '010101'.

8.4.4 Boolean Data Type

Boolean data type stores two values "TRUE" and "FALSE". The Boolean data type does not require the use of any parameters. The two values are case insensitive. Some examples among many for Boolean data type values are 'TRUE' and 'FALSE'.

If we want to compare any field with Boolean data type then we will have to use the two keywords "True" and "False". If we compare any Boolean data type field with any other value then we will get an error.

8.4.5 Date and Time Data Type

Date and time data type is used to represent and store date and time information in SQL. Some of the date and time data types are listed below.

- **Date (DATE):** The date (DATE) data type stores the date information like years, months and days in SQL. This data type does not require any parameters in declaration. The date values are stored in the "YYYY-MM-DD" format. The range for years (YYYY) is from 0001 to 9999, for months (MM) is from 1 to 12 and for days (DD) from 1 to 31. Although for some databases like Informatica, the years (YYYY) range is from 1582 to 9999. Some examples among many for date data type values are '1990-05-12', '2017-12-25' and '8999-01-01'.

- **TIME (TIME):** The time (TIME) data type stores the time information like hours, minutes and seconds in SQL. This data type does

not require any parameters in declaration. The time values are stored in the "HH:MM:SS" format. The range for hours (HH) is from 0 to 23, for minutes (MM) is from 0 to 59 and for seconds (SS) is from 0 to 59. Some examples among many for time data type values are '05:35:47', '23:32:25' and '17:01:59'.

- **TIMESTAMP (TIMESTAMP):** The timestamp (TIMESTAMP) data type stores timestamp values that include both the date and time information like years, months, days, hours, minutes and seconds in SQL. This data type like date and time data types does not require any parameters in declaration. The timestamp values are stored in the "YYYY-MM-DD HH:MM:SS" format. The date and time portions of the timestamp data type is separated by a space. The ranges for timestamp are the same as applicable to date and time data type. The range for years (YYYY) is from 0001 to 9999, for months (MM) is from 1 to 12, for days (DD) from 1 to 31, for hours (HH) is from 0 to 23, for minutes (MM) is from 0 to 59 and for seconds (SS) is from 0 to 59. Some examples among many for timestamp data type values are '1990-05-12 05:35:47', '2017-12-25 23:32:25' and '8999-01-01 17:01:59'.

Thus the common and frequently used data types have been described in this section. There are many more data types in SQL and all of these data types have not been described here. We will discuss these other data types as and when we will encounter them in the two new units.

Note: Different data types might have different names in different databases. It is always wise to read the documentation of the database you are using. It may happen that even though the names of the data types are same, the precision values may be different for the same data types in different databases.

Table 8.2: Data Type Summary

Data Type	Keyword	Range	Valid Examples	Invalid Examples
Integer	INT	– 2,147,483,648 to + 2,147,483,647	267, 9876, -128934	‘Assam’, TRUE, ’10:20:25’
Small integer	SMALLINT	-32,768 to + 32,767	345, -19876, 30345	-128934, ‘Assam’, TRUE
Decimal	DECIMAL(i.j)	- 10^38 + 1 to 10^38 - 1	1345.67, -690.25	False, ‘India’
Numeric	NUMERIC(i,j)	- 10^38 + 1 to 10^38 - 1	-1345.67, 690.25	true ’16:45:37’
Character	CHAR(n)	Length of n is from 1 to 8000	‘Assam’, ‘GUWAHATI’	674, ’16:45:37’
Variable length character	VARCHAR(n)	Length of n is from 1 to 8000	‘KKHSOU’, ‘database’	-90223, FALSE
Bit string	BIT(n)	0 to 1	‘0101’, ’10001’	‘Assam’, 100
Variable Bit string	BIT VARYING (n)	0 to 1	‘11110’, ‘1010’	‘Assam’, 4050
Boolean	BOOLEAN	True and False	false, FALSE, True	1001, 1234, ‘Assam’

Date	DATE	0001 to 9999 for years, 1 to 12 for months, 1 to 31 for days	'1870-08-23', '2015-11-11', '7824-12-30'	True, 365, 'India', '2017-13-32'
Time	TIME	0 to 23 for hours, 0 to 59 for minutes, 0 to 59 for seconds	'07:13:57', '21:32:45', '13:01:05'	False, 1234, 'Assam', '13:11:65'
Timestamp	TIMESTAMP	0001 to 9999 for years, 1 to 12 for months, 1 to 31 for days, 0 to 23 for hours, 0 to 59 for minutes, 0 to 59 for seconds	'1870-08-23 07:13:57', '2015-11-11 21:32:45', '7824-12-30 13:01:05'	False, 12, 'India', '2017-13-32 13:11:65'

8.5 Types of SQL Commands

We now have some basic ideas about the various data types available in SQL. Now, let us look at the commands that we use in SQL. SQL uses commands to implement the relations, fields and tuple of relational algebra into tables, rows and columns fields in database. These commands help in building and maintaining the databases. The most commonly used SQL commands broadly falls into two language categories: *Data Definition Language (DDL)* and *Data Manipulation Language (DML)*. Besides these two categories, some SQL commands fall under Transaction Control Statements (TCS) which manage the changes made by the earlier two categories of commands.

8.5.1 Data Definition Language (DDL)

Data Definition Language (DDL) commands are used to create and restructure database objects like databases, tables, views and index. Some of the commands in data definition language are: CREATE, ALTER, DROP, TRUNCATE, and RENAME.

The functionalities of these commands are described in Table 8.2.

Table 8.2: Data Definition Language Commands

Command	Function
CREATE	Used to create database, tables, views and index
ALTER	Used to alter the structure of an already existing database table
DROP	Used to delete database, tables, views and index
TRUNCATE	Used to delete all details from an existing database table but not the table itself
RENAME	Used to rename a database table

8.5.2 Data Manipulation Language (DML)

Data Manipulation Language (DML) commands are used to manipulate data in database objects. Some of the commands in data manipulation language are: INSERT, UPDATE, and DELETE. The functionalities of these commands are described in table 8.3.

Table 8.3: Data Manipulation Language Commands

Command	Function
INSERT	Used to insert data in a table
UPDATE	Used to update the old data with new data in a table
DELETE	Used to delete data in a table
SELECT	Used to retrieve selected portions of data from a table according to preferences

8.5.3 Transaction Control Statements (TCS)

Transaction Control Statements (TCS) commands are used to manage the changes that are made in database after the use of data definition language and data manipulation language commands. Some of the commands in Transaction Control Statements (TCS) are COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION. The functionalities of these commands are described in table 8.4.

Table 8.4: Transaction Control Statements (TCS) Commands

Command	**Function**
COMMIT	Used to save the changes in the database permanently
ROLLBACK	Used to restore the database to the point of the last COMMIT
SAVEPOINT	Used to mark a save point to which we can rollback later if required
SET TRANSACTION	Used to configure and start a transaction

Most of these commands are discussed in detail in the next few units.



CHECK YOUR PROGRESS

Q4. How is CHAR data type different from VARCHAR data type?

Q5. Give the range for small integer data type.

Q6. How can we save our transactions permanently in the database?

Q7. Give the difference between TRUNCATE and DROP command.



8.6 LET US SUM UP

- Structured Query Language (SQL) is the standard language for relational databases. SQL provides a common standard and defines methods to create and maintain databases across different platforms.
- SQL is based on relational algebra and tuple relational calculus. It contains both data definition and data manipulation commands needed to create and provide maintenance of the databases.
- The basic data types in SQL fall into numeric, character string, bit string, Boolean, date and time category.
- Integer data type is used to describe integer type data in SQL. Small integer data type is used to describe integer values of small sizes.
- Formatted numbers can be used for both numeric (NUMERIC(i,j))and decimal values (DECIMAL(i,j)). For both the formatted data types DECIMAL(i,j) and NUMERIC(i,j), the i stands for precision, that is the number of total digits in the number and j stands for scale, that is the number of digits before the decimal point.
- Character (CHAR) data type is used to represent character data in SQL whose size is fixed meaning that we know the exact number of characters to be put in that column. Variable character (VARCHAR) data type is used to represent character data in SQL whose length may vary from 1 up to the maximum length that is specified in the declaration.
- Bit-string data type is used to represent strings of values 0's and 1's.
- Boolean data type stores two values "TRUE" and "FALSE". The Boolean data type does not require the use of any parameters. The two values are case insensitive.
- The date (DATE) data type stores the date information like years, months and days in SQL. This data type does not require any parameters in declaration. The date values are stored in the

"YYYY-MM-DD" format.

- The time (TIME) data type stores the time information like hours, minutes and seconds in SQL. The time values are stored in the "HH:MM:SS" format.
- The timestamp (TIMESTAMP) data type stores timestamp values that include both the date and time information like years, months, days, hours, minutes and seconds in SQL. The timestamp values are stored in the "YYYY-MM-DD HH:MM:SS" format.
- Most commonly used SQL commands broadly falls into two language categories: Data Definition Language (DDL) and Data Manipulation Language (DML).
- Data Definition Language (DDL) commands are used to create and restructure database objects like databases, tables, views and index. Some of the commands in data definition language are: CREATE, ALTER, DROP, TRUNCATE, and RENAME.
- Data Manipulation Language (DML) commands are used to manipulate data in database objects. Some of the commands in data manipulation language are: INSERT, UPDATE, and DELETE.
- Transaction Control Statements (TCS) commands are used to manage the changes that are made in database after use of data definition language and data manipuZ



8.7 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



8.8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: Two advantages of SQL are:

i. SQL provides a well-defined standard for database query languages

ii. Easy to learn and understand

Answer to Q2: Two characteristics of SQL are:

i. SQL is the standard database query language for creating and maintaining database

ii. Database system software's based on SQL are portable across different databases platforms

Answer to Q3: Two disadvantages of SQL are:

i. Interfacing an SQL database is complex

ii. Although SQL databases conform to ANSI & ISO standards, some databases may go for proprietary extensions

Answer to Q4: A CHAR data type field is fixed length whereas for VARCHAR data type it is variable length field. This means that the storage requirements are different - a CHAR always takes the same amount of space regardless of what we store, whereas the storage requirements for a VARCHAR varies depending on the specific string stored.

Answer to Q5: The range for small integer data type is from -32,768 to +32,767

Answer to Q6: We can save our transactions permanently in the database by using the COMMIT command.

Answer to Q7: DELETE command is used to delete the whole database table whereas TRUNCATE command is used to only empty the contents of the table.



8.9 MODEL QUESTIONS

Q1. Describe the different data types in SQL.

Q2. Explain the need for SQL.

Q3. Describe the different DML commands.

Q4. Describe the different DML commands.

Q5. Name the different commands used in transaction control state ments.

Q6. Differentiate among DATE, TIME and TIMESTAMP commands using examples.

Q7. Describe the different types of Bit-String data type.

Q8. What are formatted data types? Describe them in brief.

Q9. Describe in brief with examples where we can use Boolean data type.

Q10. Why do we need SQL?

*****

UNIT 9: WORKING WITH SQL PART I

UNIT STRUCTURE

9.1 Learning Objectives

9.2 Introduction

9.3 Basic SQL Commands

- 9.3.1 Creating and Deleting Database
- 9.3.2 Creating Table
- 9.3.3 Altering Tables
- 9.3.4 Deleting Tables
- 9.3.5 Inserting Data
- 9.3.6 Updating Data
- 9.3.7 Deleting Data

9.4 Let Us Sum Up

9.5 Further Reading

9.6 Answers to Check Your Progress

9.7 Model Questions

9.1 LEARNING OBJECTIVES

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

- Create a new database, delete an existing database
- create, alter and delete tables in a database.
- insert data into tables
- update data in tables
- delete data from a table

9.2 INTRODUCTION

In the previous unit you have been introduced to the data types in SQL. The Data Definition Language and Data Manipulation Language commands have

been covered in the previous unit.

In the present unit you will be introduced to the basic SQL commands. In this unit, we will learn how to create and delete a database. We will also discuss how to create, alter and delete a table in SQL. This unit also covers how we can insert, update and delete data from a table.

In this unit you will be made familiar with the basic SQL commands and then in the next unit we will discuss additional SQL commands.

9.3 BASIC SQL COMMANDS

We have been introduced to SQL in the previous unit. We already have an idea of data definition and data manipulation language commands from the previous unit. Most of these commands will be covered in detail in this unit and in the next unit. In this section we will go through each of the basic SQL commands in detail.

9.3.1 Creating and Deleting a Database

Creating a database

Creating a database is generally the first step towards designing a database system. We can create a database using the create command. The syntax for creating a database is listed below.

```
CREATE DATABASE database_name;
```

In the above syntax, create database followed by the identifier name is the command that tells the database system that we want to create a database with that identifier name. Every command is followed by a semicolon to indicate the end of the command. Now, let us try to create a KKHSOU university database. We will write the following command to create this database.

```
CREATE DATABASE KKHSOU;
```

This command will create a database named "KKHSOU". The following statement is another example where we will create another database for keeping company details in a database named "Company".

```
CREATE DATABASE company;
```

Listing all databases present in the database system

If we want to see all the databases present in our database system then we can use the show command. This command can also be used to verify if we have created a database successfully.

```
SHOW DATABASES;
```

This command will list all the databases present in our database system. The output of the show database command will be the names of the two databases that we have created earlier.

```
KKHSOU
COMPANY
```

Using a database

Now, if we want to create tables in the "KKHSOU" database then we will have to enter the specified database. To do so, we will use the use command. The syntax for using a particular database is given below.

```
USE database_name;
```

For example, if we want to use the KKHSOU database and create tables there then we will use the following statement.

```
USE KKHSOU;
```

Deleting a database

If we want to delete a particular database then we use the drop command. The syntax for deleting a database is given below.

```
DROP DATABASE database_name;
```

For example, if we want to drop the "company" database that we created earlier then we can write the following statement.

```
DROP DATABASE company;
```

This statement will delete the database named "company" from the database system.

9.3.2 Creating Table

Creating a table means naming a table, defining its column names and assigning data types to the columns. A table is created using the create table command. The syntax of create table statement is given below.

```
CREATE TABLE table_name
(
      column_name1 datatype,
      column_name2 datatype,
      column_name3 datatype,
      ... ...
      column_nameN datatype,
      primary key (one or more columns)
);
```

In the above syntax, create table is the command that tells the database system that we want to create a table. This is followed by the unique identifier name for the table. Then the list of columns along with their data type and sizes are mentioned within brackets followed by a semicolon. Let us look at an example to understand the creation of tables.

```
CREATE TABLE students
(
      rollno          int             not null,
      name            varchar(30)     not null,
      course          varchar(10)     not null,
      date_of_birth   date            not null,
      primary key     (rollno)
);
```

In this example, we have created a table named **"students"** which will keep student information details for the university. The table has four columns where the first column represents roll number, so it is defined as of type integer data type. The second and third fields represent the name and course taken by the student and so are of variable character type. The fourth field represents the date of birth of the student and so is declared as date data type. Each student can be represented uniquely by the roll number of the student, so the roll number is defined as the primary key in this student table. The "NOT NULL" is defined for all the columns which means that the database system will not accept NULL values for any field. The data type and the size of the fields for the table must be carefully defined. Declaring a field with large size which in reality is not needed leads to wastage of memory space. Let us create another table "course", which keep the information of various courses offered by the university in the same database. In this "course" table, we have defined three columns - course_id, course_name and semesters.

```
CREATE TABLE course
(
      course_id       varchar(5),
      course_name     varchar(30),
      semesters       int
);
```

Viewing structure of a database

We can view the structure of the table we have created using the desc command. This command can also be used to verify if the database has been successfully created according to our specifications. This command gives us the field information of any existing table in six columns. The information provided includes the field name, the data type of the field, whether the field is defined as NULL, whether the field is primary key, its default values and extra information. The syntax for using this command is given below.

```
DESC table_name;
```

This statement will show us the structure of the database table that is mentioned. For example, if we want to view the structure of the "student" table we created then we will write the following statement.

```
DESC students;
```

This statement will give us an output as below.

```
mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(10) | NO   |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

9.3.3 Altering Tables

Altering a table means modifying the structure of an existing table. The ***alter table*** command is used to alter a table. Altering the table may include renaming a table or a column, adding columns, deleting columns, changing the size of columns and adding primary keys to an already existing table. Different statements are used to make all of these changes in a table. Let us look at the alter table statements in detail.

Renaming a table

```
ALTER TABLE table_name

RENAME TO new_table_name ;
```

This is the alter table statement for renaming a table. We specify the name of the table to be renamed after the **"alter table"** keywords. This is followed by the keywords **"rename to"** and the new table name.

If we use the show tables command then we will get a list of the existing tables. We can see the names of the two tables we created "students" and "course" in this list. Now, let us change the name of table "course" that we have earlier created to "course_details". To make this change we can use this statement below.

```
ALTER TABLE course

RENAME TO course_details ;
```

The above statement will change the name of the table "course" from "course" to "course_details". We can easily verify whether this change has been successful by using the show tables command. The output of the show tables command will now show a table with name "course_details" instead of "course".

Renaming a column in table

```
ALTER TABLE table_name

CHANGE COLUMN old_column_name new_column_name

     column definition ;
```

This is the alter table statement for renaming a column. We first specify the name of the table in which the column is to be renamed after the "alter table" keywords. This is followed by the keywords "change col-

umn" after which we specify the old column name and the new column name. We also have to give a column definition specifying its data type and whether the column is NULL or NOT NULL. Now, let us change the name of the column "semesters" in the table "course_details" to "sem". To make this change we can use this statement below.

```
ALTER TABLE course_details

CHANGE COLUMN semesters sem

      int NULL ;
```

The above statement will change the name of the column in the table "course" from "semesters" to "sem". We can easily verify whether this change has been successful by using the ***desc*** command.

```
DESC course_details;
```

The output of the desc command will now show a table with a column name "sem" instead of "semesters".

```
mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| semesters   | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE course_details
    -> CHANGE COLUMN semesters sem
    ->  int NULL ;
Query OK, 0 rows affected (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| sem         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

Adding column to a table

```
ALTER TABLE table_name

ADD column_name column definition;
```

This is the alter table statement for adding a column in the table. We first specify the name of the table in which we want to add a new column after the **"alter table"** keywords. This is followed by the keyword **"add"** after which we specify the name of the column that we want to add. We also have to give a column definition specifying its data type, size and whether the column is NULL or NOT NULL. Now, let us add a column "gender" to the "students" table which will keep the gender information of the student. To make this change we can use the statement below.

```
ALTER TABLE students

ADD gender varchar(15) NOT NULL;
```

The above statement will add a new column "gender" in the table "students". We can easily verify whether this change has been successful by using the desc command.

```
DESC students;
```

The output of the desc command will now show that a new field "gender" has been added to table "students".

```
mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(10) | NO   |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Modifying a column in a table

```
ALTER TABLE table_name
MODIFY column_name column definition;
```

This is the alter table statement for modifying a column in the table. We first specify the name of the table in which we want to modify the column after the "alter table" keywords. This is followed by the keyword "modify" after which we specify the name of the column that we want to modify. We also have to give a column definition specifying its new modified data type, size and whether the column is NULL or NOT NULL. Now, let us modify the column "course" in the "students" table. Let it now store course names with 20 characters instead of 10 characters like in the earlier definition. Also, let us add another modification to this column to accept NULL values. To make this changes we can use the statement below.

```
ALTER TABLE students
MODIFY course varchar(20) NULL;
```

The above statement will modify the column "course" in the table "students" to store new up to 20 characters and to accept NULL values in this field. We can easily verify whether this change has been successful by using the desc command.

```
DESC students;
```

The output of the desc command shows that the field "course" has been modified in table "students". The figure below shows the modification of the column "course" in the table "students" after the execution of alter table statement.

```
mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(10) | NO   |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql> ALTER TABLE students
    -> MODIFY course varchar(20) NULL;
Query OK, 0 rows affected (0.25 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(20) | YES  |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Deleting column from a table

```
ALTER TABLE table_name

DROP COLUMN column_name ;
```

This is the alter table statement for deleting a column from a table. Just like in the earlier definitions, here also, we first specify the name of the table from which we want to delete a column after the **"alter table"** keywords. This is followed by the keywords **"drop column"** after which we specify the name of the column that we want to delete. Now, let us delete the column "gender" from the "students" table. To make this change we can use the statement below.

```
ALTER TABLE students

DROP COLUMN gender;
```

The above statement will delete the column "gender" from the table "students". We can easily verify whether this change has been successful by using the ***desc*** command.

```
DESC students;
```

The output of the **desc** command will now show that field "gender" has been deleted from the table "students".

```
mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(20) | YES  |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE students
    -> DROP COLUMN gender ;
Query OK, 0 rows affected (0.28 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(20) | YES  |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.08 sec)
```

Adding a primary key in a table

```
ALTER TABLE table_name

ADD PRIMARY KEY (column_name);
```

This is the alter table statement for adding primary keys to the table. Like in the previous definitions, here also, we first specify the name of the table in which we want to add primary keys after the "alter table" keywords. This is followed by the keywords "add primary key" after which we specify the column name/s that we want to add as primary key within brackets. Now, let us add the column "course_id" as pri-

mary key in the "course_details" table. To make this change we can use this statement below.

```
ALTER TABLE course_details
ADD PRIMARY KEY (course_id);
```

The above statement will add the column "course_id" as primary key in the table "course_details". We can easily verify whether this change has been successful by using the ***desc*** command.

```
DESC course_details;
```

The output of the ***desc*** command will now show that the column "course_id" has been added as primary key to the table "course_details".

```
mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| sem         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> ALTER TABLE course_details
    -> ADD PRIMARY KEY (course_id);
Query OK, 0 rows affected (0.20 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | NO   | PRI |         |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| sem         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

9.3.4 Deleting Tables

Deleting a table means removing the table from the database system. A table is deleted by using the drop table command. The syntax

of ***drop table*** statement is given below.

```
DROP TABLE table_name ;
```

For example, if we want to delete the table "course_details" then we can use the following statement.

```
DROP TABLE course_details ;
```

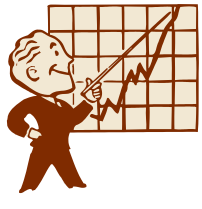
The above statement will delete the table "course_details". We can easily verify whether the table deletion has been successful by using the show tables command.

```
SHOW TABLES;
```

The output of the show tables command will now display the list of existing tables in this database. Since "course_details" table has just been deleted so it would not be displayed in this list.

```
mysql> show tables;
+------------------+
| Tables_in_kkhsou |
+------------------+
| students         |
+------------------+
1 row in set (0.00 sec)
```

Till now we have discussed how to create, delete and alter tables. We have learnt how to add columns, delete columns, add primary columns, rename tables and columns till now. In the next few pages, we will learn how to insert, update and delete data from a table.



CHECK YOUR PROGRESS

Q1. Give the command to view the structure of a table.

Q2. What is the syntax for renaming a database table?

Q3. Write the statement to add a column "age" to the "students" table.

Q4. Give the command to see the list of tables in a database.

9.3.5 Inserting Data

The INSERT INTO command is used to insert data into a table. The syntax for this command is given below.

```
INSERT INTO table_name (column1, column2, ..., columnN)
VALUES (value1, value2, ..., valueN);
```

In the above syntax, we specify the name of the table where we want to insert values after the **"insert into"** keywords. This is followed by the list of column names separated by commas before the keyword **"values"**. A list of values separated by commas and within single quotes is written after the keyword "values" corresponding to the order of the list of columns. Let us insert two tuples into the "students" table that we had created earlier.

```
INSERT INTO students(rollno,name,course,date_of_birth)
VALUES('1','ria','mca','1986-07-25');
```

```
INSERT INTO students(rollno,name,course,date_of_birth)
VALUES('2','bina','mscit','1990-03-11');
```

We can verify that whether the data has been successfully inserted by using the select command. We will learn more about **"select"** command in the later unit. The records inserted into the table "students" will be displayed when we use the following statement.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table.

```
mysql> select * from students;
+--------+------+--------+---------------+
| rollno | name | course | date_of_birth |
+--------+------+--------+---------------+
|      1 | ria  | mca    | 1986-07-20    |
|      2 | bina | mscit  | 1990-03-11    |
+--------+------+--------+---------------+
2 rows in set (0.00 sec)
```

There is another way to insert values in a table without mentioning the column names. The syntax for this is given below.

```
INSERT INTO table_name VALUES (value1, value2, ..., valueN);
```

This is an alternative way to insert values into the table in which we do not have to mention the column names. Only the values are provided in this syntax but in this way we have to make sure that the values are provided in the order the columns are arranged. Let us insert two tuples into the "students" table that is using this method.

```
INSERT INTO students VALUES('3','ram','pgdca','1985-12-31');
```

```
INSERT INTO students VALUES('4','abdul','mca','1972-08-01');
```

We can verify that whether the data has been successfully inserted by using the select command. The records inserted into the table "students" using the new method will be displayed when we use the following statement.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table.

```
mysql> select * from students;
+--------+-------+--------+---------------+
| rollno | name  | course | date_of_birth |
+--------+-------+--------+---------------+
|      1 | ria   | mca    | 1986-07-20    |
|      2 | bina  | mscit  | 1990-03-11    |
|      3 | ram   | pgdca  | 1985-12-31    |
|      4 | abdul | mca    | 1972-08-01    |
+--------+-------+--------+---------------+
4 rows in set (0.00 sec)
```

9.3.6 Updating Data

The UPDATE command is used to modify the existing records in a table. The syntax for this command is given below.

```
UPDATE table_name
SET column1 = value1, column2 = value2,...,columnN = valueN
WHERE condition;
```

In the above syntax, we specify the name of the table where we want to update records after the "update" keyword. This is followed by the list of column names and their new assigned values after the keyword "set". The "where" keyword is used which provides us the condition to select only those records in which we want to make the changes. We will learn about where clause in more details in the next unit. Let us update a record in the "students" table.

```
UPDATE students
SET name = 'gazal'
WHERE rollno = '2';
```

In the above statement, we have modified the name column of the record to "gazal" where the "rollno" is "2". We can verify whether the data has been successfully modified by using the select command.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table.

```
mysql> select * from students;
+--------+-------+--------+---------------+
| rollno | name  | course | date_of_birth |
+--------+-------+--------+---------------+
|      1 | ria   | mca    | 1986-07-20    |
|      2 | gazal | mscit  | 1990-03-11    |
|      3 | ram   | pgdca  | 1985-12-31    |
|      4 | abdul | mca    | 1972-08-01    |
+--------+-------+--------+---------------+
4 rows in set (0.00 sec)
```

9.3.7 Deleting Data from Table

The DELETE command is used to delete any record from the table. The syntax for this command is given below.

In the above syntax, we specify the name of the table from where we want to delete records after the "delete from" keywords. The "where" keyword is used which provides us the condition to select only those records which we want delete. Let us delete a record from the "students" table.

```
DELETE FROM students
WHERE rollno='3';
```

In the above statement, we have chosen to delete the record where the "rollno" is "3". We can verify whether the data has been successfully deleted by using the select command.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table. In the figure below, we can see that the record which had the value of field "rollno" as "3" has been deleted from the table.

```
mysql> select * from students;
+--------+-------+--------+----------------+
| rollno | name  | course | date_of_birth  |
+--------+-------+--------+----------------+
|      1 | ria   | mca    | 1986-07-20     |
|      2 | gazal | mscit  | 1990-03-11     |
|      3 | ram   | pgdca  | 1985-12-31     |
|      4 | abdul | mca    | 1972-08-01     |
+--------+-------+--------+----------------+
4 rows in set (0.00 sec)

mysql> delete from students
    -> where rollno='3';
Query OK, 1 row affected (0.06 sec)

mysql> select * from students;
+--------+-------+--------+----------------+
| rollno | name  | course | date_of_birth  |
+--------+-------+--------+----------------+
|      1 | ria   | mca    | 1986-07-20     |
|      2 | gazal | mscit  | 1990-03-11     |
|      4 | abdul | mca    | 1972-08-01     |
+--------+-------+--------+----------------+
3 rows in set (0.00 sec)
```



CHECK YOUR PROGRESS

Q5. Let us consider a table employee(id, name, department, salary) where field "id" is the primary key. Write the statements in SQL to perform the following functions.

i. Create this table

ii. Insert two records in the table.

iii. Update the first record in the table so that the salary is now "35000".

iv. Delete the records from the table where salary is less than 30000.



9.4 LET US SUM UP

- We can create a database using the create command. The syntax for creating a database is:
  CREATE DATABASE *database_name;*

- If we want to delete a particular database then we use the drop command. The syntax for deleting a database is **DROP DATABASE** *database_name;*

- Creating a table means naming a table, defining its column names and assigning data types to the columns. A table is created using the ***create table*** command.

- Altering a table means modifying the structure of an existing table. The ***alter table*** command is used to alter a table.

- Altering the table may include renaming a table or a column, adding columns, deleting columns, changing the size of columns and adding primary keys to an already existing table

- Deleting a table means removing the table from the database system. A table is deleted by using the ***drop table*** command. The syntax of drop table statement is **DROP TABLE** *table_name;*

- The INSERT INTO command is used to insert data into a table. The syntax for this command is

  INSERT INTO table_name (column1, column2, ..., columnN) **VALUES** (value1, value2, ..., valueN);

- The UPDATE command is used to modify the existing records in a table. The syntax for this command is

  UPDATE *table_name*
  SET *column1 = value1, column2 = value2,...,*columnN = *valueN*
  WHERE *condition;*

- The DELETE command is used to delete any record from the table. The syntax for this command is

  DELETE FROM *table_name*
  WHERE *condition;*



9.5 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



9.6 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: We can view the structure of the table by using the ***desc*** command. The syntax for using this command is:

```
DESC table_name;
```

Answer to Q2: The syntax for renaming a database is:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

We specify the name of the table to be renamed after the "alter table" keyword. This is followed by the keywords "rename to" and the new table name.

Answer to Q3: The following statement can be used to add a column age to the "students" table.

```
ALTER TABLE students ADD age int NOT NULL;
```

Answer to Q4: The command "SHOW TABLES" displays the list of tables present in a database.

Answer to Q5: The statements to perform the functions are:

i.
```
CREATE TABLE employee
(
id      int,
name varchar(30),
dept  varchar(20),
salary  int,
PRIMARY  KEY(id)
);
```

ii.
```
INSERT INTO employee VALUES('01','Pratibha Bora','Accounts','25000');
INSERT INTO employee VALUES('02','Jaynta Medhi','Marketing','20000');
```

iii.
```
UPDATE employee SET salary='35000' WHERE id='01';
```

iv.
```
DELETE FROM employee WHERE salary<'30000';
```



9.7 MODEL QUESTIONS

Q1. Describe the different ways in which a table can be altered.

Q2. Write SQL commands for the following:

i. Create a database "hostel"

ii. Create two tables "boys" and "girls" in hostel database. Include room_number, broader_name, course, semester as attributes in both the tables

iii. Insert three records in both the tables

iv. Update the field "semester" to a value "4" in table "boys" where "course" is "mca"

v. Update the field "course" to a value "mscit" in table "girls" where is "room_number" is "1"

vi. Delete the record in "boys" table where "room_number" is less than 3.

Q3. Give the command to show all the databases in the database system.

Q4. Give the syntax for deleting a database.

Q5. Describe how "create table" is used to for creating tables in a database using example.

Q6. How can a primary key be added after the table has already been created?

*****

UNIT 10: WORKING WITH SQL PART II

UNIT STRUCTURE

10.1 Learning Objectives
10.2 Introduction
10.3 Advanced SQL Commands
    10.3.1 Select
    10.3.2 Where
    10.3.3 Order By
    10.3.4 Group By
    10.3.5 Having
    10.3.6 Logical Operators
10.4 Let Us Sum Up
10.5 Further Reading
10.6 Answers to Check Your Progress
10.7 Model Questions

10.1 LEARNING OBJECTIVES

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

- describe Select statement
- describe where clause
- describe Order By
- describe Group By
- describe Having
- use AND logical operator
- use OR logical operator
- use NOT logical operator

10.2 INTRODUCTION

In the previous unit you have been introduced to the concepts of Structured Query Language (SQL). Data Definition Language (DDL) and Data Manipulation Language (DML) SQL commands were introduced in the previous unit. In addition to this, topics like how to create and delete SQL databases and tables and also how to insert and update data in tables were also covered in the earlier unit.

In the present unit you will be introduced to the many other SQL commands. Clauses like Select, Where, Group By, Having are being discussed in this unit. In the next unit, different types of Join, Queries and Subqueries will be introduced along with a brief introduction to views and indexes.

10.3 ADVANCED SQL COMMANDS

We have been introduced to basic SQL commands in the previous unit. We already have an idea of inserting and updating data in tables from the previous unit. We will now learn to access these data from tables in different ways. In this section we will go through many advanced SQL commands in detail.

10.3.1 Select

The SELECT statement is used to retrieve data from a database table which is displayed in the form of a database table. We can retrieve data from the database using the select command. The basic syntax for select statement is listed below.

```
SELECT column1, column2, ..., columnN from tablename;
```

In the above syntax, we mention the names of those columns from which we want to retrieve information after the select keyword. The names of these columns are followed by the keyword from followed by the name of the database table. Let the table contain the following

rows as displayed in the figure 1 below.

rollno	name	course	date_of_birth
1	ria	mca	1986-07-20
2	gazal	mscit	1990-03-11
4	abdul	mca	1972-08-01

Figure 1: Example showing all records of table "students"

Now, let us try to retrieve only rollno and name information from the "students" table from the database system. We will write the following command to retrieve this information.

```
SELECT rollno, name from tablename;
```

The information that will be displayed by the above statement is shown in figure 2 below.

```
mysql> select rollno,name from students;
+--------+-------+
| rollno | name  |
+--------+-------+
|      1 | ria   |
|      2 | gazal |
|      4 | abdul |
+--------+-------+
3 rows in set (0.00 sec)
```

Figure 2: Example showing rollno and name field information only from table "students"

If we want to retrieve information from all the fields of the database table, then there is another easier way to do it using Select statement. The syntax for retrieving all fields of the database table is given below.

```
SELECT * from tablename;
```

If we want to retrieve information from all the fields of "students" database table, then we use the following statement.

```
SELECT * from students;
```

This statement will display all the records in the students table as displayed in figure 1.

10.3.2 Where

The WHERE clause is used when we want to specify a condition while retrieving data from any database table. It returns as output only those rows which fulfill the specified condition. The syntax of where clause is given below.

```
SELECT column1, column2, ..., columnN

from tablename

WHERE [condition];
```

In the above syntax, we mention the names of those columns from which we want to retrieve information after the select keyword. The names of these columns are followed by the keyword from followed by the name of the database table. Now we specify the condition after the where clause. The condition can be specified by using logical and comparison operators like =, <, >, LIKE, NOT etc. Now, let us try to retrieve only those rollno, name and course field information from the "students" table where the students have opted for MCA course. We will write the following command to retrieve this information.

```
SELECT name, rollno, course

from students

WHERE course='mca';
```

In the above statement, we select to display the name, rollno and course fields of only those students who have chosen their course as MCA as shown in figure 3. It should be noted here that the string should be mentioned inside single quotes.

```
mysql> select rollno, name, course
    -> from students
    -> where course='mca';
+--------+-------+--------+
| rollno | name  | course |
+--------+-------+--------+
|      1 | ria   | mca    |
|      4 | abdul | mca    |
+--------+-------+--------+
2 rows in set (0.03 sec)
```

Fig 3: Example showing use of where clause in table "students"

Now, let us look at another example to show the use of where clause. For this new example, let us consider another table "employee" which has the following records.

```
mysql> select * from employee;
+----+---------------+-----------+--------+
| id | name          | dept      | salary |
+----+---------------+-----------+--------+
|  1 | Pratibha Bora | Accounts  |  35000 |
|  2 | Priya         | Marketing |  40000 |
|  3 | Harry         | Finance   |  25000 |
|  4 | Sunil         | Accounts  |  20000 |
|  5 | Ravi          | Finance   |  35000 |
+----+---------------+-----------+--------+
5 rows in set (0.03 sec)
```

Figure 4: Example showing all records of table "employee"

Suppose, we want to know the name of those employees whose salary is greater than 30,000 from the "employees" table. We can use the Where clause to write this query as given in the following statement.

```
SELECT name
from employee
WHERE salary > 30000;
```

The output of the above statement is displayed in the figure 5.

```
mysql> select name
    -> from employee
    -> where salary > 30000;
+---------------+
| name          |
+---------------+
| Pratibha Bora |
| Priya         |
| Ravi          |
+---------------+
3 rows in set (0.00 sec)
```

Figure 5: Example showing use of where clause in table "employee"

Now, let us look at another example to show the use of Where clause along with Like clause. Suppose, we want to know the names and salary of those employees whose names start from the letter "P" in the "employees" table. We can use the Where clause along with Like clause to write this query.

LIKE Clause

The LIKE clause is used along with WHERE clause to specify a particular pattern. The pattern is expressed with the help of wild cards operators. The pattern may then be used to represent any individual or group of words, numbers or dates. Two commonly used wild car operators are listed below.

" Percent sign (%)

" Underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single character. Both of the wild card operators can be used in combinations. The syntax for using LIKE clause along with WHERE clause id listed below.

```
SELECT column1, column2, ..., columnN
from tablename
where [condition] LIKE [pattern];
```

The above syntax is similar to the where clause syntax except the LIKE part. Here, the desired pattern is specified after the LIKE clause. We can use the where clause along with like clause to write the query to get details of only those employees whose name starts with 'P'. The SQL statement for this query is listed below.

```
SELECT name, salary
from employee
WHERE name LIKE 'P%';
```

In the above statement we select the name and salary fields for display from the table 'employee'. We specify the pattern after the where

and like clause. To specify the pattern that we want all names that start with "P" we denote the pattern using "P%", where P is the initial letter and % means that P can be followed by zero, one or multiple characters . The output of the above statement is displayed in figure 6.

```
mysql> select name, salary
    -> from employee
    -> where name LIKE 'P%';
+---------------+--------+
| name          | salary |
+---------------+--------+
| Pratibha Bora |  35000 |
| Priya         |  40000 |
+---------------+--------+
2 rows in set (0.00 sec)
```

Figure 6: Example showing use of like clause

10.3.3 Order By

The ORDER BY clause is used to sort the output of the query in ascending or descending order according to the columns specified. When we simply use select statement then the output records are not ordered in any specific order. But when we use ORDER BY clause the output is sorted in ascending order by default. The keyword DESC is used to specify if we want the output to be sorted in descending order. The syntax of the ORDER BY clause is listed below.

```
SELECT column1, column2, ..., columnN

from table_name

ORDER BY column1, column2, ..., columnM ASC|DESC;
```

If only one column is specified after the ORDER BY keywords then the resulting table must be displayed in order according to that particular column. However, if more than one columns are specified, then the records are order according to the column whose name appears first, followed by the second and later column names. Let us look at

the example to understand this clause better.

```
SELECT *

from employee

ORDER BY dept, name ASC;
```

In the above SQL statement, we display all the records of the table 'employee' in ascending order first by column name 'dept' and then by column name 'name'. The columns are first ordered according to the first column name that is mentioned and then these are again ordered by the second column name. Hence in this example, the records are ordered in ascending order by 'dept' field values and then if there are same 'dept' values then the records are ordered according to 'name' field. The output of the SQL statement is shown in the figure below.

```
+----+---------------+------------+--------+
| id | name          | dept       | salary |
+----+---------------+------------+--------+
|  1 | Pratibha Bora | Accounts   |  35000 |
|  4 | Sunil         | Accounts   |  20000 |
|  3 | Harry         | Finance    |  25000 |
|  5 | Ravi          | Finance    |  35000 |
|  2 | Priya         | Marketing  |  40000 |
+----+---------------+------------+--------+
5 rows in set (0.03 sec)
```

Figure 7: Example showing use of ORDER BY clause

10.3.4 Group By

The GROUP BY clause is used to group the output records in accordance with one or more columns. Group By clause is used along with the select clause in SQL. The Group by clause is generally used with aggregate functions like COUNT, SUM, Max, MIN etc. The syntax for group by clause is listed below.

```
SELECT column(s)
from table_name
WHERE [condition]
GROUP BY column(s)
ORDER BY column(s);
```

In the above syntax, we choose one or more columns from a table to be displayed in the result set which satisfies the where condition. This is followed by Group By keyword where we specify one or more columns which are to be grouped and then display the result according to some order if the order by clause is used. The Group By clause is generally specified after the where clause and before the order by clause in SQL if the Where and Order By clause are used. Let us look at an example to understand the group by clause.

```
SELECT dept, count(dept)
from employee
GROUP BY dept
ORDER BY count(dept);
```

In this example, we try to count the number of employees in each department. For this, we use the aggregate function COUNT to count the number of times a dept has been repeated in the group. The GROUP BY clause is used to group the records according to their 'dept' values. The ORDER BY clause arranges the output in ascending order of the count of 'dept' values. The output of the above statement is displayed in figure 8.

```
mysql> select dept,count(dept)
    -> from employee
    -> group by dept
    -> order by count(dept);
+-----------+--------------+
| dept      | count(dept)  |
+-----------+--------------+
| Marketing |            1 |
| Finance   |            2 |
| Accounts  |            2 |
+-----------+--------------+
3 rows in set (0.00 sec)
```

Figure 8: Example showing use of GROUP BY clause

10.3.5 Having

The HAVING clause is used to set conditions on the SQL query. It is similar to the Where clause which also sets condition. The HAVING clause is used to set conditions on the groups of rows that is returned by GROUP BY clause. Like the WHERE clause the HAVING clause also returns only those groups of records which satisfy the condition. The syntax of HAVING clause is listed below.

```
SELECT column(s)
from table_name
WHERE [condition]
GROUP BY column(s)
HAVING [condition]
ORDER BY column(s);
```

In the above syntax, we can see that the HAVING clause comes after the GROUP BY clause and before the ORDER BY clause in SQL. This order for clauses needs to be followed. Let us look at an example to understand the concept of HAVING clause.

```
SELECT dept, count(dept)
from employee
GROUP BY dept
HAVING count(dept) > 1
ORDER BY dept;
```

In this example, we want to display the departments where the number of employees is greater than 1. The GROUP BY clause is used to group the records according to their 'dept' values. We then use the aggregate function COUNT to count the number of times a dept has been repeated in each group. Now we use the HAVING clause to set the condition to display only those departments whose count is greater than 1. The HAVING clause will now filter all the groups and pass only

those groups that have count more than 1. The ORDER BY clause then arranges the output in ascending order of the names of 'dept' values. The output of the above statement is displayed in figure 9.

```
mysql> select dept, count(dept)
    -> from employee
    -> group by dept
    -> having count(dept) > 1
    -> order by dept;
+----------+--------------+
| dept     | count(dept)  |
+----------+--------------+
| Accounts |            2 |
| Finance  |            2 |
+----------+--------------+
2 rows in set (0.00 sec)
```

Figure 9: Example showing use of HAVING clause

10.3.6 Logical Operators

Logical operators like AND, OR and NOT can be used in SQL. The logical operators are used along with the WHERE clause. The logical operators AND and OR are used to filter records with more than one condition. The AND operator is used when we want all the specified conditions to be fulfilled. The OR operator is used when we want only any one of specified conditions to be fulfilled. The NOT operator is used to display the records which do not fulfill the conditions. The syntax of AND, OR and NOT logical operators is listed below.

```
SELECT column(s)
from table_name
WHERE [condition1] AND [condition2] ... AND [conditionN];
```

```
SELECT column(s)
from table_name
WHERE [condition1] OR [condition2] ... OR [conditionN];
```

```
SELECT column(s)
from table_name
WHERE NOT [condition];
```

The AND, OR and NOT logical operators can be used in combination with each other. Let us look at some of the examples to understand this concept better.

```
SELECT name, dept
from employee
WHERE dept='Accounts' OR dept='Finance';
```

In the above example, we select to display records with only name and department fields from the table 'employee' where the department values may be 'Accounts' or 'Finance'. The output of this query is displayed in figure 10.

```
mysql> SELECT name, dept
    -> from employee
    -> WHERE dept='Accounts' OR dept='Finance';
+---------------+----------+
| name          | dept     |
+---------------+----------+
| Pratibha Bora | Accounts |
| Harry         | Finance  |
| Sunil         | Accounts |
| Ravi          | Finance  |
+---------------+----------+
4 rows in set (0.00 sec)
```

Figure 10: Example showing use of OR logical operator

Now, let us try to write a query that will display all fields of the table 'employee' whose departments is other than 'Accounts'. This query can be written as in the SQL statement listed below. The output of this query is displayed in figure 11. In figure 11, it can be seen that only records where department values are not "Accounts" are displayed.

```
SELECT name, dept
from employee
WHERE NOT dept ='Accounts';
```

```
mysql> SELECT name, dept
    -> from employee
    -> WHERE NOT dept ='Accounts';
+-------+-----------+
| name  | dept      |
+-------+-----------+
| Priya | Marketing |
| Harry | Finance   |
| Ravi  | Finance   |
+-------+-----------+
3 rows in set (0.00 sec)
```

Figure 11: Example showing use of NOT logical operator



CHECK YOUR PROGRESS

Q1. Let us consider a table **employee(id, name, department, salary) where** field "id" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display all the records with only name and salary fields.
ii. Display the records where salary is greater than 30,000.
iii. Display the name in the table where the name starts from "S".
iv. Display the records in descending order of their departments.
v. Display the records where salary is less than 20000 and department is "Finance".



10.4 LET US SUM UP

- The SELECT statement is used to retrieve data from a database table which is displayed in the form of a database table.
- The WHERE clause is used when we want to specify a condition while retrieving data from any database table. It returns as output only those rows which fulfill the specified condition.

- The LIKE clause is used along with WHERE clause to specify a particular pattern. The pattern is expressed with the help of wild cards operators.
- The ORDER BY clause is used to sort the output of the query in ascending or descending order according to the columns specified.
- The GROUP BY clause is used to group the output records in accordance to one or more columns. The Group by clause is generally used with aggregate functions like COUNT, SUM, Max, MIN etc.
- The HAVING clause is used to set conditions on the SQL query. The HAVING clause is used to set conditions on the groups of rows that is returned by GROUP BY clause.
- The logical operators AND and OR are used to filter records with more than one condition. The NOT operator is used to display the records which do not fulfill the conditions.



10.5 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



10.6 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: The statements to perform the functions are:

i. select name, salary from employee;

ii. select * from employee where salary > 30000;

iii. select name from employee where name LIKE 'S%';

iv. select * from employee order by department desc;

v. select * from employee where salary < 20000 AND department='Finance' ;



10.7 MODEL QUESTIONS

Q1. Consider a table home_loan (loan_no, account_no, name, total_amount, amount_due, interest, start_date, end_date) where field "loan_no" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display the names of customers where the amount of loan is more than 10lakhs.

ii. Display the names and account no of customers who have less than 5lakhs payment due.

iii. Display the number of loans given from 1/1/2011 to 31/12/2016.

iv. Display the account_no of customers whose interest is less than 5%.

v. Perform any function using group by clause.

Q2. Consider a table bank_customer (account_no, name, amount, account_type, city) where field "account_no" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display the names of customers where name starts with "As".

ii. Display the names of customers whose account has more than 50000 but less than 200000.

iii. Display the names and account_no of customers with city as "Guwahati".

iv. Display the account_no of customers whose account_type is not "savings"

v. Perform any function using Having By clause.

vi. Display the number of account holders with city as "Guwahati".

UNIT 11: WORKING WITH SQL PART III

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Join

- 11.3.1 Inner Join
- 11.3.2 Left (Outer) Join
- 11.3.3 Right (Outer) Join
- 11.3.4 Full (Outer) Join

11.4 SQL Sub-queries

11.5 SQL Constraints

- 11.5.1 Not Null Constraint
- 11.5.2 Unique Constraint
- 11.5.3 Primary key Constraint
- 11.5.4 Foreign key Constraint

11.6 Let Us Sum Up

11.7 Further Reading

11.8 Answers to Check Your Progress

11.9 Model Questions

11.1 LEARNING OBJECTIVES

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

- describe different types of join
- describe Queries, Sub-queries constraint considerations

11.2 INTRODUCTION

In the previous unit, we learnt about different statements like Select, Where etc. In this unit, we will learn about how to join two tables and write sub-queries.

11.3 JOIN

In SQL, JOIN is used to combine records from two or more tables provided there is a common field between these tables.

The different types of Join are

- Inner Join
- Left (Outer) Join

- Right (Outer) Join
- Full (Outer) Join

Let us look at each of these different types of join in detail.

11.3.1 Inner Join

INNER JOIN is used to retrieve only records that have matching values in all the tables considered. INNER JOIN is also referred to as EQUIJOIN. There must be a common field between all the tables that are considered of inner join. The INNER JOIN will return only those records which have matching values for the common field. The concept of Inner Join can be understood clearly from figure 1 where the shaded part is the selected region from both the tables.

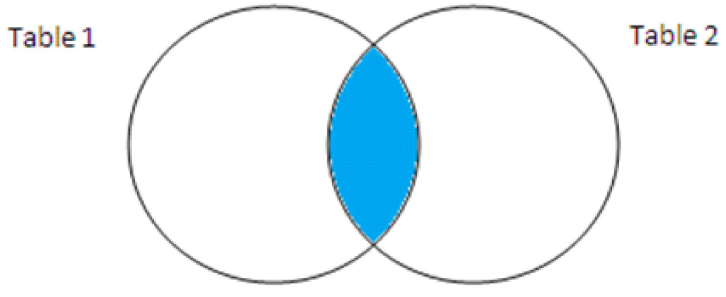


Figure 1: Inner Join

It can be clearly understood from Figure 1 that only the common set values from both the tables are considered in INNER JOIN. The query for INNER JOIN compares each row of the first table with each row of the second table for the common field. The records from both the tables which match the values of the common field are then joined together and displayed as result. The syntax for INNER JOIN is listed below.

```
SELECT table1.column, table2.column

from table1

INNER JOIN table2

ON table1.common_field = table2.common_field;
```

In the above syntax for INNER JOIN, we first select the columns whose values we want to display from both the tables. We then specify the name of the first table after the keyword "'from". The name of the second table is specified after the keywords "inner join". The common field name for both the tables is mentioned after the keyword "on". An "equal to" sign is used between the common field names. Let us look at an example to understand this concept better.

Let us look at two tables "customer" and "odr" with the following structure and record set values. We will use these two tables to explain the different types of join. The structure and record details for table 'customer' are displayed in figure 2. The structure and record details for table 'odr' are displayed in figure 2.

```
mysql> desc customer;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| CustomerName | varchar(30) | YES  |     | NULL    |       |
| CustomerID   | varchar(5)  | YES  |     | NULL    |       |
| City         | varchar(15) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> select * from customer;
+--------------+------------+----------+
| CustomerName | CustomerID | City     |
+--------------+------------+----------+
| bina         | cus01      | ghy      |
| gita         | cus02      | chennai  |
| gita         | cus02      | ghy      |
| mary         | cus03      | shillong |
+--------------+------------+----------+
4 rows in set (0.00 sec)
```

Figure 2: Structure and record values for 'customer' table

```
mysql> desc odr;
+------------+------------+------+-----+---------+-------+
| Field      | Type       | Null | Key | Default | Extra |
+------------+------------+------+-----+---------+-------+
| orderid    | varchar(3) | YES  |     | NULL    |       |
| customerid | varchar(5) | YES  |     | NULL    |       |
| orderdate  | date       | YES  |     | NULL    |       |
+------------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> select * from odr;
+---------+------------+------------+
| orderid | customerid | orderdate  |
+---------+------------+------------+
| 001     | cus01      | 1989-07-02 |
| 002     | cus02      | 1995-05-22 |
+---------+------------+------------+
2 rows in set (0.00 sec)
```

Figure 3: Structure and record vlues for 'odr' table

Let us now perform INNER JOIN on these two tables. Let us try to write a SQL statement where we select to display customer names from the "customer" table and order id from the "odr" table after performing inner join on both the tables using the common field "customerid". The SQL query for INNER JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 15. In figure 4, it is seen that only the records that have matching values for the common field "customer id" are displayed.

```
SELECT customer.customername, odr.orderid

from customer

INNER JOIN odr

ON customer.customerid = odr.customerid;
```

```
mysql> select customer.customername,
    -> odr.orderid
    -> from customer
    -> inner join odr
    -> on customer.customerid=odr.customerid;
+--------------+---------+
| customername | orderid |
+--------------+---------+
| bina         | 001     |
| gita         | 002     |
| gita         | 002     |
+--------------+---------+
3 rows in set (0.03 sec)
```

Figure 4: Example showing use of INNER JOIN

11.3.2 Left (Outer) Join

LEFT (OUTER) JOIN or LEFT JOIN is used to retrieve all the records from the left table (or table 1) and only matching records from the right table (table 2). If there are no matching records from the right table then a value of NULL is displayed. Like INNER JOIN there must be a common field between the tables that are considered for LEFT JOIN. The concept of LEFT JOIN can be understood clearly from figure 5 where the shaded part is the region that is selected when we use LEFT JOIN.

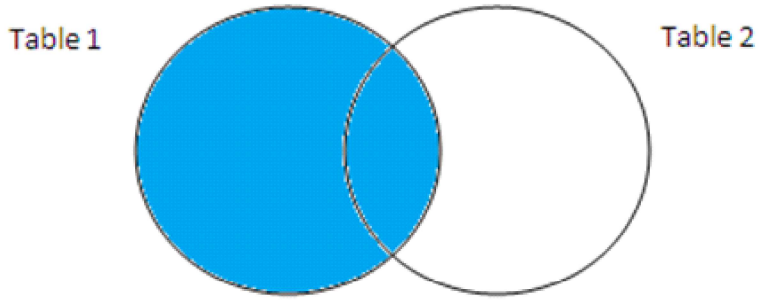


Figure 5: Left (Outer) Join

It can be clearly understood from Figure 16 that all the records from the left table are displayed along with only the matching values from right table in LEFT JOIN. The query for LEFT JOIN compares each row of the first or left table with each row of the second or right table for the common field and displays only those rows of the second table where matching values are found. The records from both the tables are then joined together using the common field and displayed as result. The syntax for LEFT JOIN is listed below.

```
SELECT table1.column, table2.column
from table1
LEFT JOIN table2
ON table1.common_field = table2.common_field;
```

In the above syntax for LEFT JOIN, we first select the columns whose values we want to display from both the tables. We specify the name of the first table after the keyword "'from" and the name of the second table is specified after the keywords "left join". The common field name for both the tables is specified after the keyword "on". An "equal to" sign is used between the common field names. Let us look at an example to understand this concept better.

Let us now perform LEFT JOIN on the tables 'customer' and 'odr' that we have also used in inner join. Let us try to write a SQL statement

where we select to display customer names from the "customer" table and order id from the 'odr' table after performing left join on both the tables using the common field "customerid". The SQL query for LEFT JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 6. In figure 6, it is seen that all the records from table 1 or left table are displayed and only those records of table 2 or right table that have matching values for the common field "customer id" are displayed. For the record with customer id "cus03" in table 'customer' since there is no matching customer id in table 'odr' a NULL value is displayed for "order id" that record.

```
SELECT customer.customerid, customer.customername,
odr.orderid
from customer
LEFT JOIN odr
ON customer.customerid = odr.customerid;
```

```
mysql> select customer.customerid, customer.customername,
    -> odr.orderid
    -> from customer
    -> left join odr
    -> on customer.customerid=odr.customerid;
+------------+--------------+---------+
| customerid | customername | orderid |
+------------+--------------+---------+
| cus01      | bina         | 001     |
| cus02      | gita         | 002     |
| cus02      | gita         | 002     |
| cus03      | mary         | NULL    |
+------------+--------------+---------+
4 rows in set (0.03 sec)
```

Figure 6: Example showing use of LEFT JOIN

11.3.3 Right (Outer) Join

RIGHT (OUTER) JOIN or RIGHT JOIN is used to retrieve all the records from the right table (or table 2) and only matching records from the left table (table 1). If there are no matching records from the left table then

a value of NULL is displayed. Like INNER JOIN and LEFT JOIN there must be a common field between the tables that are considered for RIGHT JOIN. The concept of RIGHT JOIN can be understood clearly from figure 18 where the shaded part is the region that is selected when we use RIGHT JOIN.

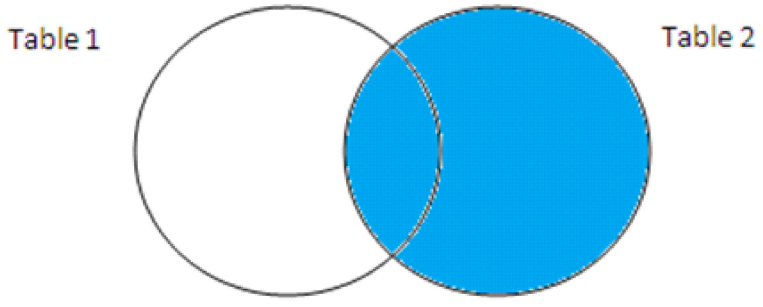


Figure 7: Right (Outer) Join

It can be clearly understood from Figure 18 that all the records from the right table are displayed along with only the matching values from left table in RIGHT JOIN. The query for RIGHT JOIN compares each row of the second or right table with each row of the first or left table for the common field and displays only those rows of the first table where matching values are found for the common field. The records from both the tables are then joined together using the common field and displayed as result. The syntax for RIGHT JOIN is listed below.

```
SELECT table1.column, table2.column
from table1
RIGHT JOIN table2
ON table1.common_field = table2.common_field;
```

In the above syntax for RIGHT JOIN, we first select the columns whose values we want to display from both the tables. We specify the name of the first table after the keyword "'from" and the name of the second table is specified after the keywords "right join". The common field name for both the tables is specified after the keyword "on". An "equal to" sign is used between the common field names. Let us look at an example to understand the concept of right join.

Let us now perform RIGHT JOIN on the tables 'customer' and 'odr' that we have also used earlier. Let us try to write a SQL statement where we select to display customer id and city names from the "customer" table and order id from the 'odr' table after performing right join on both the tables using the common field "customerid". The SQL query for RIGHT JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 8.

In figure 19, it is seen that all the records from table 2 or right table are displayed and only those records of table 1 or left table that have matching values for the common field "customer id" are displayed. So the record with customer id "cus03" is not displayed in the output records since there is no matching value of this customer id in table 'odr'.

```
SELECT customer.customerid, customer.city,
odr.orderid
from customer
RIGHT JOIN odr
ON customer.customerid = odr.customerid;
```

```
mysql> select customer.customerid, customer.city,
    -> odr.orderid
    -> from customer
    -> right join odr
    -> on customer.customerid=odr.customerid;
+------------+---------+---------+
| customerid | city    | orderid |
+------------+---------+---------+
| cus01      | ghy     | 001     |
| cus02      | chennai | 002     |
| cus02      | ghy     | 002     |
+------------+---------+---------+
3 rows in set (0.02 sec)
```

Figure 8: Example showing use of RIGHT JOIN

11.3.4 Full (Outer) Join

FULL (OUTER) JOIN or FULL JOIN is used to retrieve all the records from both the tables when there is a match. If some values are miss-

ing from matched records then NULL values are displayed. Like INNER JOIN, LEFT JOIN and RIGHT JOIN there must be a common field between the tables that are considered for FULL JOIN. The concept of FULL JOIN can be understood clearly from figure 9 where the shaded part is the region that is selected when we use FULL JOIN.

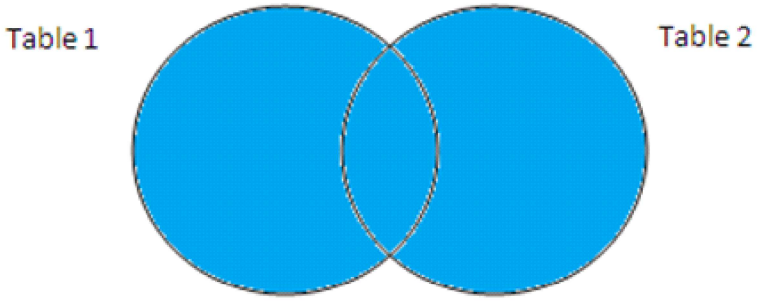


Figure 9: Full (Outer) Join

It can be clearly understood from Figure 20 that all the matched records from both the right and left table are displayed in FULL JOIN. The records from both the tables are then joined together using the common field and displayed as result. The syntax for FULL JOIN is listed below.

```
SELECT table1.column, table2.column
from table1
FULL JOIN table2
ON table1.common_field = table2.common_field;
```

In the above syntax for FULL JOIN, we first select the columns whose values we want to display from both the tables. We specify the name of the first table after the keyword "'from" and the name of the second table is specified after the keywords "full join". The common field name for both the tables is specified after the keyword "on". An "equal to" sign is used between the common field names. Full join is not supported in some database software's like MySQL. So, instead full join is realized by combining left join and right join using UNION. The syntax to achieve FULL JOIN by combining LEFT JOIN and RIGHT JOIN

is listed below. This syntax can be said to be divided into three parts. The first part of the syntax is the syntax for LEFT JOIN, the second part is the keyword "UNION" and the third part is the syntax for RIGHT JOIN. Here, FULL JOIN is achieved by performing union on the individual LEFT JOIN and RIGHT JOIN of the tables.

```
SELECT table1.column, table2.column
from table1
LEFT JOIN table2
ON table1.common_field = table2.common_field
UNION
SELECT table1.column, table2.column
from table1
RIGHT JOIN table2
ON table1.common_field = table2.common_field;
```

Let us look at an example to understand the concept of FULL JOIN. Now let us perform FULL JOIN on the tables 'customer' and 'odr' that we have also used earlier. Let us try to write a SQL statement where we select to display customer id from the "customer" table and order id from the 'odr' table after performing FULL JOIN on both the tables using the common field "customerid". This FULL JOIN is achieved by performing union of LEFT JOIN and RIGHT JOIN of both the tables. The SQL query for FULL JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 10. In figure 10, it is seen that all the records are displayed.

```
SELECT customer.customerid, customer.city,
odr.orderid
from customer
LEFT JOIN odr
ON customer.customerid = odr.customerid
UNION
SELECT customer.customerid, customer.city,
odr.orderid
from customer
RIGHT JOIN odr
ON customer.customerid = odr.customerid;
```

```
mysql> SELECT customer.customerid, customer.city,
    -> odr.orderid
    -> from customer
    -> LEFT JOIN odr
    -> ON customer.customerid = odr.customerid
    -> UNION
    -> SELECT customer.customerid, customer.city,
    -> odr.orderid
    -> from customer
    -> RIGHT JOIN odr
    -> ON customer.customerid = odr.customerid;
+------------+----------+---------+
| customerid | city     | orderid |
+------------+----------+---------+
| cus01      | ghy      | 001     |
| cus02      | chennai  | 002     |
| cus02      | ghy      | 002     |
| cus03      | shillong | NULL    |
+------------+----------+---------+
4 rows in set (0.32 sec)
```

Figure 10: Example showing use of FULL JOIN

11.4 SQL SUB-QUERIES

A subquery is a query written within another query. Subqueries are also called as nested query or inner queries. Subqueries return data to the main query. Subqueries are often used with the SELECT clause and embedded

within the WHERE clause in SQL. Subqueries do not have a defined syntax but it can be defined as in the general syntax listed below.

```
SELECT table1.column, table2.column
from table1
WHERE condition1 IN
(SELECT table1.column, table2.column
from table2
WHERE condition2);
```

The subqueries are written inside brackets. Let us now look at an example to understand how subqueries work.

```
SELECT *
from customer
WHERE customerid IN
(SELECT customerid
from customer
WHERE city='ghy');
```

In the above subquery example, the subquery is used to get the customerid of all customers whose city is "ghy". Then the main query is used to display all the information of these selected customers. The output of the above SQL statement is displayed in figure 22.

```
mysql> SELECT *
    -> from customer
    -> WHERE customerid IN
    -> (SELECT customerid
    -> from customer
    -> WHERE city='ghy');
+--------------+------------+---------+
| CustomerName | CustomerID | City    |
+--------------+------------+---------+
| bina         | cus01      | ghy     |
| gita         | cus02      | chennai |
| gita         | cus02      | ghy     |
+--------------+------------+---------+
3 rows in set (0.01 sec)
```

Figure 11: Example showing use of subquery

11.5 SQL Constraints

Constraints can be said to be rules and conditions that are applied on the data of the database. The constraints can either be enforced when we create the database or even later on as an update to the database. Constraints are used to limit the type of data that we enter in the database. Constraints can be applied to either a specific column of the table or to the database table as a whole.

Different types of constraints can be applied to the database. Some of them are:

- Not Null Constraint
- Unique Constraint
- Primary Key Constraint
- Foreign Key Constraint

Let us look at some of the constraints in detail.

11.5.1 Not Null Constraint

In SQL database tables, by default NULL values are accepted. The NOT NULL constraint is used to limit the database from accepting NULL values in the database table or column. When a NOT NULL constraint is applied on a table or a column it ensures that the table or column cannot have not null values.

NOT NULL constraints are generally applied at the creation of the table. Let us look at an example to understand how this constraint can be applied.

```
CREATE TABLE book
(
     book_id          int              not null,
     book_name        varchar(30)      not null,
     price            int,
     publisher        varchar(10),
     primary key      (book_id)
);
```

In the above example, we have applied NOT NULL constraints on the columns "book_id" and "book_name" in the database table "book". The other columns in this table "book", "price" and "publisher" can accept NULL values since by default NULL values are accepted and no NOT NULL constraint has been applied on these two columns.
NOT NULL values can also be applied after the table has been applied using an UPDATE command. Let us make all the columns of the table "book" not accept NULL values. For this, we will have to update the remaining columns of the table "price" and "publisher" to not accept NULL values. The SQL statement to update the table with NOT NULL constraints for both the columns "price" and "publisher" is listed below.

```
ALTER TABLE book MODIFY
price      int              NOT NULL;


ALTER TABLE book MODIFY
publisher  varchar(10)      NOT NULL ;
```

In the first SQL statement we use ALTER command to modify the column "price" in the table "book" to not accept NOT NULL values. In the second statement, we modify the column "publisher" in the table "book" to not accept NULL values.

11.5.2 Unique Constraint

The UNIQUE constraint is used to limit the columns of database to have identical values. The UNIQUE constraint restricts the database to have unique values for those columns for which this constraint has been applied. Like the NOT NULL constraint the UNIQUE constraint can also be either applied at the creation of the database or after creation. Let us look at an example where the UNIQUE constraint is declared at the creation of the database table.

```
CREATE TABLE book
(
     book_id          int              not null,
     book_name        varchar(30)      not null  unique
);
```

In the above SQL statement, a table "book" has been created where both the columns "book_id" and "book_name" have NOT NULL constraints and the column "book_name" has UNIQUE constraint. This means that no two values for the column "book_name" should have identical values. All the values for "book_name" should be different.
The UNIQUE constraint can also be applied afterwards using ALTER TABLE command, if the table has already been created. Let us look at an example where the UNIQUE constraint is declared after the table is already created. Let us make even the "book_id" column of the table "book" have the UNIQUE constraint. For this, we will have to update the table "book". The SQL statement to update the table with UNIQUE constraints for "book_id" column is listed below.

```
ALTER TABLE book MODIFY
book_id    int   NOT NULL UNIQUE;
```

In this SQL statement we use ALTER command to modify the column "book_id" in the table "book" to accept only UNIQUE vales.

11.5.3 Primary Key Constraint

A primary key is a column or a set of columns used to uniquely identify the records in a database table. A primary key cannot have NULL values and all the values for primary key column must have unique values. Since the primary key uniquely define each record in the database table, so for each record to be unique, the values for those col-

umns declared as primary key must not have identical values. Let us look at an example where the PRIMARY KEY constraint is declared at the creation of the database table.

```
CREATE TABLE book
(
     book_id          int              not null,
     book_name        varchar(30)      not null,
     primary key (book_id)
);
```

In the above SQL statement, a table "book" has been created where both the columns "book_id" and "book_name" have NOT NULL constraints and the column "book_id" has PRIMARY KEY constraint. This means that no two values for the column "book_id" should have identical values. All the values for "book_id" should be different.

The PRIMARY KEY constraint can also be applied afterwards using ALTER TABLE command, if the table has already been created. Let us look at an example where the PRIMARY KEY constraint is declared after the table is already created. Suppose we want to add PRIMARY KEY constraint to "book_id" column of the table "book" after the table "book" has already been created. For this, we will have to update the table "book". The SQL statement to update the table with PRIMARY KEY constraints for "book_id" column is listed below.

```
ALTER TABLE book ADD PRIMARY KEY (book_id);
```

In this SQL statement we use ALTER command to add primary key constraint to the column "book_id" in the table "book".

11.5.4 Foreign Key Constraint

A Foreign Key is used as a link between two tables in a database. A foreign key is a column or a collection of columns in one table and a

primary key in the other table. The foreign key is a primary key that uniquely defines the records in one table and all the values of the foreign key in the second table must correspond with those values in the first table. Let us look at an example where a column is declared as PRIMARY KEY in one table and as FOREIGN KEY in another table at the time of creation of both the database tables.

```
CREATE TABLE book
(
     book_id          int              not null,
     book_name        varchar(30)      not null,
     primary key (book_id)
);
```

In the above SQL statement, the column "book_id" is declared as primary key in the first database table "book". In the SQL statement below, the same column will be referenced as foreign key in a second database table "customer".

```
CREATE TABLE customer
(
     customer_id      int              not null,
     customer_name    varchar(30)      not null,
     book_id          int   references book(book_id),
     primary key (customer_id)
);
```

The FOREIGN KEY constraint can also be applied afterwards using ALTER TABLE command, if the second table has already been created. Let us look at an example where the FOREIGN KEY constraint is declared after the table is already created. Suppose we want to add FOREIGN KEY constraint to "book_id" column of the table "customer" after the table "customer" has already been created. For this, we will

have to alter the table "customer". The SQL statement to alter the table with FOREIGN KEY constraints for "book_id" column is listed below.

```
ALTER TABLE customer ADD FOREIGN KEY (book_id)
REFERENCES book(book_id);
```

In this SQL statement we use ALTER command to add foreign key constraint to the column "book_id" in the table "customer", which references it from the column "book_id" in the table "book".

Apart from these four constraints there are many other constraints like Check constraint, Default constraint etc.



CHECK YOUR PROGRESS

Q1. Give the syntax of INNER JOIN.

Q2. Define subqueries.

Q3. Define constraints.

Q4. What is the purpose of using Not Null constraint?



11.6 LET US SUM UP

- INNER JOIN is used to retrieve only records that have matching values in all the tables considered.
- LEFT (OUTER) JOIN or LEFT JOIN is used to retrieve all the records from the left table (or table 1) and only matching records from the right table (table 2). If there are no matching records from the right table then a value of NULL is displayed.
- RIGHT (OUTER) JOIN or RIGHT JOIN is used to retrieve all the records from the right table (or table 2) and only matching records

from the left table (table 1). If there are no matching records from the left table then a value of NULL is displayed.

- FULL (OUTER) JOIN or FULL JOIN is used to retrieve all the records from both the tables when there is a match. If some values are missing from matched records then NULL values are displayed.
- A subquery is a query written within another query. Subqueries are also called as nested query or inner queries. Subqueries return data to the main query.
- In SQL database tables, by default NULL values are accepted. The NOT NULL constraint is used to limit the database from accepting NULL values in the database table or column.
- The UNIQUE constraint is used to limit the columns of database to have identical values. The UNIQUE constraint restricts the database to have unique values for those columns for which this constraint has been applied.
- A primary key is a column or a set of columns used to uniquely identify the records in a database table.
- A foreign key is used as a link between two tables in a database. A foreign key is a column or a collection of columns in one table and a primary key in the other table.



11.7 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.

11.8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: The syntax of INNER JOIN is:

```
SELECT table1.column, table2.column
from table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

Answer to Q2: A subquery is a query written within another query. Subqueries are also called as nested query or inner queries.

Answer to Q3: Constraints are rules and conditions that are applied on the data of the database. The constraints can either be enforced when we create the database or even later on as an update to the database. Constraints are used to limit the type of data that we enter in the database.

Answer to Q4: The NOT NULL constraint is used to limit the database from accepting NULL values in the database table or column. When a NOT NULL constraint is applied on a table or a column it ensures that the table or column cannot have not null values

11.9 MODEL QUESTIONS

Q1. Consider a table home_loan (loan_no, account_no, name, total_amount, amount_due, interest, start_date, end_date) where field "loan_no" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display the names of customers where the amount of loan is more than 10lakhs.

ii. Display the names and account no of customers who have less than 5lakhs payment due.

iii. Display the number of loans given from 1/1/2011 to 31/12/2016.
iv. Display the account_no of customers whose interest is less than 5%.
v. Perform any function using group by clause.

Q2. Consider a table bank_customer (account_no, name, amount, account_type, city) where field "account_no" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.
i. Display the names of customers where name starts with "As".
ii. Display the names of customers whose account has more than 50000 but less than 200000.
iii. Display the names and account_no of customers with city as "Guwahati".
iv. Display the account_no of customers whose account_type is not "savings"
v. Perform any function using Having By clause.
vi. Display the number of account holders with city as "Guwahati".

Q3. Perform Inner Join on the tables "bank_customer" and "home_loan".

Q4. Perform Left Join on the tables "bank_customer" and "home_loan".

Q5. Describe NOT NULL constraints. Why do we need NOT NULL constraints?

Q6. Explain the concept of Right Join.

Q7. Describe subqueries using examples.

Q8. Describe the use of different logical operators in database.

Q9. Describe Primary key and Foreign key constraints.

Q10. Perform FULL OUTER JOIN on the tables ""bank_customer" and "home_loan".

UNIT 12: WORKING WITH GUI

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 Introduction to WAMP

12.4 Installation of WAMP

12.4.1 Starting the Wampserver

12.5 Working with phpMyAdmin

12.5.1 Creating a Database and Table

12.5.2 Inserting Records into a Table

12.5.3 Searching/Displaying Records

12.5.4 Updating Records

12.5.5 Database Backup

12.5.6 Database Restore

12.6 Let Us Sum Up

12.7 Answers to Check Your Progress

12.8 Further Readings

12.9 Model Questions

12.1 LEARNING OBJECTIVES

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

- explain the purpose of WAMP and its components (Windows, Apache, mysql, and PHP).
- install WAMP on a Windows computer.
- navigate and use the phpmyadmin web interface for managing databases and tables.
- create a database in phpmyadmin and define its properties
- create a table in phpmyadmin and define its fields
- insert records into a table in phpmyadmin
- update records in a table in phpmyadmin
- search and display records in a table in phpmyadmin
- backup a database in phpmyadmin and save it as a file.
- restore a database backup in phpmyadmin and recover its data.

12.2 INTRODUCTION

In the world of web development, having a local environment for testing and developing applications is crucial. This is where WAMP (Windows, Apache, MySQL, and PHP) comes in handy. WAMP is a software stack that allows developers to set up a local web server on their Windows machines, including Apache as the web server, MySQL as the database management system, and PHP as the programming language. In this unit, we will cover the installation and usage of WAMP, including creating a database and table, inserting and updating records, searching and displaying records, and backing up and restoring databases using phpMy Admin.

12.3 INTRODUCTION TO WAMP

WAMP is an acronym for Windows, Apache, MySQL, and PHP. It's a software stack that is commonly used for web development on the Windows operating system. Here's a brief explanation of each component:

- Windows: The operating system used for hosting the web server and running the web application.
- Apache: A popular web server software that provides the necessary infrastructure to serve web pages over the internet. Apache is used to process HTTP requests and responses.
- MySQL: A database management system used to store data for web applications. MySQL is used to create, modify, and manage databases.
- PHP: A popular programming language used for web development. PHP is used to write server-side code that interacts with the database and generates dynamic web pages.

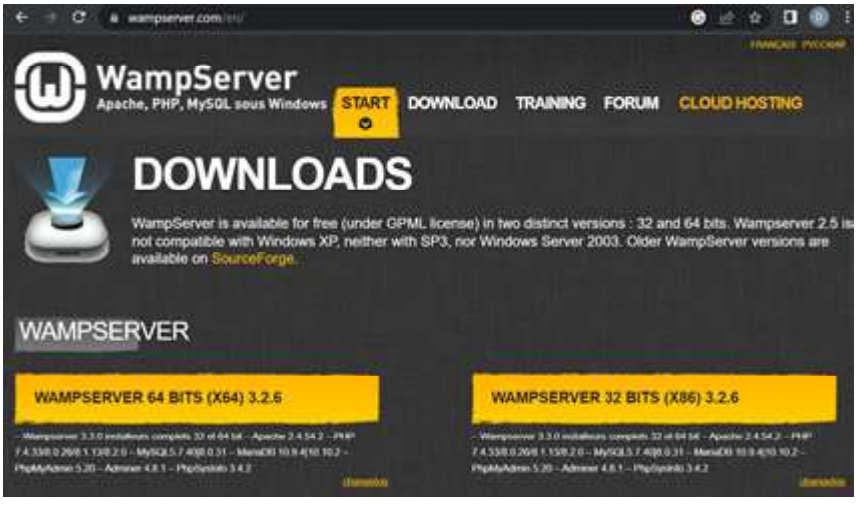
WAMP is a complete package that includes all the necessary components required for web development on a Windows machine. It's an easy way to get started with web development as all the components are

pre-configured and installed together, allowing developers to focus on writing code rather than setting up the infrastructure. Additionally, WAMP also includes tools for managing databases and configuring the web server, making it a convenient option for developers looking to set up a local development environment.

phpMyAdmin is a web-based application that provides a graphical user interface (GUI) for managing MySQL databases. With WAMP, phpMyAdmin is included as one of the tools that comes pre-installed.

12.4 INSTALLATION OF WAMP

WAMP can be downloaded from the official website: http:// www.wampserver.com/en/.



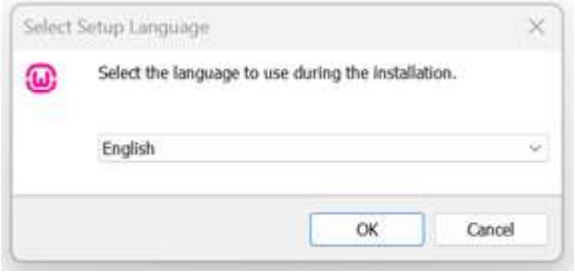
On the homepage, you'll find the download link for the latest version of WAMP. Click on the "Download" button to start the download process.

Once the download is complete, run the installer and follow the installation wizard to install WAMP on your Windows system. During the installation process, you can choose the components you want to install and configure the settings according to your requirements.

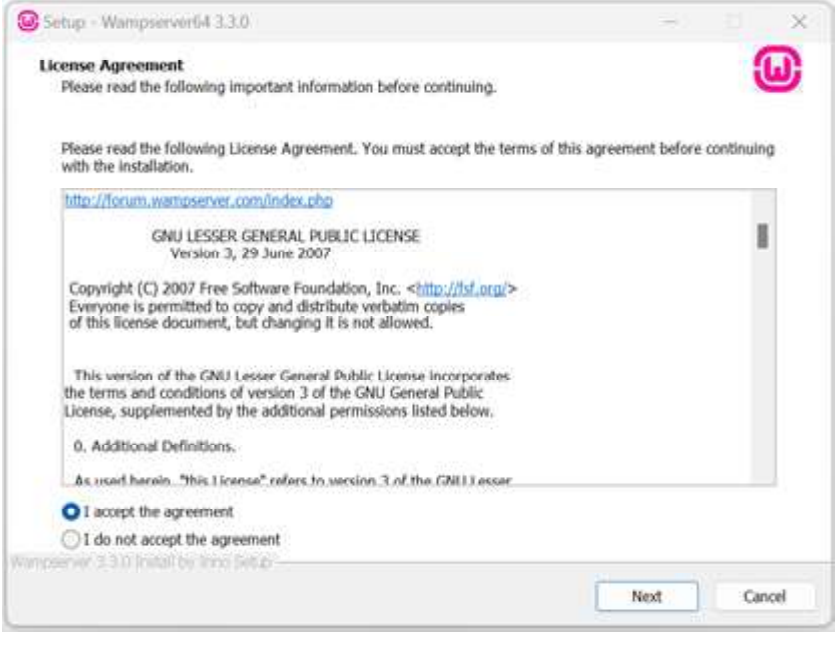
It is recommended to download WAMP from the official website to ensure that you have the latest and most stable version. Additionally,

downloading from the official website reduces the risk of downloading malware or other security threats.

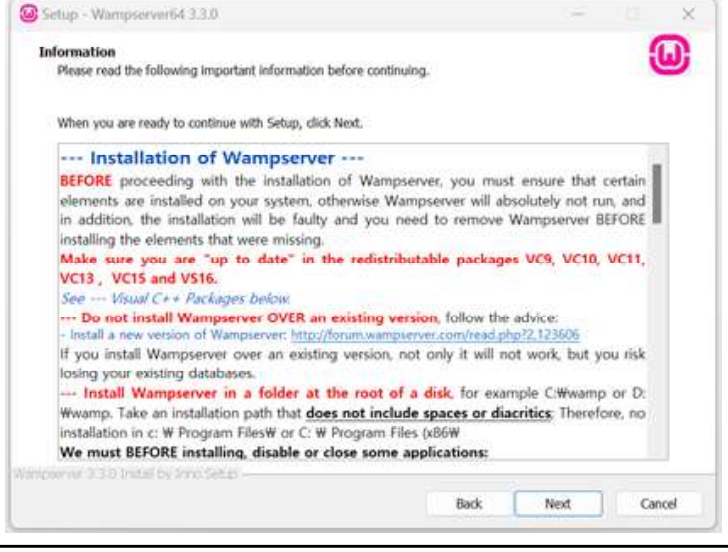
Step 1:



Step 2:

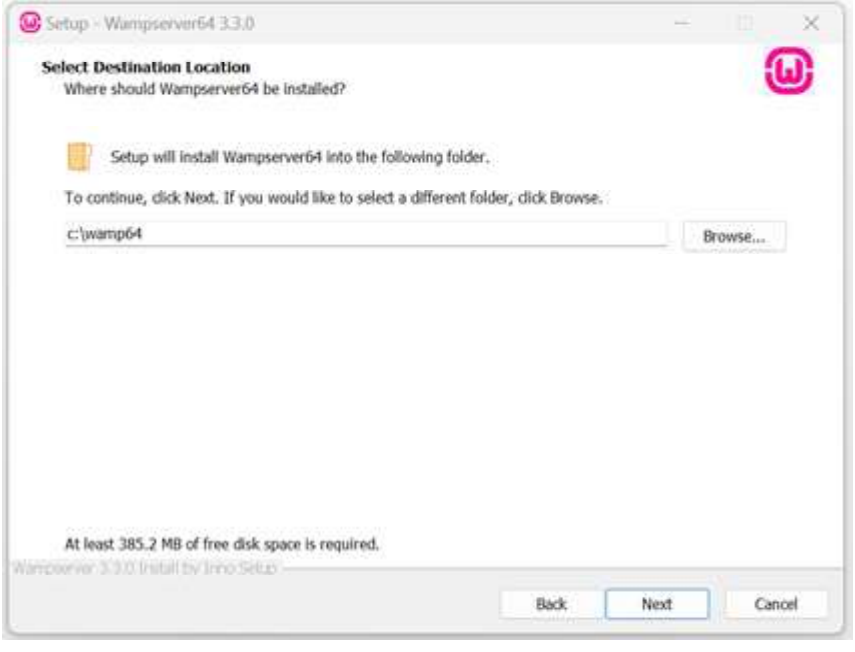


Step 3:

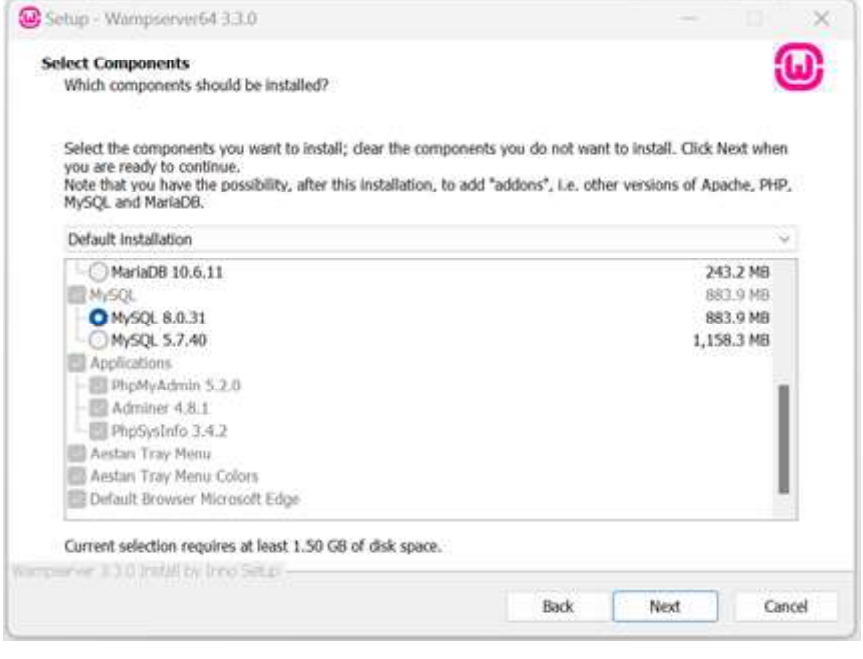


Note: Please click all the links that are provided in the above window, download the files and install all one by one. Don't skip anything from there. Then only the installation process will succeed. Otherwise, the Wampserver will not work.

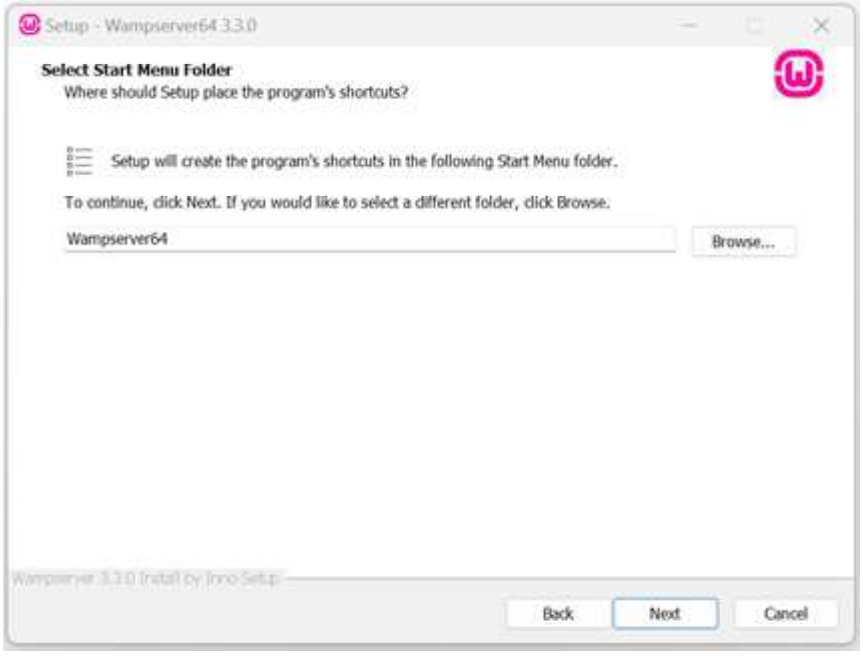
Step 4:



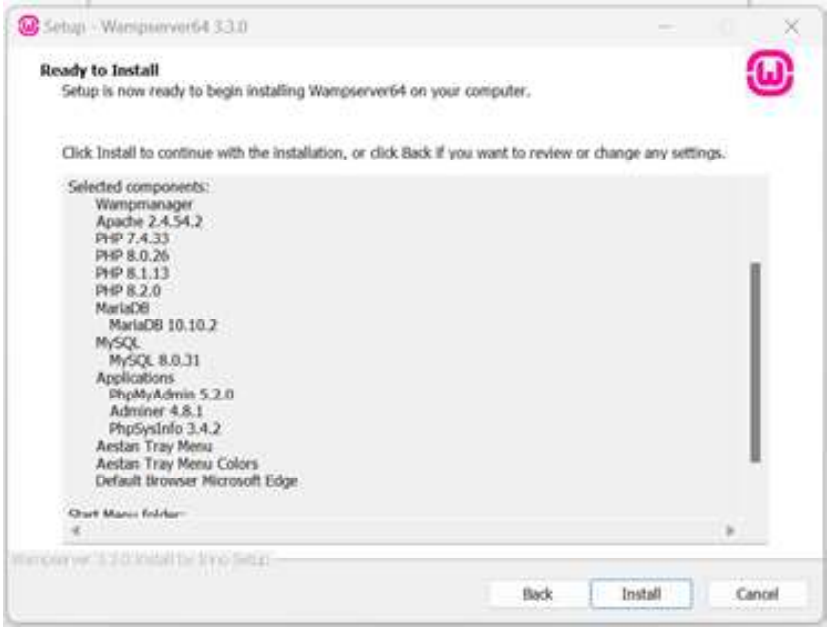
Step 5:



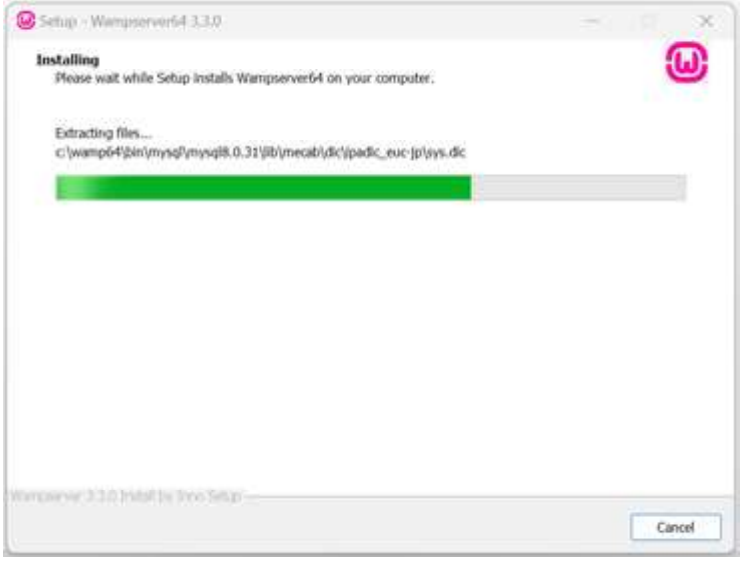
Step 6:


Setup - Wampserver64 3.3.0
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Wampserver64
Browse...
Back
Next
Cancel

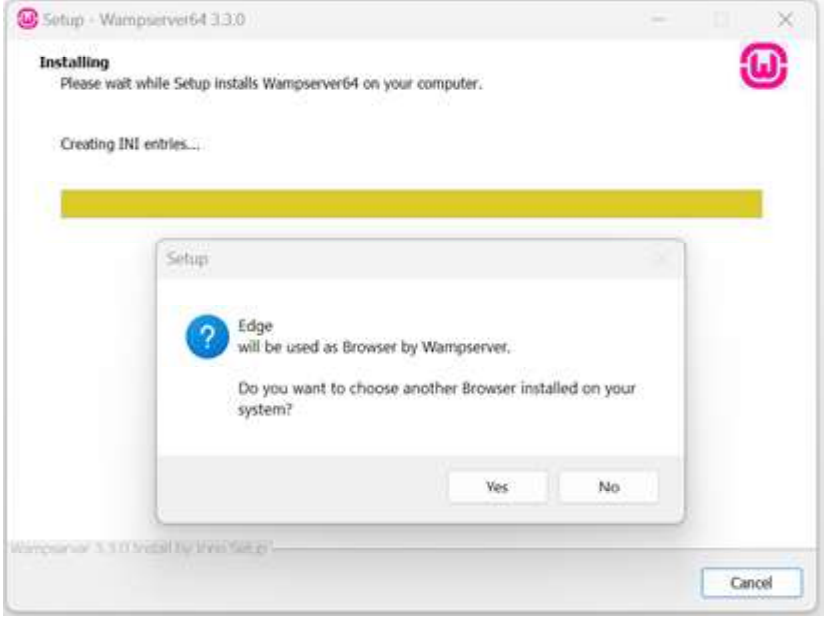
Step 7:


Setup - Wampserver64 3.3.0
Ready to Install
Setup is now ready to begin installing Wampserver64 on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Selected components:
Wampmanager
Apache 2.4.54.2
PHP 7.4.33
PHP 8.0.26
PHP 8.1.13
PHP 8.2.0
MariaDB
MariaDB 10.10.2
MySQL
MySQL 8.0.31
Applications
PhpMyAdmin 5.2.0
Adminer 4.8.1
PhpSysInfo 3.4.2
Aestan Tray Menu
Aestan Tray Menu Colors
Default Browser Microsoft Edge
Back
Install
Cancel

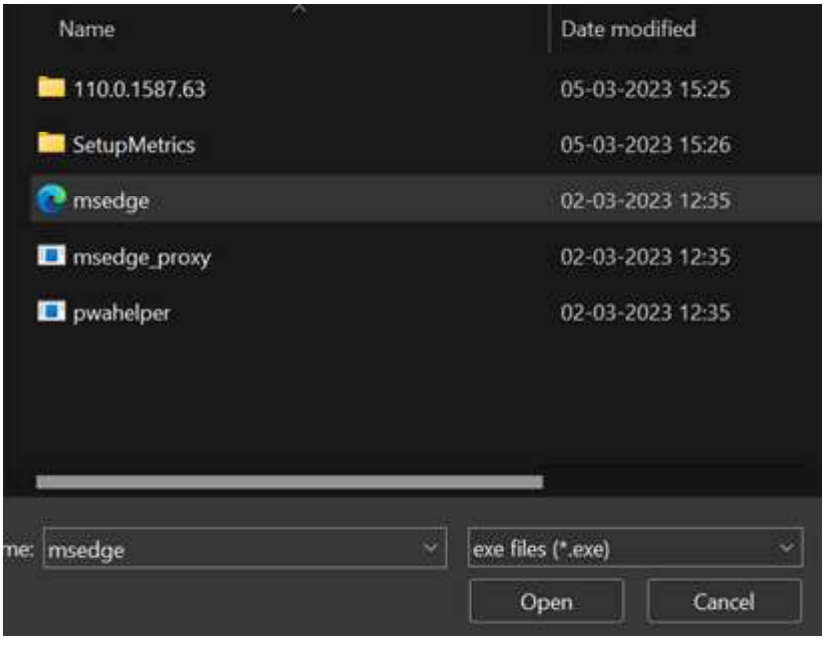
Step 8:


Setup - Wampserver64 3.3.0
Installing
Please wait while Setup installs Wampserver64 on your computer.
Extracting files...
c:\wamp64\bin\mysql\mysql8.0.31\lib\mecab\dic\ipadic_euc-jp\sys.dic
Cancel

Step 9:

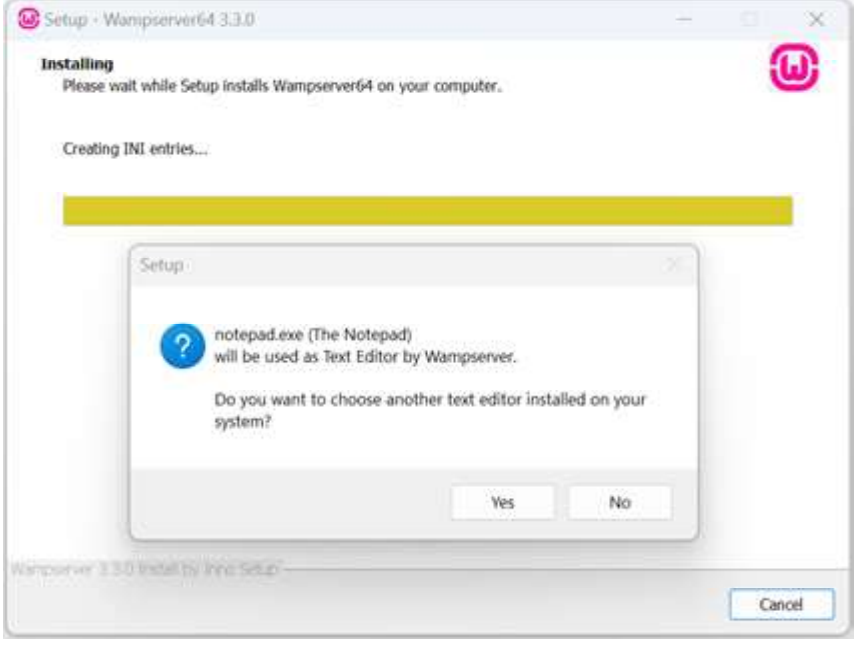

Setup - Wampserver64 3.3.0
Installing
Please wait while Setup installs Wampserver64 on your computer.
Creating INI entries...
Setup
Edge
will be used as Browser by Wampserver.
Do you want to choose another Browser installed on your system?
Yes
No
Cancel

Step 10:

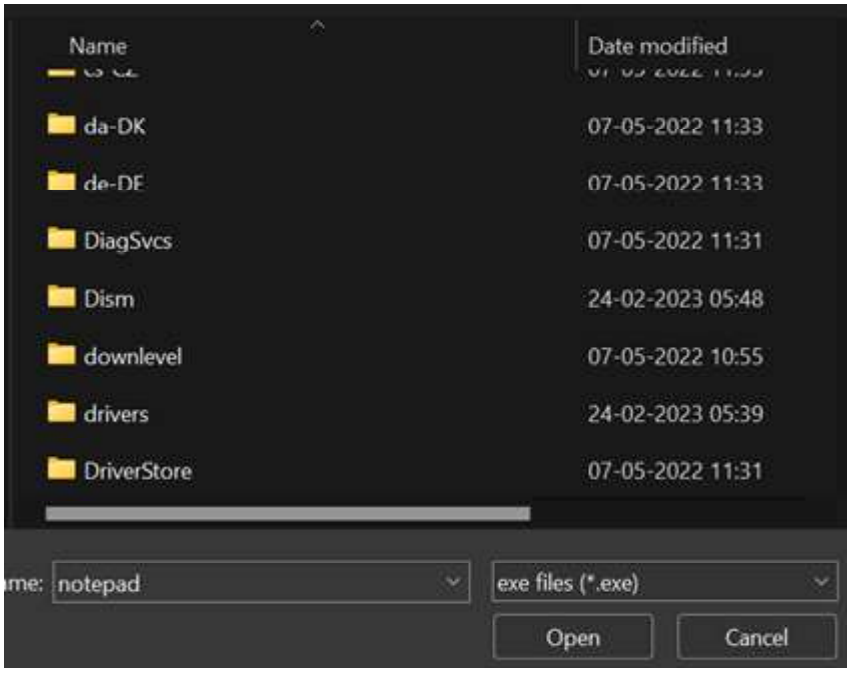


Choose the default browser and click "Open".

Step 11:

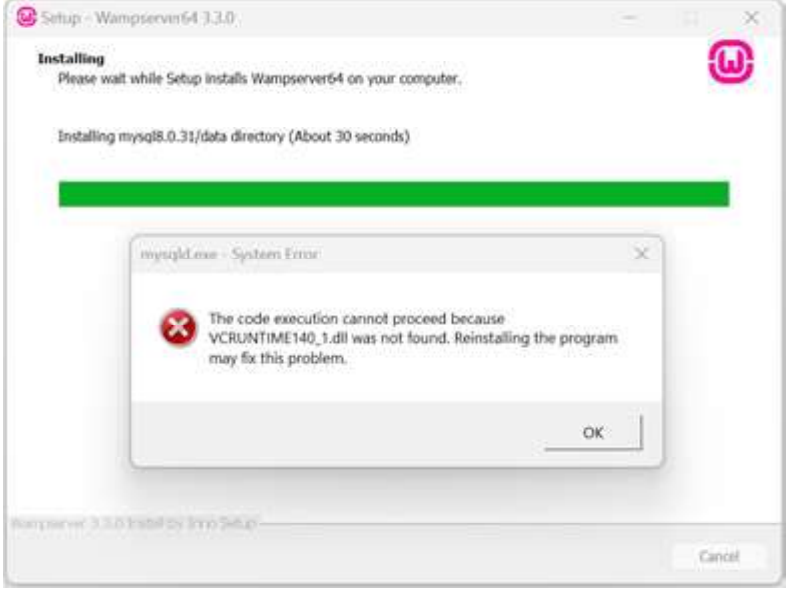


Step 12:

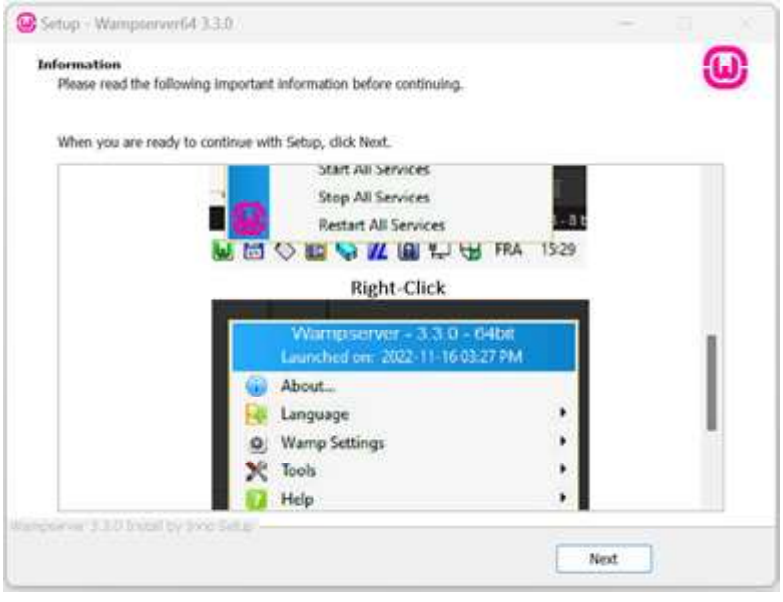


Step 13:

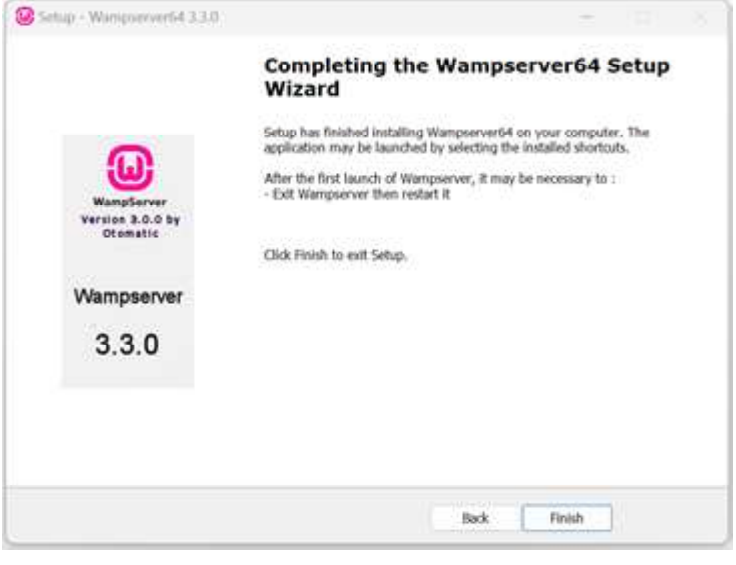
If you skipped the installation of any of the files that were mentioned in Step 3, you may encounter a screen that prompts you to install the missing files. In that situation, you have to uninstall the Wampserver that you have installed and freshly install it again.



Step 14:



Step 15:



12.4.1 Starting the Wampserver

Once the installation completes, let's discuss how to start the Wampserver now.

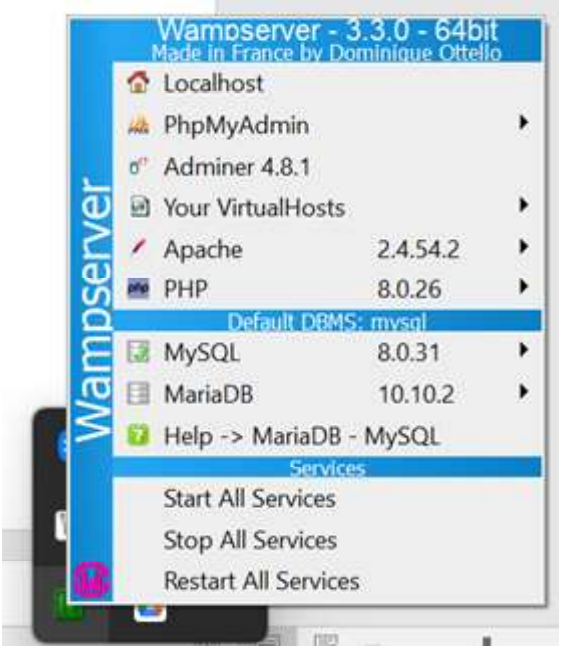
Step 1: Double click the following shortcut icon in the Desktop.



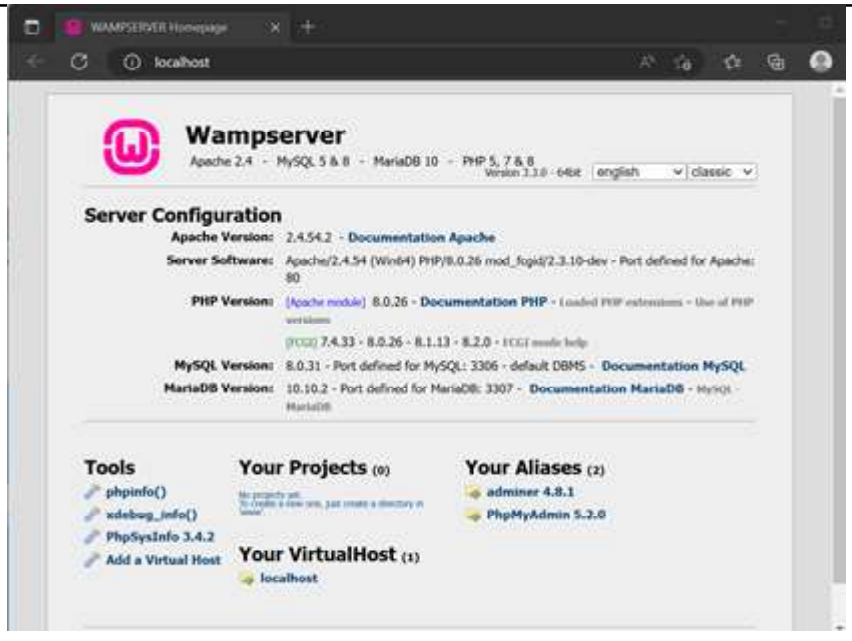
Step 2: After the Wampserver starts, the following green icon in the status bar of the Windows will be seen.



When you click the green icon, the following window will be displayed. Now you can explore the various menu items in the window.



When you click the Localhost option, the following screen will be opened in your browser. It shows the information of Wampserver installed in your computer.



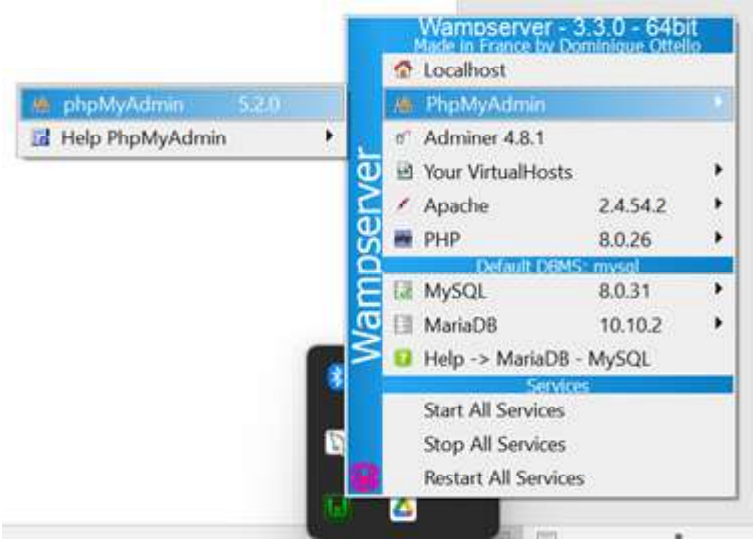
12.5 WORKING WITH PHPMYADMIN

phpMyAdmin allows users to easily manage their MySQL databases without needing to use the command-line interface. It provides a user-friendly interface for performing tasks such as creating and deleting databases, creating and editing tables, managing users and permissions, executing SQL queries, and exporting or importing data.

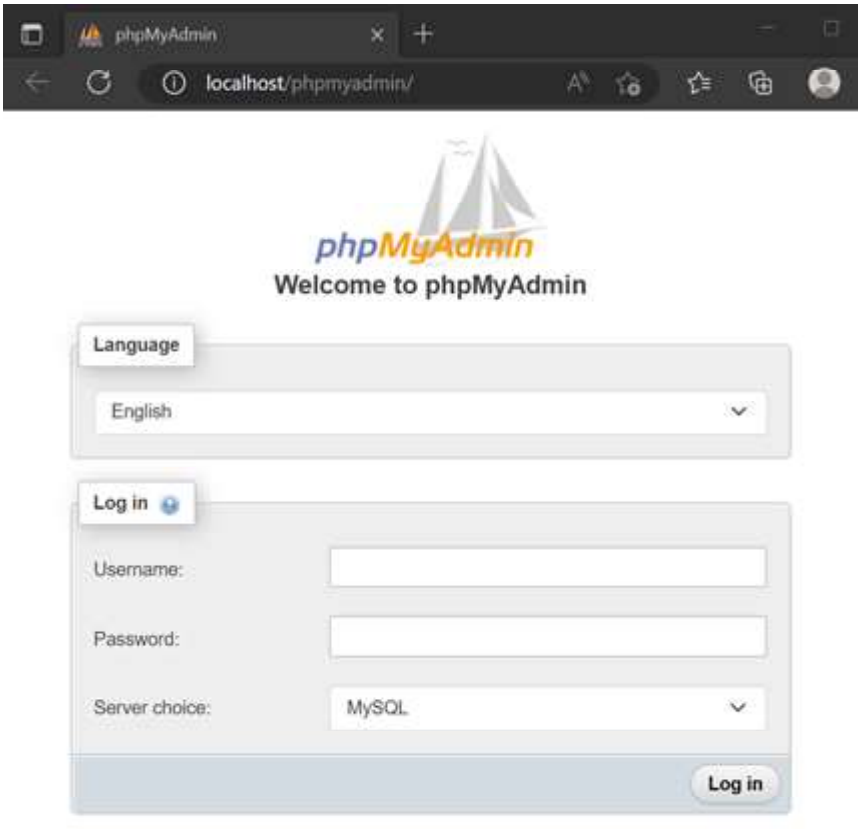
In WAMP, phpMyAdmin can be accessed through a web browser by navigating to http://localhost/phpmyadmin/. Once logged in, users can manage their MySQL databases using the phpMyAdmin interface.

Overall, phpMyAdmin is an essential tool for managing MySQL databases in a user-friendly way, and it simplifies database management for users who are not familiar with command-line tools.

To start phpMyAdmin, click the following options as shown the in the following window:

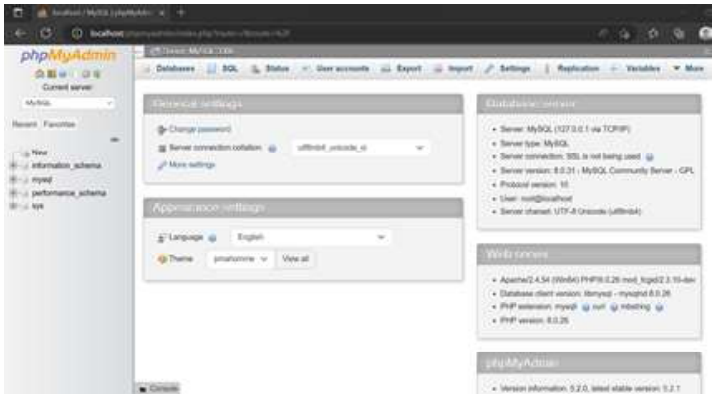


The following window will be opened in the browser.



Here, you have to give the Username and Password to interact with the MySQL server.

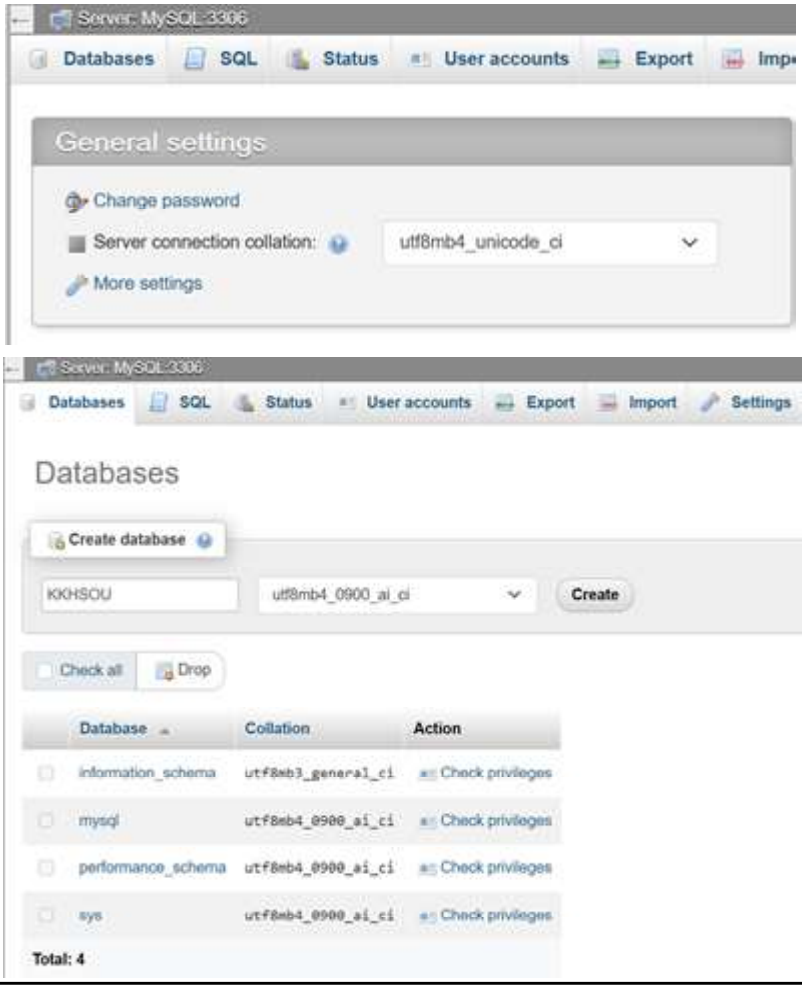
Put Username as root, leave the Password field as blank and click the Log in button. The following window will be displayed.



Now, you are ready to work with MySQL with phpMyAdmin.

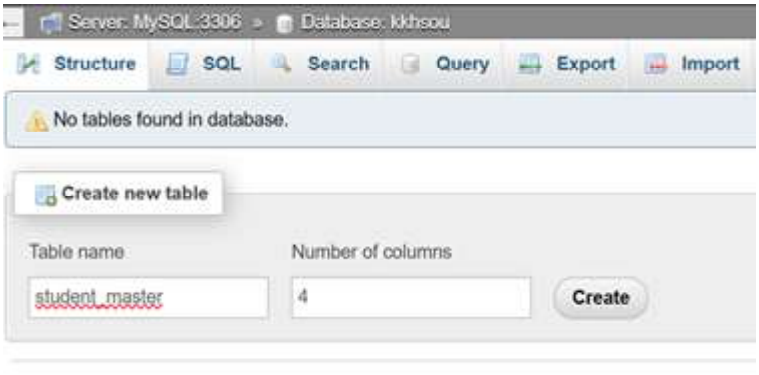
12.5.1 Creating Database and Table

1. See the navigation pane on the middle part of the above page. Click on the "Databases" tab to create a new database.

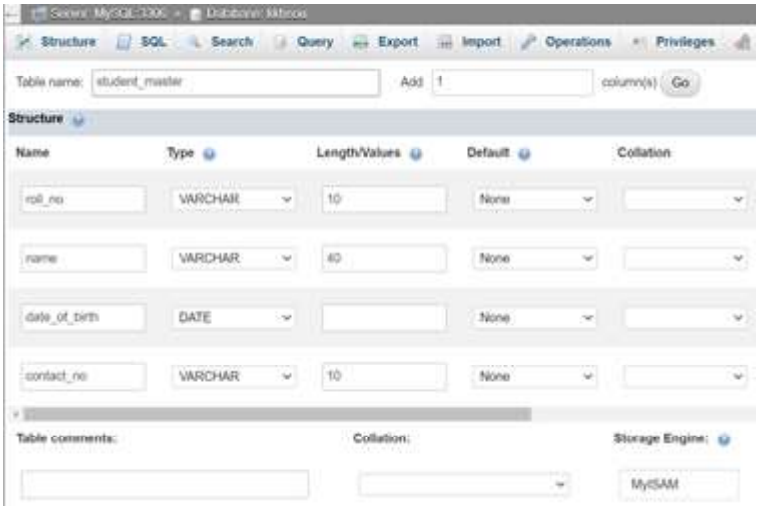


You have to provide the database name. Here, we have given the database name as KKHSOU. Now, when we create the Create button the following page will be displayed where we have to give the table name to create.

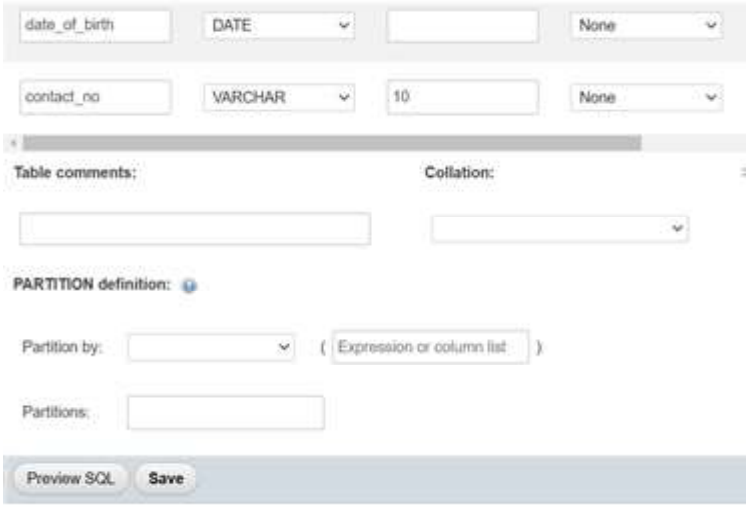
Let us create a table named "student_master" with four fields roll_no, name, date_of_birth, contact_no. So, type student_master in the Table name field and 4 in the Number of columns text boxes and click the Create button as shown below:



The following screen will appear. Here, we have to put the four field names along with Types and Length/Values.



Then we have to click the Save button as shown below:

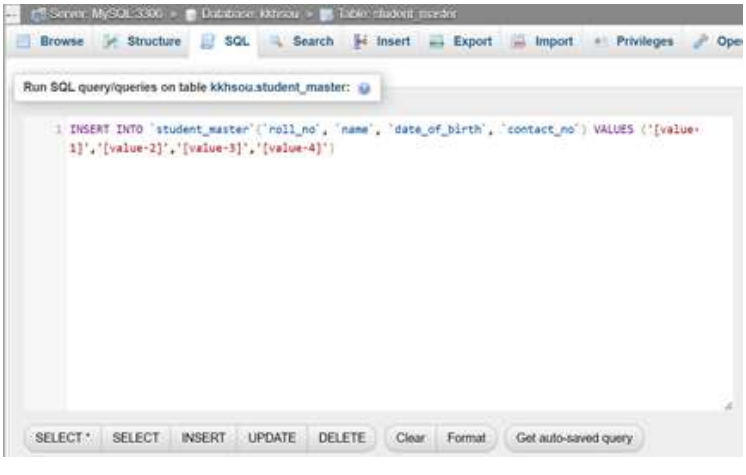


The table "student_master" will be created and the following screen will be displayed.



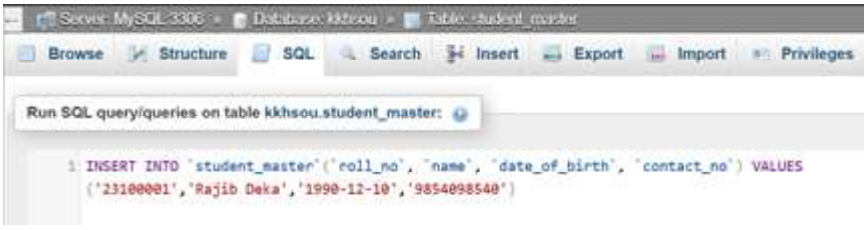
12.5.2 Inserting Records into a Table

To inert record into the table 'student_master', click the database named 'KKHSOU', then click the 'student_master' table. Then click the SQL tab, and click the INSERT tab as shown in the screen below:



Here, you will notice that the SQL statement needed to insert record into the 'student_master' table will be displayed as ready reference. You have to modify the SQL statement there.

Let us modify the SQL statement as:



To execute the SQL statement, click the Go button as shown in the screen below:



One record will be inserted.

In this way, you can insert number of records into the table.

12.5.3 Searching/Displaying Records

To view the records stored in a database table, you can use the Browse or SQL tabs in your database management system.

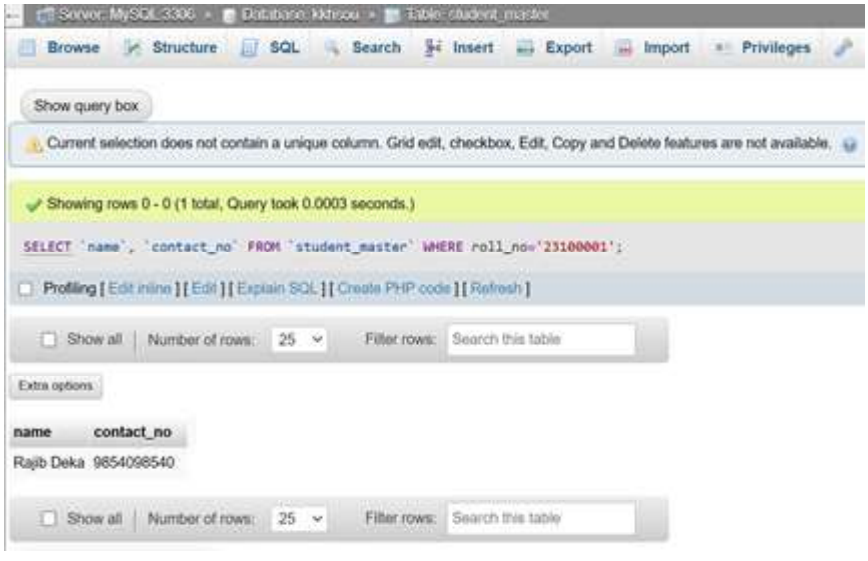
In the Browse tab, you can see all the existing records in the table. Alternatively, you can switch to the SQL tab and use the SELECT statement to display specific records from the table.

For example, to display the name and contact no of the student whose roll_no is 23100001, you can click on the SQL tab and enter the following SELECT statement:



Execute the statement by clicking the Go button.

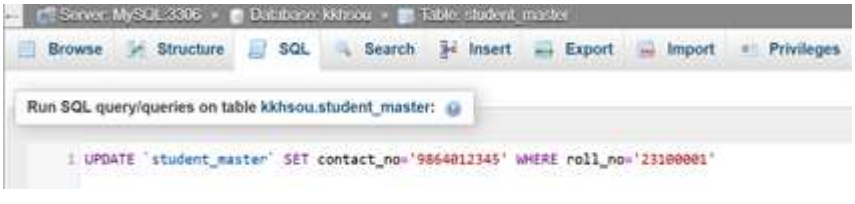
The following screen will be displayed.



12.5.4 Updating Records

To update a record in the table, click the UPDATE button under the SQL tab and modify the sample UPDATE statement as per your requirement.

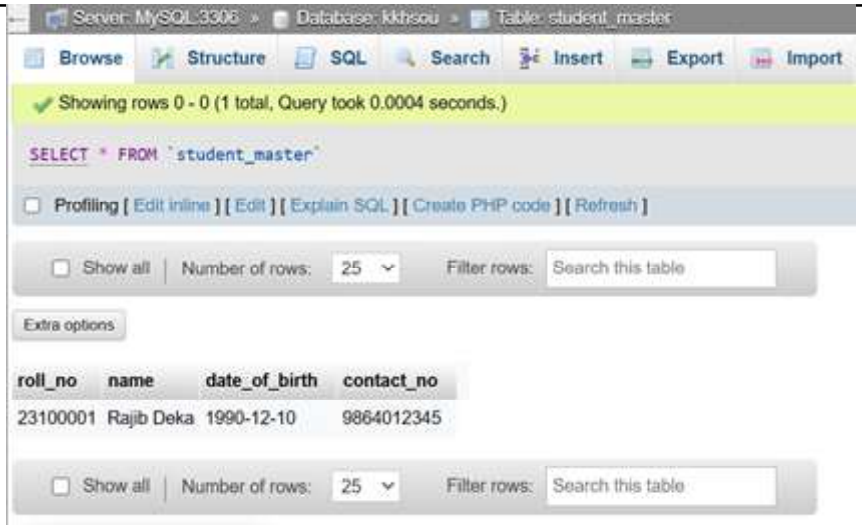
Let us update the mobile no to 9864012345 for the student whose roll_no is 23100001'



Execute the statement by clicking the Go button.

The following screen will be displayed.

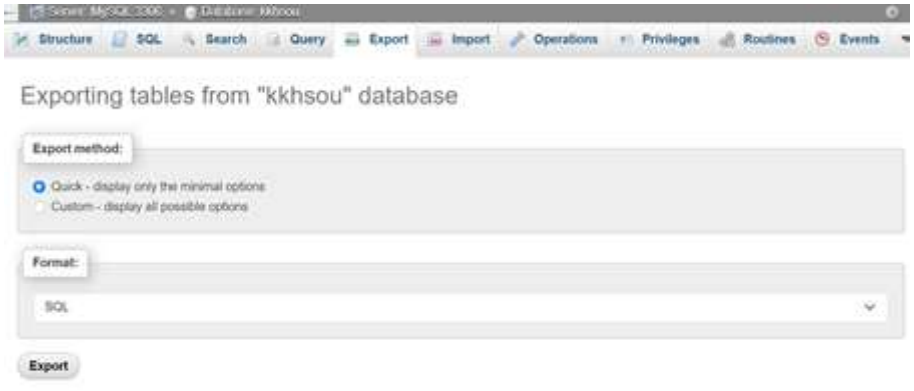
To check whether the record is updated or not, click the Browse button. The following screen will be displayed.



12.5.5 Database Backup

You can take a backup of a MySQL database using phpMyAdmin by following these steps:

1. Log in to phpMyAdmin using your username and password.
2. Select the database that you want to backup from the left-hand navigation menu.
3. Click on the "Export" tab at the top of the page.



4. Select the "Custom" export method.
5. Choose the tables you want to backup or select all tables by clicking on the checkbox next to "Select All" at the bottom.

6. In the "Output" section, choose the format of the backup file. The default option is SQL.
7. Choose the compression type for the backup file. You can choose "None" or one of the available compression options.
8. In the "Object creation options" section, choose the options you want to include in the backup file.
9. In the "Data dump options" section, choose the options you want to include in the backup file.
10. Click on the "Export" button to start the backup process.
11. Save the backup file to your local computer.

12.5.6 Database Restore

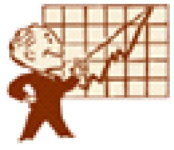
To restore a MySQL database using phpMyAdmin, you can follow these steps:

1. Log in to your phpMyAdmin account using your username and password.
2. On the left-hand side of the screen, select the database that you want to restore.
3. Click on the "Import" tab at the top of the screen.
4. Under the "File to Import" section, click on the "Choose File" button and select the SQL file you want to restore.
5. Choose the appropriate character set for your database from the "Character set of the file" drop-down menu.
6. Under the "Format" section, select "SQL" from the drop-down menu.
7. If you want to overwrite existing data, make sure the "Partial import" option is unchecked.

8. Click on the "Go" button to start the restore process.

Wait for the import process to complete. This may take some time depending on the size of your database. Once the restore process is complete, you will see a confirmation message.

CHECK YOUR PROGRESS



1. Answer the following questions:

a) Wampserver is a software that bundles together which three components?

b) What is the purpose of Wampserver?

c) What is PHPMyAdmin?

d) What is the default username and password for PHPMyAdmin in Wampserver?

e) What is the purpose of the MySQL database in Wampserver?

f) What is the purpose of the www directory in Wampserver?

g) How do you start and stop the Wampserver services?

h) How do you access PHPMyAdmin in Wampserver?



12.6 LET US SUM UP

- WampServer is a software stack for Windows operating system that includes Apache, MySQL, and PHP. It is used to create and test dynamic web applications locally on a computer.
- With WampServer, developers can easily set up a web server

environment without the need to install and configure each component separately.

- WampServer provides a user-friendly interface to manage the server and its services, including Apache and MySQL, through a web browser or a system tray icon.
- It supports multiple PHP versions, virtual hosts, and database management with phpMyAdmin.
- MySQL is a popular open-source relational database management system used for storing and managing data for web applications.
- phpMyAdmin is a web-based graphical user interface tool used for managing MySQL databases. It allows users to create, modify, and delete databases, tables, and columns.
- You can create a new MySQL database using phpMyAdmin and import an existing SQL file or manually create tables and columns.
- phpMyAdmin also provides tools for managing users, setting up user permissions, and creating backups of your MySQL databases.



12.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1:(a) Apache, MySQL, and PHP.

Ans. to Q. No. 1:(b) Wampserver is used to create a local web server environment on a Windows computer, which allows developers to test and develop web applications before deploying them to a live server.

Ans. to Q. No. 1:(c) PHPMyAdmin is a free and open-source web-based application used to manage MySQL databases.

Ans. to Q. No. 1:(d) The default username for PHPMyAdmin in Wampserver is "root", and the default password is blank (i.e., no password).

Ans. to Q. No. 1:(e) The MySQL database in Wampserver is used to store and manage data for web applications.

Ans. to Q. No. 1:(f) The www directory in Wampserver is the root directory for web applications, where all the web files and folders are stored.

Ans. to Q. No. 1:(g) You can start and stop the Wampserver services from the Wampserver icon in the system tray. Right-click on the icon and select "Start All Services" or "Stop All Services".

Ans. to Q. No. 1:(h) To access PHPMyAdmin in Wampserver, open a web browser and go to http://localhost/phpmyadmin.



12.8 FURTHER READINGS

1) https://www.phpmyadmin.net/

2) https://www.tutorialspoint.com/phpmyadmin/index.htm



12.9 MODEL QUESTIONS

Q.1: What is WampServer and how is it used for web development?

Q.2: What is the role of PHPMyAdmin in WampServer and how is it used to manage MySQL databases?

Q.3: How do you install WampServer on a Windows computer and configure it for web development?

Q.4: How do you configure PHPMyAdmin to connect to a MySQL database in WampServer?

Q.5: How can you import and export databases using PHPMyAdmin in WampServer?

Q.6: How can you create a new database and table using PHPMyAdmin in WampServer?

Q.7: How can you use PHPMyAdmin to execute SQL queries on a MySQL database in WampServer?

Q.8: How can you secure WampServer, PHPMyAdmin and MySQL to protect your web applications and data?

*** ***** ***

UNIT 13: DATABASE RECOVERY

UNIT STRUCTURE

13.1 Learning Objectives
13.2 Introduction
13.3 Database Failure
13.4 Database Recovery
13.5 Backup of Database for Catastrophic Failure
13.6 Recovery for Non-Catastrophic Failures
    13.6.1 Deferred Update
    13.6.2 Immediate Update
13.7 Let Us Sum Up
13.8 Further Reading
13.9 Answers to Check Your Progress
13.10 Model Questions

13.1 LEARNING OBJECTIVES

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

- understand the concept of database failure and the different types of failures that can occur in a database system.
- learn about the importance of database recovery and the steps involved in the recovery process.
- understand the need for backup of a database to prevent data loss in the event of a catastrophic failure.
- learn about the best practices for implementing immediate or deferred updates to recover from non-catastrophic failures.
- understand the importance of having a disaster recovery plan in place to minimize the impact of a catastrophic failure and ensure business continuity.

13.2 INTRODUCTION

We have already been acquainted with various important concepts related with database and DBMS. We have also learnt to insert, delete,

update and other associated operations in a database through SQL commands. In this unit we will learn how a database can be recovered.

In today's digital age, data is considered one of the most valuable assets for individuals and businesses alike. As such, it is crucial to have effective measures in place to ensure the security and integrity of data stored in databases. However, despite the best efforts, database failures can still occur due to various reasons such as hardware or software malfunctions, human errors, or natural disasters.

This unit on Database Failure and Recovery aims to provide students with a comprehensive understanding of the common types of database failures, their causes, and the methods used for recovery. The unit covers various topics such as database backup strategies, catastrophic and non-catastrophic failures, and deferred and immediate updates.

By the end of this unit, students will be equipped with the necessary knowledge and skills to identify, diagnose, and recover from database failures. They will also learn how to implement effective backup and recovery strategies to ensure the continuity of database operations in case of unexpected failures.

13.3 DATABASE FAILURE

A database failure occurs when a database system is unable to perform its functions as expected. Database failures can be caused by a variety of reasons, such as hardware or software issues, network problems, human errors, or natural disasters.

- Database failures can lead to a range of negative consequences, including:
- Data loss: A database failure can result in the loss of important data that may be critical to an organization's operations.

- Downtime: A database failure can cause downtime, which can result in lost productivity and revenue.

- Security breaches: A database failure can lead to security breaches, as hackers or unauthorized users may exploit vulnerabilities in the system to gain access to sensitive data.

- System crashes: A database failure can cause the entire system to crash, which can be difficult and time-consuming to recover from.

To prevent database failures, it's important to have regular backups, implement redundancy and failover mechanisms, monitor system performance, and perform regular maintenance and updates. In case of a failure, it's important to have a disaster recovery plan in place to quickly restore the database system to its previous state.

13.4 DATABASE RECOVERY

Database recovery is the process of restoring a database system to its previous state after a failure or an interruption. Recovery is necessary to ensure the integrity and availability of data in the database system.

The recovery process typically involves the following steps:

- Identification of the failure: The first step in the recovery process is to identify the type and cause of the failure. This can be done through log analysis, error messages, and system alerts.

- Backup restoration: If a recent backup exists, the next step is to restore the database from that backup. This can be a full back up or a differential/ incremental backup depending on the nature of the failure.

- Transaction log replay: If a backup does not exist or if the backup is not recent, the transaction logs are used to replay the transactions that were not yet committed before the failure occurred. This process ensures that the database is restored to a consistent state.

- Testing and validation: Once the recovery process is complete, it's important to test and validate the database to ensure that it's functioning as expected.

- Post-mortem analysis: Finally, a post-mortem analysis should be performed to identify the root cause of the failure and to implement corrective actions to prevent similar failures in the future.

In addition to these steps, it's important to have a disaster recovery plan in place that outlines the steps to be taken in the event of a catastrophic failure. This plan should include backup and recovery procedures, failover mechanisms, and communication protocols.

By following these best practices, database recovery can be a seamless process that ensures the integrity and availability of data in the database system.

13.5 BACKUP OF DATABASE FOR CATASTROPHIC FAILURE

Regular backups of a database are essential to minimize the risk of data loss in the event of a catastrophic failure. A catastrophic failure refers to a severe failure that causes significant damage or disruption to a database system, such as a hardware failure, software corruption, or a natural disaster.

Here are some best practices for backing up a database for catastrophic failure:

- Schedule regular backups: Set up a regular backup schedule that meets the needs of your organization. This can be daily, weekly, or monthly backups depending on the amount and importance of data.

- Use multiple backup methods: Use different backup methods, such as full, differential, and incremental backups to ensure maximum data protection.

- Store backups offsite: Store backups in a secure offsite location, such as a
- cloud storage service, to ensure data availability in the event of a catastrophic failure at your primary site.
- Test backup and recovery procedures: Regularly test your backup and recovery procedures to ensure they are working as expected and can quickly restore data in the event of a failure.

Maintain backups for an appropriate duration: Keep backups for an appropriate duration that aligns with your organization's retention policies and regulatory requirements.

By implementing these best practices, you can minimize the risk of data loss and ensure that your organization can quickly recover from a catastrophic failure of your database system.



CHECK YOUR PROGRESS

Q1. Choose the correct option:

(a) What is a catastrophic failure in a database system?

(i) A minor issue that can be resolved quickly

(ii) A severe failure that causes significant damage or disruption

(iii) A planned outage for maintenance

(iv) A temporary interruption of service

(b) What is the purpose of database recovery?

(i) To create a backup of the database

(ii) To restore the database system to its previous state after a failure

or interruption

(iii) To monitor system performance and conduct regular maintenance

(iv) To implement redundancy and failover mechanisms

(c) What is the first step in the database recovery process?

(i) Backup restoration

13.6 RECOVERY FOR NON-CATASTROPHIC FAILURES

Non-catastrophic failures refer to less severe database failures that are not as disruptive as catastrophic failures but still need to be addressed promptly to avoid data loss and minimize downtime. Examples of non-catastrophic failures include software bugs, human errors, and network disruptions.

Here are some steps to follow for recovery from non-catastrophic failures:

- Identify the cause of the failure: The first step is to identify the root cause of the failure, which could be anything from a software bug to a user error.
- Rollback or restore: Depending on the nature of the failure, you may need to rollback the changes made to the database or restore the database from a recent backup.
- Reapply changes: If you rollback the changes, you may need to reapply the changes that were made before the failure occurred. If you restore the database from a backup, you may need to apply any changes that were made after the backup was taken.

- Verify the recovery: After the recovery process is complete, it's important to verify that the database is functioning as expected and that all data is intact.

- Prevent future failures: Finally, take steps to prevent future failures by addressing the root cause of the failure, implementing system updates and patches, and ensuring that users are properly trained on database usage.

By following these steps, you can recover from non-catastrophic failures and ensure that your database system remains reliable and functional.

13.6.1 Deferred Update

Deferred update is a technique used in database recovery to ensure data consistency in non-catastrophic failures. Non-catastrophic failures refer to less severe database failures that do not result in complete data loss or system downtime, such as software errors, network failures, or user errors.

In deferred update recovery, changes made to the database are not immediately written to disk. Instead, they are kept in memory until a commit statement is issued, indicating that the transaction is complete and all changes should be permanently saved. This approach allows for the recovery of a consistent state of the database in the event of a non-catastrophic failure.

Here are some key considerations when using deferred update for non-catastrophic failures:

- Ensure data consistency: Deferred update must be used in conjunction with transaction management to ensure data consistency. If a transaction is incomplete at the time of failure, any changes made by the transaction must be rolled back to maintain data consistency.

- Monitor transaction duration: Long-running transactions can increase the risk of failure and compromise data consistency. To mitigate this risk, limit the duration of transactions and ensure that they complete within a reasonable time frame.

- Implement recovery procedures: Have recovery procedures in place to ensure that the database can be restored to a consistent state in the event of a failure. This may involve rolling back incomplete transactions or replaying committed transactions from a transaction log.

By implementing deferred update recovery techniques, you can ensure data consistency in the event of a non-catastrophic failure and minimize the risk of data loss or corruption.

13.6.2 Immediate Update

Immediate update of recovery is essential to minimize the impact of non-catastrophic failures and to ensure that the database system is available as soon as possible. Non-catastrophic failures are issues that do not cause severe damage or disruption to a database system, such as a transaction failure, a network connectivity issue, or a software error.

Here are some best practices for immediate update of recovery for non-catastrophic failures:

- Identify the root cause: Quickly identify the root cause of the failure to determine the appropriate recovery action.

- Determine the severity of the failure: Assess the severity of the failure to determine whether it requires immediate recovery or can be resolved in the next scheduled maintenance window.

- Implement the appropriate recovery action: Based on the severity of the failure, implement the appropriate recovery action. This may include rolling back the transaction, restarting the application or database, or

restoring from a backup.

- Communicate with stakeholders: Keep stakeholders informed of the situation, the recovery action taken, and the estimated time to restore the system to normal operations.
- Perform a post-mortem analysis: After the system is restored to normal operations, perform a post-mortem analysis to identify the root cause of the failure and implement corrective actions to prevent similar failures in the future.

By implementing these best practices, you can minimize the impact of non-catastrophic failures and ensure that your database system is available and performing as expected.

The immediate update algorithm is a recovery technique used in database systems to ensure that any changes made to the database are immediately recorded in non-volatile storage such as disk. This algorithm helps to ensure that data is not lost in the event of a system failure or crash.

The basic steps involved in the immediate update algorithm are as follows:

1. Any changes made to the database are immediately recorded in the database buffer cache.
2. The changes are then written to the transaction log in non-volatile storage, such as disk.
3. Once the changes have been written to the transaction log, they are considered durable and can be safely committed to the database.
4. The transaction is marked as complete in the transaction log.

If a system failure occurs before the changes have been written to the transaction log, the immediate update algorithm ensures that the changes

are not lost. When the system is restarted, the transaction log is used to recover the database to the point in time just before the failure occurred.

The immediate update algorithm is a simple and efficient technique for ensuring database recovery. However, it can impact performance as changes are immediately recorded to disk. As such, it is important to monitor the performance of the database to ensure that it is not adversely affected.



CHECK YOUR PROGRESS

Q2. Fill in the blanks:

(a) Deferred update is a technique used in database recovery to ensure _____ in non-catastrophic failures.

(b) In deferred update recovery, changes made to the database are not immediately written to disk, instead, they are kept in _____ until a commit statement is issued.

(c) To ensure data consistency when using deferred update, incomplete transactions must be _____ if a failure occurs.

(d) Immediate update of recovery is essential to minimize the impact of _____ failures and to ensure that the database system is available as soon as possible.

(e) After restoring the system to normal operations, it is important to perform a _____ analysis to identify the root cause of the failure and implement corrective actions.

UNDO/NO-REDO recovery algorithm

The UNDO/NO-REDO recovery algorithm is a technique used in database systems to recover from a system failure or crash. It is a two-phase recovery algorithm that involves undoing uncommitted transactions and then skipping the redo phase.

The basic steps involved in the UNDO/NO-REDO recovery algorithm are as follows:

- Identify all transactions that were active at the time of the system failure.
- For each active transaction, undo any changes made by the transaction that were not committed to the database.
- Once all uncommitted changes have been undone, mark the transaction as aborted.
- Skip the redo phase and proceed directly to the transaction commit phase.

The UNDO phase of the recovery algorithm ensures that any uncommitted changes are rolled back, so that the database is left in a consistent state. The NO-REDO phase is possible because all committed transactions will have already written their changes to the database before the system failure occurred.

One advantage of the UNDO/NO-REDO recovery algorithm is that it can be faster than other recovery techniques that involve redoing all transactions. This is because the algorithm only processes uncommitted transactions, rather than all transactions.

However, one disadvantage of the UNDO/NO-REDO recovery algorithm is that it may lead to data loss if the system failure occurred during the commit phase. In this case, changes made by committed transactions will not be written to disk, and will be lost when the UNDO phase rolls back uncommitted changes.

Overall, the UNDO/NO-REDO recovery algorithm is a useful technique for recovering from system failures in database systems, but it should be used with caution to avoid data loss.

UNDO/REDO recovery algorithm

The UNDO/REDO recovery algorithm is a technique used in database systems to recover from a system failure or crash. It is a two-phase recovery algorithm that involves undoing uncommitted transactions and then redoing committed transactions.

The basic steps involved in the UNDO/REDO recovery algorithm are as follows:

- Identify all transactions that were active at the time of the system failure.
- For each active transaction, undo any changes made by the transaction that were not committed to the database.
- Once all uncommitted changes have been undone, mark the transaction as aborted.
- Redo all committed transactions, starting from the oldest transaction, by applying their changes to the database.

The UNDO phase of the recovery algorithm ensures that any uncommitted changes are rolled back, so that the database is left in a consistent state. The REDO phase is necessary because changes made by committed transactions may not have been written to disk before the system failure occurred.

One advantage of the UNDO/REDO recovery algorithm is that it provides a high degree of data durability and consistency, since all committed changes are applied to the database during the REDO phase. This means that no data is lost and the database is left in a consistent state.

However, one disadvantage of the UNDO/REDO recovery algorithm is that it can be slow, especially when there are a large number of transactions to be redone. This is because the algorithm needs to scan the transaction log to identify committed transactions that need to be redone.

Overall, the UNDO/REDO recovery algorithm is a useful technique for recovering from system failures in database systems, especially when data durability and consistency are important. However, the performance impact of the algorithm should be carefully considered when designing and tuning database systems.



CHECK YOUR PROGRESS

Q3. Fill in the blanks:

(a) The UNDO/NO-REDO recovery algorithm is a _____ technique used in database systems.

(b) In the UNDO/NO-REDO recovery algorithm, the UNDO phase ensures that _____ changes are rolled back.

(c) The NO-REDO phase in the UNDO/NO-REDO recovery algorithm is possible because all _____ transactions have already written their changes to the database.

(d) One advantage of the UNDO/REDO recovery algorithm is that it provides a high degree of data _____ and consistency.

(e) The performance impact of the UNDO/REDO recovery algorithm should be carefully considered when designing and tuning _____ systems.



13.7 LET US SUM UP

- Database failures can result in data loss, downtime, security breaches, and system crashes, which can negatively impact an organization's operations and revenue.

- To prevent database failures, it's important to have regular backups, implement redundancy and failover mechanisms, monitor system

performance, and perform regular maintenance and updates.

- Database recovery involves identifying the failure, restoring from backups or replaying transaction logs, testing and validating the system, and performing a post-mortem analysis to prevent future failures.
- A disaster recovery plan is important to have in place to quickly restore the database system to its previous state in case of a catastrophic failure.
- Best practices for backing up a database for catastrophic failure include scheduling regular backups, using multiple backup methods, storing backups offsite, testing backup and recovery procedures, and maintaining backups for an appropriate duration.
- Non-catastrophic failures in databases include software bugs, human errors, and network disruptions. To recover from such failures, one needs to identify the root cause of the failure, rollback or restore the database, reapply changes, verify the recovery, and take steps to prevent future failures.
- Deferred update is a technique used in database recovery to ensure data consistency in non-catastrophic failures. It involves keeping changes made to the database in memory until a commit statement is issued to indicate that the transaction is complete and all changes should be permanently saved.
- Immediate update algorithm is another recovery technique used in database systems to ensure that any changes made to the database are immediately recorded in non-volatile storage. This algorithm helps to ensure that data is not lost in the event of a system failure or crash.
- To minimize the impact of non-catastrophic failures, one should quickly identify the root cause of the failure, determine its severity, implement the appropriate recovery action, communicate with stakeholders, and

perform a post-mortem analysis to identify the root cause of the failure and implement corrective actions to prevent similar failures in the future.



13.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1. (a) (ii) (b) (ii) (c) (iii) (d) (iii) (e) (ii)

Ans. to Q. No. 2. (a) data consistency (b) memory (c) rolled back

(d) non-catastrophic (e) post-mortem

Ans. to Q. No. 3. (a) recovery (b) uncommitted (c) committed

(d) durability (e) database



13.9 FURTHER READING

1) Elmasri, R. & Navathe, S. B. (2015). Fundamentals of Database Systems. Pearson.

2) Ramakrishnan, R. & Gehrke, J. (2000). Database Management Systems. McGraw Hill.

3) Singh, S. K. (2011). Database Systems: Concepts, Design and Applications. Pearson Education India.



13.10 MODEL QUESTIONS

Q.1: What are the common causes of database failure, and what are the potential consequences of such failures?

Q.2: What steps can be taken to prevent database failures, and what measures should be taken in the event of a failure?

Q.3: What is the process of database recovery, and what are the key steps involved in this process?

Q.4: Why is it important to schedule regular backups of a database, and what are some best practices for backing up a database in the event of a catastrophic failure?

Q.5: What is a disaster recovery plan, and how can it help organizations prepare for and respond to database failures?

Q.6: What is a non-catastrophic failure in a database system?

Q.7: What are some key considerations when using deferred update for database recovery?

Q.8: What are some best practices for immediate update recovery for non-catastrophic failures?

Q.9: How does the immediate update algorithm ensure data recovery in the event of a system failure or crash?

Q.10: What steps should be taken to prevent future failures after recovering from a non-catastrophic failure in a database system?

Q.11: What is the difference between the UNDO/NO-REDO and UNDO/REDO recovery algorithms in database systems? What are the advantages and disadvantages of each approach?

Q.12: How does the UNDO/NO-REDO recovery algorithm work in database systems? What are the steps involved and how does it ensure data consistency and prevent data loss?

Q.13: When would you choose to use the UNDO/REDO recovery algorithm over the UNDO/NO-REDO recovery algorithm in a database system? What factors should you consider when making this decision?

*** ***** ***

UNIT 14: DATABASE SECURITY

UNIT STRUCTURE

14.1 Learning Objectives
14.2 Introduction
14.3 Database Security
14.4 Access Control
14.4.1 Discretionary Access Control
14.4.2 Mandatory Access Control
14.5 Statistical Databases
14.6 Data Encryption
14.7 Cryptanalysis
14.8 Let Us Sum Up
14.9 Further Reading
14.10 Answers to Check Your Progress
14.11 Model Questions

14.1 LEARNING OBJECTIVES

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

- learn about database security
- describe the need of database security
- define access control
- explain various data encryption methods

14.2 INTRODUCTION

In the previous unit we have discussed various types of database failures and recovery mechanisms. In this unit, we aregoing to discuss the techniques for making a database secure from variety of threats. We will also discuss the process of granting access privileges to authorized users at the end of this unit.

14.3 DATABASE SECURITY

The basic security goals for any database are:

Confidentiality: Database confidentiality refers to the protection of data from unauthorized disclosure.

Integrity: Database integrity refers to the requirement that information be protected from improper modification.

Availability: Resources should be available to human user or a program having legitimate right, that is what database availability means.
There exist some threats that may lead to degrade or loss in achieving these security goals. Therefore, to protect the database, different control measures are used, following are some of them:

- Access control
- Inference control
- Flow control
- Encryption

14.4 ACCESS CONTROL

The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function, called access control, is handled by creating user accounts and passwords to control the login process by the DBMS.

14.4.1 Discretionary Access Control

The typical method of enforcing discretionary access control in a database system is based on the granting and revoking of privileges. This is an all-or-nothing method: A user either has or does not have a certain privilege.

Types of Discretionary Privileges:
The account level: At this level, the DBA specifies the particular privileges that each account holds independently of the relations in

the database.

The relation level: At this level, the DBA has the privilege to control the access of each individual relation or view in the database.

Revoking of Privileges:

In some cases it is desirable to grant a privilege to a user temporarily. Hence, a mechanism for revoking privileges is needed. For the purpose of cancelling privileges given, one have to use REVOKE command in SQL.

14.4.2 Mandatory Access Control

In many applications, an additional security policy is needed that classifies data and users based on security classes. This approach, known as mandatory access control (MAC), would typically be combined with the discretionary access control mechanisms. Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest.

Two main restrictions are imposed on data access depending upon the subject or object classifications:

1. A subject S is not allowed read access to an object O unless class(S) e” class(O). This is known as the simple security property.
2. A subject S is not allowed to write an object O unless class(S) d” class(O). This is known as the star property (or *-property).

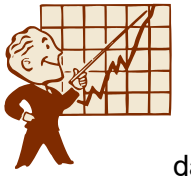
14.5 STATISTICAL DATABASES

Statistical databases are used mainly to produce statistics about various populations. The database may contain confidential data about individuals, which should be protected from user access. However, users are permitted to retrieve statistical information about the populations, such as averages, sums, counts, maximums, minimums, and standard deviations.

A population is a set of tuples of a relation (table) that satisfy some selection condition. Statistical queries involve applying statistical functions to a population of tuples. Security techniques used in statistical database

must have to protect the confidentiality of individual information.

This can be achieved by prohibiting queries that retrieve attribute values and by allowing queries that involve only statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION. Such queries are sometimes called statistical queries.



CHECK YOUR PROGRESS

Q1. What are the basic security goals for any database?

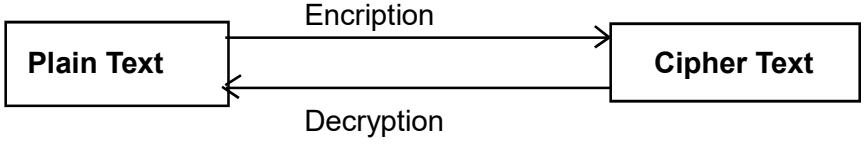
Q2. How to protect databases from threats?

Q3. What is the use of Statistical Database?

Q4. What is a population?

14.6 DATA ENCRYPTION

Encryption is the conversion of data into a form, called a cipher text, which cannot be easily understood by unauthorized persons. It enhances security and privacy when access controls are bypassed, because in cases of data loss or theft, encrypted data cannot be easily understood by unauthorized persons.



A message is plaintext. The process of disguising a message in such a way as to hide its substance is encryption. An encrypted message is ciphertext. The process of turning ciphertext back into plaintext is decryption. Mathematically,

$$E(M) = C; D(C) = M$$

where M is the message , C is the ciphertext , E is the encryption process and D is the decryption process.

A cryptographic algorithm, also called a cipher, is the mathematical function used for encryption and decryption. Keys are used for encryption and decryption process.

There are two general types of key-based algorithms: *symmetric* and *public-*

key.

Symmetric key algorithms use the same key for both the encryption and the decryption process. The key is kept secret.

$$E_k(M) = C;\ D_k(C) = M$$

In public key cryptography different keys are used for encryption and decryption process. A pair of keys is generated; out of which one key is made available to all, while the other is kept secret.



CHECK YOUR PROGRESS

Q5. What is data Encryption?

Q6. Explain public key cryptography.

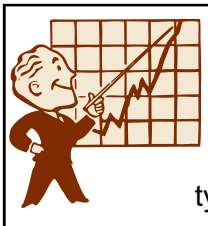
14.7 CRYPTANALYSIS

Cryptanalysis is the science of extracting the plaintext of a cipher message without knowing the key. It shows the vulnerability of a cryptosystem. An attempted cryptanalysis is called an attack.

There are four general types of cryptanalytic attacks, assuming that the cryptanalyst has complete knowledge of the encryption algorithm used:

- Ciphertext-only attack.
- Known-plaintext attack
- Chosen-plaintext attack
- Adaptive-chosen-plaintext attack

Good cryptosystems are designed to be infeasible to break with the computing power that is expected to evolve many years in the future.



CHECK YOUR PROGRESS

Q7. What is Cryptanalysis? How many different types of Cryptanalysis attack are possible?



14.8 LET US SUM UP

- Confidentiality, Integrity and availability of a database need to be maintained.
- To protect a database from threat different control measures are taken such as access control, inference control, flow control and encryption.
- The idea of secret key and public key are discussed in detail.
- Finally we have discussed about vulnerability of a cryptosystem with various possible types of attack.



14.9 FURTHER READING

1) Elmasri R. and Navathe B, “Fundamentals of Database Systems”.
2) Ramakrishnan R., “Database Management Systems”.
3) Schneier B., “Applied Cryptography”.



14.10 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q1. The basic security goals for any database are:

- Confidentiality
- Integrity
- Availability

2. For protecting databases four different control measures are taken:
   - Access Control
   - Inference Control
   - Flow Control
   - Encryption.

3. Statistical databases are used mainly to produce statistics about various populations. Statistical information about the populations, such as averages, sums, counts, maximums, minimums, and standard deviations are furnished keeping personal information hidden.

4. A population is a set of tuples of a relation (table) that satisfy some selection condition.

5. Encryption is the conversion of data into a form, called a cipher text, which cannot be easily understood by unauthorized persons. It enhances security and privacy when access controls are bypassed, because in cases of data loss or theft, encrypted data cannot be easily understood by unauthorized persons.

6. In public key cryptography different keys are used for encryption and decryption process. A pair of keys is generated; out of which one key is made available to all, while the other is kept secret.

7. Cryptanalysis is the science of extracting the plaintext of a cipher message without knowing the key.
   Four different attacks are possible:.
   - Ciphertext-only attack.
   - Known-plaintext attack
   - Chosen-plaintext attack
   - Adaptive-chosen-plaintext attack



14.11 MODEL QUESTIONS

Q1. Discuss what is meant by each of the following terms: database authorization, access control and data encryption.

Q2. What is a statistical database? Discuss the problem of statistical database security.

Q3. How is privacy related to statistical database security? What measures can be taken to ensure some degree of privacy in statistical databases?

Q4. What is flow control as a security measure? What types of flow control exist ?

Q5. How can privacy of data be preserved in a database?

Q6. What are some of the current outstanding challenges for database security?

Q7. Explain RSA algorithm.

Q8. What is a symmetric key algorithm for key-based security?

Q9. What is the public key infrastructure scheme? How does it provide security?

*****