PGCA/Sc IT/PGDCA (S2) 03
Exam. Code: FDMS

Fundamentals of Database Management Systems

SEMESTER II

MASTER OF COMPUTER APPLICATION / MASTER OF SCIENCE IN INFORMATION TECHNOLOGY / POST GRADUATE DIPLOMA IN COMPUTER APPLICATION

Block 1



KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta, Gauhati University
Prof. (Retd.) Pranhari Talukdar, Gauhati University
Prof. Sikhar Kumar Sarma, Cotton University
Dr. Jyotiprokash Goswami, Assam Engineering College

Course Co-ordinators

Dr. Tapashi Kashyap Das, KKHSOU
Ms. Sruti Sruba Bharali, KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
1 & 5	Dr. Tapashi Kashyap Das, KKHSOU
2	Mr. Arabinda Saikia, KKHSOU
3	Mr. Aniruddha Deka, Royal Global University
4	Dr. Sangeeta Kakoty, KKHSOU
6	Mr. Biswajit Das, Cotton University
7 & 8	Ms. Sruti Sruba Bharali, KKHSOU

Editorial Team

Content : Dr. Tapashi Kashyap Das, KKHSOU (Units 2, 3, 4, 6 7 & 8)
Ms. Sruti Sruba Bharali, KKHSOU (Units 1 & 5)

Language : Prof. (Retd.) Robin Goswami, Cotton College

Structure, Format & Graphics: Dr. Tapashi Kashyap Das, KKHSOU

April, 2019 ISBN: 978-93-89123-02-9

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State University is made available under a Creative Commons Attribution-Non Commercial-ShareAlike4.0 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 Office : Housefed Complex, Dispur, Guwahati-781 006; Web: www.kkhsou.in

The university acknowledges with thanks the financial support provided by the **Distance Education Bureau, UGC** for the preparation of this study material.

MASTER OF COMPUTER APPLICATION /
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY /
POST GRADUATE DIPLOMA IN COMPUTER APPLICATION

Fundamentals of Database Management Systems

Block-1

DETAILED SYLLABUS

		Pages
UNIT 1:	**File Structure and Organization** 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	**7-18**
UNIT 2:	**Database Management System** Definition of DBMS, File Processing System Vs. DBMS, Advantages and Disadvantages of DBMS, Database Architecture, Data Independence, Data Dictionary, DBMS Language, Database Administrator	**19-39**
UNIT 3:	**Data Models** Data Models: Object Based Logical Model, Record Base Logical Model, Relational Model, Network Model, Hierarchical Model	**40-50**
UNIT 4:	**Entity-Relationship Model** Entity Set, Attribute, Relationship Set, Entity Relationship Diagram (ERD), Extended Features of ERD	**51-67**
UNIT 5:	**Relational Databases** Relational Data Model; Terms: Relation, Tuple, Attribute, Cardinality, Degree, Domain; Keys: Super Key, Candidate Key, Primary Key, Foreign Key	**68-83**
UNIT 6:	**Relational Algebra** Operations: Select, Project, Union, Difference, Intersection, Cartesian Product, Natural Join	**84-94**

UNIT 7: Introduction to SQL **95-126**

Structured Query Language (SQL), Characteristics of SQL, Advantages of SQL, SQL Data Types, Types of SQL Commands, DDL, DML, Creating a Database, Creating, Altering, and Deleting Tables; Inserting New Data, Updating Data (the WHERE Clause, The Logical AND and OR Operator), Deleting Data

UNIT 8: Advanced SQL Commands **127-158**

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, 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)

COURSE INTRODUCTION

This is a course on ***Fundamentals of Database Management System***. Data and its efficient management comprise one of the most critical business objectives of an organization. Database system is a tool that simplifies the task of managing the data and extracting useful information in a timely fashion. With the rapid growth in computing technology and its appication in modern society, databases have become an integral component of our everyday life. This course introduces the basic concept of database and database management system (DBMS) and also reviews the goals of DBMS, types of data models. etc.

The course consists of two blocks.

Block 1 deals with the fundamental terms and concepts like definition of database, database management system and its advantages and disadvantages. It also discussses types of data models, E-R model, Relational database and relational algebra, concept of keys, etc. At the end, this block introduces Structured Query Language (SQL) and commands of SQL.

Block 2 discusses database design and normalization at the beginning. It also discusses transaction processing, concurrency control, databse recovery, security and authentication.

BLOCK INTRODUCTION

This is the *first block* of the course ***Fundamentals of Database Management System***. This block contains eight units.

Unit 1 describes file structure and its organization. It defines various basic terms like fields, records along with their types. Various operations associated with files are also discussed in this unit.

Unit 2 discusses database management system. This unit discusses different important concepts of database and DBMS. Its describes database architecture, DBMS languages etc.

Unit 3 is on data models. Different types of data models with their features are described in this unit.

Unit 4 is on Entity-Relationship model. It describes entity, entity sets, attributes, ER diagram etc.

Unit 5 descibes relational database besides concept of keys.

Unit 6 deals with relational algebra.

Unit 7 and **Unit 8** introduce Structured Query Language (SQL) and the various commands of SQL.

Each unit of these blocks 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 additional interesting and relevant information. Again, you will get “CHECK YOUR PROGRESS” questions. These have been designed 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 AND ORGANIZATION

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 Further Reading

1.9 Answers to Check Your Progress

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 the prerequisite for us 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 the name of a person, name of a place, the roll number of a student, the ticket number of a passenger, the 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 useful 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.

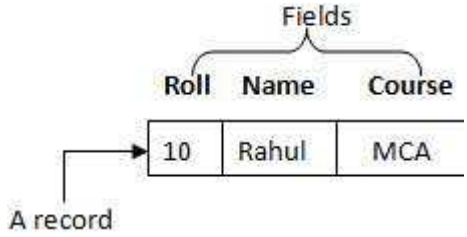


Figure 1.1: An example of Record

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 is 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 (19 characters)

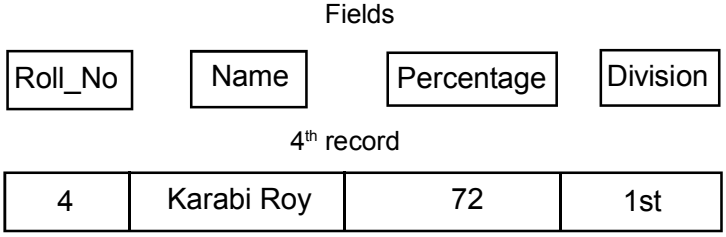
In the above record, the respective fields are Roll_No, Name, Percentage and Disvision. This record occupies 19 characters including the blank spaces and commas.

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***.



File

Roll No.	Name	Percentage	Division
1	Gautam Baruah	72	1st
2	Pritam Kashyap	68	1st
3	Rajib Sharma	44	3rd
4	Karabi Roy	72	1st
5	Debojit Saikia	81	1st
6	Bhavna Das	60	1st
7	Sweta Mishra	55	2nd
8	Manoj Bora	76	1st
9	Pritam Kashyap	83	1st
10	Mriganka Bharali	42	3rd

Field content

Figure 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 permanent in nature.

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 in 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



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

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:

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

Q.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 constitute a file.

iv) A file needed for updating a master files 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 file 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 FURTHER READING

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



1.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) True, ii) True, iii) False, iv) True, v) False, vi) True, vii) True, viii) False, ix) True, x) False



1.10 MODEL QUESTIONS

Q.1: Define Data and Information.

Q.2: Explain the following terms with example: Field, Record, File.

Q.3: What are the different types of file organization techniques? Explain.

Q.4: Write short notes on:

a) Direct access organization

b) Sequential access organization

c) Index-Sequential organization

Q.5: What are the advantages and disadvantages of sequential access organization?

Q.6: What are the advantages and disadvantages of indexed-sequential access organization?

*** ***** ***

UNIT 2: DATABASE MANAGEMENT SYSTEM

UNIT STRUCTURE

2.1 Learning Objectives

2.2 Introduction

2.3 File based Approach

2.4 Database Approach

2.4.1 Advantage of Database

2.5 Database Management System

2.5.1 Components of DBMS

2.5.2 Advantages of DBMS

2.5.3 Disadvantages of DBMS

2.6 Database Architecture

2.7 Data Independence

2.8 Data Dictionary

2.9 DBMS Language

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 database and database management system
- describe the architecture of database mamanment system
- illustrate data independence and data dictionary
- learn appropriate languages to express database queries and updates
- describe the role of database administrator.

2.2 INTRODUCTION

In the previous unit, we have learnt about the basic idea of data, information, fields and records. In addition to this, concepts of files and basic

file organization techniques are also introduced. All these are the elementary concepts relating to the database system.

In this unit, we will learn the concept of *Database* and *Database management System*. We will also discuss the three-tier database architecture and different types of DBMS languages.

Moreover, responsibilitties of database administrator will also be discussed at the end of this unit.

2.3 FILE BASED APPROACH

In case of traditional file based approach, an organization's information was stored as group of records in separate files. These file processing systems consist of data files and many application programs. Each file, termed as a *flat file,* contains and processes information for one specific function, such as accounting or inventory. Programming languages such as COBOL were used by the programmers to write application programs. Each of the application files and programs was created and maintained independent of another 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.) is stored in different format in the above applications
- whenever some information regarding a group of students are 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 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.

In the following figure (Figure 2.1), a traditional record keeping system of a University is shown, where every application is interrelated and causes repetation of the same data in different files this leads to the problem of data redundancy and inconsistency.

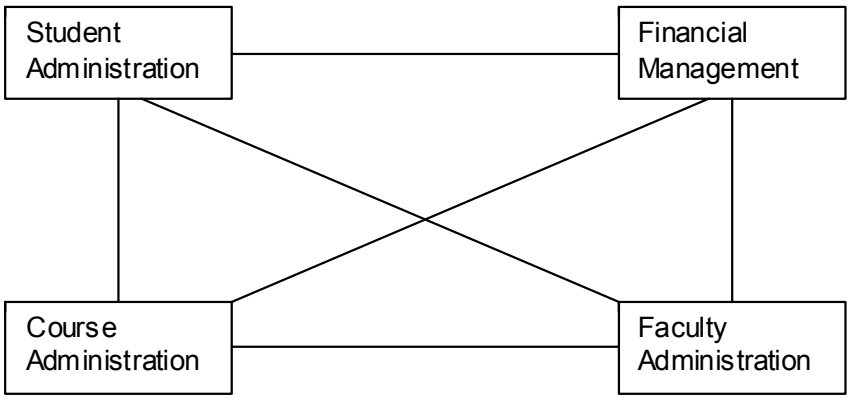


Figure 2.1: Traditional File Approach

Some of the disadvantages of file based approach are discussed below:

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 in more than one file is known as data redundancy. This leads to increase in cost of data entry and data storage.

Data integrity problem: Data integrity refers to 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 were described by specific physical formats that were coded into the application program by programmer. If the format of a certain record was 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.

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 database approach the user is free from the detailed and complicated task of keeping up with the physical structure of data. A database approach is shown by the following diagram (Figure 2.2).

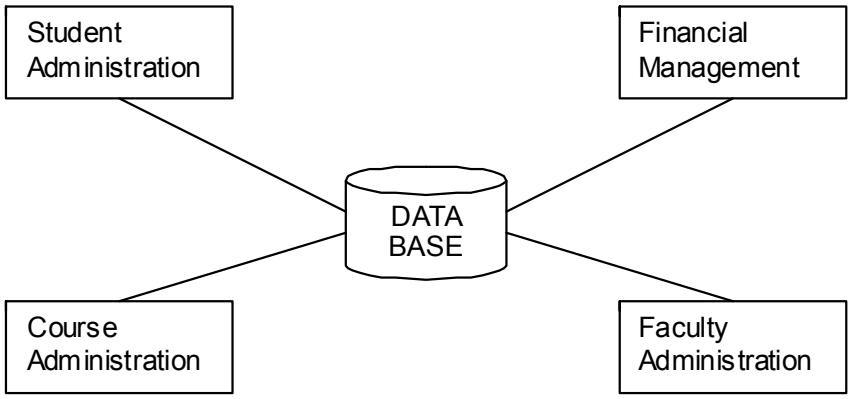


Figure 2.2: Database Approach

Here, it is shown that several applications share common data in a database approach.

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, it

- is a collection of interrelated data stored together without harmflul or unnecessary 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.

2.4.1 Advantage of Database Approach

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 in 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 to each other. However, the database approach provides an independence between the file structure and program structure.
- **Sharing of data and Data security:** Data in a database are shared among users and applications. In database approach data are protected from unauthorised access by some security mechanism.

2.5 DATABASE MANAGEMENT SYSTEM

The ***database management system****(DBMS)* is the interface between the users(application programmers) and the database(the data). A database

management system 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 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*.

2.5.1 Components of DBMS

The functional components of DBMS are:

- *Storage manager*
- *Query processor*

Storage Manager is responsible for storing, retrieving and updating data in database. Storage manager components are:

Authorization and integrity manager: Checks the integrity constraints and authority of users to access data.

Transaction manager: Transaction means a collection of operations that performs a single logical function in a database application. It ensures that the database remains in a consistent state if there is a transaction failure or system failure.

File manager: It is responsible for allocating space on disk storage. The files are used for storing collections of similar data. The file manager can create and delete file also update and retrieve records from files.

Buffer manager: The buffer is a temporary memory space from which a file is read. The buffer manager is responsible for fetching data from disk storage into main memory.

Query Processor: Query processing is a procedure of converting a query written in high level language (e.g. SQL, QBE) into a correct and efficient execution plan expressed in low level language, which is used for data manipulation. The success of a query language also depends upon its query processor.

2.5.2 Advantages of DBMS

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

Minimal data redundancy: Centralized control of data avoids the unnecessary duplication of data and effectively reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large storage of data.

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.

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.

Improved data sharing: Since, database system is a centralised repository of 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.

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

Data security: Database security is the protection of database from unauthorised users. The DBA(DataBase Administrator, we will discuss the responsibility of DBA in the next section) can define security rules to chueck unauthorised access to data. Some users may be given rights to only retrieve data, 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.

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 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.

Providing Backup and Recovery: A DBMS must provide the facilities for recovering from hardware or software failures. The backup 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.

2.5.3 Disadvantages of DBMS

Some of the disadvantages of DBMS are as follows:

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 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 serous consequences for an organization.

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.

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 many may not run as fast as they used to.

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.

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

Additional Hardware cost: The disk storage requirements for the DBMS and the database many 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.

Cost of Conversion: The cost of the DBMS and extra hardware may be significant compared with 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

Q.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

Q.2: Define database? Give a sutable example.

Q.3: Define Database Management System.

2.6 DATABASE ARCHITECTURE

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 users with an abstract view of the data. That is the system hides certain details of how the data is stored and maintained. 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 following figure (Figure 2.3.)

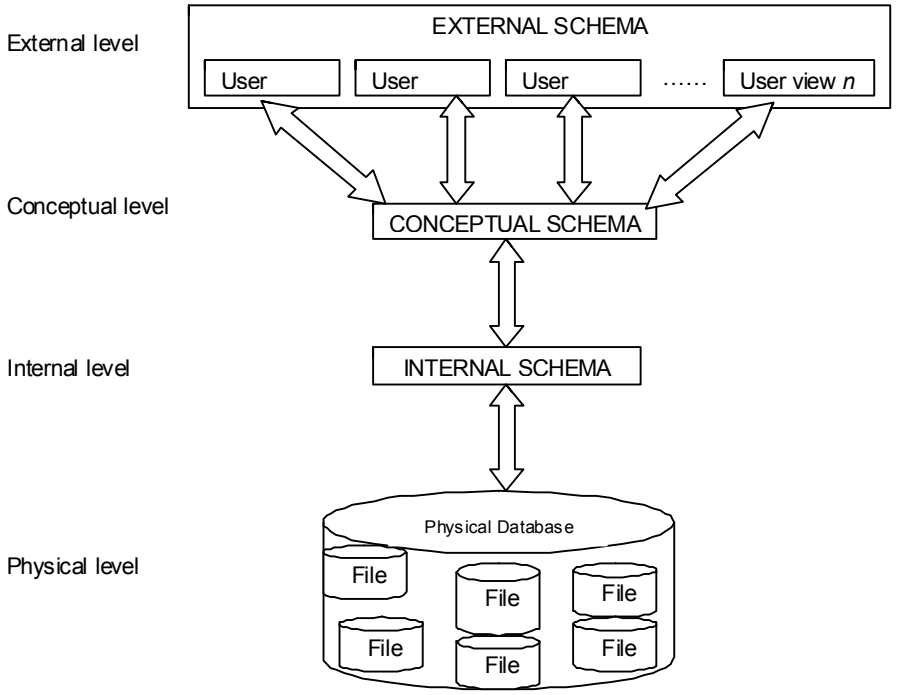


Figure 2.3: Three-tier Database Architecture

We can imagine that the whole database system is divided into levels. These are:

- 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 describes the different views of the database related to different users.

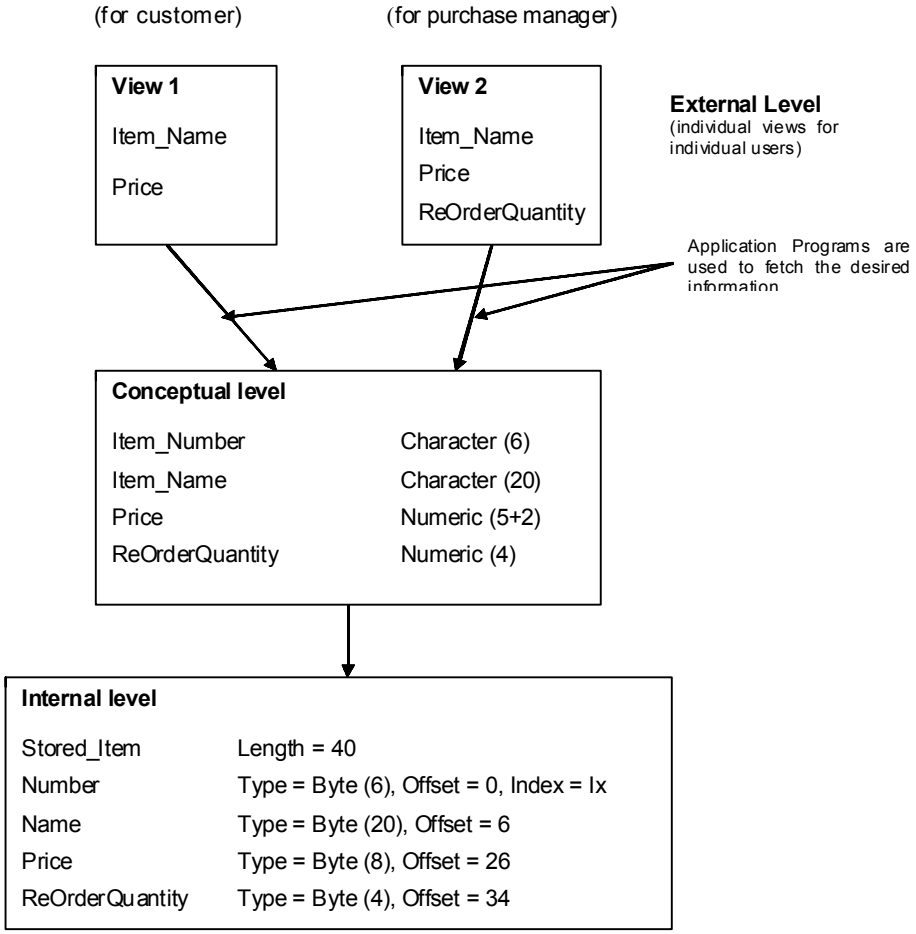


Figure 2.4: View of data at Three-tier Database Architecture

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 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 describe 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.

2.7 DATA INDEPENDENCE

Data independence is the characteristics 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–

- Logical data independence
- Physical 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 refers to the external schema construct must work as before.

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 DATA DICTIONARY

A *data dictionary* also known as a *system catalog i*s a centralized store of information about the database. It contains the information about the tables, the fields, the data types, primary keys, indexes, the joins which have been established between those tables, referential integrity, cascades update, cascade delete etc. We will come across with these terms in a later unit. The information stored in the data dictionary is called the ***metadata***. Thus, a data dictionary can be considered as a file that stores metadata. Data dictionary is a tool for recording and processing information about the data that an organization uses. The data dictionary is a central catalog for metadata. The data dictionary can be integrated within the DBMS or separate.

2.9 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 database, a DBMS is choosen to implement the database. It is important to first specify the conceptual and internal schemas for the database. Following languages are used for specifying database schemas:

- Data definition language (DDL)
- Storage definition language (SDL)
- View definition language (VDL)
- Data manipulation language (DML)

Data definition language (DDL): DDL is a special language which specify the database conceptual schema using set of definitions. DDL allows the DBA or 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 inorder 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 a EMPLOYEE table as shown below:

EMPLOYEE				
Fname	Lastname	Eno	DOB	Address

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.

View definition language (VDL): View definition language is used to specify user's views(external s chema) 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).

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.

2.10 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 batabase 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 the 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 most 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

Q.4: 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 schemas.

vi) A database is a computer-based record keeping system whose overall purpose is to record and maintain information.

Q.5: Multiple Choice:

a) A view of database that appears 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

Q.6: What is meant by metadata?

Q.7: Define the term data dictionary.

Q.8: what is meant by physical and logical data independnce?

Q.9: Define the concept of database schema? Write the names of the schemas that exists in a database complying with the three levels of ANSI/SPARC architecture.



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 provide a 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.
- A data dictionary also known as a system catalog is a centralized store of information about the database.
- DBMS provide appropriate languages and interfaces for each category of users to express database queries and updates.
- Following laguages are used for specifying database schemas:
  - Data definition language (DDL)
  - Storage definition language (SDL)
  - View definition language (VDL)
  - Data manipulation language (DML)

- A DBA provides the necessary technical support for implementing policy decisions of database.



2.13 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.
3) Ullman, J. D. (1984). *Principles of Database Systems*. Galgotia Publications.



2.14 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) (iv), b) (iv), c) (ii), d) (i), e) (i)

Ans. to Q. No. 2: A database is a collection of related data. Here, the term 'data' means that known facts that can be record. Examples of database are library information system, railway, bus or airline reservation system etc.

Ans. to Q. No. 3: DBMS is a collection of programs that enables users to create and maintain a database.

Ans. to Q. No. 4: i) F, ii) T, iii) F, iv) F, v) T, vi) T

Ans. to Q. No. 5: a) (ii), b) (i), c) (ii), d) (ii), e) (i)

Ans. to Q. No. 6: Metadata are data about the data but not the actual data.

Ans. to Q. No. 7: Data dictionary is a file that contains metadata.

Ans. to Q. No. 8: In logical data independence, the conceptual schema can be changed without changing the external schema. In physical data independence, the internal schema can be changed without changing the conceptual schema.

Ans. to Q. No. 9: Database schema is nothing but description of the database. The types of schemas that exist in a database complying with three levels of ANSI/SPARC architecture are : *external schema, conceptual schema and internal schema.*



2.15 MODEL QUESTIONS

Q.1: What is file based approach of database? Explain its limitations?

Q.2: Explain the three level database architecture. What are its objectives?

Q.3: What is data independence and what are its types? How is data indep-endence achieved?

Q.4: What are advantages of DBMS?

Q.5: Discuss the main disadvantages of a Traditional file approach?

Q.6: Discuss the main disadvantages of DBMS?

Q.7: Differentiate between DBMS approach & traditional file approach.

Q.8: Mention the differences between text files and database files. Why are database files preferred in a commercial organization?

Q.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

Q.10: What is logical data independence and why is it important?

Q.11: Explain the difference between logical and physical data independence.

Q.12: Describe the three levels of data abstraction?

Q.13: What do you mean Database Language? What are the different types of data base language?

Q.14: Explain the role of a Database Administrator.

*** ***** ***

UNIT 3: DATA MODELS

UNIT STRUCTURE

3.1 Learning Objectives

3.2 Introduction

3.3 Data Models

- 3.3.1 Object-Based Logical Model
- 3.3.2 Record-Based Logical Model
- 3.3.3 Relational Model
- 3.3.4 Network Model
- 3.3.5 Hierarchical Model

3.4 Let Us Sum Up

3.5 Further Reading

3.6 Answers to Check Your Progress

3.7 Model Questions

3.1 LEARNING OBJECTIVES

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

- learn about data model and why data models are important
- describe different types of data models like Object based data model, Record based data model, Relational data model, etc. and their advantages and disadvantages.

3.2 INTRODUCTION

In the previous unit we have acquainted with database management system and various concepts associated with database management system as well as database architecture.

In this unit, we will be able to learn about data models. There are many basic structures that exist in a database system to organize 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. We will discuss various data models with their characteristics.

3.3 DATA MODELS

A *data model* is a set of concept that can be used to describe the structure of a database. A collection of conceptual tools for describing data, data relationship, data semantics and consistency constraints is called ***data models***. It shows the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. Individual database models are designed based on the rules and concepts of whichever broader data model the designers adopt. Most data models can be represented by an accompanying database diagram. There are many kinds of data models. Some of the most common ones include:

- Hierarchical database model
- Relational model
- Network model
- Object-oriented database model
- Entity-Relationship model
- Document model
- Entity-attribute-value model
- Star schema
- The object-relational model, which combines the two that make up its name etc.

  Data models are important because they:
- are a communication tool
- give an overall view of the database Organize data for various users
- are an abstraction for the creation of good database.

3.3.1 Object-Based Logical Model

Object based logical models are used in describing data at the logical and view levels. They are characterized by the fact that they provide fairly flexible structuring capabilities and allow data constrains to be specified explicitly. This hybrid database model combines the simplicity of the relational model with some of the advanced functionality of the object-oriented database model. In essence, it allows designers to incorporate objects into the familiar

table structure. Languages and call interfaces include SQL3, vendor languages, ODBC, JDBC, and proprietary call interfaces that are extensions of the languages and interfaces used by the relational model. There are many different models and more are likely to come. Several of the more widely known ones are:

- **The Entity-Relationship (E-R) Model:** The Entity-Relationship data model is a logical data model, which has a logical representation of data for an enterprise of business establishment. It is based on a perception of a real world that consists of collection of basic objects, called *entities* and of *relationships* among these objects. This model captures the relationships between real-world entities, but it is not as directly tied to the physical structure of the database. Instead, it is often used for designing a database conceptually. Here, the people, places, and things about which data points are stored are referred to as entities, each of which has certain *attributes* that together make up their *domain*. The concept of E-R model and the terms associated with this model will be discussed in details in the next unit (*Unit 4*) of this block.

  Advantages of E-R Model:

  - Visual modeling yields conceptual simplicity
  - Graphical representation makes it an effective communication tool for better understanding.
  - Easy conversion for E-R to other data model.

  Disadvantages of E-R Model:

  - There is no industry standard notation for developing an E-R diagram
  - The E-R data model is especially popular for high-level database design.

- **The Object-Oriented Model:** Like the Entity-Relationship model, the object-oriented model is based on a collection of objects. An object contains values within the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. This model defines a database as a collection

of objects, or reusable software elements, with associated features and methods. There are several kinds of object-oriented databases:

- A multimedia database incorporates media, such as images, that could not be stored in a relational database.
- A hypertext database allows any object to link to any other object. It is useful for organizing lots of disparate data, but it is not ideal for numerical analysis.

The object-oriented database model is the best known post-relational database model, since it incorporates tables, but is not limited to tables. Such models are also known as hybrid database models.

Advantages of Object-Oriented Model:

- Semantic content is added.
- Visual representation includes semantic content.
- Inheritance promotes data integrity.

Disadvantages of Object-Oriented Model:

- Slow development of standards causing vendors to supply their own enhancements.
- Widely accepted standard being compromised.
- Complex navigational system.
- High system overhead slowing transactions.

3.3.2 Record-Based Logical Models

Record based logical models (also termed as logical model) are used for describing data at the logical and view levels. In contrast to object-based data models, they are used both to specify the overall logical structure of the database and to provide a higher-level description of the implementation. Record based data models are so named because the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length.

3.3.3 Relational Model

E. F. Codd of IBM Research centre first introduced the relational data model in a paper in 1970. Relational model stores data into tables, also known as relations, each of which consists of columns and rows. Each column lists an attribute of the entity in question, such as price, zip code, or birth date. Together, the attributes in a relation are called a domain. A particular attribute or combination of attributes is chosen as a primary key that can be referred to in other tables, when it's called a foreign key. Each row, also called a tuple, includes data about a specific instance of the entity in question, such as a particular employee.

The model also accounts for the types of relationships between those tables, including one-to-one, one-to-many, and many-to-many relationships.

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns and each column has a unique name. An example of tabular data in the relational model is shown in the following tables.

Table 3.1: Customer Table

Customer_ID	Customer_Name	Address	Mobile-No	Acc_No
142-12-1909	Antara Malakar	Beltola	9098787676	A-101
165-23-1787	Manoj Sarma	Jalukbari	7098987876	A-453
132-87-1278	Chandan Kalita	Dharapur	9678067656	A-409
142-12-1909	Antara Malakar	Beltola	9098787676	A-289
132-87-1278	Chandan Kalita	Dharapur	9678067656	A-867

Table 3.2: Account Table

Customer_ID	Balance
A-101	8000
A-453	13000
A-409	6700
A-289	21000
A-867	9600

Table 3.3: Depositor Table

Customer_ID	AccNo
142-12-1909	A-101
165-23-1787	A-453
132-87-1278	A-409
142-12-1909	A-289
132-87-1278	A-867

Advantages of Relational Model:

- Relational model is a simple data model as it frees the designers from the actual physical data storage details, allowing them to concentrate on the logical view of the database.
- Structural independence is promoted using independent tables. Changes in the data structure do not affect the data access.
- Tabular view improves conceptual simplicity.
- Isolates the end user from physical-level details
- Ease of design, implementation, maintenance and uses.
- It provides flexible and powerful query capability with the use of Structured Query Language (SQL).

Disadvantages of Relational Model:

- Requires substantial hardware and system software overheads.
- Conceptual simplicity gives untrained people the tools to use a good system poorly. With the growth of database, the poor design results into slower system, degraded performance and data corruption.

3.3.4 Network Model

Data in the network model are represented by collections of records and relationship among data is represented by links, which can be viewed as pointers. The records in the database are organized as collections of arbitrary graphs. The network model builds on the hierarchical model by allowing many-to-many relationships between linked records, implying multiple parent records. Based on

mathematical set theory, the model is constructed with sets of related records. Each set consists of one owner or parent record and one or more member or child records. A record can be a member or child in multiple sets, allowing this model to convey complex relationships. It was most popular in the 70s after it was formally defined by the Conference on Data Systems Languages (CODASYL).

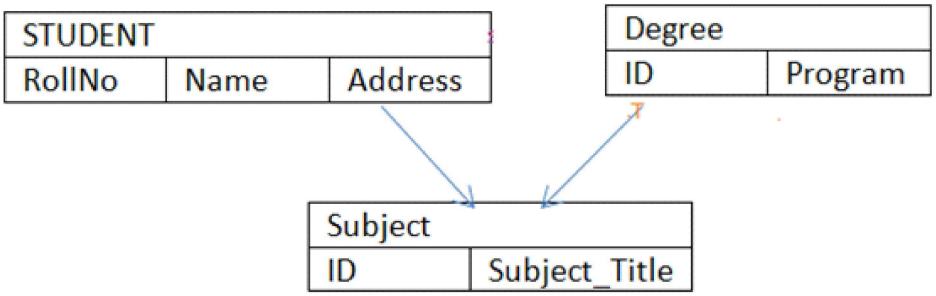


Figure 3.1: Example of a Network Model

Characteristics of Network Model:

- Supports many-to-many relationships.
- Entities are represented as network.
- One child entity can have more than one parent entity.
- Complex structure.
- High performance.
- Query facility is not available in network model.

Advantages of Network Model:

- Conceptual simplicity.
- Handles more relationship types.
- Data access is flexible.
- Data owner/member relationship promotes data integrity.
- Includes data definition language (DDL) and data manipulation language (DML).

Disadvantages Network Model:

- System complexity limits efficiency.
- Navigational system yields complex implementation, application development, and management.
- Structural changes require changes in all application programs.

3.3.5 Hierarchical Model

Hierarchical model is one of the oldest database models used by enterprise in the past. The hierarchical model is similar to the network model in the sense that data and relationships among data are represented by records and links respectively. The hierarchical data model is represented by an upside down tree. This model organizes data into a tree-like structure, where each record has a single parent or root. Sibling records are sorted in a particular order. That order is used as the physical order for storing the database. This model is good for describing many real-world relationships. For example, an educational environment can be represented as follows in figure 3.2.

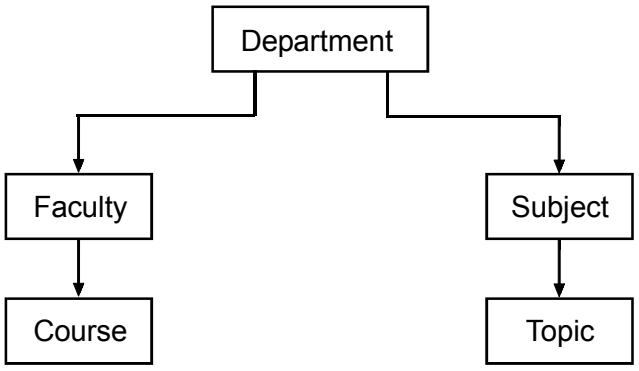


Figure 3.2: Example of a Hierarchical Model

In the diagram, Department is related to Faculty which is related to Course. Department is also related to Subject which is related to Topic. Here, we can see that Department is at the top of a hierarchy from which a large amount of information can be derived.

Characteristics of Hierarchical Model:

- In a hierarchical model, if a parent node is deleted, the child is also deleted automatically.
- Pointers are used to link the records. Pointer determines as to which record is parent record and which one is child record.
- Minimizes disk input and output.
- Due to the short distance between parent to child, database access time and performance is improved. Navigation through the database is very fast in a hierarchical model.

- All relationships are predefined. Root node, parents and child are predefined in the database schema.

Advantages of Hierarchical Model:

- Data sharing becomes practical as all data are held in a common database.
- Parent/child relationship promotes.
- Conceptual simplicity and data integrity.
- Database security is provided and enforced by DBMS.
- Efficient with 1 : M relationships.

Disadvantages of Hierarchical Model:

- Requires knowledge of physical data storage characteristics.
- Navigational system requires knowledge of hierarchical path.
- Changes in structure require changes in all application programs.
- No data definition.
- Lack of standards.



CHECK YOUR PROGRESS

Q1: Fill in the blanks:

i) Relational data model store data in the form of ..... or .....

ii) The ..... data model is represented by an upside down tree.

iii) ..... first introduced the relational data model.

iv) ..... model is a simple data model as it frees the designers from the actual physical data storage details.

Q.2: Define object based logical models?



3.4 LET US SUM UP

- A data model is a mechanism that provides the logical structure of a database. It defines how data is connected to each other and how they are processed and stored inside the system.

- Data modeling is used for representing entities of interest and their attributes. Hierarchical, Relational, Network, Object-oriented, Entity-relationship are the popular data model
- Object-based data models are used to describe data and its relationships. It uses concepts such as entities, attributes and relationships. It has flexible data structuring capabilities. Entity-Relationship (E-R) data model is one of the most widely used object-based data model.
- E-R model is used for designing of relational databases.
- E.F Codd introduced the relational data model. Relational model stores data into tables, also known as relations, each of which consists of columns and rows.
- A record-based data model is used to specify the overall logical structure of the database.
- Physical data models are used for a higher-level description of storage structure and access mechanism. They describe how data is stored in the computer, record structure and access path.
- Hierarchical data model is one of the oldest data models used by enterprise in the past. In hierarchical data model, the relationship between the files or records forms a hierarchy.
- Network data model is similar to hierarchical model except that a record can have multiple parents. The network data model has three basic components such as record type, data items or fields and links. The network model facilitates in handling of one-to-many(1:m) and many-to-many (n:m) relationships, which help in modeling the real life situation.



3.5 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Elmasri, R. & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.
3) Silberschatz, A., Korth, H. F. & Sudarshan, S. (1997). *Database System Concepts* (Vol. 4). New York: McGraw-Hill.

4) Singh, S. K. (2011). Database Systems: Concepts, Design and Applications. Pearson Education India.



3.6 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) Table, relation; ii) hierarchical; iii) E. F. Codd; iv) Relational

Ans. to Q. No. 2: Object-based logical models are used in describing data at the logical and view levels. They are characterized by the fact that they provide fairly flexible structuring capabilities and allow data constrains to be specified explicitly.



3.7 MODEL QUESTIONS

Q.1: Define data model? What is their importance?

Q.2: What are the data models in DBMS?

Q.3: What are the advantages and disadvantages of Entity-Relationship model?

Q.4: What are the advantages and disadvantages of Object-Oriented model?

Q.5: What are the advantages and disadvantages of relational model?

Q.6: Explain the characteristics of network model?

Q.7: Explain the characteristics of hierarchical model?

Q.8: What are the advantages and disadvantages of network model?

Q.9: What are the advantages and disadvantages of hierarchical model?

Q.10: Explain the record based logical model?

*** ***** ***

UNIT 4: ENTITY-RELATIONSHIP MODEL

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 Entity Relationship Model

- 4.3.1 Entity, Attribute and Relationship Type
- 4.3.2 Degree of Relationship Type
- 4.3.3 Constraints on Relationship Type
- 4.3.4 Weak and Strong Entities

4.4 Conversion of ER Model into Relational Schema

4.5 ER Modelling Symbols

4.6 Let Us Sum Up

4.7 Further Reading

4.8 Answer to Check Your Progress

4.9 Model Questions

4.1 LEARNING OBJECTIVES

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

- describe the entity relationship model
- identify the categories of data model that are used in database management system
- identify the symbols to describe the ER model
- design the ER diagram with its constraints
- learn about conversion procedure from ER diagram to relational databases and vice versa.

4.2 INTRODUCTION

In our previous units we have discussed 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. In this unit, we will discuss 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. With this we will discuss the basic conversion procedure of ER model into relational schema with suitable example. We will conclude this unit discussing ER modelling symbols.

4.3 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:

- High level conceptual data model.
- Allows describing the data involved in a real-world enterprise in terms of objects and their relationships.
- Widely used to develop an initial design of a database.
- Describes data as a collection of entities, relationships and attributes.

4.3.1 Entity, Attribute and Relationship Type

ER model uses three features to describe data– 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 4.1(a) and 4.1(b), two entity types EMPLOYEE and STUDENT are shown with their different attributes.

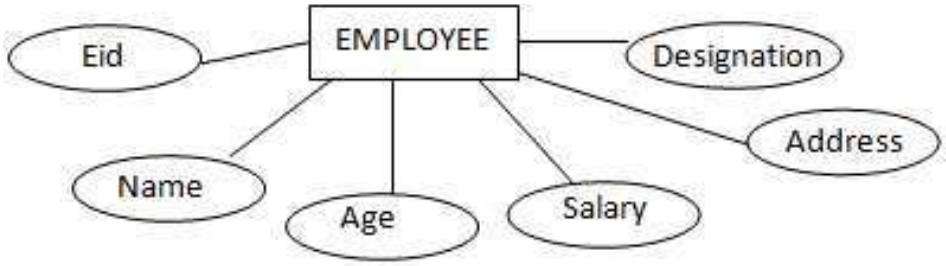


Fig. 4.1(a): EMPLOYEE entity type with six different attributes

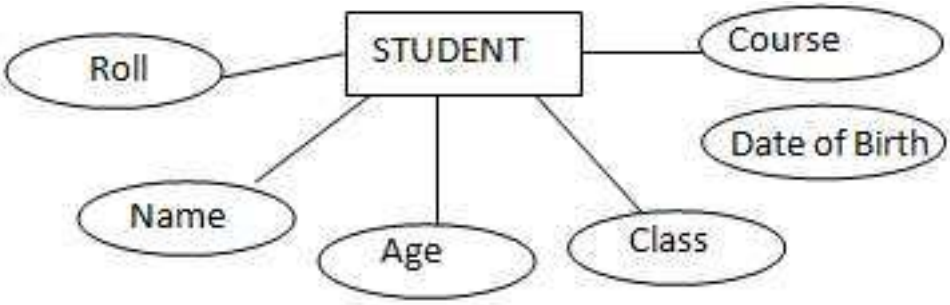


Figure 4.1(b): STUDENT entity type with six different attributes

Entity Type: A collection of a similar kind of entities having the 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.

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 entity that describe the specific feature of 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 4.1, 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 that 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 single-valued attribute.

iv) **Multi valued attribute:** attribute that have 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 must 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. For example, relationship between two entities sets *student* and *course*. Collection of all the instances of relationship forms a relationship set called relationship type.

Informally, each relationship instance r_i in relation R is an association of entities, where the association includes exactly one entity from each participating entity type. Each such relationship instance r_i represents the fact that the entities participating in r_i are related together in some way in any situation. 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.

4.3.2 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 4.2, where the degree of WORKS_FOR relationship type is two and for SUPPLY the degree is three.

a)

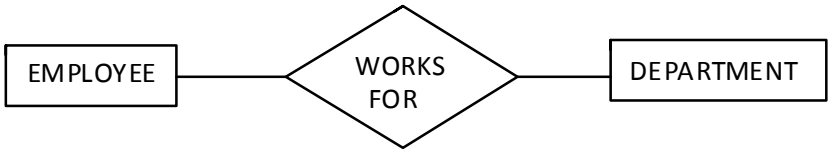


Figure 4.2 (a): binary relationship WORKS_FOR

b)

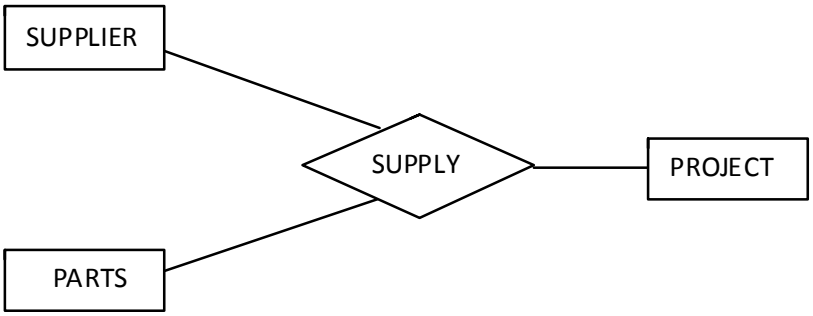


Figure 4.2 (b): Ternary relationship SUPPLY

4.3.3 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.**

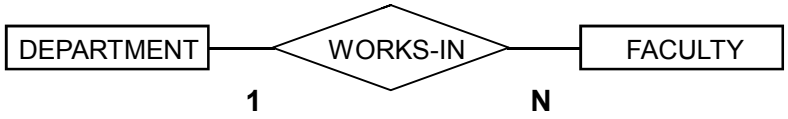
- **One-to-one (1:1):** This is the relationship where an entity in an entity type 'A' is associated with at most one entity of the entity type 'B' and an entity in entity type 'B' is associated with at most one entity in entity type 'A'. For example, the relationship between employee and department.



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.

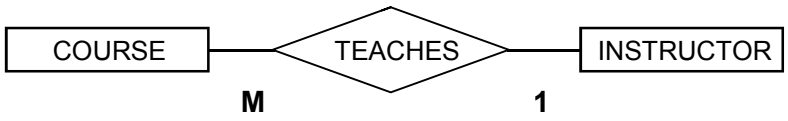
Similarly we can define the relationship between college and principal because we know that one college can have at most one principal and one principal can be assigned to only one college.

- **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.



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

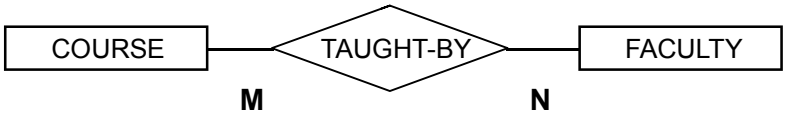
- **Many-to-one (m:1):** In this type of ratio, an entity in entity type 'A' is associated with at most one entity in entity type 'B' or an entity in entity type 'B' is associated with any number of entity in entity type 'A'. For example, the relationship between course and instructor.



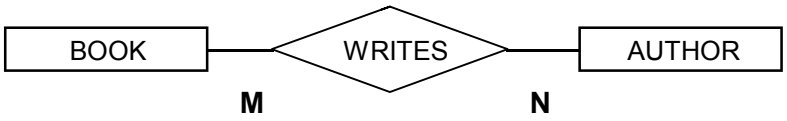
It is assumed that an instructor can teach various courses but a course can be taught only by one instructor.

- **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, 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, it 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 it is denoted by '¬%' sign.

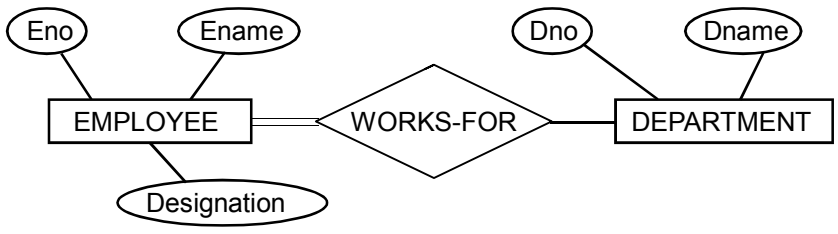


Figure 4.3: Total and Partial participation constraints

In the Figure 4.3, 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' work for any one or more department. So 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.

4.3.4 Weak and Strong Entities

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

1) **Strong entity type:** The entity type containing a key attribute is called 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.

2) **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 the 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

Q.1: 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.

Q.2: Fill in the blanks out of the following:

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 4.1

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.

4.4 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 sets 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 attributes 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 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 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.

Roll No (Primary Key)	**Name**	**Address**	**Relationship**

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 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 relationship 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)

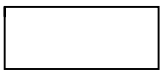
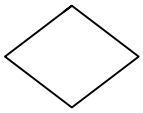
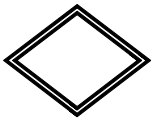
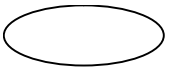
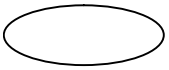
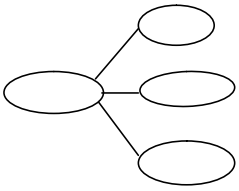
n_ary Relationship: For each n_ary relationship type R where n>2, we create a new table S to represent R. We include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n_ary relationship type (or 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.

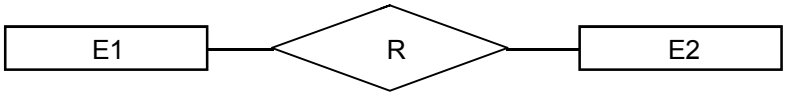
Multi valued attributes: For each multi valued attribute 'A', we create a new relation R that includes an attribute corresponding to the primary key attribute k of the relation that represents the entity type or relationship type that has as an attribute. The primary key of R is then combination of A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

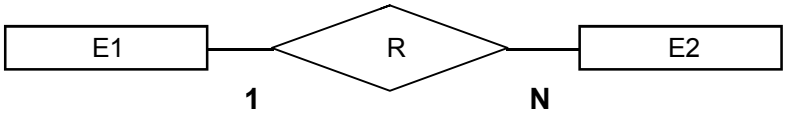
4.5 ER MODELLING SYMBOLS

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



Cardinality ratio 1:N for E1:E2



CHECK YOUR PROGRESS

Q.3: 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.

Q.4: 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 4.2

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



4.6 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 concepts 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.



4.7 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.
3) Ullman, J. D. (1984). *Principles of Database Systems*. Galgotia Publications.



4.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) True, b) False, c) True, d) True, e) False

Ans. to Q. No. 2: a) association, b) attributes, c) E-R, d) diamond, e) many-to-many

Ans. to Q. No. 3: a) True, b) False, c) True, d) False, e) False

Ans. to Q. No. 4: a) 1:N, b) Weak entity type, c) M:N, d) schema diagram



4.9 MODEL QUESTIONS

Q.1: What do you mean by Data Model? What does it define?

Q.2: Describe conceptual, logical and physical data model.

Q.3: 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.

Q.4: What do you mean by Constraints of Relationship type? Explain with example the constraint of different cardinality ratio.

Q.5: Discuss participation constraints used in E-R diagram design. Distinguish between Total participation constraints and Partial participation constraints.

Q.6: What is Cardinality Ratio in ER diagram?

Q.7: Discuss the procedure of converting E-R model into relational schema.

Q.8: Can you convert a relationship type of an E-R model into corresponding relational schema? If Yes, how?

Q.9: Explain E-R modelling symbols.

*** ***** ***

UNIT 5: RELATIONAL DATABASES

UNIT STRUCTURE

5.1 Learning Objectives

5.2 Introduction

5.3 Relational Data Model

5.4 Keys

5.5 Types of Keys

- 5.5.1 Super Key
- 5.5.2 Candidate Key
- 5.5.3 Primary Key
- 5.5.4 Alternate Key
- 5.5.5 Composite Key
- 5.5.6 Foreign Key

5.6 Let Us Sum Up

5.7 Further Reading

5.8 Answers to Check Your Progress

5.9 Model Questions

5.1 LEARNING OBJECTIVES

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

- describe the structure of the relational model
- learn about different terms associated with relational model
- 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 learnt the Entity-Relationship model. We have learnt to draw E-R diagram.

In this unit, we will discuss relational model. In this 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 is very important in case of relational databases. In fact, without keys relational database will not be usuable at all. In this unit we will also discuss the concept of keys and their use in a database.

5.3 RELATIONAL DATA MODEL

Relational data model is a simple and widely used data model which has all the properties and capabilities required to process data with storage efficiency. The relational model is based on the core concept of *relation*. In the relational model, all data is logically structured within relations (also called *table*). Relational data model stores data in the form of a table. A table has rows and columns, where rows represent records and columns represent the attributes. Each table corresponds to an application entity and each row represents an instance of that entity.

In the formal relational model terminology, a single row of a table, which contain a single record is called a ***tuple***, a column header is called ***attribute***, and each table is called ***relation***. A ***domain*** is a set of all possible values of a relation. For example, *Indian_Phone_Numbers* can be a domain name which is the set of 10-digit phone number valid in India.

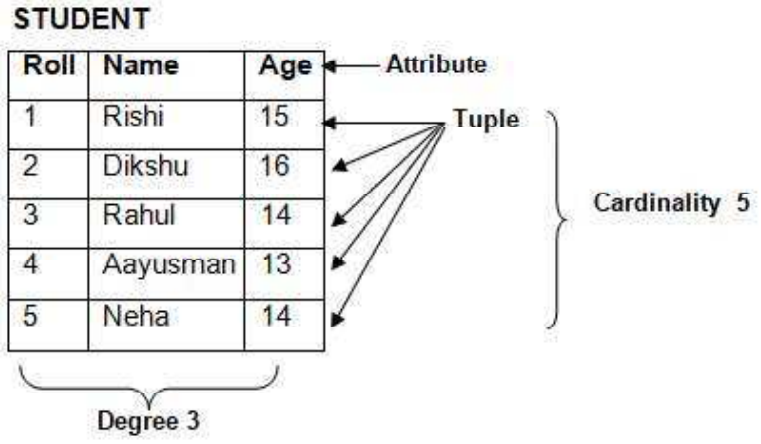


Figure 5.1: An example of a relation "STUDENT"

The number of rows in a table is called the ***cardinality*** of the table. The number of columns in table is called its ***degree***. For example, the table

in figure 5.1 has a cardinality 5 and a degree of 3. If there are *n* columns in the table, then the relation is said to be of degree *n*.

Advantages of the Relational Model:

- **Structural independence:** Relational database model has structural independence. Because, in relational model, changes in the database structure do not affect the data access. When it is possible to make change to the database structure without affecting the DBMS's capability to access data, we can say that structural independence has been achieved.
- **Conceptual simplicity:** Since the relational data model frees the designer from the physical data storage details, the designer can concentrate on the logical view of the database.
- **Design, implementation, maintenance, and usage ease:** The relational database model achieves both data independence and structural independence making the database design, maintenance, administration, and usage much easier than other models.

Disadvantages of Relational Model: As compared to the advantages of the relational model, the disadvantages of the relational model are very minor

- **Hardware overheads:** Relational database system hide the implementation complexities and the physical data storage details from the users. For doing this, i.e., for making things easier for the users, the relational database system need more powerful computers and data storage devices.
- **Ease of the design can lead to bad design:** The relational database the user need not know the complex details of the physical data storage. This ease of design and use can lead to the development and implementation of very poorly design database management systems. Since the database is efficient, these design inefficiencies will not come to light when the database is design and when there is only a small amount of data. As the database grows, the poorly designed database will slow the system down and will result in performance degradation and data corruption.

5.4 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 distinguis each tuple within a relation. A relation in a relational database must have an attribute or combination of attributes such that they can uniquely identify the tuple. This unique identifier is called ***key***. A key is a that data item that exclusively identifies a record or tuple. It may consist of one or more attributes. We can split related data into different relations or tables and logically linked 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 5.2: 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 5.2 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 student might have the same name. *Marks* field may contain same marks for more than one student. Similarly, more than one student may have 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.

5.5 DIFFERENT TYPES OF KEYS

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

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

5.5.1 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 5.3: 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 donnot 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.

5.5.2 Candidate Key

A table can have more than one columns that could be chosen as the key because they individually have the capability to identify a record uniquely. These fields are termed as ***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: 5.4: 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 5.4. The attribute ***Name*** cannot be a candidate key as more than one employee might have similar name. Similary, several employees might have same blood group. So ***Blood Group*** cannot be chosen as key.

5.5.3 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 decide 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 identifies 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 at most one primary key. For example, let us assume that 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 been appointed. ***Emp_ID*** can be a primary key as it does not change so long the person is working in the same organization.

Table: 5.5: 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 5.2., 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 a chance that there may be more than 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 it 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.



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

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

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.

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.

ii) **Minimal:** The primary key should be composed of the minimum number of fields that ensures that 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.

5.5.4 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: 5.6: 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 5.4), if we consider ***Emp_ID*** as the primary key then ***Passport_no*** will be the alternate key.



CHECK YOUR PROGRESS

Q1: Fill in the blanks:

a) The RDBMS terminology for a row is .....

b) The RDBMS terminology for a table is .....

c) The number of attributes in a relation is known as .....

d) ..... is a set of all possible values of a relation.

Q.2: 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. These 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 supplier supply item. 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

5.5.6 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 donnot 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*** 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.



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.

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.

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

Q.3: 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.



5.6 LET US SUM UP

- Relational database are based on mathematical theory of *relation*. In the relational model, all data is logically structured within a relations also called *table*.
- A database consists of tables, which consists of records, which further consist of fields.
- A table is a collection of rows. A row or a record is termed as a *tuple*.
- A relation has fixed number of named columns or *attributes*.
- The number of tuples is called *cardinality*.
- The number of attributes is called the *degree*.
- A set of values of one column in a table is called a *domain*.
- A *key* is that data item that exclusively identifies a record.
- 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.



5.7 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Elmasri, R. & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.
3) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.



5.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) tuple, b) relation, c) degree, d) Domain

Ans. to Q. No. 2: a) True, b)True, c) True, d) False, e) True

Ans. to Q. No. 3: a) True, b) True, c) False



5.9 MODEL QUESTIONS

Q.1: In the context of a relational model, discuss each of the following:

i) Relation ii) tuple

iii) cardinality iv) domain

Q.2: Explain the following keys with suitable examples:

i) Primary Key ii) Candidate Key

iii) Super Key iv) Alternate Key

v) Foreign Key.

Q.3: Create a relation student and mark the different types of keys in it.

Q.4: Explain the significance of primary key and foreign key in employee database.

Q.5: Define key?

Q.6: Differentiate between candidate key and super key.

Q.7: Discuss the properties of primary key.

Q.8: Define candidate key with appropriate example.

Q.9: Define primary key constraint.

Q.10: Define the uniqueness and irreducibility properties of candidate key.

Q.11: Design two database tables showing primary key and foreign key.

Q.12: What is the cardinality of a table with 500 rows and 10 columns?

*** ***** ***

UNIT 6: RELATIONAL ALGEBRA

UNIT STRUCTURE

6.1 Learning Objectives

6.2 Introduction

6.3 Relational Algebra and Algebraic Operations

- 6.3.1 SELECT Operation
- 6.3.2 PROJECT Operation
- 6.3.3 RENAME Operation
- 6.3.4 UNION, INTERSECTION and Difference Operation
- 6.3.5 CARTESIAN PRODUCT Operation
- 6.3.6 DIVISION Operation
- 6.3.7 JOIN Operation

6.4 Let Us Sum Up

6.5 Further Reading

6.6 Answers to Check Your Progress

6.7 Model Questions

6.1 LEARNING OBJECTIVES

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

- learn about relational algebra
- describe algebraic operation
- define different operations associated with relational algebra.

6.2 INTRODUCTION

In the previous unit we have studied Relational Data Model. We have learnt different terms associated with relational model. In the relational model of a database, all data are represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database. Concepts of keys are also discussed in the previous unit.

In this unit we will discuss Relational Algebra. Various operations associated with relational algebra are discussed in this unit.

6.3 RELATIONAL ALGEBRA AND ALGEBRAIC OPERATIONS

Relational algebra constitutes a basic set of operations for manipulating relational data. These operations enable the users to perform basic retrieval operations. The result of a retrieval operation on a table (or relation) is another relation. Thus, the relational algebraic operations produce new relations, which can be further manipulated using the same relational algebraic operation.

Relational Algebraic Operation: The relational algebraic operations are divided into two groups– *unary* and *binary*. Unary operations are those operations which operate on single relation and binary operations are those which operate on two relations.

Unary operations are:

- SELECT
- PROJECT
- RENAME

Binary operations are:

- UNION
- INTERSECTION
- SET DIFFERENCE
- CARTESIAN PRODUCT
- DIVISION
- JOIN

6.3.1 SELECT Operation

The SELECT operation is used to select a subset of the tuples from a relation that satisfy a selection condition. So we can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition. It is represented as follows:

$$\sigma_{\text{<select condition>}}(R)$$

The symbol **σ** (sigma) is used to denote the SELECT operator and the **<select condition>** is a Boolean expression specified on the attributes of the relation **R.**

The Boolean expression specified in **select condition** is made up of a number of **clauses** of the form

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

where, **<attribute name>** is the name of attribute (column) of relation R; **<comparison operator>** is one of the following comparison operator; =, “', <, d”, > or e”; **<constant value>** is constant value from the attribute domain.

For example, the SELECT operation $\sigma_{DNO=4}$**(EMPLOYEE)** select the tuples for all employee whose department is 4.

6.3.2 PROJECT Operation

The PROJECT operation is used to select certain columns (attribute) from a table, while discarding others. So if we are interested in certain columns (attribute) of a table, we can use PROJECT operation to project the relation over these attribute. The PROJECT operation is represented as follows:

$$\pi_{<\text{attribute list}>}(R)$$

The symbol **π** (pi) is used to denote the POJECT operation and the **<attribute list>** is a list of attributes from the attribute of the relation **R**. The result of the PROJECT operation has only the attributes specified in the attribute list and in the same order as they appear in the list. Hence, the degree (number of columns) of the result is equal to the number of attribute specified in the attribute list.

Following is the example of PROJECT operation:

$$\pi_{EANME,ADDRESS}(EMPLOYEE)$$

This will get the column ENAME and ADDRESS from the EMPLOYEE table.

6.3.3 RENAME Operation

The RENAME operation is used to rename a relation or attribute. The general RENAME operation when applied to a relation can take any of the following forms:

$\rho_{S(\text{new attribute names})}(R)$

$\rho_S(R)$

$\rho_{(\text{new attribute names})}(R)$

The symbol **ρ**(rho) is used to denote the RENAME operation. **S** is the new relation name and **R** original relation. The first expression renames both the relation and its attribute, the second expression renames the relation only and the third one renames only the attributes.

Following are the example of RENAME operation:

$\rho_{TEMP(SNAME,SADDRESS)}(EMPLOYEE)$

Here both the relation name and attribute names are rename.

$\rho_{TEMP}(EMPLOYEE)$

Here only the relation name is renamed:

$\rho_{(SNAME,\ SADDRESS)}(EMPLOYEE)$

In this case only the attribute names are renamed.

6.3.4 UNION, INTERSECTION, and DIFFRERENCE Operation

The operations UNION, INTERSECTION, and DIFFERENCE requires that the tables (relation) involved be union compatible. Two relations are said to be union compatible if the following conditions are satisfied.

- The two relations/tables must contain the same number of columns (i.e., having the same degree).
- Each column of the first relation/table must be either same data type as the corresponding column of the second relation/table or convertible to the same data type as the corresponding column of the second. Suppose there are two relations R and S as shown below:

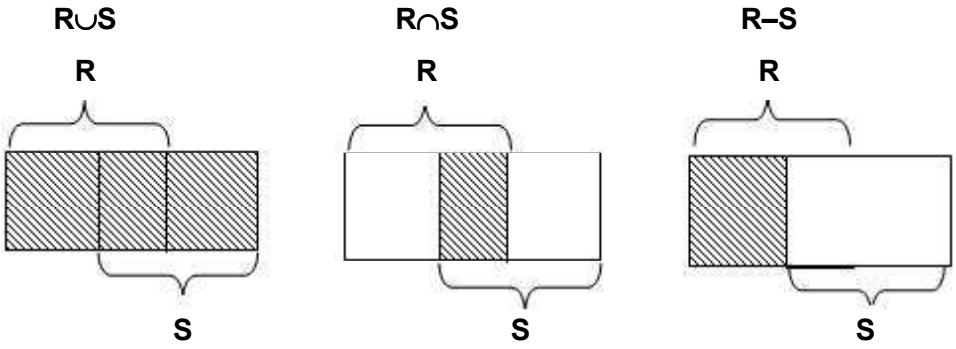
R

ROHIT	KOLKATA
HARISH	DELHI
JEEVAN	MUMBAI
KRISH	GHY

S

KRISH	GHY
RAM	DELHI
ROHIT	KOLKATA
RITESH	MUMBAI
HARSH	CHENNAI

These three operations are represented by the following pictures:



UNION **INTERSECTION** **DIFFERENCE**

- **UNION:** UNION is directly analogous of the basic mathematical operations on sets (tables). The result of this operation denoted by **R∪S**. This is the relation that includes all tuples that are either in the relation R or in the relation S, or in both R and S. Duplicates are eliminated.
- **INTERSECTION:** The result of this operation is a relation that includes all tuples that are in both R and S. The intersection operation is denoted by **R∩S.**
- **DIFFERENCE:** The difference operation is denoted by **R–S.** The result of this operation is the relation that includes all tuples that are in **R** but not in **S**:

ROHIT	KOLKATA
HARISH	DELHI
JEEVAN	MUMBAI
KRISH	GHY

R∩S

ROHIT	KOLKATA
KRISH	GHY

R–S

HARISH	DELHI
JEEVAN	MUMBAI

R∪S

6.3.5 CARTESIAN PRODUCT Operation

The **CARTESIAN PRODUCT** is also known as **CROSS PRODUCT** or **CROSS JOIN**. It is denoted by 'X'. The **CARTESIAN**

PRODUCT between relations R and S is denoted by **R X S**. The relations on which we are performing this operation need not to be union compatible. This operation is used to combine tuples from two relations in a combinational fashion.

The result of **CARTESIAN PRODUCT** of two relations, which have **m** and **n** columns, is a relation that has **m+n** columns. The resulting relation will have one tuple for each combination of tuples one from each participating relation. Hence, if the relations have $\mathbf{m_R}$ and $\mathbf{n_S}$ tuples, respectively, then the CARTESIAN PRODUCT will have $\mathbf{m_R * n_S}$ tuples. Consider the following two tables **BORROWER** and **LOAN,** one containing the borrower details and the other containing loan details.

BORROWER

BName	BIno
JHON	L-102
ROHIT	L-201
KRISH	L-315

LOAN

Loanno	Branch	Amount
L102	GUWAHATI	30000
L-201	PANBAZAR	40000

The **BORROWER** has 2 columns and 3 tuples and **LOAN** has 3 columns and 2 tuples. So the **CARTESIAN PRODUCT** has 5 columns and 6 tuples.

BORROWER X LOAN

BName	BIno	Loanno	Branch	Amount
JHON	L-102	L-102	GUWAHATI	30000
JHON	L-102	L-201	PANBAZAR	40000
ROHIT	L-201	L-102	GUWAHATI	30000
ROHIT	L-201	L-201	PANBAZAR	40000
KRISH	L-315	L-102	GUWAHATI	30000
KRISH	L-315	L-201	PANBAZAR	40000

6.3.6 DIVISION (÷) Operation

The division is a binary operation that is written as *R* ÷ *S*. The result consists of the restrictions of tuples in *R* to the attribute names unique to *R*, i.e., in the header of *R* but not in the header of *S*, for which it holds that all their combinations with tuples in *S* are present in *R*. For an example see the tables *Completed*, *DBProject* and their division:

Completed	
Student	**Task**
Juri	Database1
Juri	Database2
Juri	Compiler1
Anil	Database1
Anil	Compiler1
Varun	Database1
Varun	Database2

DBProject
Task
Database1
Database2

Completed *DBProject*
Student
Juri
Varun

If *DBProject* contains all the tasks of the database project, then the result of the division above contains exactly the students who have completed both of the tasks in the database project.

More formally, the semantics of the division is defined as follows:

$R \div S = \{t[a_1, ..., ..., ..., a_n] : t \in R \wedge \forall s \, S ((t [a_1, ..., ..., ..., a_n] \, s) R)\}$

where, $\{a_1, ..., ..., ..., a_n\}$ is the set of attribute names unique to *R* and $t[a_1, ..., ..., ..., a_n]$ is the restriction of *t* to this set. It is usually required that the attribute names in the header of *S* are a subset of those of *R* because otherwise the result of the operation will always be empty.

The simulation of the division with the basic operations is as follows. We assume that a_1, ..., ..., ..., a_n are the attribute names unique to *R* and b_1, ..., ..., ..., b_m are the attribute names of *S*. In the first step we project *R* on its unique attribute names and construct all combinations with tuples in *S*:

$$T: = \pi_{a1, ..., ..., ..., an}(R) \times S$$

In the prior example, T would represent a table such that every Student (because Student is the unique key/attribute of the Completed table) is combined with every given Task. So Eugene, for instance, would have two rows, Eugene→Database1 and Eugene → Database2 in T.

In the next step we subtract *R* from this relation:

$$U : = T \text{ “ } R$$

Note that in *U* we have the possible combinations that “could have” been in *R*, but weren’t. So if we now take the projection on the

attribute names unique to *R* then we have the restrictions of the tuples in *R* for which not all combinations with tuples in *S* were present in *R*:

$$V: = \pi_{a1, ..., ..., ..., an}(U)$$

So what remains to be done is take the projection of *R* on its unique attribute names and subtract those in *V*:

$$W: = \pi_{a1, ..., ..., ..., an}(R) \text{ “ } V$$

6.3.7 JOIN Operation

JOIN operation is denoted by ‘’ is used to combine related tuples from two relations in to a single tuple. The general form of join operation on two relations **R**(A_1 A_2, …, …, …, A_m) and **S**(B_1,B_2 ….B_n) is $R_{<\text{join condition}>}S$. The result of the join is a relation **Q** with **m+n** attributes. The order of the relation **Q** will be **Q**(A_1 A_2, …, ..., ..., A_m B_1 B_2, …, ..., ..., B_n). **Q** has one tuple for each combination of tuple, one from **R** and one from **S** whenever the join condition satisfied.

Let us take two relations:

BORROWER

BName	Blno
JHON	L-102
ROHIT	L-201
KRISH	L-315

LOAN

Loanno	Branch	Amount
L102	GUWAHATI	30000
L-201	PANBAZAR	40000
L-301	MALIGAON	50000

BORROWE ⋈ <Loanno=Blno>LOAN

BName	Blno	Loanno	Branch	Amount
JHON	L-102	L-102	GUWAHATI	30000
ROHIT	L-201	L-201	PANBAZAR	40000

- ***Theta Join:*** A general join condition is of the $A_i \theta B_j$ where A_i is an attribute of the relation **R** and B_j is an attribute of the relation **S** A_i and B_j have common domain and θ is one of the comparison operator {=, ‘“, >, <, >=, <=}. A join operation with such general join condition is called **theta join**.
- ***Natural Join:*** Natural join is an equijoin of two relations R and S over all common attributes. One occurrence of each common attribute is eliminated from the result. The Natural Join operation

performs an Equijoin over all the attributes in the two relations that have the same name. The degree of the Natural Join is the sum of the degree of the two relations R and S less the number of common attributes.

- ***Outer Join:*** Often in the joining two relations, a tuple in one relation does not have a matching tuple in other relation: in other words, there is no matching value in the join attributes. We may want a tuple from one of the relations to appear in the result even when there is no matching value in the other relation. This may be accomplished by using **Outer join**. The *left outer join* is a join in which the tuple from R that do not have matching value in the common attributes of S are also included in the result relation. Missing value in the second relation are set to null. A left outer join keeps every tuple in the left-hand relation in the result. Similarly, there is a *right outer join* that keeps every tuple in the right-hand relation in the result. There is also a *full outer join* that keeps all tuples in both relations, padding tuples with null values when no matching tuples are found.



CHECK YOUR PROGRESS

Q.1: Choose the correct answer:

i) Which operation is used to select subset of tuples from a relation?

a) Project b) Select c) Rename d) Join

ii) Which operation is used to select attributes from a relation?

a) Project b) Select c) Rename d) Join

iii) If R and S are two relations, then intersection between two relations is denoted by

a) R∩S b) R–S c) R∪S d) S–R

iv) JOIN operation is denoted by the symbol

a) σ b) π c) ρ d) ⋈



6.4 LET US SUM UP

- Relational model for database management is a database model based on first order predicate logic, first formulated and proposed in 1969 by Edgard F. Codd.
- Codd's rules are a set of twelve rules proposed by Edgard F. Codd.
- Application programs are not affected by changes in the physical data access and storage methods are the Physical Data Independence.
- Logical Data Independence is the logical changes in tables and views such as adding/deleting columns or changing field lengths need not necessitate modification in the programs or in the format of ad-hoc requests.
- In relational model terminology, a row is called ***tuple***, a column header is called ***attribute***, and each table is called ***relation***.
- Domain constraints specify that within each tuple, the value of each attribute A must be an attribute value from the domain *dom(A).*
- A relation schema may have more than one key. In this case, each of the key is called candidate key.
- The relational algebraic operations are divided into two groups-unary and binary.
- Unary operations are those operations which operate on single relation.
- Binary operations are those which operate on two relations.
- The operations UNION, INTERSECTION, and DIFFEENCE –requires that the tables (relation) involved be union compatible.
- Join operation is denoted by '' is used to combine related tuples from two relations in to a single tuple.



6.5 FURTHER READING

1) Elmasri, R. & Navathe, S. B. (2015). Fundamentals of Database Systems. Pearson.

2) Leon, A., Leon, M. *Fundamentals of Database Management System*. Tata McGraw Hill.



6.6 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (a); ii) (a); iii) (b); iv) (d)



6.7 MODEL QUESTIONS

Q.1: What are Codd's rules?

Q.2: What is logical data independence?

Q.3: What is physical data independence?

Q.4: What is primary, candidate, foreign key?

Q.5: What are the constraints include in relational model?

Q.6: What is entity and referential integrity constraint?

Q.7: What are the advantages and disadvantages of relational model?

Q.8: What are the notations used to denote Relational model?

Q.9: What is relational algebra and what are its uses?

Q.10: What is SELECT, PROJECT and RENAME operation? How is it represented? Explain with example.

Q.11: What is a JOIN operation? How is it represented? Explain with example.

Q.12: How does a tuple relational calculus differ from domain relational calculus?

Q.13: Discuss the meaning of the existential quantifier (“) and universal quantifier (“).

Q.14: Define the following terms with respect to the domain relational calculus: tuple variable, range relation, atom, formula, and expression.

Q.15: What is mean by safe expression in relational calculus?

*** ***** ***

UNIT 7: INTRODUCTION TO SQL

UNIT STRUCTURE

7.1 Learning Objectives

7.2 Introduction

7.3 Structured Query Language (SQL)

7.3.1 Characteristics of SQL

7.4 SQL Data Types

7.4.1 Numeric Data Type

7.4.2 Character Data Type

7.4.3 Bit String Data Type

7.4.4 Boolean Data Type

7.4.5 Date and Time Data Type

7.5 Types of SQL Commands

7.5.1 Data Definition Language (DDL)

7.5.2 Data Manipulation Language (DML)

7.5.3 Transaction Control Statements (TCS)

7.6 Basic SQL Commands

7.6.1 Creating and Deleting Database

7.6.2 Creating Table

7.6.3 Altering Tables

7.6.4 Deleting Tables

7.6.5 Inserting Data

7.6.6 Updating Data

7.6.7 Deleting Data

7.7 Let Us Sum Up

7.8 Further Reading

7.9 Answers to Check Your Progress

7.10 Model Questions

7.1 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
- learn commands used to create, alter and delete databases and tables
- insert, update and delete data from tables.

7.2 INTRODUCTION

In the previous unit you have been introduced to the concepts of relational algebra. The different operations like Select, Project, Union, Difference, Intersection, Cartesian Product, and Natural join have been covered in the previous unit in detail.

In this 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. 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 tables.

In the next unit we will discuss SQL commands related to conditional and logical operators. Some other additional SQL commands like IN, BETWEEN, ORDER BY and many more will also be discussed in detail.

7.3 STRUCTURED QUERY LANGUAGE (SQL)

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. It was initially developed as “SEQUEL (Structured English Query Language)” at IBM and was designed to manipulate and retrieve 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 7.1: 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 many different databases that are in use today namely– Microsoft SQL Server, Oracle, MySQL among many. The basic commands used in all of these databases use the same syntax of structured query language. This helps in maintaining uniformity among the databases. Let us consider the scenario if 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 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.

7.3.1 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 software's 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 users to create, update, delete and retrieve data objects from the database.

As any other language, SQL also has its share of advantages and disadvantages.

- Some of the ***advantages*** of SQL are listed below:
  - SQL provides a well-defined standard for database query languages
  - Easy to learn and understand
  - 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.
- 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

Q.1: Give two characteristics of SQL.

Q.2: Mention two advantages and two disadvantages of SQL.

7.4 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.

7.4.1 Numeric Data Type

Numeric data type includes integer numbers of different sizes and also floating point numbers of different precisions. 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.

7.4.2 Character-string Data Type

Character-string data types are used to represent characters of fixed and variable lengths. Some of the numeric 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. It means

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, then 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.

7.4.3 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)): Variable 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’.

7.4.4 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’, ‘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.

7.4.5 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'.

The most 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 new two 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 7.2: Data Type Summary

Data Type	Keyword	Range	Valid	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'

7.5 TYPES OF SQL COMMANDS

We now have a basic idea about the various data types available in SQL. Now, let us look at the commands that we will 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. 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.

7.5.1 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 7.3.

Table 7.3: 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

7.5.2 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 7.4.

Table 7.4: 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

7.5.3 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 7.5.

Table 7.5: 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

Q.3: How is CHAR data type different from VARCHAR data type?

Q.4: Give the range for small integer data type.

Q.5: How can we save our transactions permanently in the database?

Q.6: Give the difference between TRUNCATE and DROP command.

7.6 BASIC SQL COMMANDS

We have been introduced to basic SQL data types till now. We also have an idea of data definition and data manipulation language commands from the previous sections. Most of these commands will be covered in detail in this section and in the next unit.

7.6.1 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.

7.6.2 Creating a 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)
```

Figure 7.1: Output of *desc* command for "students" table

7.6.3 Altering a Table

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 column”** 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)
```

Figure 7.2: Output of desc command after table "course_details" is altered 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)
```

Figure 7.3: Output of *desc* command for Table "students" 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 these 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 now store 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)
```

Figure 7.4: Output of *desc* command after "students" table is altered 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)
```

Figure 7.5: Output of *desc* command after column "gender" is dropped 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 primary 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)
```

Figure 7.6: Output of desc command after "course_details" table is altered to add primary key

7.6.4 Deleting a 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 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 exiting 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)
```

Figure 7.7: Output of *show Tables* Command

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

Q.7: Give the command to view the structure of a table.

Q.8: What is the syntax for renaming a database table?

Q.9: Write the statement to add a column "age" to the "students" table.

Q.10: Give the command to see the list of tables in a database.

7.6.5 Inserting Data in Table

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)
```

Figure 7.8: Output of *select* command for "students" table

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 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)
```

Figure 7.9: Output of *select* command for "students" table after insertion of more rows

7.6.6 Updating Data in Table

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)
```

Figure 7.10: Output of *select* command for "students" table after update operation

7.6.7 Deleting Data from Table

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

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

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)
```

Figure 7.11: Output of *select* Command for "students" Table after Delete Operation



CHECK YOUR PROGRESS

Q.11: 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.



7.7 LET US SUM UP

- Structured Query Language (SQL) is the standard language for relational databases.

- SQL is based on relational algebra and tuple relational calculus.
- 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)).
- 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 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 manipulation language commands.
- Some of the commands in Transaction Control Statements (TCS) are:
  COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION.
- 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;
```

- A table is created using the ***create table*** command.
- 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.
- A table is deleted by using the ***drop table*** command.
- 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;
```



7.8 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Elsmari, R. & Navathe, S. (2000). *Fundamentals of Database Systems*. Addison-Wesley, 106, 107.
3) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.



7.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: 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.

Ans. to Q. No. 2: Two advantages of SQL are:

i) SQL provides a well-defined standard for database query languages
ii) Easy to learn and understand.

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.

Ans. to Q. No. 3: 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.

Ans. to Q. No. 4: The range for small integer data type is from –32,768 to +32,767.

Ans. to Q. No. 5: We can save our transactions permanently in the database by using the COMMIT command.

Ans. to Q. No. 6: DELETE command is used to delete the whole database table whereas TRUNCATE command is used to only empty the contents of the table.

Ans. to Q. No. 7: We can view the structure of the table by using the ***desc*** command. The syntax for using this command is:

```
DESC table_name;
```

Ans. to Q. No. 8: 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.

Ans. to Q. No. 9: The following statement can be used to add a column age to the "students" table.

```
ALTER TABLE students ADD age int NOT NULL;
```

Ans. to Q. No. 10: The command "SHOW TABLES" displays the list of tables present in a database.

Ans. to Q. No. 11: 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';
```



7.10 MODEL QUESTIONS

Q.1: Describe the different data types in SQL.

Q.2: Explain the need for SQL.

Q.3: Describe the different DML commands.

Q.4: Describe the different DML commands.

Q.5: Name the different commands used in transaction control statements.

Q.6: Differentiate among DATE, TIME and TIMESTAMP commands using examples.

Q.7: Describe the different types of Bit-String data type.

Q.8: What are formatted data types? Describe in brief.

Q.9: Describe in brief with examples where we can use Boolean data type.

Q.10: Why do we need SQL?

Q.11: Describe the different ways in which a table can be altered.

Q.12: Give the command to show all the databases in the database system.

Q.13: Give the syntax for deleting a database.

Q.14: Describe how "create table" is used to for creating tables in a database using example.

Q.15: How can a primary key be added after the table has already been created?

Q.16: 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.

*** ***** ***

UNIT 8: ADVANCED SQL COMMANDS

UNIT STRUCTURE

8.1 Learning Objectives

8.2 Introduction

8.3 Advanced SQL Commands

- 8.3.1 Select
- 8.3.2 Where
- 8.3.3 Order By
- 8.3.4 Group By
- 8.3.5 Having
- 8.3.6 Logical Operators (AND, OR, NOT)

8.4 Join

- 8.4.1 Inner Join
- 8.4.2 Left (Outer) Join
- 8.4.3 Right (Outer) Join
- 8.4.4 Full (Outer) Join

8.5 SQL Sub-queries

8.6 SQL Constraints

- 8.6.1 Not Null Constraint
- 8.6.2 Unique Constraint
- 8.6.3 Primary key Constraint
- 8.6.4 Foreign key Constraint

8.7 Let Us Sum Up

8.8 Further Reading

8.9 Answers to Check Your Progress

8.10 Model Questions

8.1 LEARNING OBJECTIVES

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

- describe select, where, having, group by clauses
- describe different types of join

- learn views and indexes
- describe queries and sub-queries
- describe different constraint considerations.

8.2 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 this unit you will be introduced to the advanced SQL commands. Clauses like Select, Where, Group By, Having etc. The different types of join are discussed in detail in this unit. Queries and sub-queries are also introduced in this unit. A brief introduction to views and indexes are also covered in this unit. In the next unit we will explore the concept of database normalization.

8.3 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.

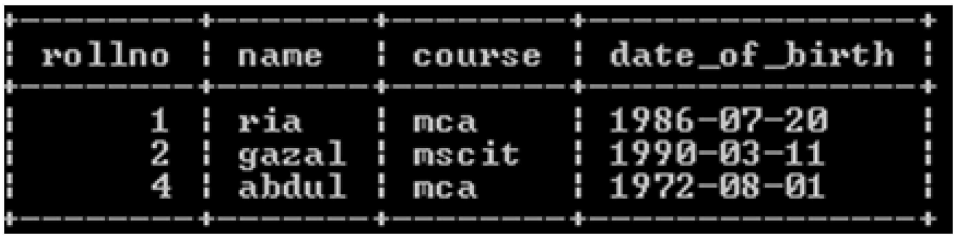
8.3.1 Select

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 8.1 below:



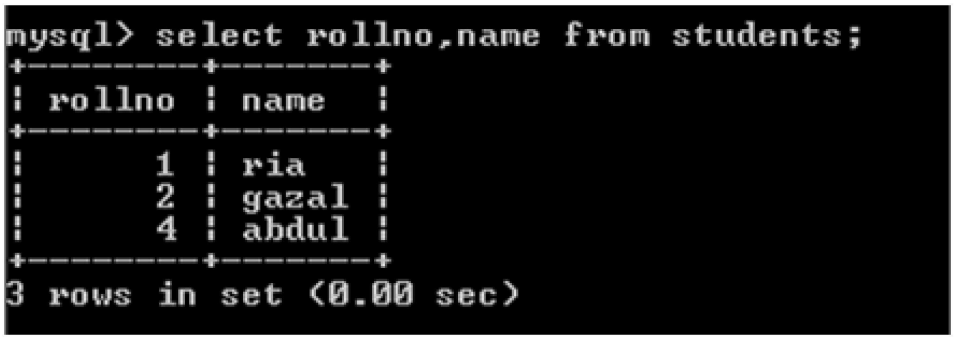
rollno	name	course	date_of_birth
1	ria	mca	1986-07-20
2	gazal	mscit	1990-03-11
4	abdul	mca	1972-08-01

Figure 8.1: 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 8.2 below:



```
mysql> select rollno,name from students;
+--------+-------+
| rollno | name  |
+--------+-------+
|      1 | ria   |
|      2 | gazal |
|      4 | abdul |
+--------+-------+
3 rows in set (0.00 sec)
```

Figure 8.2: Example showing rollno and name field information only from table "students"

```
SELECT*from tablename;
```

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;
SELECT * from students;
```

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 8.1.

8.3.2 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 8.3. 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)
```

Figure 8.3: 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 8.4: Example showing all records of table "employee"

```
SELECT name
from employee
WHERE salary > 30000;
```

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 8.5: 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 starts 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 (_)

```
SELECT column1, column2, ..., columnN
from tablename
where [condition] LIKE [pattern];
```

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];
SELECT name, salary
from employee
WHERE name LIKE 'P%';
```

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 8.6.

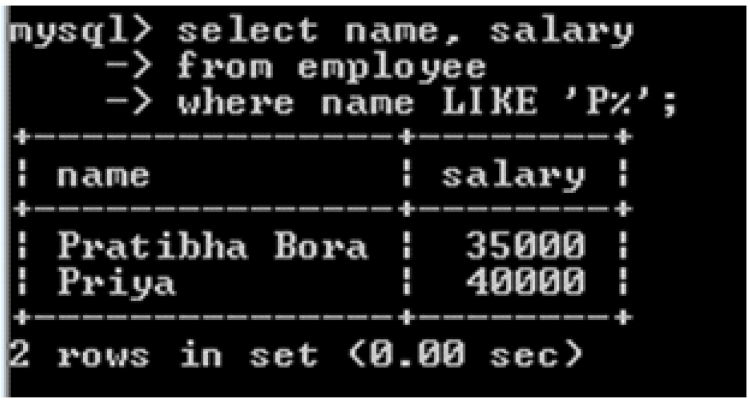


Figure 8.6: Example showing use of like clause

8.3.3 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 ordered according to the column whose name appears first, followed by the second and later column names. Let us look at the following 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 8.7: Example showing use of ORDER BY clause

8.3.4 Group By

The GROUP BY clause is used to group the output records in accordance to 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 keywod 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 clauses 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 8.8.

```
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 8.8: Example showing use of GROUP BY clause

8.3.5 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]
```

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 8.9.

```
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 8.9: Example showing use of HAVING clause

8.3.6 Logical Operators

Logical operators ike 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 8.10.

```
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 8.10: 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 8.11: Example showing use of NOT logical operator



CHECK YOUR PROGRESS

Q.1: 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".

8.4 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.

8.4.1 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 among 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 8.12 where the shaded part is the selected region from both the tables.

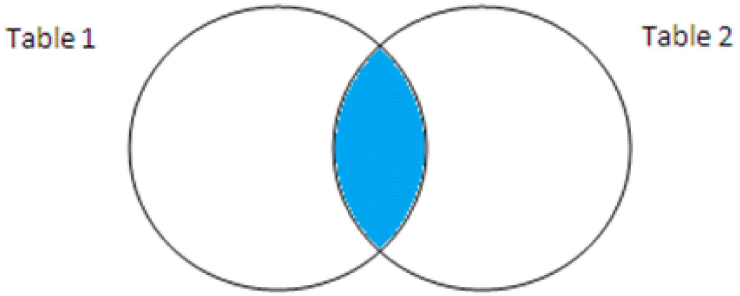


Figure 8.12: Inner Join

It can be clearly understood from figure 8.12 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 8.13. The structure and record details for table 'odr' are displayed in figure 8.14.

```
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 8.13: 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 8.14: Structure and record values 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 8.15. In figure 8.15, 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 8.15: Example showing use of INNER JOIN

8.4.2 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 8.16 where the shaded part is the region that is selected when we use LEFT JOIN.

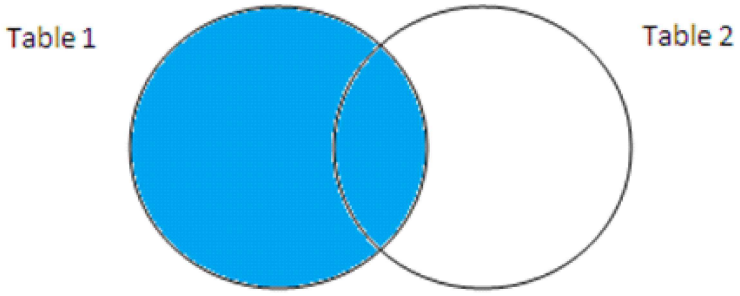


Figure 8.16: Left (Outer) Join

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 8.17. In figure 8.17, 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 8.17: Example showing use of LEFT JOIN

8.4.3 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 8.18 where the shaded part is the region that is selected when we use RIGHT JOIN.

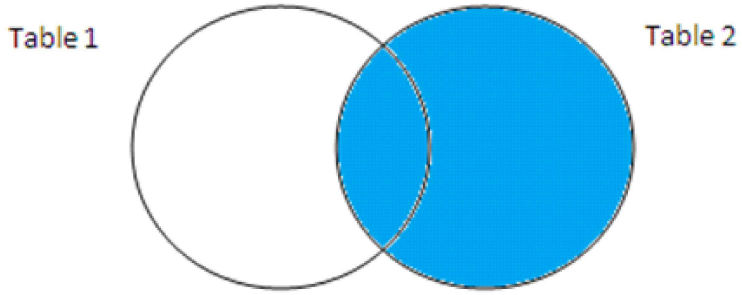


Figure 8.18: Right (Outer) Join

It can be clearly understood from figure 8.18 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 8.19.

In figure 8.19, 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 8.19: Example showing use of RIGHT JOIN

8.4.4 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 missing 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 8.20 where the shaded part is the region that is selected when we use FULL JOIN.

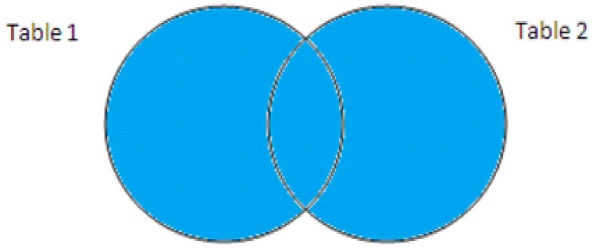


Figure 8.20: Full (Outer) Join

It can be clearly understood from figure 8.20 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 8.21. In figure 8.21, 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 8.21: Example showing use of FULL JOIN

8.5 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 8.22.

```
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 8.22: Example showing use of subquery

8.6 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.

8.6.1 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.

8.6.2 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.

8.6.3 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 columns 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”.

8.6.4 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

Q.2: Give the syntax of INNER JOIN.

Q.3: Define subqueries.

Q.4: Define constraints.

Q.5: What is the purpose of using Not Null constraint?



8.7 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.
- 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.



8.8 FURTHER READING

1) Date, C. J. (2006). *An Introduction to Database Systems*. Pearson Education India.
2) Elsmari, R. & Navathe, S. (2000). *Fundamentals of Database Systems*. Addison-Wesley, 106, 107.
3) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.



8.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: 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';`

Ans. to Q. No. 2: The syntax of INNER JOIN is:

```
SELECT table1.column, table2.column
from table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

Ans. to Q. No. 3: A subquery is a query written within another query. Subqueries are also called as nested query or inner queries.

Ans. to Q. No. 4: 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.

Ans. to Q. No. 5: 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.



8.10 MODEL QUESTIONS

Q.1: 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 count 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.

Q.2: 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”.

Q.3: Perform inner join on the tables “bank_customer” and “home_loan”.

Q.4: Perform left join on the tables “bank_customer” and “home_loan”.

Q.5: Describe NOT NULL constraints. Why do we need NOT NULL constraints?

Q.6: Explain the concept of right join.

Q.7: Describe subqueries using examples.

Q.8: Describe the use of different logical operators in database.

Q.9: Describe primary key and foreign key constraints.

Q.10: Perform FULL OUTER JOIN on the tables “”bank_customer” and “home_loan”.

*** ***** ***

Name of the Paper: .....
Course Code: .....

Centre for Internal Quality Assurance (CIQA)
Krishna Kanta Handiqui State Open University
City Office: Housefed Complex, Guwahati-781006

Learner's Feedback on Course

Dear Learner,

Regarding the course as mentioned above, we would like to know your opinions and comments so as to improve the quality of self learning materials in future. Please respond to the following statements by ticking the number you feel most reflect your opinion. After completion of the additional comments, please detach the page and send/mail the same to us at the address given below.

The Director, Centre for Internal Quality Assurance, KKHandiqui State Open University
Housefed Complex, Dispur, Guwahati-781006
(E-mail id: ciqa@kkhsou.in)

1) Approximately how many hours did you spend for studying the units in the course? ☐☐☐

2) Please give your opinions (by ✓ mark) to the following items based on your reading of the block:

Sl. No.	Statements	Strongly Agree	Agree	Neither Agree nor Disagree	Disagree	Strongly Disagree
I)	The SLMs of the course fulfil the learning objectives set out in the beginning of each unit					
II)	The units of the SLM could satisfy my academic needs and expectations					
III)	The Unit writers have excellent knowledge about the course contents					
IV)	Language and contents of the units were presented at a level which I could readily understand					
V)	Ample opportunity for participation in the activities provided in the units					
VI)	Used enough Illustrations (Diagrams, tables etc.) for conceptual clarity					
VII)	Quality of content is engaging, relevant, and up-to-date					
VIII)	The self check questions are very helpful					
IX)	The Possible/model questions and the answers to check my progress have benefited me a lot					

Additional Comments: *(Please feel free to provide your open comments)*

1) Which aspects of the SLM, according to you, worked well?

   .....

   .....

2) What sort of changes/improvements do you feel KKHSOU could implement to improve the overall quality of the SLM?

   .....

   .....

Thank you for taking the time to complete this form.