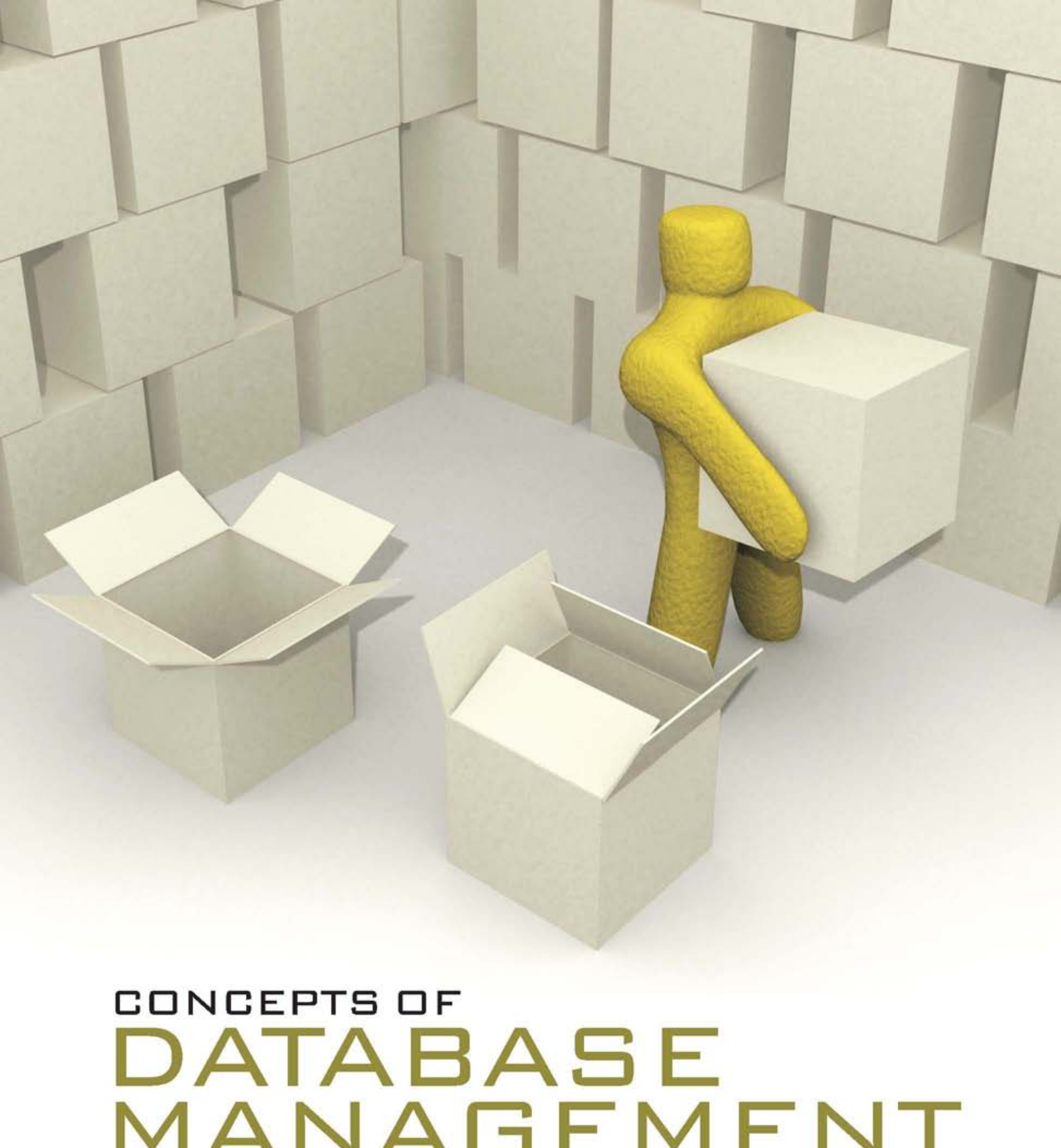


CONCEPTS OF

DATABASE MANAGEMENT SYSTEM

SHEFALI NAIK

ALWAYS LEARNING

PEARSON

Concepts of Database Management System

Shefali Naik

PEARSON

Delhi • Chennai

Dedicated to

My husband Trushit, daughter Jisha, and son Harsheev

ISBN: 9789332526280

e-ISBN: 9789332537231

10 9 8 7 6 5 4 3 2 1

Head Office: 7th Floor, Knowledge Boulevard, A-8(A) Sector 62, Noida 201 309, India.
Registered Office: 11 Community Centre, Panchsheel Park, New Delhi 110 017, India.

Contents

Foreword

Database Management System is one of the most important subjects of the computer and IT field. It is used in almost all the applications like management information systems, expert systems, business information systems, mobile applications, and many more. Over the years, the world has witnessed many inventions in database technologies. The most important invention is relational database management system. Application developers, in the IT industry, are using relational model-based databases for more than thirty years.

Students of IT, computer science and applications, are required to learn databases in one or more courses. Databases are used to store and retrieve data. There are certain rules used to manage data within a database. Database provides many features related to data, such as sharing and integration of data, consistent transaction execution, security and recovery of data through authorization and algorithms. The relational models use a common language, named as Structured Query Language (SQL) to process data. With the rise of the Internet and mobile technologies, databases are also evolving. To store huge amount of data which are spreading worldwide on the Internet and mobile devices, relational database management systems are not enough. Special types of databases, such as NoSQL (Not only SQL) are required for managing such data. Apart from NoSQL databases, the databases which are able to store information related to moving objects, multimedia data, historical data from multiple dimensions, spatial data, etc., are also needed. Automation of processes also require maintenance of the existing applications and analysis of historical data. Analysis of historical data helps in improving business functions by taking important decisions.

In this book, the concepts of databases has been clearly explained giving examples in a lucid language. All chapters are well-organized and comprehensively covering the syllabus of the course on Database Management Systems. At the end of each chapter, summary is given to quickly recap the concepts. The exercises include theory questions, multiple-choice questions, and questions for student's practice. The overview of emerging trends in databases is thoroughly explained. This book addresses the need of B.Tech, M.C.A., and IT programme students, faculty members, and professional developers. I am sure that they will be benefited from this book.

Shefali Naik, the author of this book, is working as senior faculty member, since past thirteen years, at the School of Computer Studies of the Ahmedabad University. She teaches courses on database management systems at graduate and post-graduate levels. To her credit, she has written a good number of articles and technical papers in the area of databases. I wish her good luck for authoring this book and her academic career.

—Bipin V. Mehta
Director
School of Computer Studies,
Ahmedabad University

Preface

This is the first edition of this book. I have tried to cover all the concepts of database management system. This book is useful for the students of computer science, IT, and the courses in which database is offered as an interdisciplinary subject.

The readers who are new to this subject, can start this book reading from the first chapter. Those who are already familiar with databases, can read any chapter to know more about it. Readers, who are willing to learn about any Relational Database Management System, may read Chapters 11 and 12 which gives brief details on MS-Access and Oracle RDBMS, respectively. Readers, who are interested in advancement in database, may read Chapters 8, 9 and 10 which describe advanced topics in database, such as Transactions, Distributed Database, and emerging trends in Database. Those who wish to learn programming language used in database, may read Chapters 6 and 7 in which SQL and PL/SQL is discussed.

The details covered in each chapter of this book are as follows:

- Chapter 1 gives an overview of database by explaining the basic concepts of database, such as data, information; database management system's advantages on other record-keeping system and limitations, its components, etc.
- Chapter 2 describes the evolution of database management system from different systems, such as hierarchical model and network model. It also describes the architecture of DBMS.
- Chapter 3 explains Relational Database Management System.
- Chapter 4 explains Entity-Relationship Model, and Chapter 5 describes Normalization Process.
- Chapters 6 and 7 explains the common languages SQL and PL/SQL, which is used in relational database systems to create and manage database objects; add, remove, change and retrieve data to/from tables and write small programs.
- In Chapter 8, Transaction is discussed; Chapter 9 explains Centralized and Distributed database, and Chapter 10 describes advancement in databases.
- Chapters 11 and 12 cover two well-known relational database management systems MS-Access and Oracle.

Any suggestions to improve the content of the book are welcome.

—Shefali Naik

Acknowledgements

I am indebted to many people who were directly or indirectly involved with the creation of this book.

I would like to thank Bipin V. Mehta, Director at the School of Computer Studies of the Ahmedabad University, for his inspiration and contribution with the Foreword of this book. I am grateful to my colleague and friend, Pratik Thanawala, for his technical suggestions which helped me to improve the contents of this book. I am thankful to my friends from other universities, Sonal Jain, Shivani Trivedi and Tripti Dodiya, for their guidance.

I would like to acknowledge the assistance provided by the editorial team of Pearson Education, Noida; especially, Neha Goomer and Nikhil Rakshit, for their continuous assistance in solving various queries related to the publishing of this book. I am also thankful to Uma Tamang and Naresh Sharma. A big thanks to Pearson Education for publishing this book.

I thank my parents, Girish and Bharati Naik, and children, Jisha and Harsheev along with rest of the family, for their love and patience. Finally, I owe it to my husband Trushit, for his constant support and encouragement.

—Shefali Naik

About the Author

Mrs Shefali Naik, the author of this book, is working as a senior faculty member for past 13 years at School of Computer Studies, Ahmedabad University, Ahmedabad. She teaches subjects related to Databases, Programming, Systems Analysis and Design, and Software Project Management at undergraduate and post-graduate levels. She has obtained her Master's degree in Computer Applications (M.C.A.) and Bachelor's degree in science with mathematics as a special subject (B.Sc., Mathematics) from Ahmedabad, Gujarat.

The author has written few technical papers and articles in the area of databases.

Presently, she is pursuing her Ph.D. from S.P. University, Vallabh Vidyanagar, Anand, Gujarat, in the subject of Distributed Databases.

1 CHAPTER

Basics of Database

CHAPTER OBJECTIVES

- Understanding the meaning of data and information.
- Knowing how database and database management systems are useful in organizations to keep records.
- Examples of database management system.
- Components of database system.
- Characteristics of data and DBMS.
- Differences between file-based management systems.
- Limitations of DBMS.

1.1 | INTRODUCTION

In the current era, people of all ages use database in one way or the other. Everyone uses database in different ways. For example, school children use database of e-mail programs and mobile phones, youngsters use online movie and railway ticket booking database to book tickets, housewives use database of books to order books online or access various community site's database, businessmen use database of airlines to book their trips, academicians use online journals database to do research work and many more. Nowadays, computers are used everywhere. We may reform the proverb 'Where there is a will, there is a way!' as 'Where there is a computer, there is a database.' Computerized Databases have made our life very easy and comfortable. We can search any place, product, area, thing, etc., with the help of stored data in a fraction of a second. Stored data processed with the help of database management systems extracts the desired information, every time. Let us understand the database in some more detail.

1.2 | DATA AND INFORMATION

1.2.1 | Data

Data is a plural of word 'datum'. In our daily life, we use the word data to describe facts about any person, event, place or thing. **Data** are raw facts which may be numbers, values, names, dates, etc. When we combine related data, they describe any real-world entity. **Related data** means data which belong to the same entity (person, place, event or thing). For example, If we consider the entity 'Doctor' (person type of entity), then doctor's name, doctor's address, doctor's birth date, doctor's qualification, doctor's specialization, etc., are data related to doctor. We cannot say that supplier's name and doctor's qualification are related data; because both describe two different entities named supplier and doctor. Thus, when we want to describe any real-world entity, we use data values. Data values alone do not have any meaning because they are not processed yet.

1.2.2 | Information

When we process related data it gives some **information**. Information is useful to take decisions, it can be stored for future use, it has some meaning. To obtain information, we need data. For example, when we process students' attendance data, we can get a list of students with low attendance, students who are attending lectures regularly, students who come to college to attend particular lectures, pattern of class bunking for each student, etc.

On the basis of this information, the college may decide the attendance policy, reschedule the time-table to improve attendance, decide whether to inform parents or not, determine which students should be allowed to sit for an examination, etc. This information could also be stored for future use. In case, when students need a transcript, this information can be used to fill up lecture-wise attendance details of each student or to generate attendance certificates which may be required along with migration certificates when students change universities.

Data can be stored manually or electronically. Similarly, stored data may be processed manually or electronically. Table 1.1 shows some examples of data and information.

We can show the relationship between data and information as given in Figure 1.1.

Figure 1.2 shows an example of data and information.

Table 1.1 shows some examples of data, processes which should be applied on stored data and information which could be obtained after processing certain data.

Table 1.2 shows a student's examination result data which can be processed as per the following condition to obtain grade-wise Result analysis.

Table 1.1 | Examples of Data and Information

Data	Process Description	Information
Census data	Sort records based on area and count total no. of persons gender-wise and age group-wise	Area-wise male and female ratio for different age groups
Board Exam Data	Count subject-wise, no. of students who passed or failed in an exam	Subject-wise total no. of passed or failed students
Climate Data	Maximum temperature and minimum temperature during the year	Hottest and coldest day of the year

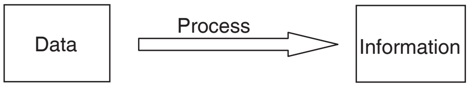


FIGURE 1.1 | Relationship between data and information.

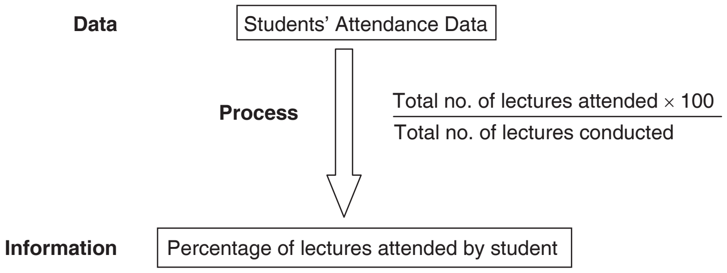

Data
Students' Attendance Data
Process
Total no. of lectures attended × 100
Total no. of lectures conducted
Information
Percentage of lectures attended by student

FIGURE 1.2 | Example of data and information.

If percentage < 40 then, Grade = 'F'
If percentage ≥ 40 and < 50 then, Grade = 'D'
If percentage ≥ 50 and < 60 then, Grade = 'C'
If percentage ≥ 60 and < 70 then, Grade = 'B'
If percentage ≥ 70 then, Grade = 'A'

The following sample information may be obtained after processing the data given in Table 1.2:

Class-wise Result Analysis

Table 1.2 | Students' Examination Result Data

Std No.	Class Code	Std Name	Percentage	Gender
1	FY	Mitali Gupta	89	Female
2	FY	Nirav Valera	91	Male
3	FY	Jainam Vora	79	Male
4	FY	Rajani Vyas	57	Female
5	FY	Nidhi Jain	64	Female
1	SY	Kartik Bhatt	82	Male
2	SY	Kanika Yadav	84	Female
3	SY	Karishma Yadav	70	Female
4	SY	Siddharth Soni	39	Male
5	SY	Akash Patel	69	Male
1	TY	Paras Sanghvi	84	Male
2	TY	Pankti Bindal	94	Female
3	TY	Richa Singh	75	Female
4	TY	Neel Shah	59	Male
5	TY	Payal Shah	60	Female

Class code: FY

No. of students who got ‘A’ Grade: 3
No. of students who got ‘B’ Grade: 1
No. of students who got ‘C’ Grade: 1
No. of students who got ‘D’ Grade: 0
No. of students who got ‘F’ Grade: 0

Class code: SY

No. of students who got ‘A’ Grade: 3
No. of students who got ‘B’ Grade: 1
No. of students who got ‘C’ Grade: 0
No. of students who got ‘D’ Grade: 0
No. of students who got ‘F’ Grade: 1

Class code: TY

No. of students who got ‘A’ Grade: 3
No. of students who got ‘B’ Grade: 1
No. of students who got ‘C’ Grade: 1
No. of students who got ‘D’ Grade: 0
No. of students who got ‘F’ Grade: 0

Overall total no. of students who passed in the exam:14
Overall total no. of students who failed in the exam:1

The above information may be stored and processed further to represent the result analysis graphically or pictorially using bar charts as represented in Figure 1.3. X-axis will contains class code and grades, and Y-axis contains total number of students.

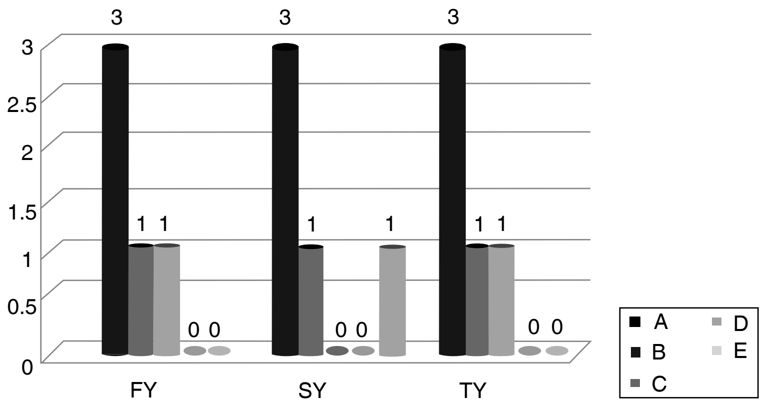


FIGURE 1.3 | Bar chart represents class-wise grade-wise total number of students.

1.3 | DATABASE

As the name suggests, database is a collection of **data**, *i.e.*, database is a storage area where we can store all related data and process them. To understand the concept of database, let us take some real-time examples of database (storage). One logical database which we carry with us all the time is our brain. The brain stores all thoughts, ideas and things which we learn, view, etc. and it relates them. We can retrieve, change or remove these stored ideas and thoughts any time. The example of real-time physical database is a grain warehouse. When it is the season for some grain/pulses, we store them and use them later as per the process requirements. When we process the grains/pulses we obtain the information in the form of floor, sprouts, etc., which could be used in further processing to cook food. The pulses/grains which we find useless could be removed from the warehouse and could be replaced (updated) with fresh stock. In real-life, we use the concepts of data, information and database everywhere.

Figure 1.4 shows an example of real-life database of children's' schoolbag. It is a stationery database which contains entities such as notebook, textbook, compass box, geometry case, etc. Entity Notebook has distinguished notebooks of various subjects; Entity Textbook has distinguished textbooks of various subjects; Entity Compass box has pencils, erasers, sharpeners, ruler, etc., and Entity Geometry box has common mathematical tools.

A **database** is like an electronic storage, which contains computerized data files (entities). It can contain one or many data files. Data files contain various related data within it. Database should contain accurate, consistent and non-redundant data which could be shared by different application programs. Data can be related by defining relationships between proper data. Also, conditions **(constraints)** may be applied on data. Different users may access different data sets from the same database by writing **application program**. We may put security and authentication procedures to provide authorised access of data. There may be more than one database within a database management system. All related **entities** are kept together in the same database. Data within database can be retrieved, updated or deleted directly by database administrator or by authorized users or application programs written by users. To describe data, other details are stored along with the data such as data type, size, constraints, description, format, etc. Using this information, the database management software generates data dictionary which contains 'data about data' or **'metadata'**.

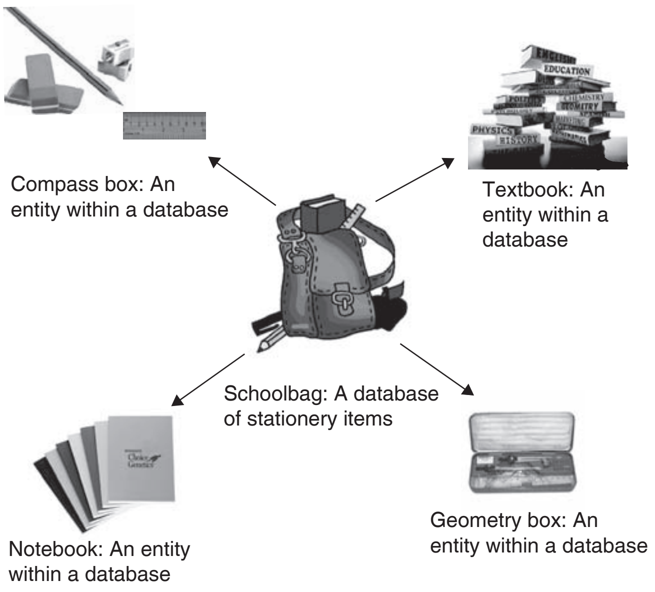

Compass box: An entity within a database
Textbook: An entity within a database
Schoolbag: A database of stationery items
Notebook: An entity within a database
Geometry box: An entity within a database

FIGURE 1.4 | Real-life example of a database.

Table 1.3 | List of Some Organizations and Related Operational Data

Organization	Operational Data
Public Library	Member data, Books data, Publisher data, etc.
Restaurant	Customer data, Employee data, Food Items data, etc.
Super Mall	Product data, Customer data, Supplier data, etc.
University	Student data, Faculty data, Exam data, etc.
Hospital	Patient data, Doctor data, etc.

Database contains data stored in computer. To process the stored data, we need application programs. The processed data could be again stored into database for future use. The data, on which we can do some processing, is known as **operational data**. Any organization contains operational data. Table 1.3 contains some examples of organizations and operational data of a particular organization.

A database stores data of various entities. These entities can be related using relationships. Data also contains description, which is known as metadata. Along with the data, one can keep constraints on its data types.

A cylindrical shape, as shown in Figure 1.5, is used to represent physical database. **Physical database** is useful for the computer (*i.e.*, how a machine sees data), while **logical database** is useful for the user (*i.e.*, how a human being sees data). It is a database of a university, which contains various related entities, such as course, college, student, class, attendance, exam, etc. There are many colleges in a university; each college contains many students in different courses and classes. Students attend lectures, appear in exams and get results.

The 'University' database contains interrelated data which could be shared by different application programs to obtain meaningful information.

1.3.1 | Components of Database System

Figure 1.6 shows components of any conventional database system.

1. **User: User** is any person who uses a database or any other object of the database. User may be of different types and at different levels in an organization. Say for example, the 'University' database may be useful for different persons who are directly or indirectly associated with the university. Following are some categories of users who may use database.

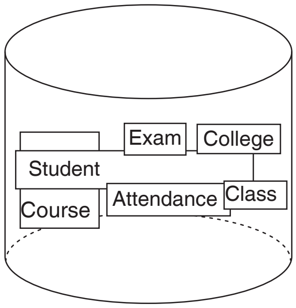


FIGURE 1.5 | Example of 'University' database.

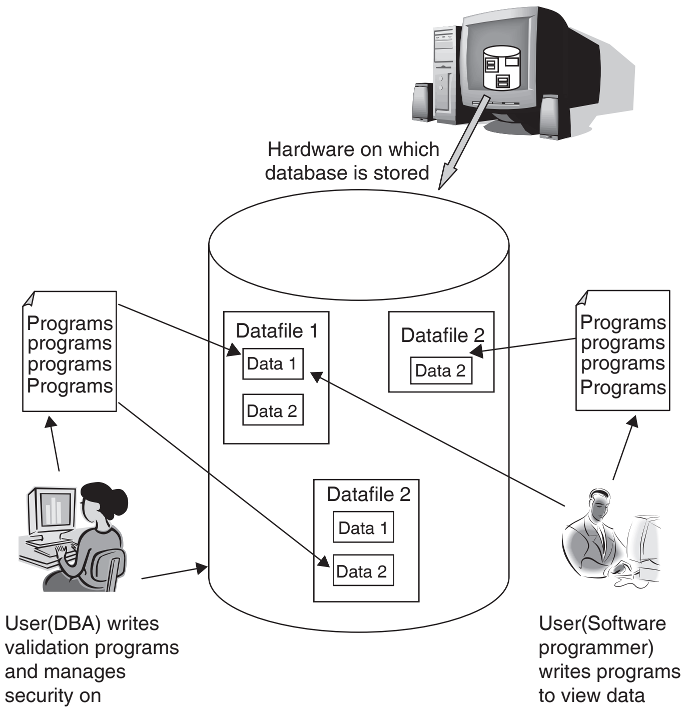

Hardware on which database is stored
Programs programs programs Programs
Datafile 1
Data 1
Data 2
Datafile 2
Data 2
Programs programs programs Programs
Datafile 2
Data 1
Data 2
User(DBA) writes validation programs and manages security on
User(Software programmer) writes programs to view data

FIGURE 1.6 | Components of database system.

a. *Naive User*, *or End-user*, *or Layman*: The clerk of the university uses the 'university' database to enter the data of applicants who have applied for various courses and the same data are retrieved to generate a merit list. The clerk does not know anything about the technical features of the database or the language, using which data is entered or retrieved. He is completely unaware about the technology. Therefore, he/she is known as an **end-user** or Layman or Naive user. Table 1.4 shows some examples of databases and end-users of that database.
b. *Software Programmer*, *or Application Programmer*, *or Application Developer*: A software programmer is a person who writes application programs or logic in some specific language to insert, delete, update or fetch data to/from database. An application programmer has brief knowledge about database and Query Language which is used for writing programs. **Query Language** is a generalized language which is available with all databases. A programmer may or may not have deep understanding about database concepts, but he/she is able to operate on data stored in the database.

Table 1.4 | Examples of End-users

Database	End-user
Online University Database	Applicants, Parents, University Staff, etc.
Hotel database	Customer, hotel Employees, etc.
Online Railway Reservation Database	Citizens of the country, Agents, Railway officials, etc.

c. *Database Designer*: A database designer decides about entities (data files) which should be stored within database, constraints to be applied on data, data types, format and other specifications regarding data. The database designer is responsible for designing of data files.

d. *Database Administrator*: A database administrator (DBA) is the person who is the overall in-charge of a database. He/she assigns authorization to users, writes validation procedures, decides backup and recovery policies, and manages users and privileges. In short, DBA keeps control on database.

2. **Hardware: Hardware** is a permanent storage where the database is stored. It may be a hard-disc, or any other secondary memory. One single database may be stored on more than one storage devices depending on the volume of data stored within the database. For security purpose, a copy of database could be kept on some other storage device. Besides storage device, other hardware, such as computer, peripherals, etc., are also required to perform database-oriented operations.
3. **Software (data dictionary management, database schema management, SQL): Software** are programs or applications which are used to access data from database. These applications reside in DBMS or there may be some applications which could be interfaced with DBMS to manage data. For example, programming languages are used to display data on monitor. There are some software programs, which are part of DBMS, that manage data dictionary or metadata, define schema for the database objects, and are used to write query on database. The common language available with all the databases is known as **Structured Query Language;** if which is popularly known as SQL and sometimes pronounced as 'Sequel'.
4. **Data:** Data is the most important component of a database system. Data is discussed in detail in Section 1.1. When data is stored in database, it should be stored along with its definition, data type and size, constraints, such as duplicate values are allowed or not, possible range of values, formula if it is derived from some other data, etc., display format, format in which it should be entered, validation rules, etc. Some examples of data files/entities (tables) and data stored within the entity are given in Tables 1.5, 1.6 and 1.7. These data files are inter-related data files which are part of the playschool's database.

Table 1.5 | Example of Data within Data File 'Kindergarten'

Data Name	Data Type (Size)	Constraint	Input Format	Display Format
		Data File Name: Kindergarten		
KG id	Integer	Unique number which Should be generated automatically.	—	—
KG name	Character(30)	Must be entered.	Should be entered in upper case.	Should be displayed in title case.
Address	Character(100)	—	—	—
No. of branches	Integer	≥0	—	—
Contact no.	Integer	—	—	—
Contact person	Character (20)	—	—	—

Table 1.6 | Example of Data within Data File 'Class'

Data Name	Data Type (Size)	Constraint	Input Format	Display Format
		Data File Name: Class		
Class code	Character (3)	Must be entered	Should be entered in upper case.	Should be displayed in upper case.
Class desc.	Character (30)	—	—	—
Class capacity	Integer	>0 and ≤30	—	—
No. of divisions	Integer	>0 and ≤4	—	—
Age criteria	Float	≥2	—	—

Table 1.7 | Example of Data within Data File 'Class'

Data Name	Data Type (Size)	Constraint	Input Format	Display Format
		Data File Name: Kindergarten Detail		
Class code	Character(3)	Must be entered	Should be entered in upper case.	Should be displayed in upper case.
KG id	Integer	Must be entered	—	—
Division	Character(1)	Upper case	—	—
No. of students	Integer	>0 and ≤30	—	—

Table 1.8 | Example of Data Values within Data File 'Kindergarten'

KG ID	KG Name	Address	No. of Branches	Contact No.	Contact Person
		Data File Name: Kindergarten			
1	Innocent Flower	Naranpura, Ahmedabad	1	27417411	Mr S. T. Pandya
2	Smart Kids	Navrangpura, Ahmedabad	3	27477471	Ms K. P. Verma
3	Kids Zone	Satellite, Ahmedabad	4	26306301	Mr A. R. Nair
4	Teacher's Pet	Naranpura, Ahmedabad	2	27567561	Mr T. R. Khanna
5	Little Star	Ambawadi, Ahmedabad	1	26466461	Ms N. J. Gupta

When data are entered into tables, Kindergarten, Class and Kindergarten Details (Tables 1.5, 1.6 and 1.7 respectively); the correctness of data are checked. Invalid data cannot be entered into data files.

Tables 1.8, 1.9 and 1.10 contain some valid data values for the tables Kindergarten, Class and Kindergarten Details, respectively.

Table 1.9 | Example of Data Values within Data File 'Class'

Class Code	KG ID	Division	No. of Students	Class Code	KG ID	Division	No. of Students
Data File Name: Kindergarten Detail				**Data File Name: Kindergarten Detail**			
PG	1	1	15	JRKG	2	1	30
PG	1	2	13	JRKG	2	2	30
NUR	1	1	25	JRKG	2	3	30
NUR	1	2	25	JRKG	2	4	30
NUR	1	3	25	SRKG	2	1	30
NUR	1	4	25	SRKG	2	2	30
JRKG	1	1	30	PG	3	1	14
JRKG	1	2	30	PG	3	2	14
JRKG	1	3	30	NUR	3	1	20
JRKG	1	4	30	NUR	3	2	20
SRKG	1	1	30	NUR	3	3	20
SRKG	1	2	30	NUR	3	4	20
PG	2	1	15	JRKG	3	1	30
PG	2	2	10	JRKG	3	2	30
NUR	2	1	25	JRKG	3	3	30
NUR	2	2	25	JRKG	3	4	30
NUR	2	3	25	SRKG	3	1	20
NUR	2	4	25	SRKG	3	2	20

The data in a database must have the following characteristics:

- Same data should be **shared** between different applications. For example, if there are two departments , namely 'accounts' and 'examination', in a university, then data related to student should be shared by these two departments. There should be no need to create a copy of the same data.
- When data are shared, there is a question of integration. Integration means, changes in one data file should also be reflected in the related data file. For example, if a clerk in the accounts department deletes a record of any student, then it should also be deleted from 'member data file' used by the 'library' department of that university.
- When data are properly **integrated**, there are minimum chances of inconsistent data. Data will be consistent if they are integrated properly.
- Data should be **non-redundant**: If possible to avoid duplication of data in different files, data should be stored in one file, and whenever required, it should be referenced from the original file. It is not possible to remove redundancy at all, but we should try to avoid redundancy. Redundant data causes inconsistency within a database. For example, if a student's address is stored in the 'enrolment' file as well as in the 'alumni' file, then 'address' entry for the same student would be redundant. Now, when the student's address is changed, the clerk changes the 'address value' in the 'student' file. He forgets to change address in

Table 1.10 | Example of Data Values within Data File 'Class'

Class Code	Class Describe	Class Capacity	No. of Divisions	Age Criteria
		Data File Name: Class		
PG	Play Group	20	2	2
NUR	Nursery	25	4	2.5
JRK	Junior KG	30	4	3.5
SRK	Senior KG	30	4	4.5

the 'alumni' file. So, now database will show different addresses for the same table which is conflicting. This is called 'data inconsistency', which occurs due to redundant data.

- Data should represent **complete** details. For example, only customer's first name entered in the name field represents incomplete detail. It should contain at least first name of the customer along with the surname.

1.4 | DATABASE MANAGEMENT

The process of managing data within database is called **database management**. To manage database, a database management software/system is required. Database management includes the following activities:

- Writing schema for creating new data files, updating structure of existing data file, deleting a data file.
- Setting relationship among data files.
- Inserting, deleting and updating data values within data files.
- Maintaining data dictionary.
- Creating, updating and deleting database objects other than data files, such as views, synonyms, procedures, functions, triggers, indexes, etc.

1.5 | DATABASE MANAGEMENT SYSTEM

Database management system is a collection of application programs which is used to manage database objects. Database Management System is a generalised software which is used to manage database and database objects, such as tables, users, procedures, functions, etc., and to connect database with any front-end (language) with the help of some hardware. Many types of database management systems are available in the market nowadays. One can purchase license of any database from its vendor and start using it. Also, there are some **open source database management systems** for which there is no license required to use it. It is available on the Internet. One can download it and use it. The source code is also available for free which could be modified by any user and redistributed. MySQL is one of the most popular open source database management system. Table 1.11 contains some examples of database management system and the vendor company who provides it.

Table 1.11 | Examples of DBMS and Its Vendors

Database Management System	Vendor (Supplier)
Oracle	Oracle
SQL Server	Microsoft
Access	Microsoft
DB2	IBM

1.6 | NEED FOR A DATABASE

Following are some reasons for the need of a database:

- Database is required for efficient and easy storage, retrieval, updation and deletion of data records.
- Interrelated data should be grouped in one named storage area for easy access. This storage area may be physical or logical which resides in computer.
- For avoiding unnecessary repetition of data values, checking correctness of data by applying some validation rule, and searching the required information faster thus saving time and effort, etc.
- Database is required for flexibility, *i.e.*, as and when required we can connect the database with different front-ends.
- Once a database is created, it can be shared by many users. Hence, to share data with many applications a database is required.
- Database is needed for storing high volume and complex data, such as documents files, photographs or images, multimedia data, mobile user's data, audio and video files.
- For managing multi-dimensional data.
- Database is required for proper transaction management or transaction handling.

1.7 | FILE-BASED DATA MANAGEMENT SYSTEM

File-based data management system is used by programmers to manage data. Languages, such as C or COBOL contain file management system within it. Figure 1.7 shows a file-based system for any 'Playgroup' in which different data files are used to manage admissions in (a) Nursery, (b) Junior KG and (c) Senior KG—for which different application programs should be written to handle different procedures. In file-based systems, data are managed using data files and these files are created and manipulated by writing application programs. Each application program contains its own data files.

File-based management system has the following disadvantages:

- File-based management system is not appropriate when volume of data is very high. For example, it will be difficult to handle when daily transactions are in thousands or more numbers.
- When number of data files increase, it becomes very complicated to manage data files, *i.e.*, if number of data files increase, number of application programs are also increased; because to insert, update, delete or view data to/from data files, an independent application program is to be written.

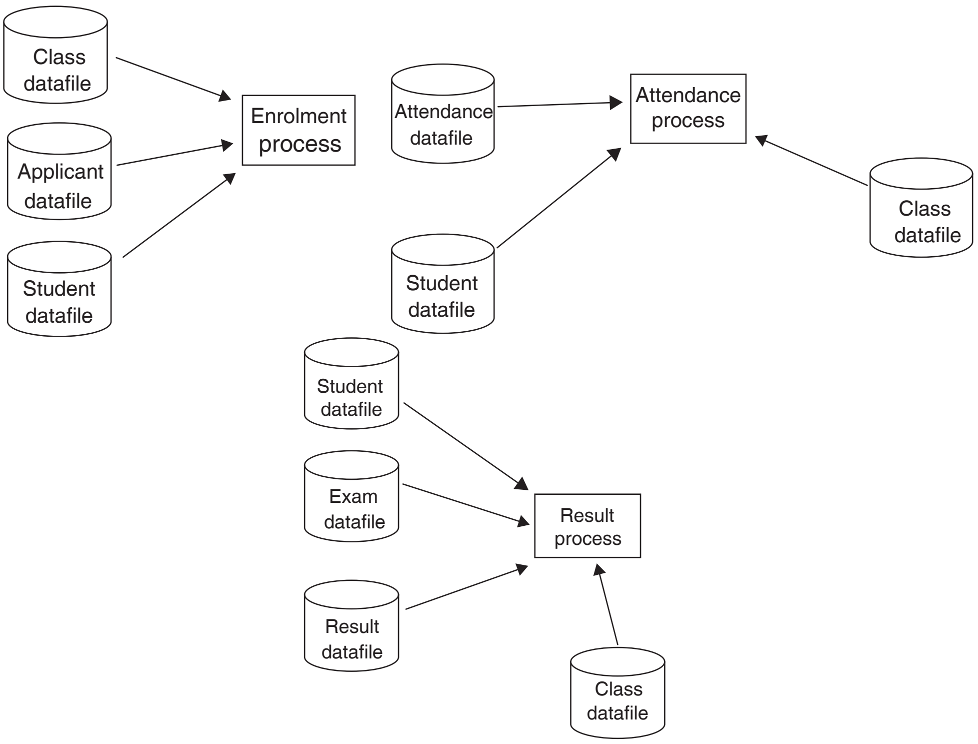


FIGURE 1.7 | File-based management system to manage data of 'Playgroup'.

- Complex data structures, such as pointers, cannot be handled easily by a file-based system.
- When the same data file is required by different programs at the same time, data sharing is not possible. To use same files at the same time, copy of that data file must be created and used. When these are two or more copies of same data file, it may result in inconsistent and redundant data, because changes made in one file may not be carried out in the other files.
- In a file-based system, the programs should only be written in a structured manner.
- It is not possible to set relationships between data files. Programs should be written to relate them.
- Security settings cannot be applied on data files.
- Set of data files created in a specific file-based system cannot be used with other file-based systems as storage formats of different file-based systems vary.

Database system is required to overcome the limitations of file-based management system. The traditional database system contains data files which could be used to store data. The examples of simple database management system are dBASE and FoxPro. These DBMS contains CUI (Character-based User Interface) which provides faster access of data using commands. There is no need to create data files manually. In simple DBMS, data files with data field names and its data type can be created. However, a simple DBMS does not provide the facility to define keys.

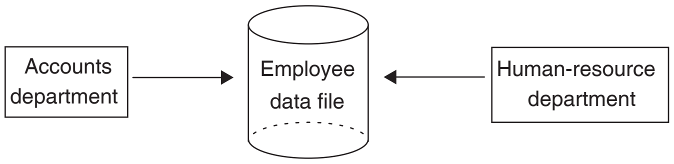


FIGURE 1.8 | Example of data file of DBMS which is shared by various departments in an Organization.

As keys cannot be defined, it is not possible to define relationship between data files either. If user wants to relate data files, then he/she has to write programs to relate two or more file. An example of such a program is given below in Figure 1.8.

But the advantage of simple DBMS, over file-based system, is that we can share data files between applications. Simple commands can be used to search, insert, update, delete and view data.

1.8 | CHARACTERISTICS, OR FEATURES, OR ADVANTAGES OF DATABASE SYSTEMS

- It provides facility to use same data file with different applications, *i.e.*, data can be shared. As shown in Figure 1.8, 'Employee' data file can be used by 'Accounts' department to generate salary slip and by 'Human-Resource' department to evaluate the performance of the employee.
- Duplication of data can be minimized. There is no need to enter same data again and again as data can be shared between different applications.
- Proper transaction management is provided by DBMS. When data are shared between applications, there is a problem of updation when two users try to change same data at the same time. Data can be changed by only one user at a time. DBMS itself decides the priority to allow only one user to change the data at a time. The priority is decided by the DBMS software on the basis of some algorithms. In this way, DBMS handles transactions more efficiently than the file-based management system.
- There is no need to write long programs to manage data. It can be done by writing a simple single line command using structured query language, which is the generalized language provided with DBMS software.
- It is easy to maintain data file structures in DBMS using structured query language.
- Data can be integrated easily, *i.e.*, change in one data is reflected automatically in the related data file's data. For example, if we delete any record from 'Customer' table, the related child records from 'Purchase Order' data file will be deleted.
- Data inconsistency can be avoided. As data are integrated, user is not bothered about updation of same data in different data files. It is handled by the database software. In this way, data will be consistent.
- User management becomes easier. There may be many users of the same database who may access the database from local or remote machines. By providing user rights and authorization checks, the DBMS can control and restrict users.

Table 1.12 | File-based Management System vs. Database Management System

File-based Management System	Database Management System
Needs individual application program to perform any operation on data file.	Any operation on data file is done using single-line commands.
Programming is done using 3GL (Third Generation Languages, such as COBOL, C, PASCAL).	Programming is done using 4GL (Fourth Generation Languages such, as SQL-Structured Query Language).
Transaction management is very difficult.	Transaction management is easy.
Same data file cannot be used simultaneously.	Same data file can be used simultaneously.
Security features cannot be enforced.	Security features can be enforced.
Backup and recovery facility is not available.	Backup and recovery facility is available.
Duplication of data cannot be minimized.	Duplication of data can be minimized.
Examples: C, COBOL, PASCAL languages' file management system.	Example: dBASE, FoxPro, MS Access, Oracle.

- Validation rules can be applied on data before data is entered in the database. It will prevent wrong data inputs.
- Change in data file structure becomes very easy.
- Security can be enforced on data by assigning privileges for different users.
- Appropriate backup procedure is available to avoid loss of data in any adverse circumstances, such as power failure, server failure, hardware crash. In case of failure, the data can be recovered using recovery procedures.
- DBMS provides Import and Export facility using which data files can be imported from one DBMS and exported to another.

Table 1.12 shows the difference between file-based management system and database management system.

1.9 | LIMITATIONS OF DATABASE

Nothing is 100% perfect. Advantages also bring along limitations with them. Database management system also has some limitations. They can be described as:

- Cost of database management system is very high. As the number of users increase, we need to pay more.
- To install database in a network, high-end hardware and skilled personnel to manage the network and database is required.
- As data can be shared through DBMS, it is difficult to control and keep track of data accessed by users. Proper encryption and decryption techniques are required to secure data over a network.
- Efficient employees are required to handle users and decide policies about data access, which requires considerable and constant training.

- If data volume is very high, performance will be poor. Also, when too many users are using database at the same time, it may generate traffic on network and slow down the response time.
- It will be more complex when DBMS contains many databases within it. It may reduce the speed of data access.

SUMMARY

- **Data** means raw facts. It may be any values, such as integer numbers, float numbers, characters, dates, images, Boolean.
- Examples of integer type of data are roll numbers form number, order number; float type of data are salary, balance amount, fees, product price; character type of data are person's name, address, qualification, product name; date type of data are birth date, admission date; retirement date, order date; image type of data are person's photo, image of property location, image of property; Boolean type of data are customer status, payment status, gender.
- Interrelated data represent any entity, *i.e.*, data are characteristics of entity. For example, student name, student birth date and student gender are data (characteristics) related to student entity. An **entity** is a distinguishable object of real-world.
- Data related to an entity are kept together in a data file, *i.e.*, data file is a collection of related data.
- Data may be stored manually or electronically. When we apply any process on stored data, it gives some valuable information. The process on data stored electronically can be applied by writing application programs.
- The data on which we do some operation, is known as operational data. Operational data belongs to any organization. For example, student's data is an operational data for the 'University' organization. By processing student's data, we can generate information like a student's mark sheet, list of college-wise total number of students, etc.
- **Database** is a collection of data files or tables which contain data within it. **Relationship** can be set to access data from different files.
- The process of managing data within database is called **database management**.
- Database system contains the components data, user, hardware and software.
- Using database we can share and integrate data between applications.
- **Database management system** is a collection of software programs through which database can be managed.
- File-based management system requires manual creation of data files which are very difficult to handle. Within file-based management system, independent programs should be written to do operations such as insert, delete, update and view data.
- Database management system provides structured query language to store and access data from database. There is no need to write long programs to access data. Data redundancy and data inconsistency problems can be avoided using database management system.

- Database management system provides automatic transaction management, backup and recovery facility, export and import facility, user management and other functionalities.
- The limitations of database management systems are: they are complex, expensive, requires knowledge to use them, data control is difficult, performance may suffer because of high data volume, etc.

Exercises

1. Define Data and Information. Show relationship between these two.
2. Give any two examples of data. Write any two types of information which could be obtained by processing these data.
3. Define the terms:
   a. Database
   b. Database management
   c. Database management system
   d. Operational data
   e. Metadata
4. For any restaurant system, which data are operational data? Write two examples of information related to that.
5. Draw a diagram of components of database system and explain.
6. List down different types of users of database system with their roles.
7. Name any four DBMS along with their supplier company.
8. What is an open source database? Give an example.
9. Which are the characteristics or features of data in a database?
10. Write a short note on file-based management system.
11. Give an example of file-based management system. Mention the disadvantages of this system.
12. List down and explain advantages of database management system over file-based management system.
13. What are the limitations of database management system?
14. Discuss data redundancy and data inconsistency with relevant example.
15. Write/Tick the correct answer.

    i. Data means:
       a. Unprocessed facts b. Processed facts
       c. Unprocessed information d. Processed information

    ii. The operational data related with 'Hostel' are:
       a. Mess data b. Customer data
       c. Patient data d. Doctor data

    iii. DBMS is an abbreviation of ______________.
       a. Database Management System b. Distributed Management System
       c. Data Management System d. Database Modification System

iv. Database contains data files or tables.

a. True	b. False

v. Data represents ______________ of an entity.

a. Relationship	b. Definition
c. Type	d. Characteristics

vi. DBMS supports structured query language (SQL) which is _________.

a. 1GL	b. 2GL
c. 3GL	d. 4GL

vii. The user who does not know working of a database is called ____________.

a. End-user	b. Database Designer
c. DBA	d. System Analyst

viii. ____________ is responsible for overall control of database.

a. Data Analyst	b. Database Administrator
c. Programmer	d. End-user

ix. Among the following, which one is not a component of database system?

a. Hardware	b. Data
c. Software	d. None

x. Data redundancy causes ________________ data in database.

a. Accurate	b. Complete
c. Meaningful	d. Duplicate

2

CHAPTER

Data Models and Architecture of DBMS

CHAPTER OBJECTIVES

- Evolution of data models.
- Knowing the traditional data models.
- Advantages and disadvantages of various types of data models.
- Three-level architecture of database management system.
- Understanding languages used to define objects, manage and control data and transaction.

2.1 | EVOLUTION OF DATA MODELS

- Data are the primary requirement of any application. It is important to store data appropriately for easy access. During the 1940s and 1950s, use of computer to write applications in programming language for automation increased. The file-based management system was not sufficient to manage data. Hence, evolution of data models took place. **Figure 2.1** shows the block diagram of evolution of data models from manual record keeping system to file-based management system, and from file-based management system to database management system.
- COBOL (Common Business-oriented Language) and FORTRAN (Formula Translation) were two primary programming languages used to create enterprise applications during the 1950s. The file systems of these languages were not able to handle data which are required by the applications developed in these languages.
- Therefore, in the 1960s, IBM and Rockwell International developed a hierarchical database system named IMS (Information Management System). Later, C.W. Bachman proposed Network Data Model and, on the basis of this model, General Electric developed a network database model named IDS (Integrated Data Store). Both IMS and IDS were accessible from the programming languages using an interface. Using these database systems, application development and data management within application had become easy, but a complex task.

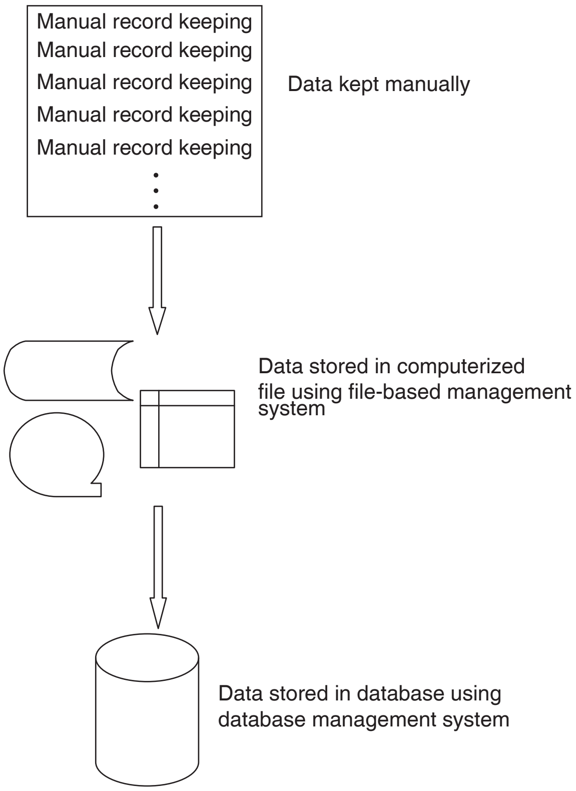

Manual record keeping
Manual record keeping
Manual record keeping
Manual record keeping
Manual record keeping
Data kept manually
Data stored in computerized file using file-based management system
Data stored in database using database management system

FIGURE 2.1 | Evolution from manual record keeping system to file-based management system and, from file-based management system to database management system.

- In 1970, Edgar F. Codd proposed a different data model, in which he had suggested that data in a database could be represented as a two-dimensional table structure, which is known as **relation**, and could be accessed without writing lengthy programs to access data. This model is known as **relational data model**. Nowadays, many vendors provide relational database management systems. Some well-known RDBMS are MS-Access and MS-SQL Server provided by Microsoft; Oracle provided by Oracle; DB2 provided by IBM, and many more.
- Along with RDBMS, the object-oriented concept evolved. The use of object-oriented programming languages increased in the 1980s, and along with it increased the need of a database system which would be able to handle classes and objects. Thus, evolved the object-oriented data model. Many vendors had developed OODBMSs namely Gem-Stone, ObjectDesign, Versant, O2, Objectivity, etc.
- Extensive use of object-oriented languages resulted in an **object-relational DBMS** which is a combination of object-oriented and relational DBMS. Many vendors, such as Oracle, IBM, provided functionalities of object-oriented concepts in their RDBMS (*see* Figure 2.2).

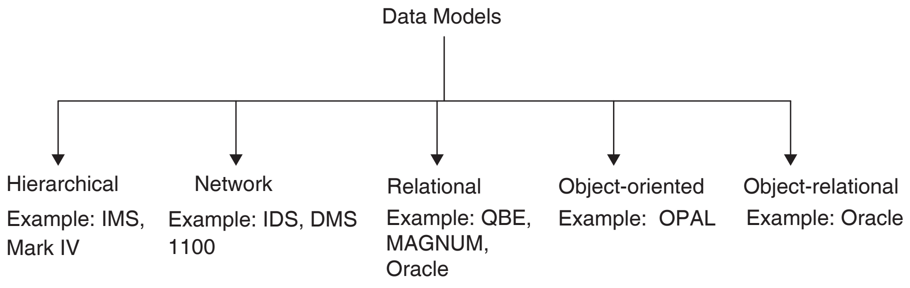


FIGURE 2.2 | Data models.

2.2 | HIERARCHICAL DATA MODEL

- The data model describes data and its definition. In case of an object-oriented data model, it describes the object and its behaviour. A **data model** is a logic which is based on concepts, while its implementation is called, 'database management system', *i.e.*, database management system is a physical implementation of data model. **Entity-relationship model** is a conceptual model which shows entities and relationships between entities.
- The **hierarchical data model** was the very first data model developed in the 1960s. The hierarchical data model named IMS (Information Management System) was developed by IBM and Rockwell Company and widely used during the 1960s and1970s. The entities and relationships between entities were managed with the help of a tree-like structure in the hierarchical model. In this tree, there exists a root and it is related with its child. A **root** is known as a parent. One parent may have many children in hierarchical structure, but one child cannot have more than one parent, *i.e.*, if there is a child entity which is related with more than one parent entities, then two independent parent nodes should be created which contains redundant child records. The redundant child records should be linked with both the parents. On root, there will be entity occurrences from the parent entity. One entity occurrence means one **segment**. If this segment is on the root, it is called **root segment**. The entity occurrence, which falls under the root segment (parent), is known as **dependent segment** (child), *i.e.*, collection of entity occurrences are called, 'segments'. Root segment and dependent segments are connected through link. In a hierarchical structure, one root segment may have many dependent segments, but a dependent segment will have only one root segment. To explain this, many-to-many relationship between root and dependent segments is not possible in a hierarchical structure.
- Entity occurrence from parent entity is shown as a root segment, and its related entity occurrences are shown as its dependent segments. The entity occurrences of same entities are shown at the same level in a tree. The related entity occurrences, which fall under it, are its branch.
- To give an example, consider the entities given in Figures 2.3 and 2.4. Figure 2.3 contains entities Zone, Region, Item and Area; while Figure 2.4 contains entities as Salesman and Sales. All the entities are related with the following relationships with each other.

- Figures 2.3 and 2.4 represents the following entities:
  - Zone
  - Region
  - Area
  - Item
  - Salesman
  - Sales

Area

Area Code	Area Name	Region ID
1	Ludhiana	1
2	Amritsar	1
3	Bilaspur	2
4	Shimla	2
5	Hamirpur	2
11	Calicut	6
12	Cochin	6
13	Munnar	6
14	Patiala	1
31	Anantnag	13
32	Srinagar	13
33	Ahmedabad	3
34	Udhampur	13
44	Surat	3
55	Baroda	3
61	Kolkata	5
62	Darjeeling	5
63	Baranagar	5
71	Patna	10
72	Nalanda	10
73	Vaishali	10
81	Guwahati	11
82	Digboi	11
83	Sibsagar	11
111	Bangalore	7
112	Mysore	7
113	Coorg	7
121	Hyderabad	8
122	Vishakhapatnam	8
123	Vijaywada	8
131	Pune	4
132	Mumbai	4
133	Nashik	4
141	Jaisalmer	9
142	Jodhpur	9
143	Bikaner	9

Region

Region ID	Region Name	Zone ID
1	Punjab	1
2	Himachal Pradesh	1
3	Gujarat	4
4	Maharashtra	4
5	West Bengal	2
6	Kerala	3
7	Karnataka	3
8	Andhra Pradesh	3
9	Rajasthan	4
10	Bihar	2
11	Assam	2
13	Jammu and Kashmir	1

Zone

Zone ID	Zone Name
1	North
2	East
3	South
4	West

Item

Item No	Item Desc.	Price (in ₹)
1	Bulldozer	200000
2	Soil Stabilizer	300000
3	Scraper	350000
4	Excavator	200000
5	Dump Truck	150000

FIGURE 2.3 | Entities Zone, Region, Area, and Item.

Salesman No.	Item No.	Total_Qty_Sold
1	1	2
1	2	1
1	3	2
2	1	2
2	2	2
3	1	2
3	3	2
4	1	4
4	3	5
5	1	4
5	2	3
6	4	2
6	5	3
11	1	2
11	5	7
12	2	3
12	3	4
29	3	2
29	4	4
30	1	4
30	2	3
51	4	3
51	5	2
52	1	10
52	2	3
52	3	1
52	4	7
52	5	3
61	1	1
62	3	2
62	5	2
101	1	2
102	2	3
109	4	3
110	5	2
111	1	3
112	1	3
123	3	2
124	4	1
145	1	1
146	1	2
147	4	3
165	2	3
175	1	3
176	1	5
178	1	2
183	1	1
184	2	2
187	2	2
188	2	2
189	1	1

Salesman		
Salesman ID	Salesman Name	Area Code
1	A. P. Singh	1
2	K. N. Kapoor	1
3	R. K. Chopra	2
4	P. G. Singh	2
5	S. N. Pathan	3
6	R. K. Khan	3
11	S. R. Trivedi	4
12	P. K. Jain	4
21	T. P. Khan	5
22	A. R. Khan	5
29	D. C. Khanna	31
30	P. T. Mehra	31
51	A. K. Garoo	34
52	D. N. Brave	34
61	T. N. Khan	32
62	A. P. Mishra	32
101	P. K. Damani	141
102	A. R. Agrawal	141
109	P. F. Karnik	131
110	A. M. Panzade	131
111	S. R. Sukhadiya	143
112	V. R. Jain	143
123	S. D. Sharma	142
124	K. K. Jain	142
145	S. E. Tendulkar	132
146	V. V. Manjrekar	132
147	P. N. Khedekar	132
165	A. R. Narayan	112
175	R. Benerjee	61
176	S. Tagore	61
178	L. M. Srinivasan	113
183	T. Ray	62
184	M. Ghosh	62
187	F. Srivastava	63
188	V. Jain	71
189	T. Chaterjee	71
190	S. B. Pillai	12
191	A. R. Nair	11
221	K. Yadav	81
222	G. F. Mishra	133
223	J. J. Raina	133
231	T. R. Naik	44
232	S. V. Joshi	44
261	A. F. Ghoshal	13
271	M. N. Shah	33
272	T. N. Sanghvi	33
273	A. A. Pathak	33
281	S. G. Gupta	55
282	K. D. Mistry	55
331	S. Chattopadhyay	82
81	D. Mathur	83
991	S. Mudaliar	111

FIGURE 2.4 | Entities, Salesman, and Sales.

- The entities as shown in Tables 2.3 and 2.4 are related with the following relationships:
  - Each Zone contains many Regions (1 Zone–Many Regions)
  - Each Region contains many Areas (1 Region–Many Areas)
  - Each Area contains many Salesman (1 Area–Many Salesman)
  - Each Salesman sells many Items, and each Item is sold by many Salesman. (1 Salesman–Many Items and Many Salesman–1 Item, *i.e.*, many-to-many relationship between Salesman and Item).
- Figure 2.5 shows the hierarchical model which represents the entities of Figures 2.3 and 2.4.
- Hierarchical data model can represent one-to-many relationships very effectively, but it is not possible to represent many-to-many relationship because a child can have only one parent in hierarchical model.
- To solve this problem, many-to-many relationship should be represented as two independent trees. For example, to represent the relationship, 'Each Salesman sells many Items and each Item is sold by many Salesmen.'; the first tree will have Salesman as parent and Item as child, and the second tree will have Item as parent and Salesman as Child. These two different scenarios are shown in Figures 2.6(a) and 2.6(b).
- The hierarchical data model has the following advantages and disadvantages.

Advantages:

1. It is easy to understand.
2. The one-to-many relationship can be handled quite effectively.

Disadvantages:

1. It is not possible to insert a dependent record without inserting a parent record. For example, as shown in Figure 2.6(b), it is not possible to insert the details of any item until it is been sold by any Salesman. Similarly, as shown in Figure 2.6(a), it is not possible to insert the details of any Salesman until he supplies any item.

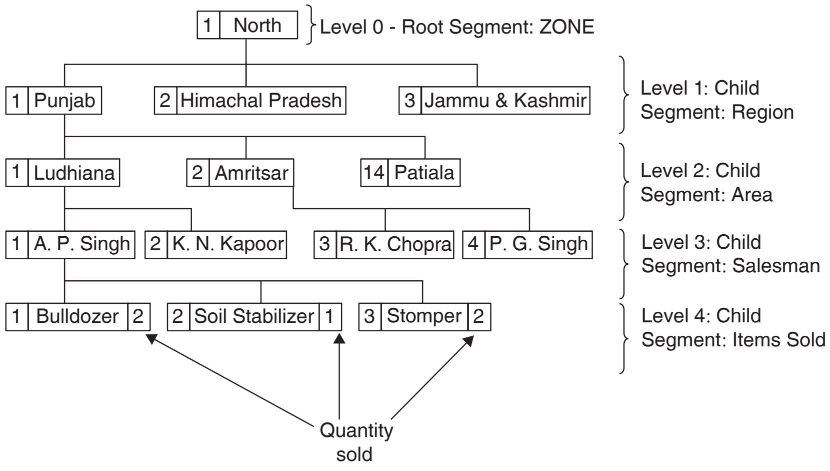
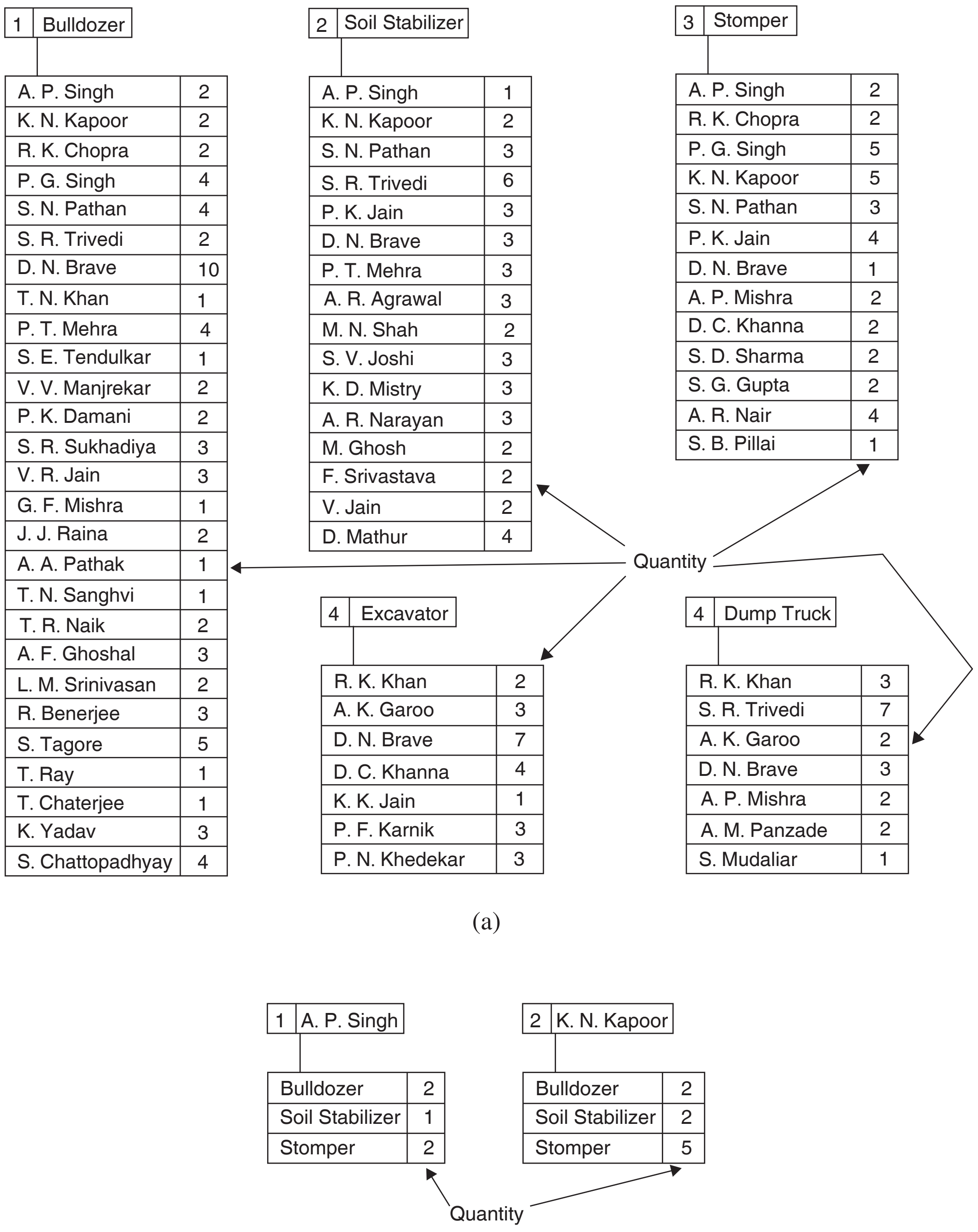


FIGURE 2.5 | The hierarchical model.



1	Bulldozer
A. P. Singh	2
K. N. Kapoor	2
R. K. Chopra	2
P. G. Singh	4
S. N. Pathan	4
S. R. Trivedi	2
D. N. Brave	10
T. N. Khan	1
P. T. Mehra	4
S. E. Tendulkar	1
V. V. Manjrekar	2
P. K. Damani	2
S. R. Sukhadiya	3
V. R. Jain	3
G. F. Mishra	1
J. J. Raina	2
A. A. Pathak	1
T. N. Sanghvi	1
T. R. Naik	2
A. F. Ghoshal	3
L. M. Srinivasan	2
R. Benerjee	3
S. Tagore	5
T. Ray	1
T. Chaterjee	1
K. Yadav	3
S. Chattopadhyay	4

2	Soil Stabilizer
A. P. Singh	1
K. N. Kapoor	2
S. N. Pathan	3
S. R. Trivedi	6
P. K. Jain	3
D. N. Brave	3
P. T. Mehra	3
A. R. Agrawal	3
M. N. Shah	2
S. V. Joshi	3
K. D. Mistry	3
A. R. Narayan	3
M. Ghosh	2
F. Srivastava	2
V. Jain	2
D. Mathur	4

3	Stomper
A. P. Singh	2
R. K. Chopra	2
P. G. Singh	5
K. N. Kapoor	5
S. N. Pathan	3
P. K. Jain	4
D. N. Brave	1
A. P. Mishra	2
D. C. Khanna	2
S. D. Sharma	2
S. G. Gupta	2
A. R. Nair	4
S. B. Pillai	1

4	Excavator
R. K. Khan	2
A. K. Garoo	3
D. N. Brave	7
D. C. Khanna	4
K. K. Jain	1
P. F. Karnik	3
P. N. Khedekar	3

4	Dump Truck
R. K. Khan	3
S. R. Trivedi	7
A. K. Garoo	2
D. N. Brave	3
A. P. Mishra	2
A. M. Panzade	2
S. Mudaliar	1

1	A. P. Singh
Bulldozer	2
Soil Stabilizer	1
Stomper	2

2	K. N. Kapoor
Bulldozer	2
Soil Stabilizer	2
Stomper	5

FIGURE 2.6 | (a) A tree representing item supplied by various salesman; (b) A tree representing salesman supplies various items.

2. If we delete any root segment, then the dependent segments which falls under it, are also deleted. For example, refer to Figure 2.6(a), if we delete root segment of the item Bulldozer, then all the Salesmen, who have supplied Bulldozer, will also be deleted. As a result, the Salesman who has sold only Bulldozer will be permanently deleted from the hierarchy model. His record will be inserted again, only when he will supply some other item.
3. It is difficult to update any Child segment. As the number of segment increases, the tree becomes extremely complex. At that time, it is very cumbersome to search for any segment and update it, *i.e.*, to search the last dependent segment of the last root segment of a tree, one has to traverse all the dependent segments of all the root segments.
4. The hierarchical model can represent only the one-to-many (1: M) relationship. Here, the many-to-many relationship causes redundant data.

2.3 | NETWORK DATA MODEL

- The **Network data model** represents data using link between records. The parent record is called **Owner Record**, and the child record is called **Member Record**. If the Owner and Member records are related with the many-to-many relationship, then they are connected through connector record which is known as Set. The entities, given in Figures 2.4 and 2.5, are represented as a network model as shown in Figure 2.7.
- Figure 2.7 shows part of a network model, where:
  - Zone records are Owner records of Region records and Region records are Member records.
  - Region records are Owner records of Area records, and Area records are Member records of Region.
  - Area records are Owner records of Salesman records, and Salesman records are Member records of Area.
  - Salesman records are Owner records of Item records, and Item records are Member records of Salesman which are connected through the 'Set' Sales. Sales record is a connector record between Salesman and Item.

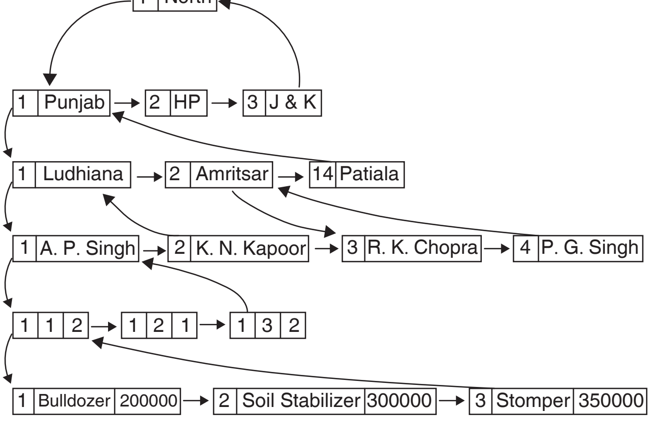


FIGURE 2.7 | The network model.

- The Owner record is linked with the first Member record, the first member record is linked with the second Member record, and the second Member record is linked with the third Member record, and so on up to the last Member record. The last Member record is again linked with the Owner record. Management of the many-to-many relationship in a network model is quite simple.
- Following are the advantages and disadvantages of a network model.

Advantages:

1. The many-to-many relationships can be represented more easily in a network data model than that of a hierarchical data model.
2. The network data model supports Data Definition Language and Data Manipulation Language.
3. To insert data of a new Item, say item no. 6, we would need to create a new Item record. There will be no connector record for the new Item until it is sold by any Salesman. Item no. ‘6’ will contain a single link from Item no. ‘6’ to Item no. ‘6’ itself, initially.

Disadvantages:

1. Searching is more complicated than hierarchical model in network model because of its complex data structure.
2. The DML is also very complex as there are many constructs, such as records and links.

2.4 | RELATIONAL DATA MODEL

The concept of relational model was given by E. F. Codd, in 1970, in his landmark paper on relational data model. In the relational model, data are represented in a tabular form which is called, relation (table), and they are associated with relationships. Therefore, the name of this model is relational data model. Each entity is converted into relation and association is handled through primary and foreign keys. The detailed explanation of relational model is given in Chapter 3. Each **entity occurrence** is known as **tuple** (record) and **characteristic** of an entity is called an **attribute** (column). It is very easy to represent many-to-many relationship using relational data model. The relational model is widely used worldwide, nowadays, to store data. Figures 2.8 and 2.9 show the relational model of data as shown in Figures 2.3 and 2.4. All the relations are associated, with each other as listed here:

- Relation Zone is related with Region through ‘zone id’.
- Relation Region is related with Area through ‘region id’.
- Relation Area is related with Salesman through ‘area code’.
- Relation Salesman is related with Sales through ‘salesman id’.
- Relation Item is related with Sales through ‘item id’.

For relations:

- Zone—‘zone id’ is a primary key which is referred in Region relation.
- Region—‘region id’ is a primary key which is referred in Area relation, and ‘zone id’ is referenced from Zone relation in Region relation.
- Area—‘area code’ is a primary key which is referred in Salesman relation and ‘region id’ is referenced from Region relation in Area relation.

Area		
Area Code	**Area Name**	**Region ID**
1	Ludhiana	1
2	Amritsar	1
3	Bilaspur	2
4	Shimla	2
5	Hamirpur	2
11	Calicut	6
12	Cochin	6
13	Munnar	6
14	Patiala	1
31	Anantnag	13
32	Srinagar	13
33	Ahmedabad	3
34	Udhampur	13
44	Surat	3
55	Baroda	3
61	Kolkata	5
62	Darjiling	5
63	Baranagar	5
71	Patna	10
72	Nalanda	10
73	Vaishali	10
81	Guwahati	11
82	Digboi	11
83	Sibsagar	11
111	Bangalore	7
112	Mysore	7
113	Coorg	7
121	Hyderabad	8
122	Vishakhapatnam	8
123	Vijaywada	8
131	Pune	4
132	Mumbai	4
133	Nashik	4
141	Jaisalmer	9
142	Jodhpur	9
143	Bikaner	9

Region		
Region ID	**Region Name**	**Zone ID**
1	Punjab	1
2	Himachal Pradesh	1
3	Gujarat	4
4	Maharashtra	4
5	West Bengal	2
6	Kerala	3
7	Karnataka	3
8	Andhra Pradesh	3
9	Rajasthan	4
10	Bihar	2
11	Assam	2
13	Jammu and Kashmir	1

Zone	
Zone ID	**Zone Name**
1	North
2	East
3	South
4	West

Item		
Item No.	**Item Desc.**	**Price (in ₹)**
1	Bulldozer	200000
2	Soil Stabilizer	300000
3	Scraper	350000
4	Excavator	200000
5	Dump Truck	150000

FIGURE 2.8 | Relations Zone, Region, Area and Item.

- Salesman—'salesman id' is a primary key which is referred in Sales relation and 'area code' is referenced from Area relation in Salesman relation.
- Sales—Combination of 'salesman id' and 'item id' is a primary key. 'Salesman id' is referenced from Salesman and 'item id' is referenced from Item relation in Sales relation.

Sales		
Salesman ID	**Item ID**	**Total_qty_Sold**
1	1	2
1	2	1
1	3	2
2	1	2
2	2	2
3	1	2
3	3	2
4	1	4
4	3	5
5	1	4
5	2	3
6	4	2
6	5	3
11	1	2
11	5	7
12	2	3
12	3	4
29	3	2
29	4	4
30	1	4
30	2	3
51	4	3
51	5	2
52	1	10
52	2	3
52	3	1
52	4	7
52	5	3
61	1	1
62	3	2
62	5	2
101	1	2
102	2	3
109	4	3
110	5	2
111	1	3
112	1	3
123	3	2
124	4	1
145	1	1
146	1	2
147	4	3
165	2	3
175	1	3
176	1	5
178	1	2
183	1	1
184	2	2
187	2	2
188	2	2
189	1	1

Salesman		
Salesman ID	**Salesman Name**	**Area Code**
1	A. P. Singh	1
2	K. N. Kapoor	1
3	R. K. Chopra	2
4	P. G. Singh	2
5	S. N. Pathan	3
6	R. K. Khan	3
11	S. R. Trivedi	4
12	P. K. Jain	4
21	T. P. Khan	5
22	A. R. Khan	5
29	D. C. Khanna	31
30	P. T. Mehra	31
51	A. K. Garoo	34
52	D. N. Brave	34
61	T. N. Khan	32
62	A. P. Mishra	32
101	P. K. Damani	141
102	A. R. Agrawal	141
109	P. F. Karnik	131
110	A. M. Panzade	131
111	S. R. Sukhadiya	143
112	V. R. Jain	143
123	S. D. Sharma	142
124	K. K. Jain	142
145	S. E. Tendulkar	132
146	V. V. Manjrekar	132
147	P. N. Khedekar	132
165	A. R. Narayan	112
175	R. Benerjee	61
176	S. Tagore	61
178	L. M Srinivasan	113
183	T. Ray	62
184	M. Ghosh	62
187	F. Srivastava	63
188	V. Jain	71
189	T. Chaterjee	71
190	S. B. Pillai	12
191	A. R. Nair	11
221	K. Yadav	81
222	G. F. Mishra	133
223	J. J. Raina	133
231	T. R. Naik	44
232	S. V. Joshi	44
261	A. F. Ghoshal	13
271	M. N. Shah	33
272	T. N. Sanghvi	33
273	A. A. Pathak	33
281	S. G. Gupta	55
282	K. D. Mistry	55
331	S. Chattopadhyay	82
81	D. Mathur	83
991	S. Mudaliar	111

FIGURE 2.9 | Relations salesman and sales.

- Item—'item id' is a primary key which is referred in Sales relation.
- The advantages and disadvantages of a relational model are as follows:

Advantages:

1. Relational model is easy to understand.
2. Data can be managed properly in it.
3. It provides structured query language to manage data, which is very easy to learn. DDL and DML are simpler in respect to the other models.
4. Transactions can be managed properly.
5. Many-to-many relationships can be represented through primary and foreign key and without any complexity.
6. Insert, delete, and update operations can be performed without any loss of data.
7. Data dictionary management is provided.

Disadvantage:

1. It is difficult to handle due to complex data types.

2.5 | OBJECT-ORIENTED DATA MODEL

- In object-oriented data models entity is represented as a class. A **class** within it contains **data** and **methods**. Data are attributes of object and methods are behaviour. For example, class Zone contains the attribute 'zone id' and 'zone name'. It contains methods to manage these attribute values. However, it is not required to define both attributes and methods in the same class. Methods can be defined separately.
- Each record of a zone is known as an **object**, which is a class member. For example, with respect to Zone class, there are four objects. Each object has different values for attributes 'zone id' and 'zone name'. But these four objects will share the same methods. **Methods** are procedures which are the programs to manage attribute values. For zone class, methods are—add zone details, change zone details, remove zone details and search zone details. Methods are invoked by **messages**. There are many built-in classes and methods are available within object-oriented system. For example, class Integer and methods available for this class are <, >, ≥, ≤, =, <>, etc.
- Figure 2.10 shows the class ZONE and its four objects. Each object has unique identification number which is known as **object identifier** (OID). OID is not visible to the users because they are addresses. It is similar to the pointer. The comparison between object-oriented terminology and traditional terminologies is given in Table 2.1

Table 2.1 | Comparison between Object-Oriented and Traditional Terminologies

Object-oriented Terminology	**Traditional Terminology**
Class	Type (it may be built-in, or user-defined)
Object (Class Instance) 1. Immutable object 2. Mutable object (which holds an object ID)	1. Value (field value) 2. Variable (field name)
Method	Operator
Message	Operator invocation

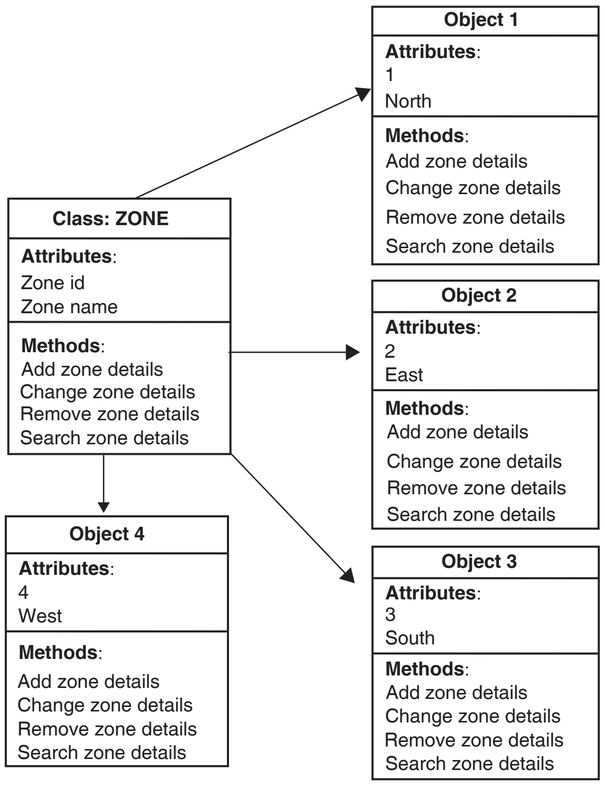


FIGURE 2.10 | Class and Objects.

Table 2.2 | Example of Object-oriented Terminology

Object-oriented Terminology	Example
Class	Integer, Character (Built-in classes) Zone, Region (User-defined classes)
Object	980, 'Ahmedabad' (immutable object = value) Zone id, zone name (mutable object = variable)
Method	Add_zone
Message	Add_zone (z1)

- To understand the object-oriented terminology, an example is given in Table 2.2.
- Object-oriented supports abstract data types, *i.e.*, one class may contain another class as its attribute. For example, Zone contains Regions within it. This approach is 'containership' approach, which is shown in Figure 2.11. Actually, Zone object does not contain the Region object, but Region object is referenced by using its OID. Object-oriented data model is ideal for complex data types, such as video, audio, image.

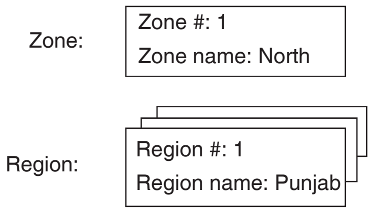


FIGURE 2.11 | Example of containment hierarchy—zone contains different regions.

- Object-oriented databases support all the features of object-oriented methodology, such as message passing (methods can pass messages to other objects), class inheritance, method overriding, encapsulation, polymorphism, and operator overloading.
- Following are the advantage and disadvantages of an object-oriented data model:

Advantage:

1. It is easy to handle complex data types.

Disadvantages:

1. It is difficult to understand and use compared to relational model.
2. It does not have ad hoc query capability.
3. Integrity issues are involved. On updation or deletion of parent object, child object is not updated or deleted automatically. Procedural code should be written for that.
4. Object-oriented systems are procedural, which are 3GL languages. It does not support SQL which is 4GL. Therefore, it is a step back from 4GL to 3GL.
5. Data dictionary is not managed automatically. Staff is required to do so.

2.6 | OBJECT-RELATIONAL DATA MODEL

- To overcome the issues of OODBMS, the **object-relational model** emerged, which is a combination of object-oriented and relational data model. It means that the model should be able to implement SQL for complex data types, such as geographical data types, geometrical data types (polygons, hexagons, etc.), and space data.
- For any relational database, it is not possible to include all object-oriented concepts. And for any object-oriented database, it is not possible to include all relational database concepts. However, some vendors, such as Oracle, IBM, Informix and others provide additional packages to handle complex data types which are sold separately and installed as plug-ins. These packages have different names, like 'data cartridge' is the name of Oracle's type package which is bought separately and plugged into the Oracle RDBMS. Other examples are 'Datablades' of Informix, 'Relational Extenders' of IBM. By using these packages, an user can define his/her own data types and operators.
- Following are the advantages and disadvantages of Object-Relational Data Model.

Advantage:

1. It allows to define user-defined data types and operators and, to access them using Structured Query Language (SQL).

Disadvantages:

1. The information related to user-defined types and operators should be kept in **system catalog (data dictionary)** which requires redesigning of the system catalog. Also, compiler should be updated to access this catalog information.
2. Storage structures and access methods become quite complex.
3. Issues related to indexing on user-defined types are experienced.
4. Besides these drawbacks, several other optimization problems may arise while handling the user-defined types and operators.

2.7 | THREE-LEVEL ARCHITECTURE OF DATABASE

The ANSI/SPARC architecture of database has three levels—Internal, Conceptual and External—which are shown in Figure 2.12.

- There are three levels of database architecture:
    1. Internal level—This is a storage where data are actually stored. It is also known as physical level, *i.e.*, this level is useful for computers to understand the data.
    2. Conceptual level—With the help of this level, the internal and external levels communicate. **Database schema** is defined at this level using **Data Definition Language (DDL).**
    3. External level—This level is concerned with the users. At this level, multiple users access stored data from the database. There may be more than one **external view** for different users.
- The detailed architecture of database is shown in Figure 2.13. The internal level contains the actual stored objects and data. It is useful only for the computers. The internal view or physical view means, how computer sees the data or objects.
- The **external level** is the view of users. There may be many users who are working on different platforms or languages, and they are accessing the data and data structures stored in a database. The users are working in different languages which have different syntaxes to access the data. Users will access only the required part of the database, not the entire database. In Figure 2.13, there are three different users who are using the same table 't', but different fields of that table. Two users are accessing table 't' and its different fields through Visual Basic language and one user is accessing the same table, but

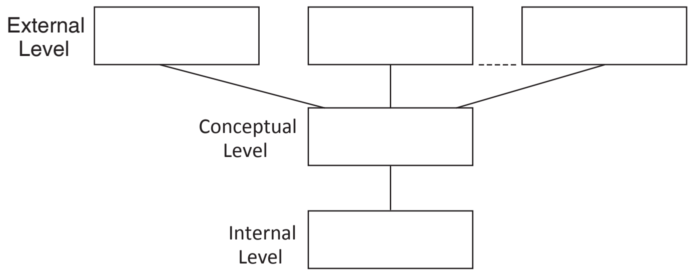


FIGURE 2.12 | Three levels of database architecture.

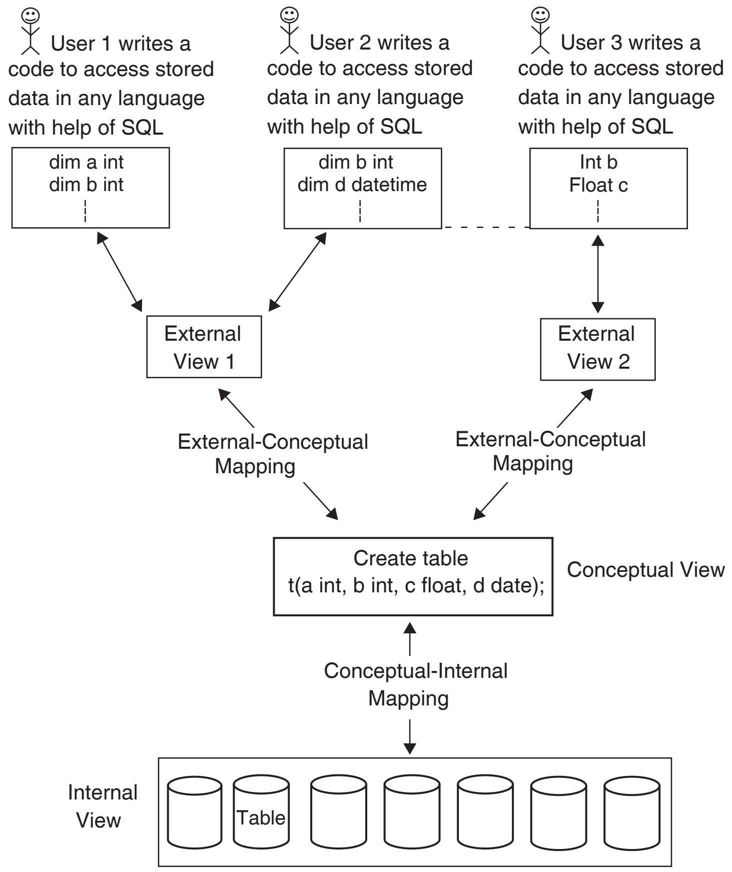

User 1 writes a code to access stored data in any language with help of SQL
User 2 writes a code to access stored data in any language with help of SQL
User 3 writes a code to access stored data in any language with help of SQL
dim a int
dim b int
dim b int
dim d datetime
Int b
Float c
External View 1
External View 2
External-Conceptual Mapping
External-Conceptual Mapping
Create table t(a int, b int, c float, d date);
Conceptual View
Conceptual-Internal Mapping
Internal View
Table

FIGURE 2.13 | Database architecture.

different fields, through structured language 'C'. Data types, to define variables in both the languages, have different syntax and may have different variable names to store fields. The users use a common language for databases, *i.e.*, Structured Query Language, to retrieve data from database. Thus, user's view is a combination of any programming language and Structured Query Language.

- **Conceptual level** works as a 'translator' between internal and external levels. External level is concerned with individual user views, while the conceptual view level, also called logical view, is meant for a group of users which is common for the group. From this common view, each user can access a part of the database relevant to them through some mappings.
- There is only one internal and conceptual view, but there may be multiple external views.
- The users who are accessing the database may be any type of user, such as programmer, end-user, system analyst. Data are retrieved at external view through Data Definition Language (DDL) and **Data Manipulation Language** (DML) which are part of SQL. The combination of DDL and DML is known as **Data Sub Language** (DSL).

- The **internal view** is defined using **internal schema** which is written using internal DDL. Similarly, conceptual view is defined using conceptual schema, and conceptual schema is written using conceptual DDL. External view is defined using external schema, and external schema is written using external DDL.
- There are mappings between these levels. The **Conceptual–Internal mapping** defines the correspondence between the conceptual view and the stored database. It specifies, how conceptual records and fields map into their relative stored fields.
- An **External–conceptual mapping** defines the correspondence between a particular external view and the conceptual view.
- A DBMS (Database Management System) is the software that handles all access to the database. A user issues an access request using some DML (select, insert, delete, or update) command; the DBMS accepts and converts the request; and then performs the necessary operations on the stored database.
- The **Database Administrator** is the person or a group of people responsible for overall control of the database system.

2.8 | DATABASE LANGUAGES

- The generalized database language, which is common in all databases, is Structured Query Language (SQL). It is divided into the following four parts:
  1. Data Definition Language (DDL): The database language which is used to define database objects; to drop database objects; to alter (change) database objects, such as tables, views, users, is known as Data Definition Language (DDL). For example:
     - DDL to create a ZONE table.
       ```
       Create table zone (zoneid integer, zonename char(20));
       ```
     - DDL to alter a ZONE table which changes the data type of a zoneid from integer to char(1).
       ```
       Alter table zone modify column zoneid char(1);
       ```
     - DDL to drop (delete) a ZONE table.
       ```
       Drop table zone;
       ```
  2. Data Manipulation Language (DML): The database language, which is used to insert data, manipulate data; delete data or retrieve data in tables or views, is known as Data Manipulation Language (DML). For example:
     - DML to insert data in a ZONE table.
       ```
       Insert into zone values(1, 'North');
       ```
     - DML to change the value of zonename from 'North' to 'East' for zoneid = 1.
       ```
       Update zone set zonename = 'East' where zoneid = 1;
       ```
     - DML to delete a record of zoneid 1 from ZONE table.
       ```
       Delete from zone where zoneid = 1;
       ```
     - DML to retrieve a record of zoneid 1 from ZONE table.
       ```
       Select * from zone where zoneid = 1;
       ```
  3. Data Control Language (DCL): The database language which is used to control data access is known as **Data Control Language** (DCL). For example, the Grant and

Revoke commands are used to assign insert/delete/update/select privileges or access of specific data. For example:

- ○ DCL to assign insert privilege on table ZONE table to user 'shefali'. To grant or revoke privileges the user himself should have privileges or rights to assign privileges to other users.

```
Grant Insert on zone to 'shefali'.
```

4. Transaction Control Language (TCL): The database language which is used to control transactions is known as **transaction control language** (TCL). For example, the checkpoint and savepoint commands are used to control transactions.
5. Data Sub Language (DSL): The combination of Data Definition Language and Data Manipulation Language is known as Data Sub Language (DSL).

2.9 | DATA AND STRUCTURAL INDEPENDENCE

- When we can change the data type or size of any field without changing the application program, **data independence** is said to be exist.
- When we can change the structure of any table without changing the application program, **structural independence** is said to be exist.
- Data and structural independence is provided by database systems, but 100% data or structural independence is not possible.
- Conversely, when we cannot change the data type or size of any field without changing the application program, **data dependence** is said to be exist.
- When we cannot change the structure of any table without changing the application program, **structural dependence** is said to be exist.

SUMMARY

- A data model is a representation of data and relationships between data.
- Various data models are available. The very first data model was proposed in the 1960s which was a hierarchical data model. It was based on the tree structure that represents the relationship between parent and child records. It was efficient in relating parent and child with the one-to-many relationship. By that time, IMS used to be the popular hierarchical model.
- The main disadvantage of hierarchical model was: it did not support the many-to-many relationship. To overcome this disadvantage, the network model emerged. It provided the functionality to relate parent and child record with the many-to-many relationship. But handling of this relationship was highly complex. IDS is an example of the network model.
- After network model, in the 1970s, the relational model was proposed by E. F. Codd who is admired as the father of relational database systems. He proposed tabular form to represent the data and relationships. It is quite easy to handle relationships in this model by defining primary and foreign keys. Using the general-purpose language of relational databases—Structured Query Language—it is easy to handle data within a database. Relational database systems are the widely used systems in the current era.
- With the evolution of object-oriented languages, the object-oriented data model also emerged which supports all the concepts of object-oriented system. But it does not have

Structured Query Language. It is mainly based on the concept of containment hierarchy. Using object-oriented data model, complex data types, such as images, videos, audios, can be handled properly. But because of the absence of SQL, it is highly difficult to handle the data. This data model is suitable for multimedia databases.

- To provide facility of SQL with object-oriented concepts, the object-relational database came in the picture. But, co-existence of these two is very complex. To handle complex data types, Structured Query Language needs additional 'type package' with RDBMS. Also, there are issues related to storage structure, data dictionary management, access method, and many more with this data model.
- In the architecture of any database, there are three levels—internal, conceptual and external. Internal level is the level where the database is actually stored. It describes how database is physically stored into memory, *i.e.*, through this level, the computer sees the data. The external level describes individual user's view. User accesses stored database by writing programs in various languages at this level, *i.e.*, through this level, the user sees the data. Conceptual view is an indirection between internal and external level. It is the view of a group of users. Internal and conceptual levels are defined using Data Definition Language (DDL), while external level is defined using combination of Data Definition Language (DDL) and Data Manipulation Language (DML) which is known as Data Sub Language (DSL). There is only one internal and conceptual view, but many external views for any database.
- The language which is used to define database objects is known as Data Definition Language (DDL). Create, Alter, Drop are Data definition commands.
- The language which is used to manipulate (change) data is known as Data Manipulation Language (DML). Insert, Delete, Update and Select are Data manipulation commands.
- The language which is used to control data access is known as Data Control Language (DCL). Grant and Revoke are Data control commands.
- The language which is used to manage transactions is known as Transaction Control Language (TCL). Savepoint and Checkpoint are Transaction control commands.
- The combination of DDL and DML is known as Data Sub Language (DSL).
- When there is no need to change application program if we change data type and size of any field, it is called data independence.
- When there is no need to change application program if we change table structure, it is called structural independence.

Exercises

1. What is a data model? Draw chart of various data models and write an example of each type of data model.
2. Which data model does not support the many-to-many relationship between entities? Explain that model with advantages and disadvantages.
3. Discuss Network Data Model with its advantages and disadvantages.
4. Describe Relational Model with its advantages. Which is the biggest drawback of this model?
5. Explain object-oriented and object-relational data models with their advantages and disadvantages.

6. Which are the three level of database architecture? Explain each in detail with diagram.
7. Write a brief note on database languages.
8. Differentiate between data independence and structural independence.
9. Tick the correct answer:

   i. IMS is an example of which data model?
   a. network b. hierarchical
   c. relational d. object-relational

   ii. IDS is an example of which data model?
   a. network b. hierarchical
   c. relational d. object-relational

   iii. OPAL is an example of which data model?
   a. network b. hierarchical
   c. object-oriented d. object-relational

   iv. Oracle is an example of which data model?
   a. network b. hierarchical
   c. relational d. object-oriented

   v. The vendor of IMS is ___________.
   a. IBM b. Oracle
   c. Microsoft d. Informix

   vi. Which model does not support many-to-many relationship?
   a. network b. hierarchical
   c. relational d. object-relational

   vii. SQL is a general-purpose language of which model?
   a. network b. hierarchical
   c. relational d. object-oriented

   viii. SQL is a ___________ generation language.
   a. first b. second
   c. third d. fourth

   ix. Which model requires installing extra package to handle types?
   a. network b. hierarchical
   c. relational d. object-relational

   x. In hierarchical model, the parent segment is known as ________ segement.
   a. branch b. dependent
   c. root d. leaf

   xi. In hierarchical model, the child segment is known as ________ segment.
   a. fruit b. dependent
   c. root d. leaf

   xii. In network model, the parent is known as ________.
   a. owner b. dependent
   c. root d. member

   xiii. In network model, the child is known as ________.
   a. owner b. dependent
   c. root d. member

xiv. In network model, the member which relates two owners with many-to-many relationship is known as.
   a. set   b. reset
   c. combiner   d. joiner

xv. In architecture of database there is (are) ________ internal view (s).
   a. only one   b. zero
   c. many   d. none of the above

xvi. In architecture of database there is (are) ________ conceptual view (s).
   a. only one   b. zero
   c. many   d. none of the above

xvii. In architecture of database there is (are) ________ external view (s).
   a. only one   b. zero
   c. many   d. none of the above

xviii. Internal view means how computer sees the data.
   a. true   b. false

xix. External view means how user sees the data.
   a. true   b. false

xx. The language which defines database object is known as __________.
   a. DDL   b. DML
   c. DCL   d. TCL

xxi. The language which manipulates data is known as __________.
   a. DDL   b. DML
   c. DCL   d. TCL

xxii. The language which controls data is known as __________.
   a. DDL   b. DML
   c. DCL   d. TCL

10. Consider the entities given below and answer the following questions.
   (a) Draw hierarchical and network models for the entities given below.
   (b) Identify the relationships between entities.
   (c) Define primary and foreign keys for each entity. (After you study the chapter on 'The Relational Model')

Batch

Batch	Year
1	1999-2002
2	2000-2003
3	2001-2004
4	2002-2005
5	2003-2006
6	2004-2007
7	2005-2008
8	2006-2009
9	2007-2010
10	2008-2011
11	2009-2012

Class

Class Code	Class Describe	Total Students	Batch
Fy-1	fy div-1	60	1
Fy-2	fy div-2	60	1
Sy-1	sy div-1	60	1
Fy-1	fy div-1	60	2
Sy-2	sy div-2	60	2
Fy-1	fy div-1	90	3
Fy-2	fy div-2	90	3

Faculty

Facid	Faculty Name
1	Shefali Naik
2	Hemal Desai
3	Heena Timani
4	Kunjal Gajjar
5	Trushali Jambudi
6	Aniruddh Parmar
7	Pratik Thanawala

Student

Std No.	Class Code	Batch No.	Std Name	Gender
1	fy-1	1	Hetal Agrawal	Female
2	fy-1	1	Hemal Bavishi	Male
82	fy-2	1	Malav Shah	Male
84	fy-2	1	Megha Mehta	Female
1	fy-1	2	Avani Kapadia	Female
82	sy-2	2	Jenish Shah	Male
63	sy-2	2	Richa Pathak	Female

Teach

Facid	Subject No.	Class Code	Batch
1	102	fy-1	1
1	201	sy-1	1
1	102	fy-2	1
2	104	fy-1	1
2	104	fy-2	1
3	202	sy-1	1
3	202	sy-2	2

Subject

Subject No.	Subject Name
101	Communicative English
102	Internet and HTML
103	C Programming
104	Business Data Processing
105	PC Software
201	Computer System Architecure
202	Mathematical Foundation of Computer Science
203	Operating System
204	Database Management System
205	Windows Programming

3 CHAPTER

Relational Database Management System

CHAPTER OBJECTIVES

- Understanding the RDBMS terminologies.
- Knowing the various types of keys.
- Managing data with integrity rules.
- Using set operators to retrieve data.

3.1 | INTRODUCTION

As we have discussed in Chapter 2, relational model was proposed by E. F. Codd in the 1970s. In his paper on relational data model, Codd explained the concept of relational database based on relational theory of mathematics.

The data models, such as hierarchical and network did not provide data and structural independence, *i.e.*, when data type or data characteristics were changed, the application program had to be changed too. Similarly, when data structures in these models were changed, the application program had to be changed too. There were problems of ordering, indexing and access path dependence. E. F. Codd defined the twelve rules of relational database.

3.2 | RDBMS TERMINOLOGY

Relation: Relational data model is based on relations and relationships between these relations. **Relation** is a combination of related entity occurrences. Relation represents the real world entity. **Entity** is a collection of related entity occurrences. The relation has the following characteristics:

1. Each row represents one **tuple** of a relation (table).
2. The ordering of rows is not important in a relation.

3. All tuples should be distinct (unique). This can be achieved by defining a unique identifier for each tuple.
4. The ordering of columns is significant because it corresponds to the ordering of the domains on which the Relation is defined.
5. The significance of each column is partially conveyed by labeling it with the name of the corresponding domain. For example, the label of a column which contains valid values of student's roll numbers could be 'stdno' or 'student_no' or 'student_ID' or 'stdid' or anything else.)
6. Relation is a two-dimensional structure which is made up of tuples and attributes.
7. The intersection of a tuple and an attribute should contain a single value.

Tuple and Cardinality: Each entity occurrence represents one object. An **entity occurrence** is represented as a tuple in relation. Total number of tuples in a relation is known as **cardinality** of that relation.

Attribute and Degree: Each entity occurrence has some attributes. Attributes describe the object, *i.e.*, an attribute is a characteristic of the object. All attributes together tells about the object. Total number of attributes in a relation is known as **degree** of the relation.

Domain: The values of attributes are derived from some valid set of predefined values, which is known as the **Domain** of an attribute. Two attributes may have the same domain within the relation.

Figure 3.1 shows the relation STUDENT, which represents various STUDENTS instances. STUDENT relation is a collection of only students, *i.e.*, the relation STUDENT can contain details of students, not suppliers or patients or doctors or customers or any other person.

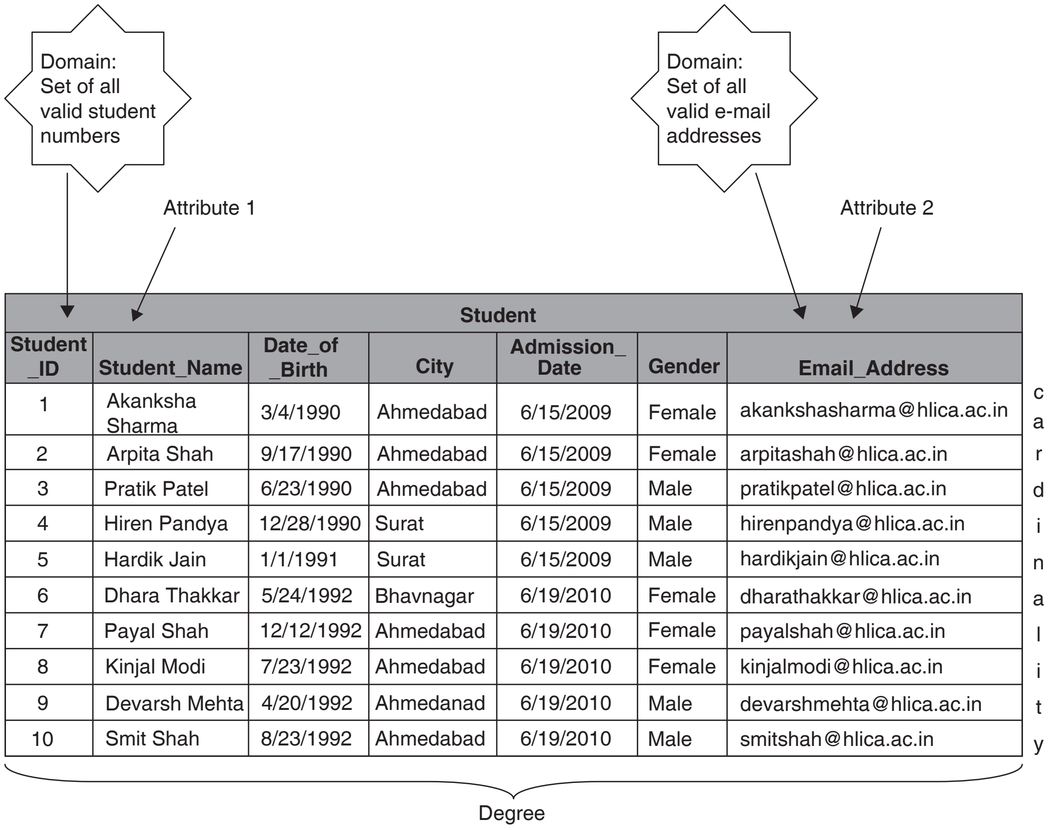
There are 10 tuples in the relation STUDENT. Therefore, cardinality of this relation is 10. There are total of seven attributes in the relation STUDENT. So, degree of this relation is seven, *i.e.*, it is a seven-ary relation.

The relation which has only one attribute is known as **unary relation**, relation with two attributes is known as **binary relation**, relation with three attributes is known as **ternary relation**… and relation with n attributes is known as **n-ary relation**.

In Figure 3.1, STUDENT relation has seven attributes *student_ID*, *student_name*, *date_of_birth*, *city*, *admission_date*, *gender* and *email_address*. The attributes have their values from some valid set of values which are their domains.

The domain of *student_ID* field is an integer. But *student_ID* cannot be any integer, there is a fixed range for *student_ID* and it has to be derived from within that range. In a relational database management system we can restrict this value by defining a constraint on *student_ID*. Thus, when *student_id* is stored into the relational database management system as a field of a table, its data type, size and constraint together will define its domain.

The domain of the *student_name* field is a set of characters. But *student_name* can contain only alphabets. Therefore, it must be restricted only to accept alphabets. Thus, when *student_name* is stored into a relational database management system as a field of a table, its data type, size[char(50)] and constraint (accept only characters from a–z or A–Z) together will define its domain. The same is applicable for the *city* attribute. (*Note:* If we consider


Domain: Set of all valid student numbers
Domain: Set of all valid e-mail addresses
Attribute 1
Attribute 2
cardinality
Degree

Student						
Student_ID	Student_Name	Date_of_Birth	City	Admission_Date	Gender	Email_Address
1	Akanksha Sharma	3/4/1990	Ahmedabad	6/15/2009	Female	akankshasharma@hlica.ac.in
2	Arpita Shah	9/17/1990	Ahmedabad	6/15/2009	Female	arpitashah@hlica.ac.in
3	Pratik Patel	6/23/1990	Ahmedabad	6/15/2009	Male	pratikpatel@hlica.ac.in
4	Hiren Pandya	12/28/1990	Surat	6/15/2009	Male	hirenpandya@hlica.ac.in
5	Hardik Jain	1/1/1991	Surat	6/15/2009	Male	hardikjain@hlica.ac.in
6	Dhara Thakkar	5/24/1992	Bhavnagar	6/19/2010	Female	dharathakkar@hlica.ac.in
7	Payal Shah	12/12/1992	Ahmedabad	6/19/2010	Female	payalshah@hlica.ac.in
8	Kinjal Modi	7/23/1992	Ahmedabad	6/19/2010	Female	kinjalmodi@hlica.ac.in
9	Devarsh Mehta	4/20/1992	Ahmedanad	6/19/2010	Male	devarshmehta@hlica.ac.in
10	Smit Shah	8/23/1992	Ahmedabad	6/19/2010	Male	smitshah@hlica.ac.in

FIGURE 3.1 | Relation STUDENT.

the mathematical domain of *student_id* then it is a set of natural values (N) which contains numbers 1, 2, 3, 4, ..., n)

The *date_of_birth* and *admission_date* have date/time data type as its domain. Additional restriction such as difference between current year and birth year should not be less than 18 at the time when student is admitted.

The *gender* attribute has domain char(6), but it must have only two values—‘Male’ or ‘Female’.

The *email_address* attribute has the domain char(50) with restriction it should contain only alphabets, numbers and three special characters ‘@’ , ‘.’ and ‘_’.

Thus, in real life, the domain is a set of valid values from where value of attribute is derived. When it is implemented through RDBMS, it could be implemented using data types and constraints put on the attribute.

A comparison between RDBMS terminology and traditional terminology is given in Table 3.1.

Table 3.1 | Comparison Between RDBMS and Traditional Terminologies

RDBMS Terminology	Traditional Terminology
Relation	Table
Tuple	Record/Row
Attribute	Field/Column
Domain	Valid set of values
Cardinality	Total number of records in a table
Degree	Total number of fields in a table

3.3 | VARIOUS TYPES OF KEYS

In a relational model, each tuple should be identified with a unique identification number. The unique identification number could be a single attribute value, or combination of attribute values. Also, to relate two relations, there should be a common attribute or combination of attributes between two relations.

An attribute, or combination of attributes which help to define relationships between entities is called a **key**. Any attribute that is part of a key is known as a **key attribute**.

There are various types of keys, such as primary key, composite key, super key, candidate key, alternate key, unique key, foreign key, secondary key and surrogate key. Consider the relations given in Figure 3.2, for which different types of keys can be defined.

1. **Primary Key:** The attribute (column) or combination of attributes which uniquely identifies each tuple(row) of a relation(table) is called a **primary key**. A relation can have only one primary key, but it can be combination of more than one attributes. Primary key cannot contain duplicate or null values. If primary key is a combination of more than one attributes, then combination of all the attribute values should be unique and none of the attributes can contain null value. **Null** means unknown value. It does not mean zero or space, or blank. For the relations as given in Figure 3.2, Table 3.2 shows the primary keys.
2. **Composite Key:** When primary key is a combination of two or more attributes, it is known as **composite key**. Composite key uniquely identifies each tuple of a relation.The

Table 3.2 | List of Primary Keys for Relations Given in Figure 3.2

Relation Name (Table)	Primary Key
Batch	Batchid
Class	Classcode+Batchid (+ sign shows the combination of attributes)
Faculty	Facid
Subject	Subjectno
Student	Stdno+classcode+batchno
Teach	Facid+subjectno+classcode+batchid

Batch	
Batch ID	**Year**
1	1999-2002
2	2000-2003
3	2001-2004
4	2002-2005
5	2003-2006
6	2004-2007
7	2005-2008
8	2006-2009
9	2007-2010
10	2008-2011
11	2009-2012

Class			
Class Code	**Class Desc**	**Total Students**	**Batch ID**
Fy-1	fy div-1	60	1
Fy-2	fy div-2	60	1
Sy-1	sy div-1	60	1
Fy-1	fy div-1	60	2
Sy-2	sy div-2	60	2
Fy-1	fy div-1	90	3
Fy-2	fy div-2	90	3

Faculty	
Facid	**Faculty Name**
1	Shefali Naik
2	Hemal Desai
3	Heena Timani
4	Kunjal Gajjar
5	Trushali Jambudi
6	Aniruddh Parmar
7	Siddhi Shah

Student				
Std No.	**Class Code**	**Batch No.**	**Std Name**	**Gender**
1	fy-1	1	Hetal Agrawal	Female
2	fy-1	1	Hemal Bavishi	Male
82	fy-2	1	Malav Shah	Male
84	fy-2	1	Megha Mehta	Female
1	fy-1	2	Avani Kapadia	Female
82	sy-2	2	Jenish Shah	Male
63	sy-2	2	Richa Pathak	Female

Teach			
Facid	**Subject No.**	**Class Code**	**Batch**
1	102	fy-1	1
1	201	sy-1	1
1	102	fy-2	1
2	104	fy-1	1
2	104	fy-2	1
3	202	sy-1	1
3	202	sy-2	2

Subject	
Subject No.	**Subject Name**
101	Communicative English
102	Internet and HTML
103	C Programming
104	Business Data Processing
105	PC Software
201	Computer System Architecure
202	Mathematical Foundation of Computer Science
203	Operating System
204	Database Management System
205	Windows Programming

FIGURE 3.2 | Database of an institute.

combination of attribute values in composite key for each tuple is unique. No attribute of a composite key can contain null value. A relation (table) can have only one composite key because ultimately it is a primary key. For the relations as given in Figure 3.2, Table 3.3 shows the composite keys.

3. **Super Key:** An attribute or combination of attributes that uniquely identify each record in a table is called a **super key**. Super key is a super set of primary keys. *i.e.*, if we add any attributes of same table into primary key, then it is called a super key. It means, super key = primary key + any other attribute(s) of that table. For the relations as given in Figure 3.2, Table 3.4 shows the super keys.
4. **Candidate Key:** A super key without redundancies or a minimal super key, is called a **candidate key**. The candidate key uniquely identifies each record of a table. In a table, there may be more than one candidate keys. After identifying the candidate keys, any one of the appropriate candidate key is selected as a primary key. The candidate keys have unique values. For the relations as given in Figure 3.2, Table 3.5 shows the candidate keys.
5. **Alternate Key:** The candidate keys, which are not a primary key, are called **alternate keys**. For example, in the table CLASS, one candidate key (*classcode* + *batchid*)

Table 3.3 | List of Primary Keys for Relations Given in Figure 3.2

Relation Name (Table)	Composite Key
Class	Classcode + Batchid (+ sign shows the combination of attributes)
Student	Stdno + classcode + batchno
Teach	Facid + subjectno + classcode + batchid

Table 3.4 | List of Super Keys for Relations Given in Figure 3.2

Relation Name (Table)	Super Key
Batch	i. (Batchid)+year
Class	i. (Classcode+Batchid)+classdesc ii. (Classcode+Batchid)+Totalstudents iii. (Classcode+Batchid)+classdesc+Totalstudents
Faculty	i. (Facid)+facultyname
Subject	i. (Subjectno)+subjectname
Student	i. (Stdno+classcode+batchno)+stdname ii. (Stdno+classcode+batchno)+gender iii. (Stdno+classcode+batchno)+stdname+gender
Teach	—

Table 3.5 | List of Candidate Keys for Relations Given in Figure 3.2

Relation Name (Table)	Candidate Key
Batch	I. Batchid II. Year
Class	I. Classcode+Batchid II. Classdesc+Batchid
Faculty	Fac id
Subject	I. Subjectno II. Subjectname
Student	Stdno + classcode + batchno
Teach	Facid + subjectno + classcode + batchid

combination is selected as a primary key. Therefore, the other candidate key (*class desc+batchid*) combination is known as an alternate key. For the relations as given in Figure 3.2, Table 3.6 shows the respective alternate keys.

6. **Unique Key:** The key which contains unique values id known as **unique key**. Unique key accepts only unique values, but it accepts null values also. For example, classdesc is a unique key in CLASS table.
7. **Foreign Key:** An attribute or combination of attributes, in one table, whose value must either match the primary key in another table or be null is known as **foreign key**. The field or combination of fields can become foreign key, if and only if, it is a primary key of another table. Whenever primary key is referred, the entire primary key should be referred, no single attribute of a primary key can be referred. Also, if the foreign key is referred from a composite key, then the sequence of the attributes should be maintained in the foreign key and data types (domains) of corresponding attributes should match. The attributes may have different names. The foreign key can accept duplicate and null values. For the relations as given in Figure 3.2, Table 3.7 shows the foreign keys.

Table 3.6 | List of Alternate Keys for Relations Given in Figure 3.2

Relation (Table)	Alternate Key
Batch	Year
Class	Classdesc + Batchid
Faculty	—
Subject	Subject name
Student	—
Teach	—

Table 3.7 | List of Foreign Keys for Relations Given in Figure 3.2.

Relation Name (Table)	Foreign Key
Batch	—
Class	Batchid is a foreign key which is referred from batch table's batchid field.
Faculty	—
Subject	—
Student	Classcode+batchid referred from class tables classcode+batchid
Teach	I. Facid referred from faculty table's facid II. Subjectno referred from subject table's subjectno III. Classcode+batchid referred from class tables classcode+batchid

8. **Secondary Key:** The attribute or combination of attributes which are used for retrieval purpose is known as **secondary key**. For example, *subjectname*, *facultyname*, etc., are secondary keys.
9. **Surrogate Key:** The artificial primary key is known as **surrogate key**. When primary key is very complicated and difficult to handle, then surrogate key is required. It is generally an attribute which is a number and can be autogenerated. For example, the table TEACH contains a primary key which has four attributes and difficult to maintain. In this case, one additional attribute say *teachingid* could be taken as a primary key which could be autogenerated number.

3.4 | INTEGRITY RULES

There were problems of integrity with traditional models, such as network and hierarchical models. Change or removal of parent records were not reflected in the child record. User had to do it manually. The relational model facilitate this with the help of defining the Primary Key and the Foreign Key. The two rules, related to these keys, are known as the integrity rules. The first integrity rule is related to the primary key which is known as Entity Integrity Rule, and the second rule related to the Foreign Key is known as the Referential Integrity Rule. Both the rules are explained below:

1. **Entity Integrity Rule (Integrity Rule-I):** 'No primary key value of a relation is allowed to be null, or to have a null component.' Here, **Component** means any attribute which is part of a primary key. If the primary key is a composite key, then each attribute of that composite key is known as one component. To explain this, if a composite key is a combination of four attributes, then there are four components. If a composite key is a combination of two attributes, then there are two components, and so on.

   The entity integrity rule states that primary key cannot contain null value and, if primary key is a composite key, then none of the attribute of this composite key can contain null value. Figure 3.3 shows a table, SUBJECT, in which the primary key is a *subjectno*, which is a single attribute. In this case *subjectno* cannot contain null value. Figure 3.4 shows a table, CLASS, in which primary key is a composite key which is a combination

Subject	
Subject No.	**Subject Name**
101	Communicative English
102	Internet and HTML
103	C Programming
104	Business Data Processing
105	PC Software
201	Computer System Architecure

FIGURE 3.3 | SUBJECT table has Subject No. primary key.

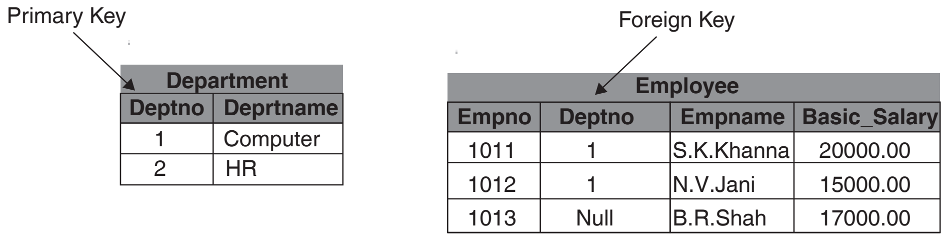
Class			
Class Code	**Class Desc**	**Total Students**	**Batch ID**
Fy-1	fy div-1	60	1
Fy-2	fy div-2	60	1
Sy-1	sy div-1	60	1
Fy-1	fy div-1	60	2
Sy-2	sy div-2	60	2
Fy-1	fy div-1	90	3
Fy-2	fy div-2	90	3

FIGURE 3.4 | CLASS table has (Classcode + Batchid) primary key.

of fields *classcode* and *batchid*. In this case, *classcode* cannot contain null and *batchid* also cannot contain null.

2. **Referential Integrity Rule (Integrity Rule-II):** 'Let **Attribute1** is an attribute for which values are derived from domain **Domain1**. Attribute1 is a primary key of **Relation1** relation. Let **Attribute2** is an attribute for which values are derived from domain of Attribute1 which is **Domain1**. Attribute2 is a part of **Relation2** relation. Then, at any given time, each value of Attribute2 in Relation2 must be either: (a) null or (b) equal to **Value1**, where Value1 is the value of an Attribute1.'

   The above rule says that when any attribute or combination of attributes is a primary key of some table and referred as a foreign key, the foreign key may have either null value or the value of a primary key from where it is referred. Foreign key can have duplicate values. The rule is, data types(domains) of corresponding attributes of primary key and foreign key should be same, the names of attributes may be different. For ex., consider two tables DEPARTMENT and EMPLOYEE in Figure 3.5. If an employee exists, he/she must be working in some department. *i.e.*, the department should exists before employee is assigned to that department. It means that *deptno* field of employee table will have only those values which exists in the *deptno* field of department table. Here, *deptno* field of both the tables share the same domain. If employee's department is not known, in case of new recruitment, then at that time *deptno* in employee table may be kept null, but it cannot have value which is not there in the *deptno* field of department table. Therefore, *deptno* field of employee table could be defined as a foreign key which is reference from the *deptno* field of department.



Department	
Deptno	**Deprtname**
1	Computer
2	HR

Employee			
Empno	**Deptno**	**Empname**	**Basic_Salary**
1011	1	S.K.Khanna	20000.00
1012	1	N.V.Jani	15000.00
1013	Null	B.R.Shah	17000.00

FIGURE 3.5 | Primary key of department table is referred as foreign key in Employee table.

3.5 | RELATIONAL SET OPERATORS

The traditional set operations are Union, Intersection, Difference and Cartesian Product. The two relations (tables) used in Union, Intersection or Difference must be **union-compatible**. It means, they must be of same degree (same number of fields), and all the fields of both the relations must be drawn from same domain (data type). For example, if employee (empno, empname, bdate, deptno) and department (dno, dname) are two relations and, if we are doing union of these two relations, then both should have same number of attributes and corresponding attributes must have same data type. To explain this, when we select fields from tables using a 'select' statements, number of fields in both select statements should be same and data types of corresponding fields should be same. (*Note:* same degree/number of fields in both relations does not mean that if employee table have 5 fields, then department table must have 5 fields. But, if we select 2 fields from the employee table, then we must have to select only 2 fields from the department table.) The same is applicable for intersection and difference (difference is called 'minus' in Oracle).

1. **Union:** The union of two union-compatible relations A and B, A Union B, is the set of all tuples t belonging to either A or B or both. For example, consider tables department and employee given in Figure 3.6. If we take union of department table and employees working in specific department, then the resultant table will display department either department names or employee names working in specific department, or both. The query should be:

   ```
   Select deptno, deptname from department
                   Union
    Select deptno, empname from employee;
   ```

   The resultant table will be Table 3.8.
2. **Intersection:** The intersection of two union-compatible relations A and B, A Intersection B, is the set of all tuples t belonging to both A and B. For example, intersection of department and employee table will display all the records for which deptno matches in both the tables. The query should be :

   ```
   Select deptno from department
            Intersect
    Select deptno from employee;
   ```

   The resultant table will be Table 3.9.
3. **Difference:** The difference between the two union-compatible relations A and B, A minus B, is the set of all tuples t belonging to A and not to B. For example, the difference

Department	
Dept No.	**Dept Name**
1	Computer
2	HR
3	Marketing
4	Production

Employee			
Emp No	**Dept No.**	**Emp Name**	**Basic_Salary**
1011	1	S. K. Khanna	20000
1012	1	N. V. Jani	15000
1013	2	B. R. Shah	17000
1014	2	T. J. Nair	12000
1015	3	V. T. Tripathi	23000

FIGURE 3.6 | Employee and department tables.

between the department table and the employee table will display all the departments which have no employee. The query should be (in Oracle database):

```
Select deptno from department
Minus
Select deptno from employee;
```

The resultant table will be Table 3.10.

4. **Cartesian Product:** The cartesian product of two relations A and B, A times B, is the set of all tuples t such that t is the concatenation of a tuple a belonging to A and a tuple b belonging to B. For example, Cartesian product of department and employee tables will display pair of department and employee. There are 4 records in department and five records in employee, so resultant table will display 4 × 5 = 20 total records. No condition should be specified in select query to display Cartesian product. The query should be:

```
Select department.deptno, deptname, empname
from department, employee;
```

The resultant table will be Table 3.11.

Table 3.8 | Union of Department and Employee

Union	
Dept No.	**Dept Name**
1	Computer
1	N. V. Jani
1	S. K. Khanna
2	B. R. Shah
2	HR
2	T. J. Nair
3	Marketing
3	V. T. Tripathi
4	Production

Table 3.9 | Intersection of Department and Employee

Intersection
Dept No.
1
1
2
2
3

Table 3.10 | Differenceof Department and Employee

Difference
Dept No.
4

Table 3.11 | Cartesian Product of Department and Employee Tables

Cartesian Product		
Dept No.	**Dept Name**	**Emp Name**
1	Computer	S K Khanna
2	HR	S K Khanna
3	Marketing	S K Khanna
4	Production	S K Khanna
1	Computer	N V Jani
2	HR	N V Jani
3	Marketing	N V Jani
4	Production	N V Jani
1	Computer	B R Shah
2	HR	B R Shah
3	Marketing	B R Shah
4	Production	B R Shah
1	Computer	T J Nair
2	HR	T J Nair
3	Marketing	T J Nair
4	Production	T J Nair
1	Computer	V T Tripathi
2	HR	V T Tripathi
3	Marketing	V T Tripathi
4	Production	V T Tripathi

3.6 | RETRIEVAL OPERATORS

To retrieve particular data from tables, retrieval operators are used. The result of any retrieval is again a table. Resultant table is retrieved from source tables. To form a resultant table, more than one database tables may be used. There are 3 types of such retrieval operators: (i) Select, (ii) Project, and (iii) Join.

- **Select:** The select operator creates a new table by taking a horizontal subset of an existing table, *i.e.*, all rows of an existing table that satisfies some condition. For example, if we want to find details of employee 1011, then it can be done by the select operator.

  ```
  Select * from employee where empno=1011;
  ```

  It will display Table 3.12.
  Employee is the original table on which we are applying select operator to find details of employee whose number is 1011.
- **Project:** The project operator displays a vertical subset of an existing table by retrieving the specified columns. For example, if we want to display the columns deptno

Table 3.12 | Result of SELECT Operator

EmpNo	Dept No.	EmpName	Basic_Salary
1011	1	S. K. Khanna	20000

and empname from employee table then it can be done by project operator. To implement the project operator the following query should be written, which will display Table 3.13.

```
Select deptno, empname from employee;
```

Employee is the original table on which we are applying project operator to find all employee numbers and employee names.

- **Join:** If two tables contain a column defined over a same domain (domain = data type), they may be joined over those two columns. The result of this join is a new table in which each row will contain attributes from selected tables. For example, if we want to display department name and employees working in that department, then it can be done by join operator. To implement the join operator the following query should be written, which will display Table 3.14.

```
Select deptname, empname from employee, department where
        department.deptno= employee.deptno;
```

3.7 | CODD'S TWELVE RULES OF RELATIONAL DATABASE

E. F. Codd represented the following twelve rules of relational database:

1. **Information Rule:** Information should be stored as a value in each cell of a table. **Cell** is an intersection of row and column.
2. **Guaranteed Access Rule:** Each value of a table should be accessed through table name and key attributes/columns.
3. **Null Values:** 'Null' represents unknown values of a column irrespective of data type.

Table 3.13 | Result of Project Operator

Dept No.	Emp Name
1	S. K. Khanna
1	N. V. Jani
2	B. R. Shah
2	T. J. Nair
3	V. T. Tripathi

Table 3.14 | Join of Department and Employee Tables

Dept Name	Emp Name
Computer	S. K. Khanna
Computer	N. V. Jani
HR	B. R. Shah
HR	T. J. Nair
Production	V. T. Tripathi

4. **System catalog maintenance:** Data about data is generated and stored automatically in the system catalog or data dictionary.
5. **Database language:** There must exists one common language for all relational databases to manage the data within database. Syntax of this language should remain same in all the databases. This language is currently known as Structured Query Language.
6. **Rule for updatable view:** views should be updatable.
7. **Insert, Update and Delete operations:** Data should be inserted, deleted and updated to/from tables.
8. **Physical Data Independence:** Application programs should be independent from memory locations.
9. **Logical Data Independence:** Application programs should be independent from data values.
10. **Integrity Independence:** All the tables within one database should be integrated properly through relationships.
11. **Distribution Independence:** Application program should be independent from table's location.
12. **Non-subversion Rule:** All the rules written on fields of a table should be enforced for each record.

3.8 | DATABASE LIFE CYCLE

Database life cycle is a part of System Development Life Cycle (SDLC). During system design phase, the database of the system is designed. Database Life Cycle (DBLC) contains the following phases:

1. Database selection: From the System Requirements Specification (SRS) Document, proper database management system is selected to manage the system's data.
2. Database design: According to SRS, the tables are designed with its fields and constraints. Tables are integrated with proper relationship.
3. Data loading or data transfer (conversion): After tables are designed, the data are stored in the tables through the application software after they are tested for the correctness. If data are already stored in some other database, then they are imported (transferred) using proper command or application.
4. Maintenance of data: In case of any changes in data due to changing requirements, the database is changed and data are maintained.

3.9 | DATA DICTIONARY

Data dictionary is also known as system catalog. It stores data about data or **metadata**. Data dictionary contains all the tables, description of tables, fields and its data types, constraints which should be applied on the fields, format of the fields, specific set of values which fields should contain, etc. Data dictionary also contains description of other objects of database.

SUMMARY

- Relational model is based on the concepts of set theory of mathematics. In relational model, entity is stored as a relation, each entity occurrence is stored as a tuple and characteristic of an entity is stored as an attribute.
- Relation means table, tuple means row/record, and, attribute means column/field.
- Total number of columns in relation is known as a degree of relation and total number of rows is known as cardinality.
- A domain is a valid set of values from where attribute's values are derived.
- Each relation must have unique tuple which can be obtained by defining unique identifier for each tuple. This unique identifier is known as the primary key of the table.
- To maintain integrity between data, foreign key can be defined, which should be referred from the parent table's primary key. The data type of both the fields should match.
- Primary key is a field or combination of fields which uniquely identifies each tuple. Primary key contains unique and not null values.
- Candidate key is also a field or combination of fields which uniquely identifies each tuple. From the list of candidate keys, any one key is selected as a primary key, the remaining keys are known as alternate keys.
- Super key is a super set of primary key. That is, if we add some other fields from the same relation in a primary key, it is known as super key.
- Unique key contains unique values. The difference between primary key and unique key is: unique key can contain null values, while primary key cannot.
- Secondary key is used for retrieval purpose.
- Surrogate key is an artificial primary key which is introduced in place of the original primary key when it is very difficult to handle the original primary key.
- To maintain integrity, two rules should be applied-Entity integrity and Referential integrity.
- Rule of entity integrity says that primary key/composite key cannot contain null values. In case of composite key, any component cannot contain null value. Component means, individual field of a composite key.
- Rule of referential integrity says that foreign key can be derived from the primary key of some other table or same table. The data type of a foreign key should be same as data type of a primary key of other table. Foreign key can have either null value or the value of a field from where it is referenced.
- There are different set operators which can be applied on tables. Union, Intersection, Difference and Cartesian Product are set operators.
- Union will display rows from first table or second table or from both the tables.
- Intersection will display common rows from first table and second table.
- Difference will display records from the first tables which do not exists in the second table.
- Cartesian Product will display cross join of two tables. It will display pair of records from both the tables.
- There are operators-Select, Project and Join which are used to retrieve data.
- Select will display horizontal subset of the table, Project will display vertical subset of the table, and Join will display fields from two tables.

Exercises

1. Explain the terminologies of a relation model.
2. What is domain? Give any five examples of domain.
3. Discuss various types of keys giving suitable examples.
4. State and explain entity and referential integrity rules.
5. Describe set operators with examples.
6. Describe retrieval operators with examples.
7. Tick the correct answer:

   i. The key which contains unique and not null values is ____________ key.
   a. foreign b. primary
   c. unique d. secondary

   ii. Unique key contain null values.
   a. true b. false

   iii. Foreign key can be derived from the ____________ key of another table.
   a. foreign b. primary
   c. unique d. secondary

   iv. Relation means ____________.
   a. field b. record
   c. table d. attribute

   v. Attribute means ____________.
   a. field b. record
   c. table d. tuple

   vi. Tuple means ____________.
   a. field b. record
   c. table d. attribute

   vii. Entity integrity rule is related with ____________ key.
   a. foreign b. primary
   c. unique d. secondary

   viii. Referential integrity rule is related with ______________ key.
   a. foreign b. primary
   c. unique d. secondary

   ix. Each composite key is a primary key, but each primary key is not necessarily composite key.
   a. true b. false

   x. ____________ is a horizontal subset of a table.
   a. Select b. Project
   c. Join d. Divide

   xi. ____________ is a vertical subset of a table.
   a. Select b. Project
   c. Join d. Divide

   xii. ____________ displays records from first or second or both the tables.
   a. Union b. Intersection
   c. Divide d. Difference

xiii. ___________ displays common records from both the tables.

a. Union b. Intersection

c. Divide d. Difference

xiv. Domain of Rollno. Field can be __________.

a. Integer b. Float

c. Date d. Text

xv. Domain of birthdate Field can be ___________.

a. Integer b. Float

c. Date d. Text

8. Identify all possible types of keys from the following tables.

Event_type

Event_type
Solo
Group

Event_category

Cat_ID	Cat_Name
1	Drama
2	singing
3	dance
4	Intellect
5	Art
6	Sports
7	Others

Participant

Pid	Pname	Gender	Class ID	RollNo	Phone	Bdate
1	Nida	F	SY-II	111		
2	Vedangi	F	SY-II	86		
3	Sushil	M	SY-II	80		
4	Poonam	F	TY-II	57		
5	Kanan	F	TY-II	115		
6	Robin	M	TY-I	20		
7	Jay	M	TY-I	50		
8	Niti	F	TY-I	23		
9	Ritu	F	TY-I	3		
10	Nilesh	M	SY-I	35		
11	Sajan	M	SY-I	1		

Event_transaction

Event_ID	Parti_ID
17	4
17	3
1	4
3	4
3	3
3	6
11	1
11	8
5	5

Eventheader

Event ID	Total_Parti	Faculty_IC	Leaderid
17	1	Heena Timani	4
17	1	Heena Timani	3
1	1	Shefali Naik	4
3	1	Hemal Desai	4
3	1	Hemal Desai	3
3	1	Hemal Desai	6
11	1	Shefali Naik	1
11	1	Shefali Naik	8
5	5	Hemal Desai	5

Winner

Event ID	Winner1_leader ID	Winner2_leader ID	Winner3_leader ID
17	3	4	NULL
1	NULL	NULL	NULL
3	3	4	6
5	NULL	NULL	5

Event_master

Event ID	Event_cat ID	Event_Type	Event_Desc	Min_Parti	Max_Parti
1	2	Solo	Solo Singing	1	1
2	2	Group	Group Singing	2	8
3	3	Solo	Solo Dancing	1	1
4	3	Group	Duet Dancing	2	2
5	3	Group	Group Dancing	3	10
6	4	Group	Debate	2	2
7	4	Solo	Elocution	1	1
8	4	Solo	Extempore	1	1
9	4	Group	Quiz	1	3
10	5	Solo	Rangoli	1	1
11	5	Solo	Mehendi	1	1
12	5	Group	Best Out Of Waste	2	3
13	6	Solo	Slow Cycling	1	1
14	6	Solo	Lemon And Spoon	1	1
15	6	Group	Treasure Hunt	3	6
16	6	Group	3 Legs	2	2
17	7	Solo	Mimicry	1	1
18	7	Group	Mime	3	8
19	7	Group	Skit	3	8
20	7	Solo	Mono Acting	1	1

4 CHAPTER

Developing Entity-Relationship Diagram

CHAPTER OBJECTIVES

- Importance of Entity-Relationship diagram in database design.
- Understanding entities, and how to identify entity from a given problem.
- Understanding how entities can be related to each other.
- Knowing relationship types.
- Knowing participation of an entity into another entity.
- Representing entity and relationships by using notations.
- Modelling Entity-Relationship diagram.
- Converting Entity-Relationship diagram into Relational Model.
- Understanding object modelling.
- Identifying classes, subclasses, and superclasses.
- Drawing class diagrams.
- Comparison between E-R diagram and class diagram.
- Summary

4.1 | INTRODUCTION

There are various semantic models, such as entity-relationship model (E-R model), data model, object model, etc. The **semantic model** represents and explains the meaning of the real-world concepts implemented in that model. An E-R model is the most popular semantic model.While we are in the process of database designing, it is important to identify the correct tables, their fields, definitions of each field (data type, size, constraints, formats, etc.) and relationship between each table of the same database. It is very important to design tables in such a way that they cover all the requirements of the system, in the form of data stored within the tables. Before we design tables, we need to keep the rules of relational database in mind. For example, each table should have a key which uniquely identifies each tuple; table should be related with a common key which should be a primary key of other table, etc. Therefore, it is advisable to identify entities and relationships between entities before we design the actual tables. Once we finalize the entity-relationship diagram for the system, it can be converted into relational model. There are many symbol sets available, nowadays, in the market to design and draw the entity-relationship diagrams. The concept of E-R Model was given by Chen in 1976. Thereafter, the

Crow's Foot model was developed in the 90's. The Chen model and Crow's Foot model symbol sets are widely accepted and used to draw E-R diagrams. In this book, we are going to use symbols from the Chen model.

4.2 | IDENTIFYING ENTITIES

When we have to develop any software or system, we need to design the database in which we are going to store data of that system or software. To design the tables, entities should be identified from the given problem definition or problem description.

Entity or entity set is a collection of any real-life similar type of objects which can be described using its characteristics. Using these characteristics, each object of an entity can be differentiated with other objects of that entity. Each object will have its own values for the characteristics. The characteristics are also known as **attributes**. For example, STUDENT is an entity which has the characteristics rollno, name, birth_date, city, contactno, email_ID, etc., (*i.e.*, each unique student can be described with the help of values of its characteristics. Figure 4.1 describes the entity STUDENT and its characteristics. Figure 4.2 shows values of characteristics for some student objects within STUDENT entity).

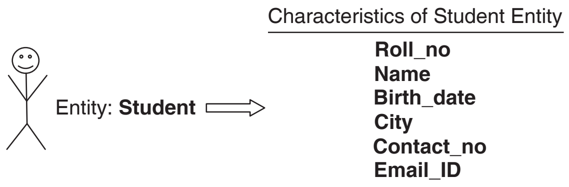


FIGURE 4.1 | Student entity and its characteristics.

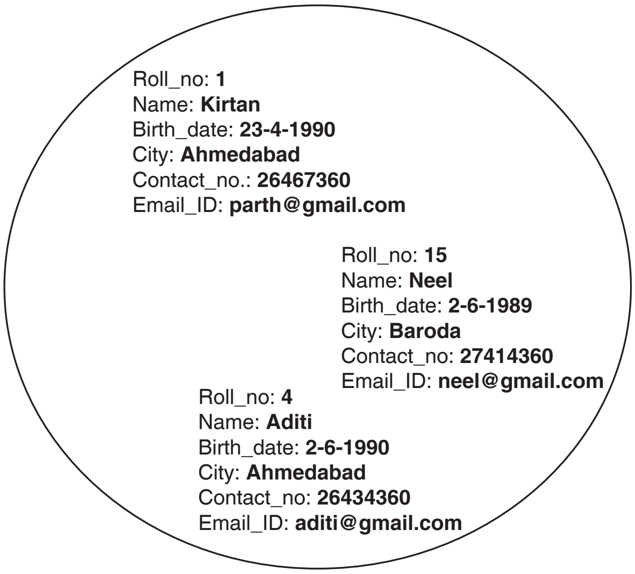


FIGURE 4.2 | Three objects with their characteristics' values within student entity.

It is not possible to combine two different objects in the same entity, because different objects have different characteristics. For example, it is not possible to combine 'STUDENT' and 'PRODUCT' objects within the same entity, as it is not possible to compare any student with any product.

The examples of entities are Supplier, Patient, Doctor, Employee, Student, Nominee, Staff, Equipment, Item, Product, Stationery, Company, Location, Department, Warehouse, University, Institute, Bank, Branch, Exam, Shipment, Competition, Treatment, etc.

We can represent any person, any place, any item or thing, any event or activity as entity. Figure 4.3 shows the possible classification of the above examples of entities as per their types.

Entity is represented as a horizontal rectangle in Chen's model. Name of the entity should be written in capital letters within the symbol of an entity. It must be any noun and singular. Figure 4.4 describes the entity COUNTRY as a symbol.

The characteristics or attributes are represented as oval shape or capsule shape and should be connected to an entity using a line segment. Figure 4.5 shows the representation of attributes country_code and country_name in a Chen's model. These are the attributes of entity COUNTRY. Table 4.1 shows the terminologies of E-R model in brief. There are following concepts related to attribute:

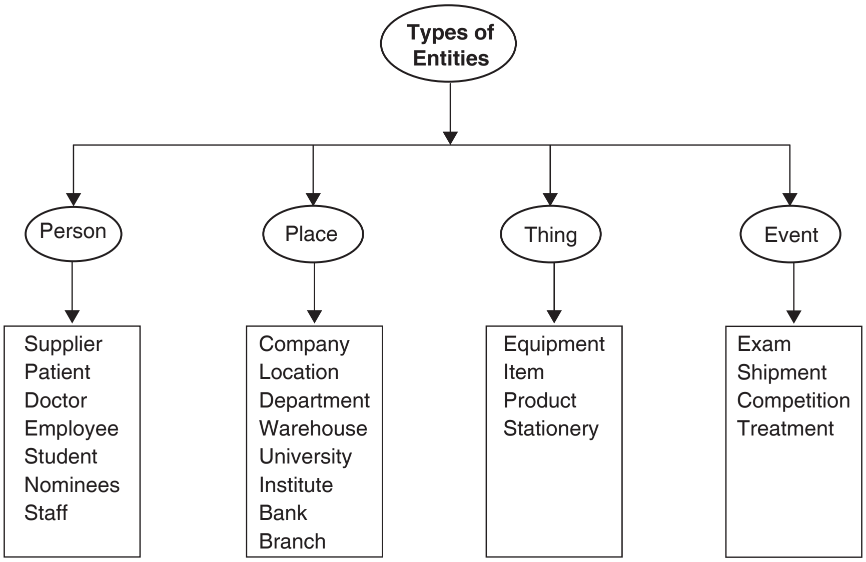


FIGURE 4.3 | Types of entities and their examples.

FIGURE 4.4 | Entity: Country

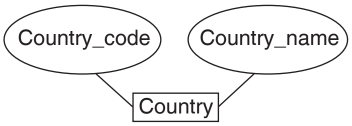


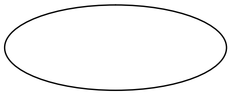
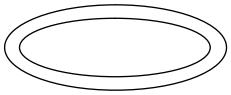
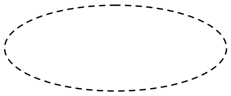
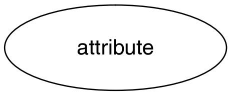
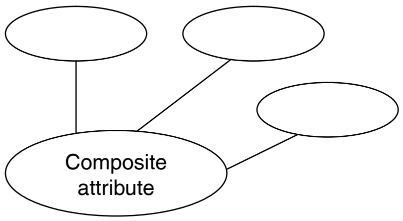
FIGURE 4.5 | Representation of attributes of an entity country.

Table 4.1 | Terminologies of E-R Model

Terminology	Meaning
Entity(Entity set)	A set of similar type of objects (but each object is unique within the set) which can be distinguished from other set of objects. For example, STUDENT, COUNTRY, BOOK
Object(Entity instance)	Here, it can be described as entity instance or entity occurrence, *i.e.*, a collection of values of characteristics for any student represents one unique object. For example, within COUNTRY entity the collection of values of (country_code, country_name) characteristics (IND, INDIA) represents one object.
Attribute	Characteristics of an entity. (Each object within the same entity will have same characteristics, but values of characteristics will be different.) For example, attributes of entity BOOK are ISBN, title, price, author, etc.
Relationship/relationship set (relationship entity set)	Association between entities. For example, the relationship between entities MEMBER and BOOK is MEMBER issues BOOK.

- **Single-valued Attribute:** The attribute which contains a single value for any entity instance is called the **single-valued attribute**. For example, gender, age, retirement_date, blood_group, name, etc. The single-valued attribute is denoted with an oval shape in Chen model.
- **Multi-valued Attribute:** The attribute which contains many values or multiple values for any entity instance is called the **multi-valued attribute**. For example, degree, hobby, contactno., etc. The multi-valued attribute is denoted with an oval within oval shape in Chen model.
- **Derived Attribute:** The attribute which is derived from another attribute is known as **derived attribute**. For example, retirement_date is a derived attribute because its value is calculated from the attribute birthdate by adding 60 years (if person's retirement age is 60) in the birth date. Another example is the value of total_years_of_experience attribute is calculated from the attribute joining_date (by subtracting current year from the joining year). The derived attribute is denoted with a dashed oval shape in Chen model.
- **Simple Attribute:** The attribute which can not be further divided into more attributes is known as **simple attribute**. For example, hobby, gender, joining_date, etc. The simple attribute is denoted with an oval shape in Chen model.
- **Composite Attribute:** The attribute which can be further divided into more attributes is known as **composite attribute**. For example, name, address, etc., are composite attributes because name can be divided into simple attributes first_name, middle_name and surname; address can be further divided into addressline1, addressline 2, city and pincode. The composite attribute is denoted with an oval shape connected with its simple attributes through line segments in Chen model.
- **Key Attribute or Identifier:** The key attribute is an attribute which uniquely identifies each entity. For example, employee's PAN Number uniquely identifies each entity instance in an entity. The key attribute is underlined and written within the oval shape in Chen model. Table 4.2 shows notations to denote different attributes in Chen model.

Table 4.2 | Different Types of Attribute Symbols in Chen model

Attributes	Symbols in Chen Model
Single-valued	
Multi-valued	
Derived	
Key attribute	
Composite attribute	

4.3 | IDENTIFYING RELATIONSHIPS

The way in which individual entities interact with each other is known as **relationship.** For example, entities TEACHER and STUDENT will interact with each other when a teacher *teaches* student. In other way, we can say that student is *taught by* teacher. The relationship between TEACHER and STUDENT entities is teaching or learning. In most of the cases, the relationship is nothing but an event or activity.

When any event or activity takes place, it is between two or more than two entities, *i.e.*, an event or activity relates two or more entities. Therefore, it is referred as relationship. In some of the cases, the entity can also be associated or related with entity itself.

4.4 | TYPES OF RELATIONSHIPS

When entities are associated with relationship, one object of an entity can be associated with one or more objects of other entity or with itself.

The symbol to represent relationship is diamond shape in Chen model of E-R. Name of the relationship is written within the diamond shape should be any verb. We can read relationship from either side. Therefore, the relationship name can be written as present verb or past perfect verb. For example, the relationship between STUDENT and INSTRUCTOR could be either 'instructed by' or 'instructs'. We can read this relationship as 'STUDENT *instructed by*

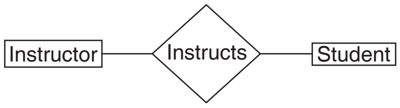


FIGURE 4.6 | Representation of relationship.

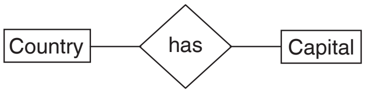


FIGURE 4.7 | Representation of 1:1 (one -to-one) relationship.

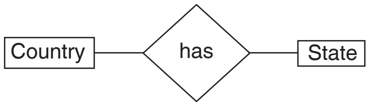


FIGURE 4.8 | Representation of 1:M (one-to-many) relationship.

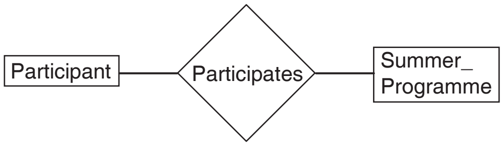


FIGURE 4.9 | Representation of M:N (many-to -many) relationship.

INSTRUCTOR' or 'INSTRUCTOR *instructs* STUDENT'. The relationship should be connected with entities with line segments. Figure 4.6 shows the representation of relationship between STUDENT and INSTRUCTOR entities. We can classify the relationships in following three types:

One-to-One Relationship: This relationship describes that one object of one entity can be associated with one object of another entity. For example, the relationship between entities COUNTRY and CAPITAL is one to one, *i.e.*, one COUNTRY can have only one CAPITAL and a CAPITAL belongs to any one COUNTRY. This relationship is represented pictorially as following. Symbolically, **one to one relationship** is represented as 1:1. Figure 4.7 shows one to one relationship.

One-to-Many or Many-to-One Relationship: This relationship describes that one object of one entity can be associated with many objects of another entity or many objects of one entity can be associated with one object of other entity. For example, the relationship between entities COUNTRY and STATE is one to many, *i.e.*, one COUNTRY can have many STATE and many STATE belong to one and only one COUNTRY. This relationship is represented pictorially as following. Symbolically, **one to many relationship** is represented as 1:M. Figure 4.8 shows one to many relationship.

Many-to-Many Relationship: This relationship describes that many objects of one entity can be associated with many objects of another entity. For example, the relationship between entities PARTICIPANT and SUMMER_PROGRAMME is many to many, *i.e.*, one STUDENT participates in many SUMMER_PROGRAMME and one SUMMER_PROGRAMME contains many PARTICIPANT. Symbolically, **many to many relationship** is represented as M:N. Figure 4.9 shows many to many relationship. We can also classify the relationship according to its degree as following:

Unary or Recursive Relationship: The degree of relationship is said to be unary when the entity is related with itself. As entity is related with itself, it is also known as **recursive relationship**. For example, the STUDENT entity represents all the student instances. From all the students, one student is a class representative, who monitors the class. Therefore, this

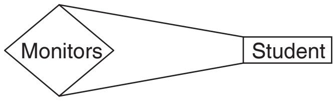


FIGURE 4.10 | Unary relationship: STUDENT (CR) monitors STUDENT.

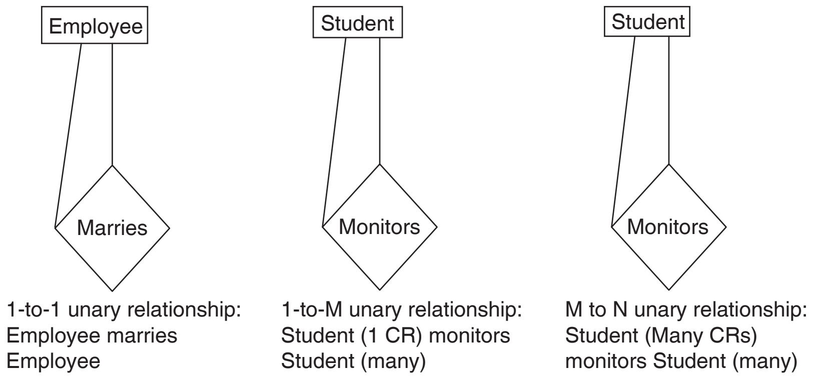

Employee
Marries
1-to-1 unary relationship: Employee marries Employee
Student
Monitors
1-to-M unary relationship: Student (1 CR) monitors Student (many)
Student
Monitors
M to N unary relationship: Student (Many CRs) monitors Student (many)

FIGURE 4.11 | 1:1, 1:M and M:N unary relationships.

relationship can be represented as STUDENT (here role of this student is CR—Class Representative and any STUDENT becomes CR if and only if he/she is first a student *monitors* STUDENT. In **unary relationship** degree of relationship is 1. Unary relationship appears rarely in E-R model. Figure 4.10 represents unary relationship. The unary relationship can be further categorized as:

1. **One-to-One Unary Relationship**: If one entity instance appears one and only once in other entity, the type of relationship is said to be 1 to 1. If this happens within the same entity, it is called 1 to 1 unary relationship. For example, consider employees working in an organization. It is possible that two employees of this organization marry each other. In this case, one EMPLOYEE marries only one EMPLOYEE. Therefore, the relationship is one to one. As the relationship exists within the same entity EMPLOYEE, it is said to be **1 to 1 unary relationship**. Figure 4.11 represents 1:1, 1:M and M:N unary relationships.
2. **One-to-Many Unary Relationship:** If one entity instance of an entity appears many times in other entity, the type of relationship is said to be 1 to M. If this happens within the same entity, it is called 1 to M unary relationship. For example, one STUDENT(CR) monitors many STUDENT, but each student is monitored by only one STUDENT(CR). Hence, this relationship is a **1 to many unary relationship**.
3. **Many-to-ManyUnary Relationship:** If many entity instances of an entity appears many times in other entity, the type of relationship is said to be M to N. If this happens within the same entity, it is called M to N unary relationship. For example, assume that there are more than one Class Representatives(CRs) who are also students, then they will monitor other students of the class. Therefore, this relationship is known as **M to N unary relationship.**

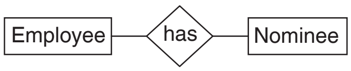


FIGURE 4.12 | 1:M binary relationship.

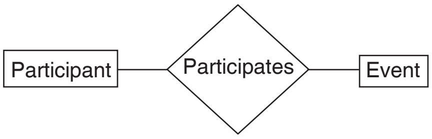


FIGURE 4.13 | M:N binary relationship.

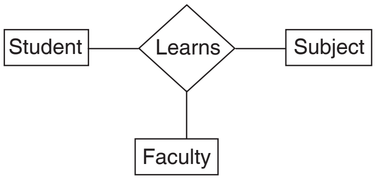


FIGURE 4.14 | M:N ternary relationship.

Binary Relationship: The degree of relationship is said to be binary when one entity is related with another entity. The relationship type between entities may be 1:1, 1:M or M:N. For example, the relationship EMPLOYEE *has* NOMINEE is a **1 to many binary relationship**. Because one employee may have many nominees, but a nominee depends on only one employee. [*Note:* Nominee is a person who is dependent on the employee. For example, if Mr Neel Shah is working in any company, then his nominees are his family members. So, Neel Shah may have many family members as nominees (dependents), but Mr Neel Shah's family members depend only on Mr Neel Shah and not on any other employee of the same company.] Figure 4.12 shows 1:M binary relationship between entities EMPLOYEE and NOMINEE. Figure 4.13 represents **many to many binary relationship** 'PARTICIPANT *participates* EVENT' between entities PARTICIPANT and EVENT, *i.e.*, one participant can participate in many events and one event can have many participants. In binary relationship, degree of relationship is 2.

Ternary Relationship: The degree of relationship is said to be ternary when three entities are related with each other with common relationship. For example, Figure 4.14 shows **M:N ternary relationship** between three entities STUDENT, FACULTY and SUBJECT, *i.e.*, one student learns many subjects from many faculties. One faculty teaches many subjects to many students and one subject is taught to many students by many faculties. Figure 4.14 represents many to many ternary relationship 'STUDENT learns SUBJECT', 'SUBJECT taught by FACULTY' and 'FACULTY teaches STUDENT'. In ternary relationship, degree of relationship is 3.

N-ary Relationship: As unary, binary and ternary, any number of entities can be associated with each other using a common relationship. Therefore, **N-ary relationship** can be defined as when n entities are related with each other using common relationship, the relationship is said to be N-ary relationship. In N-ary relationship, degree of relationship is N.

4.5 | RELATIONSHIP PARTICIPATION

When one entity is associated with another entity, this association is either mandatory (compulsory) or optional.

Optional Participation: Optional participation means that if entity instance (object) of one entity participates zero times in another entity, the participation is said to be optional. For example, consider two entities—STUDENT and SUMMER_PROGRAMME. The relationship between these two entities is 'STUDENT *attends* SUMMER_PROGRAMME'', *i.e.*, one student can attend only one summer programme, but one summer programme can be attended by many students. Here, it is not required that if summer programme exists then it should have student participants. It may be the case that there is not a single participant in some summer programmes. Therefore, we can say that participation of SUMMER_PROGRAMME in the STUDENT entity is optional. Figure 4.15 shows this optional participation. The optional participation is denoted by writing 0 (zero).

Mandatory participation: Mandatory participation means that if entity instance (object) of one entity participates one or many times in another entity, the participation is said to be mandatory. For example, consider two entities—DIVISION and STUDENT. The relationship between these two entities is 'STUDENT studies DIVISION'', *i.e.*, one division contains many students, but there must be at least one student in a division and a student is enrolled in one and only one division. Here, it is compulsory that if division exists then it must contain at least one student, *i.e.*, participation of DIVISION entity in a STUDENT entity is compulsory. Figure 4.16 shows this mandatory participation. The mandatory participation is denoted by writing 1,M. When we want to show exact one participation, 1,1 is written near the entity. Different participations symbols are shown in Table 4.3.

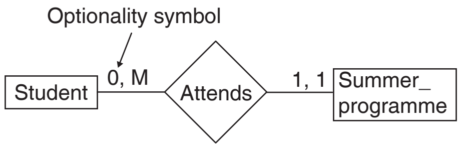


FIGURE 4.15 | Optional participation of SUMMER_PROGRAMME in STUDENT.

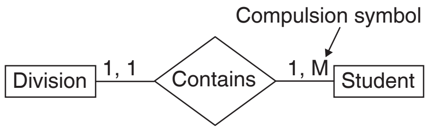


FIGURE 4.16 | Mandatory participation of DIVISION in STUDENT.

Table 4.3 | Terminologies of E-R model

Symbol	Meaning
0,1	1 is optional, *i.e.*, zero or one.
0,M	Many is optional, *i.e.*, zero or many.
1,1	1 is compulsory, but not more than 1, *i.e.*, one and only one.
1,M	Many is compulsory, *i.e.*, one or many.

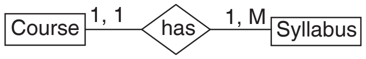


FIGURE 4.17 | Strong relationship and weak entity.

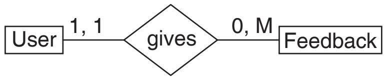


FIGURE 4.18 | Weak relationship and strong entities.

4.6 | STRONG AND WEAK RELATIONSHIP

Strong Relationship: When existence of any entity depends fully upon another entity, the relationship between these two entities is called **strong relationship**. For example, consider two entities COURSE and SYLLABUS. The existence of SYLLABUS entity totally depends on the COURSE entity, *i.e.*, if COURSE exists then it is required to create its SYLLABUS. COURSE without SYLLABUS is not possible. In Chen model of E-R, the strong relationship is denoted using diamond (relationship entity) shape. When strong relationship exists between entities, the dependent entity is called **weak entity.** Weak entity is denoted as rectangle within rectangle and two parallel lines are shown from weak entity to the relationship. We can say that when relationship between entities is strong, the dependent entity is always weak and vice versa is also true. Figure 4.17 shows strong relationship between the entities COURSE and SYLLABUS and weak entity SYLLABUS.

Weak Relationship: When existence of any entity does not depend on another entity, the relationship between these two entities is called **weak relationship**. For example, consider two entities USER and FEEDBACK. It is not required that each USER should give FEEDBACK. In Chen model of E-R, the weak relationship is denoted by using diamond within diamond (relationship entity) shape. When weak relationship exists between entities, the entities are known as **strong entities**. We can say that when relationship between entities is weak, the entities are always strong. Figure 4.18 shows weak relationship between the entities USER and FEEDBACK.

4.7 | MANAGING MANY-TO-MANY RELATIONSHIP

When there exists many-to-many relationship between entities, the many-to-many relationship should be converted into two relationships—one-to-many (1:M) and many (N:1) to one. For example, the relationship 'TEACHER teaches STUDENT' is many-to-many which says that one teacher teaches many students and each student is taught by many teachers. Figure 4.19 shows M:N relationship between entities STUDENT and TEACHER and representation of this relationship as one-to-many (one STUDENT taught by many TEACHER) and many-to-one (one TEACHER teaches many STUDENT).

4.8 | EXAMPLE OF E-R MODEL

To draw an E-R diagram from the given real-time problem statement, we may follow the following procedures:

1. Understand the problem thoroughly.
2. Identify entities and its attributes.

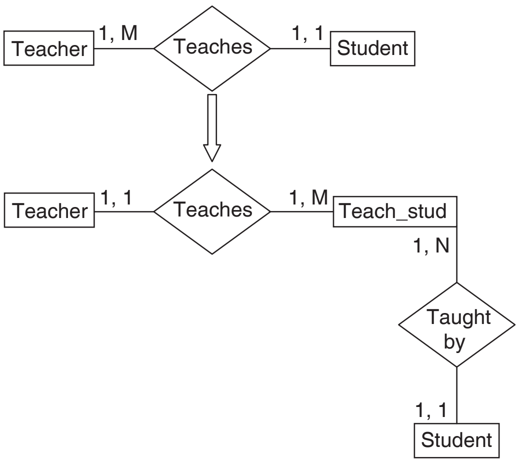


FIGURE 4.19 | Representation of M:N relationship as 1:M and N:1.

3. Identify one attribute or combination of attributes which could be used to represent each entity instance uniquely.
4. Categorize the attributes as simple, composite, single-valued, multi-valued and derived. Identify simple attributes for the given composite attribute.
5. Identify relationships between each entity. Remember that no entity should remain isolated in the E-R diagram. Specify type of relationship whether it is 1:M, M:N or 1:1.
6. Covert many to many relationship further as one to many and many to one.
7. Identify strong and weak relationships.
8. Draw E-R diagram and check the correctness of the diagram. If required, make changes and refine further.

Example 4.8.1: Consider the following problem statement and draw E-R diagram for it.

A training association organizes various programmes on different topics for the people of different age groups, different interest, different streams, etc., throughout the year. It offers membership of the organization and gives discount to its members. It also arranges different types of events, such as workshops, conferences, conventions, etc., for its members only. It has halls, auditoriums and rooms which are given unique room number according to floors. Rooms are used to organize events and programmes (courses). Its members and non-members can participate in the courses by registering themselves into courses. They invite experts from different fields who serve as resource persons for the courses.

Solution 4.8.1:

First, we will have to identify entities and attributes. To identify the entities, consider the proper nouns given in the problem definition. Think about the nouns whether they are entities or just an attribute. For example, in the first line 'A training association organizes various programmes on different topics for the people of different age groups, of different interest, of different backgrounds, etc., throughout the year.' Nouns are training association, programmes, topics, people, age, interest, stream and year. Here training association is an organization for which we are going to draw E-R diagram; so it can't become an entity. From others, we can say that

programmes and people are entities. 'Topics and year' are attributes of entity programme because they describe the programme (*i.e.*, programme is organized on which topic and in which year). Also, we can identify more attributes for programme which are not specified in the problem definition, but should be used such as starting date, ending date, duration of the programme, etc. Another entity is people, which will have attributes such as name, age, interested in, backgrounds, etc. Similarly, analyze the entire problem definition, understand it and decide entities and attributes. Question yourself while analyzing the problem and you will get the answer. Remember that, 'Practice makes man perfect'. If you will try to solve various kinds of problems, you will be an expert in drawing E-R diagram. After analyzing the above problem definition, the following entities and attributes are identified. (They might be wrong in the early stage, but that can be revised or corrected at the later stages.) There is no need to show all the attributes in the E-R diagram, only key attributes and important attributes could be shown.
Programme (Course) with attributes **topic, year in which it is organized, starting date, ending date, duration,** etc.

Note: Here, to identify each programme uniquely (*i.e.*, key attribute). We need one unique attribute. None of the attributes or combination of attributes which we have specified can serve as a key attribute because they will have duplicate values. For example, there are many programmes on same topic; starting date, ending date and duration could be same for different programmes. Therefore, we will have to consider some unique code to identify each programme (each programme means entity instance/object of an entity programme) uniquely, say prog_ID (programme identification number). Similarly, think about other entities.

Person with attributes **person_ID, name, birth_date, age, interested_in, background**, etc.
Member with attributes **member_ID, member_type** (values could be employee/student/others)**, organization** (company name/school name) **name, age, interested_in, membership_type** (values could be life time or temporary)**, discount_given, membership_date, membership_renewal_date**, etc.
Event with attributes **event_ID, event_name, starting_datetime, ending_datetime, company_name**, etc.
Infrastructure entity has attributes **floor_ID, roomno, description, capacity**, etc.
Expert (Resource Person) entity has attributes **expert_ID, name, expert_in**, etc.

Following are the key attributes in each entity specified:

- **In programme entity**, the attribute duration is a derived attribute which can be calculated from two attributes—starting_date and ending_date.
- **In person entity,** the attribute age is a derived attribute which can be calculated from other attribute birth_date. The attribute interested_in is a multi-valued attribute, because a person may be interested in many subjects. Person's name is a composite attribute because it can be further divided into fname and surname.
- **In member entity**, the attribute interested_in is a multi-valued attribute, because a person may be interested in many subjects. Member's name is a composite attribute, because it can be further divided into fname and surname.
- **In expert entity**, the attribute expert_in is a multi-valued attribute, because a person may have expertise in many topics.

The following relationships exist between the entities identified on the previous page:

- **Member registers for programme/course** is a many to many relationship, *i.e.*, one member registers for many programmes and one programme contains many members.
- **Person registers for programme/course** is a many to many relationship, *i.e.*, one person registers for many programmes and one programme contains many persons.
- **Infrastructure conducts programme** is a many to many relationship, *i.e.*, one infrastructure(room) conducts many programmes and one programme is conducted into many infrastructure(room) on different dates.
- **Expert teaches programme/course** is one to many relationship, *i.e.*, an expert can teach many programmes, but one programme can be taught by only one expert.
- **Member organizes event** is one to many relationship, *i.e.*, one member can organize many events, but an event is organized by only one member.
- **Event books infrastructure** is a many to many relationship, *i.e.*, one event is conducted into many infrastructure and an infrastructure may conduct many events.

From the above relationships, all many to many relationships can be decomposed into one to many and many to one relationships.

- **Member registers for programme/course** can be converted as
  - member *registers* reg_mem(1 : M) and
  - reg_mem *registers for* programme(N:1)
- Similarly, **person registers for programme/course** can be converted as
  - person *registers* reg_per(1:M) and
  - reg_per *registers for* programme(N:1)
- **Infrastructure conducts programme** can be converted as
  - Infrastructure conducts infra_prog(1 : M) and
  - Infra_prog conducts programme(N:1)
- Similarly, **Infrastructure conducts event** can be converted as
  - Infrastructure conducts infra_event(1 : M) and
  - Infra_prog conducts event(N:1)

The following relationships are strong relationships:

- member *registers* reg_mem(1 : M) (reg_mem is dependent entity which depends on member entity)
- reg_mem *registers for* programme(N:1) (reg_mem is dependent entity which depends on programme entity)
- person *registers* reg_per(1 : M) (reg_mem is dependent entity which depends on person entity)
- reg_per *registers for* programme(N:1) (reg_mem is dependent entity which depends on programme entity)
- infrastructure conducts infra_prog(1 : M) (infra_prog is dependent which depends on infrastructure)
- infra_prog conducts programme(N:1) (infra_prog is dependent which depends on programme)
- infrastructure conducts infra_event(1 : M) (infra_event is dependent which depends on infrastructure)
- infra_event conducts event(N:1) (infra_event is dependent which depends on event)

The following relationships are weak relationships:

- expert *teaches* programme(1 : M)(programme is dependent)
- member *organizes* event(1 : M) (event is dependent)

From the above specifications we can draw the E-R diagram for the training association which is shown in Figure 4.20.

4.9 | EXTENDED E-R MODEL

The E-R diagram can be extended further by implementing some new concepts. By implementing these concepts, it will be easier to understand the diagram.

Sometimes, an entity inherits all the properties of any other entity. In that case, the base entity from where other entity inherits all the properties is called supertype and the entity which inherits properties from base entity is called its supertype. This relationship is known as 'is a' relationship.

As the example given in Figure 4.20, the person entity is a base entity. Entities Expert and Member inherit all the properties (attributes) of entity Person. So, we can say that 'Expert is a Person' and 'Member is a Person'. Here, Person is a supertype, while Expert and Member

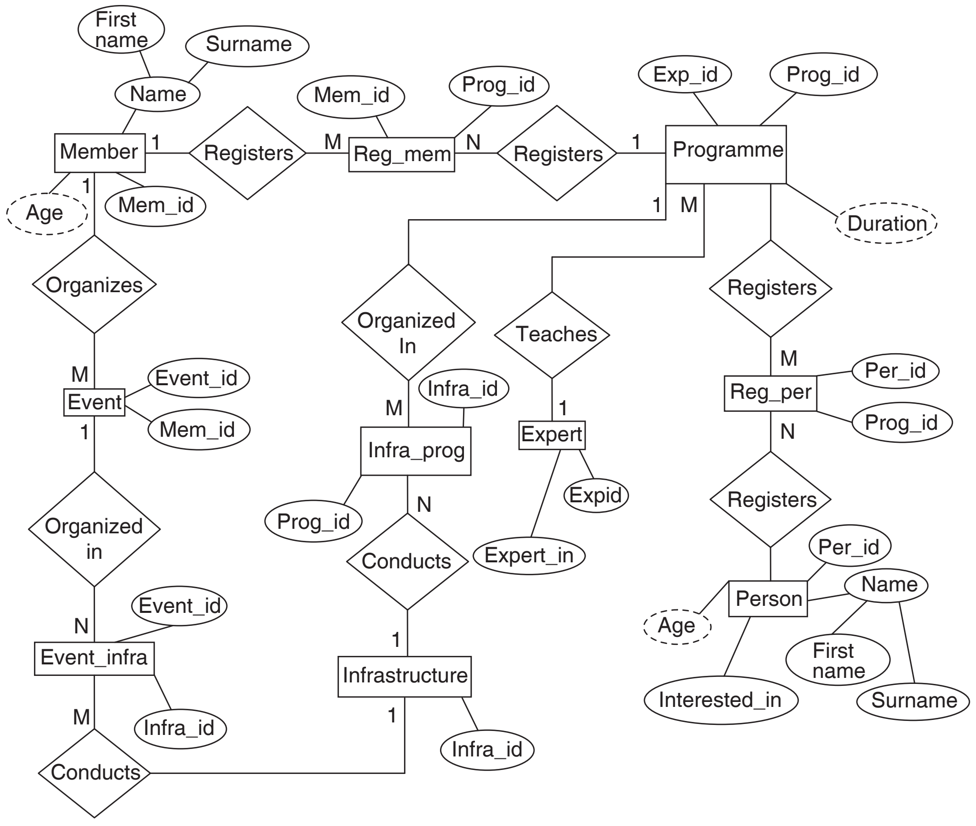


FIGURE 4.20 | E-R diagram for a training association.

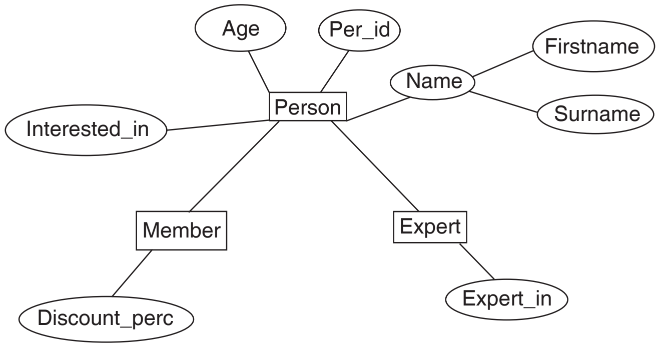


FIGURE 4.21 | Representation of supertype and subtype in E-R diagram.

are its subtypes. Figure 4.21 shows this 'is a' relationship. To denote **'is a' relationship**, the line with arrow is shown from supertype(Person) to subtype(Member and Expert).

Entities Member and Expert will have all the attributes of Person and some attributes of their own, but vice versa is not true, *i.e.*, an expert has its own attribute expert_in which is not an attribute of person entity. Similarly, a member has its own attribute discount_perc which is not an attribute of person entity.

In the above diagram, person entity is 'general' entity; while expert and member entities are 'special' entities of person type entity. These concepts are called **'Generalization'** and **'Specialization'** respectively.

4.10 | CONVERTING E-R MODEL INTO RELATIONAL MODEL

When we convert E-R diagram into Relational model, the rules we should follow are given below:

1. Each entity is converted as a 'table and entities' attributes are converted as fields. The identifier, which is underlined in E-R diagram, is converted as primary key of that table. Decide data types for each attribute from its value. Remember that the key attribute of one entity participating in another entity will have the same data type of a key attribute.
2. The derived attributes may or may not be converted as field. It depends on the database designer whether to store the derived attribute or not. There are certain advantages and disadvantages of storing derived attribute as a field. If the attribute's value is changing with time, it is advisable not to store it. For example, the value of 'age' will change with time; therefore, it should be calculated every time when we want to use it in process. If the attribute's value is not going to be changed with time, it is advisable to store it in table once it is calculated. For example, the value of 'total bill amount' is calculated from the sum of multiplication of quantity and price, *i.e.*, once it is calculated, it will not be changed in future.
3. When we convert weak entity into table, the identifier of strong entity on which the weak entity is dependent becomes foreign key of the weak entity as well as it becomes part of the primary key of that weak entity. In other cases, the foreign key will not be part of the primary key.

4. The entities with cardinality 1 should be created first as tables. After that, entities with cardinality M/N should be created.

If we convert the example given in Figure 4.20 into relation model, it will contain the tables which are given in Table 4.4. The third column in Table 4.4 shows data definition language to create tables in Oracle.

Table 4.4 | Tables which are Converted from the E-R Diagram given in Figure 4.20.

Table Name	Fields	Create Table command in Oracle
Person	Person_ID, name, birth_date, age, interested_in, background	`Create table person(person_ID int primary key, name varchar(20), birth_date date, age float, interested_in varchar2(100), background varchar2(50));`
Member	Member_ID, member_type, organization, name, birth_date, age, interested_in, membership_type, disc_given, membership_date, renewal_date	`Create table member(member_ID, int_primary key, member_type varchar(10), organization varchar(30), name varchar(30), birth_date date, age float, interested_in varchar(100), membership_type varchar(10), disc_given float, membership_date date, renewal_date date)`
Expert	Expert_ID, name, birth_date, age, expertise_in	`Create table expert(expert_ID int primary key, name varchar(30), birth_date date, age float, expertise_in varchar(30))`
Infrastructure	Infra_ID, floor_ID, roomno, description, capacity	`Create table infrastructure(infra_ID int, floor_ID int, roomno int, description varchar(30), capacity int)`
Programme	Prog_ID, expert_ID, topic, year, starting_date, ending_date	`Create table programme(prog_ID int primary key, expert_ID int references expert(expert_ID), topic varchar(30), year int, starting_date date, ending_date date)`
Event	Event_ID, member_ID, event_name, starting_date, ending_date	`Create table event(event_ID int primary key, member_ID int references member (member_ID), event_name varchar(30), starting_date date, ending_date date)`
Reg_mem	Member_ID, progid	`Create table reg_mem(prog_ID int references programme(prog_ID), member_ID int references member(member_ID), primary key(prog_ID,member_ID))`
Reg_per	Person_ID, progid	`Create table reg_per(person_ID int references person(person_ID), prog_ID int references programme(prog_ID), primary key(person_ID, prog_ID));`
Event_infra	Event_ID, infra_ID	`Create table event_infra(event_ID int references event(event_ID), infra_ID int references infrastructure (infra_ID), primary key(event_ID, infra_ID))`
Infra_prog	Infra_ID, prog_ID	`Create table infra_prog(infra_ID int references infrastructure(infra_ID), prog_ID int references programme(prog_ID), primary key(infra_ID, prog_ID))`

4.11 | OBJECT MODELLING

Object modelling is another type of **semantic model**. In this model, entities are referred as **class**; each entity instance is referred as **object** and characteristics of entity are referred as **attributes**.

The relationships between these classes along with participation of one class into another class should be shown in **object model**. If we convert the example given in Figure 4.20 into relation model, it will contain the tables which are given in Table 4.4. The third column in Table 4.4 shows data definition language to create tables in Oracle.

Object model is more semantic (meaningful) than E-R model because it represents real world concepts more clearly.

Object model represents E-R model's entity as class, entity's characteristics as attributes, entity instance as object and relationship as association. Moreover, it assigns each object a unique identification no. Attributes have their behaviour and classes have methods. Each class can send message to other class. Classes are associated with each other through association. Participation of one object into another object is shown using cardinality.

The **subtype** and **supertype** in E-R model is known as **subclass** and **superclass**, respectively, in object model. Table 4.5 shows comparison between E-R model and Object model.

Object model represents some more concepts which are not available in E-R model, such as inheritance, encapsulation, polymorphism, etc.

In object model, the class is represented as vertical rectangle. Class name is written on the top of the rectangle. Below class name, the attributes are written and below attributes, the methods are written. The associations between classes are denoted with line segment connecting two classes. The examples of class, attributes and methods are shown in Figure 4.22.

Table 4.5 | Comparison between E-R Model and Object Model

Entity-Relationship Model	Object Model
Entity	Class
Entity instance	Object
Characteristics (Attributes)	Attributes
Relationship	Association
Subtype	Subclass
Supertype	Superclass

Class name	Event
Attributes	event_ID member_ID event_name starting_date ending_date
Methods	add event remove event update event

FIGURE 4.22 | Representation of class EVENT with its attributes and methods.

4.11.1 | Subclass and Superclass

The object model represents class hierarchy as subclass and superclass. The class which inherits some or all the attributes and methods from some class is known as **subclass**. The class from where other class inherits the attributes and methods is known as **superclass**. The class hierarchy is shown in Figure 4.23.

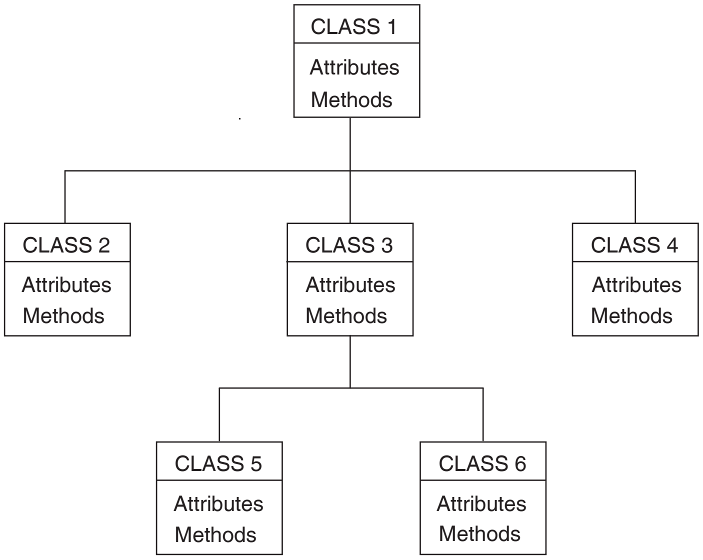


FIGURE 4.23 | Representation of class hierarchy.

In Figure 4.23, the CLASS 1 is a superclass of classes CLASS 2, CLASS 3 and CLASS 4. CLASS 2, CLASS 3 and CLASS 4 are subclasses of CLASS 1. CLASS 3 is a superclass of classes CLASS 5 and CLASS 6. CLASS 5 and CLASS 6 are subclasses of CLASS 3. Here, CLASS 3 is superclass as well as subclass.

4.11.2 | Specialization and Generalization

Subclasses inherit the attributes and methods from the superclass. In this case, we can say that subclasses are specialization of its superclass. For example, the classes EXPERT and MEMBER are subclasses of superclass PERSON, *i.e.*, EXPERT and MEMBER classes are **specialization** of class PERSON.

Conversely, superclass is a **generalization** of subclasses. For example, PERSON class is a generalization of classes EXPERT and MEMBER. Figure 4.24 shows the concept of Generalization and Specialization.

4.11.3 | Class Diagram

The **class diagram** shows association between classes. It is similar to E-R diagram. We can follow the same rules of drawing E-R diagram to draw the class diagram. Then, we convert E-R diagram into class diagram. If we draw the class diagram for the problem definition given in Example 4.8.1, it will be as shown in Figure 4.25.

SUMMARY

- E-R model is a semantic model which represents real world concepts in meaningful way. E-R model was proposed by Peter Chen.

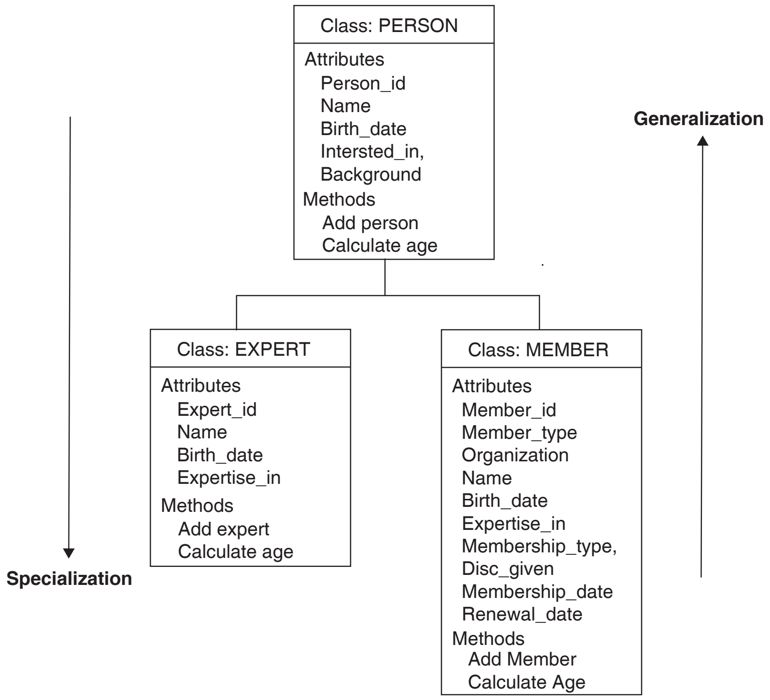


FIGURE 4.24 | Class specialization and generalization.

- Entity means any real world object which can be differentiated from any other real world object.
- Characteristics of an entity are called attributes. The different types of attributes are simple attribute, composite attribute, single-valued attribute, multi-valued attribute and derived attribute.
- Each entity is a collection of objects which share the same attributes, but each object is unique.
- Entity can be associated with each other using relationship.
- Relationship can be categorized as unary, binary, ternary and so on up to n-ary, which is known as degree of relationship.
- The relationship which associates entity with itself is called unary relationship. The relationship which associates two entities is known as binary relationship. The relationship which associates three entities is known as ternary relationship. The relationship which associates n number of entities is known as n-ary relationship.
- The relationships are of three types: one to one, one to many (or many to one) and many to many.
- One entity participates in other entity through relationship, but this participation may be optional or mandatory. This participation is denoted as cardinality in E-R diagram.
- The relationship could be strong or weak relationship. If one entity totally depends on the existence of another entity, the relationship is said to be strong relationship. In this case,

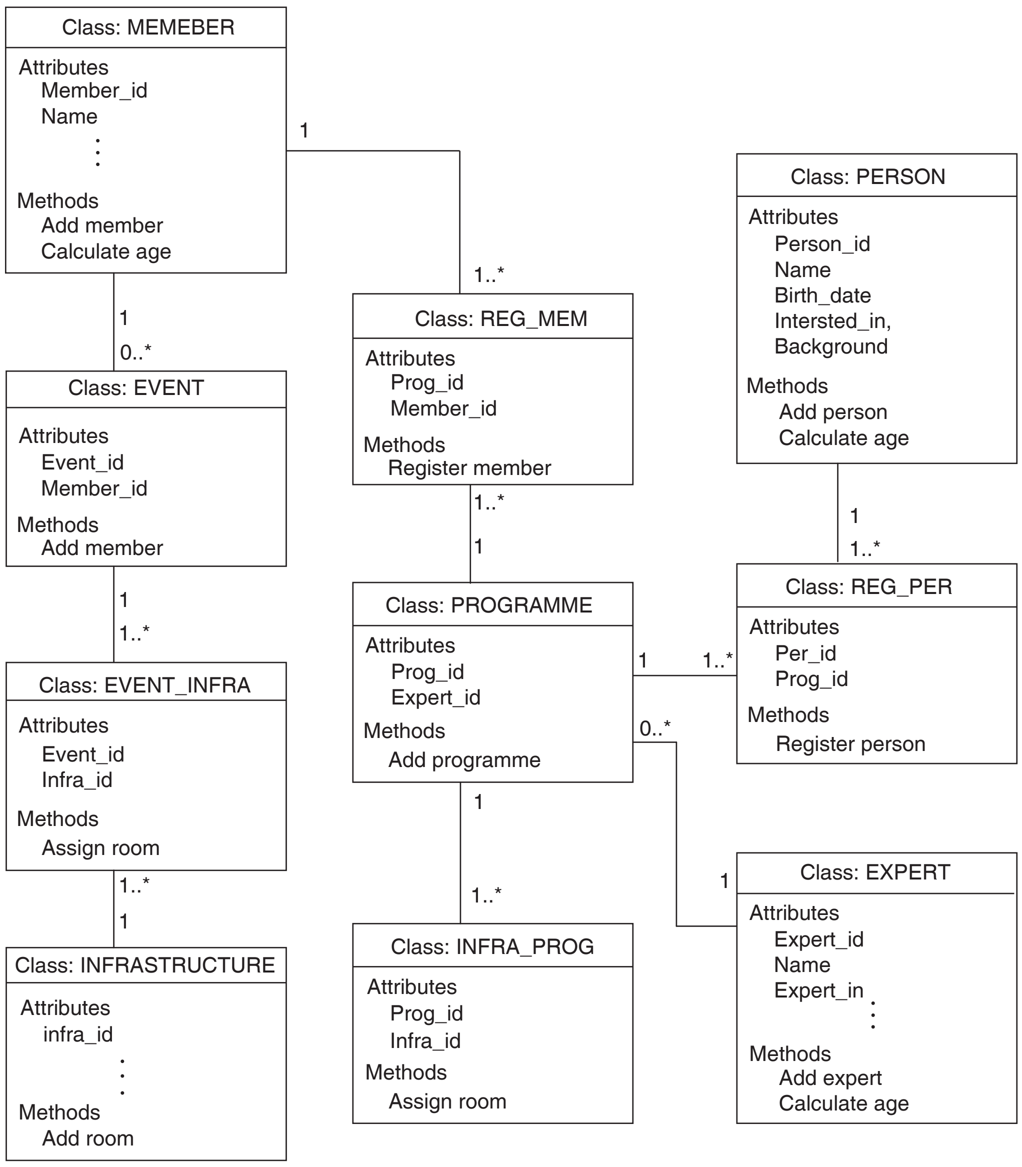


FIGURE 4.25 | Class diagram.

the dependent entity is called weak entity. If existence of an entity doesn't totally depend on other entity, the relationship is said to be weak relationship. In this case, the entities are called strong entities.

- To draw an E-R diagram various symbols are used. Some important symbols are rectangle to represent entity, oval to represent attribute, diamond to represent relationship and line segment to connect entity with relationship.

After drawing an E-R diagram, it is converted into Relational Model, *i.e.*, entities are converted into tables, attributes are converted into fields. Each table will have unique identifier, which is known as primary key.

Exercises

1. Draw symbols which are used to draw E-R diagram.
2. Define the following:
   a. Entity
   b. Relationship
   c. Entity instance
   d. Attribute
   e. Simple attribute
   f. Composite attribute
   g. Single-valued attribute
   h. Multi-valued attribute
   i. Derived attribute
   j. Strong relationship
   k. Weak entity
   l. Supertype
   m. Subtype
3. Compare E-R model and Object model.
4. Explain one to one, one to many and many to many relationships with example.
5. Explain unary/recursive relationship with example.
6. Show symbolic representation of following relationships.
   a. One singer sings many songs and a song is sung by many singers.
   b. Student gives many exams.
   c. Many food items are listed in a menu.
   d. One author writes many books and a book is written by many authors.
   e. Professor writes many research papers.
7. Draw an E-R diagram for the kindergarten according to the rules given below:
   a. There are classes like playgroup, nursery and KG.
   b. There are many divisions of each class.
   c. In each division there are many students, but one student studies in only one division.
   d. Each division is assigned to two teachers.
   e. Parent-teacher meeting is held on every even Saturday. Parents' attendance is maintained for each meeting.
   f. Progress of each student is maintained in every month.
8. Draw an E-R diagram for the summer camp which is held in the school during summer vacation. Use following rules to draw the diagram.
   a. There are many activities under different categories such as sports, intellectual, art, etc., are organized. Sports activities such as football, volleyball, badminton, table tennis, basketball, swimming, skating, etc.; art activities such as calligraphy, paper craft, sand sculptures, glass painting, etc.; intellectual activities such as effective speaking, fun with maths, fun with science, good reading habits, etc.; are organized.
   b. One participant can participate in many activities under different categories.
   c. Each activity has a schedule.
   d. Each activity is conducted by one resource person. A resource person can conduct many activities at different time.

9. Draw an E-R diagram for the newspaper distributor for the given procedure.
   The newspaper distributor daily collects various newspapers from different printing press. The newspapers are distributed area-wise among the persons who look after that area. The area distributor has many employees who will distribute the newspaper to the clients. The area distributor sorts the newspapers client-wise and hand over them to his employees. Employees distribute bunch of newspapers to each client. On demand of his client, the newspaper distributor also distributes periodicals. In the month end, he delivers bill of newspaper to each client.
10. Select the correct answer:
    a. The attribute which contains multiple attributes is called ___________.
       i. Composite attribute ii. Multi-valued attribute
       iii. Derived attribute iv. None of the given
    b. The attribute which is calculated from the value of some other attribute is called ___________.
       i. Composite attribute ii. Multi-valued attribute
       iii. Derived attribute iv. None of the given
    c. The attribute which can be further divide into simple attributes is called _________.
       i. Composite attribute ii. Multi-valued attribute
       iii. Derived attribute iv. None of the given
    d. From the following, ‘Kindergarten’ entity can be categorized as which type of entity?
       i. Person ii. Place
       iii. Thing iv. Event
    e. From the following, ‘Supervisor’ entity can be categorized as which type of entity?
       i. Person ii. Place
       iii. Thing iv. Event
    f. From the following, ‘Furniture’ entity can be categorized as which type of entity?
       i. Person ii. Place
       iii. Thing iv. Event
    g. From the following, ‘Registration’ entity can be categorized as which type of entity?
       i. Person ii. Place
       iii. Thing iv. Event
    h. Recursive relationship means _____________ relationship.
       i. Binary ii. Ternary
       iii. Unary iv. N-ary
    i. The _____________ model is more semantic than E-R model.
       i. Hierarchical ii. Network
       iii. Relational iv. Object
    j. When E-R model is converted into relational model, entity is converted in a ___________.
       i. Field ii. Table
       iii. None
    k. In Chen model of E-R, entity is denoted with which symbol?
       i. Rectangle ii. Diamond
       iii. Oval iv. Line segment

l. In Chen model of E-R, relationship is denoted with which symbol?
   i. Rectangle   ii. Diamond
   iii. Oval   iv. Line segment

m. In Chen model of E-R, simple attribute is denoted with which symbol?
   i. Rectangle   ii. Diamond
   iii. Oval   iv. Line segment

n. In Chen model of E-R, derived attribute is denoted with which symbol?
   i. Dashed oval   ii. Oval within oval
   iii. Oval   iv. Oval connecting oval

o. In Chen model of E-R, multi-valued attribute is denoted with which symbol?
   i. Dashed oval   ii. Oval within oval
   iii. Oval   iv. Oval connecting oval

p. In Chen model of E-R, key identifier (attribute) is denoted with which symbol?
   i. Dashed oval   ii. With underlined
   iii. In bold face   iv. Italic

5 CHAPTER

Normalization

CHAPTER OBJECTIVES

- Understanding importance of normalization.
- Identifying dependencies from the given Table.
- Converting Tables into various normal forms.
- Evaluating Tables after normalization for correctness and lossless decomposition.
- Understanding some more dependencies.
- Learning with examples: how to normalize Tables.

5.1 | INTRODUCTION

After drawing E-R diagram, the Tables should be designed. An E-R model is a conceptual (semantic) model which shows only entities, relationships and attributes. It does not show data types and constraints which should be put up on the attributes. **Data types** and **constraints** must be decided by studying actual values of the attributes. Moreover, we can decide input format, set of values to be inputted in the attribute, etc.

If **E-R diagram** is converted into **Relational Model**, we get Tables which are already converted into **normal forms**, but not fully. After conversion from E-R Model to Relational Model, we need to check correctness using the normalization rules. When we have only records (*i.e.*, data) available, we need to store them in the respective Tables by identifying fields, constraints, format, etc. After that, the Tables should be converted into normal forms. In this chapter, we will learn how to convert Tables into normal forms when only **records** (data) are given.

5.2 | NEED FOR NORMALIZATION

Imagine a Table which contains more than 150 fields and about ten lakhs records. It is very difficult to maintain such a big volume of records, especially when it contains so many fields.

Moreover, many of the records will contain **redundant** (duplicate/repetitive) data in this case. We need to take combination of many fields to identify each record uniquely. To do this, none of the fields may contain null values which are not possible always.

To reduce/remove these problems, we need to normalize a Table by decomposing it into many Tables. Care should be taken, so that no information is lost from the Table while we decompose it. We would get back the original information when we merge the decomposed Table again.

Normalization is a process of simplifying a complex Table into multiple Tables by decomposing it. To normalize a Table, some rules should be followed. There are mainly five normal forms. Before starting the normalization process, we need to understand some kinds of dependencies, such as functional dependency, full functional dependency, transitive dependency and multi-valued dependency.

The concept of normalization was propounded by E. F. Codd. To normalize Tables, first, we need to understand some types of dependencies which exist between the fields in a Table.

5.3 | TYPES OF DEPENDENCIES

There are many types of dependencies exist between the fields of a Table. These are as follows:

- Functional dependency
- Full-functional dependency
- Partial dependency
- Transitive dependency
- Multi-valued dependency
- Join dependency

Functional Dependency: If field1 and field2 are two attributes of a Table, then field1 is said to be functionally dependent on field2, if there exists one precise (unique) value of field1 for the corresponding value of field2. To explain, field1 is functionally dependent on fields 2, if and only if each value of field2 is associated with the precise value of field1. Symbolically, it is represented as follows:

$$\text{field2} \longrightarrow \text{field1}$$

We can read this **functional dependency** as either 'field 1 functionally dependent on field 2' or 'filed 2 functionally determines field1'

Here, for each value of field 2 in each tuple (record), we will get one precise value of field1. For an example, in Table event, if we have fields, such as *eventID*, eventname, startdate, enddate, duration, etc., then each value of *eventID* can be associated with one value of startdate. The startdate may be same for two or more different *eventID*, but each *eventID* is unique, and using *eventID* we can access precise startdate. If we change value of *eventID*, then the associated value of startdate will also be changed.

It can be simply explained as, for two different records (tuples), if there are two identical values of the *eventID* field, then there must be the same values of the startdate field, but the converse is not true (*i.e.*, for two different records, if two values of startdate are same, then it is not necessary that the corresponding values of *eventID* must be same). Consider the example as given in Table 5.1.

Table 5.1 | Event Table Showing Functional Dependency Between Fields *EventID* and Start Date

Event_ID	Event_Name	Start_Date	End_Date	Duration (in days)
1	Solo singing	2-May-2013	5-May-2013	4
2	Solo dance	2-May-2013	7-May-2013	6
3	Debate	3-May-2013	2-May-2013	1
4	Skit	2-May-2013	7-May-2013	6
5	Elocution	3-May-2013	5-May-2013	3

Table 5.2 | An Attendance Table

Date	Lecture No.	Class_ID	Std_No.	Subject_ID	Attendance
1-Apr-2013	1	FY Div-I	23	APCL	Present
1-Apr-2013	1	FY Div-II	150	DHTML	Present
1-Apr-2013	1	FY Div-I	57	APCL	Absent
1-Apr-2013	1	SY Div-I	62	OOMUL	Present
1-Apr-2013	2	FY Div-I	57	BM	Present
2-Apr-2013	1	FY Div-I	23	IMUD	Present
2-Apr-2013	1	FY Div-II	150	DHTML	Present
2-Apr-2013	1	FY Div-I	57	IMUD	Absent
2-Apr-2013	1	SY Div-I	62	SAD	Present
2-Apr-2013	2	FY Div-I	57	PM	Present

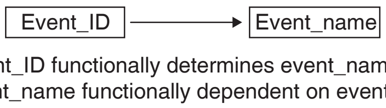

Event_ID
Event_name
Event_ID functionally determines event_name, or
event_name functionally dependent on event_ID

FIGURE 5.1 | Functional dependency.

Table 5.1 represents the details of various events. The events which have *eventID* values 3 and 5, have the same values in the field startdate. It means, two different events may start on the same dates, but it does not mean that if start dates are same, the events are also same.

Pictorially, the functional dependency can be denoted as follows:

- In functional dependency, the attribute/field which is on the left side of the arrow (*i.e.*, *eventID*) is known as **determinant**, and the attribute/field which is on the right side of the arrow, is known as **dependent**. The **determinant** can be defined as the field or combination of fields on which some other field(s) depends, is known as determinant. The dependent can be defined as the field(s) which is determined by some other field(s), is known as dependent.
- The field may be **dependent** on the combination of two or more fields. For an example, in Table 5.2, the field attendance depends on the combination of fields date, lectureno, class ID, stdno and subject ID.

Table 5.3 | Booking Table

Member_ID	Member_Name	Amenity_ID	Amenity_Desc	Booking_Date	Booking_Status
Mem003	S. R. Desai	Conf 01	Conference hall	6-May-2013	Confirm
Mem026	T. S. Pathak	Sem 03	Seminar room	3-June-2013	Not confirm
Mem123	N. C. Vora	Conf 01	Conference hall	21-May-2013	Confirm
Mem456	J. N. Patel	Aud 02	Auditorium	6-May-2013	Confirm
Mem122	S. P. Sabugola	Sem 11	Seminar room	6-June-2013	Not confirm

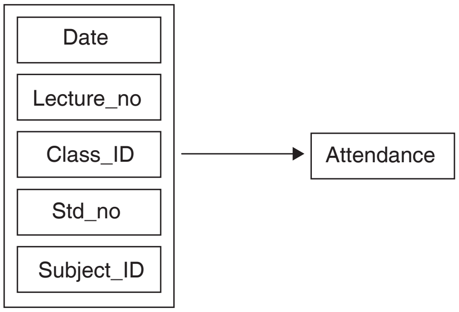


FIGURE 5.2 | Functional dependency where determinant is a combination of fields.

- From the data, as given in Table 5.2, we can say that attendance of students functionally depends on the combination of fields date, lectureno, class ID, stdno and subject ID. Because none of the fields, alone, determines attendance of the students. Only for the combination of the given five fields, we get the correct attendance of the students. Pictorially, it can be denoted as shown in Figure 5.2. Here, Attendance is a dependent and combination of fields (date, lectureno, class ID, stdno, subject ID) is determinant.

In the example as given in Figure 5.2, the composite determinant contains five **components** namely, date, lectureno, class ID, std no and subject ID.

Full Functional Dependency: When determinant contains only a single field, the dependent is fully dependent on that determinant. This dependency is called, **'full function dependency'**. But when determinant is a combination of more than one field, it is possible that dependent is not fully dependent on the combination. It may also be dependent on any component of that determinant. It means if we change any component, the corresponding value of the dependent will also be changed. In the example as given in Figure 5.2, the attendance of students fully functionally depends on the whole combination of (date, lectureno, class ID, stdno and subject ID). There is no individual component or combination of components on which attendance depends other than the whole combination. Therefore, the example as shown in Figure 5.2 shows full functional dependence between dependent attendance and determinant (date, lecture no, class ID, stdno, subject ID).

Now, consider Table 5.3. It contains data related to amenity booking of a professional society.

In Table 5.2, the following functional dependencies exist between the fields:

$$\begin{aligned}
&(\text{member\_ID, amenity\_ID}) \longrightarrow \text{member\_name}\\
&(\text{member\_ID, amenity\_ID}) \longrightarrow \text{amenity\_desc}\\
&(\text{member\_ID, amenity\_ID}) \longrightarrow \text{booking\_date}\\
&(\text{member\_ID, amenity\_ID}) \longrightarrow \text{booking\_status}
\end{aligned}$$

Here, the determinant is a combination of fields, member_ID and amenity_ID. member_ID and amenity_ID are two different components. The above four dependencies are functional dependencies, but we need to check if they are full functional dependencies or not.

Consider the case ‘(member_ID, amenity_ID) determines booking_date’. Here, booking_date is fully functionally dependent on the combination (member_ID, amenity_ID) because none of the component of (member_ID, amenity_ID) determines booking_date. The booking_ID field will have some values when a member books any amenity. Likewise, the dependency ‘(member_ID, amenity_ID) determines booking_status’ is a full functional dependency. **Figure 5.3** shows the pictorial representation of full function dependence of these two cases.

Now, consider the case ‘(member_ID, amenity_ID) determines member_name’. In this case, if we change the component member_ID, we will get a precise value of member_name. It means, member_name is not fully functionally dependent on the combination (member_ID, amenity_ID), but also depend on member_ID alone. Therefore, this dependency is known as **partial dependency**. Here, member_name is functionally dependent on the combination (member_ID, amenity_ID), and also partially dependent on member_ID.

Similarly, in the dependency ‘(member_ID, amenity_ID) determines amenity_desc’, we get precise value of amenity_desc for any value of amenity_ID. Therefore, amenity_desc is functionally dependent on the combination (member_ID, amenity_ID), and also partially dependent on amenity_ID. We can define partial dependency as follows:

Partial Dependency: When determinant (*i.e.*, on which some field depends) is a combination of more than one fields, determines any field; if any component(s) of determinant also determines that field, the field is said to be partially dependent on determinant. For an example, ‘(member_ID, amenity_ID) partially determines amenity_desc’ because amenity_ID also determines amenity_desc.‘(member_ID, amenity_ID) partially determines member_name’ because member_ID also determines member_name. **Figure 5.4** shows these partial dependencies.

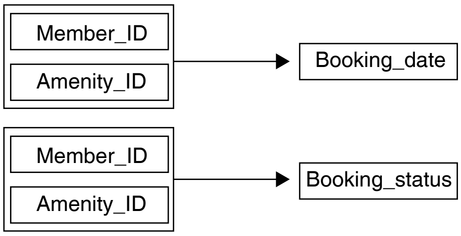


FIGURE 5.3 | Full Functional Dependencies: (member_ID, amenity_ID) Fully Functionally Determines booking_date and (member_ID, amenity_ID) Fully Functionally Determines Booking_status.

Table 5.4 | City_Team Table

City_Code	City_Name	Team_ID	Team_Name
KK	Kolkata	KKR	Kolkata Knight Riders
CH	Chennai	CSR	Chennai Super Kings
BG	Bangalore	RCB	Royal Challengers Bangalore
MH	Mohali	KXIP	Kings XI Punjab
DL	Delhi	DD	Delhi Daredevils
HD	Hyderabad	HDC	Hyderabad Deccan Chargers
JP	Jaipur	RR	Rajasthan Royals
MB	Mumbai	MI	Mumbai Indian
PN	Pune	PWI	Pune Warriors India

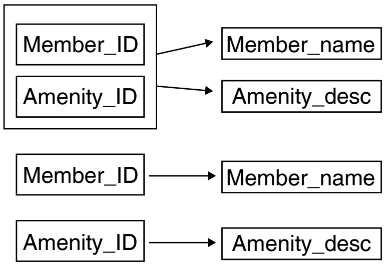


FIGURE 5.4 | Partial dependencies: (member_ID, amenity_ID) Partially determines member_name; and (member_ID, amenity_ID) Partially determines amenity_desc.

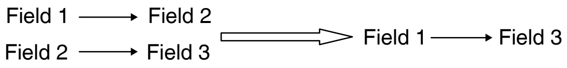


FIGURE 5.5 | Transitive dependency between field1 and field3.

Transitive Dependency: In a Table, consider that there exists functional dependency between its fields; field 1 and field 2, where field1 is a determinant and field 2 is a dependent. If any other field, namely field 3 depends on the determinant (field 1), as well as on the dependent (field 2) the type of dependency between field 1 and field 3 is said to be **transitive dependency**. It is shown in **Figure 5.5**.

For an example, in Table 5.4, the field city_code determines team_ID, and City_code determines team_name, but also team_ID determines team_name. Hence, there exists transitive dependency between the city_code and team_name fields.

Pictorially, it can be represented as shown in **Figure 5.6**.

Multi-valued Dependency: It is explained in Section 5.8.

Join Dependency: It is explained in Section 5.9.

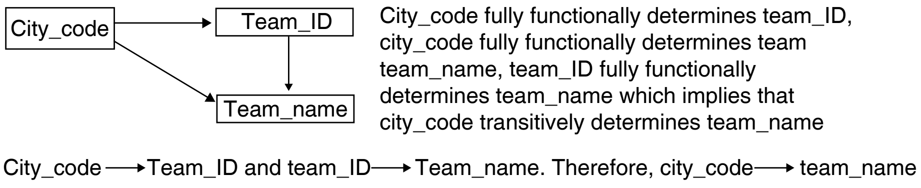

City_code
Team_ID
Team_name
City_code fully functionally determines team_ID, city_code fully functionally determines team team_name, team_ID fully functionally determines team_name which implies that city_code transitively determines team_name
City_code ⟶ Team_ID and team_ID ⟶ Team_name. Therefore, city_code ⟶ team_name

FIGURE 5.6 | Transitive Dependency between City_Code and Team_Name.

5.4 | FIRST NORMAL FORM

When we are given some problem description, then first we need to identify the fields along with the values from the Table. In case, when Table has been given directly, then we have to check whether it is in first normal form or not. Consider Table 5.5 which describes data related to IPL cricket matches.

First Normal Form/1NF Definition: The relation (Table) is said to be in **first normal form**, if and only if, all the fields of the Table contain atomic value in a record.

In other words, the Table is said to be in **1NF**, if and only if, none of the attributes is a composite attribute. In the given Table, the attributes (*i.e.*, player_name and coach_name) are composite attributes because player_name can be further divided into player_firstname and player_lastname. Likewise, coach_name can be further divided into coach_fname and coach_lname. All other attributes are simple attributes (*i.e.*, values in the fields are atomic). As the Table 5.5 contains two composite attributes, so it is not in 1NF.

To convert the IPL Table in 1NF, player_name should be divided into two fields, player_fname and player_lname. Similarly, coach_name should be divided into two fields, coach_fname and coach_lname. After converting, composite attributes into simple attributes,Table 5.6 does not contain any composite attribute. Therefore, it is in first normal form.

After converting the Table into 1NF, it should be converted into 2NF.

5.5 | SECOND NORMAL FORM

Second Normal Form/2NF Definition: The relation (Table) is said to be in **second normal form**, if and only if:

1. It is in 1NF.
2. All the attributes are fully functionally dependent on the primary key.
   or
3. None of the attribute is partially dependent on the primary key.

To check whether the Table is in **2NF** or not, first identify the primary key of a Table, and second, list out all the dependencies.

In Table 5.6, the primary key is a combination of fields (team_ID, player_ID) as it identifies each record of a Table uniquely.

Table 5.5 | IPL

City_ Code	City_ Name	Team_ ID	Team_ Name	Player_ ID	Player_ Name	Citizen_ Country ID	Citizen_ Country Name	Role	Owner_ ID	Owner_Name	Coach_ ID	Coach_ Name	Web
KK	Kolkata	KKR	Kolkata Knight Riders	P1	Gautam Gambhir	IND	India	Batsman	A1	Knight Riders Sports Private Ltd	C1	Trevor Bayliss	www.kkr.com
KK	Kolkata	KKR	Kolkata Knight Riders	P2	Brett Lee	AUS	Australia	Bowler	A1	Knight Riders Sports Private Ltd	C1	Trevor Bayliss	www.kkr.com
KK	Kolkata	KKR	Kolkata Knight Riders	P3	Brad Haddin	AUS	Australia	Wicket keeper	A1	Knight Riders Sports Private Ltd	C1	Trevor Bayliss	www.kkr.com
KK	Kolkata	KKR	Kolkata Knight Riders	P4	Eoin Morgan	ENG	England	Batsman	A1	Knight Riders Sports Private Ltd	C1	Trevor Bayliss	www.kkr.com
CH	Chennai	CSR	Chennai Super Kings	P5	Mahendra Sinh Dhoni	IND	India	Wicket keeper	A2	The India Cements Ltd	C2	Stephen Fleming	www.chennai superkings.com
CH	Chennai	CSR	Chennai Super Kings	P6	Suresh Raina	IND	India	Batsman	A2	The India Cements Ltd	C2	Stephen Fleming	www.chennai superkings.com
CH	Chennai	CSR	Chennai Super Kings	P7	Ravindra Jadeja	IND	India	All-rounder	A2	The India Cements Ltd	C2	Stephen Fleming	www.chennai superkings.com
MB	Mumbai	MI	Mumbai Indian	P8	Sachin Tendulkar	IND	India	Batsman	A3	IndiaWin Sports Pvt Ltd	C3	John Wright	www.mumbai indians.com
MB	Mumbai	MI	Mumbai Indian	P9	Rohit Sharma	IND	India	Batsman	A3	IndiaWin Sports Pvt Ltd	C3	John Wright	www.mumbai indians.com
MB	Mumbai	MI	Mumbai Indian	P10	Lasith Malinga	SL	Sri Lanka	Bowler	A3	IndiaWin Sports Pvt Ltd	C3	John Wright	www.mumbai-indians.com

Table 5.6 | IPL in First Normal Form

City_ Code	City_ Name	Team_ ID	Team_ Name	Player_ ID	Player_ fname	Player_ lname	Citizen_ Country ID	Citizen_ Country Name	Role	Owner_ ID	Owner_Name	Coach_ ID	Coach_ fname	Coach_ lname	Web
KK	Kolkata	KKR	Kolkata Knight Riders	P1	Gautam	Gambhir	IND	India	Batsman	A1	Knight Riders Sports Private Ltd	C1	Trevor	Bayliss	www.kkr.com
KK	Kolkata	KKR	Kolkata Knight Riders	P2	Brett	Lee	AUS	Australia	Bowler	A1	Knight Riders Sports Private Ltd	C1	Trevor	Bayliss	www.kkr.com
KK	Kolkata	KKR	Kolkata Knight Riders	P3	Brad	Haddin	AUS	Australia	Wicket keeper	A1	Knight Riders Sports Private Ltd	C1	Trevor	Bayliss	www.kkr.com
KK	Kolkata	KKR	Kolkata Knight Riders	P4	Eoin	Morgan	ENG	England	Batsman	A1	Knight Riders Sports Private Ltd	C1	Trevor	Bayliss	www.kkr.com
CH	Chennai	CSR	Chennai Super Kings	P3	Mahen-drasinh	Dhoni	IND	India	Wicket keeper	A2	The India Cements Ltd	C2	Stephen	Fleming	www.chennaisuperkings.com
CH	Chennai	CSR	Chennai Super Kings	P5	Suresh	Raina	IND	India	Batsman	A2	The India Cements Ltd	C2	Stephen	Fleming	www.chennaisuperkings.com
CH	Chennai	CSR	Chennai Super Kings	P7	Ravindra	Jadeja	IND	India	All-rounder	A2	The India Cements Ltd	C2	Stephen	Fleming	www.chennaisuperkings.com
MB	Mumbai	MI	Mumbai Indian	P1	Sachin	Tendulkar	IND	India	Batsman	A3	IndiaWin Sports Pvt Ltd	C3	John	Wright	www.mumbaiindians.com
MB	Mumbai	MI	Mumbai Indian	P2	Rohit	Sharma	IND	India	Batsman	A3	IndiaWin Sports Pvt Ltd	C3	John	Wright	www.mumbaiindians.com
MB	Mumbai	MI	Mumbai Indian	P10	Lasith	Malinga	SL	Sri Lanka	Bowler	A3	IndiaWin Sports Pvt Ltd	C3	John	Wright	www.mumbaiindians.com

Table 5.6 contains the following functional dependencies:

$$
\begin{aligned}
(\text{team\_ID, player\_ID}) &\longrightarrow \text{city\_code}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{city\_name}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{team\_name}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{player\_fname}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{player\_lname}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{citizen\_countryID}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{citizen\_countryname}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{role}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{owner\_ID}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{owner\_name}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{coach\_ID}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{coach\_fname}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{coach\_lname}\\
(\text{team\_ID, player\_ID}) &\longrightarrow \text{web}
\end{aligned}
$$

The above dependencies can also be written as follows:

$$
\begin{aligned}
(\text{team\_ID, player\_ID}) \longrightarrow\ & \text{city\_code, city\_name, team\_name,}\\
& \text{player\_fname, player\_lname,}\\
& \text{citizen\_countryID,}\\
& \text{citizen\_countryname, role, owner\_ID,}\\
& \text{owner\_name, coach\_ID,}\\
& \text{coach\_fname, coach\_lname, web}
\end{aligned}
$$

For the above dependencies, a dependency diagram can be shown as given in **Figure 5.7**.

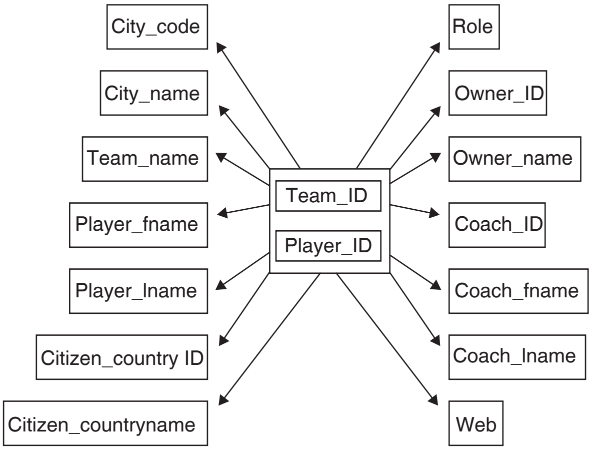


FIGURE 5.7 | Functional dependencies of IPL Table as Given in Table 5.6.

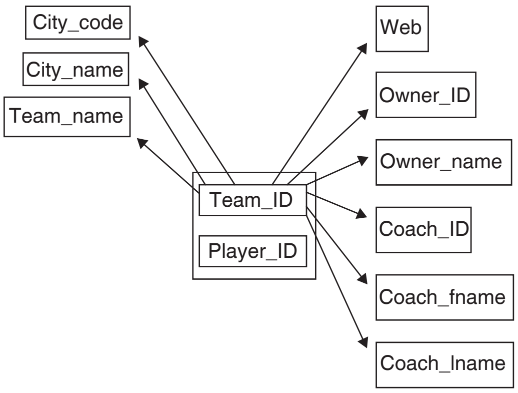


FIGURE 5.8 | Partial dependencies of IPL Table as shown in Table 5.6.

There exist the following partial dependencies in Table 5.6.

$$\text{team\_ID} \longrightarrow \text{city\_code}$$
$$\text{team\_ID} \longrightarrow \text{city\_name}$$
$$\text{team\_ID} \longrightarrow \text{team\_name}$$
$$\text{team\_ID} \longrightarrow \text{owner\_ID}$$
$$\text{team\_ID} \longrightarrow \text{owner\_name}$$
$$\text{team\_ID} \longrightarrow \text{coach\_ID}$$
$$\text{team\_ID} \longrightarrow \text{coach\_fname}$$
$$\text{team\_ID} \longrightarrow \text{coach\_lname}$$
$$\text{team\_ID} \longrightarrow \text{web}$$

The above partial dependencies can be denoted as given in **Figure 5.8**.

To remove partial dependencies from the Table, remove all the fields which are partially dependent on a component of primary key (do not remove the component on which other fields are partially dependent). Add all these removed fields along with the field on which they depend into a new Table and give that Table some meaningful name. In other words, add all the fields which are there on the left and right side of the partial dependency into one new Table. The field which is determinant (on left side of arrow) in the partial dependency will become a primary key of the new Table.

For Table 5.6, we have listed the following partial dependencies:

$$\text{team\_ID} \longrightarrow \begin{array}{l}\text{city\_code, city\_name, team\_name, owner\_ID,}\\ \text{owner\_name, coach\_ID, coach\_fname,}\\ \text{coach\_lname, web}\end{array}$$

Therefore, we are required to create a new Table, say TEAM.

TEAM Table will have attributes team_ID, city_code, city_name, team_name, owner_ID, owner_name, coach_ID, coach_fname, coach_lname and web. team_ID (which is on the left side of the arrow in partial dependency) will become primary key of Table TEAM. However, team_ID will remain as it is in the original IPL Table.

Now, after decomposition of Table IPL, we will have two Tables namely, IPL and TEAM as follows:

> **IPL** (team_ID, player_ID, player_fname, player_lname, role, citizen_countryID, citizen_countryname)
>
> **TEAM** (team_ID, city_code, city_name, team_name, owner_ID, owner_name, coach_ID, coach_fname, coach_lname, web)

The fields which are underlined are the primary keys of the respective Tables. Table 5.7 shows IPL Table after decomposition, and Table 5.8 shows TEAM Table.

The duplicate rows will be deleted from all the Tables, and hence the **redundancy** (duplication of data) will be reduced. After the decomposition, again we need to check whether there exists any partial dependency in any of the Table. If yes, then it should be removed by further decomposition. This process must be continued until all the Tables fall in 2NF.

After decomposition, both the Tables IPL and TEAM are in 2NF. TEAM has only one field as a primary key. So, there is no question of partial dependency (Remember the rule that partial dependency may exists if primary key is a composite key) in this Table.

In the IPL Table primary key is a composite key which contains combination of team_ID and player_ID. But the combination fully functionally determines all the other fields. Therefore, also in this Table, no partial dependencies exist.

Hence IPL and TEAM both the Tables are in 2NF.

Table 5.7 | IPL Which is in 2NF

Team_ID	Player_ID	Player_fname	Player_lname	Citizen_Country ID	Citizen_Country Name	Role
KKR	P1	Gautam	Gambhir	IND	India	Batsman
KKR	P2	Brett	Lee	AUS	Australia	Bowler
KKR	P3	Brad	Haddin	AUS	Australia	Wicket keeper
KKR	P4	Eoin	Morgan	ENG	England	Batsman
CSR	P3	Mahendrasinh	Dhoni	IND	India	Wicket keeper
CSR	P5	Suresh	Raina	IND	India	Batsman
CSR	P7	Ravindra	Jadeja	IND	India	All-rounder
MI	P1	Sachin	Tendulkar	IND	India	Batsman
MI	P2	Rohit	Sharma	IND	India	Batsman
MI	P10	Lasith	Malinga	SL	Sri Lanka	Bowler

Table 5.8 | TEAM Which is in 2NF

City_Code	City_Name	Team_ID	Team_Name	Owner_ID	Owner_Name	Coach_ID	Coach_fname	Coach_lname	Web
KK	Kolkata	KKR	Kolkata Knight Riders	A1	Knight Riders Sports Private Ltd	C1	Trevor	Bayliss	www.kkr.com
CH	Chennai	CSR	Chennai Super Kings	A2	The India Cements Ltd	C2	Stephen	Fleming	www.chennaisuperkings.com
MB	Mumbai	MI	Mumbai Indian	A3	IndiaWin Sports Pvt Ltd	C3	John	Wright	www.mumbaiindians.com

5.6 | THIRD NORMAL FORM

Third Normal Form/3NF Definition: The relation (Table) is said to be in **third normal form**, if and only if:

1. It is in 2NF.
2. All the attributes are non-transitively dependent on the primary key. Or,
3. None of the attribute is transitively dependent on the primary key.

To check whether the Table is in 3NF or not, first, list all the transitive dependencies. The following transitive dependencies exists in the IPL Table:

(team_ID, player_ID) $\longrightarrow$ citizen_countryID
citizen_countryID $\longrightarrow$ citizen_countryname

$$\Downarrow$$

(team_ID, player_ID) $\longrightarrow$ citizen_countryname

The transitive dependency of the IPL Table is shown in Figure 5.9.

The following transitive dependencies exists in the TEAM Table:

1. team_ID $\longrightarrow$ city_code, city_code $\longrightarrow$ city_name
   Which implies team_ID $\longrightarrow$ city_name
2. team_ID $\longrightarrow$ owner_ID, owner_ID $\longrightarrow$ owner_name
   Which implies team_ID $\longrightarrow$ owner_name
3. team_ID $\longrightarrow$ coach_ID, coach_ID $\longrightarrow$ coach_fname
   Which implies team_ID $\longrightarrow$ coach_fname
4. team_ID $\longrightarrow$ coach_ID, coach_ID $\longrightarrow$ coach_lname
   Which implies team_ID $\longrightarrow$ coach_lname

The transitive dependency of the TEAM Table is shown in Figure 5.10.

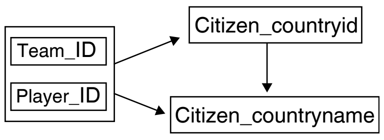


FIGURE 5.9 | Transitive dependency of the IPL Table.

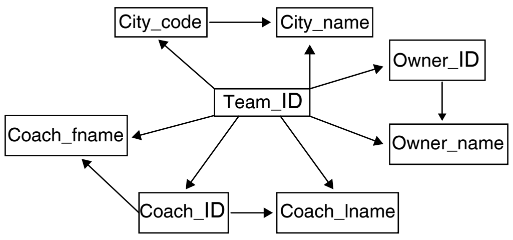


FIGURE 5.10 | Transitive dependency of the TEAM Table.

To remove transitive dependency from the Table, do the following:

1. Decompose the Table and create a new Table which will contain the field because of which transitive dependency exists (*i.e.,* the field other than primary key which also determines the third field). Do not remove that intermediate field from the original Table.
2. Remove the field which transitively depends on the primary key and add that field(s) into the new Table.

Consider the transitive dependency of the IPL Table. The field, citizen_countryID, is the field which also determines citizen_countryname field besides the primary key (teamid, player_ID). Therefore, the IPL Table will be decomposed as following two Tables. Citizen_countryID will be added in a new Table, say CITIZEN_COUNTRY along with the field citizen_countryname. But citizen_countryID will remain as it is in the IPL Table also. In a new Table, CITIZEN_COUNTRY, citizen_countryID will become a primary key.

CITIZEN_COUNTRY (citizen_countryID, citizen_country name)
IPL (team_ID, player_ID, player_fname, player_lname, role, citizen_countryID)

Similarly, from TEAM Table, when we remove transitive dependencies, it will be decomposed as TEAM, CITY, OWNER and COACH Tables.

TEAM (team_ID, city_code, team_name, owner_ID, coach_ID, web)
CITY (city_code, city_name)
OWNER (owner_ID, owner_name)
COACH (coach_ID, coach_fname, coach_lname)

After removal of transitive dependencies, now we will have 6 IPL Tables, these are as: CITIZEN_COUNTRY, TEAM, CITY, OWNER and COACH. For all these Tables, again, check whether any one of them contains transitive dependencies? If yes, then remove it by applying the rules given above until all the transitive dependencies are removed.

There are no transitive dependencies exists in any of the six Tables, IPL, CITIZEN_COUNTRY, TEAM, CITY, OWNER and COACH. Therefore, now they are in **3NF**. Tables 5.9–5.14 shows the IPL, CITIZEN_COUNTRY, TEAM, CITY, OWNER and COACH Tables which are all in 3NF. They will not contain any redundant records.

Table 5.9 | IPL Which is in 3NF

Team_ID	Player_ID	Player_fname	Player_lname	Citizen_Country ID	Role
KKR	P1	Gautam	Gambhir	IND	Batsman
KKR	P2	Brett	Lee	AUS	Bowler
KKR	P3	Brad	Haddin	AUS	Wicket keeper
KKR	P4	Eoin	Morgan	ENG	Batsman
CSR	P3	Mahendrasinh	Dhoni	IND	Wicket keeper
CSR	P5	Suresh	Raina	IND	Batsman
CSR	P7	Ravindra	Jadeja	IND	All-rounder
MI	P1	Sachin	Tendulkar	IND	Batsman
MI	P2	Rohit	Sharma	IND	Batsman
MI	P10	Lasith	Malinga	SL	Bowler

Table 5.10 | CITIZEN_COUNTRY Which is in 3NF

Citizen_Country ID	Citizen_Country Name
IND	India
AUS	Australia
ENG	England
SL	Sri Lanka

Table 5.11 | TEAM Which is in 3NF

City_Code	Team_ID	Team_Name	Owner_ID	Coach_ID	Web
KK	KKR	Kolkata Knight Riders	A1	C1	www.kkr.com
CH	CSR	Chennai Super Kings	A2	C2	www.chennaisuperkings .com
MB	MI	Mumbai Indian	A3	C3	www.mumbaiindians .com

Table 5.12 | CITY Which is in 3NF

City_Code	City_Name
KK	Kolkata
CH	Chennai
MB	Mumbai

Table 5.13 | OWNER Which is in 3NF

Owner_ID	Owner_Name
A1	Knight Riders Sports Private Ltd
A2	The India Cements Ltd
A3	India Win Sports Pvt Ltd

Table 5.14 | COACH Which is in 3NF

Coach_ID	Coach_fname	Coach_lname
C1	Trevor	Bayliss
C2	Stephen	Fleming
C3	John	Wright

5.7 | BOYCE–CODD NORMAL FORM

Boyce–Codd Normal Form (BCNF) is a special type of 3NF which was propounded by R. F. Boyce and E. F. Codd.

BCNF Definition: The relation (Table) is said to be in BCNF, if and only if:

1. It is in 3NF.
2. Every determinant (field/combination of fields) of the Table should be a candidate key.

Consider the relation IPL as given in the Table 5.15; where there are two determinants:

1. (team_ID,player_ID)
2. (captain_name, player_ID)

Both the determinants are candidate keys. All the other attributes are fully functionally dependent on both the candidate keys, *i.e,* there does not exist any partial dependency. Table 5.15 is therefore, in 2NF. Also, there does not exist any transitive dependency in the Table, and hence the IPL Table is in 3NF. But, also there exists two more determinants team _ID and captain_ID (as shown below) which determines each other.

$$\text{team\_ID} \longrightarrow \text{captain\_name and}$$
$$\text{captain\_name} \longrightarrow \text{team\_ID}$$

team_ID and captain_name are determinants, but niether team_ID nor captain_name is a candidate key (as both contains duplicate values). Therefore, Table 5.15 is not in BCNF. To convert it into BCNF, we should keep both the determinants, team_ID and captain_name, in one Table and from the original Table, remove any of the field, either team_ID, or captain_name. Table 5.15 should be decomposed as follows:

IPL (team_ID, player_ID, player_fname, player_lname, citizen_countryID, role)
CAPTAIN (captain_name, team_ID)

OR

IPL (captain_name, player_ID, player_fname, player_lname, citizen_countryid, role)
CAPTAIN (team_ID, captain_name)

Now, both the above decompositions are in BCNF, because every determinant of both the Tables in both the decomposition is a candidate key. In the first decomposition, the IPL Table has only one determinant (team_ID, player_ID) which is a candidate key and the Captain Table has two determinants, team_ID and captain_ID; and both are candidate keys. Similarly, for

Table 5.15 | Captain is in 3NF, but Not in BCNF

Team_ID	Player_ID	Captain_Name	Player_fname	Player_ lname	Citizen_ Country ID	Role
KKR	P1	G. Gambhir	Gautam	Gambhir	IND	Batsman
KKR	P2	G. Gambhir	Brett	Lee	AUS	Bowler
KKR	P3	G. Gambhir	Brad	Haddin	AUS	Wicket keeper
KKR	P4	G. Gambhir	Eoin	Morgan	ENG	Batsman
CSR	P3	M. S. Dhoni	Mahendrasinh	Dhoni	IND	Wicket keeper
CSR	P5	M. S. Dhoni	Suresh	Raina	IND	Batsman
CSR	P7	M. S. Dhoni	Ravindra	Jadeja	IND	All-rounder
MI	P1	R. Sharma	Sachin	Tendulkar	IND	Batsman
MI	P2	R. Sharma	Rohit	Sharma	IND	Batsman
MI	P10	R. Sharma	Lasith	Malinga	SL	Bowler

the second decomposition, the IPL Table has only one determinant (captain_name, player_ID) which is a candidate key and the Captain Table has two determinants, team_ID and captain_ID; and both are candidate keys.

5.8 | MULTI-VALUED DEPENDENCY

For the given relation R, consider field 1, field 2 and field 3 as attributes of R. Then, for each record of R, there exist **multi-valued dependency** between field 1 and field 2 known as field 1 multi-determines field 2 (*i.e.,* field 2 is multi-dependent on field1), if and only if, the set of field 2 values depends only on the field 1 value and is independent of field 3 values for matching pair of (field 1, field 3). In other words, we can say that in relation R, for fields (field 1, field 2, field 3), the multi-valued dependency field 1 multi-determines field 2 exists, if and only if, the multi-valued dependency field 1 multi-determines field 3 also exists. The concept of multi-valued dependency was proposed by Ronald Fagin. In notations, multi-valued dependency is denoted as:

$$\text{field 1} \twoheadrightarrow \text{field 2}$$
$$\text{field 1} \twoheadrightarrow \text{field 3}$$

Definition of multi-valued dependency: Consider a relation with A, B and C—subsets of attributes of a relation. Then, B is said to be multi-dependent on A, if and only if, in each record (tuple) of a relation, the set of B values which match with a given (A, C) value pair depends only on the value of A and is completely independent of value of C.

Consider the as data given in Table 5.16 (MOBILE), which represents company-wise mobile types and their features.

Mobile phone companies offer many types of handsets that comes fitted with multiple features. In the MOBILE Table, there exists the following multi-valued dependencies:

- Model_type and Features are multi-dependent on Company_name, but Model_type and Features are ‘ independent’, or ‘orthogonal’ of each other.
- Multi-valued dependency is a special form of functional dependency. In a functional dependency, the dependent attribute is a single-value attribute: while in multi-valued dependency, the dependent attribute is a multi-valued attribute.
- Ronald Fagin propounded a fourth normal form on the basis of multi-valued dependency.

Table 5.16 | Mobile Table with Multiple Values of Model Type and Features

Company_Name	Model_Type	Features
SAMSUNG	• Smart phone • Tablet • Touch Phones	• Bluetooth • Android OS • 7 MP Camera
NOKIA	• Smart phone • Multimedia Phone • Dual SIM	• Bluetooth • Windows 8.0 OS • 4GB memory card

$$\text{Company\_name} \twoheadrightarrow \text{Model\_type}$$
$$\text{Company\_name} \twoheadrightarrow \text{Features}$$

Fourth normal form definition: The relation (Table) is said to be in **4NF**, if and only if:

1. It is in BCNF.
2. For the existence of nontrivial (**nontrivial** means the attribute on the right side of the arrow is not a subset of left hand side attribute) multi-valued dependency between attributes A→→B in a relation, all other attributes of relation should also functionally dependent on A.

Or

2. Every nontrivial MVD in a relation is implied by the candidate key(s) of relation.

Or

4. Remove multi-valued dependency from the relation which are not also functional dependencies.

The MOBILE Table can be converted as shown in Table 5.17, which contains combination of all the three attributes as primary key, but there are some problems, such as, if we want to add new model_type for the company SAMSUNG, it is required to add three new records one for each feature. This problem occurs because model_type and features are independent.

To resolve this problem, the MOBILE Table should be decomposed into two Tables as follows:

MOB_TYPE (company_name, model_type)
MOB_FEAT (company_name, features)

Now, the Tables MOB_TYPE and MOB_FEAT are in 4NF, as they do not contain any multi-valued dependency.

Table 5.17 | MOBILE Table

Company_Name	Model_Type	Features
SAMSUNG	Smart phone	Bluetooth
SAMSUNG	Smart phone	Android OS
SAMSUNG	Smart phone	7 MP Camera
SAMSUNG	Tablet	Bluetooth
SAMSUNG	Tablet	Android OS
SAMSUNG	Tablet	7 MP Camera
SAMSUNG	Touch Phones	Bluetooth
SAMSUNG	Touch Phones	Android OS
SAMSUNG	Touch Phones	7 MP Camera
NOKIA	Smart phone	Bluetooth
NOKIA	Smart phone	Windows 8.0 OS
NOKIA	Smart phone	4GB memory card
NOKIA	Multimedia Phone	Bluetooth
NOKIA	Multimedia Phone	Windows 8.0 OS
NOKIA	Multimedia Phone	4GB memory card
NOKIA	Dual SIM	Bluetooth
NOKIA	Dual SIM	Windows 8.0 OS
NOKIA	Dual SIM	4GB memory card

5.9 | JOIN DEPENDENCY

Join dependency is a generalization of multi-valued dependency.

Definition of Join dependency: Consider any relation with projections P_1, P_2, …, P_n. There exists join dependency (denoted as $*\{P_1, P_2, \ldots, P_n\}$), if and only if, value of every record(tuple) of a relation is the join of its projections on P1, P2, …, P_n. From the Fagin's definition of multi-valued dependency, we can say that the **join dependency** in a relation with attributes A, B and C; $*\{AB, AC\}$ holds, if and only if, it satisfies the multi-valued dependencies $A \rightarrow\rightarrow B$ and $A \rightarrow\rightarrow C$.

Fifth Normal Form Definition: The relation(Table) is said to be in **5NF**, if and only if:

1. It is in 4NF.
2. Every nontrivial join dependency in a relation is implied by the candidate key(s) of relation.

   Or

   Each projection (decomposition) of a relation should contain candidate key(s) of a relation.

   Or

   Each projection of the relation should be based only on the candidate key(s).

The **fifth normal form** is also known as **projection-join normal form (PJNF)**. Consider the relation (Table) as given in Table 5.18. It describes which faculty takes which subject in which class.

Table 5.18 | TEACHING

Faculty_ID	Subject_ID	Class_ID
SN	FOP	FY-I
SN	FOP	FY-II
SN	APCL	FY-I
SN	APCL	FY-II
SN	PCL	FY-I
SN	PCL	FY-II
HT	SC	SY-I
HT	SC	SY-II
HT	MFCS	SY-I
HT	MFCS	SY-II
HD	IHTML	FY-I
HD	IHTML	FY-II
HD	DHTML	FY-I
HD	DHTML	FY-II
HD	AJ	TY-I
HD	AJ	TY-II
KG	OOP	SY-I
KG	OOP	SY-II
KG	OOMUL	SY-I
KG	OOMUL	SY-II

Table 5.18 shows the details of the faculties teaching various subjects in different classes. It has a Candidate key which is a combination of all the three attributes. The TEACHING Table is in 4NF because it has no **nontrivial multi-valued dependency**. This Table is not in 5NF because it contains join dependency which is not implied by its candidate key (combination offaculty_ID, subject_ID and class_ID). To convert this Table into 5NF, it should be further decomposed as 3 projections, namely FAC_SUB, SUB_CLASS and CLASS_FAC. These three Tables would contain the following attributes:

FAC_SUB (faculty_ID, subject_ID)
SUB_CLASS (subject_ID, class_ID)
CLASS_FAC (class_ID, faculty_ID)

The 3 decompositions FAC_SUB, SUB_CLASS and CLASS_FAC are shown in Tables 5.19, 5.20 and 5.21 respectively.

The 3 decompositions FAC_SUB, SUB_CLASS and CLASS_FAC are in 5NF.

5.10 | LOSSLESS AND LOSSY DECOMPOSITIONS

During the normalization process, when we decompose a Table into different decompositions (projections), care should be taken so that original information should not be lost when we, again, join those decompositions.

When we decompose the Table and, if original information will not be lost, the decomposition is said to be **lossless decomposition**. Lossless decomposition is also known as **nonloss decomposition**. The concept was given by Heath.

When we decompose a Table and, if original information will be lost when we, again, join the decompositions; the type of decomposition is said to be **lossy decomposition**.

The nonloss decomposition can be achieved by preserving the functional dependencies in the decompositions.

Table 5.19 | FAC_SUB

Faculty_ID	Subject_ID
SN	FOP
SN	APCL
SN	PCL
HT	SC
HT	MFCS
HD	IHTML
HD	DHTML
HD	AJ
KG	OOP
KG	OOMUL

Table 5.20 | SUB_CLASS

Subject_ID	Class_ID
FOP	FY-I
FOP	FY-II
APCL	FY-I
APCL	FY-II
PCL	FY-I
PCL	FY-II
SC	SY-I
SC	SY-II
MFCS	SY-I
MFCS	SY-II
IHTML	FY-I
IHTML	FY-II
DHTML	FY-I
DHTML	FY-II
AJ	TY-I
AJ	TY-II
OOP	SY-I
OOP	SY-II
OOMUL	SY-I
OOMUL	SY-II

Table 5.21 | CLASS_FAC

Faculty_ID	Class_ID
SN	FY-I
SN	FY-II
HT	SY-I
HT	SY-II
HD	FY-I
HD	FY-II
HD	TY-I
HD	TY-II
KG	SY-I
KG	SY-II

Definition of Nonloss Decomposition: If R is a relation, and $P_1, P_2, \ldots, P_n$ are decompositions (Projections) of R which are nonloss decompositions if join of $P_1, P_2, \ldots, P_n$ results into original relation R. We cannot omit any of the decompositions ($P_1, P_2, \ldots, P_n$) while joining them as it will result into incorrect information.

5.11 | NORMALIZING TABLES

We have already seen the normalization of Tables and its importance in database design. Before we start learning normalization, we should know all the types of dependency in detail. The first normal form gives a Table with atomic fields. The second normal form removes partial dependencies from the Table(s). The third normal form removes transitive dependencies from the

Table(s). Advanced form of 3NF is BCNF which ensures that all the determinants are candidate keys. The fourth normal form removes multi-valued dependencies from the Table which are not functional dependencies. The fifth normal form ensures that every join dependency is implied by the candidate key of a Table. During normalization process, when we decompose the Table, all the functional dependencies should be preserved into decompositions to make it sure that the decompositions are nonloss decompositions. During the normalization process, if we found primary key is a combination of too many fields and difficult to handle, then we can use surrogate key (an artificial primary key which contains positive integer's values and mostly auto-generated) as a primary key instead of an actual primary key. Also, if there are some derived fields in the Table, it depends on the database designer whether to store its value or not. Beside five normal forms which are based on functional dependency, there are some more normal forms proposed, such as domain-key normal form (DK/NF), 'PSJU/NF or Restriction-Union normal form', sixth normal form, etc.

Domain-Key Normal Form is based on domain constraints and key constraints. Domain constraint means the constraint which is put up on the values of attributes and key constraint is a constraint which is applied on the key attributes of a Table. R. Fagin proposed this normal form. The relation is said to be in **DKNF**, if and only if, it is in 5NF and every constraint is derived automatically from the enforcement of domain constraints and key constraints.

Normal forms 1NF to 5NF are achieved by decomposing a Table into its projections (vertical subset of Table, *i.e.,* decomposition which is based on columns) and ensuring that when we will again join these decomposition, we will get back the original Table with all the information. It means that 1NF to 5NF are based on 'Projection' and 'Join' operators. Is it possible to decompose the Table by taking its horizontal subset (based on the attribute's specific value) which is called, 'split/restriction' and 'recomposition' (join the decomposition again) by doing union of those restrictions? For an example, let WORKER is a Table which contains details of workers of a factory with one field 'gender'. By applying **PSJU/NF** (projection split join union/normal form), all the male workers can be kept in one Table and the female workers in the other one. PSJU/NF may result in a poor database design.

Sixth normal form is proposed for temporal databases. The database which stores historical data is known as temporal database.

Many other normal forms are also proposed, such as **NNF (Nested Normal Form)** by Z. Meral Oasoyoglu and Li-yan Yuan, Normal form for XML documents by Marcelo Arenas and Leon ID Libkin, ETNF (Essential Tuple Normal Form) by H. Darwen, C. J. Date and R. Fagin. ETNF lies between 4NF and 5NF. ETNF is for preventing or eliminating redundant tuples.

5.12 | EXAMPLES

Example 1: Normalize Table 5.22 up to its maximum possible normal form.

Table 5.22 is not in 1NF, because the field custadd can be decomposed into addline 1, addline 2, city and pincode. When we decompose this field, we will get Table 5.23 which is in 1NF.

Table 5.23 (*i.e.*, PERIODICALS) has a primary key which is a combination of fields bill no and peri_ID. It contains the following functional dependencies:

Table 5.22 | PERIODICALS

Bill No	Bill Date	Cust ID	Cust Name	Cust Add	Peri_ID	Peri_Desc	Qty	Price	Total_pr	Delivery_ Charges	Bill_Total
123	2-5-2013	12	S. G. Shah	9, Aditi Apt, Ambawadi, Ahmedabad-3800015	11	Times of India	30	2.50	75.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt, Ambawadi, Ahmedabad-3800015	15	Divya Bhaskar	30	3.00	90.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt, Ambawadi, Ahmedabad-3800015	23	Economic Times	21	3.00	63.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt, Ambawadi, Ahmedabad-3800015	3	Mint	26	3.00	78.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt, Ambawadi, Ahmedabad-3800015	51	Champak	1	20.00	20.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt, Ambawadi, Ahmedabad-3800015	33	Entrepreneur	1	100.00	100.00	10.00	436.00
5	4-5-2013	3	M. N. Dave	11, Aditi Apt, Ambawadi, Ahmedabad-3800015	11	Times of India	30	2.50	75.00	10.00	175.00
5	4-5-2013	3	M. N. Dave	11, Aditi Apt, Ambawadi, Ahmedabad-3800015	15	Divya Bhaskar	30	3.00	90.00	10.00	175.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt, Ambawadi, Ahmedabad-3800015	12	Gujarat Samachar	28	3.00	84.00	10.00	204.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt, Ambawadi, Ahmedabad-3800015	11	Times of India	30	2.50	75.00	10.00	204.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt, Ambawadi, Ahmedabad-3800015	7	Open	1	35.00	35.00	10.00	204.00

Table 5.23 | PERIODICALS in 1NF

Bill No	Bill Date	Cust ID	Cust Name	Addline1	Addline2	City	Pincode	Peri_ID	Peri_Desc	Qty	Price	Total_pr	Delivery_Charges	Bill_Total
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	11	Times of India	30	2.50	75.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	15	Divya Bhaskar	30	3.00	90.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	23	Economic Times	21	3.00	63.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	3	Mint	26	3.00	78.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	51	Champak	1	20.00	20.00	10.00	436.00
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	33	Entrepreneur	1	100.00	100.00	10.00	436.00
5	4-5-2013	3	M. N. Dave	11, Aditi Apt	Ambawadi	Ahmedabad	380015	11	Times of India	30	2.50	75.00	10.00	175.00
5	4-5-2013	3	M. N. Dave	11, Aditi Apt	Ambawadi	Ahmedabad	380015	15	Divya Bhaskar	30	3.00	90.00	10.00	175.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt	Ambawadi	Ahmedabad	380015	12	Gujarat Samachar	28	3.00	84.00	10.00	204.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt	Ambawadi	Ahmedabad	380015	11	Times of India	30	2.50	75.00	10.00	204.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt	Ambawadi	Ahmedabad	380015	7	Open	1	35.00	35.00	10.00	204.00

(Bill No., peri_ID) $\longrightarrow$ bill date, cust ID, cust name, addline 1, addline 2, city, pincode, peri_desc, qty, price, total_pr, delivery_charges, bill_total

Following are the partial dependencies in a Table, PERIODICALS.

Bill No. $\longrightarrow$ billdate, delivery_charges, bill_total, custID, custname, addline1, addline2, city, pincode

Peri_ID $\longrightarrow$ peri_desc, price

To convert Table 5.23 into 2NF, the above partial dependencies should be removed from the Table by decomposing it into the following three Tables, namely PERIODICALS, BILL and BILL DETAIL.

For each partial dependency, there will be one Table with attribute which is on left side as a primary key. This attribute will remain in original Table and right hand side attributes will be removed from the original Table.

PERIODICALS (peri_ID, peri_desc, price)

BILL (Bill no, cust ID, bill date, delivery_charges, bill_total, cust name, addline1, addline 2, city, pincode)

BILLDETAIL (bill no, peri_ID, qty, total_pr)

After decomposition of the Tables, PERIODICALS, BILL and BILL DETAIL are shown in Table 5.24, 5.25 and 5.26 respectively, which are now in 2NF.

Table 5.24 (*i.e.*, PERIODICALS) and Table 5.26 (*i.e.*, BILL DETAIL) are also in 3NF because they do not contain any transitive dependencies.

Table 5.25 (*i.e.*, BILL) is not in 3NF because there exists the following transitive dependencies:

Billno $\longrightarrow$ cust ID, and

cust ID $\longrightarrow$ cust name, addline 1, addline 2, city, pincode

To convert Table 5.25 into 3NF the above transitive dependencies should be removed by decomposing it into two Tables namely BILL and CUSTOMER. Both the Tables, after decomposition, will be as follows:

BILL (billno, cust ID, billdate, delivery_charge, bill_total)

CUSTOMER (cust ID, custname, addline1, addline2, city, pincode)

Table 5.27 and Table 5.28 shows BILL and CUSTOMER Tables respective which are in 3NF.

Now, all the Tables, BILL, BILL DETAIL, CUSTOMER and PERIODICALS are in BCNF, 4NF and 5NF also.

Example 2: Normalize Table 5.29 up to its maximum possible normal form.

Table 5.29 is in 1NF. Primary key of this Table is a combination of fields case_ID and visit_date.

By applying the rules of normalization, the above Table can be decomposed into following Tables which are all in 5NF.

CASE (case_ID, visit_date, doc_ID, pat_ID, treat_ID, next_visit_date)

DOCTOR (doc_ID, doc_name)

PATIENT (pat_ID, pat_name)

TREATMENT (treat_ID, treat_desc)

Table 5.24 | PERIODICALS in 2NF

Peri_ID	Peri_Desc	Price
11	Times of India	2.50
15	Divya Bhaskar	3.00
23	Economic Times	3.00
3	Mint	3.00
51	Champak	20.00
33	Entrepreneur	100.00
12	Gujarat Samachar	3.00
7	Open	35.00

Table 5.26 | BILLDETAIL in 2NF

Bill No	Peri_ID	Qty	Total_pr
123	11	30	75.00
123	15	30	90.00
123	23	21	63.00
123	3	26	78.00
123	51	1	20.00
123	33	1	100.00
5	11	30	75.00
5	15	30	90.00
14	12	28	84.00
14	11	30	75.00
14	7	1	35.00

Table 5.25 | BILL in 2NF

Bill No	Bill Date	Cust ID	Cust Name	Addline1	Addline2	City	Pincode	Delivery_ Charges	Bill_ total
123	2-5-2013	12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015	10.00	436.00
5	4-5-2013	3	M. N. Dave	11, Aditi Apt	Ambawadi	Ahmedabad	380015	10.00	175.00
14	2-5-2013	16	V. P. Vyas	1, Aditi Apt	Ambawadi	Ahmedabad	380015	10.00	204.00

Table 5.27 | BILL in 3NF

Bill No	Bill Date	Cust ID	Delivery_Charges	Bill_Total
123	2-5-2013	12	10.00	436.00
5	4-5-2013	3	10.00	175.00
14	2-5-2013	16	10.00	204.00

Table 5.28 | CUSTOMER in 3NF

Cust ID	Cust Name	Addline1	Addline2	City	Pincode
12	S. G. Shah	9, Aditi Apt	Ambawadi	Ahmedabad	380015
3	M. N. Dave	11, Aditi Apt	Ambawadi	Ahmedabad	380015
16	V. P. Vyas	1, Aditi Apt	Ambawadi	Ahmedabad	380015

Table 5.29 | TREATMENT

Case_ ID	Doct_ ID	Doct_Name	Pat_ID	Pat_Name	Treat_ ID	Treat_Desc	Visit_Date	Next_Visit_ Date
21	D4	S. Nanavaty	P2	L. Mathur	T2	Medicines given	21-4-2013	23-4-2013
21	D4	S. Nanavaty	P2	L. Mathur	T3	Injection	23-4-2013	
21	D3	S. Trivedi	P2	L. Mathur	T11	Dressing	2-5-2013	
134	D4	S. Nanavaty	P9	A. Soni	T6	B.P Tablet	5-5-2013	5-6-2013
134	D4	S. Nanavaty	P9	A. Soni	T0	No treatment	5-6-2013	

SUMMARY

- Normalization means, reducing redundancy in a database by decomposing Tables further by preventing functional dependency.
- Functional dependency exists between attributes of the same Table. If A and B are two attributes of a Table, then B is said to be functionally dependent on A, if and only if, each value of A there exists precise value of B. Symbolically, it is written as A→B. It can be read as 'A functionally determines B' or 'B is functionally dependent on A'.
- In functional dependency A → B, the attribute, which is on the right side of the arrow, is known as dependent(B) and the attribute, which is on the left side of the arrow, is known as determinant(A). In functional dependency, if determinant is a combination of more than one attributes, then each attribute of this composition is called a component. If the dependent depends on any of this component other than the whole combination, the dependency is called, 'partial dependency'. If the dependent depends on the whole combination and, not on any of the component, the dependency is called, 'full functional dependency'.
- In a Table, if A, B and C are attributes, then A→B and B→C implies that A→C. This type of dependency between A and C is known as transitive dependency.
- Let A, B and C are attributes of a relation. Then B is said to be multi-dependent on A, if and only if, in each record (tuple) of a relation, the set of B values which match with a given (A, C) value pair depends only on the value of A and is completely independent of value of C. This dependency is known as multi-valued dependency.
- If a Table has projections $P_1, P_2, \ldots, P_n$, there exists join dependency (denoted as $*\{P_1, P_2, \ldots, P_n\}$), if and only if, every record (tuple) value of a relation is the join of its projections on $P_1, P_2, \ldots, P_n$.
- There are different levels of normal forms ranging from 1NF to 5NF and, they can be achieved by removing certain dependency by decomposing Tables.
- The Table is said to be in 1NF if all its attributes contain atomic values.
- The Table is said to be in 2NF, if and only if, it is in 1NF and there does not exists any partial dependencies.
- The Table is said to be in 3NF, if and only if, it is in 2NF and there does not exists any transitive dependencies.

- The Table is said to be in BCNF, if and only if, it is in 3NF and every determinant is a candidate key. The Table is said to be in 4NF, if and only if, it is in BCNF, and it does not contain any multi-valued dependency which is not functional dependency.
- The Table is said to be in 5NF, if and only if, it is in 4NF, and each projection (decomposition) of a relation should contain candidate key(s) of a relation.

Exercises

1. Discuss the following dependencies with examples:
   a. Functional dependency
   b. Full functional dependency
   c. Transitive dependency
   d. Multi-valued dependency
   e. Join dependency
2. What is a component? Explain it by giving an example.
3. Answer the questions with respect to the following Table.

Table: Termwork

Tw_ ID	Date_ Given	Fac_ID	Fac_ Name	Sub_ID	Sub_Name	Class_ID	Class_ Desc	Submi_Dt
101	12-7-2012	SN	Shefali Naik	FOP	Fundamentals of Programming	FY	First Year	31-7-2012
102	8-8-2012	SN	Shefali Naik	FOP	Fundamentals of Programming	FY	First Year	3-9-2012
103	10-12-2012	SN	Shefali Naik	PCL	Programming in C Language	FY	First Year	23-1-2013
104	3-3-2013	SN	Shefali Naik	APCL	Advanced Programming in C Language	FY	First Year	1-4-2013
105	2-7-2012	HD	Hemal Desai	AJ	AdvaTnced Java	TY	Third Year	23-7-2012
106	18-8-2012	HD	Hemal Desai	AJ	Advanced Java	TY	Third Year	13-9-2012
107	10-11-2012	HD	Hemal Desai	OS	Operating Systems	SY	Second Year	3-12-2012
108	9-2-2013	KG	Kunjal Gajjar	OOP	Object-Oriented Programming	SY	Second Year	10-3-2013
109	4-3-2013	KG	Kunjal Gajjar	OOP	Object-Oriented Programming	SY	Second Year	31-3-2013

a. Identify primary key of the Table.
b. Identify full-functional, partial and transitive dependencies.
c. Draw a dependency diagram for all types of dependencies.
d. Is the Table in 1NF? Why?
e. Convert the Table up to 5NF.

4. Fill in the blanks:

a. The Table is said to be in ______________, if it contains fields which have atomic values.
b. If the primary key is a combination of more than one field, then each field is known as ______________.
c. There may exist partial dependency, if primary key is a ______________.
d. If the primary key contains only one field, then there is no need to check for ____________ dependency.
e. If a given Table is in 1NF and, if primary key contains only one field, then it is said to be in ______________ normal form.
f. If billno→customer_ID and customer_ID→customer_status then bill no→customer_status is called ____________ dependency.
g. We can obtain _____________ normal form by removing multi-valued dependencies which are not functional dependencies.
h. Fifth normal form is also known as ____________ normal form.
i. Fifth normal form is related with ____________ dependency.
j. Projection means _______________ subset of a Table. (horizontal/vertical).
k. In notations of dependency, double arrow denotes ____________ dependency and, single arrow denotes _____________ dependency.
l. In Kindergarten_ID→kindergarten_name, _____________ determines ____________.
m. In branch_ID→branch_name, ______________ is dependent on _______________.
n. In school_ID→school_name, school_ID is called, '_____________'and school_name is called, '_____________'.
o. In person_ID→degree, ______________ multi-determines _________.
p. In, person_ID→hobby, _____________ is multi-dependent on _________.
q. When we remove transitive dependency, we obtain _________ normal form.
r. In ________ normal form, every determinant should be a candidate key.

5. Discuss the following normal forms with proper examples.

a. First NF
b. Second NF
c. Third NF
d. Boyce-Codd NF
e. Fourth NF
f. Fifth NF

6

CHAPTER

Managing Data Using Structured Query Language (SQL)

CHAPTER OBJECTIVES

- Knowing usage of Structured Query Language (SQL) in managing data.
- Knowing usefulness of data definition and manipulation commands.
- Learning SELECT statement to fetch data from a database.
- Understanding different types of constraints.
- Learning how to use different functions given in a database management system.
- Understanding the application of special operators.
- Retrieving data with complex nested, or sub query.
- Summarizing data using advanced SQL, such as rollup, cube and crosstab.
- Summary

6.1 | INTRODUCTION

In Relational Database Management System, data is managed by using **Fourth Generation Language (4GL)**, named as **Structured Query Language (SQL)**. Using simple commands available in SQL, we can retrieve (fetch), update (change/edit), insert (add) and delete (remove) data. SQL is a common language of Relational Database Management System (RDBMS) which is used for data management. The syntax of statements available in SQL can be used with very minor modifications in any RDBMS. In SQL, we can also use different types of functions to display data in different formats, to summarize the data, to calculate any mathematical formula, fetch different parts of date, display string (text) in upper and lower case, etc. These functions and their syntax vary from RDBMS to RDBMS. In this chapter, we will see SQL and different functions with respect to Oracle RDBMS. All the statements are executed in 'Oracle Database 10g Express Edition'.

In advanced SQL, we can do some advanced operations on data to display them according to groups. We will also see, how to put constraints (conditions) on fields and tables by using data definition commands. It is also possible to write one SELECT statement into another for complex queries. The SELECT statement is used to retrieve data from tables.

6.2 | DATA DEFINITION COMMANDS

Data definition commands are used to create, change or delete structures in which we are going to store data (*i.e.*, data definition commands are used to define data structures). For an example, the commands used to create/change/delete table/view/user, etc., are called, 'Data Definition Commands'.

Here, we will see, how to create a table structure, change table structure and delete table structure in a Oracle database.

In syntaxes given throughout this chapter, words written in upper case shows the keywords/reserve word, but it is not required to be written in upper case when we actually execute the command; part of the syntax written in square brackets shows that part is optional in the syntax; part of the syntax written in angular brackets shows the variables.

The following syntax is used to **create table** in Oracle:

```
CREATE TABLE <table_name> (<field1> data_type [constraint1
                                                constraint2,..]
                          [, <field2> data_type [constraint1
                                             constraint2..]]……….
            [, <fieldn> data_type [constraint1 constraint2..]])
```

1. In **CREATE TABLE** command <table_name> and <field_name> are any valid variable name, we can define any number of constraints on individual fields (*Note:* We will see different field level and table level constraints in Section 6.11.).
2. For each field, the data type should be defined. Table 6.1 shows some data types which we can use in Oracle.

Table 6.1 | Oracle 10g Data Types

Data Type	Maximum Size	Description
Char(size), or character(size)	2000 bytes	Used to store text.
Varchar2(size)	4000 bytes	
Long	2 GB	
Int or integer		Used to store integer values.
Float or real		Used to store real numbers.
Number(p, s) or numeric(p,s) or dec(p,s) or decimal(p,s)	Precision (p) range up to 38.	Used to store numbers. For example., number (5,3) will have 2 digits before decimal place and 3 digits after decimal place.
Date		Used to store date in dd-mon-yyyy or dd-mon-yy format.
Timestamp		Store date with time.
Blob	4GB binary data	Stores binary large object.
Clob	4GB character data	Stores character large object.
Nclob	4GB character data	Stores Unicode data.

Table 6.2 | Table KG

KG ID	KG Name	City	Pincode
1	Eurokids	Surat	
2	Kidzee	Baroda	
3	Eurokids	Ahmedabad	380015

3. The list of fields should be enclosed in round brackets and each field should be separated with comma.
4. For an example, to create a Table 6.2, the following command should be written and executed in Oracle.

```
create table kg(kgid int primary key, kgname varchar 2 (20),
city varchar 2 (20), pincode int)
```

5. The KG Table will be created with kgid as a primary key. It will have unique and, not null values.

To view the table structure, the 'describe' command is used as follows:

```
describe <table_name> or desc <table_name>
```

For an example,

```
Describe kg or desc kg
```

Now, if we want to make some changes in the table structure, we have to use the command **ALTER TABLE**. Using the 'alter table' command, we may add new fields; delete or modify existing fields, change data type and size of the fields; add, delete or modify constraints, etc.

Following syntaxes are used to **alter table** in Oracle.

```
ALTER TABLE <table_name> ADD (<field1> data_type constraint1
                                          constraint2, ....]
                         [, <field2> data_type [constraint1
                                         constraint2..]], …)
                      [MODIFY (<field1> data_type  constraint1
                                          constraint2, ...]
                         [, <field2> data_type [constraint1
                                         constraint2..]], …)
```

OR

```
ALTER  TABLE <table_name> DROP COLUMN <fieldname>
```

OR

```
ALTER TABLE <table_name> RENAME COLUMN <old_fieldname>
TO <new_fieldname>
```

The drop column and rename column options can not be combined with other options of ALTER TABLE command. They should be written in a separate, 'alter table' command.

Following are some examples of ALTER TABLE command:

```
1. alter table kg add address1 char(20) add address2 char(20) mod-
   ify pincode number (6) modify kgid int add constraint chk_kgid
   check(kgid<100) modify kgname varchar2(20) not null unique
2. alter table kg modify (pincode int, kgname varchar2(30))
3. alter table kg add (a1 int, b1 int)
4. alter table kg drop column a1
5. alter table kg rename column kgname to kgnm
6. alter table kg drop constraint chk_kgid
7. alter table kg add constraint ckh_city check (city = upper(city))
   add constraint chk_city_len check (length(city) > 0)
```

To delete any existing table structure, the ‘drop table’ command is used. The syntax to drop any table is given below:

```
DROP TABLE <table_name>
```

Following is an example of the ‘drop table’ command:

```
DROP TABLE kg
```

Thus, any CREATE, ALTER and DROP commands are known as Data Definition Commands.

6.3 | DATA MANIPULATION COMMANDS

Data manipulation commands are used to insert, manipulate and delete data to/from table. There are three data manipulation commands in SQL-INSERT, UPDATE and DELETE.

The different syntaxes of INSERT statement is as follows:

Syntax-1: `INSERT INTO <table_name> VALUES (field1_value, field2_value, …, fieldn_value)`

- The above syntax is used when we want to insert all values of all the fields. For an example:

```
insert into kg values(1,'Eurokids','Surat',null)
```

- When the values are of type text and date they must be enclosed within single quotations. When the field value is unknown ‘null’ keyword should be written instead of the value.

Syntax-2: `INSERT INTO <table_name> (<field1>, <field2>, …, <fieldn>) VALUES (field1_value, field2_value, …, fieldn_value)`

- The above syntax is used when we want to insert values of selected fields. For an example,

```
insert into kg(kgid,kgname) values(4,'Thumbelina')
```

- The following syntax is used when we want to insert filed values from a different table. Before inserting values into a table, the table must be created and, the fields in which we are inserting values, its data type should be matched with the field data types from where we are inserting values.

Syntax-3: `INSERT INTO <table1_name>[(field_names)] SELECT <field_names>/* FROM <table2_name>`

- For an example, the following statement will insert values of all the fields of the kg Table into the corresponding fields of Table kg1.

```
            insert into kg1 select * from kg
```

- Following is an example of INSERT ALL, …, SELECT command which is used to enter multiple rows in a single INSERT statement.

```
Insert all
into kg1(kgid, kgname) values (5, 'Mothers Pet')
into kg1(kgid, kgname) values (6, 'Todan')
into kg1(kgid, kgname) values (7, 'Radiant')
select * from dual
```

- If we want users to enter values at run-time, then it can be done by using : operator before a field name. For an example:

```
insert into kg(kgid,kgname) values(:kgid,:kgname)
```

- The above statement will take inputs for kgid and kgname from user and insert into table.
- The syntax of the UPDATE statement is as follows:

```
UPDATE <table_name> SET <field1 > = field1_value [, <field2>
= field2_ value, …, <fieldn> = fieldn_value] [WHERE <condition>];
```

- The above 'update' statement will update existing values of fields according to condition specified. If we write condition, updation will be made for all the records. For an example:

```
update kg set city = 'Ahmedabad', pincode = 380009
where kg ID = 7
```

- The above 'update' statement will change value of city = 'Ahmedabad' and pincode = 380009 for records where value of kg ID = 1.
- The syntax of DELETE statement is as follows:

```
DELETE FROM <table_name> [WHERE <condition>];
```

- The above 'delete' statement will remove records from the table. If we do not specify the 'where' condition, it will delete all the records. For an example,

```
delete from kg where city = 'Ahmedabad'
```

- The above delete statement will delete records of kindergarten which are in 'Ahmedabad'.

6.4 | SELECT STATEMENT AND ITS CLAUSES

The SELECT statement is used to retrieve data from one or more than one tables and display them into appropriate format. Also, it is used to display group-wise summary and records in a particular order.

The syntax of SELECT statement is as follows:

```
SELECT DISTINCT */<field1>[, <field2>, …, <fieldn>]
FROM <table1_name> [,<table2_name>, …, <tablen_name>]
[WHERE <condition>/<subquery>]
```

```
[GROUP BY <field1> [, <field2>, …, <fieldn>]
[HAVING <condition on aggregate function]]
[ORDER BY <field1> [ASC/DESC] [,<field2> [ASC/DESC], …,
 <fieldn> [ASC/DESC]]]
```

In the SELECT statement, only SELECT and FROM clauses are compulsory, all other clauses are optional. Other clauses can be used as per the requirement.

The different **clauses of SELECT** are explained as follows:

1. **SELECT:** It is used to select fields. After writing SELECT, if we want to display all the fields, then character '*' should be written. To display values of selected fields, field names should be specified after SELECT keyword. We can also specify arithmetic calculations and, various functions after SELELCT keyword. For an example:

   ```
   Select * from kg
   ```

   The above statement will display all the field values of table kg.

   ```
   Select kgid, kgname from kg
   ```

   The above statement will display all the values of fields kgid and kgname from table kg.

   ```
   Select length(kgname) from kg
   ```

   The above statement will display length of the values of field kgname for each record.

   ```
   Select total_fee_paid_fee from fees
   ```

   The above statement will display fees amount which is pending for all the records of fees table.

   ```
   Select max(length(kgname)) from kg
   ```

   The above statement will display kgname which is the longest.
2. **DISTINCT:** When DISTINCT is specified before the field names, it displays the unique values of a field. If many fields are selected and DISTINCT is written before those fields, it will display unique combination values of those fields.
3. **FROM:** It is used after SELECT, and it is a compulsory clause of SELECT statement. After FROM, table names are written from where we want to select data. Table names are separated with comma. For an example:

   ```
   Select * from class, student
   ```

   The above statement will display all the field values of tables class and student.

   ```
   Select class.classID, classdesc, stdname from class, student
   ```

   The above statement will display values of classid, classdesc and stdname fields. Here, both the tables class and student contains the field 'classid'. Therefore, it is required to write table name before field name to tell the DBMS that which table's classid we want to display. When field names are unique between tables, there is no need to specify table name before field name.
4. **WHERE:** It is an optional clause of SELECT, but when used it should be written after the FROM clause. It is used to specify conditions on the fields. According to the conditions, field values will be displayed. For an example:

```
Select * from class, student where class.classID = 'fy'
```

The above statement will display details of students who are in FY by selecting all the fields from tables class and student.

```
Select class.classID, classdesc, stdname from class,
student where stdid > 20;
```

The above statement will display details of only those students whose roll number is greater than 20.

4. **GROUP BY:** It is an optional clause of SELECT, but when used, it should be written after the WHERE clause. If WHERE is not required, then it should be written after the FROM clause. It is used to group data on a specific field. When we want to display summarized details for any group, the clause GROUP BY should be used. We may group data on multiple fields. Multiple fields are separated by comma in GROUP BY. While using GROUP BY, the following important points should be remembered, otherwise the query will cause an error.
   - We can write only those fields which are specified after GROUP BY. For an example, if grouping is done on dept_ID and emp_no, then GROUP BY is written as 'GROUP BY dept_ID, emp_no'. In this case, after select statement, we can write only two field names, dept_ID and emp_number (*i.e.*, only 'SELECT dept_ID, emp_no' is valid). Other fields can not be written after SELECT.
   - We can also write aggregate functions after SELECT. For example, we may write max(salary), min(salary) after SELECT.
   - For an example, the following SELECT statement is valid if grouping is done on dept_ID and emp_number.

   ```
   SELECT dept_ID, emp_no, avg(salary) FROM salary GROUP BY
   dept_ID, empno
   ```

   The above statement will display department-wise each employee's average salary.
5. **HAVING:** It is an optional clause of SELECT, but when it is used, it should be written after GROUP BY clause only and because it is used to specify condition on the group level. Mostly, in HAVING clause, condition is written on aggregate function. For an example:

   ```
       select class ID, count(std ID) from student group
       by classid having count(stdid) > 50
   ```

   The above SELECT statement will display class-wise total number of students for those class in which total number of students are more than 50.

   We can also write the WHERE clause with the HAVING clause. The WHERE clause will specify condition on each row, while the HAVING clause will specify condition on a group. For an example, the following query will display total number of students of class SY, if total number of students in SY are more than 50.

   ```
   select class ID, count (std ID) from student where
   class ID = 'SY' group by classidhaving count (std ID) > 50
   ```

   The difference between the WHERE and HAVING clauses is given in Table 6.3.
6. **ORDER BY:** It is an optional clause of SELECT, which is used to display data in a specific order. Using this clause data can be arranged in ascending or descending order.

Table 6.3 | Difference Between WHERE and HAVING Clauses

WHERE	HAVING
It is used when condition is written for each row.	It is used when condition is written for a group.
Aggregate functions cannot be used in the WHERE clause.	Aggregate functions can be used in the HAVING clause.
To use the WHERE clause, GROUP BY clause is not required.	To use the WHERE clause, the GROUP BY clause required.

Ordering can be done on multiple fields. If we want to arrange data in ascending order, then keyword ASC should be written after a field name and for descending order DESC should be written after a field name. For an example, the following query will arrange records of the student table into ascending order of 'classID' and descending order of 'stdname'.

```
Select * from student order by classid asc, stdname desc
```

Or

```
Select * from student order by classid, stdname desc
```

When GROUP BY clause is written in SELECT statement, then we can write only those fields after ORDER BY which are written after GROUP BY, no other fields can be used to arrange data in order or aggregate function. For an example:

```
Select classID, count(stdID) from student group
by classID order by classID desc, count (stdID) desc
```

The above query will display class-wise total number of students in descending order of class ID and descending order of total number of students within each class.

6.5 | AGGREGATE FUNCTIONS

Aggregate functions are used to display summarized data, such as maximum and minimumsal-ary, average participation in the event, total number of students in the class, total sales amount, etc.

There are many aggregate functions available in Oracle 10g, but five main functions are listed and explained below.

1. **Count(*) or count(field_name):** Count(*) counts total number of rows. It includes NULL values also during counting. For an example, if we execute the following query on Table 6.2 KG, then it will display 3 as output.

```
Select count(*) from kg
```

Count(field_name) counts total number of values in the field 'field_name' excluding NULLs. For an example, if we execute the following query on Table 6.2 KG then it will display 1 as output.

```
Select count(pincode) from kg
```

If there are no records or no values in the field, count function returns zero. The query, given below will group records on classid and will display total number of students in the class.

```
Select classID, count(stdID) from student group by classID
```

2. **Max(field_name):** The max function will return maximum value from the column. If grouping is done, it will return maximum value from each group. If there are no values in the field, it will return NULL. For an example:

   ```
   Select max(salary) from employee
   ```

   The above query will return maximum value from the field salary of the employee table.

   ```
   Select deptID, max(salary) from employee
   ```

   The above query will return maximum value of salary from each department of employee table.
3. **Min(field_name):** The min function will return minimum value from the column. If grouping is done, it will return minimum value from each group. If there are no values in the field, it will return NULL. For an example:

   ```
   Select min(salary) from employee
   ```

   The above query will return minimum value from the field salary of the employee table.

   ```
   Select deptID, min(salary) from employee
   ```

   The above query will return minimum value of salary from each department of the employee table.
4. **Avg(field_name):** The avg function will return average value from the column. If grouping is done, it will return average value from each group. If there are no values in the field, it will return NULL. For an example:

   ```
   Select avg(salary) from employee
   ```

   The above query will return average value of the field salary of employee table.

   ```
   Select deptID, avg(salary) from employee
   ```

   The above query will return average salary of each department of employee table.
5. **Sum(field_name):** The sum function will return total value of the column. If grouping is done, it will return total value for each group. If there are no values in the field, it will return NULL. For an example:

   ```
   Select sum(sales_amt) from sales
   ```

   The above query will return total sales amount from sales table.

   ```
   Select deptID, sum(sales_amt) from sales group by deptID
   ```

   The above query will return department-wise total sales amount from sales table.

6.6 | DATE AND TIME FUNCTIONS

Date and time functions are used to display date and time in different formats and, for calculations which are based on date. There are various date and time functions available in Oracle 10g. Date is stored in dd-mon-yyyy format in Oracle. Following are some important functions:

1. **Sysdate and current_date:** Both the functions return system date in dd-mon-yy format. For an example, if system date is 28-5-2012, sysdate and current_date both will display 28-May-12, if we execute the following query.

   ```
   Select sysdate, current_date from dual
   ```

2. **Add_months:** The syntax of this function is add_months(<date_var>,no. of months). It returns new date value after adding number of months into <date_var>. For an example, If we execute the following query it will return the date value 4-JAN-14.

   ```
   Select add_months(to_date('4-Jan-2013'), 12) from dual
   ```

   Any value, written in single quotations, is treated as a character value. Therefore, before passing any date value as a parameter, it should be converted into date using to_date conversion function. If date field is passed as a parameter in add_months, then there is no need to convert it using to_date, because its data type itself tells the server that the field which is passed as a parameter is a 'date' type of field.

3. **Months_between:** The syntax of this function is months_between(<date1>,<date2>). It returns total number of months between two dates which are passed as a parameter. For an example, If we execute the following query, it will return value 12.

   ```
   Select months_between(to_date('4-Jan-2014'), to_date
   ('4-Jan-2013')) from dual
   ```

   The following query will return value 7.22.

   ```
   Select months_between(to_date('4-Jan-2014'),
   to_date ('28-May-2013')) from dual
   ```

4. **Extract:** The syntax of this function is extract(<format> FROM DATE/TIMESTAMP <date_value in yyyy-mm-dd format'>). It returns value based on the format passed in the parameter. Some formats which we can pass in extract function are – hour, minute, second, year, month and day. Some examples are given below.

   ```
   select extract (year from date '2009-5-28') from dual
   ```

   The above query will return the value 2009.

   ```
   select extract (month from date '2009-5-28') from dual
   ```

   The above query will return the value 5.

   ```
   select extract (day from date '2009-5-28') from dual
   ```

   The above query will return the value 28.

   ```
   select extract (hour from timestamp '2009-5-28
   12:01:45') from dual
   ```

   The above query will return the value 12.

   ```
   select extract (minute from timestamp '2009-5-28
   12:01:45') from dual
   ```

   The above query will return the value 1.

   ```
   select extract (second from timestamp '2009-5-28
   12:01:45') from dual
   ```

   The above query will return the value 45.

5. **Systimestamp:** This function returns system time in 'dd-mon-yy hh.mm.ss:ssssss AM/PM timezone' format. For example, if system time is 28-5-2012 10:01:45:23233,

the function will display 28-5-2012 10.01.45.23233 AM + 05:30, if we execute the following query.

```
Select systimestamp from dual
```

6. **Last_day:** This function returns the last day of the month based on the date value passed in the function. For example, the following query will return the value 31-MAY-13.

```
select last_day (to_date ('28-May-13')) from dual
```

7. **Next_day:** The syntax of this function is next_day(<date_value>,<day_name>). This function returns the the date on which next <day_name> falls after the date < date_value>. For example, the following query will return the value 13-MAY-13, because 9-may-2013 is a thursday and next monday after 9-may-13 is on 13-may-2013.

```
select next_day (to_date ('9-May-13'),'monday') from dual
```

6.7 | STRING FUNCTIONS

String functions are used to format text data. Following are some useful string functions of Oracle 10g:

- **upper:** This function will convert the string into upper case which is passed as a parameter. The following query will display the output SHEFALI.

```
select upper ('Shefali') from dual
```

- **lower:** This function will convert the string into lower case which is passed as a parameter. The following query will display the output shefali.

```
select lower ('SHEfali') from dual
```

- **initcap:** This function will convert the string's first letter into upper case and, other characters into lower case, which is passes as a parameter. The following query will display the output Shefali.

```
select initcap ('shefali') from dual
```

- **substr:** The syntax of this function is substr(<string_var>, <start_char_no>,<total_no_of_char>). This function will display sub string of the string which is passed as a parameter. The following query will display a total 8 characters starting from third character which is 'e'. The output displayed will be 'efali Na'.

```
select substr ('Shefali Naik', 3, 8) from dual
```

- **length:** This function will display total number of characters in the string which is passed as a parameter. The following query will display the output **12.**

```
select length ('Shefali Naik') from dual
```

- **ltrim:** This function will remove blank spaces from left hand side of the string which is passed as parameter. For example, on execution of following query, extra spaces will be removed from left side of the string **'Shefali Naik'.**

```
select ltrim ('Shefali Naik') from dual
```

- **rtrim:** This function will remove blank spaces from right hand side of the string which is passed as parameter. For example, on execution of following query, extra spaces will be removed from right side of the string **'Shefali Naik'.**

```
select rtrim ('Shefali Naik') from dual
```

- **trim:** This function will remove blank spaces from both right and left hand side of the string which is passed as parameter. For example, on execution of following query, extra spaces will be removed from right and left side of the string **'Shefali Naik'.**

```
select trim ('Shefali Naik') from dual
```

- **concat:** This function will concat two strings which are passed as parameters. For ex., on execution of following query, both the strings 'Shefali' and ' Naik' will be merged and the output Shefali Naik will be displayed.

```
select concat('Shefali', 'Naik') from dual
```

6.8 | CONVERSION FUNCTIONS

Conversion functions are used to convert one data type into another data type. Following are some useful conversion functions of Oracle 10g.

- **To_number:** This function will convert inputted character into number.

```
Select to_number('4.15') from dual
```

- **To_char:** This function will convert any number into character.

```
Select to _char(3453) from dual
```

- **To_date:** This function will convert inputted string into date.

```
Select to_date('4-Jan-2003') from dual
```

- **To_timestamp:** This function will convert inputted string into timestamp.

```
Select to_timestamp('10-Sep-02 11:10:10.123000') from dual
```

6.9 | MATHEMATICAL FUNCTIONS

Mathematical functions are used to numeric calculations. There are many mathematical functions in oracle. Some of them are as follows:

- **Sqrt:** It displays square root of the number.
- **Round:** It rounds off the number.
- **Mod:** The syntax of function is mod(num1,num2). It returns the remainder when num1 is divided by num2. For example,

```
Select mod (9, 2) from dual
```

will display 1.

- **Power:** The syntax of function is mod(num1,num2). It returns the 'num1 raise to num2'. For example,

```
Select power (3, 2) from dual
```

will display 9.

6.10 | SPECIAL OPERATORS

There are five **special operators** in SQL which can be used in WHERE clause to specify the condition.

1. **Is null:** This operator will check that the field value contains null or not. IS NULL will return true if field contains null, else it will return false. For example, the query given below will display the records in which value of pincode field is null.

   ```
   Select * from kg where pincode is null
   ```

   The logical operator NOT can also be combined with is null. For example, the query given below, will display the records in which value of pincode field is not null.

   ```
   Select * from kg where pincode is not null
   ```

2. **In:** This operator will check that the specific field value is contained within the list of value or not. If the value contained in the list, it displays the records based on that value. After IN operator we can specify constant values or any SELECT subquery. But in subquery, the data type of field name written after SELECT should match with the data type of field which is written after IN. For example, the following query will check whether there exists values of kgid field in the list of values (2,4,6,8,10). If value of any kgid matches with the list of values, its record will be displayed in the output.

   ```
   Select * from kg where kg ID in(2,4,6,8,10)
   ```

   The following query will display details of only those departments from dept table whose deptid matches with the deptid of employee table.

   ```
   Select * from dept where dept ID in(select deptID
   from employee)
   ```

   We can also use NOT logical operator with IN. For example, the following query will display details of only those departments from dept table whose dept ID does not match with the deptid of employee table.

   ```
   Select * from dept where deptID not in(select deptID
   from employee)
   ```

   We can also match pair of values using IN operator. For example,

   ```
   Select * from employee where(empID,empname) not
   in (select empno,empnm from employee_history)
   ```

   Another example of multiple values is given below:

   ```
   Select * from class where(classID, classdesc, capacity) not
   in((‘sy’, ‘second year’, 150), (‘ty’, ‘third year’, 120))
   ```

   We can also use AND and OR logical operators with IN. For example,

   ```
   Select * from class where class ID in (‘fy’)
   and classdesc not in (‘second year’, ‘third year’)
   ```

3. **Exists:** This operator is used with **subquery**. It checks the existence of records in a particular table and returns true to outer query if table contains at least one record, else

returns false. If the value true is returned, the outer query will display the result, else it will not display any result. For example, in the following query, first subquery will be executed. If there exists at least one record in a student table, exists will return true to the outer query and then outer query will display the values of stdid and stdname from remarks table.

```
 Select stdID, stdname from remarks where exists
(select * from student)
```

Remember that in the nested query(subquery), we have to write * after select. We can not specify fields names in the nested query. We can also use logical operator NOT with EXISTS. For example:

```
Select std ID, stdname from remarks where not
exists (select * from student)
```

We can also use logical operators AND and OR with EXISTS.

```
 Select std ID, stdname from remarks where exists
(select * from student) or exists(select * from class)
```

4. **Between:** This operator checks whether the field value lies between to specific values or not. It includes the lower and upper value while checking the condition. For example, the query given below, will display records of students for only those students whose roll number lies between 1,2,3,4, …, 10. The clause ‘std ID BETWEEN 1 and 10’ is same as ‘std ID ≥ 1 and std ID ≤ 10’

```
Select * from remarks where stdid between 1 and 10
```

We can also use logical operator NOT with BETWEEN.

```
Select * from remarks where stdid not between 2 and 5
```

5. **Like:** This operator matches the pattern and displays the result if field value matches with the specified pattern. Wildcards %(percentage) and_(underscore) are used in writing patterns. Different wildcards are used in pattern. % means many characters and_ means only one character. For example, the query as given below will display student names which start with character ‘s’.

```
Select stdname from student where stdname like 's%';
```

To display the faculty details whose name end with letter ‘I’ or ‘a’, the following query is written:

```
Select * from faculty where faculty_name like '%i'
or faculty_name like '%a';
```

To display the faculty details whose name start with any letter, but second letter should be ‘e’ and third letter should be ‘t’, and remaining letter can be anything; the following query is written:

```
Select * from faculty where faculty_name like '_et%'
```

To display the subject details which contains the ‘data’ word anywhere in the subject name, the following query is written:

```
Select * from subject where subject_name like '%data%'
```

6.11 | TYPES OF CONSTRAINTS

Basically there are two types of constraints:

- **Table-level Constraints:** The constraint which is applied on combination of more than one field is known as table-level constraint. For example, a primary key which is a combination of two or more than two fields is a table-level constraint.
- **Field-level Constraints:** The constraint which is applied on a single field is known as field-level constraint. For example, a primary key, which contains a single field, is a field-level constraint.

In Oracle, during table creation we can apply different types of constraints which are given below.

1. **Primary key:** It is a constraint which identify each row of a table uniquely. The primary key can not contain NULL and duplicate value. If primary key is a composite key (combination of more than one field), then none of the fields of this composite key can be null and the combination must be unique. For example, in the class table classid is a primary key. It contains single field, so it is a field-level constraint.

```
Create table class(classID int primary key,
classdesc char(5))
```

   In the table created below, primary key is a combination of two fields classid and stdid. Therefore, it is called, 'table-level constraint'.

```
Create table student(stdID int, classID int, std name
varchar 2 (30), primary key (std ID, class ID))
```

2. **Foreign key:** It is a constraint which refers primary key of another table and will accept, (1) only those values which are there in that primary key, or (2) null. The data type of a foreign key should match with the primary key from where it is referred, field names may be different. When we refer composite key, the entire composite key should be referred. For example, if we want to refer classid field of class table into student table, it should be written as follows :

```
Create table student(std ID int, class code int
references class (class ID), stdnamevarchar 2 (30),
primary key (std ID, class code))
```

   If we want to refer primary key of student table into result table, the entire primary key (combination of stdid and classcode) should be referred and sequence of fields should be maintained.

```
Create table result (exam ID int, stdid int, classcode
int, foreign key
(std ID, classcode) references student(std ID,
classcode),
primary key (examid, std ID, classcode))
```

   A table can have more than one foreign keys. For example, the following table has 3 foreign keys-doctorid referred from doctor table, patientid referred from patient table and treatmentid referred from treatment table.

```
create table doctor(doctorid int primary key)
create table patient(patientid int primary key)
create table treatment(treatmentid int primary key)
create table case(doctor ID int references
doctor (doctor ID), patient ID
int references patient(patient ID), treatmentid int
references treatment(treatment ID))
```

3. **Unique:** The fields will accept only unique values on which unique constraint is applied. For example, in the following table, employee, the unique constraint is applied on PAN_cardno, therefore it can not accept duplicate values. But unique key accepts null values. Those employees who have applied for PAN card, we can enter null in the PAN_cardno for them.

```
Create table employee (empno int primary key,
PAN_cardno char(10) unique, basic_salary float)
```

Now, see the result after inserting the following records in the employee table.

```
insert into employee values(1, 'gsrth6545m', 50000)
insert into employee values(2, null, 50000)
insert into employee values(3, null, 50000)
```

4. **Not null:** The fields will not accept null values on which not null constraint is enforced. For example, in the following table employee, the not null constraint is enforced on empname. If employee table is already created, it can be altered to add the filed empname with not null constraint as follows:

```
alter table employee add empname varchar2 (30) not null
```

5. **Default:** This constraint should be applied when we want to insert some default value, especially when that value is repeated many times in the field. For example, in the worker table, gender of worker is mostly 'male'. So, default constraint can be defined for the field gender as following.

```
Create table worker(worker_ID int primary key,
worker_name  varchar2(30), gender char(1) default 'm')
```

To input default value in the field, keyword 'default' should be written instead of value. When we do not want to insert default value, simply write the value which we want to insert. Following are two insert statement. In the first statement, default value('m') will be inserted in the field gender and in second statement 'f' value will be inserted in the field gender.

```
Insert into worker values(101, 'F. Chaudhary', default)
Insert into worker values(102, 'N. Chaudhary', 'f')
```

6. **Check:** This constraint is used to check specific condition before we insert value in the field, such as the length of pincode should be exactly 6 digits, itemname should start with letter 'I', empname should be entered into uppercase, etc. Following is an example of check constraints which are applied on different fields of customer table:

```
Create table customer (custid int check(custid > 100
and custid < 1000), custname varchar 2(30)
check(custname = upper(custname)),
city varchar2 (20), pincode int check( length
(pincode) = 6))
```

6.12 | TYPES OF JOIN AND SET OPERATORS

We can join multiple table using different types of joins. To join the tables, we can use different relational operators, such as <, >, ≥, ≤, = and != in WHERE clause. Following are some join types which can be used to join multiple tables:

1. **Equijoin/Simple Join/Natural Join:** When we join two or more than two tables using an '=' sign, the type of join is said to be an equi join. For example,

```
select * from class, student where class.
classID = student.classcode
```

2. **Non-equi Join:** When we join two or more than two tables using a sign other than '=', the type of join is said to be an equi join. For example,

```
select * from class, student where class.
class ID ≥ student.class code
```

3. **Cross Join:** When join condition is not specified in WHERE clause, the join is said to be cross join. Cross join displays cartesian product of the tables. For example,

```
select * from class, student
```

4. **Multiple Join:** When we join more than two tables in a single query, the type of join is said to be a multiple join. For example,

```
select * from class, student, mark where class.
classID = student.classcode and student.
stdno = mark.stdno
```

5. **Inner Join:** Inner joins displays common values from the tables which are joined on a common field. For example,

```
select * from class inner join student on class.
classID = student.class code
```

6. **Outer Join:** There are three types of outer joins.

   Full Outer Join: Full outer join displays all the records of the left table and all the records of the right table (*i.e.*, it displays union of two tables). For example:

```
select * from class full outer join student on class.
class ID = student.classcode
```

   Left Outer Join: Left outer join displays all the records which is on the left side and matching records of the right side table. If the left side values do not exist in the right side table, then null values will be displayed for the right side table's records. For example:

```
select * from class left outer join student on class.
classID = student.classcode
```

Right Outer Join: Right outer join displays all the records which is on the right side and matching records of the left side table. If the right side values do not exist in the left side table, then null values will be displayed for the left side table's records. For example,

```
select * from class right outer join student on class.
class ID = student.classcode
```

7. **Self Join:** If a table is joined with itself, the type of join is called, self join. For example:

```
Select s1.stdname, s2.stdname from student s1, student s2
where s1.city = s2. city and s1.stdno! = s2.stdno
```

The above query will create two alias s1 and s2 of table student and then compare city of each record of s1 with city of each record of s2. If city is same, then it will display pair of those students. Here, it will not display student pair whose roll number is same.

Set Operators: Set operators are also used to join tables. Following are the set operators which we can use in Oracle:

1. **Union:** Union operator displays all the records from two or more tables on the basis of selected fields. When we use union set operator, number of fields written after SELECT in both all the queries should be same and datatype of corresponding fields should match with one another. For example:

```
select class ID from class union select classcode
from student union select classID from marks
```

The above query will display union of three tables class, student and marks.

2. **Intersect:** Intersect operator displays all the common records from two or more tables on the basis of selected fields. When we use intersect set operator, number of fields written after SELECT in both all the queries should be same and datatype of corresponding fields should match with one another. For example:

```
select classID from class intersect select classcode
from student intersect select classID from marks
```

The above query will display intersection of three tables class, student and marks.

3. **Minus:** Minus operator displays all the records from the first table which are not there in second table. For example:

```
select class ID from class minus select classcode
from student minus
select class ID from marks
```

6.13 | SUB-QUERY

Sub query is a SELECT query which is written within another SELECT query. It is also known as **nested query**. We can perform complex queries using sub query. For example:

```
Select empname from employee where
salary = (select max(salary) from employee)
```

The above query will display name of the employee who earns maximum salary.

Following are some other examples of sub-query:

```
 Select * from employee where empid not in
(select empid from loan)
```

The above query will display details of employees who have not taken any loan.

```
Select max (percentage) from result where
class ID = 'TY' and percentage <
(select max(percentage) from result where classID = 'TY')
```

The above query will display the second highest percentage of class 'TY'.

```
Select r.class ID, max(r.percentage) from result
r where r.percentage <(select max
(percentage) from result where classID = r.classID)
group by r.class ID
```

The above query will display class-wise second maximum percentage. This type of query where each sub-query is executed for each row of outer query is called **co-related sub query**. In this type of query the alias is used in outer query.

```
select * from class where class ID in (select
class ID from student where std ID in (select
stdid from marks))
```

The above query will first fetch classid from student table of those students whose stdid also exists in marks table and on the basis of this, it will display class details of those classes which are returned by the inner query.

Whenever we execute sub-query, the inner most query is executed first and on the basis of output of inner query, outer query is executed.

6.14 | ADVANCES SQL ROLL-UP, CUBE, CROSSTAB

Rollup and **Cube** are used with the GROUP BY clause. With GROUP BY, it can be used as follows:

```
SELECT….GROUP BY cube/rollup (<field1>[,<field2>,….<fieldn>]
```

Rollup: Rollup will display group-wise summary, such as count, sum, average, minimum, maximum, etc. Consider Table 6.4 (EVENT).

If we execute the following query, then it will display in each event type how many total number of events of a particular category. At the end of each event_type group, it will display total number of events of each event_type. Also, at the end of total number of events of last event_type group, it will display overall total number of events.

```
select event_type,event_category, count(event_ID) from
event group by rollup (event_type,event_category)
```

The above query will display the result as given in Table 6.5

Cube: Cube also displays group-wise summary, but it starts with overall summary and then displays each group-wise summary. It is like a cross tab summary.

If we execute the following query, the output given in Table 6.6 will be displayed. First, it will display overall event count, then total number of events in each event type and, within each event type, it will display type-wise category-wise total number of events.

```
select event_type, event_category, count (event_ID) from
event group by cube (event_type, event_category)
```

Table 6.4 | EVENT

Event_ID	Event_Name	Event_Category	Event_Type	Min_Part	Max_Part
1	solo singing	music	solo	1	1
2	duet singing	music	group	2	2
3	group singing	music	group	3	8
4	solo western singing	music	solo	1	1
6	Classical Instrumental	music	solo	1	1
7	skit	drama	group	3	10
8	mime	drama	group	3	10
9	mono acting	drama	solo	1	1
10	mimicry	drama	solo	1	1
11	elocution	intellect	solo	1	1
12	extempore	intellect	solo	1	1
13	poetry recitation	intellect	solo	1	1
14	book review	intellect	solo	1	1
15	poetry writing	intellect	solo	1	1
16	debate	intellect	group	2	2
17	collage	art	group	3	3
18	poster making	art	group	3	3
19	graffiti	art	group	3	3
20	cartooning	art	solo	1	1
21	glass painting	art	solo	1	1
22	rangoli	art	solo	1	1
23	mehendi	art	solo	1	1
24	salad making	art	solo	1	1
25	solo dancing	dance	solo	1	1
26	duet dancing	dance	group	2	2
27	group dancing	dance	group	3	10
5	group western singing	music	group	3	8

Table 6.5 | Output of 'Rollup'

Event_Type	Event_Category	Count (Event_ID)
Group	art	3
Group	dance	2
Group	drama	2
Group	intellect	1
Group	music	3
Group	—	11
solo	art	5
solo	dance	1
solo	drama	2
solo	intellect	5
solo	music	3
solo	—	16
—	—	27

Table 6.6 | Output of 'Cube'

Event_Type	Event_Category	Count (Event_ID)
—	—	27
—	art	8
—	dance	3
—	drama	4
—	intellect	6
—	music	6
group	—	11
group	art	3
group	dance	2
group	drama	2
group	intellect	1
group	music	3
solo	—	16
solo	art	5
solo	dance	1
solo	drama	2
solo	intellect	5
solo	music	3

Crosstab: With cross tabulation, we can display row and column-wise summary. For example, the following query will display category-wise total number of solo event and group events.

Decode: The Decode function is used to count type-wise total number of events in each category. The first argument in decode function is a field name, second argument is a value which we want to search in the field which is specified as first argument, third argument is a field of which we want to find count. Table 6.7 shows output of the following query which shows crosstab of event category and event type.

```
select upper (event_category) 'EVENT CATEGORY',
count (decode (trim(event_type),' solo', event_ID, NULL))
'SOLO EVENTS', count( decode(trim (event_type), 'group',
event_ID, NULL)) 'GROUP EVENTS' from event group by
event_category order by event_category
```

The following query will display crosstab event type and event category. The query will display the output which is given in Table 6.8.

```
 select upper (event_type) 'EVENT TYPE',
 count(decode(trim(event_category),'art',event_ID))
'Artistic Events', count
(decode (trim (event_category), 'dance', event_ID))
'Dancing Events', count (decode (trim (event_category),'
 drama', event_ID)) 'Dramatic Events', count (decode
(trim (event_category),' intellect', event_ID))
```

```
'Intellectual Events', count (decode (trim (event_
 category), 'music', event_ID)) 'Musical Events'
 from event group by event_type order by event_type desc
```

Table 6.7 | Cross Tab of Event Table Which Shows Category-wise Total Number of Solo and Group Events

Event Category	Solo Events	Group Events
ART	5	3
DANCE	1	2
DRAMA	2	2
INTELLECT	5	1
MUSIC	3	3

Table 6.8 | Cross Tab of Event Table Which Shows Type-wise Total Number of Art, Dance, Drama, Intellect and Music Events

Event Type	Artistic Events	Dancing Events	Dramatic Events	Intellectual Events	Musical Events
SOLO	5	1	2	5	3
GROUP	3	2	2	1	3

SUMMARY

- The language which is provided to manipulate data stored in a relational database management system is known as Structured Query Language (SQL).
- SQL has many commands, such as Data definition, data commands, Data Manipulation commands, Data control commands.
- Data definition commands are used to create, modify and delete database objects, such as tables, views, procedures. Examples of data definition commands are CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW, etc.
- Data manipulation commands are used to fetch, insert, remove and change values in a table. INSERT, UPDATE, DELETE and SELECT, etc., are data manipulation commands.
- In SELECT statement, there are clauses FROM, WHERE, GROUP BY, HAVING and ORDER BY. The FROM clause is only compulsory in SELECT statement; all the other clauses are optional.
- In the SELECT statement, after SELECT keyword, field names/* is specified. * means, all the fields of a table. If unique records need to be displayed, then before the field names the DISTINCT predicate is written. Also, we can specify functions and mathematical calculations after the SELECT keyword.
- In the FROM clause, table names are specified. Also, we can write SELECT statement in FROM clause.

- WHERE is used to write conditions on record level. After WHERE we can also write sub-queries.
- GROUP BY is used to group records and display summary of groups. We can group more than one field.
- The HAVING clause is written only with the GROUP BY clause to specify condition on aggregate functions. HAVING is used to write a condition on group level, while WHERE is used to write a condition on row level.
- ORDER BY is used to arrange records in ascending or descending order. Records can be arranged on multiple fields also.
- With group by clause, we can rollup and cube to show summary on different groups.
- There are many functions which we can use to format data and for calculation, such as string functions, mathematical functions, conversion functions and date functions.
- Aggregate functions are used to show count, maximum, minimum, sum and average of column values.
- There are table level and field level constraints which we can enforce on table and field. They will validate the data before data is inserted into database. If any constraint is violated, error will be displayed. The constraints which are written on single field are called field level constraints and the constraints which are written on more than one field are called table level constraints.
- Various types of constraints are the primary key constraint, foreign key constraint, unique constraint, default constraint, not null constraint and check constraint.
- We can join multiple tables using different types of joins, such as inner join, outer join, equijoin, nonequi join, self join, cross join and multiple join.
- There are some special operators which are used with WHERE clause. They are LIKE, BETWEEN, IN, EXISTS and IS NULL. LIKE operator is used to match patterns in data, BETWEEN operator is used with numeric fields to check if value lies within the range or not, IN will check that field value match with the specified list or values or not, EXISTS will check existence of data in a table and IS NULL will check whether the field contains null or not.
- Besides join types, there are some set operators which are used to join tables. They are UNION, INTERSECT and MINUS. To use these operators, the tables or SELECT statements, used in queries, should be union-compatible. Union-compatible means the tables or SELECT statements should contain same number of fields and data types of corresponding fields should also match.
- For complex data retrieval, we can use sub-query or nested query. The SELECT statement which is written within another SELECT statement is called sub-query.
- Also, crosstab results can be displayed using decode function.

EXERCISES

1. What is SQL?
2. Define Data Definition and Data Manipulation Commands.
3. Explain SELECT statement with all the clauses.
4. Write syntax of INSERT, DELETE and UPDATE statements.

5. Give one-one examples of each of the following operators:

a. Is null	b. Is not null
c. In	d. Not in
e. Exists	f. Not exists
g. Between	h. Not between
i. Like	j. Not like

6. Explain different types of constraints. Differentiate between field level and table level constraints.
7. Write the differences between WHERE and HAVING clauses.
8. What is the difference between count(*) and count(field_name)?
9. Which points should be kept in mind while using GROUP BY clause in SELECT?
10. Describe different types of joins with examples.
11. What is co-related query? When is it used? Give an example.
12. Write syntax and return type of any two of the given functions:

a. Aggregate functions	b. Mathematical/numerical functions
c. String/text functions	d. Date functions
e. Conversion functions	

13. Create the following tables with appropriate data types and constraints. Insert given records and solve the queries.

Kindergarten

kg ID	Kg Name	Main Branch	Sub Branch	City	State
1	Kidzee	Navrangpura	Naranpura	Ahmedabad	Gujarat
2	Kidzee	Navrangpura	Satellite	Ahmedabad	Gujarat
3	Thumbelina	Naranpura		Ahmedabad	Gujarat
4	Eurokids	Naranpura	Vastrapur	Ahmedabad	Gujarat
5	Redbricks	Satellite	Paldi	Ahmedabad	Gujarat
6	Eurokids	Citylite		Surat	Gujarat
7	Eorilids	Alkapuri	Makarpura	Baroda	Gujarat

Class Details

Class Code	Desc1	Min_age_Required	Others
PG	Play Group	2	Admission given in the month of November and May.
NUR	Nursery	2.5	Admissions open in the month of April
JrKG	Junior KG	3.5	Admissions open in the month of April

KGdetails

Kg ID	Class ID	Division	Capacity
1	PG	1	30
1	NUR	1	25
1	NUR	2	25
1	JrKG	1	40
2	NUR	1	20
2	JrKG	1	30
4	NUR	1	30
4	NUR	2	30
4	JrKG	1	30

i. Display average capacity of the class ‘Nursery’.
ii. Display names of kindergartens which do not have any sub-branch.
iii. Display city-wise total number of kindergartens.
iv. Display intersection of classdetails and kgdetails tables.
v. Display class-wise minimum age requirement.
vi. Display details of kindergartens of state ‘Gujarat’.
vii. Display total capacity of class ‘Nursery’.
viii. Display the sub-branches of kindergarten, ‘Eurokids’.
ix. Display names of kindergartens which have more than two branches in the same city.
x. Display state-wise total number of kindergartens.
xi. Display details of kindergarten which exists in the kindergarten table, but not in KG details.
xii. Display kindergarten name, main branch and total number of sub branches.
xiii. Copy records of table KGdetails in a new table named KGdetails_2010
xiv. Display name of the kindergarten which has maximum number of sub-branches.
xv. Display kgid which has maximum no. of divisions of class ‘Jr. KG’.
xvi. Display union of KGdetails and Kindergarten table.
xvii. Display Kindergarten names only once from the kindergarten table.
xviii. Display details of top 2 classes which have maximum capacity.
xix. Display kindergarten names which contains the word ‘kid’.
xx. Display total capacity from KGdetails table.
xxi. Display name of the main branch for which sub-branch contains null.
xxii. Display all the details of kindergarten where capacity is 25 or 30.
xxiii. Display name of the city which has the maximum length.
xxiv. Display total no. of records of KGdetails.
xv. Display unique records from KGdetails.

14. Create the following tables with appropriate data types and constraints. Insert given records and solve the queries.

Item

Item Code	Item Desc	Price	Qty_on_Hand (in Pcs.)	Reorder_Level
I003	Polo-neck T-shirt	345.00	50	25
I004	Turtle-neck T-shirt	450.00	50	25
I005	Hooded T-shirt	750.00	5	10
I006	Round-neck T-shirt	299.00	100	50

Customer

Cust Code	Cust Name	Address	Balance
C1	A.R. Patel	23, Acme house	5000.00
C2	S.M. Sharma	Navrangpura	2000.00

Invoice

Invoice No	Invdate	Cust Code	Total_inv_Amt
1034	23-03-1999	C1	3522.00

Invoice Detail

Invoice No	Item No	Qty	Price	Total_Price
1034	I004	2	450.00	900.00
1034	I003	5	345.00	1725.00
1034	I006	3	299.00	897.00

i. Display name of items, which are below reorder level.
ii. Display information of all customers who have purchased items for which price is more than ₹1000.00.
iii. Display all item names, which are not purchased by any customer.
iv. Display information of all items, which are purchased between 1st January, 2001 to 28th February, 2001.
v. Find second maximum price from the ITEM Table along with itemcode.
vi. Display invoice wise total price in descending order of invoiceno.
vii. Display union of ITEM and INVOICE Tables.
viii. Display customer number and customer name whose name ends with letter 'a' or 'i'.
ix. Display number of items that are sold in the month of 'May'.
x. Print all the records of ITEM table for which quantity on hand is less than reordered level.
xi. Display sum of all item quantities, for those items, which are purchased by any customer.

xii. Copy all the records of the Invoice Table in a new table named inv_history.
xiii. Display first ten characters of customer name.
xiv. Display the number of items purchased up to the current date.
xv. Display information of all items for which quantity is same.
xvi. Display sum of the field 'Total_inv_amt'.
xvii. Display details of itemno 'I006'.
xviii. Change price of itemno I002 to ₹500.00.
xix. Delete record if item I001.
xx. Display join of Customer, Invoice and Invoicedetail tables.
xxi. Display item details in descending order of price.
xxii. Display 12% records of item table.
xxiii. Display names of the items which are purchased maximum.
xxiv. Display item names which contain the word 'neck'.
xxv. Display unique item names.
xvi. Decrease price of each item by ₹100.00.
xvii. Display multiplication of the fields qty and price and give the name 'Total Price' to this field.
xviii. Display details of customers with balance >1000.00 and <7000.00.
xxix. Display first ten characters of the address field.
xxx. Display length of the itemdesc field along with the description of items.

7 CHAPTER

Introduction to PL/SQL

CHAPTER OBJECTIVES

- Creating functions in Oracle.
- Creating procedures in Oracle.
- Applying validations and security through triggers in Oracle.
- Overview of packages in Oracle.

7.1 | INTRODUCTION

PL/SQL is an abbreviation of Procedural Language/Structured Query Language. We can create small programs using PL/SQL named and unnamed blocks.

By writing SQL update/select/insert/delete statement, we can apply same condition on set of records, but we cannot apply different conditions on different records depending upon the field values. It could be done by declaring cursor and accessing the cursor within the PL/SQL block.

We may also write stored procedure or function which could be called from remote computer. Using exceptions, system errors could be trapped or handled. Triggers could be written to apply validations at record level and to apply security on tables. We can combine PL/SQL blocks within a package and access them.

7.2 | BLOCK OF PL/SQL IN ORACLE

PL/SQL block has three basic parts: a declarative part (DECLARE), an executable part (BEGIN ... END) and an exception-handling part (EXCEPTION). Variables and cursors are defined in Declare section. Declare section is optional. Declare section should be written first in the block. After Declare, body part is written within Begin … End, which is compulsory to write in PL/SQL block. Operations could be performed in this executable part. Errors raised during execution can be handled within the exception-handling part. User can define his/her own exception in this part.

PL/SQL Block Structure:

```
Declare
    Declarations
    ----------------
Begin
Statements
    --------------------
Exception
Error handlers
    --------------------
End;
```

Following is an example of PL/SQL unnamed block:

```
DECLARE
     Sub_marks NUMBER;
     Sub_credit  NUMBER;
     Total_credit NUMBER;
     Grade_point float;
BEGIN
     Grade_point = (sub_marks*sub_credit)/total_credit
     dbms_output.put_line ('Grade Points = ||grade_point');
END;
/
```

Example 2

```
declare
    fname char(10):='&fname';
    lname char(10):='&lname';
    begin
    dbms_output.put_line('Full name is:
    '||rtrim(fname)||', '||lname);
end;
```

7.3 | CURSORS IN ORACLE

When any SQL statement is executed, it results into a record set. This record set is allocated some area in a memory. We can give name to this area, which is known as a **cursor**. Cursor points to that area.

When you declare a cursor, you get a pointer variable, which does not point any thing. When the cursor is opened, memory is allocated and the cursor structure is created. The cursor variable now points the cursor. When the cursor is closed, the memory allocated for the cursor is released.

Cursors allow the programmer to retrieve data from a table and perform actions on that data one row at a time. There are two types of cursors—implicit cursors and explicit cursors.

1. **Implicit cursor: Implicit cursor** could be declared for the select query which returns only one row/record. Implicit cursor is a select statement which is written in the body

part (begin…end) of a PL/SQL block. The oracle's implicit cursor is referred with the name SQL. When defining implicit cursor, it is required to write 'into' after select statement. Following is the syntax to define implicit cursor.

```
SELECT <fieldname 1>, <fieldname 2>,
        … INTO <variable_name 1>,
               <variable_name 2>,
             … FROM <table_name>;
```

The above select statement should return exact one row.
For example,

```
DECLARE
    Cnt int;
BEGIN
    SELECT count(stdno)
    INTO cnt FROM student;
    Dbms_output.put_line('Total no. of students are:
    '||cnt);
END;
```

Note: stdno is a column of the table student and cnt is a variable used to store total number of students.

2. **Explicit Cursor: Explicit cursors** is used in queries that return multiple rows. Explicit cursor is declared in the DECLARE section of PL/SQL program. Explicit cursor is a user-defined cursor. Following is the syntax to define explicit cursor.

```
CURSOR <cursor-name> IS <select statement>
```

For example,

```
DECLARE
    CURSOR cur_emp IS SELECT ename FROM EMP;
BEGIN
__________
__________
END;
```

After declaring the explicit cursor, we need to follow the steps given below to retrieve and process records which this cursor contains.

i. Open cursor. (syntax: `OPEN <cursor-name>;`)
   Example, `open cur_std;`
ii. Fetch records in a loop and process them.
   (syntax: `FETCH <cursor-name> INTO <variables>;`
   Example, `fetch cur_std into c_std_rec;`
iii. Specify condition to exit from the loop.
iv. Close cursor.(syntax: `CLOSE <cursor-name>;`)
   Example, `close cur_std;`

Attributes: Following are the attributes which are used to define variables of type—row and column.

1. **%type**—It is used to define a variable with the same datatype of any table's column/field. For example, in a SUPPLIER table there is a column named last_name. If we want to define a variable which should have datatype of this last_name column, we can write the following statement in the declare section of PL/SQL block.

   ```
   l_nm supplier.last_name%TYPE;
   ```

   It will define a variable named l_nm which will have datatype of the column last_name of table supplier.
2. **%rowtype**——It is used to define a variable of record type which matches any table's record. For example, to define a record type variable which should store records of supplier table, the following statement should be written in the declare section of PL/SQL block.

   ```
   Rec_sup supplier%rowtype;
   ```

   It will define a variable named rec_sup which will store one record of the table supplier.

Cursor Attributes: Cursor attributes start with symbol %. Following are the cursor attributes which are frequently used with cursor.

1. **%notfound**—It returns 'true' if there is not a single record in the cursor and returns 'false' if there is at least one record in a cursor.
2. **%found**—It returns 'false' if there is not a single record in the cursor and returns 'true' if there is at least one record in a cursor.
3. **%rowcount**—It returns total number of records fetched by the cursor.
4. **%isopen**—It returns 'true' if the cursor is open and 'false' if cursor is not open.

Cursor FOR loop: The **Cursor FOR loop** can be used to process multiple records. There are two benefits with cursor for Loop, these are as:

1. It implicitly declares a %ROWTYPE variable.
2. Cursor For Loop itself opens a cursor, read records then closes the cursor automatically. Hence OPEN, FETCH and CLOSE statements are not necessary in it.

   For example,

   ```
   Declare
       Cursor c1 is select * from emp;
       R1 c1%rowtype;
   Begin
       For r1 in c1 loop
         Dbms_output.put_line(r1.eid);
       End loop;
   End;
   ```

We have not used OPEN, FETCH and CLOSE in the above example as for cursor for loop does it automatically.

To update or delete rows, the cursor must be defined with the **FOR UPDATE** clause. 'For update of' clause locks the current row exclusively for updation. The Update or Delete statement must be declared with **WHERE CURRENT OF** clause.
For example,

```
Declare
   Cursor c1 is select * from emp for update of salary;
   r1 c1%rowtype;
Begin
   For r1 in c1 loop
        if r1.sal<20000 then
          Update emp set salary = r1.salary + 1000 where current
          of c1;
        End if;
   End loop;
End;
```

7.4 | PROCEDURES IN ORACLE

Procedure is a named PL/SQL block because it should contain some name. A procedure contains declaration and body part. Procedure can take parameters. Procedures which do not take parameters are written without a parenthesis. Following is the syntax to write a procedure.

```
CREATE OR REPLACE PROCEDURE
   <procedure-name>(<parameter 1> [in/out/in out] <datatype>, …….)
   AS <variable declaration>
BEGIN
   <procedure body>
EXCEPTION
   <Exception handlers>
END;
```

For example,

```
     create or replace procedure emp_age(sdt date, ldt date) as
        cursor c_emp is select fname, deptno,
        trunc(to_char(sysdate-bdate)/365.5)
        age from employee where bdate between sdt and
        ldt and rownum = 1 group by deptno, fname,
        trunc(to_char (sysdate-bdate)/365.5);
                r_emp c_emp%rowtype;
             begin
                open c_emp;
                loop
                fetch c_emp into r_emp;
                exit when c_emp%not found;
                dbms_output.put_line(r_emp.fname||' '||r_emp.
                deptno||' '||r_emp.age);
                end loop;
                close c_emp;
             end;
```

Parameters: Parameters are used to pass the values to the procedure. There are three types of parameters—IN, OUT and IN OUT.

> **IN parameter** is used to pass the values to the called procedure. It works as read only within the procedure.
>
> **OUT parameter** returns the value from the procedure.
>
> **IN OUT parameter** allows to pass and return values from the procedure. Default parameter type is IN.

A procedure can be executed by writing **EXEC <procedure_name>;** on the SQL prompt. It can also be executed by using a calling block. Following is a calling block which calls procedure named proc1.

```
BEGIN
  PROC1;
END;
```

7.5 | FUNCTIONS IN ORACLE

A **function** is a named block, which is used to compute a value. Function returns only one value. Following is the syntax to create the function.

```
CREATE [OR REPLACE] FUNCTION <function-name> ([pa-
rameter 1, parameter 2, …)] RETURN <datatype> IAS
  [variable declarations…]
BEGIN
  Executable statements
EXCEPTION
  Exception handlers
END;
```

For example,

```
Create or replace function cal(n1 int, n 2 int) return int as
  S int;
Begin
  S: = n 1 + n 2;
  Return S;
End;
```

The function is executed by writing the select statement on the SQL prompt or it can be called using a **calling block**. For example, select cal(20, 30) from dual will call the function call and will display sum of 20 and 30 which is 50.

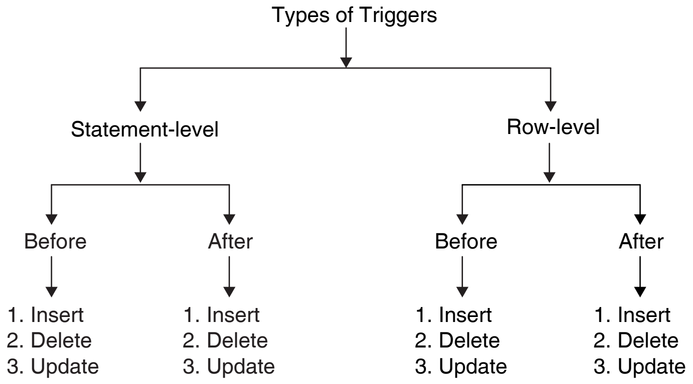


FIGURE 7.1 | Types of triggers.

7.6 | TRIGGERS IN ORACLE

Trigger is a **PL/SQL block** which is used to write validation rules on record when data is manipulated. Trigger is fired automatically depending on the manipulation operation.

Figure 7.1 shows different types of triggers.

Statement level triggers are fired for any DML operation. When we define statement level trigger, **'for each row'** statement is not written and 'old' and 'new' variables cannot be used. Statement level triggers are used when data are not manipulated.

Row level triggers are fired for each row which are affected by DML operation. We must write 'for each row' statement when we define row level trigger. 'Old' and 'new' variables can also be used with this type of trigger.

Each row and statement level triggers could be written before insert or delete or update and after insert or delete or update.

Following is an example of statement-level trigger.

```
create or replace trigger emp1 before insert on employee
            begin
               if (to_char(sysdate,'dy') = 'thu'
               or to_char(sysdate,'dy') = 'sun')
               then
               raise_application_error(-20501,'may not
               change employee table during the weekend');
               end if;
               if (to_char(sysdate,'hh24')<10 or
               to_char(sysdate,'hh24')> = 18) then
               raise_application_error(-20501,'may change
               employee table during working hours');
               end if;
            end;
```

Following is an example of row-level trigger.

```
create or replace trigger tr1_11 after insert on mark for each row
declare
    t int;
    p float;
    g char (1);
    status char (4);
begin
    t: = :new.mark 1 + :new.mark 2 + :new.mark 3;
    p: = t/3;
    if (:new.mark 1 < 36 or:new.mark 2 < 36 or:new.mark 3 < 36 )
    then
        status: = 'fail';
        g: = 'c';
    else
        status: = 'pass';
        if p> = 70 then
        g: = 'A';
    else
        g: = 'B';
    end if;
    end if;
    insert into result values(:new.stdno, t, p, status,g);
end;
```

Following is an example of **instead of trigger** which is written on a view instead of table.

```
create or replace trigger instr1 instead of insert on v2 for
each row
   begin
      if (substr(:new.itemname, 1, 1) = 's' or substr(:new.
      itemname, 1, 1) = 'S') then
      raise_application_error(-20000, 'Invalid item name... ');
      end if;
   end;
```

7.7 | OVERVIEW OF PACKAGES IN ORACLE

Package is a collection of functions and procedures. Package has two sections—package declaration and package body. Package declaration contains function or procedure declaration and package body contains body of functions or procedures.

SUMMARY

- PL/SQL is an extension of SQL, which is used to write small programs such as functions and procedures.
- Procedures and functions are named PL/SQL block.
- Procedure could be called through calling block or by writing 'exec proc_name' statement.
- Function could be called through calling block or by writing select statement.
- Function returns exactly one value.
- We can manipulate individual row by defining cursor.
- There are two types of cursors—implicit and explicit.
- The built-in cursor of oracle is known as implicit cursor and it could be referred with name SQL.
- Explicit cursor is a user-defined cursor which has some name.
- Cursor contains active recordset which is assigned some area in memory. Cursor is a kind of pointer which points to the current record in this recordset.
- Triggers are fired automatically when any DML statement is executed. Triggers could be defined on statement level or row level.
- Row level trigger is used when some data is manipulated.
- Each statement and row level triggers are written before insert or delete or update and after insert or delete or update.

EXERCISES

1. What is a cursor? Explain implicit and explicit cursors. With which name the implicit cursor is referred?
2. Explain the cursor attributes %type, %rowtype, %found and %notfound.
3. Write syntax of cursor for loop and give an example.
4. Differentiate between procedure and function.
5. What are 'in' and 'out' parameters?
6. Fill in the blanks.
   a. A function can return __________ value.
   b. To define a variable whose data type matches the data type of a table's field, the ________ cursor attribute is used.
   c. ____________ command is used to execute the procedure.
   d. PL/SQL is an abbreviation of _______________________.
   e. ___________ command retrieves the record from a cursor.

LAB ACTIVITIES

1. Create a procedure which will display the employees in descending order of employee name.
2. Write a procedure to accept start date, end date and subject code as inputs and print student attendance report in ascending order of student number with student name, total number of lectures attended of that subject and percentage.

3. Create a function that will return total number of employees whose joining date is between two inputted dates.
4. Create a function that will return age of a given employee.
5. Write a function which will take emp_id as an argument and returns a day on which employee was born.
6. Write a trigger that is fired after an insert statement is executed for the MARK table. The trigger writes the student's identification number, total, percentage, grade and status in the RESULT table. (Student's status is fail if he fails in any subject and pass if he gets more than 35 marks in each subject. Grade is A if percentage ≥70, B if percentage ≥60 and percentage <70, C if percentage ≥50 and percentage <60, D if percentage ≥40 and percentage <50 and E if percentage <40.
7. Write a function that will take department number as an input and will return employee name along with salary from EMPLOYEE table who earned maximum.
8. Write a PL/SQL block that will display student's result with total number of passed and failed students. Also display number of students who get distinction, first class, second class and pass class.

8 CHAPTER

Transaction Management in Database

CHAPTER OBJECTIVES

- Defining transaction.
- Understanding transaction properties.
- Knowing different states of transactions.
- Understanding concurrent execution of a transaction and problems which occur during concurrent execution.
- Identifying problem of deadlock.
- Learning backup and recovery procedure.
- Understanding importance of security, integrity and authorization in database transaction.

8.1 | INTRODUCTION

When the database contains large volume of data, which is shared among many users for reading and writing, it is very important that data is read and updated correctly during the transaction. **Transaction** should be completed successfully; otherwise, the database will contain inconsistent data.

Moreover, when the transaction fails due to some errors (application or system or another error), data should be recovered properly after the **failure**. In this chapter, transactions are discussed with recovery and security requirements. Also, the problems which may occur during the simultaneous execution of transaction and how they can be avoided are also discussed.

8.2 | DEFINITION OF TRANSACTION

When user reads/writes/removes data from/to the database (Tables), in simple language it is called transaction. To do data manipulation, SQL commands, *e.g.*, INSERT, UPDATE, DELETE and SELECT are used. Therefore, in other words, when we execute any data manipulation commands in database, it is called transaction. A single transaction can contain sequence of many data manipulation commands. In this case, the transaction is said to be completed successfully when all the commands are executed successfully and changes are recorded in the database permanently. This can be achieved by executing **'COMMIT'** command at the end of the transaction.

Because of some reason, if some of the statements are executed, but some are not, then updations are done partially in the database. It is called unsuccessful completion of transaction. In this case, the partial changes made should be undone and the transaction should be executed again. To undo the partial changes, which are made to the database, the **'ROLLBACK'** command is used.

The care for saving changes permanently (through COMMIT) and undoing the changes (through ROLLBACK) is taken by the system component **'transaction manager'** in most of the DBMS.

Figure 8.1 shows an example of a transaction in which there are two SQL statements which should be executed and committed together. The two statements are as follows:

```
insert into instalment values(yr, st, cl, amt, sysdate);
update student set remaining_amt = remaining_amt
where year = yr and stdno = st and trim(class ID) = cl;
```

The INSERT statement adds details of fees instalment paid and UPDATE statement subtracts the instalment amount from the remaining payable amount.

The procedure given in Figure 8.3 shows how fee-payment transaction process is done. The Tables involved in the transaction are STUDENT, INSTALMENT and FEES_TO_BE_PAID. To create these Tables in Oracle, the syntax is given in Figure 8.2. To execute and see the result of this procedure, some sample data is given in Table 8.1 FEES_TO_BE_PAID and

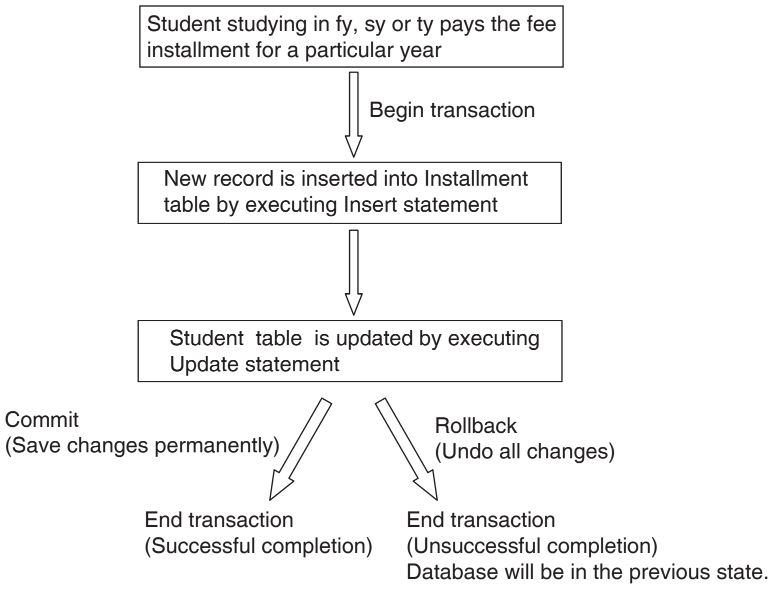

Student studying in fy, sy or ty pays the fee installment for a particular year
Begin transaction
New record is inserted into Installment table by executing Insert statement
Student table is updated by executing Update statement
Commit (Save changes permanently)
Rollback (Undo all changes)
End transaction (Successful completion)
End transaction (Unsuccessful completion) Database will be in the previous state.

FIGURE 8.1 | Example of a fees payment transaction.

```
create table fees_to_be_paid (year int, classid char(5),
annual_fees int)
create table student (year int, stdno int, classid
char(5),stdname char(30), remaining_amt int)
create table installment (year int, stdno int, classid
char(5), amt_paid int, inst_date date)
```

FIGURE 8.2 | Syntax to create Tables 8.1 and 8.2.

```
create or replace procedure calfees(yr int, st int, cl char, amt int) as
        cursor c_fees is select * from student where year = yr and
        stdno = st and
                trim (class ID) = cl;
        cursor c_inst is select count(*) m from installment
        where year = yr and stdno = st
                and trim(class ID) = cl;
        cursor c_anfees is select * from fees_to_be_paid
        where year = yr and
                trim(class ID) = cl;
        r_fees c_fees%rowtype;
        r_inst c_inst%rowtype;
        r_anfees c_anfees%rowtype;
begin
        open c_fees;
        open c_inst;
        open c_anfees;
        fetch c_inst into r_inst;
        fetch c_fees into r_fees;
        fetch c_anfees into r_anfees;
        if c_inst%notfound then
           insert into installment values (yr, st, cl, amt, sysdate);
           update student set remaining_amt = r_anfees.annual_fees-amt;
        else
            if r_fees.remaining_amt ≤ 0 then
                dbms_output.put_line (‘Fees already paid...’);
            else
                insert into installment values(yr,st,cl, amt,sysdate);
                update student set remaining_amt = remaining_amt-amt
                where year = yr
                               and stdno = st and trim(class ID) = cl;
            end if;
        end if;
        close c_fees;
        close c_inst;
        close c_anfees;
end
```

FIGURE 8.3 | Procedure to execute fee payment transaction.

Table 8.1 | Sample Data in Table FEES_TO_BE_PAID

Year	Class ID	Annual_Fees
2012	fybca	48000
2012	sybca	45000
2012	tybca	42000

Table 8.2 | Sample Data in Table STUDENT

Year	STD No	Class ID	STD Name	Remaining_AMT
2012	1	fybca	Sumeenkaur Rattan	48000
2012	2	fybca	Jimit Shah	48000
2012	3	fybca	Nikhil Chopra	48000
2012	4	fybca	Shubhangi Goel	48000
2012	5	fybca	Gurmukhsingh Jandu	48000
2012	6	fybca	Aditi Kothawala	48000
2012	7	fybca	Smit Shah	48000
2012	8	fybca	Rushang Shah	48000

Table 8.2 STUDENT and Data will be inserted automatically into the INSTALMENT Table when we will run this procedure by executing calling block given in Figure 8.4.

Student pays fees in instalments throughout the year. Here, a transaction means ‘payment of fees of each student in each instalment’. When student pays fees, records are selected from three Tables and two Tables instalment and student are updated using INSERT and UPDATE statements respectively. One transaction is said to be completed when these two statements are executed successfully. If, for some reason, any one of the statement is executed and another is not executed, the transaction will not be completed and the database will be in **inconsistent state**. This should not happen. Therefore, in this case, the whole transaction should be rolled back or should be committed. No intermediate state is possible, *i.e.*, **atomicity** should be maintained.

There are two commands ‘commit’ and ‘rollback’ which are used to save changes permanently and undo changes respectively. If we want to save part of transaction, we may use ‘savepoint’ command. The ‘savepoint <savepoint_name>’ command can be written anywhere in the process and later in the program ‘rollback to <savepoint name>’ command can be used which will undo the changes up to that specified savepoint. The changes which are done before that savepoint will remain as it is. They would not be rolled back.

For example, in the procedure as given in Figure 8.5, there is a savepoint ‘undo_payment’ which will undo the changes which are made after this savepoint, in case, if the instalment amount entered by the calling block is more than the fees to be paid. For example, if actual fees is ₹48,000 and, by mistake instalment amount entered in calling block is ₹4,80,000 (it means, whether we are paying more than the actual amount).

Thus, if the amount entered is more than the instalment amount, the changes done after savepoint will be undone.

```
declare
begin
    calfees(2012,8,‘fybca’,16000);
end
```

FIGURE 8.4 | Calling block to execute the procedure given in Figure 8.1.

```
create or replace procedure calfees (yr int, st int, cl char, amt int) as
   cursor c_fees is select * from student
   where year = yr and stdno = st and trim (class ID) = cl;
   cursor c_inst is select count(*) m from installment
   where year = yr and stdno = st and
                   trim(class ID) = cl;
   cursor c_anfees is select * from fees_to_be_paid
   where year = yr and trim (class ID) = cl;
   r_fees c_fees%rowtype;
   r_inst c_inst%rowtype;
   r_anfees c_anfees%rowtype;
   a int;
begin
   open c_fees;
   open c_inst;
   open c_anfees;
   fetch c_inst into r_inst;
   fetch c_fees into r_fees;
   fetch c_anfees into r_anfees;
   savepoint undo_payment;
   if c_inst%notfound then
         insert into installment values(yr, st, cl, amt,sysdate);
         update student set remaining_amt = r_anfees.annual_fees-amt;
   else
         if r_fees.remaining_amt<=0 then
              dbms_output.put_line('Fees already paid...');
         else
              insert into installment values (yr, st, cl, amt, sysdate);
              update student set remaining_amt = remaining_amt - amt
              where year=yr and
                         stdno=st and trim(class ID) = cl;
         end if;
   end if;
   select sum(amt) into a from installment
   where year=yr and stdno = st and
                trim(class ID) = cl;
   if a > r_anfees.annual_fees  then
       rollback to undo_payment;
   end if;
   commit;
   close c_fees;
   close c_inst;
   close c_anfees;
end
```

FIGURE 8.5 | Example of savepoint during transaction.

8.3 | PROPERTIES OF TRANSACTION

To ensure the successful execution of a transaction, the four properties of ACID (*i.e.*, Atomicity, Consistency, Isolation and Durability) should be satisfied. If not, then the database may contain inconsistent and incomplete data on failure of a transaction. The four ACID properties are explained below:

1. **Atomicity:** By atomic, it means that all the statements of a transaction are either successfully completed or rolled back, *i.e.*, in both the cases database should contain consistent data. If transaction is successful, then changes are saved permanently and if transaction

is not successful, all the changes should be undone. No intermediate state is possible. This is shown in Figure 8.1. COMMIT ensures that changes are permanently made into database and rollback insures that because of failure, the partial changes which are made should be undone and the database should be in the previous consistent state. The incomplete transaction could be executed later on. Oracle supports statement-level (any DML statement) atomicity.

2. **Consistency:** An execution of a transaction should keep database into consistent state. The consistent state may be the previous or the next one. Consider the following example of a transaction, in which there are two SQL statements which should be executed into sequence and committed together. It is explained in Figure 8.6.

```
 Insert into participant values (103,'Shefali Nik',
'DBMS23', '21-May-2013', '31-May-2013', 'HLICA', 2000.00);
 Update registration set total_fees = total_fees + 2000.00
 where workshop_ID = 'DBMS23';
```

   The example in Figure 8.6 shows that the database should be in consistent state in case of either successful execution or unsuccessful execution of a transaction. If it is partially committed or rolled back, the database will contain inconsistent data and will be in an inconsistent state, which should not happen.

3. **Isolation:** The transaction should be isolated from other transactions until its execution is completed, *i.e.*, when one transaction is updating the data, no other transaction can update that data until the first one releases the data. Isolation can be guaranteed with serializable execution of a transaction. If only single transaction is executed at a time,

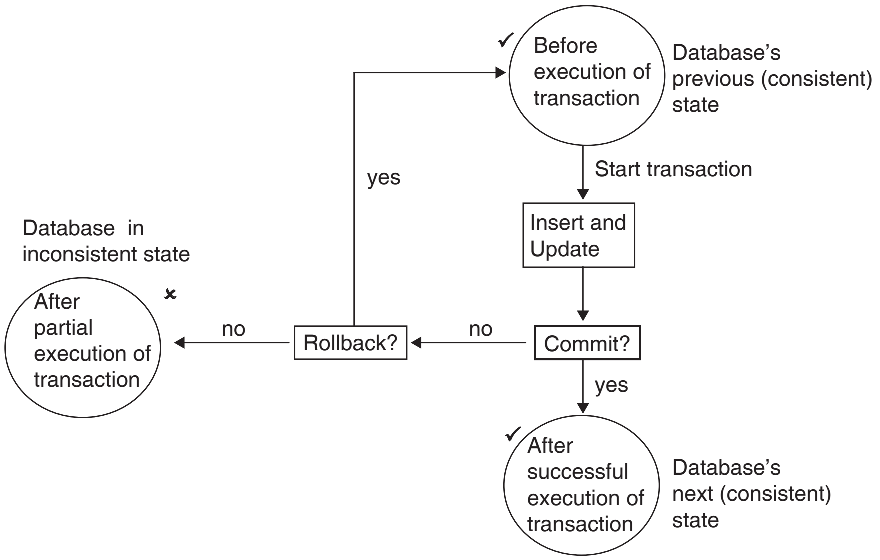

✓
Before execution of transaction
Database's previous (consistent) state
Start transaction
yes
Database in inconsistent state
Insert and Update
After partial execution of transaction
×
no
Rollback?
no
Commit?
yes
✓
After successful execution of transaction
Database's next (consistent) state

FIGURE 8.6 | Example of a transaction which should ensure consistent data after it is executed.

it automatically ensures the **serializability** and, hence isolation. But when many transactions are executed simultaneously, there are three types of interactions between transactions which are defined as follows:

i. **Dirty read:** When transaction reads uncommitted data, it is called **dirty read**.
ii. **Non-repeatable read:** When transaction reads data at different time which is committed by another transaction and view new data, it is called **non-repeatable read**.
iii. **Phantom read:** When a transaction runs query again and view newly inserted rows by another committed transaction, it is called **phantom read**.

There are four types of isolation levels:

i. **Read uncommitted:** Oracle does not allow **read uncommitted**, *i.e.*, dirty read.
ii. **Read committed:** It is the default isolation mode for Oracle database.
iii. **Repeatable read:** When in set transaction command isolation mode is set to 'serializable', Oracle supports it, otherwise not.
iv. **Serializable:** When in set transaction command isolation mode is set to 'serializable', Oracle supports it, otherwise not. The example to set isolation mode to 'serializable' is reflected as follows:

```
declare
      sno int;
  begin
      set transaction isolation level serializable;
      select count (std no) into sno from student;
      dbms_output.put_line ('Total students: '||sno);
      commit;
  end
```

In Oracle, we may set isolation level either 'serializable' or 'read committed'. The example of isolation level 'read committed' is given as follows:

```
declare
      sno int;
  begin
      set transaction isolation level read committed;
      select count (stdno) into sno from student;
      dbms_output.put_line('Total students: '||sno);
      commit;
  end
```

4. **Durability:** When transaction is completed and changes are made in database, the changes should be permanent or long lasting or durable. This property is called durability. Database's recovery system takes care of durability.

8.4 | STATES OF TRANSACTION

When a transaction has been started, it can be in different states, such as successfully completed (COMMIT), unsuccessfully completed (ROLLBACK), partially completed and running.

When transaction is started, it is in the **running or active state**, in which data is manipulated through data manipulation commands.

If there exists some error due to any reason, the transaction is stopped in between with partially updated data. Because of this, database will contain inconsistent data. Therefore, the changes made by the transaction should be undone through ROLLBACK statement. In this case, transaction is said to be in **unsuccessfully completed** state. To execute the transaction successfully, it should be restarted.

If all the changes are made successfully after execution of COMMIT statement, the transaction will be in the **successfully completed** state.

8.5 | CONCURRENCY CONTROL USING LOCKS

When more than one transaction is executed simultaneously at the same time and accessing data from the same database, it is called **concurrent execution** of a transaction. For example, there are three transactions which are started at the same time and accessing Tables from employee's payroll database is shown in Table 8.3.

During execution of a transaction, the following sequence of steps are done:

1. Reading data
2. Changing data
3. Writing data
4. Changes stored permanently (COMMIT), or changes are undone (ROLLBACK).

Each transaction follows the above-mentioned four steps. There are three problems which may occur during concurrent execution of transactions.

Table 8.3 | Example of Concurrent Transactions.

Time	Transaction 1	Transaction 2	Transaction 3
T1	`Insert into salary values (1002,'2-May-2013', 49000.00);`	`Insert into employee (empno, empname, joining_date, basic_salary) values (1345, 'P M Rana', sysdate, 23000.00);`	`Insert into loan_payment values(1022,'2-May-2013',10000.00);`
–	`Update comp_acct set amt = amt-49000 where sal_month = to_char( to_date ('2-May-2013), 'month');`	`Select count(empno) from employee;`	`Update loan_due set remaining_amt = remaining_amt-10000 where empno = 1022;`
–	`Commit;`	`Commit;`	`Commit;`

1. **Lost update:** Consider there are two transactions, T1 and T2, which are executed simultaneously. During execution, transaction T2 has updated some data which is read by transaction T1. Now, T1 has made changes in that data and committed (changes saved permanently). Later on, transaction T2 made changes in the same data and committed it. Therefore, the changes made by transaction T1 in the same data are overwritten by transaction T2. Thus, updation made by transaction T1 has been lost. This problem is called lost update problem. In short, when one transaction overwrites the data which is saved by another transaction, the problem of **lost update** occurs.
2. **Uncommitted data (dirty read):** Consider there are two transactions, T1 and T2, which are executed simultaneously. Transaction T2 is updating the data which is required by transaction T1. After transaction, T2 writes the data; transaction T1 reads that data and proceed. Then, transaction T2 is rolled back. So, the changes which are made are not committed. But transaction T1 has already read the uncommitted data and it is continuing with that data only. This problem is called **uncommitted data** which occurs due to dirty read.
3. **Inconsistent retrieval (unrepeatable read):** When one transaction is performing some summary on data and other transaction is adding or updating records which are being used in summary, the problem of inconsistent retrieval occurs, *i.e.*, when transaction reads inconsistent data, the problem of **inconsistent retrieval** occurs.

Due to above problems, it is very essential to control concurrent execution of transactions, otherwise the database will contain inconsistent data. To control concurrency, isolation (serial execution of transactions) is required. To avoid the above problems, locks are required.

Different isolation levels are already discussed in Section 8.3 . Oracle supports ‘serializable’ and ‘read committed’ **isolation levels**. Oracle implicitly provides appropriate **locks** on data depending on the isolation level. If isolation level is not defined by users, Oracle defines its default isolation level which is ‘read committed’. Additionally, in Oracle, we can set transaction as read only to maintain **read consistency**. An example of read-only transaction can be reflected as follows:

```
declare
     sno int;
  begin
     set transaction read only;
     select count (stdno) into sno from student;
     dbms_output.put_line('Total students:'||sno);
     commit;
  end
```

‘Set transaction read only’ allows executing only select queries, commit, rollback or DDL statements. The read only transaction is terminated when DDL statement, COMMIT or ROLLBACK is completed.

Locks: Lock is required when two transactions are executed simultaneously and using the same data and one of the transactions is updating the data. Conflict between transactions can occur if any one of the transaction is writing (changing) data. To solve this conflict, data should be locked for the transaction which is updating it and lock can be released after transaction is completed.

Table 8.4 | Example of Shared/Exclusive Lock

Data	Lock Type	Transaction T1	Transaction T2
Emp_salary	Shared lock is granted to both T1 and T2	Read	Read
Emp_salary	Exclusive lock for T2	Read	Write
Emp_salary	Exclusive lock for T1	Write	Read
Emp_salary	Exclusive lock will be granted to transaction which will request first to update emp_salary	Write	Write

Locks can be acquired on different levels of database such as database level, Table level, row level and field level. When lock is acquired on a particular level for a transaction, changes in that portion of database cannot be allowed by other transaction. If other transaction wants to read that data, then it may be allowed. It can be applied by **shared/exclusive lock**.

Shared/exclusive lock has two states—shared and exclusive. Shared lock is provided on the data when more than one transaction wants to read the same data. Exclusive lock is provided when more than one transaction are accessing the data and any one transaction wants to update the data. The example of shared/exclusive lock is given in Table 8.4.

Shared lock is provided to read data and **exclusive lock** is provided to write data. Oracle provides implicit shared/exclusive lock on data automatically when it is required for multi-user database.

Lock conversion: In shared/exclusive lock, **lock conversion** is also possible. When shared lock (read) is converted into exclusive (write) lock, it is called **lock upgrade**. When exclusive lock is converted into shared lock, it is called **lock downgrade**. Oracle supports lock conversion and does it automatically whenever required.

Lock escalation: When lock is upgraded from lower level to higher level, it is called **lock escalation**. For example, consider a Table EMPLOYEE. Suppose, transaction T1 wants to update the row of emp_no 203. Therefore, row-level lock will be provided to T1. Thereafter, T1 wants to update records of emp_no 345, 231 and 211. Again, row-level lock is provided for these three records to transaction T1. Later on, T1 request to update records of emp_no > 200. Now, as number of rows which should be updated is more, DBMS automatically decides to escalate lock from row-level to Table-level. Therefore, the lock will be provided on entire EMPLOYEE Table by releasing all the previous row-level locks which were acquired on EMPLOYEE Table. This is called lock escalation. Lock escalation decreases the number of locks, but increases restrictions on data. Oracle does not support lock escalation because it increases chances of **deadlocks**.

User can also define lock at transaction level in Oracle by using the following statements. Locks will be released once the transaction is completed.

1. Set transaction isolation level: Refer Section 8.3 for the detail of this statement.
2. Lock Table: It will lock the entire Table. For example, the following PL/SQL block will lock the Table student exclusively and will release the lock after execution of rollback statement.

```
declare
begin
    lock table student in exclusive mode;
    delete from student;
    rollback;
end;
```

Similarly, the following block will lock the Table in shared mode and will allow read records from student Table. The lock will be released after execution of commit statement.

```
declare
    c int;
begin
    lock table student in share mode;
    select count (*) into c from student;
    commit;
end;
```

3. Select….for update of it locks the field(s) exclusively which is to be updated. For example, the following statement will lock two fields, stdname and remaining_amt, of student Table for updation.

```
select * from student for update of stdname, remaining_amt;
```

In distributed database system, there are multiple **fragments** of database stored on different nodes. In this case, transaction updates multiple portions of database. The correctness of data is ensured using **two-phase commit** during transaction execution in distributed database. The two phases are 'prepare' and 'commit'. In prepare phase, the node which has started the transactions asks all the nodes other than the node who is responsible to execute commit, to get prepared for commit. If any of the node is not prepared, the transaction is rolled back. After all, the nodes agree to do commit, then in commit phase, the node responsible to execute commit will execute commit and after that the node which has started transaction tells other nodes to commit.

8.6 | DEADLOCKS

The lock which is forever permanent or which can never be released is called **deadlock**. Assume that there are two transactions which are running concurrently. During the execution, T1 is waiting for the data which is locked by T2 and T2 is waiting for the data which is locked by T1. In this case, none of the transaction will release the lock because none of the transaction is completed. Both will wait infinitely for data to be released by other transaction which will never happen. This situation is called deadlock. [A simple real time example of deadlock: Suppose, two friends are sharing ruler and eraser to draw a diagram. Currently, ruler and eraser are not occupied by any of the friend. During drawing, first friend needs an eraser, so he will use (lock) the eraser. Meanwhile, second friend needs ruler, so he will use (lock) ruler. Now, first friend needs ruler to continue drawing. But second friend will not give it, because he is using it currently. Later on, second friend needs eraser, but first friend will not give it as his drawing is

Table 8.5 | Example of Deadlock

Time	Transaction T1	Transaction T2
t1	Running	Running
t2	Running	Running
t3	Lock EMPLOYEE Table	Running
t4	Running	Lock SALARY Table
t5	Running	Running
t6	Request to Lock SALARY Table	Running
t7	Waiting	Running
t8	Waiting	Running
–	Waiting	Running
–	Waiting	Running
–	Waiting	Request to Lock EMPLOYEE Table Waiting
–	Waiting	Waiting
–	Waiting	
–	Waiting	Waiting
–	Waiting	Waiting

not completed and he is waiting for ruler. Thus, none of the friend will release the stationery and both will wait infinitely for other one to release the stationery. This situation is called deadlock.]

Table 8.5 shows an example of deadlock. In this example, both the transactions T1 and T2 start at time t1. At t3 time, transaction T1 requests a lock on Table EMPLOYEE and acquires the lock because EMPLOYEE Table is not locked currently. At t4 time, transaction T2 requests a lock on Table SALARY and acquires the lock because SALARY Table is not locked currently. At t5 time, transaction T1 requests a lock on Table SALARY, but does not acquire the lock and will be in wait mode because SALARY is being used by transaction T2 currently. At t8 time, transaction T2 requests a lock on Table EMPLOYEE, but does not acquire the lock and will be in wait mode because EMPLOYEE is being used by transaction T1 currently. Both the transactions will wait infinitely for other one to release a lock and will be in deadlock.

To end deadlock, one of the transaction should be aborted and restarted. DBMSs detect deadlocks automatically and roll back the transaction which causes deadlock.

To avoid deadlock, all the data which are required by the transaction should be locked in advance.

8.7 | DATABASE BACKUP AND RECOVERY

When some transactions fail before data are saved permanently, it causes loss of data. The data should be regenerated using some process or using previous data or related data which are stored into database. This process of regenerating data is called **recovery**.

By keeping in mind the failure of transaction and data loss, some precautionary steps can be taken by copying last updated data somewhere in memory or by keeping log of transaction for the reference is called keeping backup of data. The **backup** of data taken could be used in future to recover the lost data by executing some commands or procedures or simply by copying all the data again into database. Thus, backup means maintaining copy of data. Recovery means, restoring or regenerating the lost data from the backup taken or by applying some methods.

Failure can be due to **system crash**, or any **media crash**, or **application failure**. Failures may affect one transaction or all the transactions which are running. When failure occurs, transactions are partially completed and, therefore, when system restarts or application restarts the incomplete transactions should be rolled back and should be restarted. When system failure occurs, it destroys the data stored into **main memory** (buffer cache/buffer) which is volatile. Hence, the data stored in memory buffer which are updated by execution of a transaction are also getting destroyed.

At the time of system failure, the state of transaction during that period is not possible to know. May be updates of some transactions are already completed in buffer, but not written into database (data files) permanently. These transactions need not be undone when the system restarts; they need to be redone only.

As a solution to know the transaction state at the time of failure, **checkpoints** can be issued at some fixed interval of time, so that the transactions for which changes are stored in buffer can be redone; the transactions for which changes are not stored even in buffer can be undone and the transactions for which changes are stored in the database permanently need not be undone/redone as they are already completed. Thus, to recover the data, checkpoints can be used. Checkpoints could be kept at fixed time interval and in case of failure the last checkpoint could be used to decide which transaction should be redone/undone or should not be redone/undone.

Figure 8.7 shows that transaction T3 was completed successfully before failure and its updations are already reflected in the database; therefore, it should not be undone/redone when system restarts. Transaction T1 should be redone because it is completed and updations made by this transactions are stored in buffer, but still not reflected in database (*i.e.*, COMMIT is not

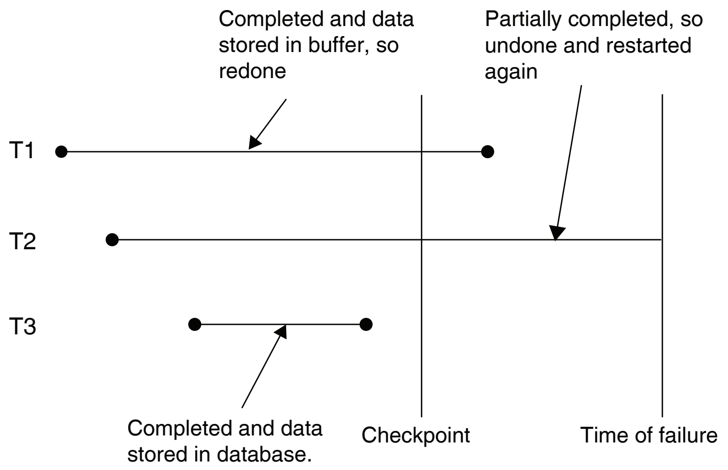

Completed and data stored in buffer, so redone
Partially completed, so undone and restarted again
T1
T2
T3
Completed and data stored in database.
Checkpoint
Time of failure

FIGURE 8.7 | Example of checkpoint.

executed). Transaction T2 should be undone and should be restarted because it was partially completed when failure occurs.

Oracle implicitly issues checkpoints depending on the internal settings. Checkpoints can also be defined explicitly in Oracle using the statement ALTER SYSTEM CHECKPOINT.

Also, to recover the lost data, backup should be taken on some separate devices in fixed time duration such as daily, monthly, yearly, etc. Backup can be taken online or offline. In Oracle, the online backup can be taken using the following statements.

```
ALTER TABLESPACE <tablespace_name> BEGIN BACKUP;
ALTER TABLESPACE <tablespace_name> END BACKUP;
```

In Oracle, flashback queries are also used to recover the data.

8.8 | SECURITY, INTEGRITY AND AUTHORIZATION

Security means, data protection. **Integrity** means, correct updations of data. **Authorization** means, preventing unauthorized users from accessing data. Security can be enforced at different levels such as user level, application level, object level, etc.

User can be allowed to access data by providing them username and password, which is called **authentication**. When user logs into the database using username and password, it is checked by the DBMS whether he/she is an authorized user to access the database or not.

After logging into the database, user cannot access all the data. To access the selected data or all the data, users are given privileges. The **privileges** can be given for creating objects, manipulating objects, deleting objects, execution and manipulating data, etc. Also, **roles** can be defined to assign common rights to a group of users. The types of data access are read only and read/write. Users can be provided access of limited fields of Table by creating views. **Views** can be defined to assign read only access as well as read/write access for some fields. We can also hide some fields from user by defining views. **Database triggers** can also be used to keep track of who is updating what. To see how privileges and roles are assigned to users in Oracle, refer Chapter 12. For more security of data, data can be encrypted which can be accessed only by authorized users.

Database integrity ensures the consistency of data by correct updation. Integrity can be enforced with the help of constraints and validation rules put up on the data.

SUMMARY

- Transaction is a group of statements which should be executed in sequence and should be committed together.
- Transaction must have four properties: ACIDS (Atomicity, Consistency, Isolation and Durability).
- Execution of a transaction should be atomic, *i.e.*, there should be only two possibilities—either successfully completed (COMMITTED) or completely unsuccessful (Rolled back), otherwise it can put database in an inconsistent state.
- Transaction should be isolated, *i.e.*, changes made by one transaction should be available to other transactions after it is fully completed (COMMITTED or ROLLED BACK).

Different DBMSs provide the facility to set isolation levels to ensure serial execution of transactions.

- Transaction should ensure consistency, *i.e.*, it should bring database from previous (last) consistent state to next consistent state or keep it in the previous consistent state only.
- Transaction should ensure durability, *i.e.*, once the changes are committed, system or device or any other failure should not affect the data.
- Database systems take care of starting and ending of a transaction, but user can also mark starting and ending of a transaction. Generally, transaction starts with execution of any SQL statement and ends with successful completion of that SQL statement or COMMIT statement or ROLLBACK statement.
- The transaction can be different states such as running (active), completed successfully (COMMITTED), completed unsuccessfully (ROLLED BACK) or completed partially. The transaction which is partially completed should be rolled back and should start again.
- In DBMSs, many transactions can access the data from same database simultaneously, which is known as concurrent transactions. The concurrent transactions should be controlled otherwise they will create problems such as lost update, uncommitted data and inconsistent retrieval. The concurrent execution of transactions can be controlled by setting isolation levels and by locking data which are being used by the transactions.
- There will be no problem if all the active transactions are reading the same data, but problem occurs when transaction is updating the data. During data updation, the data should be locked and should be released after transaction is completed.
- Data can be locked into two modes—shared and exclusive. Shared lock can be acquired to read data and exclusive lock can be acquired to update (write) data. Locking is implicitly done by DBMSs, but user can also lock data explicitly by issuing some commands.
- Shared lock can be converted into exclusive lock and exclusive lock can be converted into shared lock during execution of transactions. This is called lock conversion. Conversion from shared to exclusive lock is called lock upgrade and conversion from exclusive to shared lock is called lock downgrade.
- Locks can be escalated from lower level to higher level (*e.g.*, from field level to row level) which is known as lock escalation. Lock escalation increases restriction on data, but decreases number of locks.
- There are different levels of locking such as database level, Table level, record level, field level, etc. Assume that lock is provided to a transaction T4 on database level; then no other transaction can access the database until the transaction T4 gets completed. Similarly, if Table is locked by one transaction, other transactions can't access it until the first transaction gets completed.
- During concurrent execution, locking may generate deadlocks also. When two transactions are executed simultaneously and both are waiting for one another to release data so that they could complete the transaction, both will be in infinite waiting state which is called deadlock.
- To avoid deadlocks, the data required by the transaction should be locked in advance.
- To recover the lost data due to any failure, copy of the data is being kept timely which is known as backup.

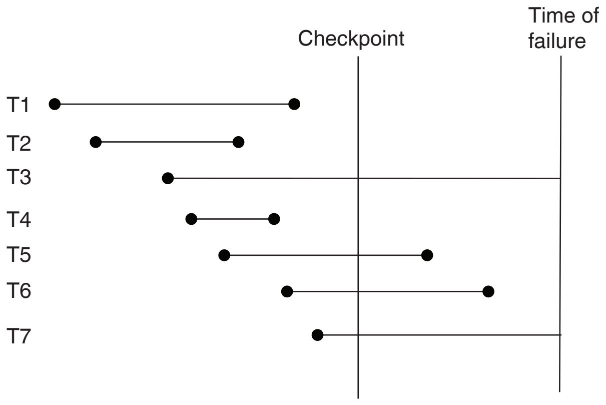
- Restoring the lost data using backup or any other method is known as data recovery. Checkpoints are used to recover the data in database. Checkpoint keeps track of transaction which can be used to know which transactions should be undone or redone in case of failure.
- Security means data protection. Integrity means correct updation of data. Authorization means preventing unauthorized users from accessing data.

Exercises

1. What is a transaction? Explain it by giving an example.
2. Discuss ACIDS properties of a transaction.
3. Discuss different states of a transaction.
4. Which of the two isolation levels does Oracle support? Explain them. Which isolation level is default in Oracle?
5. Define the following:
   a. Dirty read
   b. Phantom read
   c. Non-repeatable read
6. What does it mean by concurrent execution of a transaction? Which three problems can occur during concurrent execution?
7. Explain shared/exclusive lock.
8. What is the difference between lock conversion and lock escalation?
9. What is deadlock? Explain it.
10. Discuss backup and recovery, in your own words.
11. Explain importance of database security, integrity and authorization.
12. Select the correct answer from the multiple choices:
    a. From the following which one is not a property of transaction?
       i. Inconsistency ii. Durability
       iii. Atomicity iv. Consistency
    b. In ACIDS, A stands for ______________
       i. Authorization ii. Atomicity
       iii. Augmentation iv. Accessibility
    c. What is the significance of COMMIT statement?
       i. Undo changes ii. Save changes permanently
       iii. Restart the transaction iv. None
    d. When conflict for data occurs between two transactions?
       i. When both are reading the same data.
       ii. When both the transactions want to update same data.
       iii. When both the transactions want to update different data.
       iv. When both are reading different data.
    e. Shared lock is provided to ___________ data and exclusive lock is provided to _____________ data.
       i. Write, read ii. Read, write
       iii. Read, read/write iv. Write, read/write

f. What is lock escalation?
   i. Changing lower level lock to upper level.
   ii. Changing upper level lock to lower level.
   iii. Converting shared lock to exclusive lock.
   iv. Converting exclusive lock to shared lock.

13. Fill up 'yes' or 'no' values in the Table for the transactions given in the following figure. Sample values are given for transaction T1.



Transactions →	T1	T2	T3	T4	T5	T6	T7
Undone	No						
Redone	No						
Do not need to be redone or undone	Yes						
Changes made in database (COMMITTED or ROLLEDBACK)	Yes						
Changes made in buffer, but not in database (completed, but not committed)	No						
Changes not made in buffer or database (transaction is incomplete)	No						

9 CHAPTER

Centralized and Distributed Database Management System

CHAPTER OBJECTIVES

- Knowing different types of databases.
- Defining centralized and distributed databases.
- Understanding difference between centralized and distributed databases.
- Learning components of distributed database management system.
- Understanding distributed processing.
- Describing advantages and disadvantages of distributed data.

9.1 | INTRODUCTION

In present days, users are using the applications on the Internet and mobile phones, for which satellite networks and wireless networks are required. The amount of data, which is stored and used, is huge. It is impossible to keep these data on a single site. The data need to be stored at different sites, but the user should be allowed to access the data stored at any site. To implement this functionality, the distribution of data is required. Using the networking technologies, it is possible to access data stored at different sites geographically.

On the other hand, if data are stored at a single site and users can access the data from that single site, the concept is known as centralization of data.

9.2 | TYPES OF DATABASES

The databases can be broadly classified as Centralized and Distributed.

In **centralized database management system**, the database is located at a single (central) site whereas in **distributed database management system** (DDBMS), the database is distributed on different sites.

Database can also be referred as single-user and multi-user database. The database which allows only one user to access the database at a time, is called **single-user database**. The database, which allows many users to access the database at the same time, is called **multi-user database**.

9.3 | CENTRALIZED DATABASE MANAGEMENT SYSTEM VS. DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Centralized Database: In a centralized database management system, the entire database is located at a single (central) site, which is called **server**.

The server computer is connected with other computers through communication networks. Other computers can access the database, which resides on the server machine, by sending the request to the server.

The processing of the data may be executed at the server computer or at the local computer. If the local computer has the ability to process the data, which is retrieved from the server, the processing is called **distributed processing**. If the local machine cannot process the data, then data is processed on server itself, which is known as **centralized processing**. For distributed processing, the database need not be distributed.

Figure 9.1 shows the diagram of a centralized database with three computers connected through communication networks. The database named 'stu' is located at the site 'computer 1'. Users of 'computer 2' and 'computer 3' can access the 'stu' database from the machine 'computer 1'. After processing has been done on the data, the new data is stored again at the site 'computer 1'. The 'stu' database contains the tables, class, exam, student, faculty, attendance and result. All the tables are located at the site of 'computer 1'.

The advantages of centralized database are—it is easy to manage and maintain. It is cost effective. The disadvantages are—when server fails at central site, the computers at different locations cannot access the database. To overcome this problem, the mirror copy of the centralized database should be maintained at some other site which increases the overhead.

Distributed Database: In distributed database management system, the entire database is divided into **fragments** (parts). These parts are kept on computers at different sites. Sometimes, the copies of same database are also maintained on different sites and this process is known as **replication**.

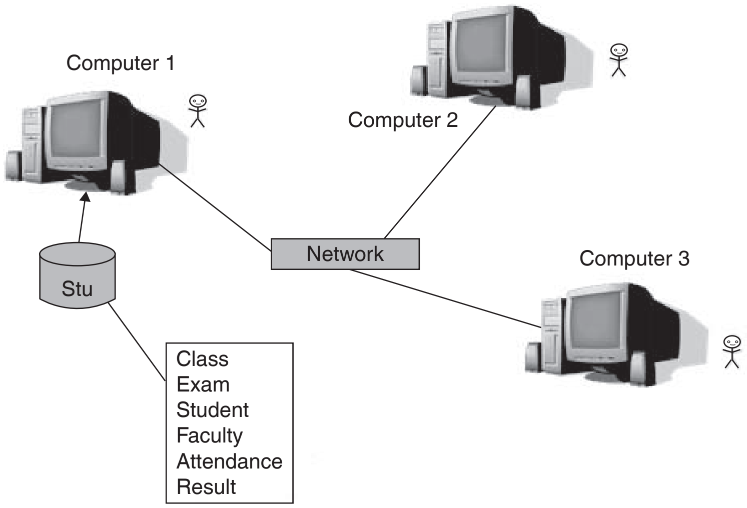


FIGURE 9.1 | Centralized database.

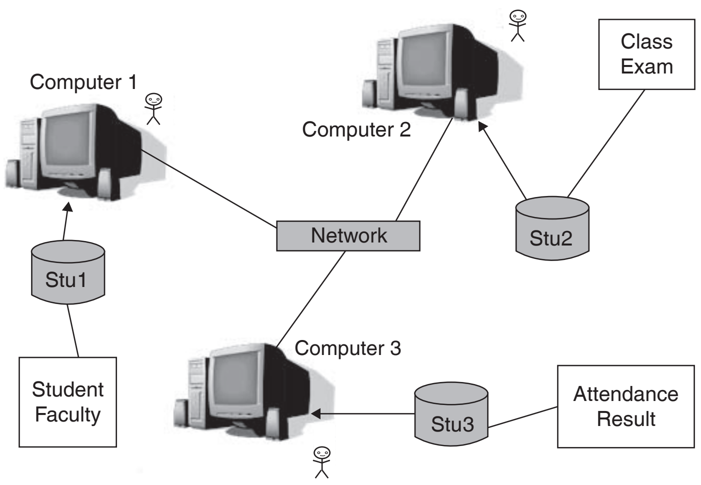


FIGURE 9.2 | Distributed database.

All the computers are connected with each other through a **communication network**. Each user from different computers can access the data stored at their own computer as well as computers located at remote sites. The processing can be done at any site.

Figure 9.2 shows the diagram of a distributed database with three computers which are connected through a communication network. The database named ‘stu’ is divided into three fragments named ‘stu1’, ‘stu2’ and ‘stu3’. The fragment ‘stu1’ is located at the site ‘computer 1’ which contains the tables of student and faculty; ‘stu2’ is located at the site ‘computer 2’ which contains the tables of class and exam; and ‘stu3’ is located at the site ‘computer 3’ which contains the tables of attendance and result. User at site Computer1 can access all the three database fragments stu1, stu2 and stu3 through communication networks. Same is applicable for user 2 and user 3.

If failure occurs at any computer in distributed database, then it will affect working of other computers in the network. The computer which has failed can be replaced with other computer.

Distributed database uses **client-server architecture** to process the data. It is possible that computers which are parts of distributed database may be of different configuration. They may use different hardware and software. It is possible that the database fragments stu1, stu2 and stu3 shown in Figure 9.2 may be implemented in different databases, operating systems, computers, etc. For example, assume that stu1 fragment is stored in DB2, stu2 fragment is stored in Oracle and stu3 fragment is stored in Ingres.

If failure occurs at any computer in distributed database, then it will affect working of other computers in the network. The computer which has failed can be replaced with other computer.

There are two types of distributed databases:

- **Homogeneous Distributed Database:** It is a type of distributed database in which computers on all the sites use the same database management software. In this type of distributed database, there will not be any problems regarding the database software as all the computers are using the same database software at different sites. Figure 9.3 shows an example of homogeneous distributed database in which databases at all the sites are stored in the same database management software, Oracle.

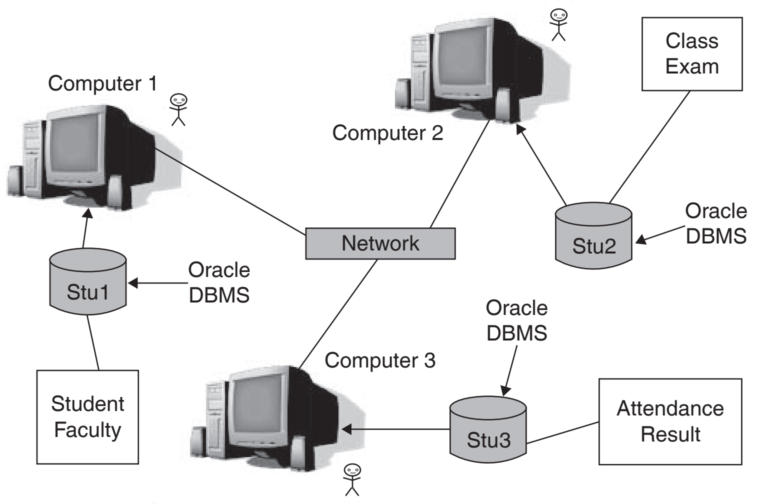


FIGURE 9.3 | Homogeneous distributed database which uses Oracle DBMS at different sites.

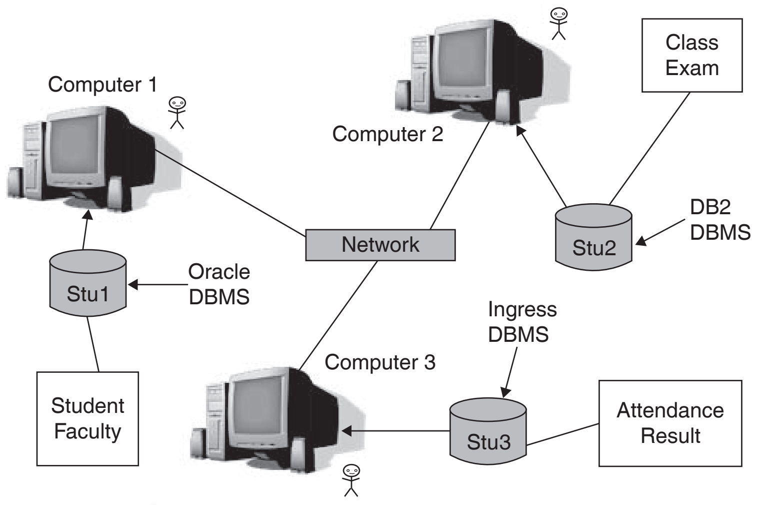


FIGURE 9.4 | Heterogeneous distributed database which uses Oracle, DB 2 and Ingress DDBMSs at different sites.

- **Heterogeneous Distributed Database:** It is a type of distributed database in which computers on all the sites use different database management software. Figure 9.4 shows an example of heterogeneous distributed database in which databases at all the sites are stored in the different database management software. The database at site Computer1 is stored in Oracle DBMS, database at site Computer 2 is stored in DB2 DBMS and database at site Computer 3 is stored in Ingress DBMS.

In DDBMS, data are replicated or fragmented as per the need. Data fragmentation means database is divided into parts (fragments) and these fragments are stored at different sites. Data replication means copies of database fragments are created on the sites where it is required.

9.4 | DDBMS COMPONENTS

Distributed Database needs the following hardware and software components:

Computer System: Distributed DBMS needs computers at different sites to store fragments of database, to access data from database fragments stored at other sites and to process the data. Distributed DBMS must be independent of hardware, *i.e.*, it should support different hardware (Computer System) at different sites.

Communication Network: It is required to transfer data from one site to other. Distributed database should provide network transparency, *i.e.*, it should support different types of communication networks.

Data Processor: Data Processor is a software component which is required to access from and store data at different sites. Data Processor is also known as **Data Manager**.

Transaction Processor: Transaction Processor is also a software component which processes the transaction, *i.e.*, it processes the retrieved data at local site. Transaction Processor is also known as **Transaction Manager** or **Application Processor**.

Figure 9.5 shows the components of DDBMS.

9.5 | DISTRIBUTED PROCESSING

When the database is distributed, it requires distributed processing. It means that computers at each site must be able to process the data, *i.e.*, processors are required at each site. If there is no support for distributed processing, the database cannot be distributed. However, it is possible that if the database is not distributed, the processing can be distributed.

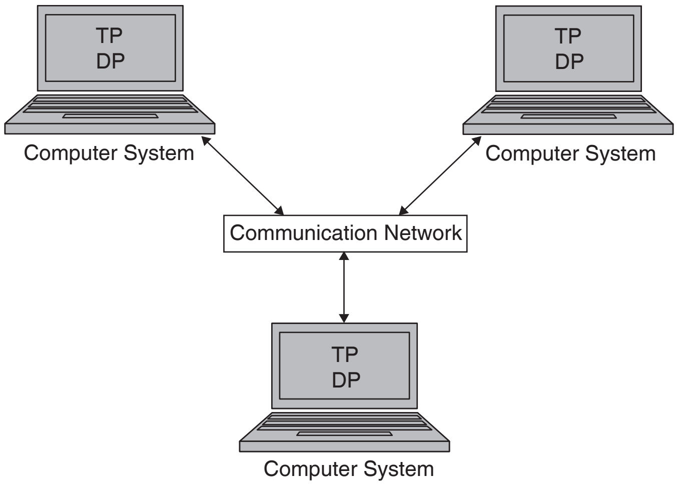


FIGURE 9.5 | Components of distributed database management system.

For both distributed database and distributed processing, the communication network is required.

9.6 | DDBMS ADVANTAGES AND DISADVANTAGES

Advantages of DDBMS: Following are the advantages of DDBMS:

- User does not know the things, such as at which site which data is stored, whether database's fragments are created or database's copies are maintained, which network technology is used to share data, etc.
- Data can be shared using common communication network.
- Failure of any site will not affect other sites. So, it is more reliable and it will give better performance than centralized database.
- Computers can be added and removed easily from the network.
- Easy to maintain as database is distributed among sites.
- It provides data independence, hardware independence, network independence, DBMS independence, fragmentation independence, replication independence, location independence, etc.

Disadvantages of DDBMS: Following are the disadvantages of DDBMS:

- It is more complex because of independence provided at different levels.
- It increases cost because maintenance and security of each individual site is required.
- As data is replicated and shared, the data integration issues occur.
- Problems may occur during concurrent transaction processing.
- When different hardware and software are implemented at different sites, it is difficult to handle them technically.

SUMMARY

- There are two types of databases: centralized and distributed.
- In centralized database management system, the database is located on a single site; whereas in distributed database management system, the database is distributed in parts and theses parts are stored on different sites. All the sites are connected through communication protocol.
- There are two types of distributed databases—homogeneous and heterogeneous. In homogeneous distributed database, computers on all the sites use the same database management software. In heterogeneous distributed database, computers on all the sites use different database computers.
- Distributed database requires distributed processing. Distributed processing means that processing is done on different sites.
- The components of distributed database are transaction processor, data processor, computer system and communication network.
- Transaction processor is a software which processes data on different sites. Data processor is software which accesses data from remote/local site and stores data at remote/local site. Computer system is a hardware, on which database fragment, transaction processor

and data processor are stored. Through communication network, the computers at different sites communicate with each other.

- The advantages of distributed database are data sharing, failure of one site does not affect other sites, computers can be added or removed easily, easy to maintain and provides independence at different levels.
- The disadvantages of distributed database are complexity, increased cost, data integration issues, problems in concurrent transaction processing, difficult to handle different hardware and software at different sites.

Exercises

1. Differentiate between centralized and distributed databases.
2. Explain homogeneous and heterogeneous distributed databases with diagrams.
3. Prepare a list of advantages and disadvantages of distributed database.
4. Explain components of a distributed database system.
5. What is distributed processing?
6. Name some database management systems which provide distributed database functionalities.

10 CHAPTER

Advancements in Databases

CHAPTER OBJECTIVES

- Knowing multidimensional databases.
- Understanding mobile databases.
- Knowing multimedia databases.
- Learning concept of data mining and data warehousing.
- Defining open source databases.
- Understanding spatical databases.
- Knowing moving object databases.
- Understanding NoSQL databases.
- Summary

10.1 | MULTIDIMENSIONAL DATABASE

Multidimensional database is a special type of relational database management system (RDBMS) which supports data to be stored and retrieved as multiple dimensions. The simple relational database model supports only two dimensions—rows and columns. The data within multidimensional databases are stored as a cube.

Multidimensional database represents the aggregated data for each of its dimension. The aggregate values are computed from the base values. Multidimensional databases are useful in **On-Line Analytical Processing (OLAP)**. Microsoft SQL Server Analysis, Oracle's Hyperion Essbase and Oracle 11g OLAP option are examples of multidimensional database. Figure 10.1 shows an overview of Multidimensional database.

10.2 | MOBILE DATABASES

The database which is connected with the mobile device is known as **mobile database**. Many mobile applications are based on the following three architectures:

1. **Mobile Client-Fixed Host architecture:**
   - In this, **Host** (Data Service Provider) will provide data to the **mobile client** (device) (*e.g.*, users' personal data such as bank account details). Another type of host is **servers** which broadcast information (*e.g.*, broadcast servers which provide weather information).

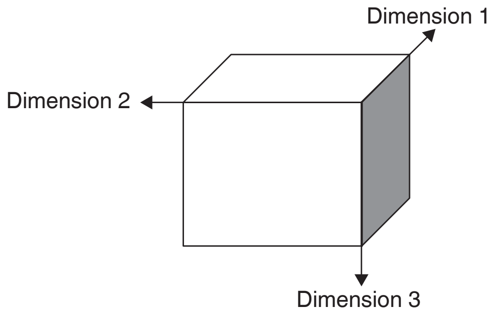


FIGURE 10.1 | Multidimensional database.

- The main copy of the data is managed by the server, while the **replica** of data is managed by user's mobile device.

2. **Mobile Client-Mobile Host:** In this case, the database is embedded in the user's mobile device. For example, phonebook, multimedia data (photos, videos, etc.) in the SIM card.
3. **Fixed Client-Fixed Host:** In this case, the applications may use mobile database to keep track of moving objects. For example, to keep track of fleet during transportation, traffic database, etc.

The mobile database can be designed for light-weighted devices such as PDAs and mobile phones. The database can also be designed for smart cards. To process the data stored on a smart card, the smart card should be connected with the host computer. The host computer contains the application through which user can interact with the card.

Some examples of mobile database are SQLite (developed by D. Richard Hipp), SQL Anywhere (provided by Sybase iAnywhere), IBM DB2 Everyplace, Oracle Lite, Microsoft SQL server CE, etc.

Figure 10.2 shows an overview of how data is accessed through **mobile application** from mobile database. The primary data resides on the main database which are accessed through mobile application and copied on to mobile device.

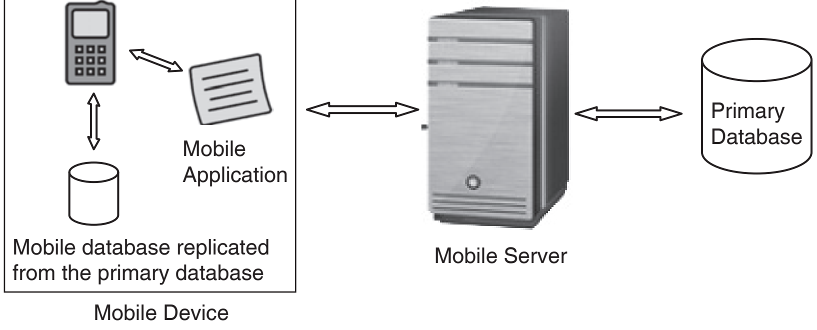


FIGURE 10.2 | Mobile database.

10.3 | MULTIMEDIA DATABASES

The database which can store data such as images, video, audio, graphics, animation, text, etc., is known as **multimedia database**. The multimedia database should be able to store huge amount of data and should provide multimedia data which are widely used in applications, such as healthcare, distance learning, web applications, weather forecast, etc. Also, multimedia database is capable of storing metadata related to multimedia data.

While designing multimedia database the volume, complexity and nature of data needs to be considered. In a multimedia database, inputs come from different devices, such as CCTV camera, digital camera, mobile phones, fingerprint readers, barcode readers, biometric scanners, audio and video files, microphone, etc.

Similarly, the stored data are retrieved from the database and displayed on screen or any other output device.

The multimedia data can be stored in a relational database or **object-oriented database**. It depends on the nature of data, that is, the particular data model should be used to store multimedia data.

Oracle provides the feature 'Oracle Multimedia' (formerly Oracle interMedia) that enable user to store and retrieve multimedia data from heterogeneous environments. It also provides integration of such data.

10.4 | DATA WAREHOUSING AND DATA MINING

Data Warehousing: The relational database can store limited data into related tables which are normalized, while **data warehouse** is used to store large amount of data.

Data stored in data warehouse are accessed from multiple **heterogeneous databases** and stored in the uniform format. Once data are stored into data warehouse, they are not updated or changed (*i.e.*, read only). Users can only retrieve the stored data from warehouse.

Data warehouse is used to store enterprise's historical and summarized data which can be used for future analysis and to take decisions. Before data is stored into warehouse, inconsistent data are removed. Data warehouse contains Server, **OLAP (On-Line Analytical Processing) engine** and Client. Data are accessed from various heterogeneous databases and stored into data warehouse server after **cleaning**. Client contains various tools for data processing, such as reporting tools, data mining tools, etc. Figure 10.3 represents data warehouse system in which data are stored from various sources.

Some of the popular data warehousing tools are Oracle Data Integrator, MS-SQL Server's Integration Services, IBM Cognos Data Manager, etc.

Data Mining: The analysis which is done on static data stored in data warehouse, is known as **Data Mining**. It is also known as **Knowledge Discovery in Database (KDD)**. Data mining is used to discover hidden patterns from large volume of data. For example, in real life, there are many hidden patterns which are still not known to human, such as patterns found in stars in the sky; patterns in the DNA of human, animals and birds; patterns lies in nature which cause natural calamities, etc. If we store and analyze historical data about all these things, we would get some amazing information. To analyze these data, we need powerful methods, tools and formats.

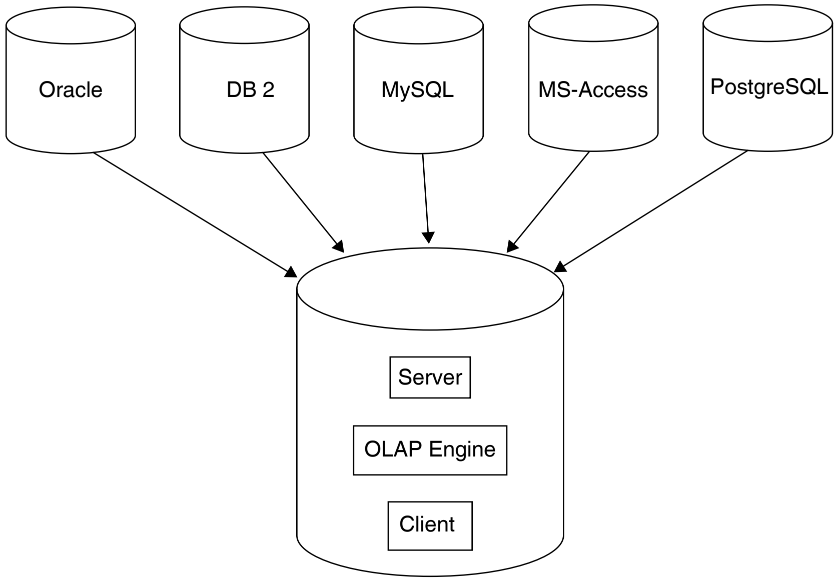


FIGURE 10.3 | A data warehouse.

There are many data mining methods which can be used to discover patterns from stored data, such as structured data analysis, classification, association rule learning, cluster analysis, decision tree, neural networks, regression analysis, etc. Mining can be done on text data, multimedia data, web data, etc.

There are many tools available for data mining which also includes **open source tools**. Some of these tools are Microsoft SQL Server Analysis Services, Weka, RapidMiner, GeoDa, AR-Miner, etc.

10.5 | OPEN SOURCE DATABASE

Open source database means, the database whose source code is open to all. Anyone can download the source code and use it. There is no need to purchase license to use that database. Moreover, users can modify its source code and can redistribute it after modification.

MySQL, PostgreSQL, etc., are some very good open source databases which provide features of RDBMS.

In open source environment, the warranty of product and support is not provided.

10.6 | SPATIAL DATABASES

Spatial (pictorial/image) database is used to store geometrical or space related data, such as line, region, different shapes, etc. It represents the object defined in a geometric space. Special kind of spatial queries are used to store geometrical shapes.

The main application of spatial database is GIS (Geographical Information System) which is used to store and display geographical information on computers.

10.7 | MOVING OBJECT DATABASES

The moving object database enables user to store data related to any moving object. There are two approaches of moving object database: location-based and spatio-temporal.

In location-based approach, to store the moving object, location of the object is required. The location of moving object gets changing with time. The updations to database are very important in case of moving objects. To keep data of moving object the direction in which it is moving, speed of movement, size of object, etc., should be considered besides the location.

In the location-based moving object database, the history of object is not kept. Only the current position is stored.

The moving object database can be used to keep track of vehicles, criminals, land movements, animals, etc.

To store moving objects, special kind of query language is required. TSQL2 is a Temporal Structured Query Language which is used to store data related to moving objects.

10.8 | NoSQL DATABASE

Social media such as facebook, twitter, etc., generates bulk volumes of data across the Internet. These data are unstructured. Relational database management systems are very efficient in storing **structured data**, but to store **unstructured data** some special kind of databases are required.

In relational database, **schema** (table structure using CREATE TABLE) should be defined before data is stored into table. It can store similar types of data within a table. If different types of data are to be stored in the same table, it requires schema to be modified (using ALTER command). Changing of schema definition rapidly as per the requirement is known as **agile development**. SQL (relational) databases fail to cope up with frequently changing unstructured requirements. To cope up with these changing requirements, **NoSQL database** is evolved.

NoSQL means, not only SQL. NoSQL database is non-relational, distributed and open-source database. NoSQL database is schemaless. It does not require schema to be created before storing the data. NoSQL databases are based on document store, graph store, wide-column store, key-value/tuple store, etc. NoSQL database is used for very large sets of **distributed data**. It spreads data automatically on servers and, also balances them. It replicates data automatically across network. It keeps frequently used data in system's cache memory.

NoSQL database which is based on **key-value** contains only two columns—'key' and 'value'. The actual information is stored within the 'value' column.

NoSQL database which is based on 'document store' keeps **key-document** pair which stores all the information in a single document in JSON (Java Script Object Notation), XML (Extensible Markup Language), etc., data interchange formats, which can contain hierarchical values or nested documents.

Graph store NoSQL databases are used to store network information such as social connections.

Wide-column NoSQL database stores columns of data instead of records, *i.e.*, it adds columns dynamically.

Some examples of NoSQL database are Hadoop, MongoDB, Redis, Apache Cassandara and Hbase.

SUMMARY

- To store and retrieve data as multiple dimensions, the multidimensional database is used. Multidimensional database represents the aggregated data for each of its dimension.
- The database which is connected with the mobile device is known as mobile database.
- The database which can store data such as images, video, audio, graphics, animation, text, etc., is known as multimedia database.
- Data warehouse is used to store enterprise's historical and summarized data which can be used for future analysis and to take decisions.
- The analysis which is done on static data, stored in data warehouse, is known as Data Mining. It is also known as Knowledge Discovery in Database (KDD). Using data mining, hidden patterns can be found from the data.
- Open source database means the database whose source code is open to all. Anyone can download the source code and use it.
- Spatial (pictorial/image) database is used to store geometrical or space-related data, such as line, region, different shapes, etc.
- NoSQL database means, not only SQL database which does not require schema to be created before storing the data. NoSQL databases are based on document store, graph store, wide-column store, key-vale/tuple store, etc.

EXERCISES

1. Select the correct answer from the following multiple choices:
   a. The database which stores data as many dimensions is called __________ database.
      i. Multidimension
      ii. Single Dimension
      iii. Multi
      iv. Mini
   b. OLAP means ______________ .
      i. On-Line Application Processing
      ii. On-Line Analytical Processing
      iii. Off-Line Application Processing
      iv. Off-Line Analytical Processing
   c. Oracle Lite is an example of ____________ database.
      i. Mobile
      ii. Multi

   iii. Hierarchical
   iv. Network

d. _______________ architecture of mobile database is used to keep track of moving objects.
   i. Mobile Client-Fixed Host
   ii. Mobile Client-Mobile Host
   iii. Fixed Client-Fixed Host
   iv. All of these

e. The database, which can store data such as images, video, audio, graphics, animation, text, etc., is known as multidimensional database.
   i. True
   ii. False

f. Oracle Data Integrator is an example of _________________ tool.
   i. Data mining
   ii. Data warehousing
   iii. Spatial database
   iv. None of the given

g. Weka is an example of _________________ tool.
   i. Data mining
   ii. Data warehousing
   iii. Spatial database
   iv. None of the given

h. In the following list, which one is not a feature of open source?
   i. Source code is not available
   ii. License to use a software is not required
   iii. Source code can be modified
   iv. Support is not provided

i. Which of the followings is an example of open source database?
   i. Oracle
   ii. MS Access
   iii. MS SQL Server
   iv. PostgreSQL

j. Which database is used to store geometrical or space-related data, such as line, region, different shapes, etc.?
   i. Spatial database
   ii. NoSQL database
   iii. Network database
   iv. Hierarchical database

k. In the ____________ moving object database(s), the history of object is not kept.
   i. location-based
   ii. spatio-temporal

iii. both (i) and (ii)
iv. none

l. NoSQL database is ______________.
   i. schema-based
   ii. schemaless

m. Hadoop, MongoDB, Redis, Apache Cassandara, Hbase are examples of:
   i. Relational database
   ii. Hierarchical database
   iii. Network database
   iv. NoSQL database

2. Which features differentiate NoSQL database with relational database?
3. Discuss the architectures on which mobile applications are based.
4. List down the features of open source database.
5. Write short notes on the followings:
   a. Multidimensional database
   b. NoSQL database
   c. Open source database
   d. Spatial database
   e. Moving object database

11 CHAPTER

Overview of MS-Access 2007

CHAPTER OBJECTIVES

- How to use MS-Access.
- Knowing elements of MS-Access.
- Managing data using form.
- Displaying data using reports.
- Designing and writing queries.
- Creating macros.
- Creating switchboard.

11.1 | MS-ACCESS AS AN RDBMS

MS-Access is a Relational Database Management System (RDBMS) which allows us to define and manage relationships between tables using primary and foreign keys.

Using MS-Access, we can write queries, create forms to manage data, create reports, write macros to execute list of instructions in a sequence, etc. These elements are created within Access database files.

We can use various controls to design forms and reports. The queries, tables, forms and reports are imported and exported from/to another Access database.

11.2 | ELEMENTS OF MS-ACCESS

Following are the elements of MS-Access database.

Tables: A table contains information about entity such as student, exam, item, bill. Table is a collection of related record. Each record contains details of one entity occurrence. For example, a student table contains records of various students. Each record represents distinct student. Each record contains many fields. Field describes the characteristics of an entity. For example, each student record contains fields such as stdno, stdname, birthdate, which describe each unique student. Record is also known as row and field is known as column.

Queries: A query is used to create subsets of any table. According to the table and criteria defined in a query, the result is generated. Query will display records from tables or existing queries in MS-Access. We can store query as an independent object which can be executed at any time. It will display up-to-date result whenever it is executed.

Forms: A form is an independent object which could be used to manage data stored in a table. Using forms, user can insert, delete, update, view and navigate records. Besides this, user can validate data and restrict data using form. We can also create a special type of form named switchboard, using which links can be provided to access any other database object such as tables, queries.

Reports: A report is an object which displays data. The data are fetched from tables or queries. Users cannot change the fetched data; they can only view it.

Macros: A macro is an object which contains list of instructions to be executed in a sequence. It helps user to automate any task like opening report of employee and displaying only records with salary amounting less than ₹10,000.

Modules: It is a type of program which an user can write.

11.3 | CREATING DATABASE AND TABLES

Field Naming Convention

In MS-Access, there are some rules to define field names. When we give a field name in MS-Access 2007, we should follow the rules as given below:

1. The field name should not exceed 64 characters.
2. Leading space (the space which is given in the starting of a field name) is not allowed in a field name.
3. Letters, Numbers and Spaces (not leading space) are allowed.
4. The characters, such as period (.), Exclamation Mark (!), Square brackets [] and non-printable characters such as carriage return (enter key) are not allowed.
5. Reserve words and keywords are not allowed as a field name. (Reserved words can never be used as identifiers. Keywords can be used as identifiers, but this is not recommended.)
6. Field names are not case sensitive.

Creating a Database

There are two ways to create a database:

1. Creating a blank database.
2. Creating a database using a template.

To create a blank database, do the following:

Step 1: Open MS-Access. Select Blank Database option from the new blank database pane. If any MS-Access is already open, then to create a new database click on the Microsoft Office button and choose New. Then select the Blank Database option.

Step 2: The blank database pane opens where user can enter name for the new database and click on the folder icon to select the path where user wants to store the database.

Step 3: Click on Create.

To create a database using a template, do the following:

Step 1: Open MS-Access. Select any database from the pane 'featured online Templates'.

Step 2: Give the file name, select path where you want to store the database.

Step 3: Click on Download.

Creating Tables

There are three ways to create a table:

1. Creating a table using design view.
2. Creating a table using datasheet view.
3. Creating a table using a template.

To create a table using datasheet view, do the following:

Step 1: Click on Create menu given on a ribbon. From 'Table' group, select table option. It will open a blank table in a datasheet view with one field ID. We can change it by double clicking on the field name. Similarly, we can add more fields by pressing a table.

Step 2: To set the data type and formats for the fields, click on the datasheet tab given in the menu. Select Data type and Formatting group. Change data type and format by clicking on the down arrow.

Step 3: Right click on the table name to save it with a new name.

To create a table using design view, do the following:

Step 1: Click on Create menu on a ribbon. From 'Table' group, select table design option. If we select table option, then the blank table will be opened in a design view.

Step 2: Give field names and data types and set the field properties.

Step 3: Select the fields which you want to define as a primary key and click primary key option of 'tools' group in a design tab.

Step 4: Right click on table name to save it with a new name.

To create a table using a template, do the following:

Step 1: Click on Create menu on a ribbon. From 'Table' Group, select table templates option.

Step 2: Select the table out of five given templates—contacts, tasks, issues, events and assets.

Step 3: Make changes in a field as required and save it.

Table 11.1 shows the difference between Design view and Datasheet view of a table:

Table 11.1 | Difference Between Design View and Datasheet View

Design View	Datasheet View
We can set primary key using this view.	We cannot set primary key using this view.
We cannot enter records using this view.	We can enter records using this view.
We cannot change formatting of text entered in a table using this view.	We can change formatting of text entered in a table using this view.
We can set the field properties such as validation rule, default, validation text, indexed, by using this view.	We cannot set the field properties such as validation rule, default, validation text, indexed, by using this view.

Navigation Pane: The left pane in the MS-Access is called Navigation Pane. It displays all the objects such as tables, queries, forms, reports, macros, created by the user. User can navigate through this pane to open any existing object.

11.4 | DATA TYPES OF MS-ACCESS

MS-Access 2007 provides 10 different data types, and each type has a specific purpose. Table 11.2 lists all the data types.

Changing from one data type to other data type:

- Change data types in Datasheet view
  1. Opens the table in Datasheet view.
  2. Select the field (the column) for which you want to change the data type.
  3. On the Datasheet tab, in the Data Type & Formatting group, click the arrow in the drop-down list next to Data Type, and then select a data type.
  4. Save your changes.
- Change data types in Design view
  1. Open table in Design view.
  2. Select the field (the column) for which you want to change the data type, and select a new data type from the list in the Data Type column.
  3. Save your changes.

Table 11.2 | Data Types of MS-Access 2007

Data Type	Used to Store	Maximum Limit
Text	Alphanumeric data	255 characters
Memo	Alphanumeric data	65,535 characters
Number	Numeric data	Field size could be set to 1, 2, 4, 8, or 16 bytes
Date/Time	Dates and times	
Currency	Financial data	8 bytes with four decimal places
Auto Number	Unique values created by MS Access for a new record	4 bytes
Yes/No	Boolean (true or false) data	−1 for all Yes values and 0 for all No values
OLE Object	Images, documents, graphs and other objects	Up to 2GB of data (the size limit for all Access databases)
Hyperlink	Web addresses	Up to 1 gigabyte of data
Attachment	Any supported type of file	

- Restrictions on changing data types
  As a rule, we can change the data type of all fields, except for:
  - Number fields with the Replication ID property enabled
  - OLE Object fields
  - Attachment fields

  When we change data type of the field which contains data, Access truncates or deletes some data or may not allow convert the data type.

Field Properties: Each data type has some properties:

1. **Number field properties:** Table 11.3 shows 'Number' field properties.
2. **Text field properties:** Table 11.4 shows 'text' field properties.

'Format' property: We may use some predefined Access formats or may define our own formats by using some specific characters:

- Table 11.5 shows predefined formats for the Number, AutoNumber and Currency data types.
- Table 11.6 shows predefined formats for the date/time data type.

Applying custom (user defined) formats to numeric data: We can also apply custom formatting by using some characters.

Input Mask: The following table lists and describes the placeholder and literal characters that you can use in an input mask. Table 11.7 shows input mask characters.

Table 11.3 | 'Number' Field Properties

Property	Description
Field Size	Controls the size of the value that you can enter and store in the field.
Format	Values in the column will be displayed as per the defined format.
Decimal Places	Sets the number of decimal places for the values in the field.
Input Mask	Controls how user enters data.
Caption	Defines the column heading.
Default Value	Value which will be displayed in the column for each new record.
Validation Rule	Defines conditions on column data.
Validation Text	Defines error message which should be displayed when validation rule is violated.
Required	Shows compulsory data entry for the field.
Indexed	Index is defined for fast data retrieval.
Smart Tags	Smart tag recognizes the data type and performs action accordingly.
Text Align	Aligns the data.

Table 11.4 | 'Text' Field Properties

Property	Description
Field Size	Maximum 255 characters.
Format	Values in the column will be displayed as per the defined format.
Input Mask	Controls how user enters data
Caption	Defines the column heading
Default Value	Value which will be displayed in the column for each new record.
Validation Rule	Defines conditions on column data.
Validation Text	Defines error message which should be displayed when validation rule is violated.
Required	Shows compulsory data entry for the field.
Allow Zero Length	It allows entering zero-length strings (" ").
Indexed	Index is defined for fast data retrieval.
Unicode Compression	Any character whose first byte is 0 is compressed when it is stored and uncompressed when it is retrieved.
IME Mode	Input Method Editor, a tool for using English versions of Access with files created in Japanese or Korean versions of Access.
IME Sentence Mode	Specifies the type of data you can enter by using an Input Method Editor.
Smart Tags	Smart tag recognizes the data type and performs action accordingly.
Text Align	Aligns the data.

Table 11.5 | Predefined Formats for Number, Autonumber, and Currency Data Types

Format	Description	Example
General Number	Displays the number as entered.	456
Currency	Applies the currency symbol.	$4, 342
Euro	Applies the Euro symbol.	€4,234.23
Fixed	Displays numbers without thousand separators and with two decimal places.	9823.23
Standard	Displays numbers with thousand separators and two decimal places.	9,823.23
Percent	Displays numbers as percentages (number multiplied by 100) with two decimal places percent sign.	563.15%
Scientific	Displays numbers with scientific (exponential) notations. For example, 1230000 is displayed as 1.23E + 04	1.23E+04

Table 11.6 | Predefined Formats for the Date/Time Data Type

Format	Description	Example
General Date	(Default) Displays date and time values followed by AM or PM.	08/29/2006 10:10:42 AM
Long Date	Displays long date.	Monday, August 29, 2006
Medium Date	Displays the date as dd/mon/yyyy.	29/Aug/2006 29-Aug-2006
Short Date	Displays short date.	8/29/2005 8-29-2006
Long Time	Displays hours, minutes, and seconds followed by AM or PM.	10:10:42 AM
Medium Time	Displays hours and minutes followed by AM or PM.	1:10 AM
Short Time	Displays only hours and minutes.	13:10

Table 11.7 | Input Mask Characters

Character	Description
0	We can enter any Digit (0-9) where 0 is used in the input mask. Data entry is compulsory at the position where 0 is specified.
9	We can enter any Digit (0-9) where 9 is used in the input mask. Data entry is optional at the position where 9 is specified.
#	We can enter a digit (0-9), a space, or a plus or minus sign in this position. If users skip this position, Access enters a blank space. Data entry is optional for this position.
L	We can enter any alphabet (A-Z or a-z) where L is used in the input mask. Data entry is compulsory at the position where L is specified.
?	We can enter any alphabet (A-Z or a-z) where ? is used in the input mask. Data entry is optional at the position where ? is specified.
A	We can enter any alphabet or digit (A-Z or a-z or 0-9) where A is used in the input mask. Data entry is compulsory at the position where A is specified.
A	We can enter any alphabet or digit (A-Z or a-z or 0-9) where a is used in the input mask. Data entry is optional at the position where a is specified.
&	Any character (including symbols) or space. Data entry is compulsory at the position where & is specified.
C	Any character (including symbols) or space. Data entry is optional at the position where C is specified.
. , : ; /	Decimal and thousands placeholders, date and time separators. The character you select depends on your Windows regional settings.
<	All characters that follow appear in uppercase.
>	All characters that follow appear in lowercase.
!	Causes the input mask to fill from left to right instead of right to left.
\	Forces Access to display the character that immediately follows. This is the same as surrounding a character in double quotation marks.
'Literal text'	Surround any text that you want users to see in double quotation marks.
Password	In Design view for tables or forms, setting the Input Mask property to Password creates a password entry box. When users type passwords in the box, Access stores the characters but displays asterisks (*).

Defining a Foreign Key: By defining a foreign key, we can relate two tables. Do the following to define a foreign key:

1. First, create the parent table and define a primary key in this table.
2. Create a child table which should contain the field with the same data type of parent table's primary key.
3. Select database tools tab from show/hide group and click relationships.
4. Select parent and child tables.
5. Drag parent table's primary key to the related field of child table.
6. Click OK.
7. We can also select Cascade option while defining a foreign key.

11.5 | SORTING AND FILTERING RECORDS IN MS-ACCESS

Sorting Records:

- MS-Access automatically sorts records by the value in primary key field.
- We can sort records based on the value in a specific field.
- We can sort up to 255 characters.
- Ascending sort order arranges text values in alphabetical order (A to Z), date/time values from earliest to latest, number/currency values from lowest to highest, yes/no values are sorted first by 'yes' values then by 'no' values.
- We can sort memo fields using first 255 characters.
- Hyperlink fields can be sorted by the text to display (if any) or the address.
- We cannot sort OLE object or attachment fields.
- We can sort single or multiple fields.

Sorting by a single field:

- To sort by a single field in datasheet view, click within the field you want to sort by; and then to sort by a single field on the 'home' tab in the 'sort and filter' group, do any of the following:
  i. Click ascending/descending command.
  ii. Right click in the field and choose sort A to Z/Z to A from the shortcut menu.
- To restore the records to their original order, on the 'home' tab's sort and filter group, click the 'clear all sorts' command.

Sorting by two or more fields:

- To sort by more than one field, the sorted fields must be adjacent in the datasheet.
- Sorting done from left to right, so the records are sorted first by the values in left column. If the duplicate values appear in first column, sorting is performed on those records by the values in the next column to the right.
- If columns are not adjacent, then we must move the columns before sorting the records.
- When we close the table after sorting is done, access asks to save changes. Click yes to save changes permanently.

Table 11.8 | Difference Between Finding and Filtering Records

Finding Records	Filtering Records
When Access finds record, the cursor moves to the record in a datasheet and all the other records remain on the screen.	Filter removes the records from the screen which do not meet the condition and displays only those records which you want to see. But records remain in the table.

Table 11.9 | Difference Between Removing and Clearing Filters

Removing filters	Clearing filters
Removing a filter simply returns all the records to the datasheet or form.	Clearing the filter erases the filter criteria.
We can reapply the filter later.	We cannot reapply the filter without reconstructing it.

Filtering Records:

- When we want to see only certain records in our datasheet or form, we can filter out the records which we do not want to see. The filter process displays only those records that meet the criteria (conditions).
- Advantages of using filter: Filtering can help you save time by displaying only those records which are important to you at the moment.
- Filtering does not remove the records from the table; it only removes them from your view of the table.
- Filter can display result of a condition consisting simple and complex rules.
- Difference between finding records and filtering records: Table 11.8 shows the difference between finding and filtering records.
- In MS-Access, there are four ways to filter records depending on the conditions and a particular sorted order. These are as follows:
    1. Common context filters: They are available in the shortcut menus depending on the field type.
    2. Filter by selection: It leaves only the records with the same value as the one you select in one of the records or the records that do not include the same value.
    3. Filter by form: It displays records with the criteria entered into a table.
    4. Advanced filter/sort: It gives the capability of specifying a complex sort. With a complex sort, we can sort the records by two or more fields using different orders—ascending or descending.
- Difference between removing and clearing filters: Table 11.9 shows the differences between Removing and Clearing filters.

11.6 | CREATING QUERIES IN MS-ACCESS

SQL is the standard command set that allows the users to interact with the relational database management systems. (Some examples of relational database management systems are MS-Access, MS-SQL Server, DB2, Oracle).

All tasks related to relational data management, such as creating tables, querying the database for information, modifying the data in the database, deleting them, granting access to users and so on, can be easily performed using SQL. We can create a query in MS-Access using query design view, query wizard or SQL view. To create a query using query wizard, follow the steps as given below:

1. Create the necessary tables.
2. Click on ‘create’ tab.
3. Click ‘Query wizard’ option available in the ‘other’ group on the ribbon.
4. As per the requirement, select any one option from the given four options—simple query wizard, crosstab query wizard, find duplicates query wizard and find unmatched query wizard.
5. Select tables and fields which you want to see in the result.
6. Click finish to view the result.

To create a query using query design, follow the steps as given below:

1. Create the necessary tables.
2. Click on ‘create’ tab.
3. Click ‘Query design’ option available in the ‘other’ group on the ribbon.
4. The ‘Show table’ dialog box will appear on the screen. Select the tables and click close.
5. Select the fields from the table and specify the criteria in the design pane and view the result using datasheet view or by clicking on run option.

We can write any select statement in a ‘SQL view’ of Access. To write a query in SQL view, follow the steps as given below:

1. Create the necessary tables.
2. Click on ‘create’ tab.
3. Click ‘Query design’ option available in the ‘other’ group on the ribbon.
4. The ‘Show table’ dialog box will appear on the screen. Click close.
5. Right click on ‘query1’ and select ‘SQL view’. Write select statement in the window and then execute the query using ‘run’ option available in ‘results’ group of ‘Design’ tab.

Table 11.10 shows the differences between filter and query.

Table 11.10 | Difference between Filter and Query

Filter	Query
Filter is saved with a table.	Query is a separate database object which appears in the navigation pane.
To view the result of a filter, we must open the table first.	There is no need to open a table first to view the result because a query is an independent object.
We can apply filter on a single table.	We can create a query using multiple tables.
We cannot perform operations (insert, update, delete) on data using a filter.	We can perform operations (insert, update, delete) on fields using a query.

Types of Queries in MS-Access: There are four categories of queries in MS-Access.

Select queries: It is the most common category and is used to fetch records from database tables. We can create three types of select queries using query wizard.

a. Simple Select query: It displays data from one or more tables grouped or sorted in a specific order. To create a simple query, follow the steps as given below:
   1. Click on create → query wizard.
   2. Select simple query wizard option and click OK.
   3. Select table and fields which you want to display and click next.
   4. Give any valid name to a query and click finish.

b. Find duplicate query: It displays all records with duplicate values in one or more specified fields. To create 'find duplicate query' follow the steps as given below:
   1. Click on create → query wizard.
   2. Select find duplicates query wizard option and click OK.
   3. Select table from which you want to find duplicate values and click next.
   4. Select the fields which may contain duplicate values and click next.
   5. Select the fields which you want to display and click next.
   6. Give any valid name to a query and click finish.

c. Find unmatched query: It displays records in one table that have no related records in another table. To create a find unmatched query, follow the steps given below:
   1. Click on create → query wizard.
   2. Select find unmatched query wizard option and click OK.
   3. Select table from which you want to display fields and click next.
   4. Select the table which contains related field and click next.
   5. Select the matching field from both the tables and click next.
   6. Select the fields which you want to display and click next.
   7. Give any valid name to a query and click finish.

Action queries: They are used to insert, update or delete records from/to tables. There are four types of action queries in Access:

a. Make table query: It is used to create a new table out of data from one or more tables. To create a make table query follow the steps as given below:
   1. Click on create → query design.
   2. Select the table(s) from which you want to retrieve records.
   3. Click on design tab's query type group and select make-table option. MS-Access will display the make-table dialog box and asks for the target table in which you want to paste records.
   4. Select the table name and database where you want to copy your records and click OK.
   5. Drag fields from the available tables and write criteria if any.
   6. Click on 'Run' option given in the results group.

b. Append query: Add a group of records from one or more tables to the end of one or more other tables. To create an append query follow the steps as given below:
   1. Click on create → query design. Select the table(s) from which you want to append records.
   2. Click on design tab's query type group and select append option.

3. Select the table name in which you want to append records.
4. Select fields of source table in the design pane and select related fields of destination table in 'append to' cell.
5. Click on 'Run' option given in the results group.

c. Update query: Make global changes to a group of records in one or more tables. To create an update query follow the steps as given below:
   1. Click on create → query design. Select the table(s) which you want to update.
   2. Click on design tab's query type group and select update option.
   3. Select fields which you want to update and the fields on which you want to put conditions.
   4. Enter the new values of fields in the 'Update to' cell and field conditions in the 'criteria' cell.
   5. Click on 'Run' option given in the results group.

d. Delete query: Remove a specific group of records from one or more tables. To create a delete query follow the steps as given below:
   1. Click on create → query design. Select the table from which you want to delete records.
   2. Click on design tab's query type group and select delete option.
   3. Select fields on which you want to put conditions.
   4. Enter the conditions in the 'criteria' cell.
   5. Click on 'Run' option given in the results group.

Special purpose queries: It is used to summarize values from one field in the table, grouped in two ways or automatically fill in data or prompt for criteria.

a. Parameterized query: Display a dialog box where you enter the criteria for retrieving data or a value to insert into a field. We can apply parameters to other types of queries as well. To create a parameterized query, follow the steps as given below:
   1. Click on create → query design and select the table.
   2. Select the fields which you want to see in the result.
   3. Click on the field for which you want to set parameter.
   4. Write name of the parameter in the criteria. The parameter name should be enclosed within the square brackets.
   5. Click on parameter in the 'show/hide' group and define the parameter in the available window by specifying its datatype.
   6. To view the result click on datasheet view. MS-Access will prompt to enter the parameter value. Give parameter value and click OK.

b. Autolookup query: Special select queries that automatically fill in certain field values in a new record in one or more tables. To create an autolookup query, follow the steps as given below:
   1. Click on create → query design and select the parent and child table.
   2. Select related field from the child table and other fields from the parent table.
   3. Click on datasheet view to see results.

c. Crosstab query: Calculate a sum or count and group the results in a spreadsheet format that correlates the data with two types of information.

Table 11.11 | Predicates Used with Select Statement

Predicate	Meaning
Distinct	To display unique values of fields. For example, Select distinct stdname from student.
Top n	To display n (any integer number) top most values. For example, select top 10 stdno from student order by stdno desc.

Table 11.12 | Wildcards

Wildcard	Meaning
?	Single character
#	Single digit (between 0-9)
*	Many characters
[char/digit_list]	Specify range of characters or digits for a single character

SQL-specific queries: These are accessible only through SQL statements. All queries have SQL statements in the background, but SQL-specific queries are constructed with the programming language instead of a design grid like other types of queries.

a. Union: Combine fields from one or more tables into one field in the result.
b. Sub query: SQL SELECT or other server statements that form a SELECT query within another query.
c. Pass-through query: Send instructions directly to open database connectivity (ODBC) databases using commands specific to the server.
d. Data definition query: Create or change database objects in MS-Access, SQL server or other server database.

Predicates: Predicates shown in Table 11.11 are written between select keyword and list of fields as per the requirement.

Special Operators: There are five special operators available with select statement: 1. Like; 2. Is Null; 3. In; 4. Exists; and 5. Between.

1. **Like:** It is written after where clause to match specific pattern in a text or string. MS-Access allows the following wildcards to be used with the like operator. Table 11.12 shows the wildcard characters which could be used with select statement in MS-Access. Here are some examples of Like operators:

   - Display names of the student in which first letter is A or Z.

     ```
     SELECT students' details from student
     where stdname like '[A,Z]*'
     ```

   - Display students' details containing the word 'kan'.

     ```
     SELECT * from student where stdname like '*kan*'
     ```

   - Display students' details in which second letter is a digit.

     ```
     SELECT * from student where bdate like '?#*'
     ```

   - Display students' details in which first letter lies between 0 and 9.

     ```
     SELECT * from student where bdate like '[0-9]*'
     ```

- Display students' details in which first letter does not lie between 0 and 9.

```
SELECT * from student where bdate like '![0-9]*'
```

2. **Is null:** It is needed to check and display the records in which specific field contains any null value or not. For example, `select * from student where stdname is null`.
3. **In:** It is written before the subquery (query written within the select statement) to check the field value lies in specific list of values or in list of values selected by another subquery. The subquery will be executed first; and based on the result returned by a subquery, the main query will be executed. For example, `select * from student where stdno in (select stdno from mark)`.
4. **Exists:** It is written before the subquery (query written within the select statement) to check the existence of records returned by the subquery. It will return true if the subquery contains any record else returns false if the subquery doesn't contain any record. For example, `select * from student where exists(select * from mark)`.
5. **Between:** It is written after the clause to check the field value lies in a specific range. The upper and lower values are also included in the range. For example, `select * from student where stdno lies between 10 and 20`.

Functions: Following functions are available in MS-Access:

1. **Date and Time Functions:** Table 11.13 shows date and time functions.
   Some examples: You can write the following select queries to view the result of above functions.

- Display current date and time, only current date, only current time.

```
SELECT now (), date (), time () from student;
```

- Display use of datepart, datediff and dateadd functions.

```
SELECT datepart ('d', date ()) as 'today's date',
datepart ('ww', date ()) as 'week no of the year',
datepart ('y', date ()) as 'day of the year',
datepart ('yyyy', date()) as year_no,
datepart ('q', date ()) as 'quarter_no' ,
datepart ('m', date()) as 'month of the year',
datepart ('w', date()) as 'day of the week',
month (#12-12-2010#) as 'month no',
year (#12-12-2010#) as 'year no',
day (#12-12-2010#) as 'day of the month',
weekday (#12-12-2010#) as 'day of week' ,
datediff('d', #12-12-2008#,#1-12-2010#) as 'date difference',
datediff('m', #12-12-2008#,#1-12-2010#) as 'month difference',
datediff('yy yy', #12-12-2008#,#1-12-2010#)as 'year difference',
dateadd ('d',234, #12-12-2008#) as 'Add no of days',
dateadd ('yyyy', 3,#12-12-2008#) as 'Add no of years',
dateadd('m',24,#12-12-2008#) as 'Add no of months' from student;
```

- Display different uses of a format function.

```
SELECT format ('shefali', '>') as 'string in upper case',
format('Shefali', <') as 'string in lower case',
format (date (), 'd-mmmm-yyyy'),
format (date (),'long date'),
format (time (),'long time'),
format (date (), 'dd-mm-yyyy'),
format (date (), 'mmmm'),
format (date (), 'dddd'),
format (date (), 'yyyy'),
format (time (), 'hh'), format (time (), 'ss') ,
format (time (), 'n'),
format (time (), 'hh:n:ss am/pm')
from student;
```

2. **String functions:** Table 11.14 shows the string functions of MS-Access.

Table 11.13 | Date and Time Functions

Function	Meaning
Now ()	Current date and time
Date ()	Current date
Time ()	Current time
Datepart ('partofdate',dateexpression)	Display any part from the specified date
Datediff ('partofdate',from_dateexpress-ion, to_dateexpression)	Display difference of two dates as number of days or number of months or number of years
Dateadd ('partofdate', duration, dateexpression)	Add number of days, months or years and returns date.
Day (dateexpression)	Extracts day from the date
Weekday (dateexpression)	Extracts week day number from the date
Month dateexpression ()	Extracts month number from the date
Year (dateexpression)	Extracts year number from the date

Table 11.14 | String Functions

Function	Meaning
Ucase(stringexpression)	Display string in capital letters.
Lcase(stringexpression)	Display string in small letters.
Ltrim(stringexpression)	Removes spaces from left side of the string.
Rtrim(stringexpression)	Removes spaces from right side of the string.
Trim(stringexpression)	Removes spaces from left and right side of the string.
Left(stringexpression,no._of_char)	Display specific number of characters from left side of the string.
Right(stringexpression,no._of_char)	Display specific number of characters from right side of the string.
Mid(stringexpression,start_ position,no._of_char)	Display specific number of characters from the position specified.
Len(stringexpression)	Display length of a string.

Some examples: You can write the following 'select' queries to view the result of the functions as given in Table 11.14.

```
SELECT ucase('shefali'),
       lcase('SHEFELI'),
       ltrim('shefali'),
       rtrim('shefali'),
       trim('shefali'),
       left('shefali', 4),
       right('shefali', 4),
       mid('shefali', 2, 3),
       len('shefali')
from student;
```

3. **Mathematical functions:** Table 11.15 shows the mathematical functions.
   For example:

   ```
   SELECT round(34.57, 0),int(34.34), rnd() from student;
   ```

4. **Aggregate functions:** These functions are used for summary or collective results. Table 11.16 shows the aggregate functions.

Types of Joins: We can join two or more than two tables to retrieve records from more than one table. When you include multiple tables in a query, you use joins to help you get the results you are looking for. A join helps a query return only the records from each table you want to see based on how those tables are related to other tables in the query. Following are the types of joins available in MS-Access:

1. **Equi Join:** When we join two or more than two tables using a '=' sign, then the type of join is said to be an equi join. For example, `select * from class, student where class.classcode = student.classcode`.

Table 11.15 | Mathematical Functions

Function	Meaning
Round (number,decimal_places)	Rounds a number
Int (number)	Displays only quotient part of a number
Rnd ()	Generates unique random numbers

Table 11.16 | Aggregate Functions

Function	Meaning
Count(fieldname) or count(*)	Displays total no. of values or records
Max(fieldname)	Displays maximum value from the list of values
Min(fieldname)	Displays minimum value from the list of values
Avg(fieldname)	Displays average values
Sum(fieldname)	Displays sum of values

2. **Inner Join:** Inner joins are the most common type of join. They tell a query that rows from one of the joined tables correspond to rows in the other table on the basis of the data in the joined fields. When a query with an inner join is executed, only those rows where common values exists in the tables which are joined will be displayed in the result. Use an inner join if you want to return only those rows from both tables in the join that match on the joining field. For example, `select * from class inner join student on class.classcode = student.classcode`.
3. **Left Outer Join:** Outer joins tell a query that although some of the rows on both sides of the join correspond exactly, the query should include all of the rows from one table, and also those rows from the other table that share a common value on both sides of the join. Outer joins can be left outer joins or can be right outer joins. In a left outer join, the query includes all of the rows from the first table in the SQL statement FROM clause, and only those rows from the other table where the joining field contains values common to both tables. For example, `select * from class left join student on class.classcode=student.classcode`.
4. **Right Outer Join:** In a right outer join, the query includes all of the rows from the second table in the SQL statement FROM clause, and only those rows from the other table where the joining field contains values common to both tables. Use an outer join if you want all of the rows from one of the tables in the join to be included in your results and you want the query to return only those rows from the other table that match the first table on the joining field. For example, `select * from class right join student on class.classcode = student.classcode`.
5. **Cross Join:** When join criteria is not being specified after where clause, the query will display Cartesian product of selected tables and this type of join is said to be cross join. For example, `select * from class, student`.
6. **Multiple Join:** When we join more than two tables in a single query, the type of join is said to be multiple join. For example, `select * from class, student, mark where class.classcode = student.classcode and student.stdno=mark.stdno`.
7. **Union:** A union query uses the UNION operator to combine the results of two or more select queries and combines the results of two or more independent queries or tables. When we use union keyword, number of fields in both the queries should be same and datatype of corresponding fields should match with one another. For example, `select classcode from class union and select classcode from student`.

11.7 | CREATING FORMS IN MS-ACCESS

Forms: Forms are used for viewing and entering data. In MS-Access, we can design a form that presents data in a way that makes the information easy to understand, enter (add/insert) and manage (change). There are three views available using which we can view, insert or change data and change design or properties of a form. The three views are: Form View, Design View, Layout View.

- **Form View:** We can navigate the records, filter the records, search record and add, delete or edit/change records in a form view.

- **Design View:** We can change the design or layout of a form; set properties for different controls and sections of a form; change tab orders; set default properties for controls; add calculated fields or add any control from the control group in a design view.
- **Layout View:** It displays layout of a form. We cannot enter data in this view. We cannot add new controls on a form using this view.

Types of Form: On the basis of table or query, we can create a form using the following form structures:

1. Form: It is a form structure which creates a simple form. To create this type of form, the steps are:
   a. Select a table or query from the navigation pane for which you want to create a form.
   b. Click the ‘Create’ tab.
   c. Select the ‘Form’ option from the form group.
2. Split form: A split form is one of the form design structure which offers the data in two views at the same time—the form layout view and the datasheet view. We can add, delete or edit data in either part of the form. Both the views are synchronized as the changes made in one view are reflected in the other view. To create a form using ‘Split form’ design structure the steps are:
   a. Select a table or query from the navigation pane for which you want to create a form.
   b. Click the ‘Create’ tab.
   c. Select the ‘Split Form’ option from the form group.
3. Multiple Items: This type of structure displays several records in the Layout view. To create a form using the ‘Multiple Items’ design structure, the steps are:
   a. Select a table or query from the navigation pane for which you want to create a form.
   b. Click the ‘Create’ tab.
   c. Select the ‘Multiple Items’ option from the form group.
4. Pivot Chart: It provides the tools to create a graphical analysis of the data in a table or query. It can be used to add chart in an existing form.
5. Blank Form: To create a form using ‘blank form’ design structure the steps are:
   a. Select a table or query from the navigation pane for which you want to create a form.
   b. Click the ‘Create’ tab.
   c. Select ‘blank form’ option from the form group. This option opens an empty form in the Layout View with the field List pane open at the right side. To add fields to the form, select table and drag the fields to the form layout or we can set ‘recordsource’ property of a form with the name of a table. Then drag required controls from the ‘controls’ group and bind each control with the table’s field using ‘controlsource’ property.
6. Form Wizard: To create a form using the ‘form wizard’ design structure the steps are:
   a. Select a table or query from the navigation pane for which you want to create a form.
   b. Click the ‘Create’ tab.
   c. Select the ‘form wizard’ option from the ‘more forms’ option from the form group. It takes you through the form design process where you can select table/query, layout, style and name for your form.

There are four layout options available in a form wizard.

1. Columnar: It arranges all the fields on the screen in one or more columns, depending on the number and size of the fields.
2. Tabular: It places all the data from one record in a line across the form.
3. Datasheet: It is similar to table's Datasheet view. This style is used for subforms.
4. Justified: The fields are arranged in rows, but the row of fields is wrapped to multiple lines as necessary.
5. Datasheet: It opens the table or query that is the basis for the form currently selected in the navigation pane in datasheet view.
6. Modal Dialog: It is used to create a dialog box which is not based on table data but it includes user-interaction controls such as command buttons, option groups and drop-down lists.
7. Pivot Table: It summarizes and analyzes data and builds a table.

Recordsource: The record source is one of the properties of every form and report which provides the source of data from table, query or an SQL statement. We can set or change the record source property of a form by doing the following:

1. In the form design view, click the form selector. Then in the tools group, click the property sheet command or right-click the selector and choose properties from the shortcut menu or click F4.
2. Open the 'Data' tab and click the down arrow in the recordsource property box.
3. Choose the record source from the drop-down list of all tables and queries in the current database.

Form Sections: There are five sections of a form.

1. Detail: It contains the data.
2. Form Header: It contains information to show at the top of the screen. For example, title, instructions, etc. This information is printed at the top of the first page.
3. Form Footer: It contains information to show at the bottom of the screen. This information is printed at the bottom of the last page.
4. Page Header: It contains information to show at the top of each page when the form is printed or previewed. This section is not visible in Form View.
5. Page Footer: It contains information to show at the bottom of each page when the form is printed or previewed. This section is not visible in Form View.

Property Tabs: There are five tabs available in the property sheet.

1. Format: It lists all formatting properties of particular control such as caption, visible, back colour, back style, height, width.
2. Data: Properties listed in this tab determines what data is displayed and how form handles the data. For example, controlsource, filter, and sort order, etc.
3. Event: It specifies what happens when some event occurs. For example, when form opens or closes, etc.
4. Other: It includes miscellaneous properties such as Popup and Modal.
5. All: It displays complete list of properties.

- **Bound, Unbound and Calculated Controls:**
  - Bound control: The control which gets its value from a field in the table or query and as data changes, the value of the control changes.
  - Unbound control: The control which has no tie to the underlying table data and retains the value user enters is called an unbound control.
  - Calculated control: The control which gets its value from values in the table and is an expression containing functions and operators that produces some result is called a calculated control.
- **Set default properties for any control:** To set default properties for any control follow the steps given below:
  - From the control groups given in the Design tab, click on the control for which you want to set the default properties.
  - In the property sheet window, you will get the selection type: Default for that control.
  - Set the properties which you want as default properties.
- **To place a calculated control on a form which will display the age of a student:**
  - Select textbox control from the controls group.
  - Open property sheet for that textbox.
  - Select control source property and type =datediff('yyyy',birthdate,date()) in that.
- **Controls available in 'Controls' group of MS-Access 2007:**
  - Textbox: A control that displays field data from tables, queries or calculated fields is called a textbox.
  - Label: A control which displays descriptive text, such as titles, captions or instructions is called a label. It is an unbound control.
  - Command Button: A control which initiates an action, such as opening a linked form, running a macro or calling a Visual Basic for Application (VBA) procedure is called a command button.
  - Combo Box: It is a combination box which combines two controls-drop down list box and check box. It is used to display multiple items.
  - Check Box: It displays a check mark in a small box if the underlying field is yes. The box is empty if the value is 'no'.
  - Option button: It is also known as radio button. It displays a black dot inside a circle if the value in the underlying field is yes. The circle is empty if the value is 'no'.
  - Toggle button: When toggle button appears selected, the value in the underlying field is yes. If the button appears raised, the value is 'no'.
  - Option Group: It provides groups of Toggle buttons, Check boxes or Option buttons.
  - Bound object frame: It displays an object (*e.g.*, image) which is stored in the field of a table.
  - Unbound object frame: It is used to display an object which does not belong to any field of a table.
  - Image: It is used to display unbound image. For example, Logo of a company.
  - Line: Used to draw straight line. It is an unbound control.
  - List box: A control that displays a list of choices, such as values for a field or search criteria.

- ○ Logo: A picture to be used as a logo on a form or report. It is usually placed in the form or report header.
- ○ Page break: A control that creates a form with more than one page or causes a report to move on the next printed page.
- ○ Rectangle: It is used to draw rectangle. It is an unbound control.
- ○ Subform/subreport: A form or report contained within another form or report that shows data from related tables.
- ○ Tab: A control that shows a multiple-page form with tabs at the top of each page.

- **Hierarchical Form:** Hierarchical form represents 1:M (1 to many) type of relationship. It consists of a main form and one or more subforms. The main form shows data from records on the 'one' side of a one-to-many relationship and the subforms show data from records on the 'many' side.

  Steps to create a hierarchical form: Following steps should be followed to create a hierarchical form:

  1. Create parent table and child table.
  2. Set relationship between parent and child table using a linked field.
  3. Start form wizard and select parent table. In the first dialog box, select fields which you want from the parent table.
  4. Then select child table. Select fields which you want from the child table. Click next.
  5. The second dialog box asks how you want to view the data, *i.e.*, which records you want in the main form and which in the subform. Click next.
  6. Select the option 'form with subform' and click next.
  7. Select the layout (tabular or datasheet) for the subform and click next.
  8. Choose the style for the form in the next dialog box and click next.
  9. In the final dialog box, name the form and the subform and click finish.

- **Put a hyperlink on the form:** To put a hyperlink on the form, do the following:

  1. Open the form in design view. On the create tab's controls group, click the insert hyperlink command.
  2. From the dialog box, select the filename from the list or type the path and file name.
  3. If you want to open an object from the current database then click on bookmark and select the object. Click OK.
  4. Drag the hyperlink control to the position you want in the form design.

- **Set tab order of different controls on a form:** To set tab order of different controls on a form follow the steps given below:

  1. Right click on the form.
  2. Select tab order option from the pop-up menu.
  3. Adjust controls using record selector available in the tab order dialog box.

- **Steps to give conditional formatting to any control:** We can specify format for a field for specific condition by setting conditional formatting. To create conditional formatting, select the control on a form and right click. Then select the option conditional formatting and set format according to the condition.
- **Steps to create multiple pages:** There are two ways to create a multiple-page form, as:

1. By inserting a page-break control: Page breaks are used to separate the form horizontally into two or more pages. To insert a page break on the 'Design' tab in the 'controls' group, click the 'page break' control and then click in the form where you want the split.
2. By using a tab control: Tab control produces multiple-page tabbed forms that combine all the pages into a single control. Tab controls are useful for presenting grouped information that can be assembled by category. A tab control has pages, each with a tab of its own. Each tab page can contain all types of controls such as text boxes, labels, combo boxes. To create a tab control on the 'Design' tab in the 'controls' group, click the 'tab' control and then click in the form where you want it. Then add controls on each tab. We can rename the tab's name by double clicking it or by changing new name in the 'caption' property of that page. To add or remove a page in a tab control, right click on a tab control and select 'insert page' or 'delete page' option.

11.8 | CREATING REPORTS IN MS-ACCESS

Reports: Reports are used for viewing and printing data in some specific format. There are four views available using which we can view, insert or change data and change design or properties of a report. The four views are: Report View, Design View, Layout View and Print Preview.

1. Report View: We can view the details of data, but can't change or insert anything. The report will be available as read only in this view.
2. Design View: We can change the design or layout of a report; set properties for different control and sections of a report; add calculated fields or add any control from the control group in a design view.
3. Layout View: It displays layout of a report. We can add some controls such as logo, title, page no. and date/time in this view. We can change properties of existing controls of a report using this view.
4. Print Preview: It will display the layout of a report and shows how the report will look when it will be printed.

Creating a Report: We can create a report in the following five ways. We may use

- **'Report' option given in the 'Reports' group**: To create a report using 'report' option, the steps are:
  a. Select a table or query from the navigation pane for which you want to create a report.
  b. Click 'Create' tab and select 'report' option from the reports group.
- **'Report Wizard' option given in the 'Reports' group**: To create a report using 'report wizard' design structure, the steps are:
  a. Select 'report wizard' option from the 'reports' group. It takes you through the report design process where you can select table/query, layout, style and name for your report.
  b. Select table/query using which you want to create a report and select fields of that report.
  c. Click next and select the filed/fields on which you want to group the records. After selecting the field, you may select the grouping intervals for the fields. For example,

   if the field's data type is number then MS-Access gives you the choice for no. of intervals (10s, 50s, 100s, and so on.) on which you want to group the data.
   d. Click next. Then we can select the sorting option and summary option.
   e. Click next. We can select layout (Stepped, Outline or Block) and orientation (Landscape or Portrait).
   f. Click next. We can select style for the report from the given option.
   g. Click next and give name to the report and click finish.

- **'Labels' option given in the 'Reports' group:** To create a report of label type, the steps are:
   a. Select a table/query and click the 'labels' option from the 'reports' group.
   b. The label wizard will be opened where you need to select 'label type' (sheet feed or continuous), 'unit of measure' [English (it will display label size in inches) or Metric (it will display label size in millimeter] and click Next.
   c. Select font name, size, colour, etc., and click Next.
   d. Select fields which you want to display on a label and click Next.
   e. Select field name on which you want to sort your labels and click Next.
   f. Give name to this report and click Finish.
- **'Report Design' option given in the 'Reports' group**: This option will open the design view of a report where we can paste controls and bind them with the fields manually or drag fields from the window 'field list'.
- **'Blank Report' option given in the 'Reports' group**: To create a report using 'blank form' design structure, the steps are:
   a. Click the 'Create' tab and select the 'blank form' option from the reports group.
   b. This option opens an empty report in a Layout View with the 'Field List' pane open at the right side. To add fields to the report, select table and drag the fields to the report layout or we can set 'recordsource' property of a report with the name of a table/query. Then drag required controls from the 'controls' group and bind each control with the table's field using 'controlsource' property.

Report Sections: There are seven sections of a report:

- Detail: It contains the data.
- Report Header: It contains information to show at the top of the screen. For example, title, instructions, etc. This information is printed at the top of the first page.
- Report Footer: It contains information to show at the bottom of the screen. This information is printed at the bottom of the last page.
- Page Header: It contains information to show at the top of each page.
- Page Footer: It contains information to show at the bottom of each page.
- Group Header: It contains information to show at the top of a particular group. For each group, individual group header section will be displayed.
- Group Footer: It contains information to show at the bottom of a particular group. For each group, individual group footer section will be displayed.

Subreport: A report which is inserted into another report is called a 'subreport'. A main report can include as many subreports and subforms as necessary. A first level subreport can contain another subreport or a subform. If the first level is a subform, it can contain only another subform, not a subreport, as the second level. We can use the subform/subreport control given in the create tab's reports group to create a new subreport in the current report design. We can also use the subform/subreport wizard to create a subreport. To create a subreport, do the following:

1. Create a main report. Open it in the design view.
2. Select subform/subreport control from the controls group and drag it into the main report's detail section.
3. MS-Access will open the wizard and ask for the selection, where you can select an existing report as the subreport or create a new one using an existing table or query.
4. Choose use existing tables and queries to create the new subreport and click Next.
5. Select fields from the list of fields which you want to place in a subreport.
6. Select the linked field using which you want to relate records of main report and sub report. Click Next.
7. Give any name to the subreport and click on finish.

Parameterized Report: To create a parameterized report, go with the following steps:

1. Create a parameterized query using a query design command or using an SQL view.
2. Save the query.
3. Select report wizard from the create tab's reports group and select the saved parameterized query in the dialog box.
4. Select fields which you want to see in the report and click Next.
5. Select grouping (if any) and click Next.
6. Select sorting fields (if any) and click Next.
7. Select layout and orientation and click Next.
8. Select style and click next. Save the report by giving appropriate name and click finish.

To print parameters in the report header, do the following:

1. On the Design tab's controls group, click the text box and place the control in the Report Header section.
2. Double click the text box control to open the property sheet and enter the expression in the control source property box. When expression is entered, write the message (text) in double quotation marks and parameters in square brackets.
3. To concatenate parameters and message (text), use '&' character. For example, if the parameter name is 'enter student number', then it should be enclosed within square brackets. If the message is 'Details of student number', then it should be enclosed in double quotation marks and, then both should be concatenated using '&' character. Therefore, the final expression in the control source property of a text box will be = 'Details of student number: '& [enter student number]

Adding groups or sorts in the report: To add any group in the report, do the following:

1. Select the design tab. Click the 'group and sort' option given in the 'grouping and totals' group. The 'grouping, sort and totals' window will be available below the report window.
2. Click on the 'Add a group' in that window. Select field name on which you want to group your records. Group will be available in that window with some options which we can set as per our requirement. Using these options, we can set sorting order, can apply

aggregate function on the data, can set title for the group, can select option to hide/show group header/footer, and can select the printing position for the group.

3. We can group the data on some part of a field also, *i.e.*, you may select the grouping intervals for the fields. For example, if the field's data type is number, then MS-Access gives you the choice for no. of intervals (10s, 50s, 100s, and so on) on which you want to group the data. If the field's data type is text, then MS-Access gives you the choice, such as by entire value, by first character, by first two characters or we can set the number of characters using custom option. If the field's data type is date/time, then MS-Access gives you the choice such as by entire value, by day, by week, by month, by quarter, by year, or we can set any number for hour, minute and second.
4. To set the caption of an aggregate function, select the textbox and right click. Select the option 'Set caption'.
5. Similarly, we can click on 'Add a sort' in grouping, sort and totals window. Select field name on which you want to sort your records.

Printing serial numbers on a report: To display serial numbers or line numbers in a report which does not contain any group, do the following:

1. Add a calculated text box control to the detail section at the required position.
2. Remove the label of the text box and set the control source property of a text box=1.
3. Set the running sum property of a textbox to Over All.
4. If you want to restart the serial numbers for a new group, then set the running sum property of a textbox to Over Group.

Exporting a report: To export the report, click on external data and select any option from the 'export' group. We can export the report as following files:

1. Word file [(It will save the file with .rtf (rich text format) extension.]
2. Text file (It will save the file with .txt extension.)
3. Snapshot Viewer file (It will save the file with .snp extension. This file will keep report formatting as it is.)
4. HTML file (It will save the file with .html extension.)
5. XML file (It will save the file with .xml extension.)
6. Save report in another database.

After selecting the option, choose path where you want to store your report and give file name. If you want to save the steps of export, then select the option 'save export steps' and click on Finish.

Removing duplicate values or repeating values: When we group our data, the data are repeated for the grouped column. To display the group detail only once, we can shift the textbox of grouped field in the group header section or we can set 'hide duplicates' format property with the value 'yes' of a textbox on which grouping is done.

Creating a chart: A chart is composed of elements, some of which relate to the data, while others relate to the structure of the chart itself. Following are the main elements of a standard chart:

- Category (x) axis: It is the horizontal line at the bottom of the chart that usually identifies the data in the chart. For example, when you plot class-wise total number of students, the class name appears on the Category (x) axis.

- Value (y) axis: The vertical line that measures the values in the chart data. For example, when you plot class wise total number of students, the total number of students appears on the Value (y) axis.
- Z axis: It is optional. It appears in 3D charts, and also measures values.
- Series: A group of related data values from one field in the underlying record source. For example, in a class-wise total number of students, each class's total number of students would represent one of the series values. The values are grouped together in one category—the classid.
- Titles: Explain the purpose and scope of the chart. Titles are optional and can appear at the top of the chart and by each axis.
- Tick marks: Short lines that appear on the axes to mark evenly spaced segments. They help you to read values and determine the scale of the chart.
- Gridlines: Horizontal or vertical lines that appear across or up and down the chart at the tick marks.
- Scale: Defines the range of values in the chart and the increments marked by tick marks on the axes.
- Slice: It can be defined as a wedge of a single field in a pie chart which represents the relative value of one data point with respect to the whole.
- Data markers: The elements that show the value of the data. For example, bars, columns, slices of a pie chart, etc.
- Data labels: The actual values that can be displayed above or near the data markers.
- Legend: The list that identifies the members of a series of data values.

To create a chart, open a report and select 'insert chart' control from 'controls' group and paste it on a report. A chart wizard will be available where we can specify the field which should be displayed on x-axis, the value to be displayed on y-axis, chart type, chart title, etc.

11.9 | CREATING MACROS AND SWITCHBOARD

Macro: A macro is a list of one or more actions that work together to carry out a particular task in response to an event. Each action carries out a particular operation. We can create the list of actions in the order in which we want them to execute. We can also specify other details of the action called, 'argument' which provide additional information such as which form to open, or how to filter the records to be displayed. We can also set conditions, under which the macro action to be performed such as to display a message box if a field contains a certain value, or is blank. The macro action runs only if the condition evaluates to True. If the condition is False, the action is skipped. Then, if another action is in the macro, it is executed. If not, the macro stops. To run a macro, we can assign it to the event property of a form, report, report section or control. When the event occurs, a macro automatically executes, beginning with the first action in the list. For example, a macro that opens a form and moves to a blank record for data entry can be assigned to the 'on click' event property of a command button in a dialog box or another form. When we click the button, the macro executes.

Types of Macros: There are two types of macros: (i) Standalone macro; (ii) Embedded macro. The difference between standalone and embedded macro is provided inTable 11.17.

Table 11.17 | Differences Between Standalone and Embedded Macros

Standalone Macro	Embedded Macro
Standalone macros are individual Access objects which we can create by clicking on macros group.	Embedded macros are created within forms or reports for the specific control.
Standalone macros listed in the Navigation Pane.	Embedded macros are not listed in the Navigation Pane.
We can re-use Standalone macros.	We cannot re-use embedded macros.
We can debug Standalone macros.	We cannot debug embedded macros.
We can save standalone macro with some name as they are individual Access objects.	We cannot save embedded macro with some name as they belong to specific control of a specific form or report.

Steps to create a Standalone Macro:

1. On the Create tab's Other group, click the Macro drop down box and select Macro option.
2. The Macro window will be opened with three columns named Action, Arguments and Comment. We can select the action, set arguments and write comments for that action.
3. To add conditions for the actions, in design tab's show/hide group, select the option conditions. Fourth column conditions will be available in the macro window where we can specify condition for the particular action. If the condition is true then only the related action is being executed.
4. Save the macro with some name. The macro will be available in the navigation pane.

We can assign this macro to the event property of a form, report, report section or control.

Steps to create an Embedded Macro:

1. Open the form or report and select the control for which we want the macro. Open property sheet of that control and click on event property tab. Choose the event's builder dialog box by clicking … (ellipsis) for which you want to define macro.
2. The Macro window will be opened with three columns named Action, Arguments and Comment. We can select the action, set arguments and write comments for that action.
3. To add conditions for the actions, in design tab's show/hide group, select the option conditions. Fourth column conditions will be available in the macro window where we can specify condition for the particular action. If the condition is true, then only the related action is being executed.

Different columns of a Macro sheet: There are five columns available in the macro sheet.

- Condition: The condition can be specified for a particular action. The macro action runs if the condition evaluates to true, otherwise the action is skipped. The condition column will be available when we click on 'condition' option given on the show/hide group of design tab. A condition applies only to the action on the same row in the macro sheet. If the condition is not met, the next action is executed. To continue the condition to the next action, enter an ellipsis (…) in the condition column of the next row. We can apply the condition to several sequential actions. Some examples of conditions are:

Table 11.18 | Examples of Conditions

Expression	Returns True If
[state]='Gujarat'	Value of the control state is 'Gujarat'
Not Isnull ([stdname])	Stdname does not contain null
Forms![student]![birthdate]>date ()	Birthdate control of student form contains future date.

- Action: Each action carries out one particular operation of a macro such as moving among record in a form, playing sounds, displaying message boxes, etc. If a particular action is not listed in the action drop down list box, then click on design tab's show/hide group and select 'show all actions' command.
- Argument: Argument specifies the details of an action which provides additional information such as which form to open or how to filter the records to be displayed.
- Comment: We can write description for a particular action in this column.
- Macro: It is used when we want to define a macro group. The macro column will be available when we click on 'macro names' option given on the show/hide group of design tab.

Testing and Debugging a Macro:

- After creating a macro, we can run it to see if it gives correct output or not. We have a choice of running the complete macro at once or stepping through the macro one action at a time. If an error occurs in the macro or we don't get the expected results, we can use the single step method of running the macro to see what went wrong. We can run the macro either by double clicking it in a navigation pane or by right clicking the macro name and by choosing run from the shortcut menu.
- If the error occurs during the operation, MS-Access displays an error message explaining the reason for the error. Read the message and click OK to open the Action Failed dialog box. It will show the error number and other details such as condition, action name and arguments. Select 'stop all macros' option to stop execution. After that correct the problem and run the macro again.
- Stepping through a macro: To debug the macro step by step, click on 'single step' command given in the tools group and click the run command to carry out the actions one by one. In Macro single step dialog box, the following three options are available:
    1. Step (default): If we click on step, it will move to the next action. It will show outputs after each step if there is no error.
    2. Stop All Macros: It stops macro execution.
    3. Continue: If we click on continue, it stops single step mode and runs the rest of the macro without stopping. It will show the final output if there is no error.

Common uses of a macro:

1. Displaying a message box: Using the 'msgbox' action we can display warnings, alerts or other information. It is one of the most useful macro actions when interacting with the user.
2. Validating data: We ensure that valid data is entered in a form by specifying a validation rule for the control in the form by setting record and field validation rules in the underlying table design. For more complex data validation, we use a macro or an event procedure to specify the rule.

3. Filtering records: We can create a macro to limit the records we want to print by adding a Where condition to the OpenReport action. We can also set filters for forms.
4. Setting values and properties: 'Set value' is a useful macro action that sets the value of a field, control or property of a form, a form datasheet or a report. We can also set property of a control at run time.
5. Changing the flow of operations: We can control the flow of operations by adding conditions that determine whether a macro action is carried out. If the condition evaluates to True, the corresponding action takes place. We can add the msgbox function to a macro condition to let the user decide which action to carry out.

Nested Macro: If we want to run one macro from another macro, 'RunMacro' action is used and 'macro name' argument should be set to the name of the macro that we want to run. With this RunMacro action, we can repeat the macro many times. The RunMacro action has two arguments in addition to the Macro Name:

1. Repeat Count: It specifies the maximum number of times the macro is to run.
2. Repeat Expression: It contains an expression that evaluates to true (−1) or false (0). The expression is evaluated each time the RunMacro action occurs. When it evaluates to False, the called macro stops.
   - The Repeat Count and Repeat Expression arguments work together to specify how many times the macro runs.
   - If both are blank, the macro runs only once.
   - If Repeat Count contains a number, but the Repeat Expression is blank, the macro runs the specified number of times.
   - If the Repeat Count is blank, but the Repeat Expression contains an expression, the macro runs until the expression evaluates to False.
   - If both arguments contain entries, the macro runs the specified number of times or until the expression evaluates to False, whichever occurs first.

When the called macro is finished, Access returns to the calling macro and runs the next action after RunMacro.

Create a Macro Group: If we create several macros that apply to controls on the same form or report, we can group them together as one file. Using macro groups offers two advantages:

1. It reduces the no. of macro names in the Navigation Pane.
2. We can find all the macros for a single form or report in one place, where they are easy to edit, if necessary.
   - Steps to create a macro group:
     1. Open the macro sheet and in the show/hide group, click the Macro Names command to display the Macro Name column.
     2. Add a macro to the sheet and enter a name for it in the Macro Name column of the first row of the macro.
     3. Add the rest of the actions to the macro.
     4. To add another macro, enter the macro name in the Macro Name column and add actions that you want to occur.
     5. After adding all the macros to the group, close and save it as usual with the group name.

To assign macro from a macro group to an event property of any control, select a specific macrogroupname.macroname from the drop down list box.

'Autokeys' Macro group: We can create a special macro group named 'autokeys', in which we can assign an action or a set of actions to a specific key or key combination. These work as the shortcut key that we can use to carry out a ribbon command. Pressing a key or combination of keys carries out the action that we specify. We can add as many individual macros to the group as we need, each one named with the key or key combination that runs it. The following table shows a list of valid Autokeys key combinations. The carat symbol (^) represents CTRL and the plus sign (+) represents SHIFT. Function keys and other key names are enclosed in curly brackets ({}). Table 11.19 shows examples of some autokeys.

A database can have only one autokeys named macro.

'Autoexec' Macro: We can create a special macro that runs when we first open a database. The 'autoexec' macro can carry out actions such as opening a form for data entry, displaying a message box prompting the user to enter his/her name or playing a sound greeting. To create 'autoexec' macro, create the macro with the actions that we want to carry out at start up and save it with the name 'autoexec'. A database can have only one autoexec named macro.

Steps to protect the database with a password:

1. Click Microsoft Office button and click Open.
2. Locate and select the database in the open dialog box. Then click the Open down arrow and choose Open Exclusive from the drop-down list.
3. With the database open, on the Database Tools tab's Database Tools group, click the Encrypt with Password command.
4. In the Set Database Password dialog box, enter the password and press TAB. Enter the password again to verify it.
5. Click OK.

Table 11.19 | Examples of Autokeys

Key Combination	Examples of Macro Name
CTRL with any letter or number key	^A, ^4
Any function key	{F1}
CTRL with any function key	^{f1}
SHIFT with any function key	+{f1}
INS	{INSERT}
CTRL with INS	^{INSERT}
SHIFT with INS	+{INSERT}
DEL	{DELETE} or {DEL}
CTRL-DEL	^{DELETE} or ^{DEL}
SHIFT-DEL	+{DELETE} or +{DEL}

Steps to remove the password from the database:

1. Click Microsoft Office button and click open.
2. Locate and select the database in the open dialog box. Then click the Open down arrow and choose Open Exclusive from the drop-down list.
3. With the database open, on the Database Tools tab's Database Tools group, click the 'Decrypt database' command.
4. In the Unset Database Password dialog box, enter the password and verify it.
5. Click OK.

Switchboard: A switchboard system for a database consists of a hierarchical arrangement of switchboard pages, including a main switchboard page that usually branches out to two or more subordinate pages. Each page consists of a set of items with commands that carry out a specified activity. Most items also include an argument that specifies which form to open, which report to preview, which macro to run, etc.

To create a switchboard, **do the following:**

1. Click the switchboard manager given on the database tools tab's database tools group.
2. The first switchboard manager dialog box starts with the mandatory default main switch-board page. We can rename the main switchboard page.
3. Then we can add new switchboard pages by clicking on 'new' option.
4. After creating all the pages, we can link these pages with other switchboard. To link pages with main switchboard, double click the main switchboard and add the created pages to link with main switchboard.
5. Similarly, we can double click on any other switchboard page and add items which we want to link with that page.
6. At the last level, we can put pages to open form in edit or add mode, to open report, run any macro or code, exit from the application, go to a particular switchboard, etc.

   After creating a switchboard we can change its formatting by opening it into design view because switchboard is created as a form.

To open the switchboard, when we start a database, do the following:

1. Click on MS Office Access button.
2. Click on 'Access Options' button.
3. Select 'Current Database' option.
4. Select switchboard from the 'Display form' drop down list box.

Difference between switchboard and navigation pane: Table 11.20 shows the difference between navigation pane and switchboard.

Creating Documentation: To create a documentation, for any MS-Access object do the following.

1. Click on Database Tools and select 'Database Documenter' from the 'Analyze' group.
2. The Documenter dialog box will be opened where you can select the object type such as forms, reports, macros.
3. After selecting object type, the object names of that type will be available in the list.

Table 11.20 | Switchboard vs. Navigation Pane

Navigation Pane	Switchboard
It is always available, even when closed.	We can close it.
All the objects are available to view.	We can restrict/limit the objects to view by creating a specific activity.
We can't hide the object's design view.	We can hide the object's design view and hence restrict the user to change the design.
We can't hide navigation pane when object is open.	We can hide switchboard when object is open.

4. Select the object name for which you want to prepare the documentation. If you want to create documentation for more than one object then you may select many objects from the list.
5. Click OK.
6. The documentation will be created as a report and you cannot save it or change it.
7. You may take the printout of that report or export it as a word, text, XML, HTML, or Snapshot Viewer file.

SUMMARY

- MS-Access is a Relational Database System.
- We can create tables, queries, forms, reports, macros, etc., in Access.
- Tables can be created using design view and datasheet view.
- Records can be inserted in a table only using datasheet view.
- Access is case insensitive.
- Blanks are allowed in the field names in Access, but when it is retrieved, it should be enclosed within square brackets.
- There are some templates available to create a table, which can also be modified.
- When we open Access, there is a pane available on left side, which is called navigation pane. It contains all user-defined objects such as tables, queries, forms, reports, macros. We can minimize navigation pane, but cannot close it.
- Different 10 data types such as text, memo, number, date/time, currency, autonumber, yes/no, OLE object, hyperlink and attachment are available in Access.
- Most of the data types, except Number fields with the Replication ID property, enabled OLE Object fields and attachment could be converted into any other data type.
- Each field data type has some properties.
- We can Filter and sort records in Access.
- Filter removes the records from the screen which do not meet the condition and displays only those records which you want to see. But records remain in the table.
- There are four ways to filter records—Common context filters, Filter by selection, Filter by form and Advanced filter/sort.
- We can create a query in MS-Access using query design view, query wizard or SQL view.
- Forms are used for viewing and entering data. We can create various types of forms in Access such as split form, multiple items, pivot chart.

- The data can be retrieved from the table, query or other form and displayed on the form. For that, the recordsource property of the form should be set with the name of table, query or existing form.
- To display data in each field of the form, the controlsource property should be set with the field name.
- There are five form sections—Detail, Form Header, Form Footer, Page Header and Page Footer.
- Form or report contains five property tabs—Format, Data, Event, Other and All.
- There are three types of controls which could be kept on the form or report—Bound control, Unbound control and Calculated control.
- We can set default property for any of the control.
- Some controls available in the 'controls' group of Access are—Textbox, Label, Command Button, Combo Box, Check Box, Option Button, Toggle Button, Option Group, Bound object frame, Unbound object frame, Image, Line, List box, Logo, Page break, Rectangle, Subform/subreport and Tab.
- Hierarchical form represents 1:M (1 to many) type of relationship. It consists of a main form and one or more subforms. The main form shows data from records on the 'one' side of a one-to-many relationship and the subforms show data from records on the 'many' side.
- We can put a hyperlink on the form, set tab order of controls, apply conditional formatting and create multiple pages.
- Reports are used for viewing and printing data in some specific format. There are four views available—Design View, Layout View, Print Preview and Report View.
- There are seven sections of a report—Detail, Report Header, Report Footer, Page Header, Page Footer, Group Header and Group Footer.
- We can create subreport within the main report.
- Also, parameterized report can be generated by using parameterized query as a recordsource property of a report.
- We can export report as a word file, text file, Snapshot Viewer file, HTML file, XML file or Save report in another database.
- We can also create charts in Access.
- A macro is a list of one or more actions that work together to carry out a particular task in response to an event.
- There are two types of macros: Standalone macro and Embedded macro.
- Standalone macros are individual Access objects which we can create by clicking on Macros group.
- Embedded macros are created within forms or reports for the specific control.
- Macro run within another macro is known as nested macro.
- A database can have only one autokeys and autoexec macro.
- Autokeys macro is used to define shortcut keys for actions.
- Autoexec macro is executed automatically when we open the database within which it is defined.
- We can assign password to the database to protect it. To assign a password, the database should be opened in exclusive mode.

Exercises

1. Write steps to create a new database in MS-Access. What is the extension of an Access database? How we can create a copy of existing Access database?
2. Which views are available in MS-Access to create a table? What is the difference between Design View and Datasheet View?
3. List and explain data types of MS-Access along with the size options.
4. What is the usefulness of Lookup? Write steps to create a Lookup Column?
5. What is a navigation pane?
6. Write steps to assign a primary key and foreign key in the table.
7. Write rules to give names to field or a table in MS-Access.
8. What is sub datasheet?
9. Write steps to create a database using database templates? Which templates are available in MS-Access 2007?
10. Discuss field properties of 'Text' data type.
11. Which field sizes are available with the 'Number' data type? Write the storage capacity of each.
12. Which formatting symbols can be used with all data types? Write significance of each.
13. List all 'date/time' formats with examples.
14. Which data types can be converted into other data types?
15. Fill in the blanks.
    - i. The extension of MS-Access 2007 database is _________.
    - ii. A character ________ causes the input mask to fill numbers from left to right instead of from right to left.
    - iii. A character ___________ forces MS-Access to display the character that immediately follows this character, which is same as enclosing a character in double quotation marks.
    - iv. A character ________ will display all characters that follow this character in uppercase.
    - v. To enter the value 079-26467760, the input mask property of a field should be set with _________.
    - vi. To enter the value (206) 555-TELE, the input mask property should be set with ___________.
    - vii. To enter the date in dd-mm-yyyy (*e.g.*, 30-Oct-2010) format, the input mask property should be set with _________.
    - viii. To display the positive values in green colour, negative values in red colour and zeros in blue colour, the format property of a field should be set with ____________.
    - ix. If we click on Create → Table then table will be opened in ___________ view.
    - x. Maximum _________ characters are allowed in field name.
    - xi. __________ character is not allowed at the first position in field name.
    - xii. Duplicate and Null values are not allowed in a __________ key.
    - xiii. Processed data are called _______________.
16. Write a brief note on sorting.
17. State the difference between finding records and filtering records.

18. Which are the four ways to filter records? Define each.
19. What is the difference between removing and clearing a filter?
20. Differentiate between Filter and Query.
21. Discuss three types of select queries (Simple select query, Find duplicate query and Find unmatched query) available in MS-Access.
22. Discuss three types of special purpose queries (parameter query, Autolookup query and Crosstab query) available in MS-Access.
23. Discuss Action queries (Update query, Append query, Delete and Make table query) available in MS-Access.
24. Which are the four types of SQL-specific queries (Union query, Pass-through query, Data-definition query and Subquery)? Define all.
25. Write steps to create a query using query wizard option.
26. Write steps to create a query using query design option.
27. In query design view, when you move out of the criteria cell after entering the expression, Access automatically parses the expression and inserts characters to complete the syntax:
    i. ________ around field names
    ii. _________ around dates
    iii. _______ around text
    iv. _________ before a calculated field expression
28. Fill in the blanks.
    i. The ________ process screens the records and displays only those that meet the criteria.
    ii. The _______ is a set of conditions.
    iii. Filter by ________ leaves only the records with the same value as the one you select in one of the records or the records that do not include the same value.
    iv. Filter by _______ screens records with the criteria you enter into a table skeleton.
    v. _________ filters are not available for yes/no, OLE object or attachment fields.
    vi. The statement 'When we apply or remove a filter to/from a datasheet, the same is applied to any sub-datasheet within it', is _________ (true/false).
29. Explain the LIKE operator with all the wildcards used to specify the pattern. Give examples of each.
30. What is a Recordsource? How can we set Recordsource for the Form?
31. Explain different design controls available in MS-Access (Textbox, Listbox, Option Button, etc.). Also, categorize the controls as bound or unbound controls.
32. Write steps to create a new form using Form/Split Form/Multiple Items/Datasheet structure.
33. Which properties should be set to select a record set, to bind any control with the data field, allow user to view data and allow user to edit or view data?
34. Explain different sections (Detail, Form header, Form footer, Page header and Page footer) of Form.
35. Explain procedure to create a hierarchical form from related tables.
36. How many sections are there in a report? Which are they?
37. What is a subreport? Write steps to create subreport or hierarchical report.

38. What is conditional formatting? List steps to write conditional formatting on a column.
39. What is a parameterized report? Write steps to create a parameterized report. Write steps to print parameters in the report header.
40. Which type of information is represented by Pivot table and Pivot chart?
41. Which are the two types of macros? Explain both types.
42. What is a macro group? Write steps to refer a particular macro from a macro group.
43. What is a switchboard? Write steps to create a switchboard with options forms, reports and exit.
44. Differentiate between switchboard and navigation pane.
45. Write steps to create documentation of any macro.
46. Tick the correct answer.

 i. The ______________ property changes column heading.
  a. Field size b. Caption
  c. Validation rule d. Format

 ii. The left pane MS-Access, which displays all the objects, is known as ______________.
  a. Toolbar b. Ribbon
  c. Navigation pane d. Form

 iii. Which property specifies the value that automatically appears when new record is inserted?
  a. Caption b. Default
  c. Format d. Field size

 iv. MS-Access is case ______________for field names.
  a. Sensitive b. Insensitive

 v. We can enter records in a __________view.
  a. design b. datasheet
  c. report d. print

 vi. In a report, page header and page footer are displayed only in the ___________ view.
  a. print preview b. datasheet
  c. report d. layout

 vii. __________ are used for entering data.
  a. Forms b. Reports
  c. Macros d. Modules

 viii. Macro which is created within another macro is known as ________macro.
  a. Nested b. Embedded
  c. Linked d. Super

 ix. The database can have _____________autoexec macro.
  a. Many b. Only one
  c. Two d. Three

 x. To secure the database with a password, the database should be open in which mode?
  a. Read only b. Exclusive
  c. Non exclusive d. Write

LAB ACTIVITIES

1. Write and store a query which will display the details of faculty name, faculty's birth date, subject name and number of lectures assigned. Create a form with appropriate headers and footers to display the details from this stored query. Select 'Justified' Layout and 'Concourse' style for this form. Also display retirement date of faculty using a calculated control. Display logo of an organization.

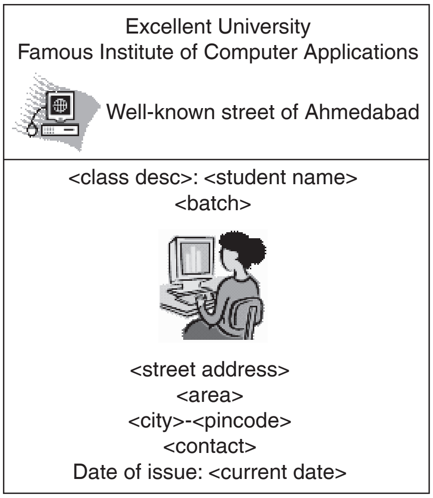
   Faculty (**faculty ID**, facultyname, birthdate, photo, gender)
   Subject (**subject ID**, subjectname, nooflectures)

2. Create a form using split form option which will display the details of colleges. Set appropriate headers and footers. Put command buttons to add and delete records and to close the form. Format the form properly. Put a hyperlink which will open the word document named 'conditionst.doc'. Use option button control to input registered and combo box control to input city of a college. Display logo of an organization.

   College (**college ID**, collegename, address, city, pincode, registered, photo)

3. Create a report to generate I-cards of each student in the following format using following tables:

   Class(classcode, classdesc)
   Student (stdno, stdname, classcode, street_address, area, city, pincode, contact, photo, batch, gender)


Excellent University
Famous Institute of Computer Applications
Well-known street of Ahmedabad

<class desc>: <student name>
<batch>

<street address>
<area>
<city>-<pincode>
<contact>
Date of issue: <current date>

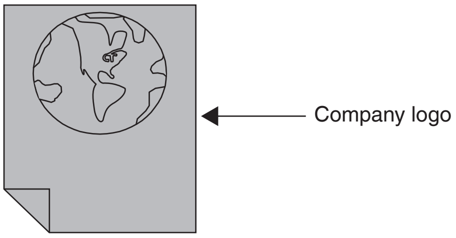
4. Create a chart which will show category of stationary on x-axis and total number of stationeries of particular category on y-axis. Format the graph properly.
5. Create a hierarchical form which will allow to insert and edit details of stationeries category-wise. Put a command button on this form which will open the chart created in Ex. no. 3. The chart must be changed if we change records throughout the form.

6. Create a summary report which will display annual total amount of sales, average amount of sales, maximum order amount and minimum order amount. Also display overall total amount of sales, average amount of sales, maximum order amount and minimum order amount.
7. Create a report which will take from date and to date as an input and will display details of item wise sales with summary.

   Item (**itemcode**, itemname, price, unit_of_measurement)
   Order (**orderno**, orderdate, customerno, total_order_amt)
   Sold_items (**orderno**, **itemno**, qty, total_amt)
   Customer (**custno**, custname)

8. Create a report with suitable headers and footers which will display sales report of each area of each region of each zone. Display salesman's name only once for each item sold by him. If total sales amount of any salesman exceeds ₹1000000.00, then display it in bold, italic and 14-size coloured font.

   Tables: zone (**zone_no**, zone_name)
   region (**region_code**, region_name, zone_no)
   area (**area_no**, area_name, region_no)
   salesman (**salesman_no**, salesman_name, area_no)
   sales (**salesman_no**, **itemno**, total_qty_sold, total_amount)
   item (**itemno**, itemname)



Zone-wise detailed Sales Report

Zone: ________ **Date: ________**

Region: ________

Area: ___________

Salesman Name	Item name	Amount (in ₹)
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------

Area: ____________

Salesman Name	Item name	Amount (in ₹)
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------

Region: ________

Area: ___________

Salesman Name	Item name	Amount (in ₹)
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------

Area: ___________

Salesman Name	Item name	Amount (in ₹)
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------

Zone: ________

Region: ________

Area: ___________

Salesman Name	Item name	Amount (in ₹)
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------

Area: ____________

Salesman Name	Item name	Amount (in ₹)
-------------------	-------------	--------------------
-------------------	-------------	--------------------
-------------------	-------------	--------------------

9. Create a hierarchical report which will display list of customers and details of orders of those customers. Display serial numbers with each record of order and with each record of a customer. Display total number of customers and total number of orders given by each customer. Display count of orders in blue colour for the highest count. Don't show duplicate values in any column within a group.

   Customer (custno, custname)
   Order (orderno, custno, orderdate, total_order_amt)

10. Create a main report which contains details of different categories of stationeries. Embed a subreport which contains details of stationeries of particular category and details of faculty members to whom that stationary is issued. Display serial numbers with each record, count of total categories, count of total no. of stationeries within the category and count of total no. of faculty members to whom particular stationary is issued. Put a command button to close the report.

Category

Categoryid	Categoryname
C1	Paper
C2	Storage device
C3	File

Stationery

Stat ID	Stat Name	Cat ID
S1	Ruled	C1
S2	Blank	C1
S5	Pen Drive	C2
S6	Spring	C3

Issued to

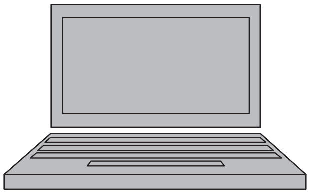
Faculty ID	Stat ID	Issued Date	Quantity
F1	S5	11-09-2005	25
F2	S7	12-9-2001	2

11. Create a parameterized report which will display I-card of user-entered roll no. Display inputted roll number with proper message in the report header.
12. Create a report which will generate certificates for each participant with the following details. Tables: student (stdno, name, classcode, gender)

    class (**classcode**, classdesc)
    event (**eventID**, eventname)
    participant (**stdno**, **classID**, **eventide**, participant_name, gender)
    winner (**eventID**, **stdno**, **classID**, position)

    Display the last line 'He/she stood ________ in the event' if and only if the value of 'position' field is not null.

ABC Institute of Computer Applications



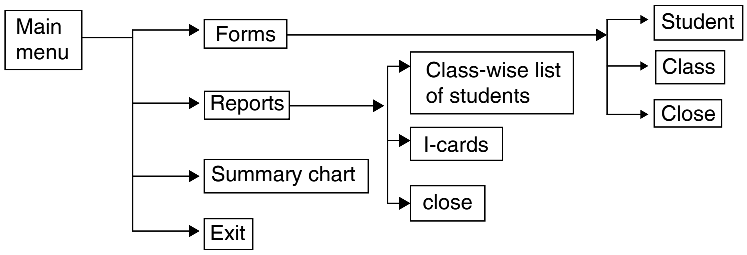
Talent Evening 2012

Certificate

This is to certify that Mr/Ms ______________________ of class _____________ has participated in the event _________________ in the talent evening organized by ABC Institute of Computer Applications on 12th December, 2012. He/She stood _________ in the event.

13. Create a standalone macro to close the specified report. Embed this macro into the click event of a command button which is placed on the report which you want to close after viewing.
14. Put a command button in the student form, and create an embedded macro for the click event of this command button which will open the report of student details for student names which starts with letter 'a'.

15. Put a command button in the student form and create an embedded macro for the click event of this command button which will open the report of student details for student numbers between 1 and 10.
16. Create a stand-alone macro named 'search_report' which contains two actions 'navigateto' and 'selectobject' to search the specified object from the navigation pane.
17. Create a blank form. Set the 'unload' event of this form with the macro created in Ex. 13.
18. Create a stand-alone macro which contains two actions 'openreport' and 'setproperty'. Open the report which contains student details using 'openreport' action. Add a condition in the condition column which will check whether student name is null or not. If the student name is not null, then using 'setproperty' action set the forecolour property of a student name textbox to red colour.
19. Create a stand-alone macro named check_stdno which will check whether the entered student number lies in a specific range or not. Write a condition in the condition sheet of a macro to check the entered stdno in a 'student' form is ≥100 or not. If the number is ≥100, then display the message using a message box that 'The number is invalid… please enter the number between 1 and 99'. Bind the macro with the 'on lost focus' event property of stdno textbox.
20. Create a stand-alone macro named check_stdname which will check whether the entered student name is null or not. Write a condition in the condition sheet of a macro to check the entered stdname in a 'student' form. If the name is null, then display the message using a message box that 'The name can't be null… please enter the valid name'. Bind the macro with the 'on lost focus' event property of stdname textbox.
21. Create a macro group named check_marks_data which contains three macros named chk_stdno to check whether the stdno <0, chk_marks to check marks ≥ 100 and chk_classid to check classcode is null or not. Display proper messages if any of the condition is false. Bind the proper macro from the created macro group with the 'lost focus' event properties of stdno, classcode and marks controls on a marks form.
22. Create an autoexec macro which will display the message welcome to my database! When the user opens the database, set password to open the database.
23. Create a switchboard which will display the following hierarchy:



12

CHAPTER

Overview of Oracle

CHAPTER OBJECTIVES

- Understanding Oracle as a relational database.
- Knowing commands of Oracle Database 10g Express Edition.
- Learning how to manage users, roles and privileges.

12.1 | ORACLE AS AN RDBMS

Oracle is a relational database management system (RDBMS) which also supports object-oriented features.

It allows creating tables and setting relationships between tables. Users can write stored procedures and functions with the use of PL/SQL. Automatic transaction support is provided by Oracle. It allows developing applications which can run on the internet. Backup and recovery features are available in Oracle.

12.2 | LOGGING INTO ORACLE

Oracle provides 'Oracle Database 10g Express Edition (Oracle Database 10g XE)' which can be downloaded for free to use. After logging into it, user can execute all SQL and PL/SQL statements. When installing Oracle Database 10g XE, the default user named 'system' is created with administrator's rights. At the time of installation, user has to provide password for 'system' user. On completion of installation, user can log in using username 'system' and the password (which is specified during installation).

After installation, user can open Oracle Database 10g XE in the Windows operating system by following the steps given below.

1. Click on Start.
2. Select Programs → Oracle Database 10g XE → Select Go To Database Home Page.
3. The login form will be displayed in which enter user name 'system' and password which you have given at the time of installation. Let us assume that the password given

during installation for 'system' user is 'admin'. The window will look as specified in Figure 12.1.

After successful login, the window given in Figure 12.2 will appear. Here, user can select SQL → SQL commands → Enter command to type any SQL or PL/SQL statements. After typing the statement, it can be executed by selecting and clicking on 'Run' button. The output will be displayed in the window given below the command window. It will look as the snapshot shown in Figure 12.3.

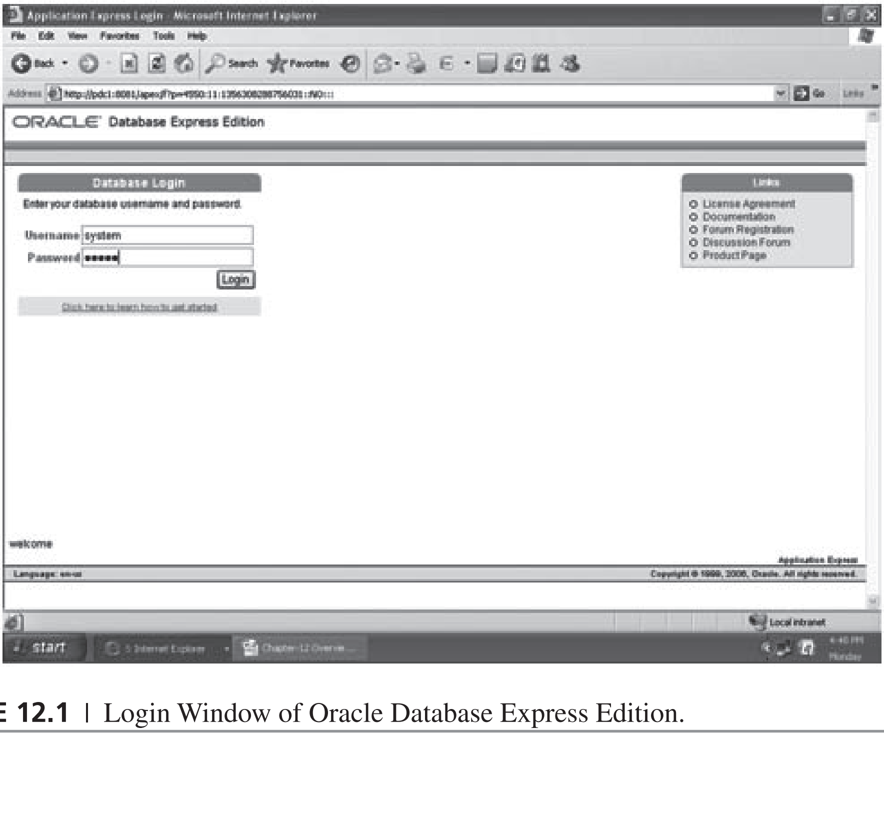
To logout from the Oracle, click on 'Logout' link given on the right side top of the command window. The written statements or commands can be saved using 'save' option.

The 'system' user has administrator's rights which allow him/her to create other databases, users, roles, etc.

12.3 | COMMAND SUMMARY OF ORACLE DATABASE 10G XE

After logging into the Oracle Database 10g XE, there are main five options available. These are as:

1. **Administration:** Using this option, one can view and change storage and memory settings, create new users and manage users which are already created, monitor sessions, operations and system statistics, view details about current database and change his/her own password. Following is a list of commands which is available when we click on the 'Administration' option.



Enter your database username and password.

Username: system

Password: •••••

FIGURE 12.1 | Login Window of Oracle Database Express Edition.

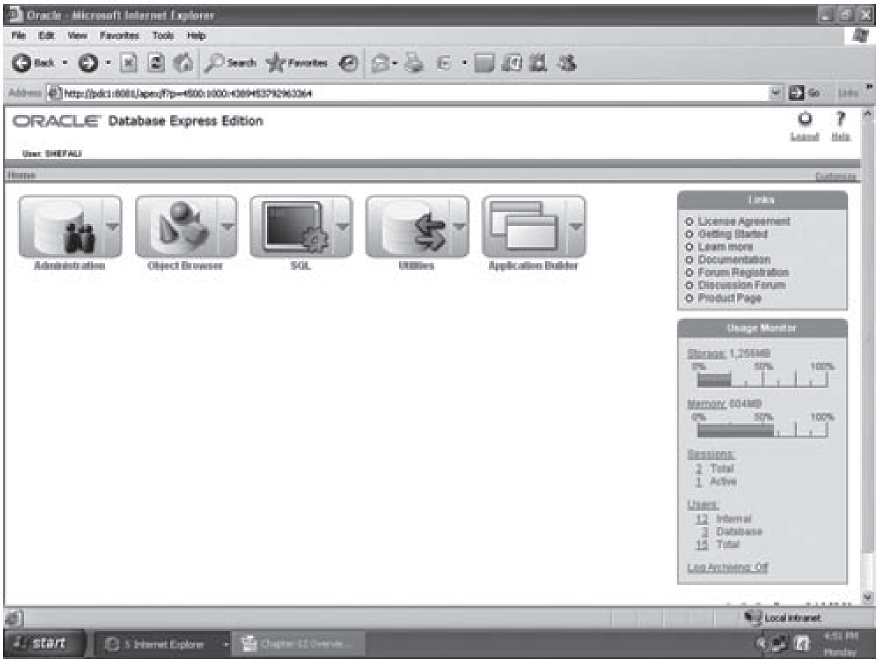
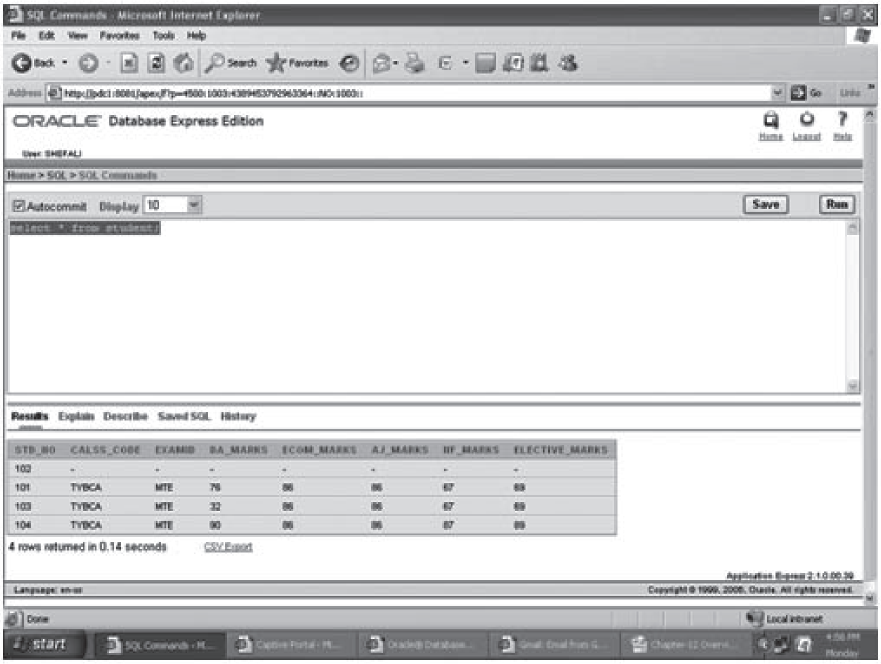


FIGURE 12.2 | Oracle Database Express Edition.


SQL Commands - Microsoft Internet Explorer
ORACLE Database Express Edition
User: SHEFALI
Home > SQL > SQL Commands
Autocommit
Display 10
Save
Run
Results
Explain
Describe
Saved SQL
History

```
select * from student;
```

STD_NO	CALSS_CODE	EXAMID	BA_MARKS	ECOM_MARKS	AJ_MARKS	IIT_MARKS	ELECTIVE_MARKS
102	-	-	-	-	-	-	-
101	TYBCA	MTE	76	86	86	67	69
103	TYBCA	MTE	32	86	86	67	69
104	TYBCA	MTE	90	86	86	87	89

4 rows returned in 0.14 seconds CSV Export

Language: en-us

Application Express 2.1.0.00.39
Copyright © 1999, 2006, Oracle. All rights reserved.

FIGURE 12.3 | Writing and Executing Command in Oracle Database Express Edition.

i. Storage: It has the following suboptions.
    a. Storage utilization: This option shows the statistics regarding space utilized.
    b. View tablespaces: It shows details of tablespaces.
    c. View datafiles: It shows details of datafiles.
    d. Compact storage: It gives the option to do compact storage using which the unused free space in the database can be recovered.
    e. View logging status: It shows details of database log files.

ii. Memory: It shows the following options related to memory.
    a. Memory utilization: It shows the memory allocation to System Global Area (SGA) and Program Global Area (PGA).
    b. Manage SGA: It shows memory allocation to the components of SGA. The System Global Area (SGA) is a memory area that contains data shared between all database users such as buffer cache and a shared pool.
    c. Manage PGA: It shows memory allocation statistics of PGA. The Program Global Area (PGA) is a memory buffer that is allocated for each individual database session and it contains session specific information.

iii. Database users: It shows the following options using which we can create and manage database users.
    a. Manage users: It shows all the users of database.
    b. Create users: New users can be created by assigning roles and privileges using this option. Also, existing user's privileges, role and other settings can also be changed using this option.

iv. Monitor: It shows the following options.
    a. Sessions: It shows user-wise session details.
    b. System statistics: It shows physical and logical I/O settings statistics, cursor statistics, transaction statistics, memory statistics and time statistics.
    c. Top SQL: It shows the details of SQL execution in a specific order such as disk and bugger read wise, execution wise, CPU time wise, etc.
    d. Long operations: It shows details of long operations.

v. About database: It shows the following options.
    a. Version: It shows version of database.
    b. Settings: It shows settings of database.
    c. National language support: It shows details of national language supported by database.
    d. CGI environment: Shows details of CGI environment parameters such as PL/SQL gateway, request and server protocol, remote user name, etc.
    e. Parameters: It shows parameters of init.ora file and their values.

vi. Change my password: Using this option, user can change his/her password.

vii. Manage login message: User can set login message which is to be displayed on the login page.

viii. Manage HTTP access: It is used to control HTTP access as local or remote for the existing database.

2. **Object Browser:** It is used to create and browse various objects such as tables, views, types, etc. Following is a list of commands which is available when we click on 'Object Browser' option.
   i. Create: It shows the following options using which user can create and manage various objects of database.
      a. Tables: It is used to create a table.
      b. Views: Used to create a view from base table.
      c. Indexes: Used to create an index on table.
      d. Sequences: Used to create a sequence which can be used to auto increment or decrement value of any column of a table.
      e. Types: Used to create a type which is equivalent to class.
      f. Packages: Used to create a package. Package is a combination of functions and procedures.
      g. Procedures: Used to create a procedure which is a named PL/SQL block.
      h. Functions: Used to create a function which is also a named PL/SQL block.
      i. Triggers: Used to create trigger on any table which is fired automatically when any manipulation is made in a record of that table.
      j. Database links: Used to create database link.
      k. Materialized views: Used to create a view which is stored physically in the database.
      l. Synonyms: Used to create a synonym of a table which is a copy of table.

   ii. Browse: It shows the following options.
      a. Tables: Used to search existing tables.
      b. Views: Used to search existing views.
      c. Indexes: Used to search existing indexes.
      d. Sequences: Used to search existing sequences.
      e. Types: Used to search existing types.
      f. Packages: Used to search existing packages.
      g. Procedures: Used to search existing procedures.
      h. Functions: Used to search existing functions.
      i. Triggers: Used to search existing triggers.
      j. Database links: Used to search existing database links.
      k. Materialized views: Used to search existing materialized views.
      l. Synonyms: Used to search existing synonyms.

3. **SQL:** It is used to perform SQL-related tasks. By clicking on SQL, it shows the following options.
   i. SQL commands: It shows the following option.
      a. Enter command: By clicking on this option, a window opens in which user can write and execute any SQL command or PL/SQL block. Also, user can save it as SQL file, view history, etc.

ii. SQL scripts: It shows the following options.
  a. Create: Used to create or download SQL script.
  b. Upload: Used to upload the existing SQL script.
  c. View: Used to view the existing SQL script.
  d. Export: Used to export SQL script from current database to any other database.
  e. Import: Used to import SQL script from any other database into the existing database.

iii. Query builder: It shows the following options.
  a. Create: It is used to create and execute query automatically by selecting fields and selecting SELECT clauses from the drop down list boxes. It is just like wizard.
  b. View saved queries: It is used to view existing queries.

4. **Utilities:** It gives the utilities such as load or unload data, generates data definition commands from the existing objects, shows object reports, etc. When we click 'Utilities', it gives the following options.
   i. Data load/unload: It shows the following options.
     a. Load: Used to load data into existing or new table from the text file, excel sheet or XML file.
     b. Unload: Used to unload (here it means copy) data from existing table into text file or XML file.
     c. Repository: It shows the status of loaded text data.
   ii. Generate DDL: Used to generate data definition command of any existing database object. For example, if we want to create table syntax of any existing table, then the definition (create table command) of that table will be generated automatically by selecting that table from the list of existing tables.
   iii. Object reports: It displays reports of the following existing database objects.
     a. Tables: It displays the following options.
       - Columns: Shows table-wise list of columns.
       - Comments: Shows comments written on tables.
       - Constraints: Shows table wise constraints.
       - Statistics: Shows the statistics of tables such as when the table last analyzed, etc.
       - Storage sizes: Shows storage size of existing tables.
       - Exception reports: It shows the following options.
         - No indexes: Displays list of tables without indexes.
         - No primary key: Displays list of tables without primary key.
         - Unindexed foreign keys: Displays list of tables with unindexed foreign keys.
     b. PL/SQL: It shows the following options.
       - Program unit arguments: It shows details of package arguments.
       - Program unit line counts: It shows total lines in objects such as procedures, packages, types, triggers, etc.
       - Search source code: It allows searching code for the selected object.
     c. Security: It shows the following options.
       - User privileges: It shows details of privileges granted to each user.
       - Role privileges: It shows details of roles granted to each user.
       - System privileges: It shows details of system privileges.

d. All objects: It shows the following options.
   - All objects: It shows details of all the objects created within the database.
   - Invalid objects: It shows the details of invalid objects, *i.e.*, objects which are not created successfully or objects which are created with some errors.
   - Object counts by type: It shows total no. of each type of objects created within the database.
   - Object creation calendar: It shows names and types of objects which are created or modified on a particular date in the calendar.

e. Query data dictionary: It shows all the views and their description which describe the metadata.

iv. Recycle bin: When user drops any table, Oracle does not remove it, but renames the table and keeps it in the Recycle Bin. The dropped object can be recovered later from here.

5. **Application Builder:** It contains some sample applications which user can run. It also allows user to create new applications. Application builder will be available to only those users who have admin rights. Application builder contains the following options.

   i. Create application: It shows the following options.
      a. Create application: Used to create new application.
      b. Create from spreadsheet: Used to create new application from spreadsheet (excel file) data.

   ii. View application: It is used to view existing applications.

   iii. Demonstrations: It is used to execute sample applications which are already available in Oracle.

   iv. Application administration: It shows the following options.
      a. Manage services: It is used to manage services such as sessions, logs, etc.
      b. Manage application Express Edition users: It is used to create and view users, create and view groups and assign user to a specific group.
      c. Monitor activity: It is used to monitor different activities such as page views, sessions and application changes.
      d. Email configuration: It is used to change email settings to manage email logs and email queue.
      e. About application Express: It shows the details such as current user name, host schema name, time of last DDL executed, etc.

   v. Import: It shows the following options.
      a. Application: It is used to import any application file which can be installed after it is imported.
      b. Page: It is used to import any Page (same as application) which can be installed after it is imported.
      c. Theme: It is used to import any theme file which can be installed after it is imported.
      d. User interface defaults: It is used to import any user interface defaults which can be installed after it is imported.

vi. Export: It shows the following options.
  a. Application: It is used to export any application file from the database.
  b. User interface defaults: It is used to export user interface defaults from the database.

12.4 | DATABASE ADMINISTRATION

12.4.1 | Managing Users

The user with administrator's rights can create new users and update the existing users. To create a new user, one has to login as SYS or SYSTEM user. Database users can use the database and can perform tasks as per their roles or privileges assigned to them. To create a new user, the following syntax is used:

```
CREATE USER <user_name>
IDENTIFIED {BY PASSWORD/EXTERNALLY/GLOBALLY
                AS '<directory_name>'
DEFAULT TABLESPACE <tablespace_name>
TEMPORARY TABLESPACE <tablespace_name>
QUOTA {<size>/UNLIMITED} ON <tablespace_name}
PROFILE <profile_name>
PASSWORD EXPIRE
ACCOUNT {LOCK/UNLOCK};
```

The clauses of CREATE USER command are explained below:

1. **CREATE USER:** After CREATE USER, name of the user is specified which should be any valid name.
2. **IDENTIFIED:** It is a compulsory clause of CREATE USER command; all other clauses are optional. There are three options which we can specify after IDENTIFIED keyword.
   i. BY PASSWORD: In this clause, the password of user should be specified within double quotation marks. When we specify this option, the local user is created.
   ii. EXTERNALLY: This option is specified to create an external user. This type of user is authenticated externally (not by Oracle database) by operating system.
   iii. GLOBALLY AS '<directory_name>': This option creates a global user. This type of user should be authorized by the enterprise directory.
3. **DEFAULT TABLESPACE:** The objects which will be created by this user will also be stored in the tablespace specified after this clause. If this clause is not written, the objects created by this user will be stored into default tablespace of the database.
4. **TEMPORARY TABLESPACE:** The user's temporary segments will be stored into the tablespace which is specified after this clause. If this clause is not written, the temporary segments are stored into database's default tablespace or into SYSTEM tablespace.

5. **QUOTA…ON:** This clause is used to specify amount of space to be allocated by user in the specific tablespace. There are two options for size—either UNLIMITED or specify size into bytes. After ON keyword, tablespace name should be written. We can't specify this clause for temporary tablespace. If we don't write this clause, unlimited quota is allocated to user for the tablespaces.
6. **PROFILE:** It is used to specify the profile name which we want to assign to user. If we don't specify this clause, the default profile will be assigned to user.
7. **PASSWORD EXPIRE:** It causes the user's password to expire immediately after user is created and will prompt the user to set new password when he/she logs in to the database.
8. **ACCOUNT:** There are two options—LOCK or UNLOCK. If we don't specify this clause, the default setting is unlock. If we specify 'lock' option, the user's account will be locked.

We can also change or update the existing user by writing ALTER USER command. The clauses which are specified in the CREATE USER command are used in this command also. The GRANT/REVOKE THROUGH clause is additional which will allow user to connect through proxy user.

To drop the existing user, the command is DROP USER <user_name>. Table 12.1 shows some examples to create, alter and drop users with different options.

If we want to see the details of a particular user, it can be displayed by selecting specific columns or all columns from the view named dba_users. Following is a command which will display all user names and their passwords (in encrypted form) of current database.

```
select username, password from dba_users;
```

Oracle database converts and stores all the data in uppercase. Therefore, whenever we specify some values in WHERE condition, it must be written in uppercase. The following command will display password of user 'jisha' in encrypted form.

```
select password from dba_users, where username='JISHA';
```

12.4.2 | Managing Roles

Role is a set of privileges which can be granted to users or other roles. When any role is created, it doesn't contain any privilege. The privileges can be granted to a specific role by using GRANT statement.

Following is a syntax of CREATE ROLE.

```
CREATE ROLE <role_name> {NOT IDENTIFIED/
        IDENTIFIED {BY <password>/USING
       <package_name>/EXTERNALLY/
        GLOBALLY}};
```

NOT IDENTIFIED shows that there is no need of password to use this role. It is authorized by the database itself. IDENTIFIED shows that user must be authorized by password externally or globally or using some packages. Following is an example.

```
CREATE ROLE newadmin NOT IDENTIFIED;
```

Table 12.1 | Examples of CREATE USER command

Example	Meaning
`create user shefali identified by 'mahadev'`	It will create a local user named shefali with the password 'mahadev'.
`create user shefali_ex identified externally;`	It will create an external user named shefali_ex who will be authenticated by operating system, but the user name should match with the operating system's user name.
`create user jisha identified by fairy default tablespace users temporary tablespace temp quota 20m on system quota 10m on users password expire account unlock;`	The user name 'jisha' will be created with password 'fairy' and with the specified settings. Her password will be expired immediately after user is created. When she will login with user name 'jisha', Oracle will prompt to set new password 'jisha'. User's default tablespace will be 'users' and temporary tablespace will be 'temp'. The user 'jisha' will get maximum 20MB on 'system' tablespace and '10MB' on 'users' tablespace.
`create profile sheev_profile limit failed_login_attempts 3 password_life_time 3 password_reuse_time 2 password_reuse_max 2 password_lock_time 1/24 password_grace_time 20;` `create user harsheev identified by champ profile sheev_profile;`	When we want to assign profile to a user, the profile should be created first before we create a user. Here, sheev_profile has been created by executing the command 'CREATE PROFILE' with some parameters. Then, the user name 'harsheev' is created. **Note:** After creating and assigning profile to the user, the database initialization parameter RESOURCE_LIMIT should be set to TRUE, otherwise it will not be enforced.
`alter user harsheev grant connect through jisha;`	It will change the current user 'harsheev' and will allow him to connect with the database through the user 'jisha'.
`alter user jisha identified by elegant;`	It will change the password of 'jisha' and set the new password as 'elegant'.
`create user tybca009 identified by tybca009;`	The user name 'tybca009' will be created with password 'tybca009'.
`drop user tybca009;`	And then it will be dropped.

After role is created, privileges should be granted to the role using GRANT statement as following.

```
GRANT CREATE TABLE to newadmin;
```

This statement will grant privilege to create a table to the role newadmin. Now, the user can create a table to whom this newadmin role is assigned.

Role can be assigned to the user using GRANT statement as following.

```
GRANT newadmin to jisha;
```

The above grant statement will assign newadmin role to the user named 'jisha'.

Many privileges can be assigned together to a specific role. For example,

```
GRANT INSERT, UPDATE on student to newadmin;
```

The privileges can also be granted directly to the users. For example,

```
GRANT create table to jisha, harsheev;
```

This is not convenient when many privileges should be granted to selected users or group of users. At that time, roles are useful.

Following are advantages of role.

1. Many privileges can be granted to a role and then the common role can be granted to many users.
2. User doesn't have to remember which roles should be assigned to which users.
3. Also, it is easy to revoke the set of privileges which are assigned to a role.

There are some predefined roles in Oracle database such as CONNECT, DBA, RESOURCE, EXP_FULL_DATABASE, etc. These roles can be assigned to users directly using GRANT statement. For example, the following statement will grant the role CONNECT to user named 'harsheev'.

```
GRANT CONNECT to harsheev;
```

To view all the predefined roles and roles which are created by user, the data dictionary view dba_roles is used as follows.

```
SELECT * from dba_roles;
```

The user who is given the role with the ADMIN OPTION or given the system privilege GRANT ANY ROLE can grant role to users.

The role assigned can be revoked using REVOKE statement as following. Execute the following sequence of statements to create a user and a role, grant a role to the user and revoke a role from the user.

```
create user harsheev identified by generous;
create role newadmin;
grant connect to newadmin;
grant newadmin to harsheev;
REVOKE newadmin from harsheev;
```

12.4.3 | Managing Privileges

After logging into the database, the user can perform only those tasks for which he/she is given privileges. Privileges can be assigned to an individual user, group of users or roles. Role is a group of privileges. We can assign a name to the role.

There are two types of privileges in Oracle: System privileges and Object privileges.

1. **System privilege:** There are many system privileges which allow user to perform a specific database operation. The user, who is given the system privilege with the ADMIN OPTION or given the system privilege ANY PRIVILEGE, is able to grant privileges to other users or roles. The syntax to grant system privilege is:

   ```
   Grant <privilege_name> to <user_name>;
   ```

Following grant statement will grant the system privilege 'create user' to the user 'shefali'. After having this privilege, the user named 'shefali' can create new users.

```
grant create user to shefali;
```

Many privileges can be assigned together as following.

```
grant create any table, drop any table to shefali;
```

Similarly, one or more privileges can be granted to many users. The following grant statement will grant two system privileges create any table and drop any table to two users —shefali and jisha.

```
grant create any table, drop any table to shefali, jisha;
```

The following grant statement will grant all privileges to user shefali.

```
grant all privileges to shefali;
```

If we want to take back the privilege which is assigned to any user, then REVOKE statement is used as following.

```
revoke all privileges from shefali;
revoke drop any table from jisha;
```

The following statement will grant privilege 'group any table' to role newadmin.

```
grant drop any table to newadmin;
```

After assigning privilege to a role, a role can be assigned to user as following.

```
grant newadmin to jisha;
```

Role and privilege can be assigned together to a user as following. The privilege 'create any table' and a role newadmin is granted to user 'harsheev'.

```
grant create any table, newadmin to harsheev;
```

2. **Object privilege:** Object privileges can be granted to user and role. The user who has created any object can grant privilege of that object to other users. Also, the user who has the privilege 'GRANT ANY OBJECT PRIVILEGE' can grant any object to other users or the user who was granted 'WITH GRANT OPTION' on object can grant privilege to use that object to other users. For example,

```
grant insert, select, delete on student to jisha, harsheev;
```

Here, 'student' is a table which is an object. The above statement will allow users 'jisha' and 'harsheev' to insert, select or delete records to/from table 'student'.

```
grant all on student to jisha;
```

The above statement will grant all the object privileges on table 'student' to the user 'jisha'.

```
grant insert(stdno), select, delete on student to jisha, harsheev;
```

The above statement will allow user 'jisha' and 'harsheev' to insert value only in the field 'stdno' of 'student' table and will allow to 'delete' and 'select' records from/to 'student' table.

The given object privileges can be taken back from the user by using 'REVOKE' statement as follows. The following statement will revoke the object privilege SELECT and DELETE on STUDENT table from user 'harsheev'.

```
Revoke delete, select on student from harsheev;
```

The following statement will grant all the object privileges to the role 'newadmin'.

```
grant all on student to newadmin;
```

Later, the role 'newadmin' can be assigned to any user (*e.g.*, 'harsheev') using grant statement as the following one.

```
grant newadmin to harsheev;
```

The granted role can be revoked using REVOKE statement.

SUMMARY

- Oracle Database 10g XE can be downloaded free from the internet and can be used for writing and executing SQL queries and small PL/SQL blocks.
- User has to provide user name and password when installing Oracle. The same user name and password can be used afterwards to work in Express Edition of Oracle.
- Five components are available within Oracle XE which are Administration, Object Browser, Utilities, SQL and Application Builder. The fifth component Application Builder is available only if the user had admin rights.
- SQL component is used to write and execute queries and PL/SQL blocks.
- SYS and SYSTEM users can create new users, profiles and roles using create user, create profile and create role commands, respectively.
- Using 'grant' command, the roles and privileges can be assigned to users. Using 'revoke' command, the assigned roles and privileges can be withdrawn from the users.
- There are two types of privileges—system and object.

EXERCISES

1. Explain the features of Oracle as a relational database management system.
2. Write steps to login into Oracle 10g XE.
3. Write syntaxes of 'create user', 'create role' and 'create profile'. Explain all these users with examples.
4. Discuss system and object privileges.
5. List advantages of creating role.
6. Select the correct answer from the following multiple choices.
   i. ______________ is a relational database management system which also supports object-oriented features.

   a. IMS b. Dbase
   c. Oracle d. Sybase

   ii. Using which option, one can view and change storage and memory settings, create new users and manage users in Oracle 10g?

   a. Administration b. Utilities
   c. SQL d. Object Browser

iii. Using which option, one can create and browse various objects such as tables, views, types, etc., in Oracle 10g?
   a. Administration
   b. Utilities
   c. Application Builder
   d. Object Browser

iv. Using which option, one can write and execute queries in Oracle 10g?
   a. Administration
   b. Utilities
   c. SQL
   d. Object Browser

v. Using which option, one can create and browse various objects such as tables, views, types, etc., in Oracle 10g?
   a. Administration
   b. Utilities
   c. SQL
   d. Object Browser

vi. Using which option, one can load or unload data, generate data definition commands from the existing objects, view object reports, create and browse various objects such as tables, views, types, etc., in Oracle 10g?
   a. Administration
   b. Utilities
   c. SQL
   d. Object Browser

vii. Using which option, one can import or export applications to/from Oracle 10g?
   a. Administration
   b. Utilities
   c. SQL
   d. Application Builder

References and Bibliography

BOOK REFERENCES

1. C. J. Date, A. Kannan and S. Swamynathan, *An Introduction to Database Systems*, Pearson Education, Eighth Edition, 2009.
2. Abraham Silberschatz, Henry F. Korth and S. Sudarshan, *Database System Concepts*, McGraw-Hill Education (Asia), Fifth Edition, 2006.
3. Shio Kumar Singh, *Database Systems Concepts, Designs and Application*, Pearson Education, Second Edition, 2011.
4. Peter Rob and Carlos Coronel, *Database Systems Design, Implementation and Management*, Thomson Learning-Course Technology, Seventh Edition, 2007.
5. Patrick O'Neil and Elizabeth O'Neil, *Database Principles, Programming and Performance*, Harcourt Asia Pte. Ltd., First Edition, 2001.
6. Atul Kahate, *Introduction to Database Management Systems*, Pearson Education Pte. Ltd., First Edition, 2004.
7. Raghu Ramakrishnan and Johannes Gehrke, *Database Management Systems*, Tata McGraw-Hill Education (Asia), Third Edition, 2003.
8. Dr Arun Kumar R., John Kanagaraj and Richard Stroupe, *Oracle Database 10g: Insider Solutions*, Pearson Education, First Edition, 2006.
9. Sam Anahory and Dennis Murray, *Data Warehousing in the Real World: A Practical Guide for Building Decision Support Systems*, Pearson Education Ltd., Eleventh Impression, 2012.
10. John Kauffman, Brian Matsik and Kevin Spencer, *Beginning SQL Programming*, Shroff Publishers and Distributors Pvt. Ltd., First Reprint, 2001.
11. Pang-Ning Tan, Michael Steinbach and Vipin Kumar, *Introduction to Data Mining*, Pearson Education Inc., Fourth Impression, 2009.
12. Arun Pujari, *Data Mining Techniques*, Universities Press (India) Pvt. Ltd., Second Edition, 2010.
13. Jia Wei Han and Micheline Kamber, *Data Mining Concepts and Techniques*, Morgan Kaufmann Publishers, Second Edition, 2006.
14. Vikram Pudi and P. RadhaKrishna, *Data Mining*, Oxford University Press, First Edition, 2009.
15. John L. Viescas and Jeff Conrad, *Microsoft Office Access 2007 Inside Out*, Prentice-Hall of India Private Limited, First Edition, 2007.
16. Virginia Andersen, *The Complete Reference Microsoft Office Access 2007*, Tata McGraw-Hill Publishing Company Limited, Second Edition, 2008.

17. Ivan Bayross, *SQL, PL/SQL The Programming Language of ORACLE*, BPB Publications, Third Edition, 2008.
18. E-Book: *Introduction to Database Systems*, By: ITL Education Solutions Limited, Publisher: Pearson Education India, 2008, e-book ISBN-10: 81-3174-319-5; e-book ISBN-13: 978-8-131-74319-5.
19. E-book: *Oracle Database Application Developer's Guide-Fundamentals, 10g Release 2* (10.2), Oracle, 2005, By: Primary Author: Lance Ashdown; Contributing Authors: D. Adams, M. Cowan, R. Moran, J. Melnick, E. Paapanen, J. Russell and R. Strohm.

RESEARCH PAPERS/ARTICLES REFERENCES

1. E. F. Codd, *Extending the Database Relational Model to Capture More Meaning*, ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, Pages 397–434.
2. I-Min A. Chen, Richard Hull and Dennis Mcleod, *An Execution Model for Limited Ambiguity Rules and Its Application to Derived-Data Update*, ACM Transactions on Database Systems, Vol. 20, No. 4, December 1995, Pages 365–413.
3. William C. Mcgee, *On User Criteria for Data Model Evaluation*, ACM Transactions on Database Systems, Vol. 1, No. 4, December 1976, Pages 370–387.
4. Feng Shao, Antal Novak and Jayavel Shanmugasundaram, *Triggers over Nested Views of Relational Data*, ACM Transactions on Database Systems, Vol. 31, No. 3, September 2006, Pages 921–967.
5. Stanley Y. W. Su, Herman Lam and Der Her Lo, *Transformation of Data Traversals and Operations in Application Programs to Account for Semantic Changes of Databases*, ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, Pages 255–294.
6. Gad Ariav, *A Temporally Oriented Data Model*, ACM Transactions on Database Systems, Vol. 11, No. 4, December 1986, Pages 499–527.
7. Millist W. Vincent, Jixue Liu and Chengfei Liu, *Strong Functional Dependencies and Their Application to Normal Forms in XML*, ACM Transactions on Database Systems, Vol. 29, No. 3, September 2004, Pages 445–462.
8. Martin Gogolla and Uwe Hohenstein, *Towards a Semantic View of an Extended Entity-Relationship Model*, ACM Transactions on Database Systems, Vol. 16, No. 3, September 1991, Pages 369–416.
9. D. S. Batory, T. Y. Leung and T. E. Wise, *Implementation Concepts for an Extensible Data Model and Data Language*, ACM Transactions on Database Systems, Vol. 13, No. 3, September 1988, Pages 231–262.
10. Antonio Albano, Luca Cardelli and Renzo Orsini, *Galileo: A Strongly-Typed, Interactive Conceptual Language*, ACM Transactions on Database Systems, Vol. 10, No. 2, June 1985, Pages 230–260.
11. Debabrata Dey, Veda C. Storey and Terence M. Barron, *Improving Database Design through the Analysis of Relationships*, ACM Transactions on Database Systems, Vol. 24, No. 4, December 1999, Pages 453–486.
12. Eugene Wong, *A Statistical Approach to Incomplete Information in Database Systems*, ACM Transactions on Database Systems, Vol. 7, No. 3, September 1982, Pages 470–488.

13. Carlos A. Hurtado, Claudio Gutierrez and Alberto O. Mendelzon, *Capturing Summarizability with Integrity Constraints in OLAP*, ACM Transactions on Database Systems, Vol. 30, No. 3, September 2005, Pages 854−886.
14. Yair Wand, Veda C. Storey and Ron Weber, *An Ontological Analysis of the Relationship Construct in Conceptual Modeling*, ACM Transactions on Database Systems, Vol. 24, No. 4, December 1999, Pages 494–528.
15. Dennis Shasha and Tsong-Li Wang, *Optimizing Equijoin Queries in Distributed Databases Where Relations are t-lash Partitioned*, ACM Transactions on Database Systems, Vol. 16, No. 2, June 1991, Pages 279–308.
16. Debabrata Dey and Sumit Sarkar, *A Probabilistic Relational Model and Algebra*, ACM Transactions on Database Systems, Vol. 21, No. 3, September 1996, Pages 339−369.
17. Gü Ltekin Ö Zsoyoğ Lu, Ismail Sengö R Altingö Vde, Abdullah Al-Hamdani, Selma Ays¸ E Ö Zel, Ö Zgü R Ulusoy and Zehra Meral Ö Zsoyoğ Lu, *Querying Web Metadata: Native Score Management and Text Support in Databases*, ACM Transactions on Database Systems, Vol. 29, No. 4, December 2004, Pages 581−634.
18. Laks V. S. Lakshmanan, Fereidoon Sadri and Subbu N. Subramanian, *SchemaSQL—An Extension to SQL for Multidatabase Interoperability*, ACM Transactions on Database Systems, Vol. 26, No. 4, December 2001, Pages 476–519.
19. Luca Forlizzi, Ralf Haxtmut Güting, Enrico Nardelli and Markus Schneider, *A Data Model and Data Structures for Moving Objects Databases*, ACM SIGMOD 2000 5/00 Dallas,
20. Michael Stonebraker, Eugene Wong, Peter Kreps and Gerald Held, *The Design and Implementation of INGRES*, ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, Pages 189-222.
21. Mcchael Stonebraket and Lawrence A Rowe, *The Design Of Postgres*, ACM 0-89791-191-1/86/0500/0340.
22. Tirthankar Lahiri, Amit Ganesh, Ron Weiss and Ashok Joshi , *Fast-Start: Quick Fault Recovery in Oracle*, ACM SIGMOD 2001 May 21–24.
23. Evaggelia Pitoura and Bharat Bhargava, *A Framework for Providing Consistent and Recoverable Agent-based Access to Heterogeneous Mobile Databases*, SIGMOD Record, Vol. 24, No. 3, September 1995.
24. Krishna Kunchithapadam, Wei Zhang, Amit Ganesh and Niloy Mukherjee, *Oracle Database File System*, SIGMOD'11, June 12–16, 2011, Athens, Greece.
25. Jos Moreira and Ribeiro, *Query Operations for Moving Objects Database Systems*, 8th ACM symposium on GIS 11/00 Washington, D. C., USA.
26. Hyun Jin Moon and Carlo Zaniolo, *Scalable Architecture and Query Optimization for Transaction-time DBs with Evolving Schemas*, SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.
27. Thomas Behr, Victor Teixeira de Almeida and Ralf Hartmut Güting, *Representation of Periodic Moving Objects in Databases*, ACM-GIS'06, November 10–11, 2006.
28. M. M. Astrahan, ht. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. A. Jones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade and V. Watson, *System R: Relational Approach to Database Management*, ACM Transactions on Database Systems, Vol. 1, No. 2. June 1976, Pages 97–137.

29. Jorge F Gana and Won Kim, *Transaction Management in an Object-Oriented Database System*, ACM 0-89791-268-3/88/0006/0037.
30. Ewing L. Lusk, Ross A. dverbeek, and Bruce Parrello, *A Practical Design Methodology for the Implementation of IMS Databases Using the Entity-Relationship Model*, ACM 0-89791418-4/80/0500/0009.
31. Bogdan Czejdo, Ramez Elmasri and Marek Rusinkiewicz, *An Algebric Language for Graphical Query Formulation Using an Extended Entity-Relationship Model*, ACM 0-89791-218-7/87/0002-0154.
32. Alexander Egyed, *Automated Abstraction of Class Diagrams*, ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 4, October 2002, Pages 449–491.
33. Sudha Ram, *Deriving Functional Dependencies from the Entity-Relationship Model*, Communications of the ACM, September 1995, Vol. 38, No. 9.
34. Carol Chrisman and Barbara Beccue, *Entity Relationship Models as a Tool for Data Analysis and Design*, ACM-O-89791-178-4/86/0002/0008.
35. Antonio Badia, *Entity-Relationship Modeling Revisited*, SIGMOD Record, Vol. 33, No. 1, March 2004.
36. Joseph Fong, *Mapping Extended Entity Relationship Model to Object Modeling Technique*, SIGMOD Record, Vol. 24, No. 3, September 1995.
37. Victor M. Markowitz and Arie Shoshani, *On the Correctness of Representing Extended Entity-Relationship Structures in the Relational Model*, ACM 0-89791-317-5/89/ooO5/0430.
38. Ramez Elmasri and Gio Wiederhold, *Properties of Relationships and Their Representation*, National Computer Conference, 1980.
39. James Rumbaugh, *Relations as Semantic Constructs in an Object-Oriented Language*, OOPSLA' 87 Proceedings October 4–8, 1987.
40. Victor M. Markowitz and Arie Shoshani, *Representing Extended Entity-Relationship Structures in Relational Databases-A Modular Approach*, ACM Transactions on Database Systems, Vol. 17, No. 3, September 1992, Pages 423−464.
41. Sikha Bagui, *Rules for Migrating from ER and EER Diagrams to Object-Relationship (OR) diagrams*, 43rd ACM Southeast Conference, March 18–20, 2005.
42. Peter Pin-Shan Chen, *The entity-relationship model—A basis for the enterprise view of data*, National Computer Conference, 1977.
43. Peter Pin-Shan Chen, *The Entity-Relationship Model-Toward a Unified View of Data*, ACM Transactions on Database Systems, Vol. 1, No. 1. March 1976, Pages 9–36.
44. Hafeez Osman, Dave R. Stikkolorum, Arjan van Zadelhoff, Michel R.V. Chaudron and Niels Bohrweg, *UML Class Diagram Simplification: What is in the developer's Mind*?, EESSMOD'12, October 1−5, 2012.
45. Fred J. Maryanski and Veda C. Storey, *Understanding Semantic Relationships*, VLDB Journal, 2, 455−488 (1993).
46. Kofi Apenyo, *Using the Entity-Relationship Model to Teach the Relational Model*, No. 2 June 1999, SIGCSE Bulletin.
47. Michael Schrader, William Endress and Fred Richards, *Understanding an OLAP Solution from Oracle*, An Oracle White Paper, April 2008.

48. John C. Peck, *Distributed Database/File Systems, Introduction*, ACM Computer Science Conference-Agenda for Computing Research: The Challenge for Creativity, 1985 March 12–14.
49. Toby J. Teorey, *Distributed Database Design: A Practical Approach and Example*, SIGMOD RECORD, Vol. 18, No. 4, December, 1989.
50. Carlos Ordonez, Javier García-García and Zhibo Chen, *Measuring Referential Integrity in Distributed Databases*, CIMS'07, November 9, 2007.
51. Ralf Hartmut Güting, *An Introduction to Spatial Database Systems*, Special Issue on Spatial Database Systems of the VLDB Journal, Vol. 3, No. 4, October 1994.
52. Z. Meral Ozsoyoglu and Li-Yan Yuan, *A New Normal Form for Nested Relations*, ACM Transactions on Database Systems, Vol. 12, No. 1, March 1987.
53. Ronald Fagin, *A Normal Form for Relational Databases That is Based on Domains and Keys*, ACM Transactions on Database Systems, Vol. 6, No. 3, September 1981.
54. Marcelo Arenas and Leonid Libkin, *A Normal Form for XML Documents*, ACM Transactions on Database Systems, Vol. 29, No. 1, March 2004.
55. P. A. Bernstein, J. R. Swenson and D. C. Tsichritzis, *A Unified Approach to Functional Dependencies and Relations*.
56. Tok-Wang Ling, Frank W. Tompa and Tiko Kameda, *An Improved Third Normal Form for Relational Databases*, ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981.
57. William Kerr, *A Simple Guide to Five Normal Forms in Relational Database Theory*, Communications of the ACM February 1983, Vol. 26, No. 2.
58. Don-Min Tsou and Patrick C. Fischer, *Decomposition of a Relation Scheme into Boyce-Codd Normal Form*, ACM 0-89791-028-1/80/1000/0411.
59. W. W. Armstrong and C. Delobel, *Decompositions and Functional Dependencies in Relations*', ACM Transactions on Database Systems, Vol. 5, No. 4, December 1980.
60. Sven Hartmann, Markus Kirchberg and Sebastian Link, *Design by Example for SQL Table Definitions with Functional Dependencies*, 24 June 2011, Springer–Verlag, 2011.
61. Victor Vianu, *Dynamic Functional Dependencies and Database Aging*, Journal of the Association for Computing Machinery, Vol. 34, No. 1, January 1987.
62. K. V. S. V. N. Raju and Arun K. Majumdar, *Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems*, ACM Transactions on Database Systems, Vol. 13, No. 2, June 1988.
63. Dennis J. McLeod, *High Level Domain Definition in a Relational Database System*, IBM Research Laboratory.
64. William Kent, *Limitations of Record-Based Information Models*, ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979.
65. Betty Salzberg, *Third Normal Form Made Easy*, SIGMOD RECORD, Vol. 15, No. 4, December 1986.
66. Ronald Fagin, *Multivalued Dependencies and a New Normal Form for Relational Databases*, ACM Transactions on Database Systems, Vol. 2, No. 3, September 1977.
67. Hugh Darwen, C. J. Date, Ronald Fagin, *A Normal Form for Preventing Redundant Tuples in Relational Databases*, ICDT 2012, March 26–30, 2012.

68. Margaret S. Wu, *The Practical Need for Fourth Normal Form*, ACM 0-89791-468-61921000210019.
69. Ronald Fagin, *Normal Forms and Relational Database Operators*, ACM 0-89791-001-x/79/0500-0153.
70. Zahir Tari, John Stokes and Stefano Spaccapietra, *Object Normal Forms and Dependency Constraints for Object-Oriented Schemata*, ACM Transactions on Database Systems, Vol. 22, No. 4, December 1997.
71. Catriel Beeri, *On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases*, ACM Transactions on Database Systems, Vol. 5, No. 3, September 1980.
72. C. J. Date and Ronald Fagin, *Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases*, ACM Transactions on Database Systems, Vol. 17, No. 3, September 1992.
73. Philip A. Bernstein, *Synthesizing Third Normal Form Relations from Functional Dependencies*, ACM Transactions on Database Systems, Vol. 1. No. 4, December 1976.
74. Gang Luo, Jeffrey F. Naughton, Curt J. Ellmann and Michael W. Watzke, *Locking Protocols for Materialized Aggregate Join Views*, Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.
75. Wen-Syan Li, K. Sel¸cuk Candan, Kyoji Hirata and Yoshinori Hara, *Supporting Efficient Multimedia Database Exploration*, The VLDB Journal (2001) 9: 312–326 / Digital Object Identifier (DOI), 10.1007/s007780100040.
76. Edmond Lau and Samuel Madden, *An Integrated Approach to Recovery and High Availability in an Updatable, Distributed Data Warehouse*, VLDB ‘06, September 1215, 2006.
77. Jeffrey Fischer and Rupak Majumdar, *Ensuring Consistency in Long Running Transactions*, ASE ’07, November 5–9, 2007, ACM, 978-1-59593-882-4/07/0011.
78. Alexander Thomasian, *Concurrency Control: Methods, Performance, and Analysis*, ACM Computing Surveys, Vol. 30, No. 1, March 1998.
79. Shahidul Islam Khan and Dr. A. S. M. Latiful Hoque, *A New Technique for Database Fragmentation in Distributed Systems*, International Journal of Computer Applications (0975–8887), Vol. 5, No. 9, August 2010.
80. Hiiko Schuldt, Gustavo Alonso, Catriel Beeri and Hans-Jö RG Schek, *Atomicity and Isolation for Transactional Processes*, ACM Transactions on Database Systems, Vol. 27, No. 1, March 2002.
81. Arpita Mathur, Mridul Mathur, Pallavi Upadhyay and Arpita Mathur *et al.*, *Cloud Based Distributed Databases: The Future Ahead*, International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397, Vol. 3, No. 6, June 2011.
82. Sharad Mehrotra, Henry F. Korth and Avi Silberschatz, *Concurrency Control in Hierarchical Multidatabase systems*, The VLDB Journal (1997).
83. Jean-Pierre Briot, Rachid Guerraoui and Laus-Peter Lo¨ Hr, *Concurrency and Distribution in Object-Oriented Programming*, ACM Computing Surveys, Vol. 30, No. 3, September 1998.

WEB REFERENCES

1. http://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm
2. http://docs.oracle.com/cd/E11882_01/server.112/e25789/transact.htm
3. http://lsirwww.epfl.ch/courses/iis/2009ss/slides/slides-11-Transactions.pdf
4. http://docs.oracle.com/cd/A57673_01/DOC/server/doc/SPS73/chap22.htm
5. http://docs.oracle.com/cd/B12037_01/network.101/b10777/protdata.htm
6. http://www.inf.unibz.it/dis/teaching/DDB/ln/ddb02.pdf
7. http://docs.oracle.com/cd/B12037_01/network.101/b10777/protdata.htm
8. http://www.oracle.com/us/solutions/business-intelligence/064300.pdf
9. http://link.springer.com/chapter/10.1007%2F978-1-4302-0528-9_12#page-1
10. http://www.peterindia.net/MultimediaDatabase.html
11. http://www.cs.cf.ac.uk/Dave/Multimedia/node141.html
12. http://docs.oracle.com/cd/B28359_01/appdev.111/b28415/ch_intr.htm
13. http://dna.fernuni-hagen.de/Lehre-offen/Kurse/1675/KE1.pdf
14. http://blog.safaribooksonline.com/2012/08/17/moving-to-nosql-databases/
15. http://nosql-database.org/
16. http://www.oracle.com/technetwork/database/nosqldb/learnmore/nosql-database-498041.pdf
17. http://www.10gen.com/nosql
18. http://www.ehow.com/info_10069998_spatial-databases.html
19. http://en.wikipedia.org/wiki/Data_mining
20. http://www.rdatamining.com/resources/tools
21. http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/sqloperations.htm#BABJIHCC

Index