PG CA S2-03 (DSC)

TWO YEAR MASTER'S DEGREE COURSE

(CBCS in Accordance with NEP 2020)

SEMESTER - II

FUNDAMENTALS OF DATABASE MANAGEMENT SYSTEM

MSc IT



Hiranya Chandra Bhuyan School of Science and Technology

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakoti Mahanta, Gauhati University
Prof. Hiren Kumar Deva Sarma, Gauhati University
Dr. Lipi B Mahanta, IASST, Guwahati
Dr. Sanjib Kr. Kalita, Gauhati University

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

SLM Preparation Team

UNITS	CONTRIBUTORS
1, 5	Dr.Tapashi Kashyap Das, KKHSOU
2, 10	Mr. Arabinda Saikia, KKHSOU
3, 9	Mr. Aniruddha Deka, Royal Global University
4	Dr. Sangeeta Kakoty, KKHSOU
6, 15	Mr. Biswajit Das, Cotton University
7, 8	Dr. Sruti Sruba Bharali, KKHSOU
11, 12	Mr. Dwipen Laskar, Gauhati University
13	Dr. Uzzal Sharma, Assam Don Bosco University
14	Dr. Ridip Dev Choudhury, KKHSOU

Editorial Team

Content	:	Dr.Tapashi Kashyap Das, KKHSOU (Units 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
		Dr. Sruti Sruba Bharali, KKHSOU (Units 1 & 5)
Language	:	Prof. (Retd.) Robin Goswami, Cotton College
Structure, Format & Graphics	:	Dr. Tapashi Kashyap Das, KKHSOU

April, 2024

ISBN :

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

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

Headquarters: Patgaon, Rani Gate, Guwahati-781017
City Office: Resham Nagar, Khanapara, Guwahati-781022; Web: www.kkhsou.in

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

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY
Fundamentals of Database Management System

CONTENTS

		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** 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.	**95-126**

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).

UNIT 9: Database Design **159-168**

Design of Relational Database, Functional Dependency Diagram, Dependency-preserving property.

UNIT 10: Normalization of Database **169-186**

Introduction to Normalization, Anomalies of un-normalized Database, Normalization of Database: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF.

UNIT 11: System Implementation Techniques **187-208**

Query processing and optimization- translation between SQL queries and relational algebra.

UNIT 12: Transaction Processing **209-227**

Transaction and System concepts, Desirable properties, Schedules and Recoverability.

UNIT 13: Concurrency Control **228-251**

Locking Techniques, Concurrency Control based on timestamp ordering, Multiversion Concurrency Control Techniques .

UNIT 14: Database Recovery **252-266**

Concepts and techniques, Recovery in Multi Database Systems.

UNIT 15: Security and Authentication **267-281**

Issues, Access Control Techniques, Introduction to Multilevel Security.

Reference **282**

COURSE INTRODUCTION

Welcome to the "Fundamentals of Database Management System" course. In this course, we will explore the basics of organizing and managing. data Databases are crucial for storing and retrieving information in today's digital world, and this course will help you understand how they work.

In the first unit , we'll start by learning about data and information. We'll explore things like fields, key fields, records, and the different ways data can be stored. We will also learn the functioning of files and explore the primary operations that can be executed on them. Moving forward, we will introduce Database Management Systems (DBMS), highlighting their distinctions from traditional file systems. Discussions will encompass the advantages and disadvantages of DBMS, the structural aspects of databases, and the responsibilities associated with being a Database Administrator. The subsequent unit will focus on data models, serving as the blueprints for databases. Various types, including logical models like the relational model, will be explored, shedding light on how they represent data. We'll also explore how data relationships are represented using Entity-Relationship (ER) Diagrams. We'll learn about entities, attributes, and relationships, as well as some advanced features of ERDS. The course will further navigate through the realm of relational databases, where we will become acquainted with terms such as tables, rows, columns, and keys, contributing to efficient data organization. We'll also learn about the basic operations of Relational Algebra, which is a way to work with data in databases. With Structured Query Language (SQL) we will learn how to interact with databases This section covers the creation of databases and tables, along with operations such as adding, updating, and deleting data. Advanced commands for selecting and manipulating data in SQL will also be explored. Normalization, a process aimed at reducing redundancy and errors in data organization, will be discussed in detail, covering various normalization levels. We'll also explore how databases handle queries and optimize their performance. We'll learn about transactions, which are sequences of database operations. We'll discuss their properties and how to ensure data consistency. Managing multiple users accessing a database simultaneously will be explored in the concurrency control unit. We'll also discuss how databases recover from errors and crashes. Finally, we'll address the important topic of database security and authentication, including access control and multilevel security. By the end of this course, you'll have a solid understanding of how databases work and how to design and manage them effectively. Whether you're interested in a career in database administration, programming, or data analysis, this course will provide you with valuable skills in handling and organizing data. Get ready to explore the world of databases!

Each unit 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. Again, we have included some relevant concepts in "LET US KNOW" along with the text. And, at the end of each section, you will get "CHECK YOUR PROGRESS" questions. These have been designed to self-check your progress of study. It will be better 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 PPROGRESS" 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 us 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 being 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.

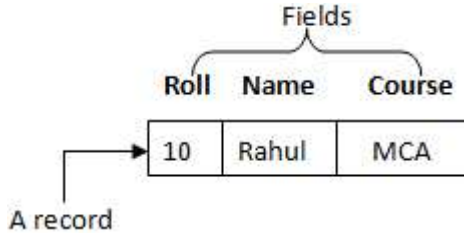


Figure 1.1: An example of Record

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)

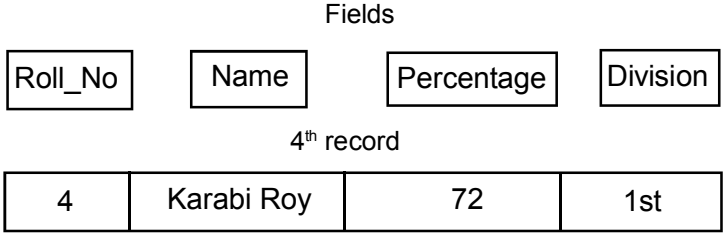
In the above record, the respective fields are Roll_No, Name, Percentage and Disvision. This record occupies 19 characters including the blank spaces and commas.

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 files, 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 Advantages of Database Approach

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, responsibilities 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.

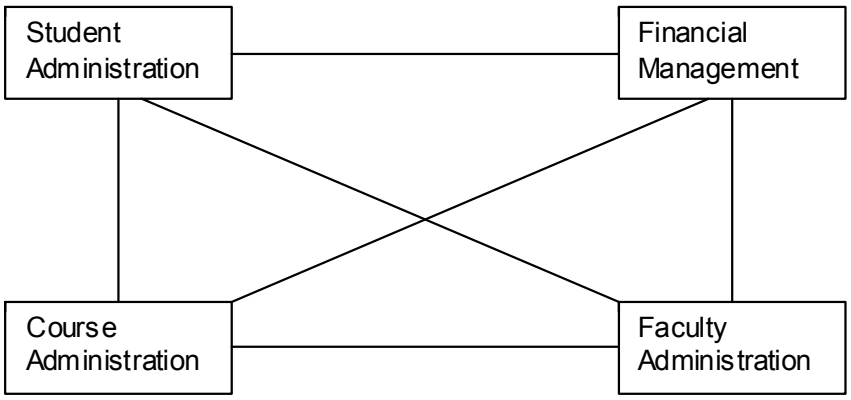


Figure 2.1: Traditional File Approach

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).

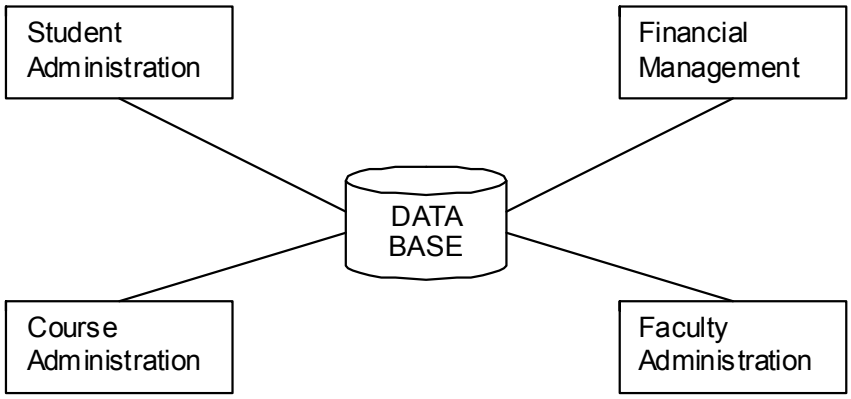


Figure 2.2: Database Approach

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 Advantages 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.)

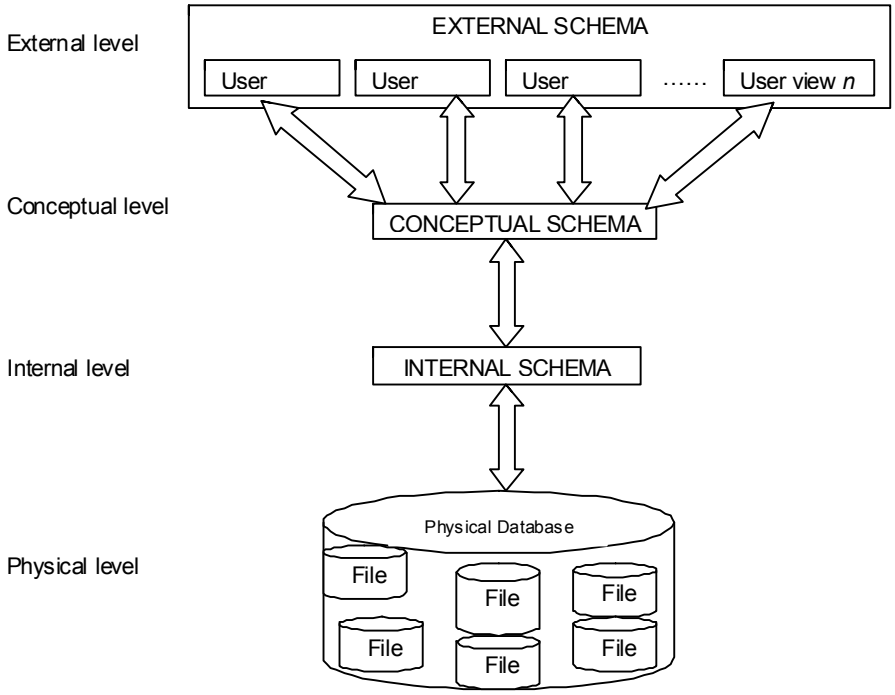


Figure 2.3: Three-tier Database Architecture

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.

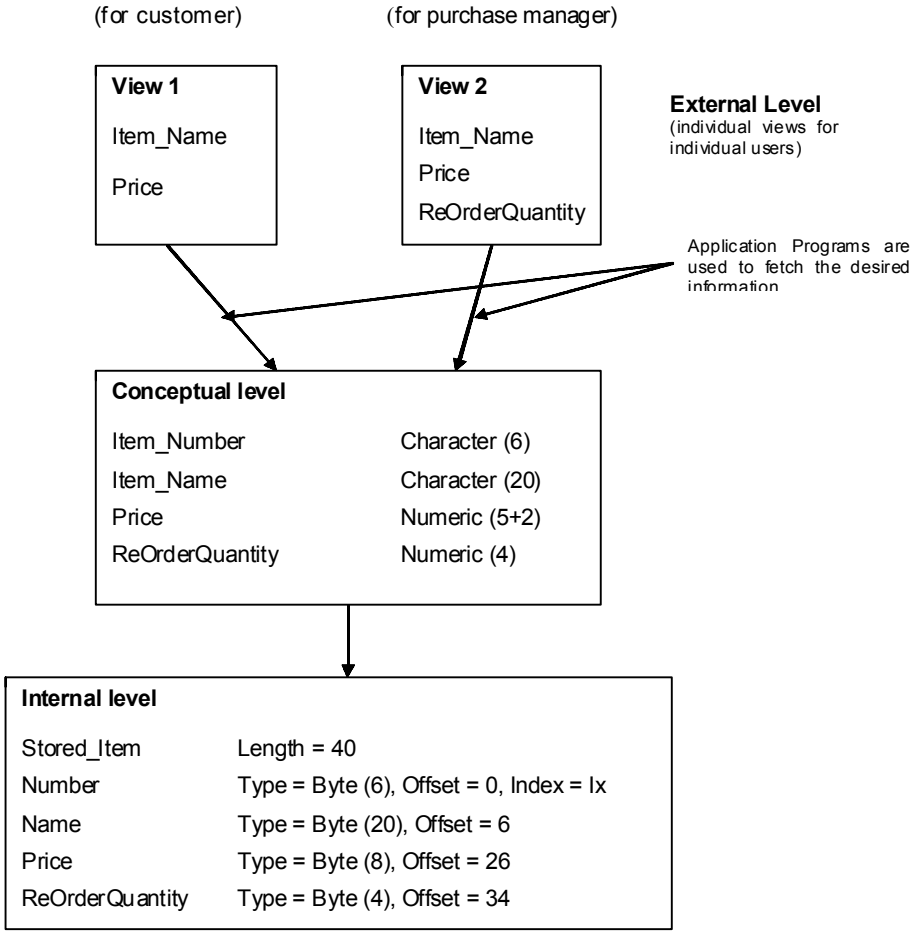


Figure 2.4: View of data at Three-tier Database Architecture

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).

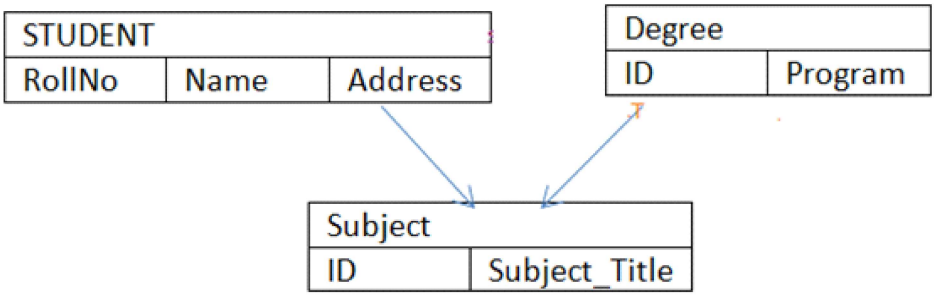


Figure 3.1: Example of a Network Model

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.

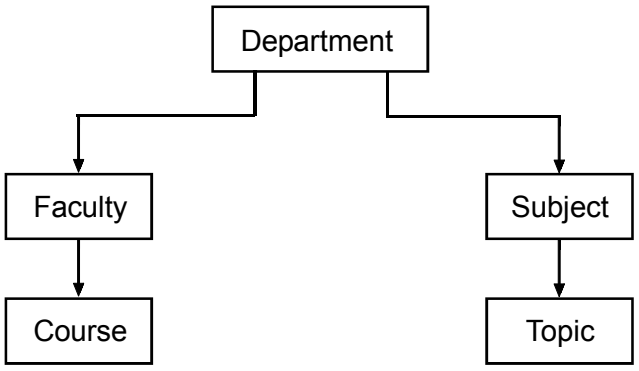


Figure 3.2: Example of a Hierarchical Model

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 Answers 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.

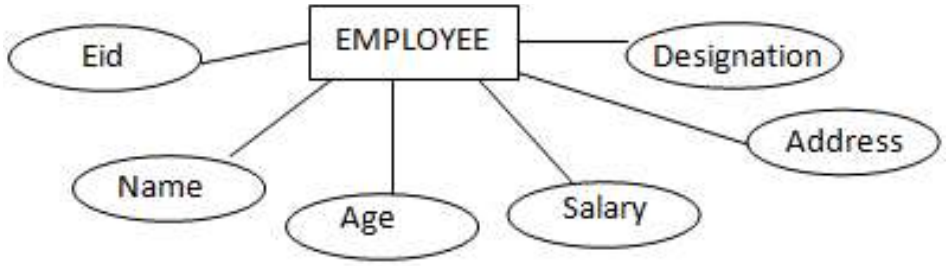


Fig. 4.1(a): EMPLOYEE entity type with six different attributes

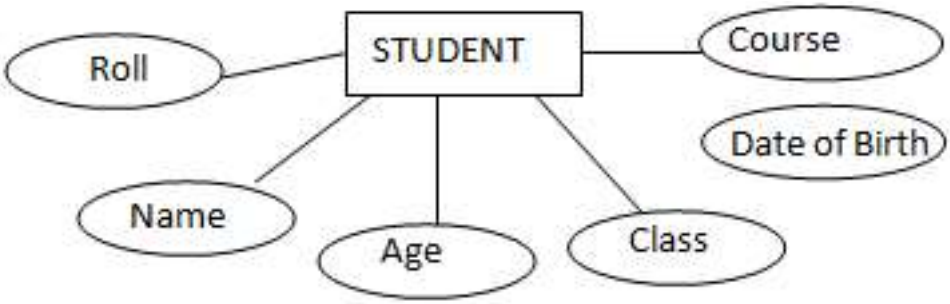


Figure 4.1(b): STUDENT entity type with six different attributes

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)

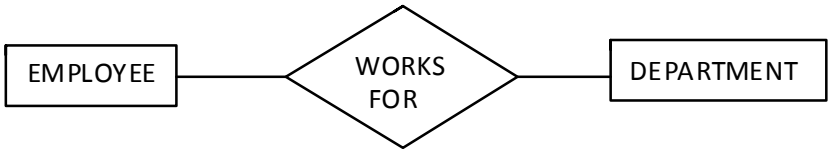


Figure 4.2 (a): binary relationship WORKS_FOR

b)

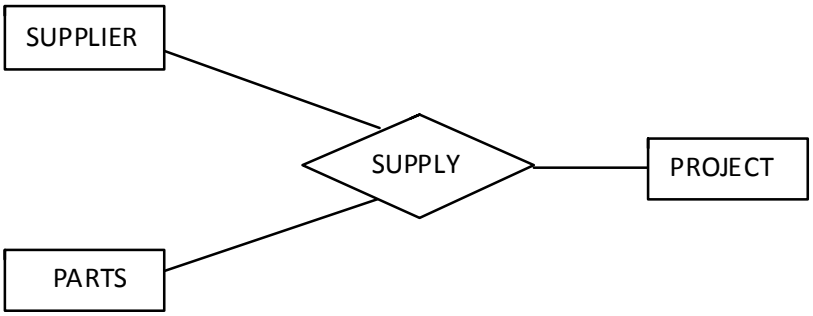


Figure 4.2 (b): Ternary relationship SUPPLY

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.**

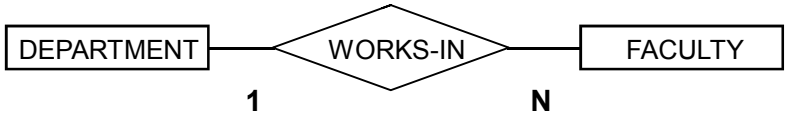
- **One-to-one (1:1):** This is the relationship where an entity in an entity type 'A' is associated with at most one entity of the entity type 'B' and an entity in entity type 'B' is associated with at most one entity in entity type 'A'. For example, the relationship between employee and department.



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.

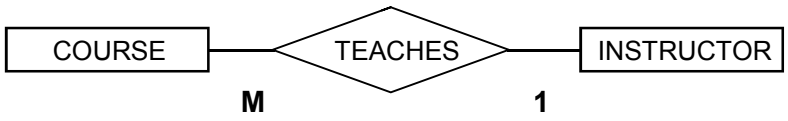
Similarly we can define the relationship between college and principal because we know that one college can have at most one principal and one principal can be assigned to only one college.

- **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.

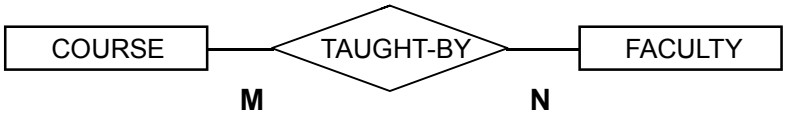
- **Many-to-one (m:1):** In this type of ratio, an entity in entity type 'A' is associated with at most one entity in entity type 'B' or an entity in entity type 'B' is associated with any number of entity in entity type 'A'. For example, the relationship between course and instructor.



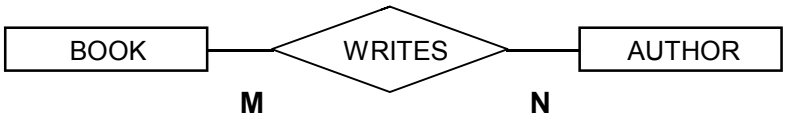
It is assumed that an instructor can teach various courses but a course can be taught only by one instructor.

- **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.

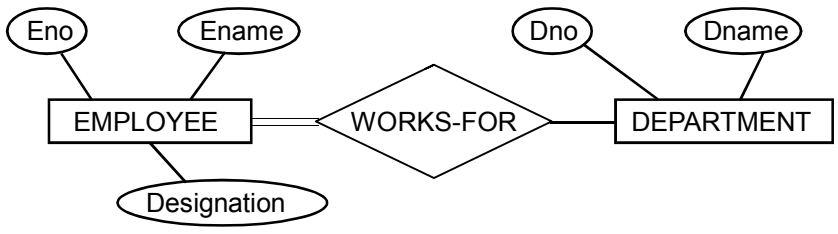


Figure 4.3: Total and Partial participation constraints

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)

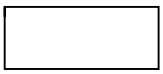
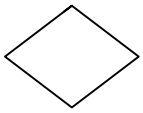
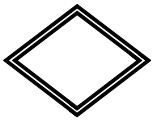
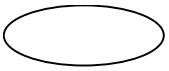
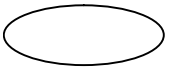
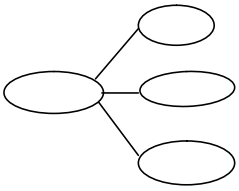
n_ary Relationship: For each n_ary relationship type R where n>2, we create a new table S to represent R. We include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n_ary relationship type (or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

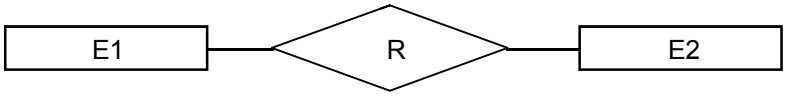
Multi valued attributes: For each multi valued attribute 'A', we create a new relation R that includes an attribute corresponding to the primary key attribute k of the relation that represents the entity type or relationship type that has as an attribute. The primary key of R is then combination of A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

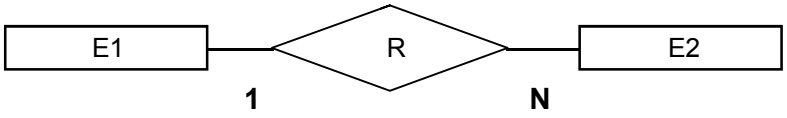
4.5 ER MODELLING SYMBOLS

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.

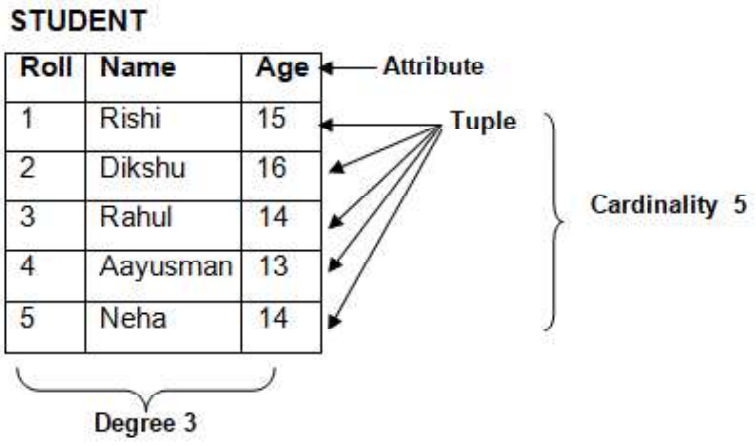


Figure 5.1: An example of a relation "STUDENT"

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:

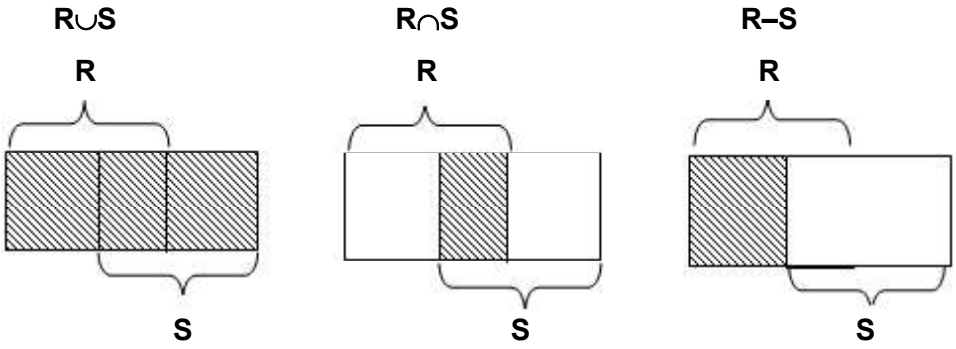
R

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	Blno
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	Blno	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 $R \div S$. 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 a and Deleting a Database
- 7.6.2 Creating a Table
- 7.6.3 Altering a Table
- 7.6.4 Deleting a Table
- 7.6.5 Inserting Data in Table
- 7.6.6 Updating Data in Table
- 7.6.7 Deleting Data from Table

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 Data Type

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

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 design.

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.

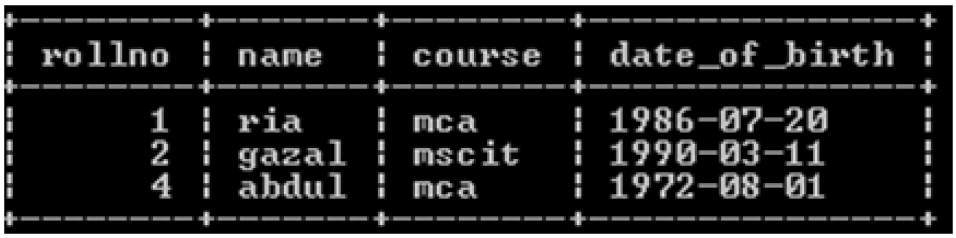
8.3.1 Select

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:



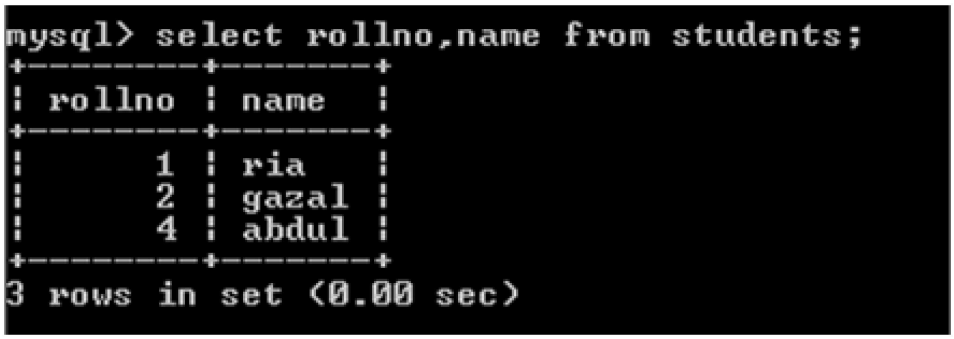
rollno	name	course	date_of_birth
1	ria	mca	1986-07-20
2	gazal	mscit	1990-03-11
4	abdul	mca	1972-08-01

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.

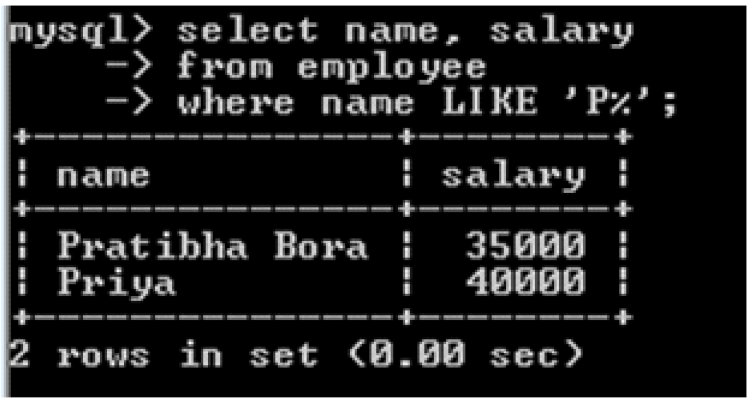


Figure 8.6: Example showing use of like clause

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.

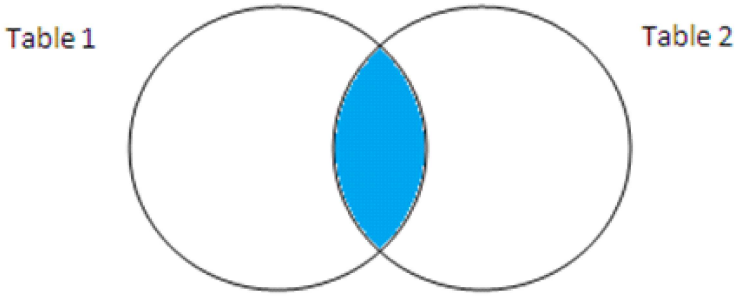


Figure 8.12: Inner Join

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.

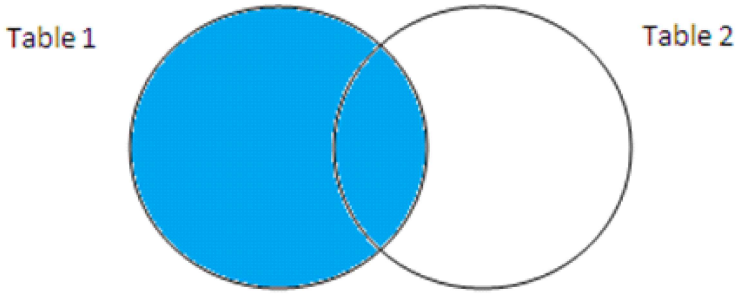


Figure 8.16: Left (Outer) Join

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.

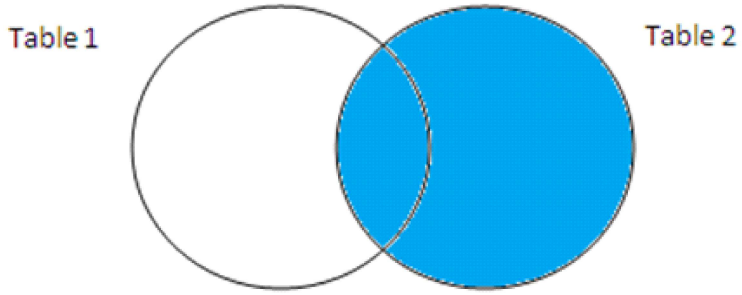


Figure 8.18: Right (Outer) Join

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.

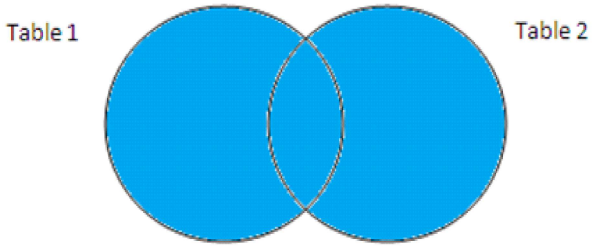


Figure 8.20: Full (Outer) Join

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.**c**olumn, table2.column
from *table1*
WHERE condition1 **IN**
(SELECT table1.**c**olumn, 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.**c**olumn, 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”.

*** ***** ***

UNIT 9: DATABASE DESIGN

UNIT STRUCTURE

9.1 Learning Objectives
9.2 Introduction
9.3 Design of Relational Database
9.4 Functional Dependency
9.5 Functional Dependency Diagram
    9.5.1 Rules for Functional Dependency Diagram
9.6 Let Us Sum Up
9.7 Further Reading
9.8 Answers to Check Your Progress
9.9 Model Questions

9.1 LEARNING OBJECTIVES

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

- learn about the design of a relational database
- describe functional dependency, functional dependency diagram and its rules.

9.2 INTRODUCTION

In the previous unit we have learned some advanced SQL commauds. Different types of join as well as constraints are also discussed in the previous unit

This unit is on database design. We will learn various important concepts like functional dependency, rules for drawing functional dependency diagram etc. Next unit is about database normalization.

9.3 DESIGN OF RELATIONAL DATABASE

We have already learned various terms of relational data model and concept of keys in Previous units These concepts are prerequisite to learn relational database design. A relational database organizes data in tables (or relations). A table is made up of rows and columns. A row is also called

a record (or tuple). A column is also called a field (or attribute). A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store huge amount of data, and effectively retrieve selected data. A language called SQL (Structured Query Language) was developed to work with relational databases. A well-designed database shall:

- **Eliminate Data Redundancy:** The same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- **Ensure Data Integrity and Accuracy:** Relational database design (RDD) models information and data into a set of tables with rows and columns. Each row of a relation/table represents a record, and each column represents an attribute of data. The Structured Query Language (SQL) is used to manipulate relational databases. The design of a relational database is composed of four stages, where the data are modeled into a set of related tables. These four stages are as follows:
  i) Relations and attributes: The various tables and attributes related to each table are identified. The tables represent entities, and the attributes represent the properties of the respective entities.
  ii) Primary keys: The attribute or set of attributes that help in uniquely identifying a record is identified and assigned as the primary key
  iii) Relationships: The relationships between the various tables are established with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. The types of relationships that can exist between the relations (tables) are: One-to-one, One-to-many, Many-to- many. An entity-relationship diagram can be used to depict the entities, their attributes and the relationship between the entities in a diagrammatic way.
  iv) Normalization: This is the process of optimizing the database structure. Normalization simplifies the database design to avoid redundancy and confusion.

By applying a set of rules, a table is normalized into the above normal forms in a linearly progressive fashion. The efficiency of the design gets better with each higher degree of normalization.

9.4 FUNCTIONAL DEPENDENCY

Functional dependency is a term derived from mathematical theory, which states that for every element in the attribute (which appears on some row), there is a unique corresponding element (on the same row). A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the *determinant* and the other attribute is called the *determined.* For each value of the determinant there is associated one and only one value of the determined.

If A is the determinant and B is the determined then we say that A functionally determines B and it is graphically represented as A→B. The following table 9.1 illustrates A→B since for each value of A there is associated one and only one value of B.

Table 9.1: A functionally determines B

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Again, the table 9.2 illustrates that A does not functionally determine B.

Table 9.2: A does not functionally determine B

A	B
1	1
2	4
3	9
4	16
3	10

Since, for A = 3, there is associated more than one value of B.

Functional dependency can also be defined as follows:

An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Let us consider the database having the following tables:

Table 9.3: Supplier table

Sno	Sname	Status	City
S1	Suneet	20	Guwahati
S2	Ankit	10	Tezpur
S3	Amit	10	Tezpur

Table 9.4: Part table

Pno	Pname	Color	Weight	City
P1	Nut	Red	12	Guwahati
P2	Bolt	Green	17	Tezpur
P3	Screw	Blue	17	Tezpur
P4	Screw	Red	14	Guwahati

Table 9.5: Shipment table

Sno	Pno	Qty
S1	P1	270
S1	P2	300
S1	P3	700
S2	P1	270
S2	P2	700
S3	P2	300

Here, in ***Supplier*** table (Table 9.3)

Sno – Supplier number of supplier that is unique

Sname – Supplier name

City – City of the supplier

Status – Status of the city e.g., A grade cities may have status 10, B grad cities may have status 20 and so on.

Here, ***Sname*** is functionally dependent on ***Sno***. Because, ***Sname*** can take only one value for the given value of ***Sno*** (e.g., S1) or in other words there must be one ***Sname*** for supplier number S1.

Functional dependency (FD) is shown by the symbol → which means that ***Sname*** is functionally dependent on ***Sno***.

Sno→Sname

Similarly, ***City*** and ***Status*** are also functionally dependent on ***Sno***, because for each value of ***Sno*** there will be only one City and Status.

Similarly, in case of ***Shipment*** table, the attributes are:

Sno – Supplier number of the supplier

Pno – Part number supplied by supplier

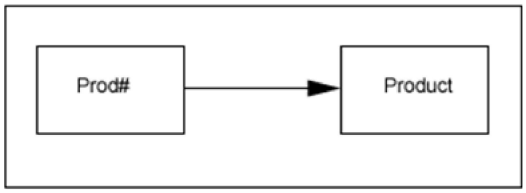
Qty – Quantity supplied by supplier for a particular Part no

In this case, **Qty** is functionally dependent on combination of ***Sno, Pno*** because each combination of **Sno** and **Pno** results only for one Quantity.

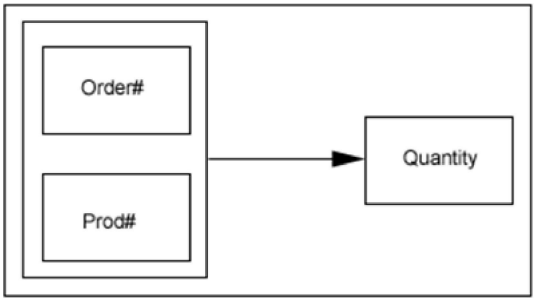
Shipment (Sno, Pno) → Shipment.QTY

9.5 FUNCTIONAL DEPENDENCY DIAGRAM

A set of Functional Dependencies for a data model can be documented in a Functional Dependency Diagram. In a Functional Dependency Diagram each attribute is shown in a rectangle with an arrow indicating the direction of the dependency. The figure below illustrates the functional dependency **Prod# > Product**.

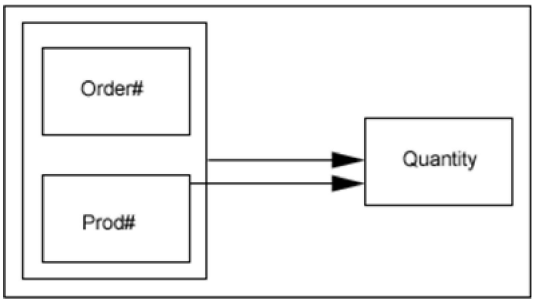


A Functional Dependency with Multiple Attributes is shown below, for the functional dependency **Order#, Prod# > Quantity**.



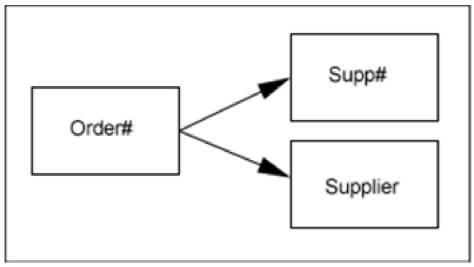
A derived Functional Dependency involving **Partial Key Dependency** is shown in the following figure.

The arrow connected to the outer rectangle, which represents **Order#, Prod# > Product** can be deleted without loss of information.



A derived Functional Dependency involving ***Transitive Dependency*** is shown in the following figure below.

The arrow which represents **Order# > Supplier** can be deleted without loss of information.

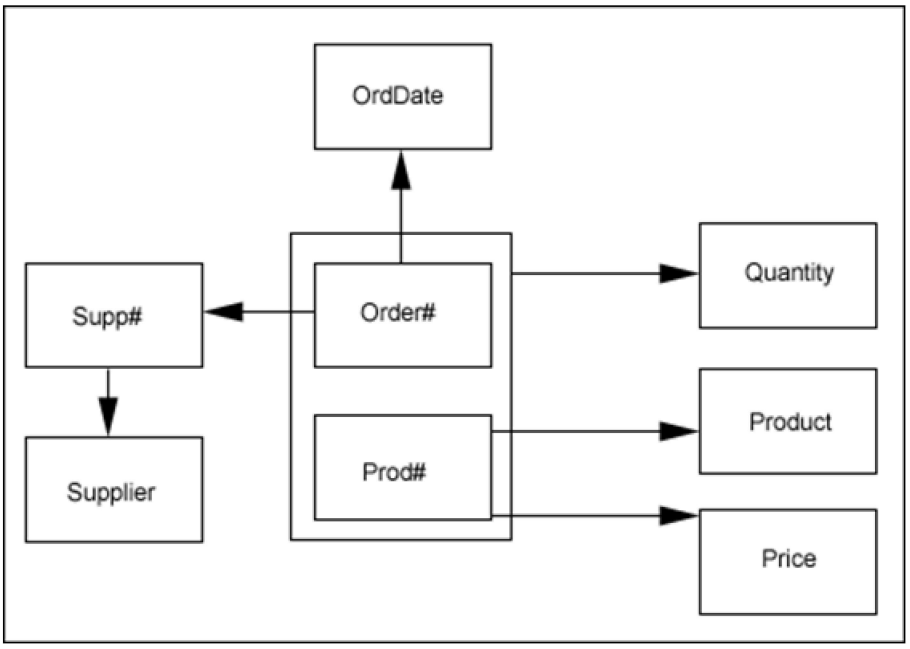


9.5.1 Rules for Functional Dependency Diagram

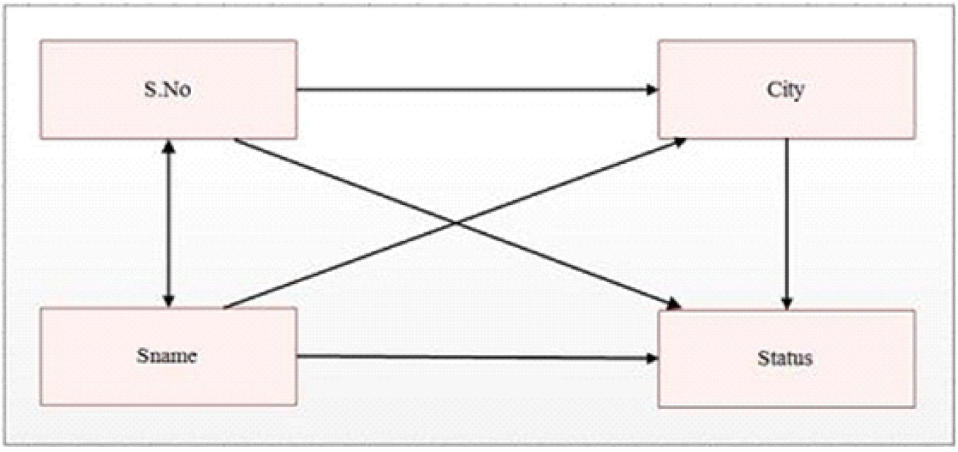
The following rules should be applied to Functional Dependency Diagrams:

- each attribute appears only once on the Functional Dependency Diagram
- all the attributes of interest appear on the Functional Dependency Diagram
- no partial key dependencies appear on the Functional Dependency Diagram
- no transitive dependencies appear on the Functional Dependency Diagram

The complete Functional Dependency Diagram for the *Purchase Order* data model is shown below:



A dependency diagram consists of the attribute names and all functional dependencies in a given table. The dependency diagram of Supplier table is:



Here, following functional dependencies exist in ***Supplier*** table:

Sno→Sname

Sname→Sno

Sno→City

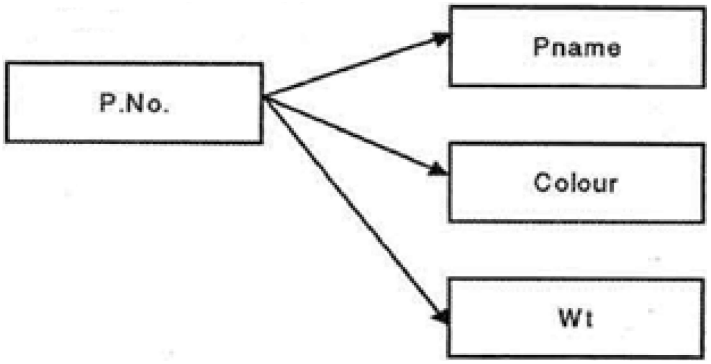
Sno→Status

Sname→City

Sname→Status

City→Status

The FD diagram of relation ***Part*** is:



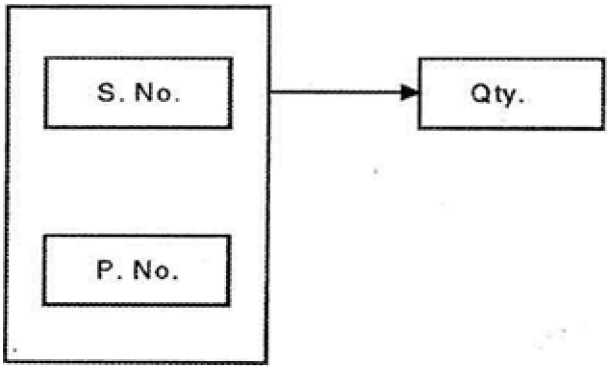
Here, following functional dependencies exist in ***Part*** table:

Pno→Pname

Pno→Color

Pno→Wt

The FD diagram of relation ***Shipment*** is:



Here, following functional dependencies exist in ***Parts*** table:

Shipment(Sno, Pno)→Shipment.QTY



CHECK YOUR PROGRESS

Q.1: Define functional dependency?

Q.2: Consider the following table:

EmpID	EmpName	Age
E101	Kaveri	27
E102	Arjun	25
E103	Bikram	29

Do you see any functional dependence between any two columns of the table?



9.6 LET US SUM UP

- A well-designed database shall eliminate data redundancy, ensure data integrity and accuracy.
- A functional dependency is an association between two attributes of the same relational database table.
- We can define functional dependency for two columns A and B of a table, as follows: Column B is said to be functionally dependent on column A if, given A we can precisely determine B.
- Dependency preservation and lossless join property are important factor for database design.



9.7 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.



9.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called the determined.

Ans. to Q. No. 2: Let us consider a combination of the **EmpID** and the **EmpName** columns. Again consider the following two statements:

- Given an EmpID, we can certainly determine a precise EmpName

- Given an EmpName, we can certainly determine a precise EmpID.

We can observe that the first statement is true, but the second one is not. This is because, EmpID is the primary key of the table and therefore, always unique. However, EmpName can repeat and, therefore, can be a duplicate in the table. Thus, we can determine with confidence an employee's name, given her employee ID. However, the reverse is not true.

Thus, **EmpName** is functionally dependent on **EmpID**. Symbolically, this is written as **EmpID→EmpName**

But, **Age NOT→EmpID** (because more than one employee can have the same age)



9.9 MODEL QUESTIONS

Q.1: Explain the concept of functional dependency.

Q.2: What is functional dependency diagram?

Q.3: In the following column combinations, identify the functional and transitive dependencies:

a) ItemID, ItemName, ItemPrice, QuantitySold, BillAmount

b) RollNo, StudentName, Address, Subject, Marks.

*** ***** ***

UNIT 10: NORMALIZATION OF DATABASE

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 Normalization and Its Objectives

10.4 Normal Forms

- 10.4.1 First Normal Form (1NF)
- 10.4.2 Second Normal Form (2NF)
- 10.4.3 Third Normal Form (3NF)

10.5 Boyce-Codd Normal Form (BCNF)

10.6 Fourth Normal Form (4NF)

10.7 Fifth Normal Form (5NF)

10.8 Let Us Sum Up

10.9 Further Reading

10.10 Answers to Check Your Progress

10.11 Model Questions

10.1 LEARNING OBJECTIVES

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

- learn about normalization
- list out the objectives of normalizations
- describe first, second and third normal forms
- illustrate BCNF, fourth and fifth normal forms.

10.2 INTRODUCTION

In the previous units, we have acquainted you with two important concepts in database design i.e., functional dependency and decomposition which usually help in minimizing the redundancy in database design.

In this unit we will discuss normalization which is one of the most important concept in database. Normalization is the process of efficiently organizing data in a database. Normalization is the name given to the process of simplifying the relationship among data elements in a record. In this unit we will discuss different normal forms. Learners will be able to normalize database tables into different normal forms after going through this unit.

10.3 NORMALIZATION AND ITS OBJECTIVES

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

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

- minimizing redundancy,
- minimizing insertion, deletion and update anomalies.

The process of normalization was first proposed by ***E.F. Codd***. Normalization is a bottom up design technique for relational database. The objectives of the normalization process are as follows:

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

10.4 NORMAL FORMS

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

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

10.4.1 First Normal Form (1NF)

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

Let us consider the relation TRAVEL_INFO as shown in Table 10.1.

Table 10.1: Un-normalized relation TRAVEL_INFO

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

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

Table 10.2 Modified relation TRAVEL_INFO

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

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

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

Table 10.3: Relation PATIENT_DOCTOR

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

From the relational table we have observed that a doctor cannot have two simultaneous appointments so D_NAME and DATE_TIME is a compolsite key. Similarly, a patient cannot have

same time from two different doctors. Therefore, P_NAME and DATE_TIME attributes are also a candidate key.

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

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

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

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

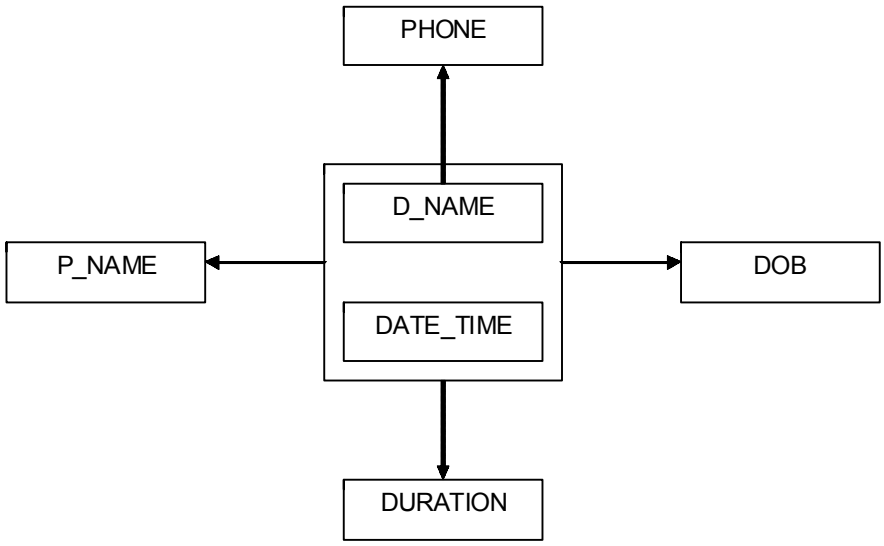


Figure 10.1: FDD of relation PATIENT_DOCTOR

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

10.4.2 Second Normal Form (2NF)

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

a) it is in 1NF,

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

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

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

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

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

DOCTOR (D_NAME, PHONE)

Table 10.4: Relation PATIENT_DOCTOR

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

Table 10.5 Relation DOCTOR

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

The functional dependency diagram of the above two relations are shown below:

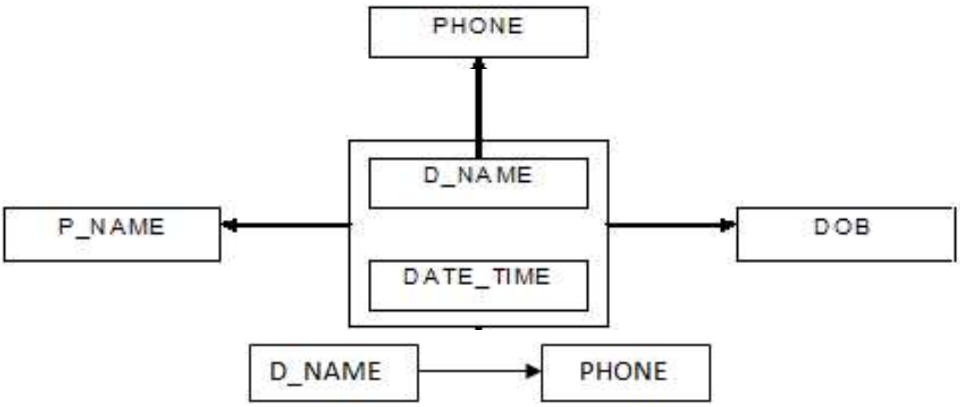


Figure 10.2: FFD of relation PATIENT_DOCTOR

Problems in 2 NF:

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

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

10.4.3 Third Normal Form (3NF)

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

a) mutually independent,

b) functionally dependent on the primary key.

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

In our example, in the relation PATIENT_DOCTOR (Table 10.4), there is no dependency between the attributes P_NAME and DURATION. Again, P_NAME and DOB are mutually dependent. So, the relation is not in 3NF.

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

PATIENT_DOCTOR(P_NAME, D_NAME, DATE_TIME, DURATION)

DOCTOR (D_NAME, PHONE)

PATIENT (P_NAME, DOB)

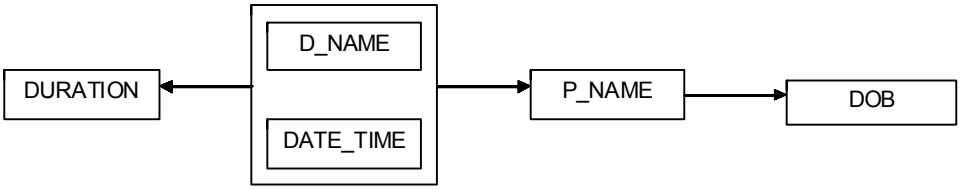


Figure 10.3: FDD for relation PATIENT_DOCTOR

Now, let us consider another relation STUDENT as shown in Table 10.6.

Table 10.6: STUDENT relation

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

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

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

Roll_No $\rightarrow$ **Year**

Year $\rightarrow$ **Hostal_Name**

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

STUD1 (Roll_No, Name, Dept)

STUD2 (Year, Hostel_Name)

Table 10.7: Relation STUD1

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

Table 10.8: Relation STUD2

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

In the above examples, the conversion into 3NF is not hard, but whenever a relation has more than one combination of attributes that may be considered as primary key then the conversion takes number of steps. Let us consider the following relation UTILIZE as shown in Table 10.9.

Table 10.9: Relation UTILIZE

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

The relation UTILIZE stores the machines information used by both projects and project managers. Each project has one project manager and each project manager manages one project. Now, it is obvious from the table that , we can consider any one of the combination of attributes as primary key, namely, {Project, Machine} or {Proj_Manager, Machine}. The FDD for relation UTILIZE is shown figure 10.4.

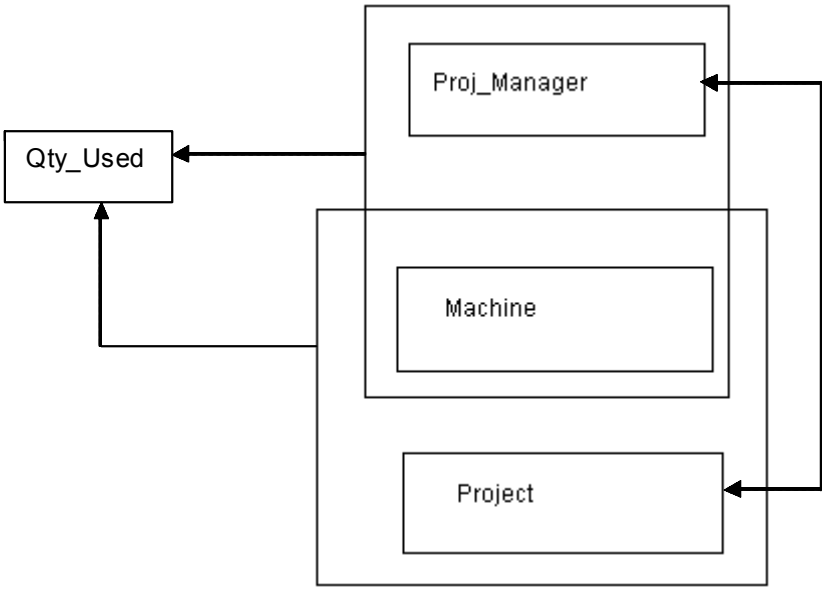


Figure 10.4: FDD of UTILIZE

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

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

a) The project manager of each project is stored more than once.
b) A project manager cannot be stored until the project has ordered some machines.
c) A project cannot be entered unless that project's manager is known.
d) If a project's manager changes, some rows also must be changed.

10.5 BOYCE-CODD NORMAL FORM (BCNF)

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

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

$$X \rightarrow Y$$

Here, we call X the determinant. From this, we can define BCNF as follows:

A table is in Boyce-Codd normal form (BCNF) if the only determinants in the table are the candidate keys.

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

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

From the above conditions we have come to know that, a relation must only have candidate keys as determinants. Any relation in BCNF is also in 3NF and consequently in 2NF. However, a relation in 3NF is not necessarily in BCNF.

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

10.6 FOURTH NORMAL FORM (4NF)

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

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

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

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

Table 10.10: STUDENT relation

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

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

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

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

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

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

Table 10.11: Relation Student_Subject

Student	Subject
Geeta	Mythology
Geeta	Psycology
Sikhar	Gardening

Table 10.16: Relation Student_Language

Student	Language
Geeta	English
Geeta	Hindi
Sikhar	English

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

10.7 FIFTH NORMAL FORM (5NF)

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

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



CHECK YOUR PROGRESS

Q.1: Select the correct answer from the following:

a) Normalization is a process of–

i) decomposing of a set of relation.

ii) successive reduction of relation schema.

iii) deciding which attributes in a relation to be grouped together.

iv) all of these.

b) A normal form is–

i) a state of a relation that results from applying simple rules regarding functional dependeny.

ii) the highest normal form condition that it meets.

iii) an indication of the degree to which it has been normalized.

iv) all of these.

c) In 1NF–

i) all domains are simple.

ii) in a simple domain, all elements are atomic

iii) both (i) & (ii) iv) none of these

d) 2NF is always in–

i) 1NF ii) BCNF

iii) MVD iv) none of these

e) A relation R is said to be in 2NF,

i) if it is in 1NF.

ii) every non-prime key attributes of R is fully functionally dependent on each relation key of R.

iii) if it is in BCNF. iv) both (i) & (ii).

f) A relation R is said to be in 3NF if the,

i) relation R is in 2NF

ii) non prime attributes are mutually independent.

iii) functionally dependent on the primary key.

iv) all of these.

g) 4NF is concerned with dependencies between the elements of compound keys composed of,

i) one attributes ii) two attributes

iii) three or more attributes iv) none of these

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

a) ..... first developed the process of normalization.

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

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

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

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

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

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

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



10.8 LET US SUM UP

- Normalization is a design technique that is widely used in designing relational model. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table).
- Functional dependencies play an important role in differencing good database design from not so good database design.
- A relation is said to be in First Normal Form (1NF) if only if every entry of the relation (the insertion of a tuple and a column) has at most a single value. In other words “a relation is in First Normal Form if only if all underlying domains contain atomic values or single value only”.
- A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.
- A relation or table is said to be in third normal form (3NF) if the relation is in 2NF and the non-prime attributes are mutually independent and functionally dependent on the primary key.
- A relation R is in 4NF if and only if the following conditions are satisfied simultaneously: (i) R is already in 3NF or BCNF. (ii) If it contains no multi-valued dependencies.
- A relation (or table) R is said to be in BCNF if for every nontrivial FD : X $\rightarrow$ Y between attributes X and Y holds in R.
- A table is in the fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.
- A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.



10.9 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.



10.10 ANSWERS TO CHECK YOUR PROGRESS

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

Ans. to Q. No. 2: a) E. F. Codd, b) atomic, c) composite, attribute, d) another non prime attributes, e) X $\rightarrow$ Y, f) candidate key, g) 3NF, 2NF, h) multi-valued dependencies.



10.11 MODEL QUESTIONS

Q.1: What is meant by normalization? Why do we normalize the database? Describe the data normalization process.

Q.2: What are different normal forms?

Q.3: Define 1NF, 2NF and

Q.4: How do we convert a database in 1NF to 2NF?

Q.5: How do we convert a database in 2NF to 3NF?

Q.6: Explain 3NFwith example.

Q.7: Define BCNF. How does it differ from 3NF ? Why is BCNF stronger than 3NF? Illustrate with an example.

Q.8: Consider a relation:

Student(Student_Id, Student_Name, Student_Address, Date_of_Birth, Course_Id, Course_Name, Duration, No_of_Papers)

The functional dependencies are:

Student_Id-> Student_Name

Student_Id->Student_Address

Student_Id->Course_Id

Student_Id->Date _of_Admission

Student_Id->Date_of_Birth

Course_Id->Course_Name

Course_Id->Duration

Course_Id->No_of_Papers

Decompose the above relation into equivalent Third Normal Form (3NF) relations.

Q.9: Following relation for published books is given:

BOOK(BOO-TITLE, AUTH-NAME, BOOK-TYPE, LIST-PRICE, AUTH-AFFL, PUBLISHER)

Here, AUTH-AFFL refers to the affiliation of author. Suppose that the following FDs exists:

BOOK-TITLE $\rightarrow$ PUBLISHER, BOOK-TYPE

BOOK-TYPE $\rightarrow$ LIST-PRICE

AUTH-NAME $\rightarrow$ AUTH-AFFL

a) What normal form is the relation in ? Explain your answer.

b) Apply normalization until the relations cannot be decomposed any further. State the reason behind each decomposition.

Q.10: Explain the following terms with an example each :

a) Normalization b) 1NF c) BCNF

d) 3NF e) 4NF f) 5NF

g) MVD h) Candidate key i) Primary Key

j) Super Key

*** ***** ***

UNIT 11: SYSTEM IMPLEMENTATION TECHNIQUES

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Query Processing

- 11.3.1 Basic steps in Query Processing
- 11.3.2 Translating SQL Queries into Relational Algebra
- 11.3.3 Query Tree
- 11.3.4 Measure of Query Cost

11.4 Query Algorithms

- 11.4.1 Selection Algorithms
- 11.4.2 Join Algorithms
- 11.4.3 Other Operations

11.5 Evaluation of Query Expressions

11.6 Transformation Rules for Relational Algebra Expressions

11.7 Methods of Query Optimization

11.8 Let Us Sum Up

11.9 Further Readings

11.10 Answers to Check Your Progress

11.11 Model Questions

11.1 LEARNING OBJECTIVES

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

- define a Query
- learn about Query Processor
- learn about different steps involved in Query Processing
- learn about parsing and translation the Query
- learn about translation between SQL queries and Relational Algebra
- learn the different query processing algorithms such as *selection*, *join* etc.

- describe the different transformation rules used for equivalent query transformation
- learn about the importance of Query Optimization and different methods of query optimization.

11.2 INTRODUCTION

We are already familiar with SQL query and Relational Algebra. We are already acquainted about different parts, components and query structures used in SQL such as DDL, DML statements, WHERE clause, Aggregation Function etc. We are also familiar with different relational algebraic expressions such as SELECT, PROJECT, DIFFERENCE, INTERSECTION, NATURAL JOIN etc. The execution and processing of query is the fundamental part of any DBMS system.

In this unit, we will learn about Query processing and Optimization of Query Processing. In the next unit, we will explore transaction processing in database system and we will learn about transaction and system concepts, desirable properties, schedules and Recoverability.

11.3 QUERY PROCESSING

Query processing is the process of transforming the high level language into low level language which can be used at the physical level of the file system, query optimization and perform the requested action for user.

11.3.1 Basics Steps in Query Processing

The fundamental steps involved in processing a query processing are *parsing and translation, optimization* and *Evaluation* as shown in figure 11.1.

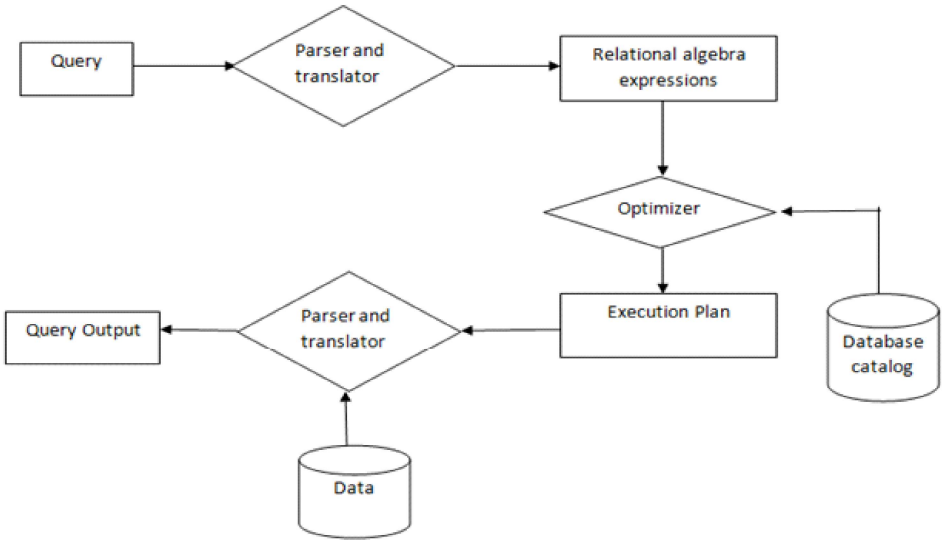


Figure 11.1: Steps in Query Processing

The *parsing and translation* is the first step of any query processing. The parser in the query processor checks the syntax of the query and validity of the relations (i.e. tables) and attributes which are used in the query. The *translator* converts each token into relational expressions, *query trees* and *query graphs*.

In the *optimization* step, the query processor applies rules to the internal data structures of the query to transform these structures into equivalent expressions (i.e. execution plan) that are more efficient to execute. The optimizer uses the statistical data i.e. information about the size of the table, the length of records, the indexes created on the table, etc. for finding out the various evaluation plans. The Optimizer also checks for the conditions and conditional attributes which are parts of the query. The different evaluation plans for a given query can have different cost in terms of time consumption to execute the query, or the Space required for executing the query. The query optimizer is responsible to find out the most efficient evaluation plan for the execution of the given query.

The final step in processing a query is the *evaluation phase*. In this phase the query evaluation engine selects the best evaluation plan generated by the query optimization engine, executes that plan and returns the answer to that query.

For example: Consider for example following SQL query:

SELECT BALANCE **FROM** ACCOUNT **WHERE** BALANCE < 2000.

This can be translated into either of the following relational algebra expressions:

- $\sigma_{BALANCE<2000}(\Pi_{BALANCE}(ACCOUNT))$
- $\Pi_{BALANCE}(\sigma_{BALANCE<2000}(ACCOUNT))$

This can also be represented as either of the following query trees:

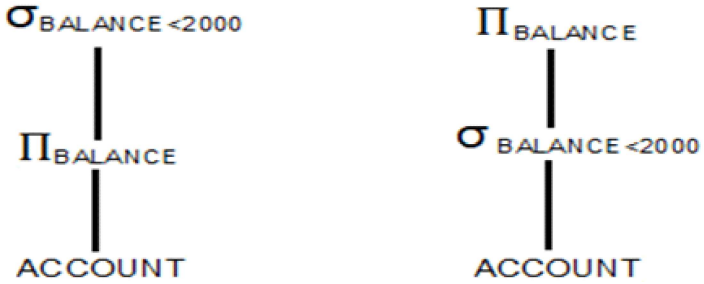


Figure 11.2: Query Evaluation Plan

The job of the query optimizer is to select the best query evaluation plan and execute it. The selection of the evaluation depends on the cost of execution of evaluation plan.

11.3.2 Translating SQL Queries into Relational Algebra

An SQL query is first translated into an equivalent relational algebra expression which is represented as a query tree data structure. The transformed data-structure is then optimized for efficient execution. An SQL query may be a composition of many sub-queries and are decomposed into *query blocks* which form the basic units. The query blocks can be translated into algebraic operators and then optimized by the query optimizer. A query block can contains a single SELECT-FROM-WHERE expression as well as clauses like GROUP BY and HAVING if these are part of the block. If there are nested queries within a query then they are identified as a separate query block.

Let us consider the EMPLOYEE relation of COMPANY relational database schema as shown in fig. 11.3.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATION

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Fig. 11.3: COMPANY relational database schema

Now, consider the following SQL query on EMPLOYEE relation in figure 11.3.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Salary > (SELECT MIN(Salary)
                FROM EMPLOYEE
                WHERE Dno=6);
```

This query includes a nested sub-query and hence would be decomposed into two query blocks: *inner block* and *outer block*. The inner block is:

(SELECT **MIN** (Salary) **FROM** EMPLOYEE **WHERE** Dno=5);

and the outer block is

(**SELECT** Fname, Lname **FROM** EMPLOYEE **WHERE** Salary>x)

Where, x represents the result returned from the inner block. The inner block could be translated into the extended relational algebra expression

$$F_{MAX\ Salary}(\sigma_{Dno=6}(EMPLOYEE))$$

And the outer block into the expression

$\Pi_{Fname,Lname}(\sigma_{Salary>x}(EMPLOYEE))$

Now, the query optimizer chooses a query execution plan for each block. A query execution plan or a query evaluation plan is the sequence of primitive operations that can be used to evaluate a query. Query execution plan minimizes the cost of query evaluation.

11.3.3 Query Tree

A **Query tree** is a data structure that corresponds to a relational algebra expression. In this tree data-structure, the leaf nodes represent the input and the internal node represents the relational algebra operations. In query tree execution, An internal node can be executed whenever its operands are available and Node is replaced by the result of the operation it represents. The root node is executed last, and is replaced by the result of the entire tree.

A **query graph** is the data structure that corresponds to a relational calculus expression. It does not indicate an order on which to perform the operations first. There is only a single graph corresponding to each query.

Let us consider the following SQL query:

SELECT P.Pnumber, P.Dnum, E.Lname **FROM** PROJECT **AS** P, DEPARTMENT AS D, EMPLOYEE AS E **WHERE** P.Dnum = D.Dnumber AND D.Mgr_ssn = E.Ssn AND P.Plocation = 'Delhi';

This can be translated into following relational algebra expression as follows:

$\Pi_{Pnumber, Dnum, Lname}(((\sigma_{Plocation="Delhi"}(PROJECT))$

$\bowtie_{Dnum=Dnumber}(DEPARTMENT))\bowtie_{Mgr_ssn = Ssn}(EMPLOYEE))$

The above query can be represented by the Query tree as shown in figure 11.4(a) where, the three relations PROJECT, DEPARTMENT and EMPLOYEE are represented by leaf nodes P, D and E, while the relational algebra operations of the expressions are represented by internal tree nodes. When this query is executed,

the nodes marked (1) must be executed before the node marked (2) because some resulting tuples of operation (1) must be available to the operation (2) to start with. Similarly, node (2) must begin executing and producing results before node (3) can start execution.

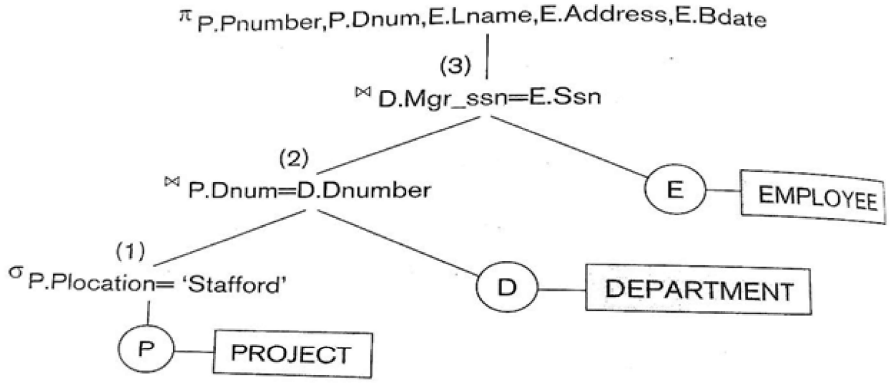


Figure 11.4 (a): Query Tree

The query graph representation of the above query is:

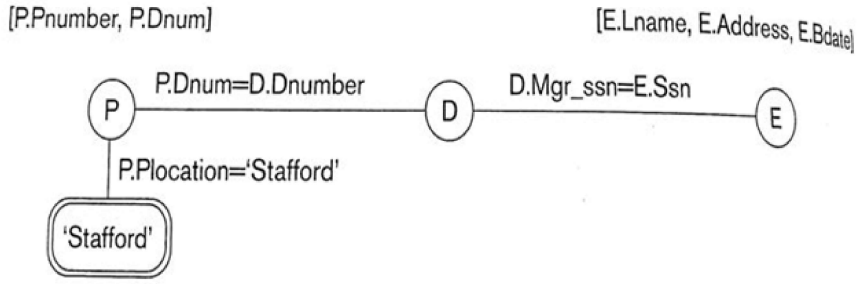


Figure 11.4 (b): Query Graph representation of the above expression

EXERCISE 11.1



Draw the Query tree for the following Relational algebra expression:

SELECT Lname **FROM** EMPLOYEE, WORKS_ON, PROJECT **WHERE** P_name='Aquarius' **AND** P_number=Pno **AND** Essn=Ssn **AND** Birth_date>'1957-12-31'

11.3.4 Measure of Query Cost

Cost of query is the time taken by the query to hit the database and to return the result. It involves query processing time i.e.; time taken to parse and translate the query, optimize it, evaluate, execute

and return the result to the user is called cost of the query. Many factors can contribute to time cost such as disk accesses, CPU or even network communication. Therefore, the cost of query evaluation can be measured in terms of following:

- *Access cost to secondary storage:* This is the cost of transferring i.e., reading and writing) data blocks between secondary disk storage and main memory buffers. The cost of searching for records in a disk file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. In addition, factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.
- *Disk storage cost:* This is the cost of storing on disk any intermediate files that are generated by an execution strategy for the query.
- *Computation cost:* This is the cost of performing in-memory operations on the records within the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join or a sort operation, and performing computations on field values. This is also known as CPU (central processing unit) cost.
- *Memory usage cost:* This is the cost pertaining to the number of main memory buffers needed during query execution.
- *Cost of communication (in distributed or parallel database system):* This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated. In distributed databases, it would also include the cost of transferring tables and results among various computers during query evaluation.

In large database systems, however, disk accesses are usually the most important cost, since disk accesses are slow compared to in-memory operations. Also CPU speeds have been improving much faster than have disk speeds. Thus, it is likely that the time spent in

disk activity will continue to dominate the total time to execute a query. Finally, estimating the CPU time is relatively hard compared to estimating the disk-access cost. Therefore, most people consider the disk-access cost a reasonable measure of the cost of a query-evaluation plan.

11.4 QUERY ALGORITHMS

To query a table, there are different ways to query it to get the same result. The query execution engine can have a large number of specialized algorithms designed to process particular relational operation and access path combinations. Let us discuss some examples of algorithms for various operations.

11.4.1 Selection Algorithms

The SELECT operation can be performed in a number of ways. The SELECT operation scans the records of a file to search for and retrieve records that satisfy the selection criteria. The following are some examples of simple selection algorithms.

File Scan: A number of selection algorithms are possible for selecting records from a file. These algorithms are also known as **file scans**, because they scan the records of a file to search for and retrieve records that satisfy a selection condition. Two basic file scan algorithms for selection operation are:

- **S1: Linear Search (brute force):** In linear search, the system scans each file blocks and test each record from the file is to test whether it satisfy the selection criteria. This is sometimes referred to as a *full table scans*.

  Cost of the algorithm (in terms of block transfer) is defined as:

  a) Cost of searching records satisfying a condition=Number of blocks in a database = N_b

  b) Cost of searching a key attribute value=average number of block transfer for locating the value (on average only half of

the blocks would be searched before the specific record is found) = $N_b/2$

- **S2: Binary search on primary key:** it can be used, if the selection condition involves an equality comparison on a key attribute on which the file is ordered. It is more efficient than linear search. Cost of the algorithm: Cost of finding the first tuple using the binary search method + sequence of other blocks that continue to satisfy the condition.

$$\lceil \log_2(N_b) \rceil + \frac{s}{bfr} - 1$$

Where *s* is the average number of tuples with same value and *bfr* blocking factor (number of tuples in a block) of the relation)

Index Scan: This method of selection uses indexes to traverse the record. The search key should be indexed and should be used in the filter condition for an Index scan to work.

- **S3: a) Primary index-scan for equality:** if the selection condition involves an equality comparison on a key attribute with a primary index. e.g.,

  $\sigma_{SSN='123456789'}$(EMPLOYEE),

  where *ssn* a key attribute for the EMPLOYEE. The cost of this method is:

  Cost = (number of index level) + 1 (block of the primary index is traverse to access)

  b) Hash key: it retrieves a single a single record in a direct way. The cost of this method is: Cost = 1

- **S4: Using an ordering index to retrieve multiple records:** if the comparison condition is >, >=, < or <= on a key field with an ordering index, then we need to locate the first record satisfying the condition after which the records are scanned forward or backward as the condition may be, displaying all records. Assuming uniform distribution, then we would expect half the records to satisfy the condition. The cost of this method is:

  Cost = number of index level + $N_b/2$

- **S5: Using a clustering (secondary) index to retrieve multiple records:** if the selection condition involves an equality comparison on a **non-key attribute** with a clustering index. The cost of this method is:

  Cost = number of index level + $\left\lceil \frac{s}{bfr} \right\rceil$

- **S6: Using a secondary (B+ tree) index:**
  a) **On equality condition:** on an equality comparison condition, *s* tuples will satisfy the condition on the indexing attribute. Since, the index is non-clustering, we have to assume that the tuples may reside on different blocks. The estimated cost is: Cost = number of index level + *s*
  b) **On inequality condition:** if the comparison condition is >, >=, <, or <= and assuming uniform distribution, we would expect half the file records to satisfy the condition, then the first level index blocks are accessed, plus the file records via index. The estimated cost of this method is:

  Cost=number of index level + $\left\lceil \frac{b_{I1}}{2} \right\rceil + \left\lceil \frac{r}{2} \right\rceil$

Selection Algorithms for Complex Selection:

Conjunction: Conjunction is basically a set of **AND** operations. e.g. $\sigma_{DNO=5\ \textbf{AND}\ Salary>3000\ \textbf{AND}\ Sex='F'}(EMPLOYEE)$. The DBMS can use following additional methods to implement the operations:

- **S7: Conjunctive Selection using an individual index**: Select any algorithm from S1 to S6 on one or more conditions. If possible, test other conditions on these tuples after fetching them into memory buffer.
- **S8: Conjunctive selection using a composite index**: Use appropriate composite (multiple-key) index if available.
- **S9: Conjunctive selection by intersection of identifies:** Requires indices with record pointers. Use corresponding index for each condition, and take intersection of all the obtained sets

of record pointers. Then fetch records from file if, some conditions do not have appropriate indices, test them in memory after fetching from memory buffer.

Disjunction: Disjunction is basically a set of **OR** conditions. e.g., $\sigma_{DNO=5\ \textbf{OR}\ Salary>3000\ \textbf{OR}\ Sex='F'}(EMPLOYEE)$. *Disjunctive Selection Conditions* are much harder to process and optimize because the records satisfying the disjunctive condition are the union of the records satisfying the individual conditions.

- **S10: Disjunctive selection by union of identifies** is applicable if *all* conditions have available indices, otherwise use linear scan. Use corresponding index for each condition, and take union of all the obtained sets of record pointers, eliminate duplicates, then fetch records from file.



CHECK YOUR PROGRESS

Q.1: Suppose that a *EMPLOYEE* relation as shown in figure 11.3. has r_E = 10,000 records stored in b_E = 2000 disk blocks with blocking factor bfr_E = 5 records/block and following statistics in the system catalog:

- A secondary index on the key attribute *Ssn*, with $X_{Ssn} = 4 (S_{Ssn} = 1)$
- A secondary index on the non-key attribute *Dno* = 2, with X_{Dno} = 2 and first-level index blocks b_{I1Dno} = 4. There are d_{Dno} = 125 distinct values for *Dno,* so the selection cardinality of Dno is $S_{Dno} = (r_E / d_{Dno}) = 5000$

Find out the estimated cost for following operations:

OP1: $\sigma_{Ssn='123456789'}(EMPLOYEE)$

OP2: $\sigma_{Dno>5}(EMPLOYEE)$

OP3: $\sigma_{Dno=5}(EMPLOYEE)$

11.4.2 Join Algorithms

Like selection, the JOIN operation can be implemented in a variety of ways. In terms of disk accesses, the JOIN operations can be very expensive; so implementing and utilizing efficient join algorithms is critical in minimizing a query's execution time. The following are 4 well-known types of join algorithms:

The join operation is of the form: $R \bowtie_{A=B} S$

J1: Nested-Loop Join (brute force): This algorithm consists of an inner for loop nested within an outer for loop. For each record *t* in *R* (outer for loop) retrieve every record *s* from *S* (inner for loop) and tests whether the two records satisfy the join condition t[R]=s[S]. The estimated cost of this approach is: $N_r + (N_r * N_s)$. Where, N_r is the number of block of R has to be read and N_s is the number of block of S has to be read for each block of R. The second term is fixed, but the first term could vary depending on the relation chosen for the outer loop.

J2: Index Nested-Loop Join: if there is an index (or hash function) on the join attributes of the inner relation, then we can replace the inefficient file scan with an index lookup. For each tuple in R, we use the index to retrieve the matching tuple S. The cost of scanning R is N_r, as before. However, the cost of retrieving the matching tuple is S depends on the type of index and number of matching tuples. Let, c be the cost for the lookup, then the worst -case cost for joining rand sis $N_r + n_r * c$, where, n_r is the number of tuples of R. If the join attribute A is the primary key, then c= number of index level + 1.

J3: Sort-Merge Join: For Equijoins, the most efficient join is achieved when both the relations R and S are physically sorted on the join attributes. Both relations R and S are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If R and S are not sorted then, they should be sorted by using external sorting. The estimated cost of this approach is:

$N_r + N_s$ (if already sorted) or $(N_r + N_s + N_r * \log_2 N_r + N_s * \log_2 N_s)$

J4: Hash Join: The records of files *R* and *S* are both hashed to the same hash file, using the same hashing function on the join attributes *A* of R and B of *S* as hash keys. A single pass through the file with fewer records (say, R) hashes its records to the hash file buckets known as *partitioning phase.* A single pass through the other file (S) then hashes each of its records to the appropriate bucket, where the record is combined with all matching records from R in that bucket known as *probing phase*. The estimate cost of this approach is: $3 * (N_r + N_s)$

11.4.3 Other Operations

There are many other operations that are performed in database systems such as *Duplicate Elimination*, *Projection*, *Aggregate Function operation*, *SET operation.*

Duplicate Elimination: There are two methods for the elimination of duplicate tuples: *Sorting* and *Hashing*. In *Sorting*, duplicates in a relation R will be adjacent to each other and may be identified easily and deleted. The estimated cost of this approach is: $N_r + N_r * \log_2 N_r$.

Another optimized method for duplicate elimination is using *Hashing* where, deletion of duplicates are done during generation as well as at intermediate merge steps in external sort-merge. Here, duplicates will be clubbed together in the same bucket and therefore may be deleted easily.

Projection: A PROJECT operation $\Pi_{<\text{attribute lits}>}(R)$ defines a relation S containing vertical subset of a relation R extracting the values of specified attributes and eliminating duplicates. Therefore, it requires two steps:

1) Removal of attributes that are not required
2) Duplicate elimination of tuples (if any)

Aggregate Function Operation: Aggregate functions (MIN, MAX, AVG, COUNT, SUM) can be implemented in a manner similar to duplicate elimination. Sorting or hashing can be used to bring tuples

in the same group together, and then aggregate functions can be applied to each group.

SET Operations: the binary SET operations Union (∪), Intersection (∩), and Set-Difference (–) can be applied to relations that are union-compatible. We can implement these operations by first sorting both relations R and S on the same attributes and then scanning through each of the sorted relations once to obtain the desired result. We can also use hashing algorithm to implement these operations.

In Union operation, $\cup_r = \{t_r \mid t_r \in R \text{ or } t_r \in S\}$

In Intersection operation, $\cap_r = \{t_r \mid t_r \in R \text{ and } t_r \in S\}$

In Set-difference operation, SD_r = {and}



CHECK YOUR PROGRESS

Q.2: Choose the appropriate option:

i) The files from which selection algorithms are selected for records are called:

a) Scan file b) Spanned file

c) Non dense file d) Indexed file

ii) The procedure of choosing a suitable query out of all the queries is classified as:

a) Query optimization b) Parser optimization

c) Processor optimization d) All of the above

iii) The representation of the query in the form of data structure is classified as:

a) Query graph b) Query tree

c) Scanner tree d) Parser tree

iv) The representation of the queries in the form of data structure graph is classified as:

a) Query graph b) Query tree

c) Scanner tree d) Parser tree

11.5 EVALUATION OF QUERY EXPRESSIONS

In the previous section we have seen the algorithm for individual operations. Now, let us discuss the methods for evaluating an entire expression. The obvious way to evaluate an expression is simply to evaluate one operation at a time, in an appropriate order. The result of each evaluation is *materialized* in a temporary relation for subsequent use. A disadvantage to this approach is the need to construct the temporary relations, which (unless they are small) must be written to disk. An alternative approach is to evaluate several operations simultaneously in a *pipeline*, with the results of one operation passed on to the next, without the need to store a temporary relation. In general there are two general methods:

- Materialization
- Pipelining

- **Materialization:** In materialization approach, each operation in the expression is evaluated one by one in appropriate order starting from the lowest-level operations in the expression (at the bottom of the tree). The result of each operation is materialized (created) in a temporary relation which becomes input to the subsequent operations. Let us consider the query tree in fig 11.4(a). In this example, we start at the selection operation on *Project* and a temporary relation is created. At the next level, this temporary relation is taken as input and join operation is performed with *Department* relations and output is stored in a temporary relation. By repeating this process, we get the ûnal result by executing the projection operation at the root of the tree, using as input the temporary relations created by the join. Materialised evaluation is always possible though; the cost of writing/ reading results to/from disk can be high if system has limited buffer size. This problem can be tackled by using *double buffer* (using two buffers, with one continuing execution of the algorithm while the other is being written out). Double buffering allows the algorithm to execute more quickly by performing CPU activity in parallel with I/O activity.

- **Pipelining:** the query-evaluation can be improved by reducing the number of temporary ûles that are produced. This is achieved this combining several relational operations into a pipeline of operations, in which the results of one operation are passed along to the next operation in the pipeline. Evaluation as just described is called pipelined evaluation. Combining operations into a pipeline eliminates the cost of reading and writing temporary relations. For example, consider the expression $\Pi_{A1,A2}(R \bowtie S)$. When the join operation generates a tuple of its result, it passes that tuple immediately to the project operation for processing instead of creating temporary relations to hold the result of the join, and then reading back in the result to perform the projection. By combining the join and the projection, we avoid creating the intermediate result, and instead create the ûnal result directly.

11.6 TRANSFORMATION RULES FOR RELATIONAL ALGEBRA EXPRESSIONS

By applying transformation rules, the query optimizer can transform one relational algebra expression into an equivalent expression that is known to be more efficient. Two relational algebraic expressions are said to be equivalent if on every legal database instance the two expressions generate the same set of tuples (order of tuples is irrelevant). Following are some of the transformation Rules for Relational Algebraic expressions:

- Cascade of σ : A select with conjunctive conditions on the attribute list is equivalent to a cascade of selects upon selects:

$$\sigma_{A_1 \wedge A_2 \ldots\ldots \wedge A_n}(R) \equiv \sigma_{A_1}(\sigma_{A2}(\ldots(\sigma_{A_n}(R))\ldots))$$

- Commutativity of σ : The select operation is commutative:

$$\sigma_{A_1}(\sigma_{A_2}(R)) \equiv \sigma_{A_2}(\sigma_{A_1}(R))$$

- Cascade of Π : A cascade of project operations is equivalent to the last project operation of the cascade:

$$\Pi_{C_1}(\Pi_{C_2} \ldots(\Pi_{C_n}(R))\ldots) \equiv \Pi_{C_1}(R)$$

- Cascade of σ with Π : If the selection condition *c* involves only those attributes $A_1, A_2, ..., ..., ..., A_n$ in the projection list, the two operations can be commuted:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_c(R)) \equiv \sigma_c(\Pi_{A_1, A_2, \ldots, A_n}(R))$$

- Commutativity of ⋈ with ×: The ⋈ operation is commutative, as is the × operation.

$$R \bowtie_c S \equiv S_c$$
$$R \times S \equiv S \times R$$

- Commutating σ with ⋈ or ×: The Select can be commuted with join (or Cartesian product) as follows.

$$R \bowtie_c S \equiv S_c$$
$$R \times S \equiv S \times R$$

a) If all of the attributes in the select's condition are in relation R then,

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S)$$

b) Given select the condition *c* composed of conditions c_1 and c_2, and c_1 contains only attributes from R, and c_2 contains only attributes from S, then

$$\sigma_c(R \bowtie S) \equiv (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$$

- Commutativity of set operations (∩, ∪, –): Union and intersection operations are commutative; but the difference operation is not:

$$R \cup S \equiv S \cup R,\ R \cap S \equiv S \cap R,\ R - S \neq S - R$$

- Associativity of (⋈, ×, ∩, ∪, –): All four of these operations are individually associative. Let θ be any of these operations, then:

$$(R\ \theta S)\ \theta T \equiv R\ \theta (S\ \theta T)$$

- Commutativity of σ with set operations (∩, ∪, –): Let, θ be any one of the three set operations then:

$$\sigma_c(R\ \theta S)\ \theta T \equiv (\sigma_c(R))\ \theta (\sigma_c(S))$$

- The projection operation distributes over the union operation:

$$\Pi_C(E_1 \cup E_2) \equiv (\Pi_C(E_1)) \cup (\Pi_C(E_2))$$

Using these theorems, an algorithm can be defined to transform the original query expression/tree created by the parser into a more optimized query.



CHECK YOUR PROGRESS

Q.3: Choose the appropriate option:

i) Each relational-algebra expression represents a particular sequence of–

a) Parameters b) Differentiation

c) Operations d) Routines

ii) The operations of theta-join are:

a) Distinctive b) Conjunctive

c) Commutative d) Associative

iii) Which of the following are steps in query processing?

a) Parsing and translation b) Optimization

c) Evaluation d) All of the mentioned

iv) A sequence of primitive operations that can be used to evaluate a query are called as .....

a) Query evaluation algebra

b) Query evaluation plan

c) Query evaluation primitive

11.7 METHODS OF QUERY OPTIMIZATION

Query optimization is the process of selecting an efficient execution plan for evaluating the query. There are two methods of query optimization:

- Cost based Query Optimization
- Heuristical Query Optimization
- **Cost based Query Optimization**: This is based on the cost of the query. A cost-based query optimizer works as follows: First, it generates all possible query execution plans. Next, the cost of each plan is estimated. Finally, based on the estimation, the plan with the lowest estimated cost is chosen. The query can use different paths

based on indexes, constraints, sorting methods etc. This method mainly uses the statistics like record size, number of records, number of records per block, number of blocks, table size, whether whole table fits in a block, organization of tables, uniqueness of column values, size of columns etc. Since the decision is made using estimated cost values, the plan chosen may actually not be optimal.

- **Heuristic Based Optimization:** Heuristic based optimization uses transformation rules to convert one relational algebra expression into an equivalent form that is known to be more efficient. It is also known as rule-based optimization approaches. The scanner and parser generates *initial query tree* a without doing any optimization. In heuristic optimization, query optimizer generates the *final query tree* by using heuristics rules. Heuristics are based on rules for equivalence among relational algebra expressions.

  Some of the common heuristic rules are:

- Perform SELECT and PROJECT operations before JOIN operations. This is done by moving the SELECT and PROJECT operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive SELECT/PROJECT operations at first before the other operations.
- Replace CARTESIAN PRODUCT and a subsequent SELECTION operation with JOIN operation.



11.8 LET US SUM UP

- Query processing and optimization is the most crucial phase a DBMS system.
- The high level query is translated to a low level operation by the query processor.
- Given a query, there are generally a variety of methods for computing the answer. It is the job or responsibility of the query optimizer to translate the query to an equivalent query in internal form that can be computed more efficiently.

- Selection of appropriate evaluation plan is impotent for query optimization.
- There are various searching algorithms available for selection operations.
- There are also various types of possible join algorithms available for choosing proper evaluation plan for join operations.
- A lot of transformation rules help the query optimizer to translate the relational query operations into equivalent expressions with optimizations.
- There are two methods of query optimization: *Cost-based approach* and *Heuristical approach*.



11.9 FURTHER READING

1) Elmasri, Ramez; Navathe, Shamkant B. *Fundamentals of Database Systems*. Pearson Education, Fifth Edition.
2) Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Database System Concepts*, Tata McGraw Hill, Sixth Edition.



11.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: The estimated cost for ***OP1***:

Cost = b_E = 2000 (using *S1a*)

Cost = $b_E/2$ =2000/2 = 1000 (*S1b*)

Cost = X_{Ssn}+1 = 4+1 = 5 (using *S6a*)

The estimated cost for ***OP2***:

Cost=b_E=2000 (using *S1a*)

Cost=X_{Dno}+ b_{I1Dno} +$r_E/2$=2+4/2+(10000.2)=5004 (using *S6b*)

The estimated cost for ***OP3***:

Cost=b_E=2000 (using *S1a*)

Cost=X_{Dno}+ S_{Dno} =2+80=82 (using *S6a*)

Ans. to Q. No. 2: i) (a), ii) (d), iii) (b), iv) (a)

Ans. to Q. No. 3: i) (c), ii) (c), iii) (d), iv) (b)



11.11 MODEL QUESTIONS

Q.1: Define Query optimization and explain its significance for DBMS.

Q.2: Explain the different steps followed in query processing.

Q.3: Explain the different steps followed in query Optimization.

Q.4: What is a relational query tree?

Q.5: What is query evaluation plan?

Q.6: What are the different algorithms available for implementation of selection operation?

Q.7: What are the different algorithms available for implementation of Join operation?

Q.8: Discuss, how the SQL query is converted into relational algebra queries before optimization is done?

Q.9: Discuss the different types of transformation Rules for finding equivalent expressions.

Q.10: Discuss the different methods of Query Optimization.

Q.11: Explain briefly how query expressions are evaluated.

*** ***** ***

UNIT 12: TRANSACTION PROCESSING

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 Concept of Transaction

- 12.3.1 A Basic Transaction Model
- 12.3.2 Single-user versus Multi-user Systems
- 12.3.3 Need of Concurrency Control
- 12.3.4 States of a Transaction
- 12.3.5 Desirable Properties of a Transaction
- 12.3.6 The System Log and Commit Point of a Transaction

12.4 Schedules of Transactions

- 12.4.1 Recoverability of Schedules
- 12.4.2 Serializability of Schedules

12.5 Let Us Sum Up

12.6 Further Reading

12.7 Answers to Check Your Progress

12.8 Model Questions

12.1 LEARNING OBJECTIVES

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

- define a transaction
- learn about transaction processing of DBMS
- define different primitive operations of a transaction
- describe ACID properties of transactions processing
- differentiate between Single-user and multiuser systems
- define the Lost-update problem
- define the schedule of a transaction
- learn the concept of view-serializable schedule and conflict serializable
- define recoverable and cascadeless schedule
- learn about the concepts of recovery management.

12.2 INTRODUCTION

We are already familiar with Query processing and optimization. We are also acquainted with different steps and query processing and how query optimization is done internally by the system in a DBMS.

In this unit we will learn about Transaction processing where we will learn about transaction and system concepts, desirable properties, schedules and Recoverability. In the next unit we will discuss concurrency control in database system and will learn about how to ensure the serializability of transactions in a multi-user database management system.

12.3 CONCEPT OF TRANSACTION

A database transaction is a set of operations that must be processed as one unit of work. The transaction may be an entire program or a single command and may involve any number of operations on the database. A transaction is bounded by **begin transaction** and **end transaction** statement. All the operations of a transaction lie between the two above mentioned statements. A simple application program may contain several transaction boundaries. A transaction is a known as **read-only transaction** if operations in the transaction do not update the database but only retrieve data. A transaction can contain the following two basic database access operations:

- **read (X):** reads a database item named X into a program variable which is also named as X for simplicity (say).
- **write (X)**: writes the program variable named X into the database item named X.

12.3.1 A Basic Transaction Model

Consider the following transaction T consisting of T_1 and T_2. Let, T transfers an amount of £100 from account X to account Y. This transaction can be defined as below:

Begin_Transaction T1
Open_Account(X)
X.Balance = X.Balance – 100
Close_Account(X)
End transaction

Begin_Transaction T2
Open_Account(Y)
Y.Balance = Y.Balance + 100
Close_Account(Y)
End transaction

By, using the simple database access operations, the transaction model for the above transaction can be represented as follows:

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

Figure 12.1: Example of a transaction T.

12.3.2 Single user versus Multi user Systems

A database system can be classified as single user or multi user based on number of user which uses the system. A database system is said to be **singleuser** if at most one user at a time can use it and it is called as **multiuser** if more than one user can use the system at a time, i.e. the system can be used concurrently. Single-user system is restricted to personal computer system. A railway reservation system is a multi-user system. Multi-user system is possible due to the concept of **multiprogramming**. Multiple programs can run simultaneously in a multiprogramming system.

12.3.3 Need of Concurrency Control

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. Concurrency Control is the process of managing simultaneous operations on the database without having them interfere with one another and deals with interleaved execution of more than one transaction.

Following are the problems that can occur when concurrent transactions execute in an uncontrolled manner.

The lost update problem: This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect. Hence, the problem occurs when one user's update overwrites another user's update.

Let us consider two transactions T_1 and T_2 and suppose their operations are interleaved as shown in figure 12.2. The final value of item X is incorrect because T_2 reads the value of X before T_1 changes it in the database, and hence the updated value resulting from T_1 is lost. For example, let X = 200, Y = 300 at the start. The aim of the transaction is to transfer 100 from X to Y and then add 50 more to X. The correct result after the transaction is X = 150, Y = 400. But, in the interleaving operation shown in the fig. 12.2., it is X = 250, Y = 400, because the update in T_1 that transfers the 100 from X was *lost*.

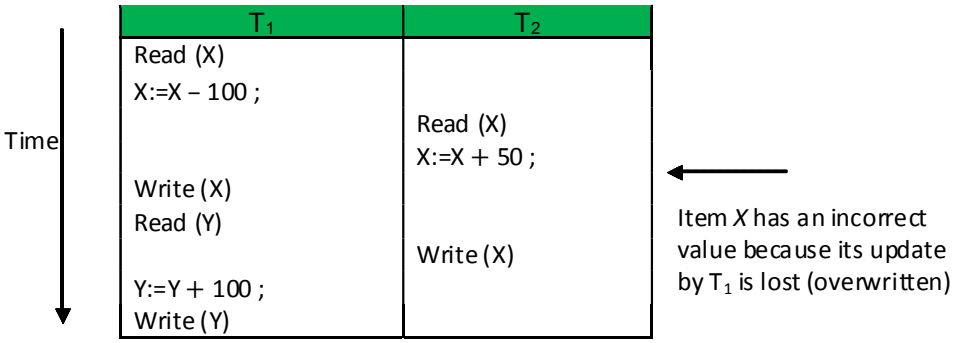


Figure 12.2: Lost Update Problem

The Temporary Update (Dirty Read) Problem: This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value. This happens when one transaction can see intermediate results of another transaction before it has committed.

For example, let X = 200, Y = 300 at the start. Say, the transaction T_1 updates X by reducing 100 but fails before completion. So, the

system must change X to original state i.e. X = 200. But, before it can do so, however, transaction T_2 reads the *temporary* value of X, i.e. X = 100., which is not recorded permanently due to the failure of transaction T_1. The value of X that is read by T_2 is known as *dirty data* and the problem is known as *temporary update problem*.

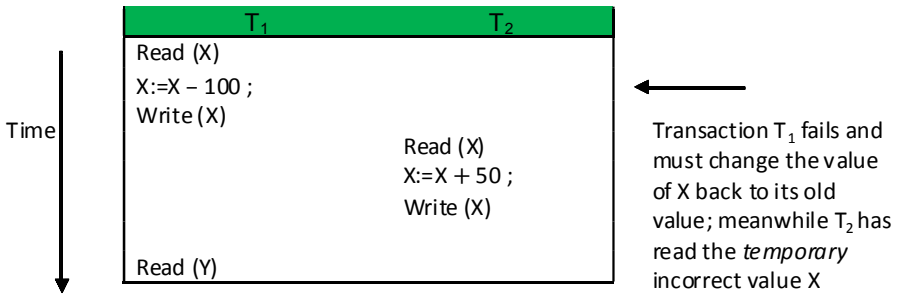


Figure 12.3: Temporary Update (Dirty Read Problem)

The Incorrect Summary Problem: This problem is caused when one of the transactions is executing an aggregate operation on several data items, and other transactions are updating one or more of those data items. This causes an inconsistent database state and is known as Inconsistent analysis or incorrect summary problem.

For example, Let T_1 and T_2 are two transaction in which transaction T_1 transfers money from account A to account B and in the mean time, transaction T_2 calculates the sum of 3 accounts namely, A, B, and C. Let A = 500, B = 500, C = 500 before the start of the transactions.

T_1	T_2
	sum = 0;
	avg = 0;
	Read(C);
	sum := sum + C;
	avg = 0;
Read(A);	
A := A – 50;	
Write(A);	
	Read(A);
	sum := sum + A;
	Read(B);
	sum := sum + B;
	avg := sum/3;
Read(B);	
B := B + 50;	
Write(B);	

Figure 12.4: Incorrect Summary Problem

Transaction T_2 reads the value of account A after A is updated and reads B before B is updated. If all the instructions in T_1 are executed before T_2 starts, then A will be 450, B will be 550 and average value will be 500.

If all the instructions in T_1 are executed after T_1 finishes, then A will be 450, B will be 550 and average value will be 500. But, due to this interleaved execution, the final value of A is 450, B is 550, and average is 483.33 which is wrong. Hence, the aggregate operation is end up with an inconsistent result. This problem is called as Inconsistent analysis *or incorrect summary problem*.

12.3.4 States of a Transaction

Figure 12.5 shows the different states of a transaction. When a transaction starts executing it immediately goes into an ***active*** state. It is the initial state of the transaction where it can issue READ and WRITE operations. When the transaction executes its final statement it moves to the **partially committed** state. If a transaction executes all its operations successfully and a transaction commit point is reached, it moves to **committed** state where all its effects of transaction operations are permanently established on the database system. If the transaction can no longer proceed further with normal execution then it is moved to the failed state.

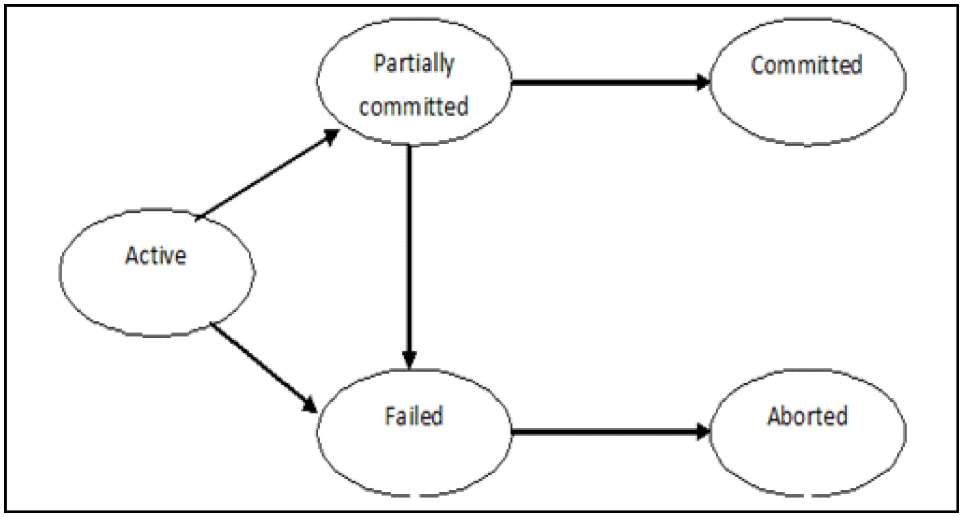


Figure 12.5: Sates of a transaction

However, if any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called **aborted**. In this stage system has two options:

- *Restart the transaction*: A restarted transaction is considered to be new transaction which may recover from possible failure.
- *Kill the transaction*: A transaction can be killed to recover from failure.

12.3.5 Desirable Properties of Transactions

To ensure the integrity of data during a transaction, the database system maintains the following properties. These properties are widely known as ACID properties:

- **Atomicity:** A transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. For example, let us consider a transaction where an account A has a balance of 200¹ and B has 300¹. Account A is transferring 100¹ to Account B. This is a transaction that has two operations: first, debiting 100¹ from A's balance and crediting 100¹ to B's balance. Let's say first operation passed successfully while second failed, in this case A's balance would be 100¹ while B would be having 300¹ instead of 400¹. This makes the account states inconsistent. Either the transaction should fail without executing any of the operation or it should process both the operations. The Atomicity property ensures that.
- **Consistency:** The transaction should be consistency preserving i.e. it should not have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

- **Isolation:** The property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will interfered by any other transaction executing concurrently.
- **Durability:** Once a transaction completes successfully (i.e. committed), the changes it has made into the database should be persistent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.



CHECK YOUR PROGRESS

Q.1: Choose the appropriate option:

i) Collections of operations that form a single logical unit of work are called as:

a) Views b) Networks
c) Units d) Transactions

ii) The "all-or-none" property is commonly referred to as:

a) Isolation b) Durability
c) Atomicity d) Consistency

iii) Execution of translation in isolation preserves the ..... of a database:

a) Atomicity b) Consistency
c) Durability d) All of the above

iv) Which of the following is not a transaction state?

a) Active b) Partially committed
c) Failed d) Compensated

12.3.6 The System Log and Commit Point of a Transaction

The transaction processing system maintains a **log** file i.e. **system log** to recover from failures that affect the transaction processing. It keeps track of all transaction operations that affect the values of database items which may be needed to permit recovery

from failures. The log is kept in the disk, so it is not affected by any type of failure except disk or catastrophic failure. It is periodically backed up to archival storage to guard against such catastrophic failures. The entries in the log file are known as **log records**. Following are the list of log records with its actions in a system log file where T refers to a unique **transaction id** which is generated automatically by the system:

- [**begin_transaction, T**]: Indicates that transaction *T* has started.
- [**write, T,Q, old_value, new_value**]: Indicates that transaction *T* has changed the value of the database item *Q* from *old_value* to *new_value*.
- [**read, T,Q**]: Indicates that transaction *T* has read the value of the database item *Q*.
- [**commit, T**]: Indicates that transaction *T* has completed successfully and confirms that its effect can be recorded permanently to the database.
- [**abort, T**]: Indicates that transaction *T* has been aborted.

Since the system logs maintain each READ and WRITE operations so, it is possible to undo or redo of operations whenever necessary for system recovery.

A transaction T reaches its **commit point** when all its operations have been executed successfully and the effect of all its operations on the database items have been recorded in the log. Beyond the commit point, the transaction is said to be **committed** and its effect is saved permanently into the database. At this point, the transaction writes a commit record **[commit, T]** into the system log file. If the transaction T, fails due to system failure, the recovery process search back in the log file for all transactions T that have written a **[begin_transaction, T]** record into the log but have not written **[commit, T]** record yet. These transactions may have to be rolled back to undo their effect on the database during the recovery process.

12.4 SCHEDULES OF TRANSACTIONS

Transactions can execute concurrently in interleaved fashion. The order of execution of operations from the various transactions is known as schedule. A **schedule** *S* of *n* transactions $T_1, T_2, ..., ..., ..., T_n$ is an ordering of the operations subject to the constraint that, for each transaction T_i that participates in *S*, the operations of T_i in *S* must appear in the same order in which they occur in T_i. However, operations from other transactions T_j can be interleaved with the operations of T_i in S. For the purpose of recovery and concurrency control, we are only interested on *Read*, *Write*, *Commit* and *Abort* operations of the transactions. For describing a schedule we can use a shorthand notations *r*, *w, c* and *a* for the operations *Read*, *Write*, *Commit* and *Abort* respectively, and can append the transaction id (i.e. transaction number) as subscript. So, the Schedule of fig. 12.2, which we shall call S_1 can be written as follows in the notation:

$$S_1: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

Two operations in a schedule are said to be in ***conflict*** (conflicting pair) if they satisfy all of the following three conditions:

1) The operations belong to different transactions.
2) At least one of the operations is a *write (X*) operation.
3) The actions access the same item *X* (*read(X)* or *write(X)*).

So, the following set of actions is conflicting:

- $r_1(X); w_2(X); w_3(X);$ (3 conflicting pairs)

While the following sets of operations are not conflicting:

- $r_1(X); r_2(X); r_3(X)$ and $r_1(X); w_2(Y); r_3(X);$

A schedule *S* of *n* transactions $T_1, T_2, ..., ..., ..., T_n$ is said to be in ***complete schedule*** if they satisfy following conditions:

1) The operations in S are exactly those operations in $T_1, T_2, ..., ..., ..., T_n$ including a commit or abort operation as the last operation for each transaction in the schedule.
2) For any pair of operations from the same transaction T_i, their order of appearances in S is the same as their order of appearances in T_i.
3) For any two conflicting operations, one of the two must occur before the other in the schedule S.

12.4.1 Recoverability of Schedules

For some schedule, it is easy to recover from failure and some are not. A recoverable schedule is one where a committed transaction T should never be rolled back.

Recoverable Schedules: A **recoverable schedule** is one where, for each pair of Transaction T_i and T_j such that if T_j reads data item previously written by Ti, then the commit operation of T_i appears before the commit operation T_j.

For example let us consider the following schedule S as shown in figure 12.6. of two transactions T_1 and T_2 and let X = 100 initially. In this schedule, the transaction T_1 reads X = 100 and updates X as 110. The transaction T_2 reads the item X as 110 and updates as X = 105 and it is committed. Now, let T_2 fails due to some failure after T_2 committed. T_1 fails, so it is rolled back and set X to its original value as X = 100.

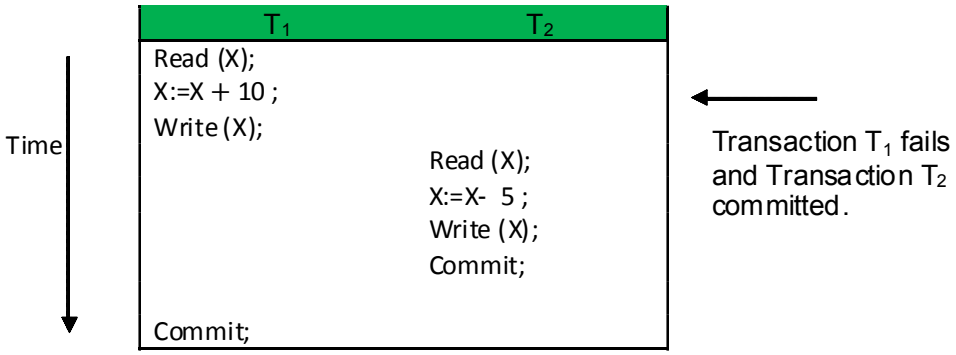


Figure 12.6: Non-Recoverable Schedule

Since, T_2 uses the value X updated by T_1, so it should also be rolled back which is not possible because a committed transaction never be rolled back. This schedule is therefore called as non-recoverable.

But, if commit operations of T_1 appears before commit operation of T_2 as shown in figure 12.7. Now, if transaction T_1 fails and rolled backed, T_2 has never to be rolled back due to T_1. It will always use the committed value of X and hence the schedule now becomes as recoverable.

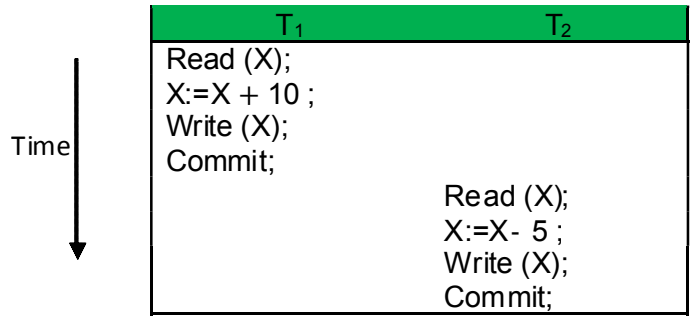


Figure 12.7: Recoverable Schedule

Cascadless Schedules: A cascading rollback occurs in database systems when a transaction T_1 causes a failure and a rollback must be performed. Other transactions dependent on T_1's actions must also be rollbacked due to T_1's failure, thus causing a cascading effect. This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called **Cascading rollback**. Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those where cascading rollbacks cannot occurs, Such schedules are called **Cascadeless Schedules.**

Formally, a *cascadeless schedule* is one where for each pair of transaction T_i and T_j such that T_j reads data item, previously written by T_i, the commit operation of T_i must appears before the read operation of T_j. Every Cascadeless schedule is also recoverable schedule.

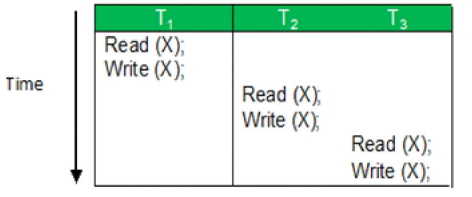


Figure 12.8: Cascading Rollback

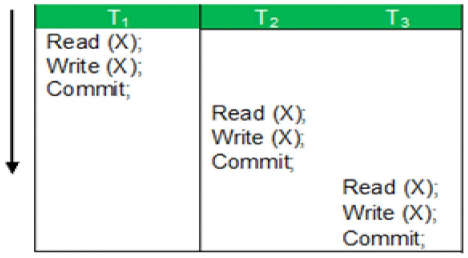


Figure 12.9: Cascading Schedule

Let us, take the example of a schedule S as shown in fig 12.8. Where, Transaction T_1 writes X that is read by Transaction T_2. Transaction T_2 writes X that is read by Transaction T_3. Suppose at this point T_1 fails. T_1 must be rolled back, since T_2 is dependent on

T_1, T_2 must be rolled back and since T_3 is dependent on T_2, T_3 must be rolled back. If we arrange the above transactions as a schedule as shown in Figure 12.9. it will avoid the cascading abort and such a schedule is known as **cascadeless** schedule.

Strict Schedule: A Strict schedule is one where transactions can *neither read nor write* an item X until the last transaction that wrote X has committed (or aborted). Strict schedule simplify the recovery process.

Following schedule as shown in figure 12.10 is a strict schedule.

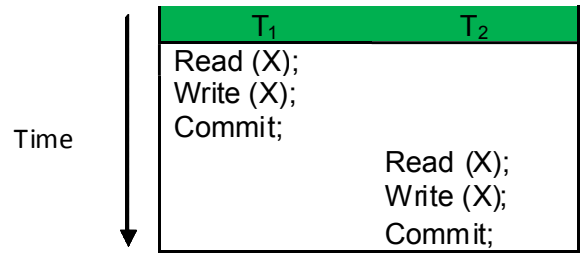


Figure 12.10: Strict Schedule

12.4.2 Serializability of Schedules

Serializability is the classical concurrency scheme. Serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedule. A schedule is called “correct” if we can find a serial schedule that is “equivalent” to it.

Serial Schedules: A schedule is called as a **serial** schedule when the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction. Formally, a schedule S is serial if, for every transaction T_i participating in the schedule, all the operations of T_i are executed consecutively in the schedule; otherwise, the schedule is called **nonserial** schedule. In serial schedule only one transaction at a time is active. Figure 12.11(a) and 12.11(b) shows example of serial schedules S_1 and S_2 of transaction T_1 and T_2.

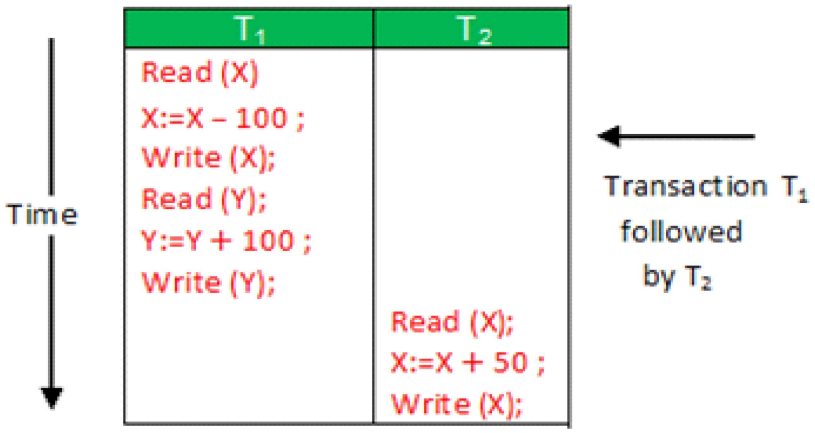


Figure 12.11(a): Serial Schedule S_1: followed by T_2

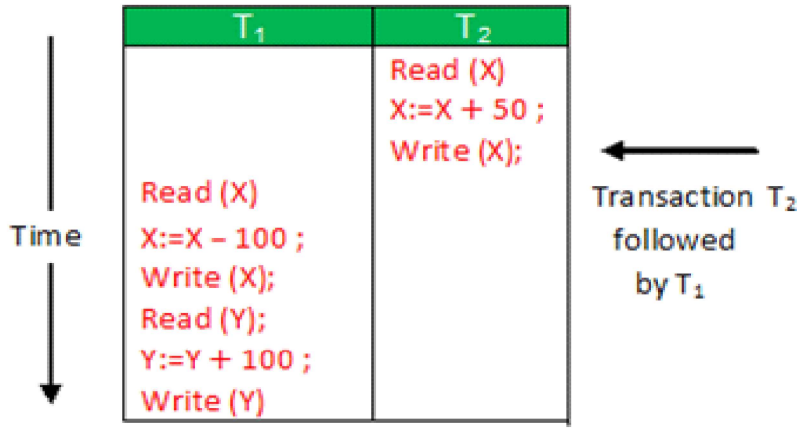


Figure 12.11(b): Serial Schedule S_2: followed by T_1

Figure 12.11(c) and 12.11(d) shows example of non-serial schedules S_3 and S_4 of transaction T_1 and T_2 with interleaving of operations of T_1 and T_2.

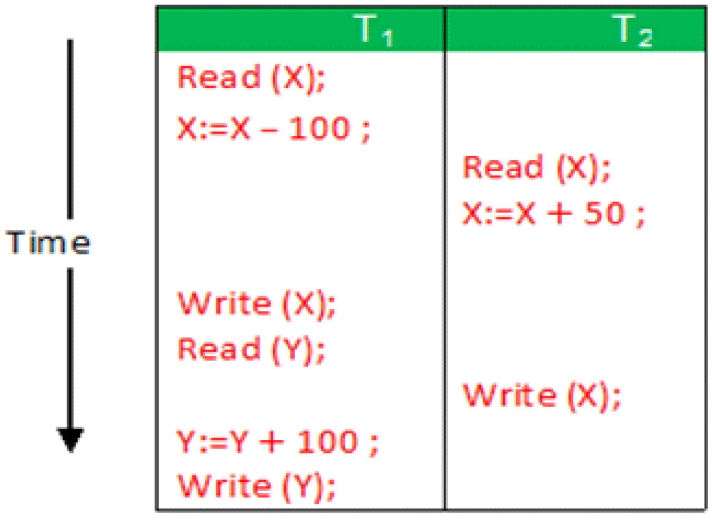


Figure 12.11(c): Non-Serial Schedule S_3 with interleaving of T_1 and T_2

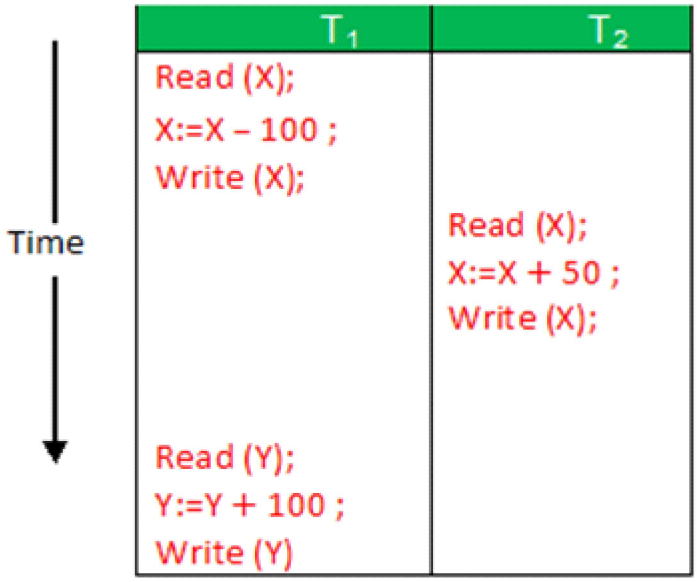


Figure 12.11(d): Non-Serial Schedule S_4 with interleaving of T_1 and T_2

Serializable Schedule: A serial schedule is inefficient because no interleaving is allowed. This leads to low CPU utilization, and slowing down processes in general. A serializable schedule gives the benefits of concurrent execution without giving up correctness. A schedule S of n transactions $T_1, T_2, ..., ..., ..., T_n$ is **serializable** if it is **equivalent** to some serial schedule of the same n transactions $T_1, T_2, ..., ..., ..., T_n$. There are n! possible serial schedules of n transactions and many more possible nonserial schedules.

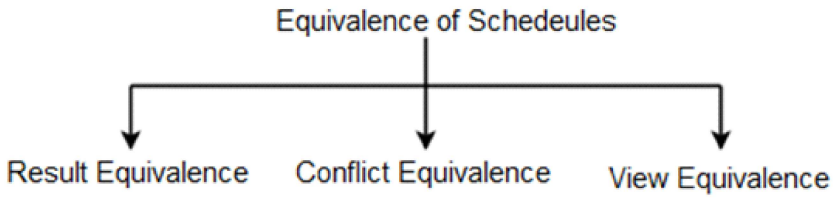


Figure 12.12: Equivalence of Schedules

A non-serial schedule S is equivalence schedule to a serial schedule if both are correct.

Schedules may have three different kinds of equivalence schedules among them which are: **Result Equivalence**, **Conflict Equivalence** and **View Equivalence**.

- **Result Equivalence:** If any two schedules generate the same result after their execution, then such schedules are called as **result equivalent schedules**. This equivalence relation is

considered of least significance because some schedules might produce the same results for some set of values and they might produce the different results for some other set of values.

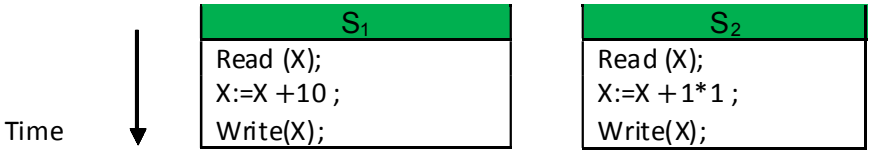


Figure 12.13: Non Result Equivalence

Two schedules S1 and S_2 are result equivalent for initial value but are not result equivalent in general.

- **Conflict Equivalence:** Two schedules are conflict equivalent if the order of any two conflicting operations is the same in both schedules. This is the more commonly used definition of equivalence. A schedule is conflict serializable if it is equivalent to some serial schedule. We can reorder the nonconflicting operations if necessary. A schedule is called **conflict serializable** if it can be transformed into a serial schedule by swapping non-conflicting operations. The definition of conflict operations are discussed in section 12.4.

  Consider the schedule S_4 of figure 12.11(d) which is equivalent to the serial schedule S_1 of figure 12.11(a). In both the schedule read(X) of T_2 reads the value of X written by T_1, while other read operations read the values from the initial database state. Furthermore, T_1 is the last transaction to write Y, and T_2 is the last transaction to write X in both the schedules. Since, schedule S_1 is a serial schedule and schedule S_4 is equivalent to S_1. So, S_4 is a serial schedule. Now, notice that the operations $R_1(X)$ and $W_1(X)$ of schedule S_4 do not conflict with the operations $R_1(X)$ and $W_1(X)$ of schedule S_1, since they access different data items. Hence, we can move $R_2(X)$, $W_2(X)$ before $R_2(X)$, $W_2(X)$, leading to the equivalent serial schedule T_1, T_2. Hence, Schedule S_4 is conflict serializable schedule to schedule S_1.

- **View Equivalence:** Two schedules are said to be view equivalence, if the transaction in one schedule is same as the

one in other. That means, both the transactions in two schedules perform same tasks. Formally, two schedules Si and Sj are said to be view equivalent if the following three conditions hold:

- The same set of operations participates in S_i and S_j, and S_i and S_j include the same operations of those transactions.
- For any operation $R_i(Q)$ of T_i in S_i, if the value of Q read by the operation has been written by the operation $W_j(Q)$ of T_j (or if it is the original value of Q before the schedule started), the same condition must hold for the value of Q read by operation $R_i(Q)$ of T_i in S_j.
- If the operation $W_k(P)$ of T_k is the last operation to write item P in $_{Si}$, then $W_k(P)$ of T_k must also be the last operation to write item P in S_j.

A schedule S is said to view serializable if it is view equivalent to a serial schedule.



CHECK YOUR PROGRESS

Q.2: Choose the appropriate option:

i) If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, then S and S' are:

a) Non conflict equivalent b) Equal

c) Conflict equivalent d) Isolation equivalent

ii) A schedule is ..... if it is conflict equivalent to a serial schedule.

a) Conflict serializable b) Conflicting

c) Non serializable d) None of the mentioned

iii) The phenomenon in which one failure leads to a series of transaction rollbacks is called as .....

a) Cascadeless rollback b) Cascading rollback

c) Cascade cause d) None of the mentioned

iv) If a schedule is equivalent to a serial schedule, it is called as a .....

a) Serializable schedule b) Equivalent schedule

c) Committed schedule d) Conflicting Schedule



12.5 LET US SUM UP

- A transaction is a set of operations that must be processed as one unit of work.
- Transaction always follows **A**tomicity, **C**onsistency, **I**solation and **D**urability properties
- A transaction may be any one of the five states such as *Active*, *Partially Committed*, *Committed*, *Failed* and *Abort*.
- Concurrency control is needed to avoid *Dirty Read*, *Lost Update* and *Incorrect Summary* problems
- Transactions can be executed concurrently in interleaved fashion and the order of execution of operations from the various transactions is known as schedule.
- Scheduling of transaction is important for concurrency control and can be done in different ways.
- Schedules of transactions may be serial or non-serial.
- There may various ways to find out equivalence of schedules and serialization of schedule is equivalent to some serial schedule.
- There are three types of equivalence of transaction schedules: *Result* equivalence, *Conflict* equivalent and *View* equivalent.
- There are two types of serialization of schedules: Conflict serializability and View serialzability based on Conflict equivalent and view equivalent respectively.



12.6 FURTHER READING

1) Elmasri, Ramez; Navathe, Shamkant B. *Fundamentals of Database Systems*. Pearson Education, Fifth Edition.

2) Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Database System Concepts*, Tata McGraw Hill, Sixth Edition.



12.7 ANSWERS TO CHECK YOUR PROGRESS

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

Ans. to Q. No. 2: i) (c), ii) (a), iii) (b), iv) (a)



12.8 MODEL QUESTIONS

Q.1: With example, explain single user and multi-user system

Q.2: What do you mean by a transaction? Explain the desirable properties of a transaction.

Q.3: What do you mean by the commit point of a transaction?

Q.4: Why concurrency control is needed in transaction processing? Explain briefly *dirty read, incorrect summary* problem.

Q.5: Explain *Lost update* problem with example.

Q.6: What is system log? Explain the operations performed in the system log.

Q.7: Draw a state diagram and explain the typical states that a transaction goes through the execution.

Q.8: What is a schedule? Define the concepts of *recoverable*, *cascadeless* and *strict schedules*, and compare them in terms of recoverability.

Q.9: What is serializability of transactions? Explain different types of serializability.

Q.10: What are the conditions for conflict equivalent schedules? Explain conflict serializability.

Q.11: Consider the following schedule S_1 of two transactions T_1 and T_2. Find out whether the schedule S_1 is conflict serializable or not.

$S_1 = r_1(Q); r_2(Q); w_1(Q); w_2(Q);$

*** ***** ***

UNIT 13: CONCURRENCY CONTROL

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 Two Phase Locking Techniques for Concurrency Control

13.4 Types of Two Phase Locking

- 13.4.1 Conservative Two Phase Locking
- 13.4.2 Strict Two Phase Locking
- 13.4.3 Distributed Two Phase Locking

13.5 Concurrency Control based on Timestamp Ordering

- 13.5.1 Basic Timestamp Ordering
- 13.5.2 Strict Timestamp Ordering

13.6 Multi-Version Concurrency Control (MVCC)

- 13.6.1 The Schedulers for MVCC based Timestamp Ordering
- 13.6.2 Multiversion Two-Phase Locking

13.7 Validation (Optimistic) Concurrency Control Techniques

13.8 Granularity of Data Item

13.9 Database Recovery Concept

- 13.9.1 Category of Recovery Algorithm

13.10 Let Us Sum Up

13.11 Further Readings

13.12 Answer to Check Your Progress

13.13 Model Questions

13.1 LEARNING OBJECTIVES

After going through this unit, you will able to:

- define Concurrency
- learn about the differrent Concurrency control techniques
- describe the idea of Controlling Concurrency based on Timestamp
- descibe the types of timestamp ordering
- learn about multi-version Concurrency control and Optimistic

Concurrency control techniques

- describe the concept of granularity.

13.2 INTRODUCTION

In the previous unit, we learnt about transaction and serializability of transaction. This unit discusses the different concurrency control techniques and locking and how locking helps in transaction isolation. Most of the techniques discussed ensure the serializabality by using different protocols. The unit discusses the techniques like two phase locking, timestamp ordering, optimistic protocols etc. The effect of granularity in currency control is also discussed at the end of this unit.

13.3 TWO PHASE LOCKING TECHNIQUES FOR CONCURRENCY CONTROL

In case of shared data, locking is a mechanism commonly used to solve the problem of synchronized access. Conceptually, it is a variable associated with a data item. It is important here to note that each data item has a lock associated with it. Before a transaction T1, access a data item, the scheduler first examines the associated lock with the data item. If no transaction holds the lock on the data item, then the scheduler obtains the lock on behalf of T1. If another transaction T2, hold the lock, then T1, has to wait until T2 gives up the lock. That is, the scheduler will not give T1, the lock until T2, releases it. The scheduler thereby ensures that only one transaction can hold the lock at a time, so only one transaction can access the data item at a time. Locking can be used by a scheduler to ensure serializability of transaction. To present such a locking protocol, we need some notation.

Transactions access data items either for reading or for writing them. We therefore associate two types of locks with data items: read locks and write locks. Let us assume that rl[x] denote a read lock on data item x and wl[x] denote a write lock on x. rl[x] (or wl[x]) to indicate that transaction T, has obtained a read (or write) lock on x. We use $rl_i[x]$ (or $wl_i[x]$) to denote the operation by which T_i releases its read (or write) lock on x. In this case, we say T_i unlocks x. It is the job of a two Phase Locking (2PL) scheduler to

manage the locks by controlling when transactions obtain and release their locks. A transaction is said to follow the Two Phase Locking protocol, if all locking operations (either read or write lock) first perform the unlock operation in any transaction. The rules according to which a basic 2PL scheduler manages and uses its locks:

1) When it receives an operation $p_i[x]$ from the Transaction Manager, the scheduler tests if $pl_i[x]$ conflicts with some $ql_i[x]$ that is already set. If so, it delays $p_i[x]$, forcing T_i to wait until it can set the lock it needs. If not, then the scheduler sets $pl_i[x]$, and then sends $p_i[x]$ to the Database Manager.
2) Once the scheduler has set a lock for T_i, say $pl_i[x]$, it may not release that lock at least until after the Database Manager acknowledges that it has processed the lock's corresponding operation, pi[x].
3) Once the scheduler has released a lock for a transaction, it may not subsequently obtain any more locks for that transaction on any data item.

From the above rule it has been noticed rule (1), prevents two transactions from concurrently accessing a data item in conflicting modes. Thus, conflicting operations are scheduled in the same order in which the corresponding locks are obtained.

The rule (2) supports rule (1) by ensuring that the Database Manager processes operations on a data item in the order that the scheduler submits them. For example, suppose T_i obtains rl[x], which it releases before the Database Manager has confirmed that $r_i[x]$ has been processed. Then it is possible for T_j to obtain a conflicting lock on x, $wl_j[x]$, and send $w_j[x]$ to the Database Manager. Although the scheduler has sent the Database Manager $r_i[x]$ before $w_j[x]$. Without rule (2) there is no guarantee that the Database Manager will receive and process the operations in that order.

Rule (3), also called the two phase rule, is the base for the name two phase locking. Each transaction may be divided into two phases: a growing phase or expanding phase (first phase) during which it obtains locks, and a shrinking phase (second phase) during which it releases locks.

13.4 TYPES OF TWO PHASE LOCKING

There are basically two types of Two Phase Locking. They are:

- Conservative Two Phase Locking
- Strict Two Phase Locking

13.4.1 Conservative Two Phase Locking

It is possible to construct a Two Phase Locking scheduler that never aborts transactions. This technique is known as **Conservative Two Phase Locking** or **Static 2X**. As Two Phase Locking most of the time causes abortions because of deadlocks. Conservative Two Phase Locking avoids deadlocks by requiring each transaction to obtain all of its locks before any of its operations are submitted to the Database Manager. This is done by having each transaction to declare its readset and writeset in advance. Specifically, each transaction T_i first tells the scheduler all the data items it will want to Read or Write, for example as part of its Start operation. The scheduler tries to set all of the locks needed by T_i. It can do this by ensuring that none of these locks conflicts with a lock held by any other transaction. If the scheduler succeeds in setting all of T_i's locks, then it submits T_i's operations to the Database Manager as soon as it receives them. After the Database Manager acknowledges the processing of T_i's last database operation, the scheduler may release all of T_i's locks. If, on the other hand, any of the locks requested in T_i's Start conflicts with locks presently held by other transactions, then the scheduler does not grant any of T_i's locks. Instead, it inserts T_i along with its lock requests into a waiting queue. Every time the scheduler releases the locks of a completed transaction, it examines the waiting queue to see if it can grant all of the lock requests of any waiting transactions. If so, it then sets all of the locks for each such transaction and continues processing as just described. In Conservative Two Phase Locking, if a transaction T_i is waiting for a lock held by T_j, then T_i is holding no locks. Therefore,

no other transaction T_k can be waiting for T_i. Since deadlock is the only reason that a Two Phase Locking scheduler ever rejects an operation and thereby causes the corresponding transaction to abort, Conservative Two Phase Locking never aborts a transaction. By delaying operations sooner than it has to, namely, when the transaction begins executing, the scheduler avoids abortions that might otherwise be needed for concurrency contro1 reasons.

13.4.2 Strict Two Phase Locking

Almost all implementations of Two Phase Locking use a variant called **Strict Two Phase Locking**. This differs from the Basic 2PL scheduler described in the previous section that it requires the scheduler to release all of a transaction's locks together, when the transaction terminates. More specifically, T_i's locks are released after the Database Manager acknowledges the processing of c_i or a_j, depending on whether T_i commits or aborts respectively. The reasons for adopting this policy is Firstly, consider when a Two Phase Locking scheduler can release some $ol_i[x]$. To do so the scheduler must know that: (1) T_i has set all of the locks it will ever need, and (2) T_i will not subsequently issue operations that refer to X. One point in time at which the scheduler can be sure of (1) and (2) is when T_i terminates, that is, when the scheduler receives the c_i or a_i operation. In fact, in the absence of any information from the Transaction Manager aside from the operations submitted, this is the earliest time at which the scheduler can be assured that (1) and (2) hold. A second reason for the scheduler to keep a transaction's locks until it ends, and specifically until after the Database Manager processes the transaction's Commit or Abort, is to guarantee a strict execution.

13.4.3 Distributed Two Phase Locking

Two phase locking can also be realized in a distributed Database Management System. A distributed Database Management System consists of a collection of communicating nodes, each of

which is a centralized Database Management System. Each data item is stored at exactly one node. The scheduler manages the data items stored at its site. This means that the scheduler is responsible for controlling access to the items. A transaction submits its operations to a Transaction Manager. The Transaction Manager then delivers each Read(x) or Write(x) operation of that transaction to the scheduler that manages x. When a scheduler decides to process the Read(x) or Write(x), it sends the operation to its local Database Manager, which can access x and return its value (for a Read) or update it (for a Write). The Commit or Abort operation is sent to all nodes where the transaction accessed data items. The schedulers at all nodes, taken together, constitute a distributed scheduler. The task of the distributed scheduler is to process the operations submitted by the Transaction Managers in a globally serializable and recoverable manner. It is possible to build a distributed scheduler based on Two Phase Locking. Each scheduler maintains the locks for the data items stored at its node and manages them according to the Two Phase Locking rules. In Two Phase Locking, a Read(x) or Write(x) is processed when the appropriate lock on x can be obtained, which only depends on what other locks on x are presently owned. Therefore, each local Two Phase Locking scheduler has all the information it needs to decide when to process an operation, without communicating with the other sites. It is problematic to decide when to release a lock. To enforce the two phase rule, a scheduler cannot release a transaction T_i's lock until it knows that T, will not submit any more operations to it or any other scheduler. Otherwise, one scheduler might release T_i's lock and sometime later another scheduler might set a lock for T_i, thereby violating the two phase rule.

It would appear that enforcing the two phase rule requires communication among the schedulers at different nodes. However, if schedulers use Strict Two Phase Locking, then they can avoid such communication. Because, the Transaction Manager that manages transaction T, sends T_i's Commit to all nodes where T_i accessed

data items. By the time the Transaction Manager decides to send T_i's Commit to all those sites, it must have received acknowledgments to all of T_j's operations. Therefore, T_i has surely obtained all the locks it will ever need. Thus, if a scheduler releases T_i's locks after it has processed T_j's Commit (as it must under Strict Two Phase Locking), it knows that no scheduler will subsequently set any locks for T_i. To prove that the distributed Two Phase Locking scheduler is correct we simply ensure that any history H it could have produced satisfies the properties of Two Phase Locking histories.



CHECK YOUR PROGRESS

Q.1: Fill up the gapes:

a) A lock is a ..... associated with a data item.

b) Changing the state of lock from one type to another is known as .....

c) In ..... phase of locking, new locks on item can be granted.

d) In ..... phase, existing locks are released.

e) In conservative two phase locking technique, it never ..... transaction.

13.5 CONCURRENCY CONTROL BASED ON TIMESTAMP ORDERING

In timestamp ordering, the Transaction Manager assigns a unique timestamp, $ts(T_i)$, to each transaction T_i. It generates timestamps using some of the efficient techniques, in the context of timestamp-based deadlock prevention. The Transaction Manager attaches a transaction's timestamp to each operation issued by the transaction. It will therefore be convenient to speak of the timestamp of an operation $o_i[x]$, which is simply the timestamp of the transaction that issued the operation. A **Timestamp Ordering (TO)** scheduler orders conflicting operations according to their timestamps. More precisely, it applies the following rule, called the **TO rule**.

TO Rule: If $p_i[x]$ and $q_i[x]$ are conflicting operations, then the Database Manager processes $p_i[x]$ before $q_i[x]$ if and only if $ts(T_i) < ts(T_i)$. By enforcing the TO rule, it can be ensured that every pair of conflicting operations is executed in timestamp order. Thus, a TO execution has the same effect as a serial execution in which the transactions appear in timestamp order.

13.5.1 Basic Timestamp Ordering

The Basic TO is a simple and aggressive implementation of the TO rule. It accepts operations from the Transaction Manager and immediately outputs them to the Database Manager in first-come-first-served order. To ensure that this order does not violate the TO rule, the scheduler rejects operations that it receives too late. An operation $p_i[x]$ is too late if it arrives after the scheduler has already output some conflicting operation $q_i[x]$ with $ts(T_i) > ts(T_j)$. If $p_i[x]$ is too late, then it cannot be scheduled without violating the TO rule. Since the scheduler has already output $q_j[x]$, it can only solve the problem by rejecting $p_i[x]$.

If $e_i[x]$ is rejected, then T_i must abort. When T_i is submitted again, it must be assigned a timestamp sufficiently large so that its operations are less likely to be rejected during its second execution.

To determine if an operation has arrived too late, the Basic TO scheduler maintains for every data item x the maximum timestamps of Reads and Writes on x that it has sent to the Database Manager, denoted by max-r-scheduled[x] and max-w-scheduled[x] respectively. When the scheduler receives $e_i[x]$, it compares $ts(T_i)$ to max-q-scheduled[x] for all operation types q that conflict with p. If $ts(T_i) <$ max-q-scheduled[x], then the scheduler rejects $e_i[x]$, since it has already scheduled a conflicting operation with a larger timestamp. Otherwise, it schedules $e_i[x]$ and, if $ts(T_i) >$ max-p-scheduled[x], it updates max-p-scheduled[x] to $ts(T_i)$.

The scheduler must communicate to the Database Manager to guarantee that operations are processed by the Database

Manager in the order that the scheduler sent them. Even if the scheduler decides that $e_i[x]$ can be scheduled, it must not send it to the Database Manager until every conflicting $q_j[x]$ that it previously sent has been acknowledged by the Database Manager.

For proper maintenance of the communication with the Database Manager, the Basic TO scheduler also maintains, for each data item x, the number of Reads and Writes that have been sent to the Database Manager, but not yet acknowledged. These are denoted as r-in-transit[x] and w-in-transit[x] respectively. For each data item x the scheduler also maintains a queue, queue[x], of operations that can be scheduled in so far as the TO rule is concerned, but are waiting for acknowledgments from the Database Manager to previously sent conflicting operations.

13.5.2 Strict Timestamp Ordering

An improved version of Basic TO is the Strict TO, which is both Strict ensuring easy recoverability and serializable. Although the TO rule enforces serializability, it does not necessarily ensure recoverability. If w-in-transit[x] denotes the number of w[x] operations that the scheduler has sent to the Database Manager but that the Database Manager has not yet acknowledged. Since two conflicting operations cannot be “in transit” at any time and Writes on the same data item conflict, w-in-transit[x] at any time is either 0 or 1. The Strict TO scheduler is similar to Basic TO in every respect, except that it does not set w-in-transit[x] to 0 when it receives the Database Mamnager’s acknowledgment of a w[x]. Instead it waits until it has received acknowledgment of a_i or c_i. It then sets w-in-transit[x] to zero for every x for which it had sent $w_i[x]$ to the Database Manager. This delays all $r_j[x]$ and $w_j[x]$ operations with $ts(T_j) > ts(T_i)$ until T_i has committed or aborted. This means that the execution output by the scheduler to the Database Manager is strict.



CHECK YOUR PROGRESS

Q.2: State True or False:

a) Timestamp is an unique identifier used to identify a transaction when it is aborted.

b) Arranging of transaction based on timestamp value is known as Timestamp ordering.

13.6 MULTI-VERSION CONCURRENCY CONTROL (MVCC)

In a multiversion concurrency control algorithm, each Write on a data item x produces a new copy or version of x, thereby keeping the original copy intact. The Database Manager that manages x keeps a list of copies or versions of x, which is the history of values that the Database Manager has assigned to x. For each Read(x), the scheduler not only decides when to send the Read to the Database Manager, but it also tells the Database Manager which one of the copy or version of x to read.

The main benefit of multiple versions for controlling concurrency is to help the scheduler avoid rejecting operations that arrive too late. For example, the scheduler normally rejects a Read because the value it was supposed to read has already been overwritten. With multiversions, such old values are never overwritten and are therefore always available for Reads. The Database Manager makes the different versions explicitly available to the scheduler. The disadvantage of MVCC is the utilization of storage space in turn wasting valuable memory space and for better processing, it is also important to do the archiving thereby incurring processing overhead.

It is important to note here that when a transaction is aborted, the scheduler does not create a new version. The new versions are only created for the transaction which is either active or committed. Thus, when the scheduler decides to assign a particular version of x to Read(x), the value returned is not one produced by an aborted transaction.

13.6.1 The Schedulers for MVCC based Timestamp Ordering

According the TO algorithm, each transaction has a unique timestamp, denoted ts(T_i). Each operation carries the timestamp of its corresponding transaction. Each version is labeled by the timestamp of the transaction that wrote it. A Multiversion TO (M VTO) scheduler processes operations on first-come-first-served basis. The scheduler processes $r_i[x]$ by first translating it into $r_i[x_k]$, where x_k is the version of x with the largest timestamp less than or equal to ts(T_i), and then sending $r_j[x_k]$ to the Database Manager. It processes $w_i[X]$ by considering two cases.

If it has already processed a Read $r_j[x_k]$ such that ts(T_k) < ts(T_i) < ts(T_j), then it rejects $w_i[x]$. Otherwise, it translates $w_i[x]$ into $w_j[x_j]$ and sends it to the Database Manager. Finally, to ensure recoverability, the scheduler must delay the processing of c_i until it has processed C_j for all transactions T_j that wrote versions read by T_i.

13.6.2 Multiversion Two-Phase Locking

According to this concept there are three locking modes read, write certify instead of two, namely read and write. So, the state of LOCK(X) for an item X can be read-lock, write-lock or certify. According to the locking concept, write_lock is considered as an exclusive lock, i.e. when a transaction holds a write_lock on an item, that item cannot be locked by any other transaction.

The idea behind multiversion Two Phase Locking is to allow other transactions T' to read an item x while a single transaction T holds a write lock on x. This is accomplished by having two versions for each item x. One version must always have been written by some committed transaction, while the second version is created when a transaction T acquires a write lock on the item. So, other transaction can continue accessing the item from the committed version of x. Any transaction T can write the value of x' when required,

without affecting the value of the committed version of x. Once T is ready to commit, it must obtain a certify lock on all items that it currently holds write locks on before it can commit. To obtain a certify lock, a transaction may have to delay its commit, until all its write locked items are released by any reading transactions. Once certify lock is obtained, the committed version of the data is updated with the new value and the certify lock is released.

Thus, in case of multiversion Two Phase Locking, read can proceed concurrently with a single write operation, which is not permitted in case of standard Two Phase Locking.



CHECK YOUR PROGRESS

Q.3: Fill up the gapes:

a) In Multi Version Concurrency Control, old value of data items are kept, even after ..... of data item.

b) In Multi Version two phase locking, ... types of locking is present.

c) In Multi Version Concurrency Control, a ..... version is created, whenever there is an update.

13.7 VALIDATION (OPTIMISTIC) CONCURRENCY CONTROL TECHNIQUES

In all the concurrency control techniques that we have discussed so far, some kind of checking is done before a database operation can be executed. This kind of checking adversely affects the performance of the database by slowing down the transaction execution process. For example, in case of locking, a check is done to verify whether the item accessed by the transaction is locked or not. This gives extra overhead, thereby slowing down the transaction execution time.

In case of validation concurrency control technique, which is also known as Optimistic concurrency control technique, no checking is done while the transaction is executing. That is, a transaction is started without

locking the database item. For example, in case of update operation in a transaction, it is not directly applied to the database, until the transaction reaches the end. During the execution phase of a transaction, all updates are applied to the local copy of the database. when transaction reaches the end of the execution, it goes through a phase known as **validation phase,** which checks whether any of the transaction update violate serializability. The information needed by the validation phase is kept in the system.

If serializability is not broken or violated, the transaction is committed and the database is updated from the local copies, otherwise, the transaction is aborted and restarted later.

The different phases of validation check concurrency control protocol are:

1) **Read Phase:** A transaction read values of committed data items from the database. However, because transaction is committed, the updates or writes are applied only to the local copies of the data item kept in the transaction workspace.
2) **Validation Phase:** Checking is performed to ensure that update from a transaction will not violate the serializability.
3) **Write Phase:** Based on the status of the validation phase, if the validation phase is successful, the transaction updates are applied to the database and if it is unsuccessful, the updates are discarded and the transaction is restarted later.

In *optimistic concurrency control* technique, a transaction proceeds without any checking, that is, if proceeds with a minimum overhead, until it reaches the validation phase. If the interference between the transaction is less, they are validated successfully, and result is updated to the database. On the other hand, when the interference among the transaction is high, it results in unsuccessful validation, and the results are discarded and restarted later. The optimistic technique assumes that there is less interference among the transactions and it hardly needs any checking during the transaction execution.

In optimistic algorithm, during the validation phase of the Transaction T_i, the algorithm checks that T_i does not interfere with any committed

transactions or with any transaction which is currently running in their validation phase. The validation phase of ant transaction T_i, checks that for any other transaction T_j that is either committed or it is in its validation phase holds one of the following conditions:

1) T_j completes its write phase, before T_i starts its read phase.
2) T_i starts its write phase only after T_j completes its write phase and read_set of T_i has no item in common with the write_set of T_j.
3) The read_set and write_set of T_i have no item is common with the write_set of T_j and T_i completes its read phase before T_i completes its read phase.

The above three conditions checked serially, and stars with the condition 1. If it is false, it checks the condition 2, and if it is also false, then only it checks the condition 3. If any one of the above three conditions held, it means there is no interference and T_i is validated successfully. On the other hand, it none of the above three conditions holds, the validation phase of T_i fails and is aborted and restarted later.



CHECK YOUR PROGRESS

Q.4: State True or False:

a) Optimistic Concurrency Control is also known as validation 1 technique.

b) In Optimistic Concurrency Control technique, a check is done while a transaction is executing.

c) In read phase, a transaction read values of committed data items from the database.

d) Validation phase ensures that serializability is not maintained.

e) In Optimistic Concurrency Control, transaction execution proceeds with a minimum overhead until the validation phase is reached.

13.8 GRANULARITY OF DATA ITEM

All the concurrency control technique categorize a database in the form of a set of data items. The different data items that may be considered are as follows:

a) It may as small as a field value of a database record.
b) It may be a database record.
c) A disk block.
d) A whole file.
e) A whole database.

The granularity of database is very important as far as the performance of concurrency and recovery is concerned.

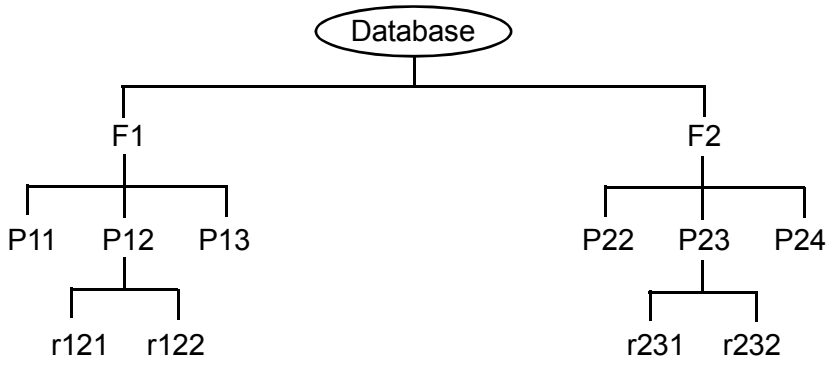
The size of the data item is called *data item granularity*. The granularity can be considered as– *fine granularity*, which refers to the smallest size of the data items and ***coarse granularity*** which refers to the largest size of the data items.

It is to be mentioned here that the degree of concurrency is dependent on the granularity of data item. When the size of the data item is large, it lowers the degree of concurrency, so the number of concurrent transaction is less. This is because, when the entire disk block is locked by a transaction, any other transaction trying to access items belonging to the same disk block is forced to wait. On the other hand, if the database size is small, for example if it is a single record only, then many transactions can proceed parallelly, because it would be locking a different data item record.

On the other hand, when the size of the data item is small, more number of data items are present in the database. Since, every item is associated with a lock, there will be more number of locks and the lock manager need to manage a large number of locks which may be active at the same time. Since, more number of lock and unlock operations need to be performed, It increases the overall overhead on the database. At the same time, the size of the lock table also increases.

The appropriate size of the data item is very difficult to decide on and it depends on the type and number of transactions executing at one

point of time. As the best size of the granularity depends on the transaction, so database system should support the different level of granularity for a set of transactions. A hierarchy of granularity is depicted in the figure below:



Suppose a transaction T1 wants to update some of the records under the file F1. So, it will lock all the items down the hierarchy starting from F1. Now, suppose another transaction T2 want to read a record, which is also belongs to the file F1. The transaction manager of the Database System will check the compatibility of the requested lock, with the locks that are already held. While verifying the compatibility, if at any point of time, a conflict occurs, the request for the read(shared) lock is denied and the transaction T2 is denied and must wait.

But when T2 request for the read (shared) lock comes first, T2's request is granted. At this time, it is very difficult to check all the need for a lock conflict, and the time consumption is also very high. So, it becomes very much in efficient and in tern defeat the purpose of having multiple granularity locks.

To overcome the above problem and to implement multiple granularity level locking, another type of lock known as ***intention lock*** is implemented. The objective of this type of lock is to indicate along the path from the root to the desired node, what type of lock it would required. There are three types of intention locks namely,

1) Intention-Shared (IS)– which indicates that a shared lock will be requested on some descendent nodes.
2) Intention-Exclusive (IX)– which indicates that a shared lock will be requested on some descendent nodes.

3) Shared-Intention-Exclusive (SIX)– it indicates that the current node is locked in shared lock but an exclusive lock will be requested on some descendent nodes.

Apart from the different types of intention lock some appropriate locking protocol also need to be implemented for better performance.



CHECK YOUR PROGRESS

Q.5: Fill up the gapes

a) Field value is the ..... granularity of data item.

b) Fine grain refers to the ..... data item size.

c) ..... grain refers to the large data item size.

d) Best data item size depends on the ..... type involved.

e) Intention lock is a type of lock applied to ..... granularity lock.

13.9 DATABASE RECOVERY CONCEPT

Recovery from transaction failure can be defined as the restoring of the database to the most recent consistent state just before the time of failure. To ensure this, the system must keep a record of all the changes that took place to the database due to the execution of various transactions. This information is maintained by the transaction log. In case of failure, with the help of this lock, the database can be recovered.

13.9.1 Category of Recovery Algorithm

There may be various possibilities due to which recovery becomes an important issue. In case of huge damage, which is mainly caused due to the catastrophic failure, the database can be recovered by restoring the database kept as a backup or from the archive and reconstruct a more current state by reapplying or redoing the operations of committed transaction from the backup log up to the

time of failure. On the other hand when the database is not physically damaged but has become inconsistent due to various inconsistencies by undoing some operations. There may be a need of redoing some of the operation in order to restore a consistent state of the database.

a) **Deferred Update and Immediate Update:** To recover from the non-catastrophic failure, there are two main techniques applied. They are:

   i) **Deferred Update:** According to deferred update, the database is not physically updated to the disk until a database reaches its commit point. When a transaction reaches a commit point, then only the transaction update is recorded. Before reaching the commit point, all transaction updates are recorded to the local database. In case transaction fails before reaching the commit point, it is not required to make any changes to that database, so no undo is needed. There may be a necessity to redo some of the operations from the transaction logs. Because their effect may not have been recorded in the database. So, deferred update is also sometime known as no-undo/redo algorithm.

   ii) **Immediate Update:** In this technique, the database may be update by some operations of the transaction before the transaction reaches its commit point. However, these operations are also recorded in the log on the disk by force writing before they are applied to the database, thereby making the recovery still possible. In case a transaction fails before reaching the commit point, the effect of its operation on the database must be undone i.e. the transaction must be rolled back. So, in case of immediate update, both undo and redo is needed. So, this algorithm is also known as **UNDO/REDO** algorithm. There may be another variation of the algorithm where all updates are recorded in the database before a transaction is committed requiring undo only, so it is known as undo/no-redo algorithm.

b) Caching of disk Blocks: The caching of disk block typically is an operating system function. But as it is one of the important parameter as far as the efficiency of recovery procedure is concerned, it is handled by the DBMS by calling low level operating system routines. Typically a collection of in-memory buffer called the DBMS cache, is kept under the control of the DBMS for the purpose of holding these buffers. Associated with each buffer in the cache is a **dirty bit**, which can be included in the directory entry to indicate whether or not the buffer has been modified. When a page is first read from the database disk into a cache buffer, the cache directory is updated with the new disk address, the dirty bit is set to 0. As soon as the buffer is modified the dirty bit for the corresponding directory entry is set to 1. When the buffer content are replaced from the cache, the content must first be written back to the corresponding disk page only if its dirty bit is set to 1.

Another bit known as pin-unpin bit also used to indicate whether a page in the cache can be written back to the disk or not. Here, pinned (bit value 1) indicates page cannot be written back.

As far as flashing of modified buffer back to the disk is concerned, two main strategies are employed. They are:

i) In-place updating

ii) Shadowing

In case of in-place updating, it writes the buffer to the same original disk location, thus overwriting the old value of any changed data item on disk. Hence, only a single copy of each database disk block is maintained.

In the case of shadowing on the other hand, it writes an updated buffer at a different disk location, so multiple versions of data items can be maintained. The old value of data item before updating is called the before image (BFIM) and the new value after update is called after image (AFIM). In shadowing both BFIM and AFIM are maintained so the necessity of maintaining log in eliminated.

c) **Write-ahead Logging:** When in place updating is used, it is necessary to use a log for recovery. Here the recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry and the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database on the disk. This process in known as write_ahead logging.

d) **Steal/No-Steal Approach:** In No-Steal approach a cache page updated by a transaction cannot be written to disk before the transaction commits. The pin-unpin bit is used to realize the approach. The bit indicates if a page cannot be written back to the disk. On the other hand, in Steal approach, a cache page updated by a transaction can be written to disk before the transaction commits.

e) **Force/No-Force Approach:** In Force approach all cache pages updated by a transaction are immediately written to disk when the transaction commits and otherwise it is called No-Force. A Typical Database systems use a Steal/No-Force strategy because of the following advantages:
   - Steal approach avoids the need for a very large buffer space to store all updated pages in memory.
   - No-Force approach provides the advantage of keeping an updated page of a committed transaction in the memory when another transaction needs to update it, thus it eliminates the I/O cost of reading that page again from disk.

f) **Check Point:** Check point is another type of entry that is maintained in the log. The recovery manager of a DBMS must decide at what intervals to put a check point. The interval may be measured in terms of time say every after 'm' second or minute or in the number 't' of committed transactions since the last checkpoint. Here both 't' or 'm' are system parameters. All transactions committed before a checkpoint entry do not need to redo their write operations.

Checkpoint creating process consists of the following actions:

a) Suspend execution of transactions temporarily.

b) Force-write all main memory buffers that have been modified to disk.

c) Write a checkpoint record to the log, and force-write the log to disk.

d) Resume executing transactions.

The time needed to force-write buffers to disk may delay transactions execution and can be a big performance issue.

g) Transaction Rollback: There may be the necessity of roll back the transaction in some situation when a transaction fails after updating the database. In case of transaction rollback the write operations are undone, and it must be restored to their previous values (BFIS).

Another variation of rollback known as cascading rollback is also there where if transaction T1 is rolled back, any Transaction T2 that has read the value of some data item X written by T1 must also be rolled back and so on.

To improve the performance, Fuzzy checkpoint is used to reduce the delay by resuming the execution of the transactions after the checkpoint record is written to the log without having to wait for force-write step to be completed. The previous checkpoint is valid until the force-write step is completed. This in done by the system by maintaining a pointer to the valid checkpoint and changes it to the new checkpoint after the success of the force-write step.



CHECK YOUR PROGRESS

Q.6: Fill up the gaps:

a) The information about the changes applied to the database item by various transactions are kept by the .....

b) Deferred update, updates the database only after the transaction is .....

c) In case of immediate update, the database is updated ..... a transaction is committed.

d) AFIM stands for .....

e) BFIM stands for .....

Q.7: State True or False:

a) Dirty bit indicates whether the buffer is modified or not.

b) In shadowing, the updated buffer is written at one location.

c) In case of Force Approach, database updates are written to the disk immediately.

d) Creation of checkpoint is the responsibility of recovery manager.

e) Rollback refers to restoring of the database to their previous value.



13.10 LET US SUM UP

- When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved. There should be some mechanism in the system to control the interaction among the concurrent transactions.
- Concurrency control is the process of managing simultaneous execution of transactions in a multiprocessing database system without having them interfere with one another.
- This control mechanism enables many transactions to access the same database at the same time without interfering with each other.
- Concurrency control is important because the simultaneous execution of transactions over shared database can create several data integrity and consistency problem.
- In case of shared data, locking is a mechanism commonly used to solve the problem of synchronized access.
- Locks are used as means of synchronizing the access by concurrent

transactions to the database items.

- Two-phase locking is a method or a protocol of controlling concurrent processing in which all locking operations precedes the first unlocking operations.
- Timestamp is a unique identifier created by the database management system to identify the relative starting time of a transaction.
- The main benefit of multiple version concurrency control is to help the scheduler avoid rejecting operations that arrive too late.



13.11 FURTHER READING

1) Elmasri, Ramez; Navathe, Shamkant B. *Fundamentals of Database Systems*. Pearson Education, Fifth Edition.

2) Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Database System Concepts*, Tata McGraw Hill, Sixth Edition.



13.12 ANSWER TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) variable, b) lock conversion, c) Growing Phase, d) shrinking phase, e) aborts

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

Ans. to Q. No. 3: a) updation, b) three, c) new

Ans. to Q. No. 4: a) True, b) False, c) True, d) False, e) True

Ans. to Q. No. 5: a) smallest, b) small, c) Coarse, d) transaction, e) multilevel

Ans. to Q. No. 6: a) system log, b) committed, c) before, d) After Image, e) Before Image

Ans. to Q. No. 7: a) True, b) False, c) True, d) True, e) True



13.13 MODEL QUESTION

Q.1: What is a lock? Explain the function of lock in concurrency.

Q.2: What is two phase locking protocol?

Q.3: Explain how two phase locking protocol guarantee serializability.

Q.4: What is a timestamp? Explain.

Q.5: What is certify lock? Explain the advantages and disadvantages of it.

Q.6: What is intention lock? Describe its different types.

Q.7: What do you mean by catastrophic failure?

Q.8: Describe the function of a Dirty bit.

*** ***** ***

UNIT 14: DATABASE RECOVERY

UNIT STRUCTURE

14.1 Learning Objectives

14.2 Introduction

14.3 Database Failure

14.4 Database Recovery

14.5 Backup of Database for Catastrophic Failure

14.6 Recovery for Non-Catastrophic Failures

14.6.1 Deferred Update

14.6.2 Immediate Update

14.7 Let Us Sum Up

14.8 Further Reading

14.9 Answers to Check Your Progress

14.10 Model Questions

14.1 LEARNING OBJECTIVES

After going through this unit, you will be able to

- understand the concept of database failure and the different types of failures that can occur in a database system.
- learn about the importance of database recovery and the steps involved in the recovery process.
- understand the need for backup of a database to prevent data loss in the event of a catastrophic failure.
- learn about the best practices for implementing immediate or deferred updates to recover from non-catastrophic failures.
- understand the importance of having a disaster recovery plan in place to minimize the impact of a catastrophic failure and ensure business continuity.

14.2 INTRODUCTION

We have already been acquainted with various important concepts related with database and DBMS. We have also learnt to insert, delete,

update and other associated operations in a database through SQL commands. In this unit we will learn how a database can be recovered.

In today's digital age, data is considered one of the most valuable assets for individuals and businesses alike. As such, it is crucial to have effective measures in place to ensure the security and integrity of data stored in databases. However, despite the best efforts, database failures can still occur due to various reasons such as hardware or software malfunctions, human errors, or natural disasters.

This unit on Database Failure and Recovery aims to provide students with a comprehensive understanding of the common types of database failures, their causes, and the methods used for recovery. The unit covers various topics such as database backup strategies, catastrophic and non-catastrophic failures, and deferred and immediate updates.

By the end of this unit, lerners will be equipped with the necessary knowledge and skills to identify, diagnose, and recover from database failures. They will also learn how to implement effective backup and recovery strategies to ensure the continuity of database operations in case of unexpected failures.

14.3 DATABASE FAILURE

A database failure occurs when a database system is unable to perform its functions as expected. Database failures can be caused by a variety of reasons, such as hardware or software issues, network problems, human errors, or natural disasters.

- Database failures can lead to a range of negative consequences, including:
- Data loss: A database failure can result in the loss of important data that may be critical to an organization's operations.
- Downtime: A database failure can cause downtime, which can result in lost productivity and revenue.
- Security breaches: A database failure can lead to security breaches, as hackers or unauthorized users may exploit vulnerabilities in the system to gain access to sensitive data.

- System crashes: A database failure can cause the entire system to crash, which can be difficult and time-consuming to recover from.

To prevent database failures, it's important to have regular backups, implement redundancy and failover mechanisms, monitor system performance, and perform regular maintenance and updates. In case of a failure, it's important to have a disaster recovery plan in place to quickly restore the database system to its previous state.

Top of Form

14.4 DATABASE RECOVERY

Database recovery is the process of restoring a database system to its previous state after a failure or an interruption. Recovery is necessary to ensure the integrity and availability of data in the database system.

The recovery process typically involves the following steps:

- Identification of the failure: The first step in the recovery process is to identify the type and cause of the failure. This can be done through log analysis, error messages, and system alerts.
- Backup restoration: If a recent backup exists, the next step is to restore the database from that backup. This can be a full back up or a differential/incremental backup depending on the nature of the failure.
- Transaction log replay: If a backup does not exist or if the backup is not recent, the transaction logs are used to replay the transactions that were not yet committed before the failure occurred. This process ensures that the database is restored to a consistent state.
- Testing and validation: Once the recovery process is complete, it's important to test and validate the database to ensure that it's functioning as expected.
- Post-mortem analysis: Finally, a post-mortem analysis should be performed to identify the root cause of the failure and to implement corrective actions to prevent similar failures in the future.

In addition to these steps, it's important to have a disaster recovery plan in place that outlines the steps to be taken in the event of a catastrophic failure. This plan should include backup and recovery procedures, failover

mechanisms, and communication protocols.

By following these best practices, database recovery can be a seamless process that ensures the integrity and availability of data in the database system.

14.5 BACKUP OF DATABASE FOR CATASTROPHIC FAILURE

Regular backups of a database are essential to minimize the risk of data loss in the event of a catastrophic failure. A catastrophic failure refers to a severe failure that causes significant damage or disruption to a database system, such as a hardware failure, software corruption, or a natural disaster. Here are some best practices for backing up a database for catastrophic failure:

- Schedule regular backups: Set up a regular backup schedule that meets the needs of your organization. This can be daily, weekly, or monthly backups depending on the amount and importance of data.
- Use multiple backup methods: Use different backup methods, such as full, differential, and incremental backups to ensure maximum data protection.
- Store backups offsite: Store backups in a secure offsite location, such as a
- cloud storage service, to ensure data availability in the event of a catastrophic failure at your primary site.
- Test backup and recovery procedures: Regularly test your backup and recovery procedures to ensure they are working as expected and can quickly restore data in the event of a failure.

Maintain backups for an appropriate duration: Keep backups for an appropriate duration that aligns with your organization’s retention policies and regulatory requirements.

By implementing these best practices, you can minimize the risk of data loss and ensure that your organization can quickly recover from a catastrophic failure of your database system.



CHECK YOUR PROGRESS

Q.1: Choose the correct option:

(a) What is a catastrophic failure in a database system?

(i) A minor issue that can be resolved quickly

(ii) A severe failure that causes significant damage or disruption

(iii) A planned outage for maintenance

(iv) A temporary interruption of service

(b) What is the purpose of database recovery?

(i) To create a backup of the database

(ii) To restore the database system to its previous state after a failure or interruption

(iii) To monitor system performance and conduct regular maintenance

(iv) To implement redundancy and failover mechanisms

(c) What is the first step in the database recovery process?

(i) Backup restoration

(ii) Transaction log replay

(iii) Identification of the failure

(iv) Testing and validation

(d) What is the purpose of regular backups in a database system?

(i) To monitor system performance and conduct regular maintenance

(ii) To create redundancy and failover mechanisms

(iii) To ensure data availability in the event of a catastrophic failure

(iv) To prevent security breaches

(e) What are the consequences of a database failure?

(i) Increased productivity and revenue

(ii) Security breaches and unauthorized access to sensitive data

(iii) Improved system performance and efficiency

(iv) Redundancy and failover mechanisms

14.6 RECOVERY FOR NON-CATASTROPHIC FAILURES

Non-catastrophic failures refer to less severe database failures that are not as disruptive as catastrophic failures but still need to be addressed promptly to avoid data loss and minimize downtime. Examples of non-catastrophic failures include software bugs, human errors, and network disruptions.

Here are some steps to follow for recovery from non-catastrophic failures:

- Identify the cause of the failure: The first step is to identify the root cause of the failure, which could be anything from a software bug to a user error.
- Rollback or restore: Depending on the nature of the failure, you may need to rollback the changes made to the database or restore the database from a recent backup.
- Reapply changes: If you rollback the changes, you may need to reapply the changes that were made before the failure occurred. If you restore the database from a backup, you may need to apply any changes that were made after the backup was taken.
- Verify the recovery: After the recovery process is complete, it's important to verify that the database is functioning as expected and that all data is intact.
- Prevent future failures: Finally, take steps to prevent future failures by addressing the root cause of the failure, implementing system updates and patches, and ensuring that users are properly trained on database usage.

By following these steps, you can recover from non-catastrophic failures and ensure that your database system remains reliable and functional.

14.6.1 Deferred Update

Deferred update is a technique used in database recovery to ensure data consistency in non-catastrophic failures. Non-

catastrophic failures refer to less severe database failures that do not result in complete data loss or system downtime, such as software errors, network failures, or user errors.

In deferred update recovery, changes made to the database are not immediately written to disk. Instead, they are kept in memory until a commit statement is issued, indicating that the transaction is complete and all changes should be permanently saved. This approach allows for the recovery of a consistent state of the database in the event of a non-catastrophic failure.

Here are some key considerations when using deferred update for non-catastrophic failures:

- Ensure data consistency: Deferred update must be used in conjunction with transaction management to ensure data consistency. If a transaction is incomplete at the time of failure, any changes made by the transaction must be rolled back to maintain data consistency.
- Monitor transaction duration: Long-running transactions can increase the risk of failure and compromise data consistency. To mitigate this risk, limit the duration of transactions and ensure that they complete within a reasonable time frame.
- Implement recovery procedures: Have recovery procedures in place to ensure that the database can be restored to a consistent state in the event of a failure. This may involve rolling back incomplete transactions or replaying committed transactions from a transaction log.

By implementing deferred update recovery techniques, you can ensure data consistency in the event of a non-catastrophic failure and minimize the risk of data loss or corruption.

14.6.2 Immediate Update

Immediate update of recovery is essential to minimize the impact of non-catastrophic failures and to ensure that the database system is available as soon as possible. Non-catastrophic failures

are issues that do not cause severe damage or disruption to a database system, such as a transaction failure, a network connectivity issue, or a software error.

Here are some best practices for immediate update of recovery for non-catastrophic failures:

- Identify the root cause: Quickly identify the root cause of the failure to determine the appropriate recovery action.
- Determine the severity of the failure: Assess the severity of the failure to determine whether it requires immediate recovery or can be resolved in the next scheduled maintenance window.
- Implement the appropriate recovery action: Based on the severity of the failure, implement the appropriate recovery action. This may include rolling back the transaction, restarting the application or database, or restoring from a backup.
- Communicate with stakeholders: Keep stakeholders informed of the situation, the recovery action taken, and the estimated time to restore the system to normal operations.
- Perform a post-mortem analysis: After the system is restored to normal operations, perform a post-mortem analysis to identify the root cause of the failure and implement corrective actions to prevent similar failures in the future.

By implementing these best practices, you can minimize the impact of non-catastrophic failures and ensure that your database system is available and performing as expected.

The immediate update algorithm is a recovery technique used in database systems to ensure that any changes made to the database are immediately recorded in non-volatile storage such as disk. This algorithm helps to ensure that data is not lost in the event of a system failure or crash.

The basic steps involved in the immediate update algorithm are as follows:

1. Any changes made to the database are immediately recorded in the database buffer cache.

2. The changes are then written to the transaction log in non-volatile storage, such as disk.
3. Once the changes have been written to the transaction log, they are considered durable and can be safely committed to the database.
4. The transaction is marked as complete in the transaction log.

If a system failure occurs before the changes have been written to the transaction log, the immediate update algorithm ensures that the changes are not lost. When the system is restarted, the transaction log is used to recover the database to the point in time just before the failure occurred.

The immediate update algorithm is a simple and efficient technique for ensuring database recovery. However, it can impact performance as changes are immediately recorded to disk. As such, it is important to monitor the performance of the database to ensure that it is not adversely affected.



CHECK YOUR PROGRESS

Q.2: Fill in the blanks:

(a) Deferred update is a technique used in database recovery to ensure _____ in non-catastrophic failures.

(b) In deferred update recovery, changes made to the database are not immediately written to disk, instead, they are kept in _____ until a commit statement is issued.

(c) To ensure data consistency when using deferred update, incomplete transactions must be _____ if a failure occurs.

(d) Immediate update of recovery is essential to minimize the impact of _____ failures and to ensure that the database system is available as soon as possible.

(e) After restoring the system to normal operations, it is important to perform a _____ analysis to identify the root cause of the failure and implement corrective actions.

UNDO/NO-REDO recovery algorithm

The UNDO/NO-REDO recovery algorithm is a technique used in database systems to recover from a system failure or crash. It is a two-phase recovery algorithm that involves undoing uncommitted transactions and then skipping the redo phase.

The basic steps involved in the UNDO/NO-REDO recovery algorithm are as follows:

- Identify all transactions that were active at the time of the system failure.
- For each active transaction, undo any changes made by the transaction that were not committed to the database.
- Once all uncommitted changes have been undone, mark the transaction as aborted.
- Skip the redo phase and proceed directly to the transaction commit phase.

The UNDO phase of the recovery algorithm ensures that any uncommitted changes are rolled back, so that the database is left in a consistent state. The NO-REDO phase is possible because all committed transactions will have already written their changes to the database before the system failure occurred.

One advantage of the UNDO/NO-REDO recovery algorithm is that it can be faster than other recovery techniques that involve redoing all transactions. This is because the algorithm only processes uncommitted transactions, rather than all transactions.

However, one disadvantage of the UNDO/NO-REDO recovery algorithm is that it may lead to data loss if the system failure occurred during the commit phase. In this case, changes made by committed transactions will not be written to disk, and will be lost when the UNDO phase rolls back uncommitted changes.

Overall, the UNDO/NO-REDO recovery algorithm is a useful technique for recovering from system failures in database systems, but it should be used with caution to avoid data loss.

UNDO/REDO recovery algorithm

The UNDO/REDO recovery algorithm is a technique used in database systems to recover from a system failure or crash. It is a two-phase recovery algorithm that involves undoing uncommitted transactions and then redoing committed transactions.

The basic steps involved in the UNDO/REDO recovery algorithm are as follows:

- Identify all transactions that were active at the time of the system failure.
- For each active transaction, undo any changes made by the transaction that were not committed to the database.
- Once all uncommitted changes have been undone, mark the transaction as aborted.
- Redo all committed transactions, starting from the oldest transaction, by applying their changes to the database.

The UNDO phase of the recovery algorithm ensures that any uncommitted changes are rolled back, so that the database is left in a consistent state. The REDO phase is necessary because changes made by committed transactions may not have been written to disk before the system failure occurred.

One advantage of the UNDO/REDO recovery algorithm is that it provides a high degree of data durability and consistency, since all committed changes are applied to the database during the REDO phase. This means that no data is lost and the database is left in a consistent state.

However, one disadvantage of the UNDO/REDO recovery algorithm is that it can be slow, especially when there are a large number of transactions to be redone. This is because the algorithm needs to scan the transaction log to identify committed transactions that need to be redone.

Overall, the UNDO/REDO recovery algorithm is a useful technique for recovering from system failures in database systems, especially when data durability and consistency are important.

However, the performance impact of the algorithm should be carefully considered when designing and tuning database systems.



CHECK YOUR PROGRESS

Q.3: Fill in the blanks:

(a) The UNDO/NO-REDO recovery algorithm is a _____ technique used in database systems.

(b) In the UNDO/NO-REDO recovery algorithm, the UNDO phase ensures that _____ changes are rolled back.

(c) The NO-REDO phase in the UNDO/NO-REDO recovery algorithm is possible because all _____ transactions have already written their changes to the database.

(d) One advantage of the UNDO/REDO recovery algorithm is that it provides a high degree of data _____ and consistency.

(e) The performance impact of the UNDO/REDO recovery algorithm should be carefully considered when designing and tuning _____ systems.



14.7 LET US SUM UP

- Database failures can result in data loss, downtime, security breaches, and system crashes, which can negatively impact an organization's operations and revenue.
- To prevent database failures, it's important to have regular backups, implement redundancy and failover mechanisms, monitor system performance, and perform regular maintenance and updates.
- Database recovery involves identifying the failure, restoring from backups or replaying transaction logs, testing and validating the system, and performing a post-mortem analysis to prevent future failures.
- A disaster recovery plan is important to have in place to quickly restore

the database system to its previous state in case of a catastrophic failure.

- Best practices for backing up a database for catastrophic failure include scheduling regular backups, using multiple backup methods, storing backups offsite, testing backup and recovery procedures, and maintaining backups for an appropriate duration.
- Non-catastrophic failures in databases include software bugs, human errors, and network disruptions. To recover from such failures, one needs to identify the root cause of the failure, rollback or restore the database, reapply changes, verify the recovery, and take steps to prevent future failures.
- Deferred update is a technique used in database recovery to ensure data consistency in non-catastrophic failures. It involves keeping changes made to the database in memory until a commit statement is issued to indicate that the transaction is complete and all changes should be permanently saved.
- Immediate update algorithm is another recovery technique used in database systems to ensure that any changes made to the database are immediately recorded in non-volatile storage. This algorithm helps to ensure that data is not lost in the event of a system failure or crash.
- To minimize the impact of non-catastrophic failures, one should quickly identify the root cause of the failure, determine its severity, implement the appropriate recovery action, communicate with stakeholders, and perform a post-mortem analysis to identify the root cause of the failure and implement corrective actions to prevent similar failures in the future.



14.8 FURTHER READINGS

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

2) Ramakrishnan, R. & Gehrke, J. (2000). Database Management

Systems. McGraw Hill.

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



14.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1. (a) (ii)(b) (ii)(c) (iii)(d) (iii)(e) (ii)

Ans. to Q. No. 2. (a) data consistency (b) memory(c) rolled back (d) non-catastrophic (e) post-mortem

Ans. to Q. No. 3. (a) recovery (b) uncommitted (c) committed (d) durability (e) database



14.10 MODEL QUESTIONS

Q.1: What are the common causes of database failure, and what are the potential consequences of such failures?

Q.2: What steps can be taken to prevent database failures, and what measures should be taken in the event of a failure?

Q.3: What is the process of database recovery, and what are the key steps involved in this process?

Q.4: Why is it important to schedule regular backups of a database, and what are some best practices for backing up a database in the event of a catastrophic failure?

Q.5: What is a disaster recovery plan, and how can it help organizations prepare for and respond to database failures?

Q.6: What is a non-catastrophic failure in a database system?

Q.7: What are some key considerations when using deferred update for database recovery?

Q.8: What are some best practices for immediate update recovery for non-catastrophic failures?

Q.9: How does the immediate update algorithm ensure data recovery in the event of a system failure or crash?

Q.10: What steps should be taken to prevent future failures after recovering from a non-catastrophic failure in a database system?

Q.11: What is the difference between the UNDO/NO-REDO and UNDO/REDO recovery algorithms in database systems? What are the advantages and disadvantages of each approach?

Q.12: How does the UNDO/NO-REDO recovery algorithm work in database systems? What are the steps involved and how does it ensure data consistency and prevent data loss?

Q.13: When would you choose to use the UNDO/REDO recovery algorithm over the UNDO/NO-REDO recovery algorithm in a database system? What factors should you consider when making this decision?

*** ***** ***

UNIT 15: SECURITY AND AUTHENTICATION

UNIT STRUCTURE

15.1 Learning Objectives

15.2 Introduction

15.3 Database Security Issue

15.4 Discretionary Access Control Based On Granting and Revoking Privileges

15.5 Mandatory Access Control

15.6 Role Based Access Control

15.7 Encryption and Public Key Infrastructures

15.8 Let Us Sum Up

15.9 Further Reading

15.10 Answers to Check Your Progress

15.11 Model Questions

15.1 LEARNING OBJECTIVES

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

- describe database security
- define discretionary access control
- describe mandatory access control
- describe role based access control
- describe encryption and public key infrastructures.

15.2 INTRODUCTION

We have already learned various concepts and techniques of database recovery in the previous unit. This unit discusses the techniques used for protecting the database against the person who are not authorized to access either certain part of database or the whole database.

15.3 DATABASE SECURITY ISSUE

Database security is a very broad area that addresses many issues, including the following:

- Legal and ethical issue regarding the right to access certain information. Sometime information may be deemed to be private and cannot be accessed legally by unauthorized persons.
- Policy issue at the governmental, institutional, or corporate level as to what kind of information should not be made publicly available- for example credit rating and personal medical record.
- System related issues such as the system levels at which various security functions should be enforced. For example, where a security functions should be handled at the physical hardware level, the operating system level or the DBMS level.
- The need in some organizations to identify multiple security levels and to categorize the data users based on these classifications; for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications must be enforced.

Threats to Databases: Threat to database results in the loss or degradation of some or all of the following security goals: integrity, availability, and confidentiality.

- **Loss of integrity:** Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to data by either intentional or accidental acts. If the loss of the system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decision.
- **Loss of availability:** Database availability refers to making objects available to human user or a program to which they have a legitimate right.
- **Loss of confidentiality:** Database confidentiality refers to the protection of data from unauthorized disclosure.

Database Security and the DBA: The database administrator (DBA) is the central authority for managing a database system. The DBA's

responsibilities include granting privileges to the users who need to use the system and classifying users and data in accordance with the policy of the organization. The DBA has a DBA account in the DBMS, sometimes called a system or super user account, which provides powerful capabilities. The DBA is responsible for the overall security of the database system.

Access Protection, User Accounts, and Database Audits: Whenever a person or group of persons need to access a database system, the individual or group must first apply for a user account. The DBA will then create a new **account number** and **password** for the user if there is a legitimate need to access the database. The user must **log in** to the DBMS by entering account number and password whenever database access is needed.

The database system must also keep track of all operations on the database that are applied by a certain user throughout each **login session**.

To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify ***system log***, which includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash.

If any tampering with the database is suspected, a **database audit** is performed, which consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period. A database log that is used mainly for security purposes is sometimes called an **audit trail**.



CHECK YOUR PROGRESS

Q.1: Choose the correct answer:

i) Modification of data includes–

a) Creation b) Insertion

c) Modification d) All of the above

ii) DBA stands for–

a) Database Access b) Database Administrator

c) Data Bound Administrator d) None of the above

iii) A database log that is used mainly for security purposes is sometimes called–

a) System log b) Database Audit

c) Database trial d) Audit trial

15.4 DISCRETIONARY ACCESS CONTROL BASED ON GRANTING AND REVOKING PRIVILEGES

The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.

There are two levels for assigning privileges:

- **The account level:** At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- **The relation level:** At this level, the DBA can control the privilege to access each individual relation or view in the database.

The privileges at the account level apply to the capabilities provided to the account itself and can include the CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation; the CREATE VIEW privilege; the ALTER privilege, to apply schema changes such as adding or removing attributes from the relation; the DROP privilege, to delete relations or view; the MODIFY privilege, to insert, delete, or update tuples, and the SELECT privilege, to retrieve information from the database using a SELECT query. Notice that these account privileges apply to the account in general. If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account.

The second level of privileges applies to the relational level, whether they are base relations or virtual relations.

To control the granting and revoking of relation privileges, each relation R in a database is assigned an owner account, which is typically the account that was used when the relation was created in the first place. The owner relation is given all the privileges on that relation.

In SQL the following types of privileges can be granted to each individual relation R:

- SELECT privilege on R: Gives the account retrieval privilege.
- MODIFY privileges on R: This gives the account the capability to modify the tuples of R.
- REFERENCE privilege on R: This gives the account the capability to reference relation R when specifying integrity constraints. This privilege can also restrict specific attributes of R.

The mechanism of views is an important discretionary authorization mechanism in its own right. For example if the owner A of relation R wants another account B to be able to retrieve only some field of R, then A can create a view V of R that includes only those attribute and then grant SELECT on V to B.

Revoking Privilege: In some case it is desirable to grant privilege to a user temporarily. For example, the owner of relation may want to grant the SELECT privilege to a user for a specific task and revoke the privilege once the task is completed. Hence, a mechanism for revoking privileges is needed. In SQL REVOKE command is included for the purpose of cancelling privileges.

Propagation of Privileges using the GRANT OPTION: Whenever the owner A of relation R grants privilege on R to another account B, the privilege can be given to B with or without the GRANT OPTION. If the GRANT OPTION is given, this means that B can also grant privilege on R to other accounts. Suppose that B is given the GRANT OPTION by A and that B grants the privilege on R to a third account C, also with GRANT OPTION. In this way, privilege on R can propagate to other account without the knowledge of the owner of R. If the owner account A now revoke the privilege granted to B, all the privilege that B propagate based on that privilege should automatically be revoked by the system.

Let us consider an example. Suppose that the DBA creates four accounts A1, A2, A3, and A4 and wants only A1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

GRANT CREATETAB TO A1;

In SQL2 the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command as follows:

CREATE SCHAMA EXAMPLE AUTHORIZATION A1;

Now user account A1 can create tables under the schema called EXAMPLE.

Suppose that A1 creates the two base relations EMPLOYEE and DEPARTMENT; A1 is then owner of these two relations and hence all the relation privileges on each of them.

Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts:

GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;

EMPLOYEE

NAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO

DEPARTMENT

DNUMBER	DNAME	MGRSSN

Suppose that A1 wants to allow A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts.

A1 can issue the command:

GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3 WITH GRANT OPTION;

A3 can grant the SELECT privilege on the EMPLOYEE relation to A4 by issuing:

GRANT SELECT ON EMPLOYEE TO A4;

(Notice that A4 cannot propagate the SELECT privilege because GRANT OPTION was not given to A4.)

Suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 can issue:

REVOKE SELECT ON EMPLOYEE FROM A3;

(The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.)

Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5.

A1 then create the view:

```
CREATE VIEW A3EMPLOYEE AS
SELECT NAME, BDATE, ADDRESS
FROM EMPLOYEE
WHERE DNO = 5;
```

After the view is created, A1 can grant SELECT on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;
```

Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE;

A1 can issue:

```
GRANT UPDATE ON EMPLOYEE (SALARY) TO A4;
```

(The UPDATE or INSERT privilege can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attribute specific.)



CHECK YOUR PROGRESS

Q.2: Choose the correct option:

i) To create a schema or base relation ..... privilege used.

a) Create table b) Drop

c) Select d) Delete

ii) To apply schema changes such as adding or removing attributes from the relation ..... privilege used.

a) Alter b) Modify

c) Delete d) Non of the above

iii) To insert, delete, or update tuples ..... privilege used.

a) Alter b) Modify

c) Delete d) Insert

iv) ..... command is used for the purpose of cancelling privileges

a) Grant b) Revoke

c) Delete d) Non of the above

15.5 MANDATORY ACCESS CONTROL

It is a security policy that classifies data and the users based on security class. Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is higher level and U is lower level. Other security classifications exist, in which the security classes are organized in a lattice. The commonly used model for multilevel security, known as the Bell-LaPadula model, classifies each subject (user, account, programs) and object (relation, tuple, column, view, operation) in to one of the security classifications TS, S, C, or U. We will refer to the clearance (classification) of a subject S as **class(S)** and to the classification of an object O as **class (O)**. Two restrictions are enforced on data access based on the subject/ object classifications:

1) A subject S is not allowed read access to an object O unless class(S) e" class(O). This is known as the simple security property.
2) A subject S is not allowed to write an object O unless class(S) d" class(O). This is known as star property (or *-property).

Mandatory policy ensures a high degree of protection- in a way, it prevents any illegal flow of information. They are therefore suitable for military types of applications, which require a high degree of protection. However, mandatory policies have the drawback of being too rigid in that they require a strict classification of subjects and objects into security levels and, therefore, they are applicable to very few environments.

To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects.

Hence, each attribute A is associated with a **classification attribute** C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole. Hence, a **multilevel relation** schema R with n attributes would be represented as:

$$R(A_1,C_1,A_2,C_2, \ldots, A_n,C_n,TC)$$

Where, each C_i represents the classification attribute associated with attribute A_i.

The value of the TC attribute in each tuple t –which is the *highest* of all attribute classification values within t –provides a general classification for the tuple itself, whereas each C_i provides a finer security classification for each attribute value within the tuple.

The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation.

A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering**.

In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the *apparent key*. This leads to the concept of **polyinstantiation** where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the *same* security classification within each individual tuple.

In addition, all other attribute values in the tuple must have a security classification greater than or equal to that of the apparent key. This constraint ensures that a user can see the key if the user is permitted to see any part of the tuple at all.

Other integrity rules, called **null integrity** and **interinstance integrity**, informally ensure that if a tuple value at some security level can be filtered

(derived) from a higher-classified tuple, then it is sufficient to store the higher-classified tuple in the multilevel relation.

15.6 ROLE BASED ACCESS CONTROL

Role-based access control (RBAC) emerged rapidly in 1990s as a proven technology for managing and enforcing security in large scale enterprisewide systems. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands. The GRANT and REVOKE commands can be used to assigned and revoke privileges from role.

RBAC ensures that only authorize user can access to the certain data or resources. Users create session during which they may active a subset of roles to which they belong. Each session can be assigned to many roles, but it maps to only one user or a single subject.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.

Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as time and duration of role activations, and timed triggering of a role by an activation of another role.

Using an RBAC model is highly desirable goal for addressing the key security requirements of Web-based applications.

15.7 ENCRYPTION AND PUBLIC KEY INFRASTRUCTURES

Encryption is a means of maintaining secure data in an insecure environment. Encryption consists of applying an **encryption algorithm** to data using some prespecified **encryption key**. The resulting data has to be **decrypted** using a **decryption key** to recover the original data.

In 1976 Diffie and Hellman proposed a new kind of cryptosystem, which they called **public key encryption**. Public key algorithms are based on mathematical functions rather than operations on bit patterns. They also

involve the use of two separate keys, in contrast to conventional encryption, which uses only one key. The use of two keys can have profound consequences in the areas of confidentiality, key distribution, and authentication. The two keys used for public key encryption are referred to as the **public key** and the **private key**. The private key is kept secret, but it is referred to as *private key* rather than a *secret key* (the key used in conventional encryption) to avoid confusion with conventional encryption.

A public key encryption scheme, or infrastructure, has six ingredients:

1) *Plaintext:* This is the data or readable message that is fed into the algorithm as input.
2) *Encryption algorithm:* The encryption algorithm performs various transformations on the plaintext.
3) and 4) *Public and private keys:* These are pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exect transformations performed by the encryption algorithm depend on the public or private key that is provided as input.
5) *Ciphertext:* This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
6) *Decryption algorithm:* This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Public key is made for public and private key is known only by owner.

A general-purpose public key cryptographic algorithm relies on one key for encryption and a different but related one for decryption. The essential steps are as follows:

1) Each user generates a pair of keys to be used for the encryption and decryption of messages.
2) Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.
3) If a sender wishes to send a private message to a receiver, the sender encrypts the message using the receiver's public key.

4) When the receiver receives the message, he or she decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver knows his or her private key.

The RSA Public Key Encryption Algorithm, one of the first public key schemes was introduced in 1978 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and is named after them as the RSA scheme. The RSA encryption algorithm incorporates results from number theory, combined with the difficulty of determining the prime factors of a target. The RSA algorithm also operates with modular arithmetic– mod n.

Two keys, *d* and *e*, are used for decryption and encryption. An important property is that d and e can be interchanged. *n* is chosen as a large integer that is a product of two large distinct prime numbers, *a* and *b*. The encryption key *e* is a randomly chosen number between *1* and *n* that is relatively prime to *(a–1) x (b–1)*. The plaintext block P is encrypted as P^e *mod n.* Because the exponentiation is performed *mod n,* factoring P^e to uncover the encrypted plaintext is difficult. However, the decryption key *d* is carefully chosen so that $(P^e)^d$ *mod n* = *P*. The decryption key *d* can be computed from the condition that *d x e= 1 mod ((a–1)x(b–1)).*Thus, the legitimate receiver who knows *d* simply computes $(P^e)^d$ *mod n* = *P* and recovers *P* without having to factor P^e.



CHECK YOUR PROGRESS

Q.3: Choose the correct option:

i) Typical security classes are top secret (TS), secret (S), confidential (C), and .....

a) Simple b) Unclassified

c) Classified d) None of the above

ii) A subject S is not allowed to write an object O unless class(S) d" class(O). This is known as:

a) Star property b) Close property

c) Simple Access d) None of the above

iii) RBAC stand for–

a) Role-based access control

b) Role-based action control

c) Role-based access concurrency

d) None of the above

iv) ..... is a means of maintaining secure data in an insecure environment.

a) Encryption b) Decryption

c) Encoding d) Decoding



15.8 LET US SUM UP

- Database security issues are legal and ethical issue, policy issue, system related issues.
- Threat to database result in the loss or degradation of integrity, availability, and confidentiality.
- The database administrator (DBA) is the central authority for managing a database system.
- If any tampering with the database is suspected, a **database audit** is performed.
- A database log that is used mainly for security purposes is sometimes called an **audit trail**.
- The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.
- In SQL REVOKE command is included for the purpose of cancelling privileges.
- Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U).
- A subject S is not allowed read access to an object O unless class(S) e" class(O). This is known as the simple security property.
- A subject S is not allowed to write an object O unless class(S) d" class(O). This is known as star property (or *-property).

- The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation.
- Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.
- Encryption is a means of maintaining secure data in an insecure environment.
- *Public and private keys*: These are pair of keys that have been selected so that if one is used for encryption, the other is used for decryption.



15.9 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.



15.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (d), ii) (b), iii) (d)

Ans. to Q. No. 2: i) (a), ii) (a), iii) (b), iv) (b)

Ans. to Q. No. 3: i) (b), ii) (a), iii) (a), iv) (a)



15.11 MODEL QUESTIONS

Q.1: What is database security?

Q.2: What are the different database security issues?

Q.3: What do you mean by Loss of Integrity?

Q.4: What is meant by granting a privilege?

Q.5: What is meant by revoking a privilege?

Q.6: List the type of privilege available in SQL.

Q.7: What is the difference between discretionary and mandatory access control?

Q.8: Define the following terms: apparent key, polyinstantiation, filtering.

Q.9: What are the relative merits of using DAC and MAC?

Q.10: What is role-based access control? In what ways is it superior to DAC and MAC?

Q.11: What is the goal of encryption? What process is involved in encrypting data and then recovering it at the other end?

Q.12: What is public key infrastructure? How does it provide security?

Q.13: Give an example of an encryption algorithm and explain how it works.

*** ***** ***



REFERENCES

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) Leon, A. & Leon, M. *Fundamentals of Database Management System*. Tata McGraw Hill.
4) Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems*. McGraw Hill.
5) Silberschatz, A., Korth, H. F. & Sudarshan, S. (1997). *Database System Concepts* (Vol. 4). New York: McGraw-Hill.
6) Singh, S. K. (2011). *Database Systems: Concepts, Design and Applications*. Pearson Education India.
7) Ullman, J. D. (1984). *Principles of Database Systems*. Galgotia Publications.

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.