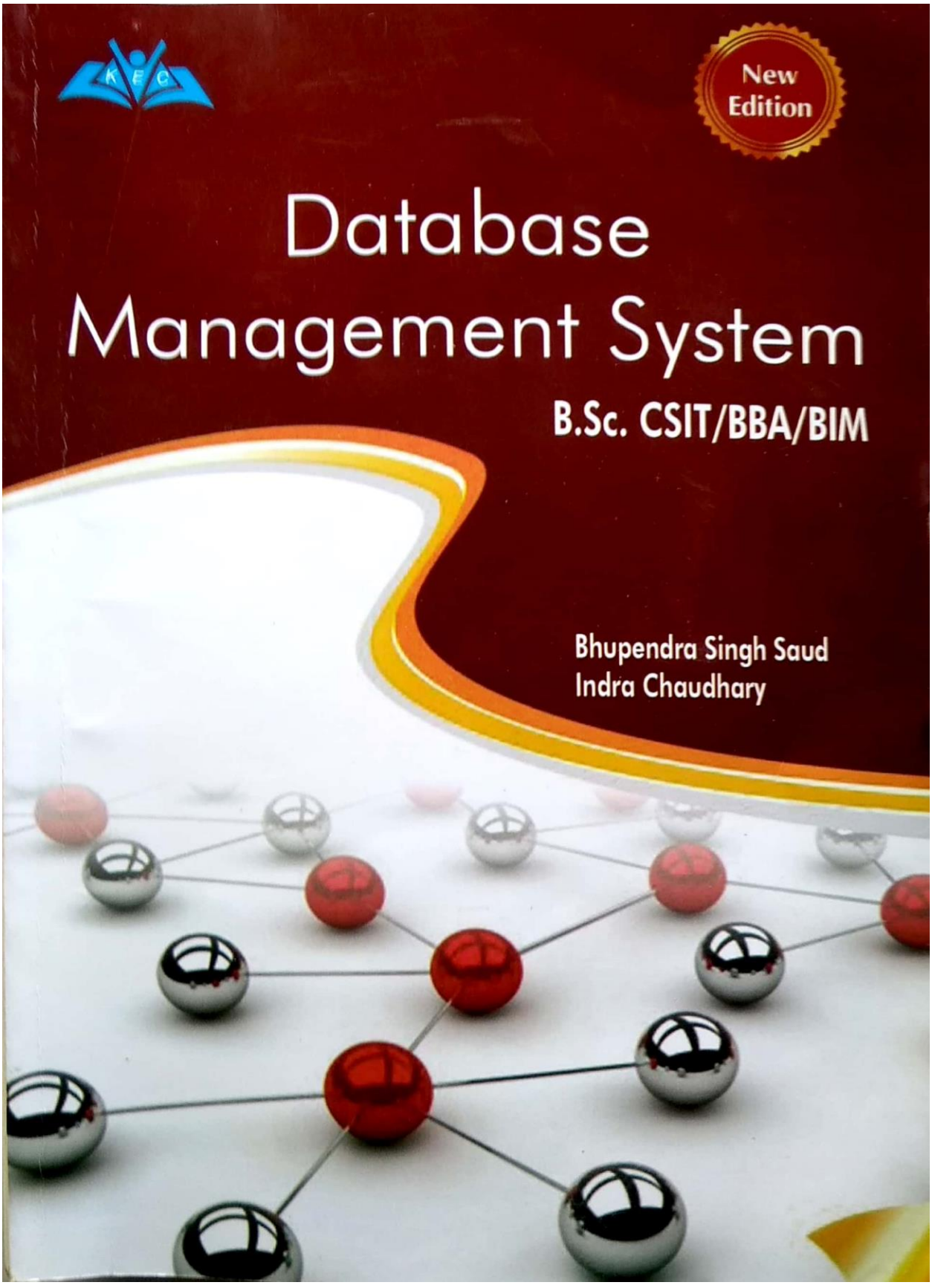

New Edition
Database Management System
B.Sc. CSIT/BBA/BIM
Bhupendra Singh Saud
Indra Chaudhary

DATABASE MANAGEMENT SYSTEM

B.Sc. CSIT/BBA/BIM

Bhupendra Singh Saud
IT Officer CIT
Faculty Member
Henrey Ford International College
New Summit College

Indra Chaudhary
Faculty Member
Academia International College
College of Applied Business
New Summit College
Sagarmatha College of Science and Technology

KEC Publication and Distribution (P.) Ltd.
Kathmandu, Nepal
Phone: 01-4168301, 01-4241777

DATABASE MANAGEMENT SYSTEM

Title

DATABASE MANAGEMENT SYSTEM
B.Sc. CSIT/BBA/BIM

ISBN No.
978-9937-724-39-5

Publisher
KEC PUBLICATION AND DISTRIBUTION PVT. LTD.
Kathmandu
01-4168301, 01-4241777

Distributor
KEC PUBLICATION AND DISTRIBUTION PVT. LTD.
Kathmandu
01-4168301, 01-4241777

Edition
First Edition 2076

Price
Rs. 475.00

PREFACE

The knowledge of database systems is an important part of an education in Computer Science. Without database, an organization is blind.

This book is designed as a primary text of Database Management System for the undergraduate level especially for B. Sc. CSIT (TU), BIM (TU) and BBA (TU). It is also convenient for BIM, BBIS, BIT, BE Computer, BCA, Bed Computer and other IT programs as well as postgraduate programs. The book "Database Management System" introduces concepts, design, applications and technologies of database in a very comprehensive manner. It assumes no previous knowledge of database and database technology. The chapters are written in a very simple language with a lot of examples and exercises. The book also contains a large number of figures to explain the concept in a better way.

This book fills the gap between theoretical learning and practical implementation of concepts of database for IT professionals and engineers. We hope that this book will be useful to the readers in all ways without any difficulty. We would welcome the suggestions for further improvement of the book given by our readers.

Thank you.

Authors
2076

Acknowledgments

First, we'd like to thank all of our colleagues who inspired us to write this book. We owe a huge debt to our family members who supported us at the preparation of this book providing all the authors space and time at home.

Special thank goes to the KEC Publication and Distribution for bringing this book to market into this shape.

Students and colleagues from different colleges where we are teaching this subject also provided us valuable remarks and suggestions.

Finally, we wish to get more feedbacks and suggestions from the students and colleagues from different colleges in the coming years and help to make this book better when it comes in newer versions.

Authors
Bhupendra Singh Saud
Indra Chaudhary

Syllabus
Database Management System

Course Title: Database Management System
Course No: CSC260
Nature of the Course: Theory + Lab
Semester: IV

Full Marks: 60 + 20 + 20
Pass Marks: 24 + 8 + 8
Credit Hrs: 3

Course Description:

The course covers the basic concepts of databases, database system concepts and architecture, data modeling using ER diagram, relational model, SQL, relational algebra and calculus, normalization, transaction processing, concurrency control, and database recovery.

Course Objective:

The main objective of this course is to introduce the basic concepts of database, data modeling techniques using entity relationship diagram, relational algebra and calculus, basic and advanced features SQL, normalization, transaction processing, concurrency control, and recovery techniques.

Course Contents:

Unit 1: Database and Database Users **(2 Hrs.)**

Introduction; Characteristics of the Database Approach; Actors on the Scene; Workers behind the Scene; Advantages of Using the DBMS Approach

Unit 2: Database System – Concepts and Architecture **(3 Hrs.)**

Data Models, Schemas, and Instances; Three-Schema Architecture and Data Independence; Database Languages and Interfaces; the Database System Environment; Centralized and Client/Server Architectures for DBMSs; Classification of Database Management Systems

Unit 3: Data Modeling Using the Entity-Relational Model **(6 Hrs.)**

Using High-Level Conceptual Data Models for Database Design; Entity Types, Entity Sets, Attributes, and Keys; Relationship Types, Relationship Sets, Roles, and Structural Constraints; Weak Entity Types; ER Diagrams, Naming Conventions, and Design Issues; Relationship Types of Degree Higher Than Two; Subclasses, Superclasses, and Inheritance; Specialization and Generalization; Constraints and Characteristics of Specialization and Generalization

Unit 4: The Relational Data Model and Relational Database Constraints **(3 Hrs.)**

Relational Model Concepts; Relational Model Constraints and Relational Database Schemas; Update Operations, Transactions, and Dealing with Constraint Violations

Unit 5: The Relational Algebra and Relational Calculus **(5 Hrs.)**

Unary Relational Operations: SELECT and PROJECT; Relational Algebra Operations from Set Theory; Binary Relational Operations: JOIN and DIVISION; Additional Relational Operations; the Tuple Relational Calculus; the Domain Relational Calculus

Unit 6: SQL **(8 Hrs.)**

Data Definition and Data Types; Specifying Constraints; Basic Retrieval Queries; Complex Retrieval Queries; INSERT, DELETE, and UPDATE Statements; Views

Unit 7: Relational Database Design **(7 Hrs.)**

Relational Database Design Using ER-to-Relational Mapping; Informal Design Guidelines for Relational Schemas; Functional Dependencies; Normal Forms Based on Primary Keys; General Definitions of Second and Third Normal Forms; Boyce-Codd Normal Form; Multivalued Dependency and Fourth Normal Form; Properties of Relational Decomposition

Unit 8: Introduction to Transaction Processing Concepts and Theory (4 Hrs.)

Introduction to Transaction Processing; Transaction and System Concepts; Desirable Properties of Transactions; Characterizing Schedules Based on Recoverability; Characterizing Schedules Based on Serializability

Unit 9: Concurrency Control Techniques (4 Hrs.)

Two-Phase Locking Technique; Timestamp Ordering; Multiversion Concurrency Control; Validation (Optimistic) Techniques and Snapshot Isolation Concurrency Control

Unit 10: Database Recovery Techniques (3 Hrs.)

Recovery Concepts; NO-UNDO/REDO Recovery Based on Deferred Update; Recovery Technique Based on Immediate Update; Shadow Paging; Database Backup and Recovery from Catastrophic Failures

Laboratory Works:

The laboratory work includes writing database programs to create and query databases using basic and advanced features of structured query language (SQL).

Text Books:

1. Fundamentals of Database Systems; Seventh Edition; Ramez Elmasri, Shamkant B. Navathe; Pearson Education
2. Database System Concepts; Sixth Edition; Avi Silberschatz, Henry F Korth, S Sudarshan; McGraw-Hill

Reference Books:

1. Database Management Systems; Third Edition; Raghu Ramakrishnan, Johannes Gehrke; McGraw-Hill
2. A First Course in Database Systems; Jaffrey D. Ullman, Jennifer Widom; Third Edition; Pearson Education Limited

BBA (Introductory Database)

Course Title: Introductory Database
Course No: IT 202
Semester: II
Module Objectives
This module aims to provide the students with the basic knowledge, issues and manipulation of database so that the students develop the skill of producing reports and managing business information.
Contents
Introduction to DBMS, Types of DBMS, Data Models, Relational Database Model, Entity Relationship models, Structured Query Language, Distributed Database Management Systems, Database Security, and Designing Good Designed Database

Chapter 1: Introduction to DBMS [5hrs]

- Introduction to Database Management System
- DBMS vs File System
- View of data
- Data models
- Database Languages: DML, DDL
- Database users and administrators
- Transaction Management
- Database System Structure
- Application architectures

Chapter 2: Types of DBMS [2hrs]

- Types of DBMS
- Relational DBMS

- Object-oriented Database Management System

Chapter 3: Data Models [5hrs]

- Basic concepts
- Constraints
- Keys
- Design issues
- The Entity Relationship Database Model
- Weak entity sets
- Extended E-R Features
- Design of an E-R database schema
- Reduction of an R-R schema to tables

Chapter 4: Relational Database Model [5hrs]

- Structure of relational databases (Basic Structures, Database schema, keys, Schema Diagram, query languages, joins (Natural, outer))
- Data Integrity
- Types of Data Integrity (Entity integrity constraint (Primary key), Referential Integrity, Domain Constraints, triggers, assertions)
- Relationships within the Relational Database

Chapter 5: Entity Relationship Modeling [6hrs]

- Entity Relationship modeling
- Entities, Attributes, Relationships
- Degree of a relationship
- Connectivity and Cardinality
- Specialization and Generalization Entity relationship diagram

Chapter 6: Structured Query Language [7hrs]

- Structured Query Language
- Data Definition Language
- SQL Constraints
- Creating Tables
- Data Manipulation Languages (Insert, Update, Delete, Select)
- SELECT Queries
- Syntax for the SELECT statement
- SQL Views
- JOIN

Chaptere 7: Distributed Database Management Systems [5hrs]

- Distributed Database Management System
- Characteristics of DDBMS
- Advantages / Disadvantages of DDBMS
- Components of Distributed Database system
- Distributed Database Design
- Fragmentation (horizontal and vertical fragmentation)

Chapter 8: Database Security [5hrs]

- **Authorization is SQL (privileges in SQL)**
- **Roles**
- **The privilege to Grant Privileges**
- **Limitations of SQL authorization**

Chapter 9: Designing Good Designed Database [5hrs]

- Normalization
- Funcational Dependencies: partial dependency, transitive dependency, join dependency, multivalued dependency
- Update Anamolies
- Normal forms: First, Second, Third, Fourth, Fifth

Text Book

1. Abraham Silberschatz, Henry Korth, S. Sudarshan, Database System Concepts,
2. C.J. Date, An Introduction to Database Systems

Reference Books

1. Philip J. Pratt and Joseph J. Adamski , Concepts of Database Management , Cengage Learning, 7 edition (June 14, 2011)
2. David Kroenke , David Auer , ,Database Concepts (6th Edition), Prentice Hall; 6 edition (July 20, 2012)

BIM (Database Management System)

Course Title: Database Management System
Nature of the Course: Theory + Lab
Semester: IV

Credit Hrs: 3
Lecture Hours: 48

Course Description:

The main objective of this module is to provide strong theoretical and practical knowledge of the database management system.

Course Objective:

Database system, Data Abstraction, Data Models, Database users, Entity-Relation Model, Constraints, E- R Diagrams, Design of E-R Database Schema, Relational Data Model, Structure of Relational Database, Relational Algebra, Fundamental Operations, Additional Operating, Modifying the database, Structured Query Language Data Definition Language, Data manipulation Language, Transaction Control Language, Join Operations, Integrity Constraints, Assertion, Triggers, Relational database design issues, Normalization, Transaction Management, Database System Architectures.

Course Contents:

Unit 1: Introduction – Database Management Systems **LH 4**

Purpose of Database Systems. Data Abstraction. Data Models: The E-R Model, The Object-Oriented Model, The Relational Model, The Network Model, The Hierarchical Model, Physical Data Models. Instances and Schemes. Data Independence. Database Administrator. Database Users. Application Architecture (One tier, two tier and n-tire). Overall Database System Structure and Components.

Unit 2: Entity-Relationship Model **LH 8**

Entities and Entity Sets. Relationships and Relationship Sets. Attributes. Mapping Constraints. Keys (Super key, Candidate key and Primary key): Primary Keys for Entity Sets and Relationship Sets. The Entity Relationship Diagram. Reducing E-R Diagrams to Tables: Representation of Strong Entity Sets, Representation of Weak Entity Sets, Representation of Relationship Sets. Generalization and Specialization. Aggregation. Mapping Cardinalities: Representation of Mapping Cardinalities in E-R Diagram. Use of Entity or Relationship Sets. Use of Extended E-R Features. Design of an E-R Database Scheme (Case study).

Unit 3: Structured Query Language (SQL) **LH 15**

Background, Data Definition Language: Domain Types in SQL, Schema Definition in SQL. Data Manipulation Language: The select Clause, The where Clause, The from Clause, The Rename Operation, Tuple Variables, String Operations, Ordering the Display of Tuples, Duplicate Tuples. Set Operations. Aggregate Functions. Null Values. Nested Subqueries: Set Membership, Set Comparison, Test for Empty Relations, Test for the Absence of Duplicate Tuples. Derived Relations: Views. Modification of the Database: Deletion, Insertion, Updates, Updates, Update of a view. Joined Relations: Join types and Conditions. Embedded SQL. Dynamic SQL. Transaction Control Language (Commit, Rollback).

Unit 4: Integrity Constraints LH 5

Domain Constraints. Referential Integrity: Basic Concepts, Referential Integrity in the E-R Model, Database Modification, Referential Integrity in SQL

Unit 5: Relational Database Design LH 6

Pitfalls in Relational DB Design. Representation of Information: Anomalies. Functional Dependencies: Basic Concepts, Closure of a Set of Functional Dependencies, Closure of Attribute Sets. Decomposition: Lossless-Join Decomposition, Dependency Preservation. Normalization: First Normal Form, Second Normal Form, Third Normal Form, Boyce-Codd Normal Form, Comparison of BCNF and 3NF.

Unit 6: Transaction Management LH 5

ACID Properties. Transaction States: Implementation of Atomicity and Durability, Serializability, Basic Concept of Concurrency Control and Recovery, Locking Protocols, Time Stamp Based Protocol.

Unit 7: Case Study LH 5

MSSQL server, ORACLE, MYSQL

Note: The students are required to undertake a project work. The project work can be done individually or in group (at most 4-5 students). The format of the project report is as follows:

- Project Description
- Description of entities or object considered in the project
- Algorithm or Diagram showing description of project
- Conclusion of the project

The project report should be original, and the reproduction of others' work is strictly prohibited. Number of pages of the report should be at least 4.

References:

1. Abraham Silberchatz, Henry F. Korth, S. Sudarshan; Database System Concepts, McGraw Hill 4th ed. Date, C.J.; An Introduction to Database System, Addison Wesley, 8th ed.
2. RAMEZ ELMASRI, B. NAVATHE, Fundamentals of Database System, Pearson Education Asia, Fifth Edition.

Contents

1 Chapter DATABASE AND DATABASE USERS

2 Chapter DATABASE SYSTEM–CONCEPTS AND ARCHITECTURE

3 Chapter DATA MODELING USING THE ENTITY-RELATIONAL MODEL

4 Chapter THE RELATIONAL DATA MODEL AND RELATIONAL DATABASE

5 Chapter THE RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

6 Chapter STRUCTURED QUERY LANGUAGE (SQL)

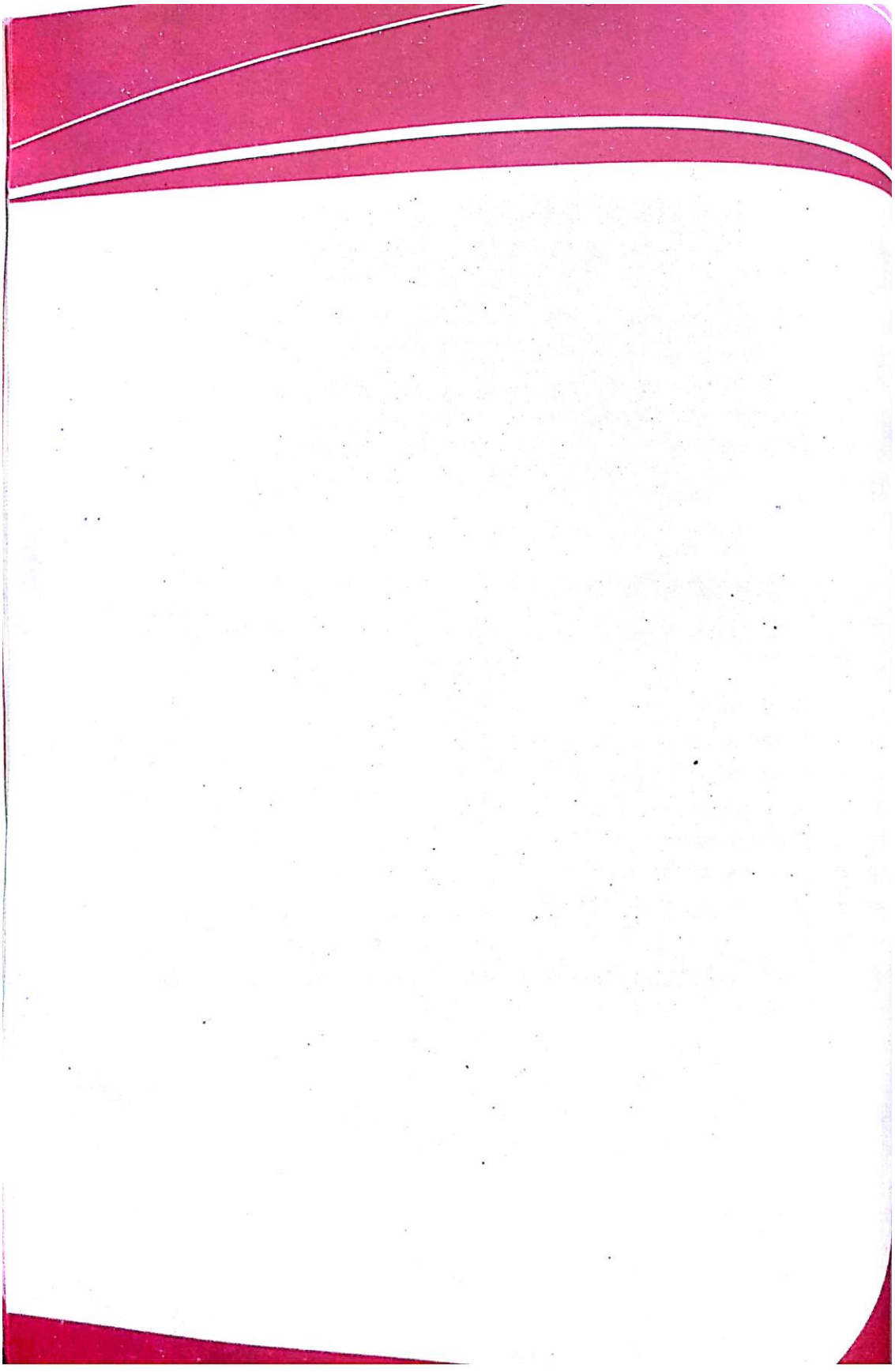
7 Chapter RELATIONAL DATABASE DESIGN

8 Chapter INTRODUCTION TO TRANSACTION PROCESSING CONCEPTS

9 Chapter CONCURRENCY CONTROL TECHNIQUES

10 Chapter DATABASE RECOVERY TECHNIQUES

11 Chapter DISTRIBUTED DATABASE MANAGEMENT SYSTEM



CHAPTER 1

DATABASE AND DATABASE USERS

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Introduction
- Characteristics of the Database Approach
- Actors on the Scene
- Workers behind the Scene
- Advantages of Using the DBMS Approach

1.1 Data vs. Information

Data is raw facts collected from environment about physical phenomena or business transactions. Data can be in any form-numerical, textual, graphical, image, sound, video etc. For example, data would be the marks obtained by students in different subjects. On the other hand information is defined as refined or processed data that has been transformed into meaningful and useful form for specific users. For example, after processing the marks obtained by student it transformed into information, which is meaningful and from which we can decide which student stood first, second and so forth. Information comes from data and takes the form of table, graphs, diagrams etc.

Here are four additional examples of how data differs from information:

1. While data does not depend on information, information does depend on data.
2. Data is **input** and information is **output.**
3. Data is raw material and information is the product.
4. Data is a single unit and information is a grouping of data.

1.2 Database

A database is an organized collection of logically related data that contains information relevant to an enterprise. The database is also called the repository or container for a collection of data files. For example, university database maintains information about students, courses and grades in university. Let's discuss few examples.

- An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc.
- An electricity service provider is obviously using a database to manage billing, client related issues, to handle fault data, etc.
- Let's also consider the Facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more.

1.3 Database components

A database is made up of several components some of main components are listed below:

- **Schema:** A database contains one or more schemas, which is basically a collection of one or more tables of data.
- **Table:** Each table contains multiple columns, which are similar to columns in a spreadsheet. A table can have as little as two columns and as many as one hundred or more columns, depending on the type of data being stored in the table.
- **Column:** Each column contains one of several types of data or values, like dates, numeric or integer values, and alphanumeric values (also known as VARCHAR).
- **Row:** Data in a table is listed in rows, which are like rows of data in a spreadsheet. Often there are hundreds or thousands of rows of data in a table.

1.4 Database Management System (DBMS)

A Database Management System (DBMS) is the set of programs that is used to store, retrieve and manipulate the data in convenient and efficient way. Main goal of database management system (DBMS) is to hide underlying complexities of data management from users and provide easy interface to them. Some common examples of the DBMS software are listed below:

- IBM DB2
- Microsoft Access
- Microsoft Excel
- Microsoft SQL Server
- MySQL
- Oracle RDBMS
- SAP Sybase ASE
- Teradata etc.

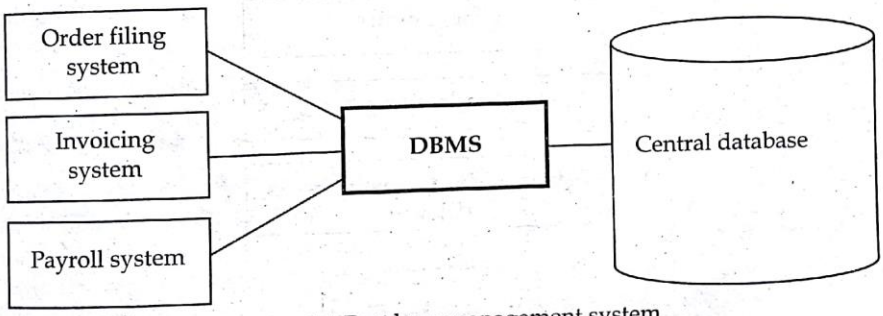


Figure 1.1 Database management system

1.5 Relational Database Management system (RDBMS)

Database management system that maintains relationship between multiple data files is called Relational Database Management system (RDBMS). RDBMS is the most common DBMSs in the market. It is widely used by the enterprises for storing complex and large amount of data. Some RDBMS supports Object Relational Model (ORM), allow developers to map their object models into relational model. MySQL, Oracle, SQL servers are well known RDBMS software.

Example

Student

S-ID	Name	Address	Course-ID
S-12	Pawan	Joshi	C002
S-14	Yamman	Karki	C021
S-51	Abin	Saud	C321
S-11	Aarav	Saud	C002

Foreign Keys

Relationships

Course

Course-ID	Course-Name
C002	C++
C021	DBMS
C321	Account

1.6 Database System

A database system consists of database, database Management system, and application programs. Simply, we can say that application software that uses DBMS for data management is called database system. For example, in a database management system, the Human Resource (HR) system, accounting management system, Project management system and Budget management programs would have a common database. This database based approach to data processing is shown in fig below:

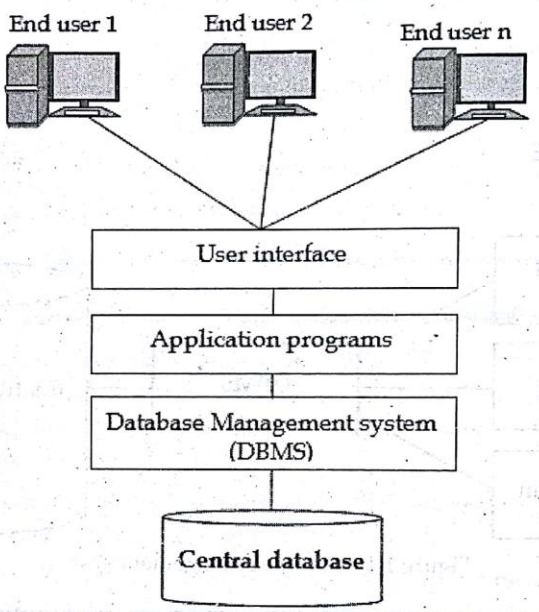


Figure 1.2 Database system approach to data processing

1.7 Difference between DBMS vs. RDBMS

DBMS	RDBMS
DBMS stores data as a file.	Data is stored in the form of tables.
DBMS system, stores data in either a navigational or hierarchical form.	RDBMS uses a tabular structure where the headers are the column names, and the rows contain corresponding values
DBMS supports single user only.	It supports multiple users.
In a regular database, the data may not be stored following the ACID model. This can develop inconsistencies in the database.	Relational databases are harder to construct, but they are consistent and well structured. They obey ACID (Atomicity, Consistency, Isolation, and Durability).
It is the program for managing the databases on the computer networks and the system hard disks.	It is the database systems which are used for maintaining the relationships among the tables.
Low software and hardware needs.	Higher hardware and software need.

DBMS does not support the integrity constants. The integrity constants are not imposed at the file level.	RDBMS supports the integrity constraints at the schema level. Values beyond a defined range cannot be stored into the particular RDMS column.
DBMS does not support Normalization	RDBMS can be Normalized.
DBMS does not support distributed database.	RBMS offers support for distributed databases.
DBMS system mainly deals with small quantity of data.	RDMS is designed to handle a large amount of data.
DBMS does not support client-server architecture	RDBMS supports client-server architecture.
Data fetching is slower for the complex and large amount of data.	Data fetching is rapid because of its relational approach.
Data redundancy is common in this model.	Keys and indexes do not allow Data redundancy.
No relationship between data	Data is stored in the form of tables which are related to each other with the help of foreign keys.
There is no security	Multiple levels of security. Log files are created at OS, Command, and object level.
Data elements need to access individually.	Data can be easily accessed using SQL query. Multiple data elements can be accessed at the same time.
Examples of DBMS are a file system, XML, Windows Registry, etc.	**Example** of RDBMS is MySQL, Oracle, SQL Server, etc.

1.8 File Management Systems (flat file system)

File management system (FMS) is also called flat file systems. It stores data in a plain text file. A flat file is a file that contains records, and in which each record is specified in a single line. Fields from each record may simply have a fixed width with padding, or may be delimited by whitespace, tabs, commas or other characters. There are no structural relationships and the data are "flat" as in a sheet of paper. In this approach each application has data files related to it containing all the data records needed by the application. Thus, an organization has to develop number of application programs each with an associated application-specific data files. For example, in a computer system, the checking account processing system would have its own data files. This file based approach to data processing is shown in fig below:

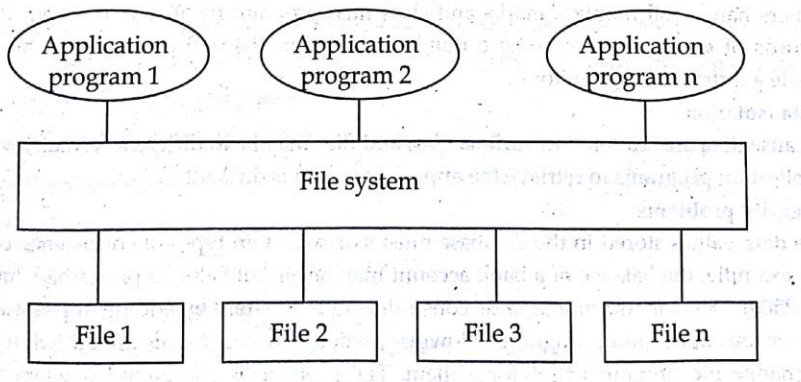


Figure 1.3 File processing system

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called file processing systems. In a typical file processing systems, each department has its own files, designed specifically for those applications. The department itself works with the data processing staff, sets policies or standards for the format and maintenance of its files. Programs are dependent on the files and vice-versa; that is, when the physical format of the file is changed, the program has also to be changed. Although the traditional file oriented approach to information processing is still widely used.

1.9 Limitations of Flat-File Systems

Keeping organizational information in a file-processing system has a number of major disadvantages. Theses drawbacks of flat file systems are solved by database management systems. Drawbacks of file processing systems or we can say advantages of database management systems are described below:

- **Data redundancy and inconsistency**
  Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.
- **Difficulty in accessing data**
  Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Accessing anomalies means that it is not easy to access data in a desired or efficient way. It makes supervision of department very difficult. If a user wants information in a specific manner then he requires creating a program for it. For **Example:** Let's say , if admin of the college wants any student information like his name, fathers name, roll number, marks and class then program for it is written but if he wants records of whose students whose numbers are more than 80 percent then he require to create a different program for it.
- **Data isolation**
  Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- **Integrity problems**
  The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, Rs. 2500). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

- **Atomicity problems**

  A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer 5000 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the 5000 was removed from account A but was not credited to account B, resulting in an inconsistent database state. It is difficult to ensure atomicity in a conventional file-processing system.

- **Concurrent-access anomalies**

  Concurrent access to data means more than one user is accessing the same data at the same time. Anomalies occur when changes made by one user gets lost because of changes made by other user. File system does not provide any procedure to stop anomalies. DBMS provides a locking system to stop anomalies to occur.

- **Security problems**

  Poor data security is the most threatening problem in File Processing System. There is very less security in File Processing System as anyone can easily modify and change the data stored in the files. All the users must have some restriction of accessing data up to a level. **For Example:** If a student can access his data in the college library then he can easily change books issued date. Also he can change his fine detains to zero.

- **Wastage of Labor and Space**

  Labor is very costly in this era and no organization can afford wastage of their precious labor. File Processing System needs lots of copied data in different files that cause wastage of labor. Also maintaining same data again and again leads to wastage of space too. **For Example:** Maintaining student's record in many departments that are not dependent on each other cause wastage of labor and space.

1.10 Advantages of Flat-file Systems

Despite of the number of drawbacks of flat-file systems they are beneficial in many situations and database management systems may not be much suitable in such situations. Benefits of file processing systems are described below:

- Flat files are relatively quick and easy to set up and use.
- Easy to understand.
- Easy to implement.
- Less hardware and software requirements.
- Less Initial Investment
- Less skill set is required to hand flat database systems.
- Best for small databases.
- Less overhead

1.11 Purpose of DBMS (Functions of DBMS)

Modern database management systems have a number of advantages over more traditional approaches to database maintenance. Here are some of the benefits of using a database management system to handle the storage and retrieval of information in database are listed below:

- **To reduce redundancy:** Repeating of the same information in database is called redundancy of data which leads to several problems such as wastage of space, duplication effort for entering data and inconsistency. When DBMS is used and database is created, redundancy is minimized.
- **To avoid inconsistency:** The database is said to be inconsistent if various copies of the same data may no longer agree. For example, a changed customer address may be reflected in saving account but not elsewhere in the system. By using DBMS we can avoid inconsistency.
- **To share data:** The data in the database can be shared among many users and applications. The data requirements of new applications may be satisfied without having to create any new stored files.
- **To provide support for transactions:** A transaction is a sequence of database operations that represents a logical unit of work. It accesses a database and transforms it from one state to another. A transaction can update a record, delete one, modify a set of records etc. when the DBMS does a 'commit'; the changes made by the transaction are made permanent. We can roll back the transaction to undo the effects of transaction.
- **To maintain integrity:** Most database applications have certain integrity constraints that must hold for the data. A DBMS provides capabilities for defining and enforcing these constraints. For example, the value of roll number field of each student in student database should be unique for each student. It is a type of rule. Such a rule is enforced using constraint at the time of creation of database.
- **To enforce security:** Not every user of the database system should be able to access all data. Different checks can be established for each type of access (retrieve, modify, delete, etc) to each piece of information in the database.
- **To provide efficient backup and recovery:** Provide facilities for recovering from software and hardware failures to restore database to previous consistent state..
- **To Concurrent Access Database:** Concurrent access means access to the same data simultaneously by more than one user. The same data may be used by many users for the purpose reading at the same time. But when a user tries to modify a data, there should be a concurrency control mechanism to avoid the inconsistency of data. A DBMS provides facilities for these operations.

1.12 Disadvantages of DBMS

- **Problem associated with centralization**

  Centralization increases the security problems.

- **Increased costs**

  One of the disadvantages of DBMS is database systems require sophisticated hardware, software, and highly skilled personnel. The cost of maintaining these requisites and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often unheeded when database systems are employed.

- **Management intricacy**

  Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives. Given the fact that database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.

- **Maintaining currency**

  To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates; apply the latest patches, and security measures to all components.

- **Frequent upgrade/replacement cycles**

  DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features.

- **Complexity of backup and recovery**

  DBMS provides the centralization of the data, which requires the adequate backups of data.

1.13 Difference between file processing system and database system

The difference between file processing system and database approach is as follow:

File based system	Database system
Here the data and program are inter-dependent.	Here the data and program are independent of each other.
File-based system caused data redundancy. The data may be duplicated in different files	Database system control data redundancy. The data appeared only once in the system.
File based system caused data inconsistency. The data in different files may be different that cause data inconsistency.	In database system data always consistent. Because data appeared only once.
The data cannot be shared because data is distributed in different files.	In database data is easily shared because data is stored at one place.
In file based system data is widely spread. Due to this reason file based system provides poor security.	It provides many methods to maintain data security in the database.

File based system does not provide consistency constrains.	Database system provides a different consistency constrains to maintain data integrity in the system.
File based system is less complex system.	Database system is very complex system.
The cost of file processing system is less then database system.	The cost of database system is much more than a file processing system.
File based system takes much space in the system and memory is wasted in this approach.	Database approach store data more efficiently it takes less space in the system and memory is not wasted.
To generate different report to take a crucial decision is very difficult in file based system.	The report can be generated very easily in required format in database system. Because data in database is stored in an organized manner. And easily retrieve to generate different report.
File based system does not provide concurrency facility.	Database system provides concurrency facility.
File based system does not provide data atomicity functionality.	Database system provides data atomicity functionality.
The cost of file processing system is less than database system.	The cost of database system is more than file processing system.
It is difficult to maintain as it provides less controlling facility.	Database provides many facilities to maintain program.
If one application fail it does not affects other files in system.	If database fail it affects all application that dependent on database
Hardware cost is less than database system	Hardware cost is high in database than file system.

1.14 Applications areas of database system

Database systems are widely used in different areas because of their numerous advantages. Some of the most common database applications are listed here.

- **Telecom:** There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry:** Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.

- **Banking System:** For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- **Sales:** To store customer information, production information and invoice details.
- **Airlines:** To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database.
- **Education sector:** Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online shopping:** We must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.
- **Human Resources:** Organizations use databases for storing information about their employees, salaries, benefits, taxes, and for generating salary checks.
- **E-commerce:** Integration of heterogeneous information sources (for example, catalogs) for business activity such as online shopping, booking of holiday package, consulting a doctor, etc.
- **Digital Libraries and Digital Publishing:** Databases are used for management and delivery of large bodies of textual and multimedia data.

1.15 Database Users and Administrators

The people who are accessing or working with the database are called database users and administrators. We know that the primary aim of the DBMS is to store the data or information and retrieve whenever it is needed by the database users. Those who are working with the database can be categorized into two; the database users and the database administrators.

Database Users

The database users also can be categorized again into five groups according to how they interact with the database. They are:

- Native Users
- Application Programmers
- Sophisticated Users
- Specialized Users
- Stand-alone Users

1. **Native Users**

   These are the database users who are communicating with the database through an already written program. **For example:** when a student is registering on a website for an online examination. He creates data in the database by entering and submitting his name, address and exam details.

2. **Application Programmers**

   These are the software developers and programming professionals who write the program codes. They use tools like Rapid Application Development (RAD) tools for creating user interfaces with minimal efforts.

3. **Sophisticated Users**

   Sophisticated users are those who are creating the database. These type of users do not write program code. And they do not use any software to request the database.The sophisticated users directly interact with the database system using query languages like SQL.

4. **Specialized Users**

   The sophisticated users who write special database application programs are called specialized users. The write complex programs for the specific complex requirements. Specialized users write applications such as computer-aided design systems, knowledge-base and expert systems that store data having complex data types.

5. **Stand-alone Users**

   Those who are using database for personal usage is called stand-alone users. There are many database packages for this type database user.

1.16 Database Administrators

A database administrator (DBA) is a specialized computer systems administrator who maintains a successful database environment by directing or performing all related activities to keep the data secure. The top responsibility of a DBA professional is to maintain data integrity. This means the DBA will ensure that data is secure from unauthorized access but is available to users.

A database administrator will often have a working knowledge and experience with a wide range of database management products such as Oracle-based software, SAP and SQL, in addition to having obtained a degree in Computer Science and practical field experience and additional, related IT certifications. There are a lot of responsibilities of DBA some of major functions of DBA in a database system are listed below:

- **Installing and Configuration of database**

  DBA is responsible for installing the database software. He/She configures the software of database and then upgrades it if needed. There are many database software like oracle, Microsoft SQL and MySQL in the industry so DBA decides how the installing and configuring of these database software will take place.

- **Deciding the hardware device**

  Depending upon the cost, performance and efficiency of the hardware, it is DBA who have the duty of deciding which hardware devise will suit the company requirement. It is hardware that is an interface between end users and database so it needed to be of best quality.

- **Managing Data Integrity**

  Data integrity should be managed accurately because it protects the data from unauthorized use. DBA manages relationship between the data to maintain data consistency.

- **Decides Data Recovery and Back up method**

  If any company is having a big database, then it is likely to happen that database may fail at any instance. It is require that a DBA takes backup of entire database in regular time span. DBA has to decide that how much data should be backed up and how frequently the back should be taken. Also the recovery of data base is done by DBA if they have lost the database.

- **Tuning Database Performance**

  Database performance plays an important role for any business. If user is not able to fetch data speedily then it may loss company business. So by tuning a modifying SQL commands a DBA can improves the performance of database.

- **Capacity Issues**

  All the databases have their limits of storing data in it and the physical memory also has some limitations. DBA has to decide the limit and capacity of database and all the issues related to it.

- **Database design / Physical organization modification**

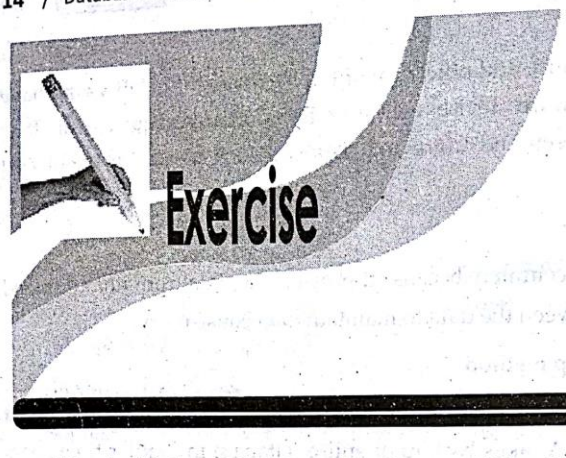
  The logical design of the database is designed by the DBA. Also a DBA is responsible for physical design, external model design, and integrity control. The DBA modifies the physical organization of the database to reflex the changing needs of the organization or to improve performance.

- **Database accessibility**

  DBA writes subschema to decide the accessibility of database. He decides the users of the database and also which data is to be used by which user. No user has to power to access the entire database without the permission of DBA.

- **Decides validation checks on data**

  DBA has to decide which data should be used and what kind of data is accurate for the company. So he always puts validation checks on data to make it more accurate and consistence.



1. What is database? How it is differ from traditional file system?
2. What are the advantages of a database system? What are the various cost and risk factors involved in implementing a database system?
3. What do you understand by the term data abstraction?
4. Write down advantages and disadvantages of flat file system.
5. What are the application areas of flat file system? Explain.
6. Discuss the various areas in which database system is used.
7. Who is DBA? List the various responsibilities of DBA.
8. What are the different types of database users who interact with the database system?
9. What are the responsibilities of DBA? Explain.
10. Describe different types of database users.
11. Explain the three-level architecture of DBMS with the help of an example. Mention its advantages also.
12. What is the difference between logical and physical data independence?
13. Describe purpose of database system with suitable example.
14. What is the difference between object-based data models and record-based data models?
15. What are the roles of system analyst and application programmer in database system?
16. Write a short note on data definition language and data manipulation language.
17. Explain the various functional components of a DBMS with the help of a suitable diagram.
18. Differentiate between DBMS and RDBMS.
19. What are the application areas of database system? Explain.
20. Explain the different criteria on the basis of which DBMS is classified into different categories.

❑❑❑

CHAPTER 2

DATABASE SYSTEM–CONCEPTS AND ARCHITECTURE

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Data Models
- Schemas, and Instances
- Three-Schema Architecture and Data Independence: Database Languages and Interfaces;
- The Database System Environment
- Centralized and Client/Server Architectures for DBMSs
- Classification of Database Management Systems

2.1 Data Models

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc. Traditionally, there are different database models which are used to design and develop the database of the organization.

1. Hierarchical Model
2. Network Model
3. Entity-Relationship Model
4. Relational Model
5. Object Oriented Model
6. Object-Relational Model

Hierarchical Model

This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes. In this model, a child node will only have a single parent node. This model efficiently describes many real-world relationships like index of a book, recipes etc. In hierarchical model, data is organized into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students. Thus hierarchical data model is able to represent one-to-one, one-to-many and many-to-one relationships.

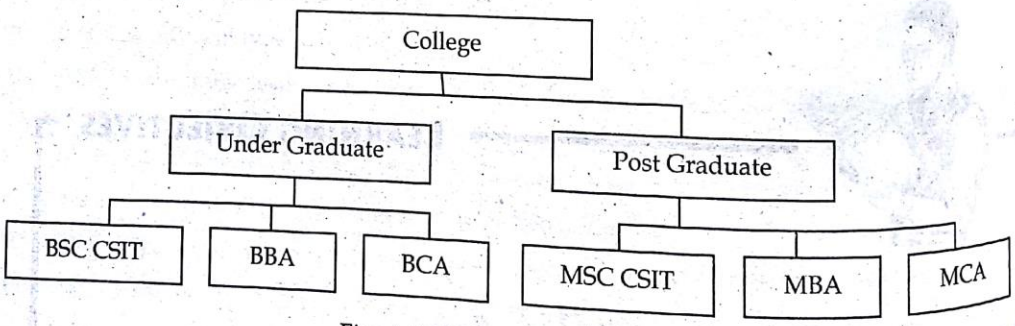


Figure 2.1 Hierarchical data model

Network Model

This is an extension of the Hierarchical model. In this model data is organized more like a graph, and are allowed to have more than one parent node. In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships. This was the most widely used database model, before Relational Model was introduced. Thus network data model is able to represent one-to-one, one-to-many and many-to-many relationships also.

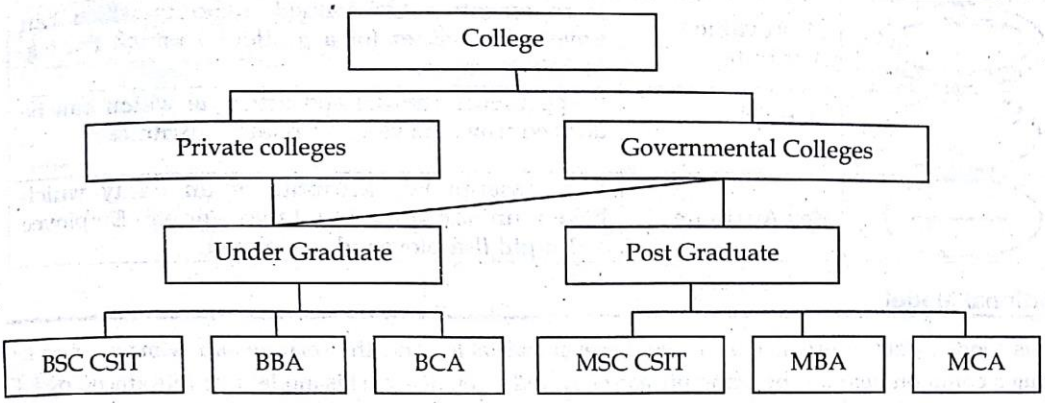


Figure 2.2 Network data model

Entity-Relationship Model

In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes. Different entities are related using relationships. **E-R** Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand. This model is good to design a database, which can then be turned into tables in relational model (explained below).

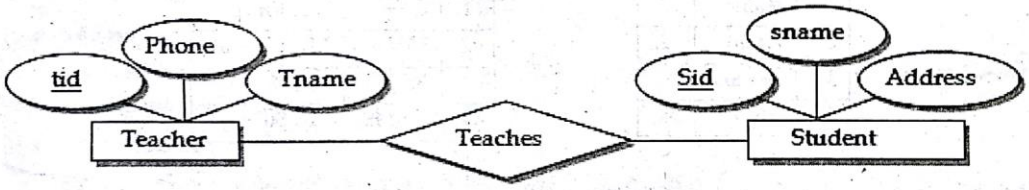
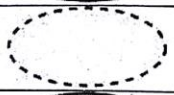
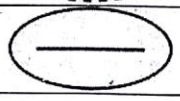


Figure 2.3 ER diagram

Following are the components of ER Diagram,

Notations	Representation	Description
	Rectangle	It represents the Entity.
	Ellipse	It represents the Attribute.
	Diamond	It represents the Relationship.
	Line	It represents the link between attribute and entity set to relationship set.
	Double Rectangle	It represents the weak entity.
	Composite Attribute	It represents composite attribute which can be divided into subparts. For e.g. Name can be divided into First Name and Last Name

	Multi valued Attribute	It represents multi valued attribute which can have many values for a particular entity. For e.g. Mobile Number.
	Derived Attribute	It represents the derived attribute which can be derived from the value of related attribute.
	Key Attribute	It represents key attribute of an entity which have a unique value in a table. For e.g. Employee → EmpId (Employee Id is Unique).

Relational Model

In this model, data is organized in two-dimensional tables and the relationship is maintained by storing a common field i.e. by using primary key and foreign key. This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, in fact, we can say the only database model used around the world. The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table. Hence, tables are also known as relations in relational model. In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.

Student

sid	sname	age
1	Aabin	17
2	Aarav	22
3	Ashana	15
4	Anuj	34

Course

cid	Cname	teacher
c1	C++	Mr. Bhupi
c2	JAVA	Mr. Deepak
c3	NM	Mr. IC
c4	DBMS	Mr. Sukraj

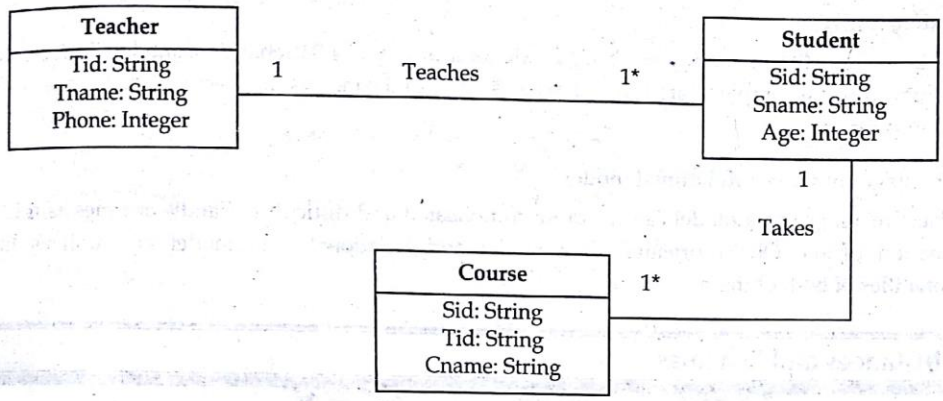
Student-course

sid	cid	marks
1	c1	77
2	c1	98
3	c3	67
4	c2	34
2	c4	98

Figure 2.4 Relational data model

Object Oriented Model

The object-oriented model can be seen as extending the E-R model with notions object oriented data model. Object model stores the data in the form of objects, classes and inheritance. This model handles more complex applications, such as Geographic Information System (GIS), scientific experiments, engineering design and manufacturing. It is used in File Management System. It represents real world objects, attributes and behaviors. It provides a clear modular structure and easy to maintain and modify the existing code.



Elements of object oriented data model

- **Objects**

  The real world entities and situations are represented as objects in the Object oriented database model.

- **Attributes and Method**

  Every object has certain characteristics. These are represented using Attributes. The behavior of the objects is represented using Methods.

- **Class**

  Similar attributes and methods are grouped together using a class. An object can be called as an instance of the class.

- **Inheritance**

  A new class can be derived from the original class. The derived class contains attributes and methods of the original class as well as its own.

Object-Relational Data Model

An Object relational model is a combination of a Object oriented database model and a Relational database model. So, it supports objects, classes, inheritance etc. just like Object Oriented models and has support for data types, tabular structures etc. like Relational data model. One of the major goals of Object relational data model is to close the gap between relational databases and the object oriented practices frequently used in many programming languages such as C++, C#, Java etc.

Advantages of Object Relational model

The advantages of the Object Relational model are:

- **Inheritance**

  The Object Relational data model allows its users to inherit objects, tables etc. so that they can extend their functionality. Inherited objects contain new attributes as well as the attributes that were inherited.

- **Complex Data Types**

  Complex data types can be formed using existing data types. This is useful in Object relational data model as complex data types allow better manipulation of the data.

- **Extensibility**

  The functionality of the system can be extended in Object relational data model. This can be achieved using complex data types as well as advanced concepts of object oriented model such as inheritance.

Disadvantages of Object Relational model

The object relational data model can get quite complicated and difficult to handle at times as it is a combination of the Object oriented data model and Relational data model and utilizes the functionalities of both of them.

2.2 Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema. Schemas are changed infrequently, if at all. A database schema is the skeleton structure of the database. It represents the logical view of the entire database.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction. The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called sub-schema that describe different views of the database.

Database Schema

Eid	Ename	Address	Salary	Age
E01	Ayan	Kathmandu	35000	25
E02	Bishnu	Pokhara	27000	28
E03	Dinu	Dodhara	25000	26
E04	Bhanu	Ghorahi	32000	37
E05	Rina	Dhangadhi	28000	27
E06	Aarav	Mahendranagar	30000	25

Database instances

2.3 Data abstraction / Three-Schema Architecture

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction. Database System provides users with an abstract view of the data. Data abstraction is provided in database management systems by using three-level schema (ANSI /SPARC) architecture.

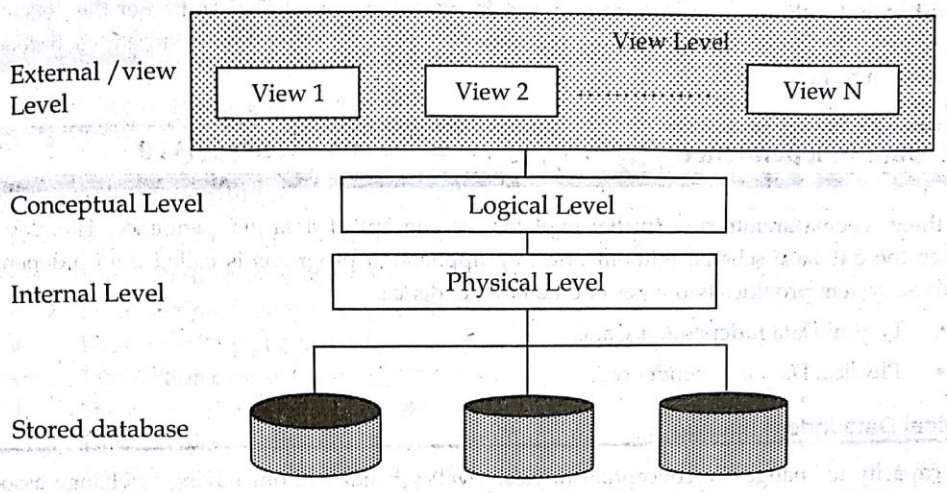


Figure 2.5 Three level schema architecture

Physical level

It is the lowest level of abstractions and describes how the data are actually stored on disk and some of the access mechanisms commonly used for retrieving this data. While designing this layer, the main objective is to optimize performance by minimizing the number of disk accesses during the various database operations. The database system hides many of the lowest level storage details from database programmer. DBMS developer is the person who deals with this level. Database administrator may be aware of certain details of the physical organization of the data.

Logical level

It is the next higher level of data abstraction which describes what data are stored in the database, and what relationships exist among those data. It is also known as conceptual level at the schema at this level is called logical schema (conceptual schema). Logical level describes the stored data in terms of the data model of the DBMS. Programmers and database administrator works at this level of abstraction. Database users do not need to have knowledge of this level.

View level

It is the highest level of abstraction and describes only a part of the database and hides some information from the user. At view level, computer users see a set of application programs that hide details of data types. Similarly at the view level several views of the database are defined and database user see only these views. Schema at this level is called external schema. Views also plays vital role to provide security mechanism to prevent users from accessing certain parts of the database such as an employee's salary for security purposes.

Example: Let's say we are storing customer information in a customer table.

- **At physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.
- **At the logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

- **At view level,** user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

2.4 Data Independence / 3. schema Architecture.

The three schema architecture further explains the concept of data independence. The capacity to change the database schema without affecting application programs is called data independence. Database system provides two types of data independence:

- Logical Data Independence and
- Physical Data Independence.

Logical Data Independence

The capacity to change the conceptual (logical level) schema without having to change associated application programs is called logical data independence. If the underlying conceptual schema is changed, the definition of a view relation can be modified so that the same relation is computed as before. Database administrator is responsible for redefining view level schema. When modification is done to the conceptual schema (i.e. tables) only the external/conceptual mapping need to be changed, if the DBMS fully supports the concept of data independence.

Physical Data Independence

The capacity to change the internal schema without affecting application programs is called physical data independence. This means we can change physical level storage details such as file structure; indexes as long as conceptual schema remains same without altering associated application programs. But performance may be affected due to changes in physical level. It is the responsibility of the DBA to manage such changes.

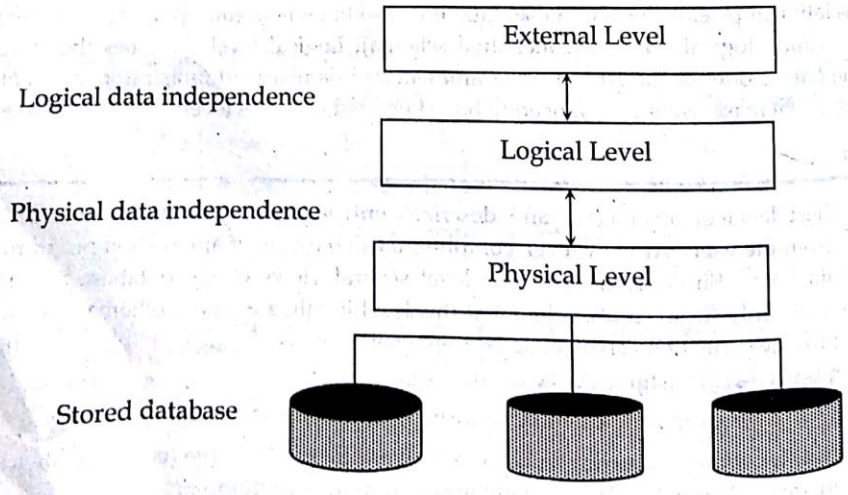


Figure 2.6 Data independence.

2.5 Database Languages and Interfaces

A DBMS has appropriate languages and interfaces to express database queries and updates. Database languages can be used to read, store and update the data in the database.

Types of Database Language

There are mainly four types of database languages which are listed below:

- Data definition language (DDL)
- Data manipulation language (DML)
- Data control language (DCL)
- Transaction control language (TCL)

a. Data definition language (DDL)

DDL stands for Data Definition Language. It is used to define database structure or pattern. It is used to create schema, tables, indexes, constraints, etc. in the database. Using the DDL statements, we can create the skeleton of the database. It is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc. Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to delete table in database instance.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

b. Data Manipulation Language (DML)

DML stands for Data Manipulation Language. It is used for accessing and manipulating data in a database. Database language that enables insert, delete, update, and retrieve data from database is called data manipulation language (DML). Query language is only a part of data manipulation language. DML statements are converted into equivalent low level statements by DML compiler. Structured query language supports following DML statements:

- **Select:** It is used to retrieve data from tables.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete records from a table.

c. Data control Language (DCL)

DCL stands for Data Control Language. It is used to retrieve the stored or saved data.

The DCL execution is transactional. It also has rollback parameters. (But in Oracle database, the execution of data control language does not have the feature of rolling back). Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

d. Transaction Control Language

Transaction Control Language (TCL) commands is used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions. Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

2.6 Difference between Rollback and Commit Commands

Rollback	Commit
Rollback command is used to undo the changes made by the DML commands.	The Commit command is used to save the modifications done to the database values by the DML commands.
It rollbacks all the changes of the current transaction.	It will make all the changes permanent that cannot be rolled back.
Syntax: DELETE FROM table_name ROLLBACK	**Syntax**: COMMIT;

2.7 Database System Environment

Database System Environment deals with the components of an organization that defines and regulate the collection, storage, management and use of data within a database environment. Database system is composed of different major parts. They are: Hardware, Software, People, Procedures, Data, etc.

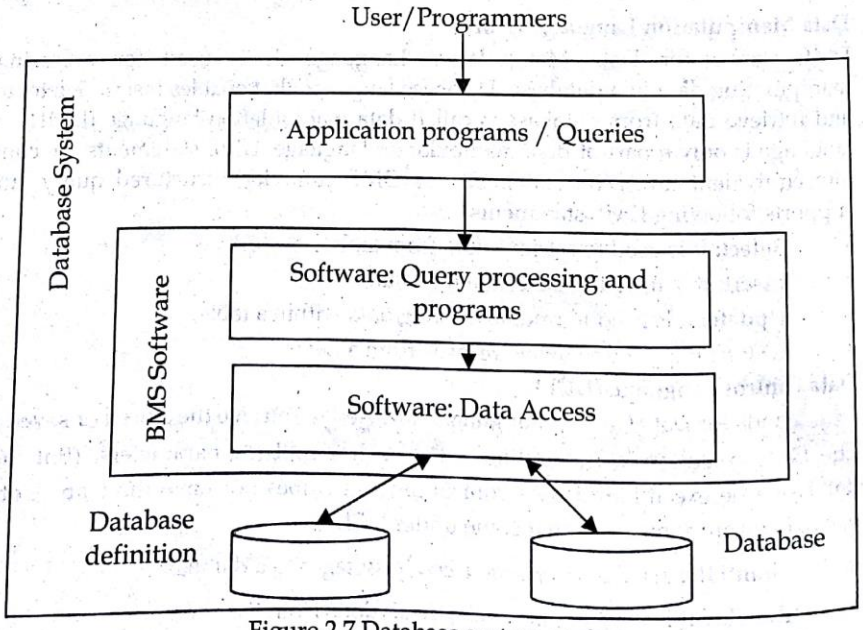


Figure 2.7 Database system environment

Hardware

It identifies all the system's physical devices. The database system's main and most easily identified hardware component is the computer, which might be a microprocessor, a minicomputer, or a mainframe computer. It also includes peripherals like keyboard, mice, modems, printers, etc.

Software

Software refers to the collection of programs used by the computers within the database system.

- **Operating system Software:** It manages all hardware components and makes it possible for all other software to run on the computers. DOS, OS/2, and Windows used by micro computers. UNIX and VMS used by mini computers, and MVS used by IBM mainframe computers.
- **DBMS Software:** It manages the database within the database system. MS-Access, MSSQL Server, Oracle, DB2 etc. are some famous DBMS software.
- **Application Programs & Utilities Software:** They are used to access and manipulate the data in the DBMS to manage the computer environment in which data access and manipulation take place.

People

- **System Administrator:** They oversee the database system's general operations. Database **Administrator:** manage the DBMS's use and ensure that the database is functioning properly.
- **Database Designers:** They design the database structure. They are in effect the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs will fail to produce a useful database environment.
- **System Analysts and Programmers:** They design and implement the application programs. They design and create the data entry screens, reports, and procedures through which end user access and manipulate the database's data.
- **End Users:** They are the people who use the application programs to run the organization's daily operations. E.g. sales clerks, supervisors, managers, and directors are all classified as end users. High-level end users employ the information obtained from the database to make tactical and strategic business decisions.

Procedures

Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play a very important role in a company, because they enforce the standards by which business is conducted within the organization and with customers. These also are used to ensure that there is an organized way to monitor and audit both the data and information.

Data

The word data covers the collection of facts stored in the database. Because data are the raw material from which information is generated, the determination of which data are to be entered into the database and how such data are to be organized is a vital part of the database designer's job.

2.8 Types of Database System

The type of computer systems that database can run be broken down into four broad categories or platforms which are listed below:

1. Centralized database system
2. Personal computer system
3. Client/server database system
4. Distributed processing database system

Centralized database system

In a centralized system, all programs run on the main host computer, including the DBMS, the application that accesses the database and the communication facilities that send and receive data from the user's terminals. The users access the database through either locally connected or dial-up (remote) terminals. The terminals are generally dumb, having little or no processing power of their own and consists of only a screen, keyboard and hardware to communicate with the host.

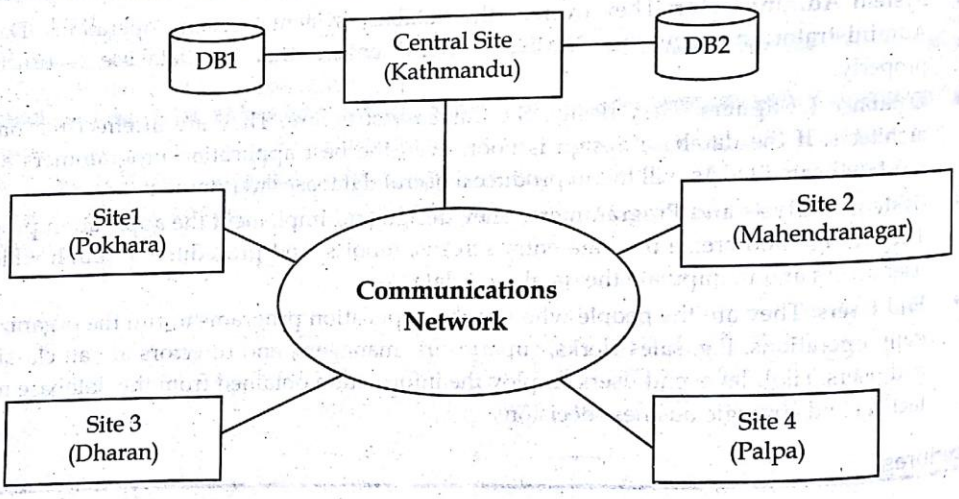


Figure 2.8 Centralized Database

Personal Computer system

When a DBMS is run on a PC, the PC acts as both the host computer and the terminal unlike the larger system. The DBMS functions and the database application functions are combined into one application. Database applications on a PC handle the user input, screen output and access to the data on the disk. Combining these different functions into one unit gives the DBMS a great deal of power, flexibility and speed, usually at the cost of decrease data security and integrity.

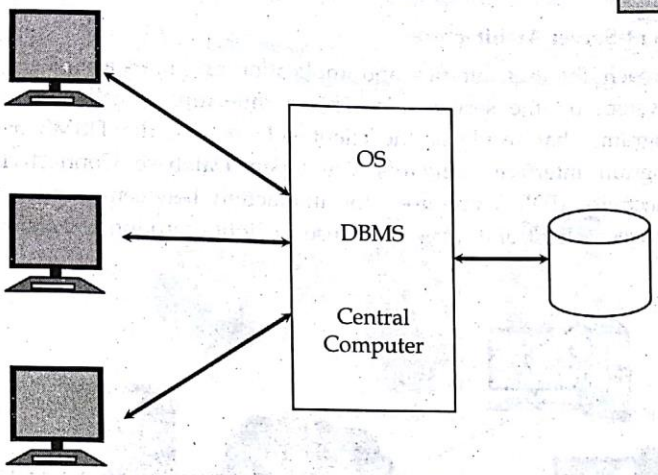


Figure 2.9 Personal computer system

Client/Server database system

In a gerneralized concept, client PC is the computer from where the user requests for data and information and the server provides the requested information. The database application on the client PC referred to as the 'front end system' athat handles all the screen and user input/output processing.

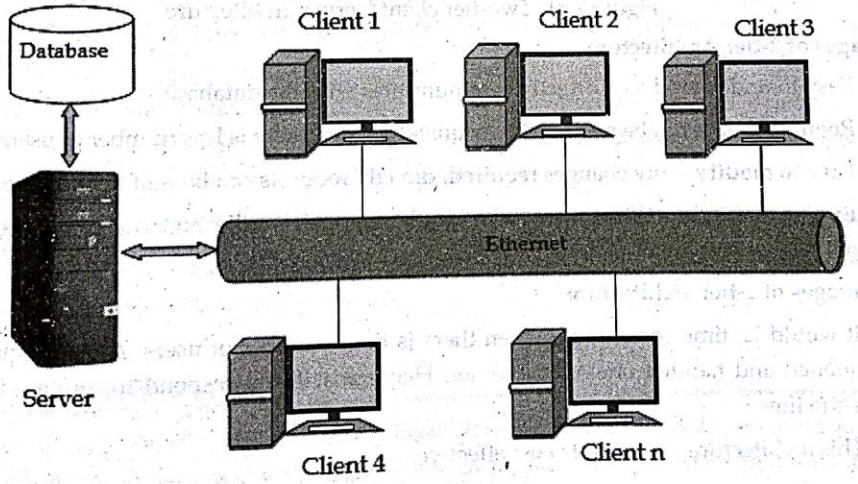


Figure 2.10: Client server database system

The 'back end system' on the database server handles data processing and disk access. For example a user on the front end creates a query for data from the database server and the front-end application sends the request across the network to the server. The database server performs the actual search and sends back only the data that answers the user's query. There are two approaches to implement client/server architecture: two-tier architecture and three-tier architecture.

a. Two Tier Client-Server Architecture

In the first approach, the user interface and application programs are placed on the client side and the database system on the server side. This architecture is called two-tier architecture. The application programs that reside at the client side invoke the DBMS at the server side. The application program interface standards like Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) are used for interaction between client and server.There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.

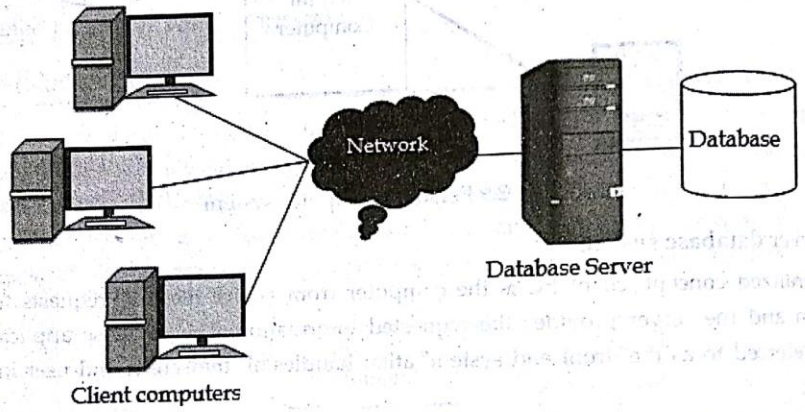


Figure 2.11: Two-tier client/server architecture

Advantages of 2-tier Architecture

- Easy to understand as it directly communicates with the database.
- Requested data can be retrieved very quickly, when there is less number of users.
- **Easy to modify** – any changes required, directly requests can be sent to database
- **Easy to maintain** – When there are multiple requests, it will be handled in a queue and there will not be any chaos.

Disadvantages of 2-tier architecture

- It would be time consuming, when there is huge number of users. All the requests will be queued and handed one after another. Hence it will not respond to multiple users at the same time.
- This architecture would little cost effective.

b. Three Tier Client-Server Architecture

Three-tier schema is an extension of the 2-tier architecture. 3-tier architecture has following layers

- Presentation layer (your PC, Tablet, Mobile, etc.)
- Application layer (server)
- Database Server

This DBMS architecture contains an Application layer between the user and the DBMS, which is responsible for communicating the user's request to the DBMS system and send the response from

the DBMS to the user.The application layer(business logic layer) also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS. Three tier architecture is the most popular DBMS architecture.

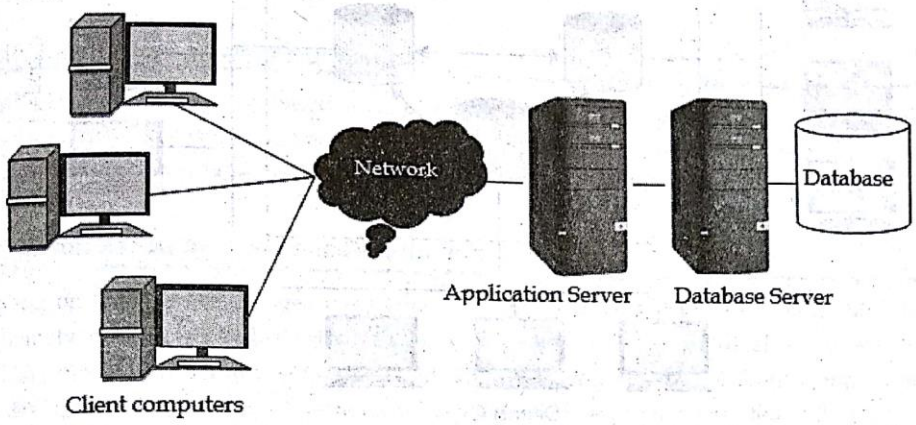


Figure 2.12: three-tier clien/server architecture

Advantages of 3-tier architecture

- **Easy to maintain and modify:** Any changes requested will not affect any other data in the database. Application layer will do all the validations.
- **Improved security:** Since there is no direct access to the database, data security is increased. There is no fear of mishandling the data. Application layer filters out all the malicious actions.
- **Good performance:** Since this architecture cache the data once retrieved, there is no need to hit the database for each request. This reduces the time consumed for multiple requests and hence enables the system to respond at the same time.

Disadvantages 3-tier Architecture

- Disadvantages of 3-tier architecture are that it is little more complex and little more effort is required in terms of hitting the database.

Distributed processing database system

In this system data is shared among various host system via updates sent either through direct connections on the same network or through remote connections via phone or dedicated data lines. A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network. Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites. The processors in the sites are connected via a network. They do not have any multiprocessor configuration. A distributed database is not a loosely connected file system. A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

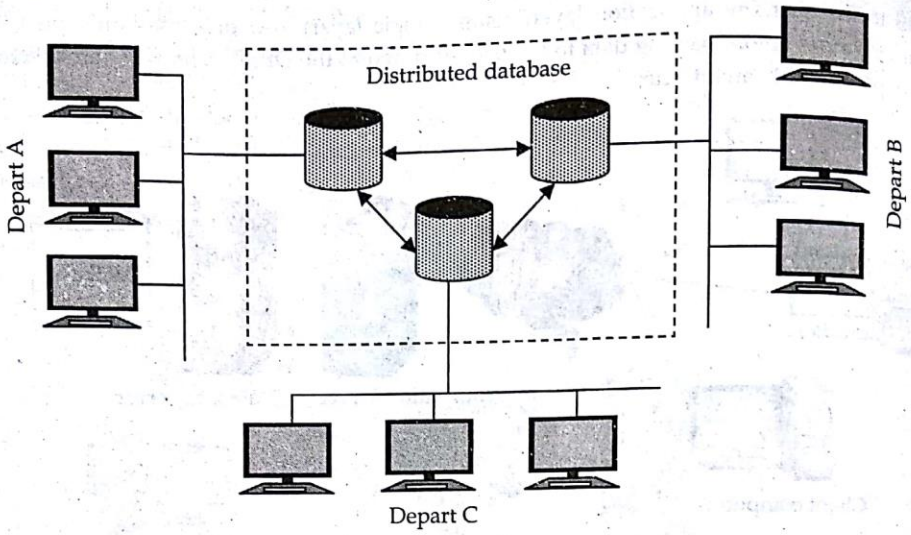


Figure 2.13: Distributed database processing system.

An application which one or more of the hosts, extracts the portion of data that has been changed during a programmer-defined period and then transmits the data to either a centralized host or other hosts in the distributed circuit. The other databases are then updated so that all the systems are in synchronized with each other. This type of distributed processing usually occurs between departmental computers or LAN's and host system; the data goes to a large central minicomputer or mainframe host after the close of the business day.

2.9 Classification of Database Management Systems

Database management systems can be classified based on several criteria, such as the data model, user numbers and database distribution, all described below.

Classification Based on Data Model

The most popular data model in use today is the relational data model. Well-known DBMSs like Oracle, MS SQL Server, DB2 and MySQL support this model. Other traditional models, such as hierarchical data models and network data models are still used in industry mainly on mainframe platforms. However, they are not commonly used due to their complexity. These are all referred to as traditional models because they preceded the relational model.

In recent years, the newer object-oriented data models were introduced. This model is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object-oriented databases are different from relational databases, which are table-oriented. Object-oriented database management systems (OODBMS) combine database

capabilities with object-oriented programming language capabilities. The object-oriented models have not caught on as expected so are not in widespread use. Some examples of object-oriented DBMSs are O2, Object Store and Jasmine.

Classification Based on User Numbers

A DBMS can be classification based on the number of users it supports. It can be a single-user database system, which supports one user at a time, or a multiuser database system, which supports multiple users concurrently. In multi-user DBMS, the data is both integrated and shared.

Classification Based on Database Distribution

Depending on the number of sites over which the database is distributed, it is divided into two types, namely, centralized and distributed database systems. Centralized database systems run on a single computer system. Both the database and DBMS software reside at a single computer site. The user interacts with the centralized system through a dummy terminal connected to it for information retrieval. In distributed database systems, the database and DBMS are distributed over several computers located at different sites. The computers communicate with each other through various communication media. Distributed databases can be classified as homogeneous and heterogeneous. In homogeneous distributed database system, all sites have identical database management system software, whereas in heterogeneous distributed database system, different sites use different database management system software.

- **Centralized systems**

  With a centralized database system, the DBMS and database are stored at a single site that is used by several other systems too.

- **Distributed database system**

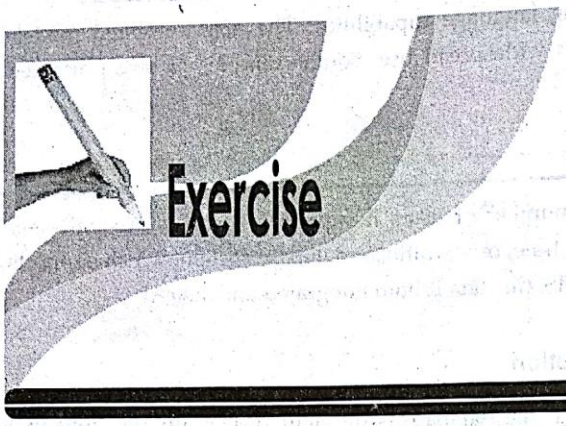
  In a distributed database system, the actual database and the DBMS software are distributed from various sites that are connected by a computer network.

- **Homogeneous distributed database systems**

  Homogeneous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily. For example, library information systems by the same vendor, such as **Geac** Computer Corporation, use the same DBMS software which allows easy data exchange between the various **Geac** library sites.

- **Heterogeneous distributed database systems**

  In a heterogeneous distributed database system, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites. For example, the various library database systems use the same machine-readable cataloguing (MARC) format to support library record data exchange.



1. What is data model? Explain ER data modeling with suitable example.
2. What is the difference between logical and physical data independence?
3. Illustrate the differences between hierarchical and network data models. Explain why relational model is preferred over the two.
4. What is the difference between object-based data models and record-based data models?
5. What are the roles of system analyst and application programmer in database system?
6. Write a short note on data definition language and data manipulation language.
7. Explain the various functional components of a DBMS with the help of a suitable diagram.
8. What is the difference between a centralized and a client/server database system?
9. How is two-tier architecture different from three-tier architecture?
10. Explain the different criteria on the basis of which DBMS is classified into different categories.
11. What are the advantages of network data model over hierarchical data model? Explain.
12. Differentiate between logical data independence and physical data independence.
13. Describe advantages of object oriented data modeling with suitable example.
14. Explain DDL and DML commands.
15. Explain DCL commands with suitable example.
16. What is transaction? Explain role of transaction in DBMS.
17. Describe purpose of data model in DBMS.
18. Describe distributed database with suitable example.
19. How centralized database differ from distributed database explain.
20. Classify types of database on the bases of number of user are used.

❑❑❑

CHAPTER 3

DATA MODELING USING THE ENTITY-RELATIONAL MODEL

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Using High-Level Conceptual Data Models for Database Design
- Entity Types, Entity Sets, Attributes, and Keys
- Relationship Types, Relationship Sets, Roles, and Structural Constraints
- Weak Entity Types
- ER Diagrams, Naming Conventions, and Design Issues
- Relationship Types of Degree Higher Than Two
- Subclasses, Super classes, and Inheritance
- Specialization and Generalization
- Constraints and Characteristics of Specialization and Generalization

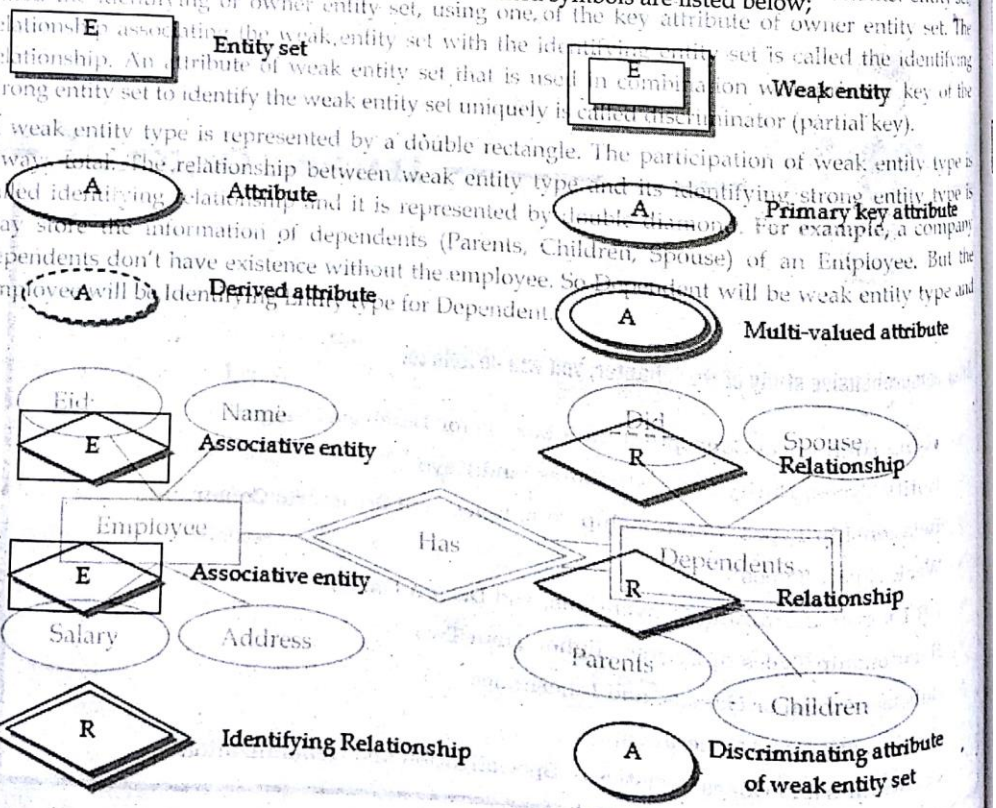
3.1 Introduction to Entity-Relationship Diagram

The E-R data models is based on a perception of real world that consist of a collection of basic objects called entities and relationship among these objects. In an E-R model a database can be modeled as a collection of entities, and relationship among entities.

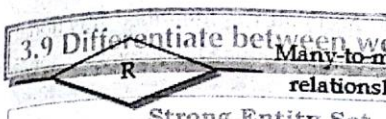
Once the entity types, relationships types, and their corresponding attributes have been identified, the next step is to graphically represent these components using entity-relationship (E-R) diagram. An E-R diagram is a specialized graphical tool that demonstrates the interrelationships among various entities of a database. It is used to represent the overall logical structure of the database. While designing E-R diagrams, the emphasis is on the schema of the database and not on the instances. This is because the schema of the database is changed rarely; however, the instances in the entity and relationship sets change frequently. Thus, E-R diagrams are more useful in designing the database. E-R diagram focuses high level database design and hides low level details of database representation therefore it can be used to communicate with users of the system while collecting information.

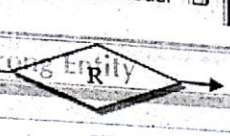
3.2 Symbols used in ER-Diagram/ER Naming Conventions

Entity-relationship diagrams (ERD) are essential to modeling anything from simple to complex databases, but the shapes and notations used can be very confusing. There are various types of symbols are used for ER diagram some of well-defined symbols are listed below;

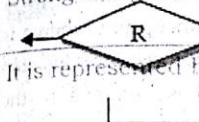


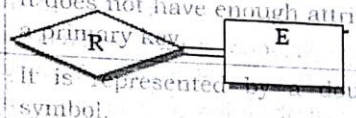
3.9 Differentiate between weak entity and Strong Entity

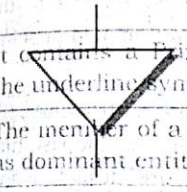


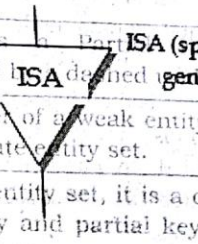


Strong Entity Set	Weak Entity Set
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial key represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.









3.3 Using High-Level Conceptual Data Models for Database Design

Following figure shows a simplified overview of the database design process.

- The first step shown is **requirements collection and analysis.** During this step, the database designers interview prospective database users to understand and document their data requirements. The result of this step is a concisely written set of users' requirements. These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user-defined operations (or transactions) that will be applied to the database, including both retrievals and updates. In example software design, it is common to use data flow diagrams, sequence diagrams, scenarios, and other techniques to specify functional requirements.
- Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called conceptual design. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model. Because these concepts do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users. The high-level conceptual schema can also be used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict. This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details. This makes it is easier to create a good conceptual data-base design.
- During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis.

This also serves to confirm that the conceptual schema meets all the identified functional requirements. Modifications to the conceptual schema can be introduced if some functional requirements cannot be specified using the initial schema.

- The next step in database design is the **actual implementation of the database**, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS. Data model mapping is often automated or semi-automated within the database design tools.
- The last step is the **physical design phase**, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications.

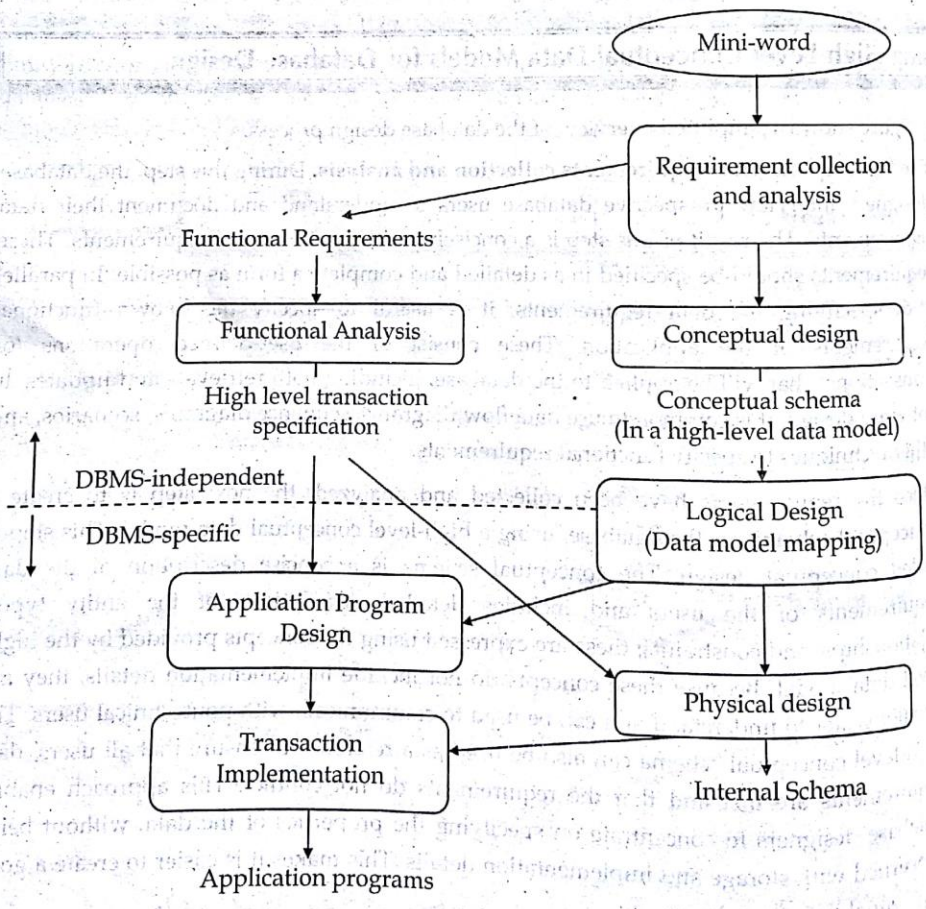


Figure 3.1 Diagram showing the main phases of database design

3.4 Elements of ER Diagram / ER Design Issues

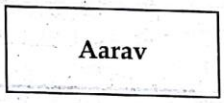
Entity, Entity type and Entity set

Entity

An entity is a real-world thing either living or non-living that is easily recognizable and non-recognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world. An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Simply, it is something which has real existence. Like tuple1 contains information about Aarav (id, name and Age) which has existence in real world. So the tuple1 is an entity. So we may say each tuple is an entity.

Example: let "Aarav" is a particular member of entity type student.

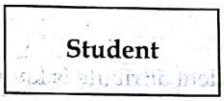


Entity

Entity type

It is collection of entities having common attribute. As in Student table each row is an entity and has common attributes. So STUDENT is an entity type which contains entities having attributes id, name and Age. Also each entity type in a database is described by a name and a list of attribute. So we may say a table is an entity type.

Example: Collection of entities with similar properties

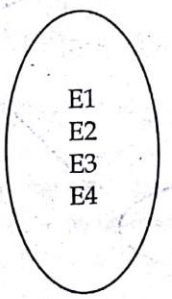


Entity type

Entity set

It is same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.

Example: let E1 is an entity having entity type Student and set of all students is called entity set.



Entity set

Attributes

Attributes are the properties which define the entity type. For example, Student_id, student_name, DOB, Age, Address, Mobile_No etc. are the attributes which defines entity type **Student**. In ER diagram, attribute is represented by an oval.

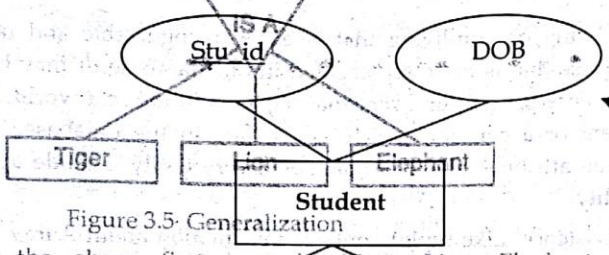


Figure 3.5 Generalization

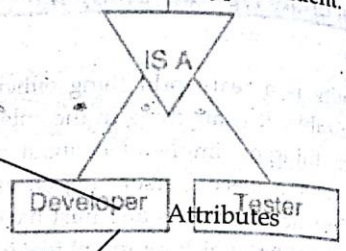


Figure 3.6 Specialization

In the above first example, Tiger, Lion, Elephant can all be generalized as Animals. Also in the above second example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

Category or Union

It is possible that single super class/subclass relationship has more than one super-class representing different entity types. In this case, the subclass will represent a collection of objects that is (a subset of) the UNION of distinct entity types; we call such a subclass a union type or a category.

Category represents a single super class or sub class relationship with more than one super classes whereas non-category super class/subclass relationships always have a single super class. It can be a total or partial participation. For example, in booking, Car owner can be a person, a bank (holds a possession) or a company.

Category Owner is a subset of the union of the three super Classes — Company, Bank, and Person. A Category member must exist in at least one of its super classes.

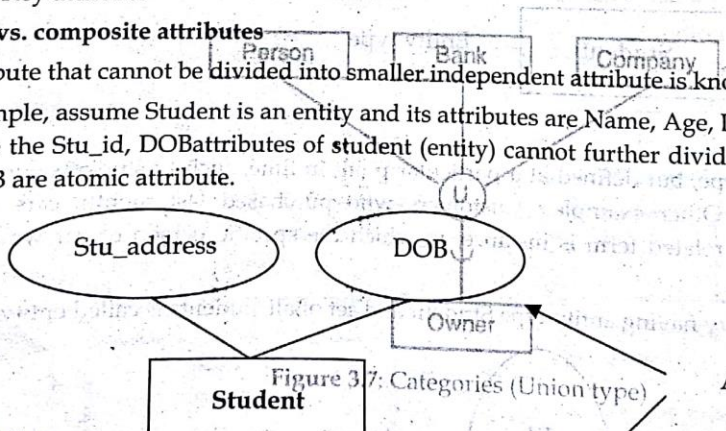


Figure 3.7: Categories (Union type)

Aggregation

The E-R model cannot express relationships among relationship sets and relationships between relationship sets and entity sets. Consider a DB with information about Colleges that offer courses and students employed by colleges and courses offered by college. One alternative in this case is to define ternary relationship between {College, course, and Student} as in figure below.

Types of Attributes

Attributes of an entity type can be further divided into following types

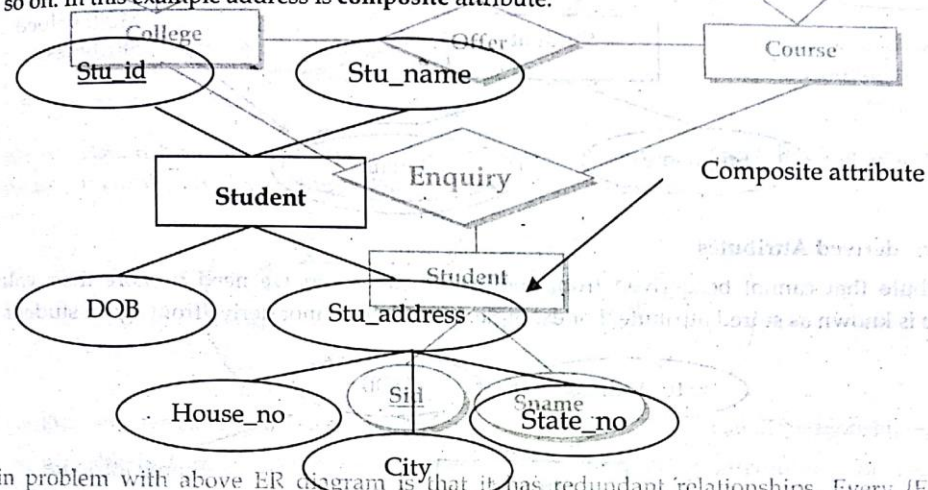
- Atomic vs. composite attributes
- Single valued vs. multi valued attributes
- Stored vs. derived attributes
- NULL value attribute
- Key attribute

Atomic vs. composite attributes

An attribute that cannot be divided into smaller independent attribute is known as atomic attribute.

For example, assume Student is an entity and its attributes are Name, Age, DOB, Address and Phone no. Here the Stu_id, DOBattributes of student (entity) cannot further divide. In this example Stu_id and DOB are atomic attribute.

An attribute that can be divided into smaller independent attribute is known as composite attribute. For example, assume Student is an entity and its attributes are Stu_id, Name, DOB, Address and Phone no. Here the address (attribute) of student (entity) can be further divide into House no, city and so on. In this example address is **composite** attribute.

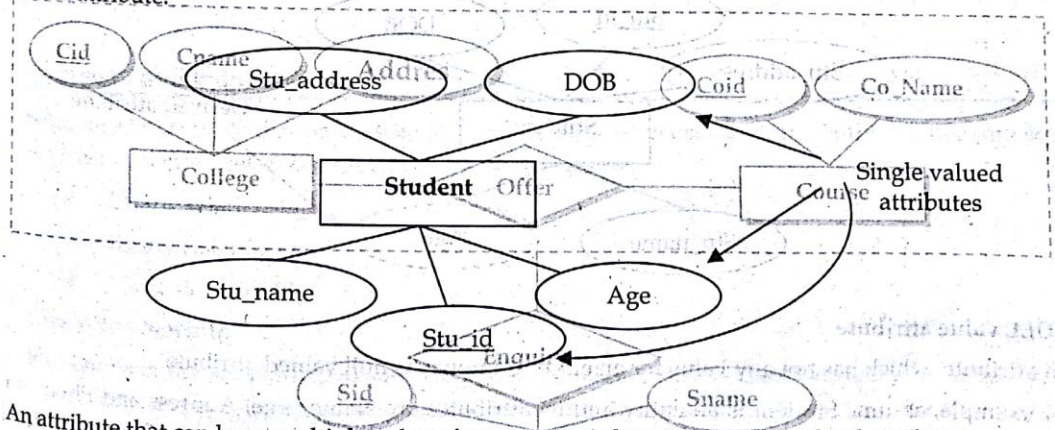


Main problem with above ER diagram is that it has redundant relationships. Every {Employee, Project} combination that appears in uses relationship set also appears in works relationship set. The solution is to use aggregation.

Single valued vs. multi valued attributes a relationship between a whole object and its component parts. Using aggregation we can express relationship among relationships. Aggregation shows 'has-a' or 'is-part-of' relationship between entities where one represents the 'whole' and other 'part'. It abstracts a relationship between objects and viewing the relationship as an object. It is a process when two entity and relationship with these entities is treated as a higher level single entity set.

An attribute that has only single value for an entity is known as single valued attribute.

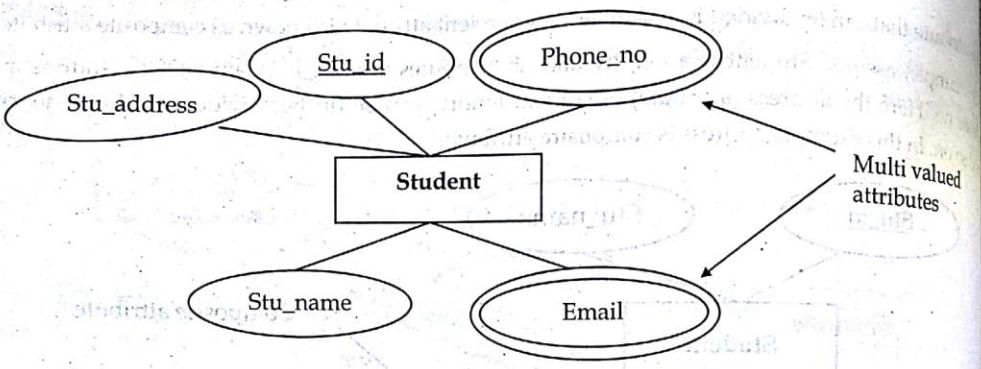
For example, assume Student is an entity and its attributes are Stu_id, Name, DOB, age, Address and Phone no. Here the age (attribute) of student (entity) can have only one value. Here, age is single valued attribute.



An attribute that can have multiple values for an entity is known as multi valued attribute.

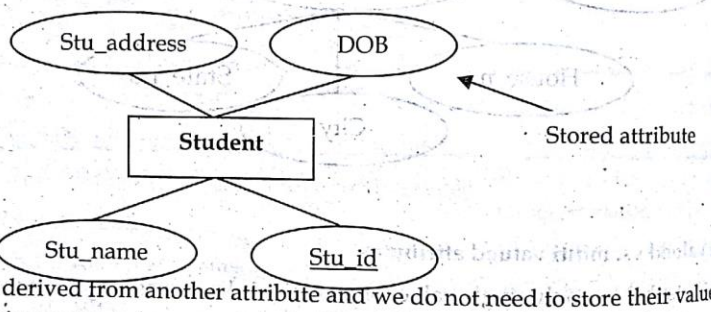
For example, assume Student is an entity and its attributes are Stu_id, Name, Age, Address and Phone no. Here the Phone no (attribute) of student (entity) can have multiple value because a student may have many phone numbers. Here, Phone no is multi valued attribute.

Figure 3.8 Aggregation in Relation with Student.

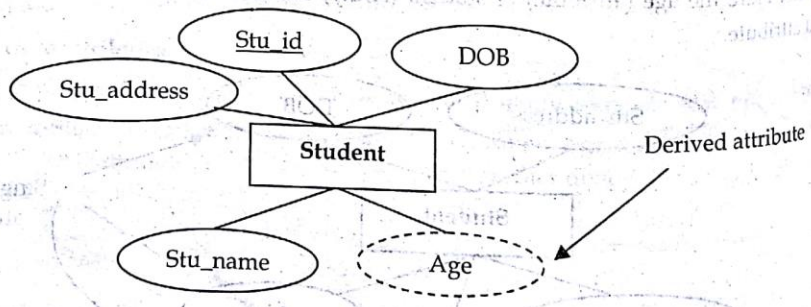


Stored vs. derived Attributes

An attribute that cannot be derived from another attribute and we need to store their value in database is known as stored attribute. For example, birth date cannot derive from age of student.



An attribute that can be derived from another attribute and we do not need to store their value in the database due to dynamic nature is known as derived attribute. It is denoted by doted oval. For example, age cannot derive from birth date of student.



NULL value attribute

An attribute, which has not any value for an entity is known as null valued attribute.

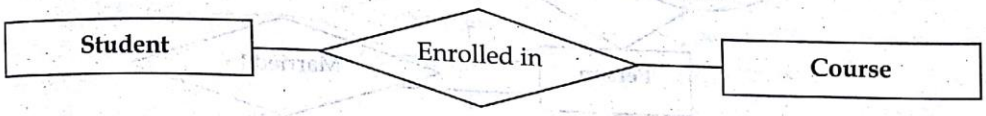
For example, assume Student is an entity and its attributes are Name, Age, Address and Phone no. There may be chance when a student has no phone no. In that case, phone no is called null valued attributes.

Key attribute

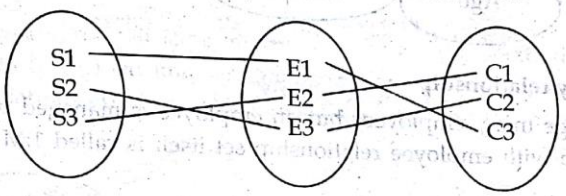
An attribute that has unique value of each entity is known as key attribute. For example, every student has unique roll no. Here roll no is key attribute.

Relationship type and Relationship set

A relationship type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

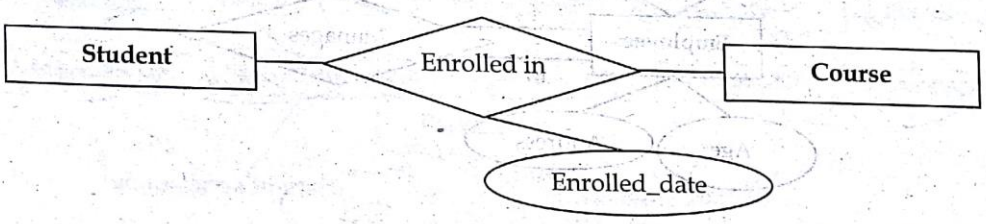


A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



In another way, we can say that association between two entity sets is called relationship set.

A relationship set may also have attributes called **descriptive attributes**. For example, the Enrolled_in relationship set between entity sets **student** and **course** may have the attribute enrolled-_date.



Degree of a Relationship

Number of entity sets that participate in a relationship set is called degree of the relationship set. On the basis of degree, relationships can be divided as below:

- Unary Relationship
- Binary Relationship
- N-ary Relationship

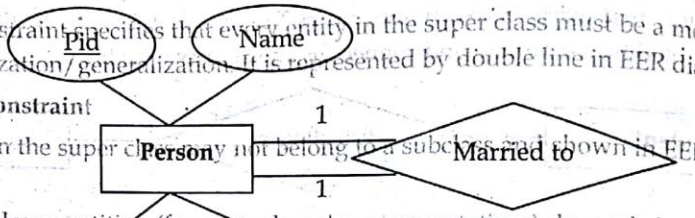
Unary Relationship

If only one entity set participates in a relation, the relationship is called as unary relationship. Here same entity set participates in relationship twice with different roles. Role names are specified above the link joining entity set and relationship set. This type of relationship set is sometimes called a recursive relationship set. There are three types of unary relationships:

- 1:1 unary relationship
- 1:M unary relationship
- M:N unary relationship

One to one (1:1) unary relationship Constraint

In the example below, one person is married to only one person.



Total participation constraint specifies that every entity in the super class must be a member of some subclass in the specialization/generalization. It is represented by double line in EER diagram.

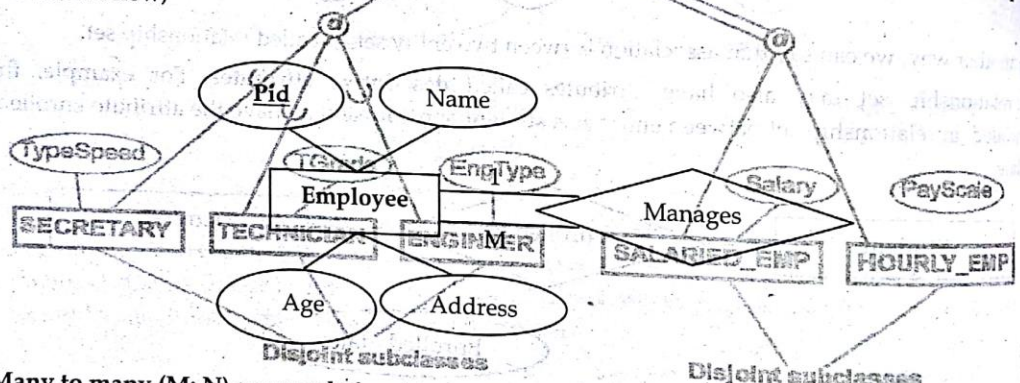
Partial participation constraint

It allows every entity in the super class may not belong to a subclass and shown in EER diagrams by a single line.

Example: if some Employee entities, (for example, sales representatives) do not belong to any of the subclasses {Secretary, Engineer, Technician}, then the specialization is partial.

One to many (1: M) unary relationship

An employee may manage many employees but an employee is managed by only one employee. This types of relationship with employee relationship set itself is called 1:M unary relationship as shown in below;

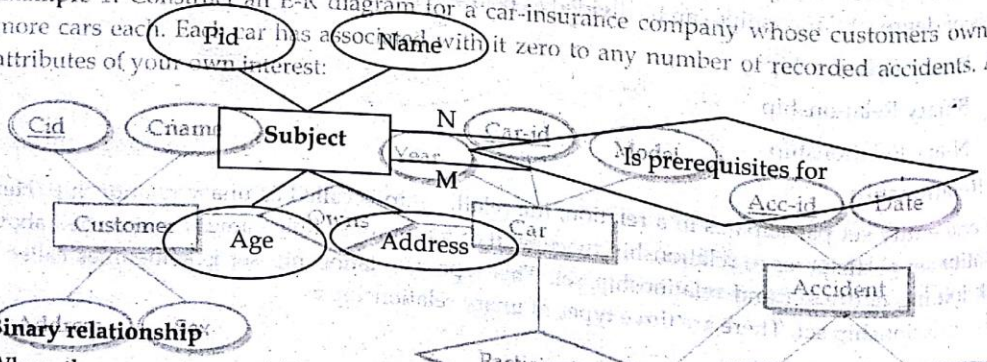


Many to many (M: N) unary relationship

A subject may have many other subjects as prerequisites and each subject may be a prerequisite to many other subjects.

3.13 Examples of ER diagram

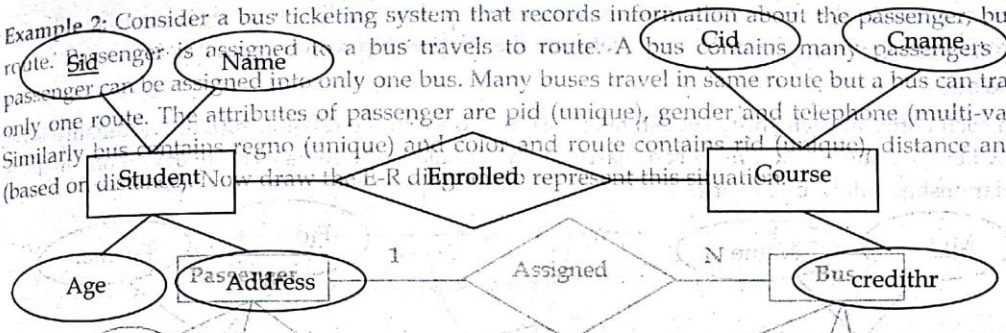
Example 1: Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Assume attributes of your own interest:



Binary relationship

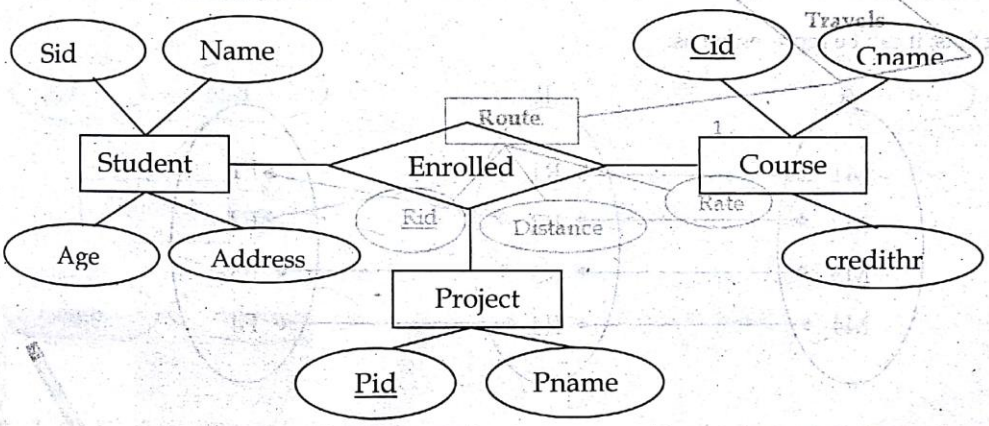
When there are two entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course. This is the most common type of relationship in database systems.

Example 2: Consider a bus ticketing system that records information about the passenger, bus and route. Passenger is assigned to a bus travels to route. A bus contains many passengers and a passenger can be assigned into only one bus. Many buses travel in same route but a bus can travel in only one route. The attributes of passenger are pid (unique), gender and telephone (multi-valued). Similarly bus contains regno (unique) and color and route contains rid (unique), distance and rate (based on distance). Now draw the E-R diagram to represent this situation.



N-ary relationship

When there are n entities set participating in a relation, the relationship is called as n-ary relationship. If n=1 then it is called unary relationship, if n=2 then it is called binary relationship. Generally in N-ary relationship there are more than two entities participating with a single relationship i.e. n > 2.



3.5 Constraints on ER Model/Structural Constraint in ER

Relationship sets in ER model usually have certain constraints that limit the possible combinations of entities that may involve in the corresponding relationship set. Database content must confirm these constraints. The most important structural constraints in ER are listed below:

- Mapping cardinalities and
- Participation constraints

Mapping Cardinality Constraints

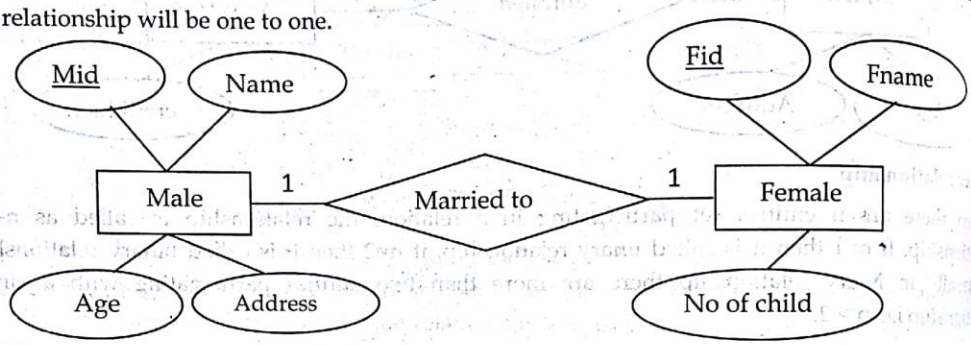
The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many

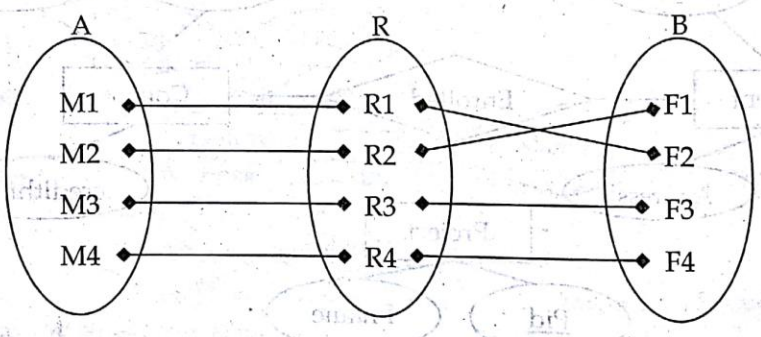
We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

One to one

When each entity in each entity set can take part only once in the relationship, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

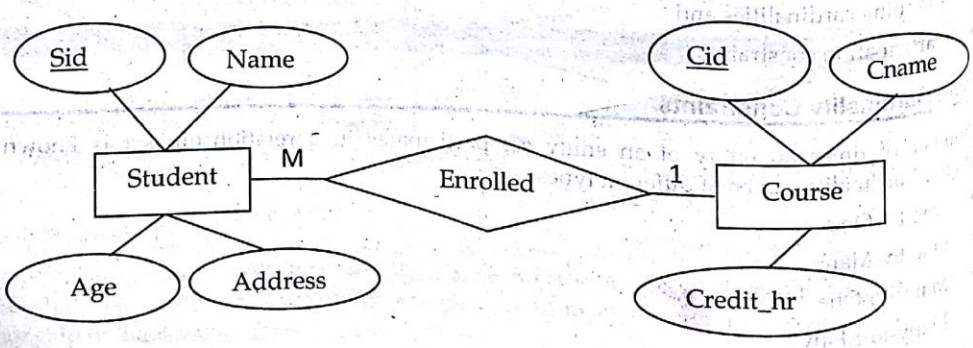


Using Sets, it can be represented as:

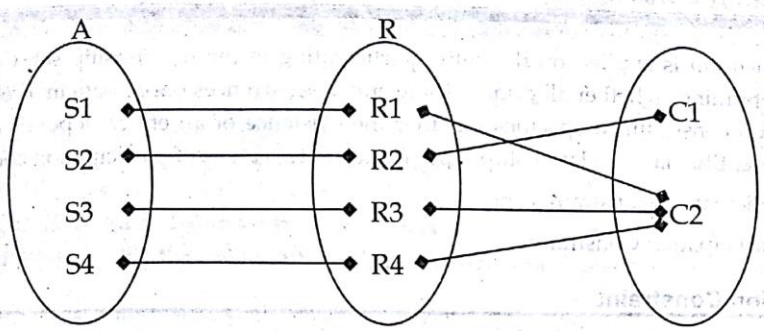


One to many or many to one

When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student there will be only one course.

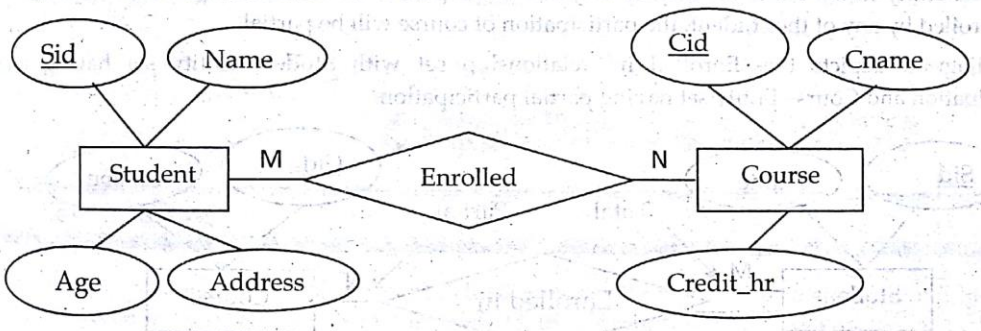


Using Sets, it can be represented as:

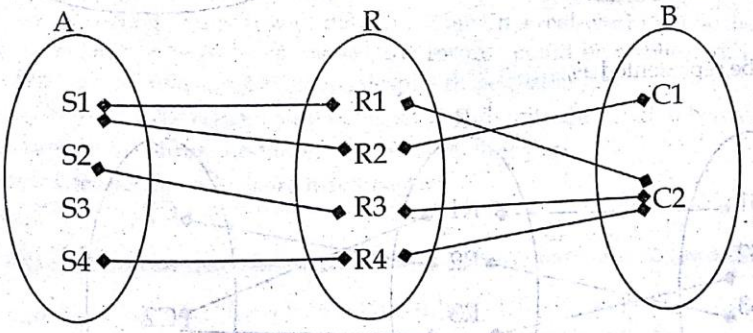


Many to many

When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



Using Sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C2 is enrolled by S1, S2 and S4. So it is many to many relationships.

3.6 Participation Constraints

Participation Constraint is applied on the entity participating in the relationship set. Constraint on ER model that determines whether all or only some entity occurrences participate in a relationship is called participation constraint. It specifies whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

- Total Participation Constraints and
- Partial Participation Constraints.

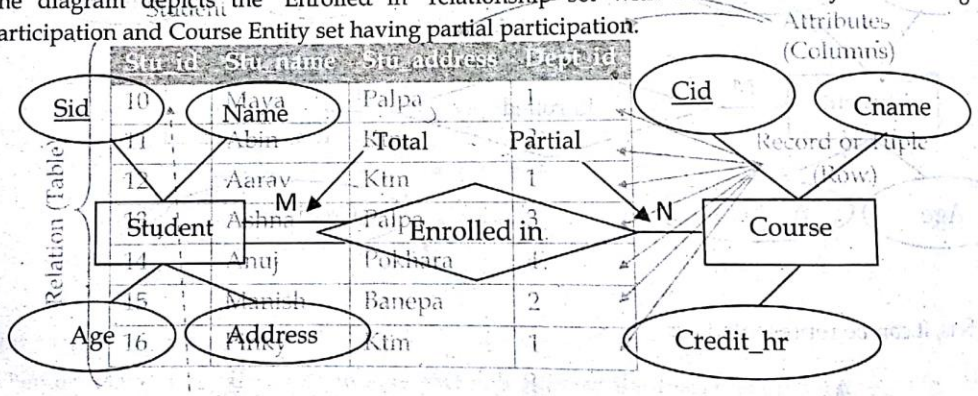
Total participation Constraint

Here each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

Partial Participation Constraint

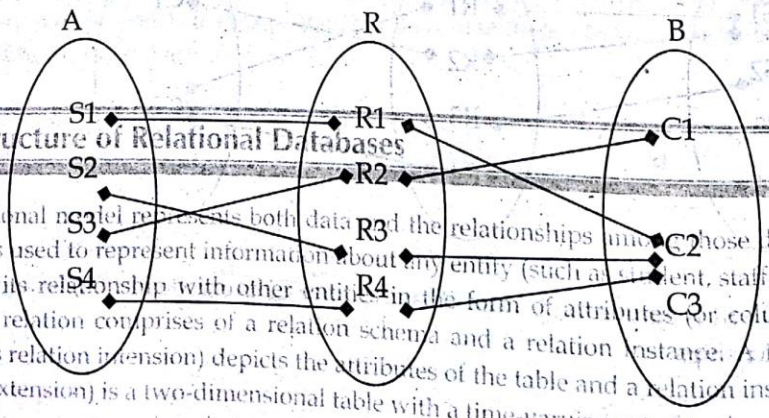
Here the entity in the entity set may or may not participate in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Alin	Ktm	3
12	Aarav	Ktm	1
13	Abhe	Palpa	3
14	Anuj	Pokhara	1
15	Manish	Banepa	2
16	Ankit	Ktm	1

Using set, it can be represented as,

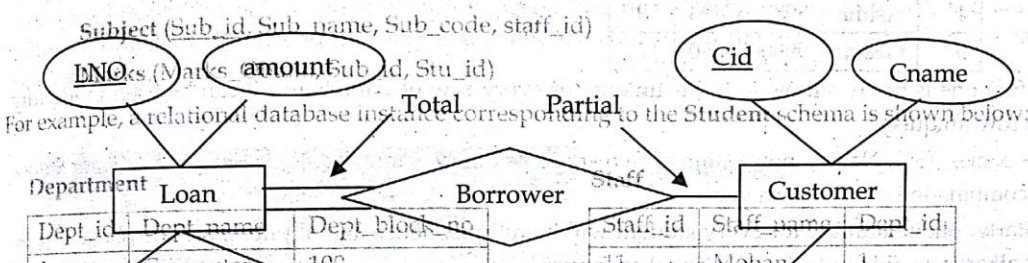


4.2 Structure of Relational Databases

The relational model represents both data and the relationships among those data using relations. A relation is used to represent information about any entity (such as student, staff, Department, subject etc.) and its relationship with other entities in the form of attributes (or columns) and tuples (or rows). A relation comprises of a relation schema and a relation instance. A relation schema (also termed as relation intension) depicts the attributes of the table and a relation instance (also termed as relation extension) is a two-dimensional table with a time-varying set of tuples.

Every student in **Student** Entity set is participating in relationship but there exists a course C3 which is not taking part in the relationship. Thus participation of student relation with relationship 'Enrolled in' is called total and participation of course relation with given relationship is called partial.

Similarly, let's take an another example, consider **Customer** and **Loan** entity sets in a banking system, and a relationship set **borrower** between them indicates that only some of the customers have Loan but every Loan should be associated with some customer. Therefore there is total participation of entity set Loan in the relationship set borrower but participation of entity set customer is partial in relationship set borrower. Here, Loan entity set cannot exist without Customer entity set but existence of Customer entity set is independent of Loan entity set.



Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	0
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pra	2
33	Madan	1
44	Kamala	3
55	Sandhya	4
66	Umesh	3

3.7 Keys in DBMS

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table. A Key can be a single attribute or a group of attributes, where the combination may act as a key.

Why we need a Key?

Here, are reasons for using Keys in the DBMS system.

- Keys help us to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that we can uniquely identify a table record despite these challenges.
- Allows us to establish a relationship between and identify the relation between tables
- Help us to enforce identity and integrity in the relationship.

There are different types of keys which are listed below:

- Super key
- Candidate key
- Primary key
- Composite key
- Foreign key
- Alternate Key
- Compound Key
- Surrogate Key

Super Key

A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity in the entity set. If K is a key attribute and any superset of K is also super key. A Super key may have additional attributes that are not needed for unique identification.

Let's try to understand about all the keys using a simple example

Sid	Name	Phone	age
1	Aarav	9876723452	17
2	Aarav	9991165674	19
3	Binod	7898756543	18
4	Ashna	8987867898	19
5	Geeta	9990080080	17

The first one is pretty simple as Sid is unique for every row of data, hence it can be used to identity each row uniquely.

Next comes, (Sid, Name), now name of two students can be same, but their Sid can't be same hence this combination can also be a key.

Similarly, phone number for every student will be unique, hence again; phone can also be a key. So they all are i.e. {Sid, (Sid, Name), phone} are super keys.

Candidate Keys

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key. Simply, a candidate key of an entity set is a minimal super key.

In our example, **Sid** and **phone** both are candidate keys for table Student.

Features of candidates key

- A candidate key is a minimal form of super key.
- There is no possible subset of candidate key that acts as a key attribute.
- A candidate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one column (attributes).

Primary key

Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.

A primary key is a candidate key that is chosen by the database designer as the principle means of uniquely identifying entities within an entity set. There may exist several candidate keys, one of the candidate keys is selected to be the primary key. Primary key must satisfy following two characteristics:

- It cannot be null
- It cannot be duplicate

For the table Student if database designer choose the candidate key **Sid** as key attribute then it acts as the primary key for their purpose.

Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called Composite key. But the attributes which together form the Composite key are not a key independently or individually. Simply, if a primary key contains more than one attribute then it is called composite key. For example, if database designer choose Sid as primary key then it not composite key but if database designer choose {Sid, phone} as primary key then it is called composite key.

Foreign Key

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables. A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce relationship between two relations (table).For example, if a student enrolls in course then course id (primary key of relation course) can be used as foreign key in student relation.

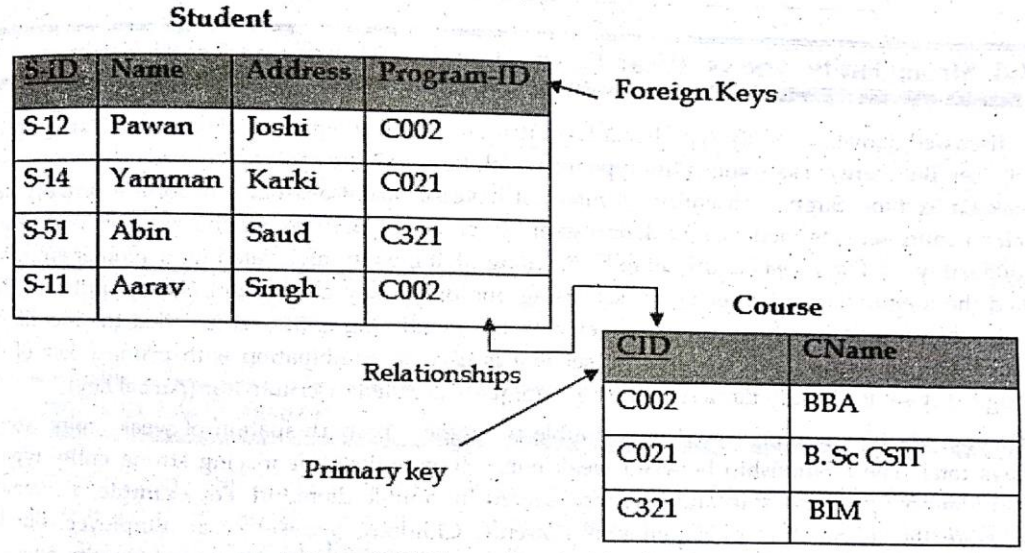


Figure 3.2 Primary key and foreign key

Alternate Key/Secondary key

Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

Compound Key

Like other keys Compound key is also used to uniquely recognize a record in relation. This can be an attribute or a set of attributes, but the attributes in relation cannot be use as independent keys. If we use them individually, we will not get any unique record.

Surrogate Key

An artificial key which aims to uniquely identify each record is called a surrogate key. These kinds of key are unique because they are created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key is usually an integer.

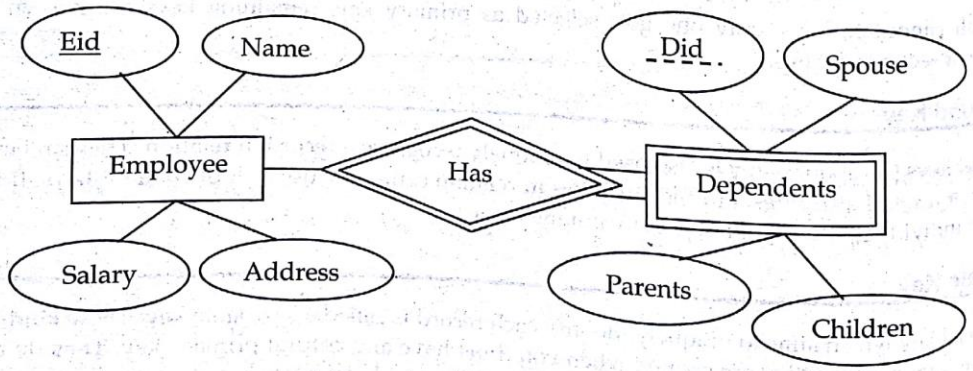
Difference between Primary key & foreign key

Primary Key	Foreign Key
It helps us to uniquely identify a record in the table.	It is a field in the table that is the primary key of another table.
Primary key never accepts NULL values.	A foreign key may accept multiple NULL values.
Primary key is a clustered index and data in the DBMS table are physically organized in the sequence of the clustered index.	A foreign key cannot automatically create an index, clustered or non-clustered. However, you can manually create an index on the foreign key.
We can have the single primary key in a table.	We can have multiple foreign keys in a table.

3.8. Strong Entity type vs. Weak Entity Type and Identifying Relationship

As discussed above, an entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists some entity type for which key attribute can't be defined. These are called Weak Entity type. Simply, an entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as a weak entity set. An entity set that has a primary key is termed as a strong entity set. For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set, using one of the key attribute of owner entity set. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship. An attribute of weak entity set that is used in combination with primary key of the strong entity set to identify the weak entity set uniquely is called discriminator (partial key).

A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond. **For example,** a company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependent.



3.9 Differentiate between weak entity and Strong Entity

Strong Entity Set	Weak Entity Set
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entities set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

3.10 Roles in E-R Diagrams

The function that an entity plays in a relationship is called its role. Roles are normally explicit and not specified. They are useful when the meaning of a relationship set needs clarification. For example, the entity sets of a relationship may not be distinct. The relationship works-for might be ordered pairs of employees (first is manager, second is worker).

In the E-R diagram, this can be shown by labeling the lines connecting entities to relationships.

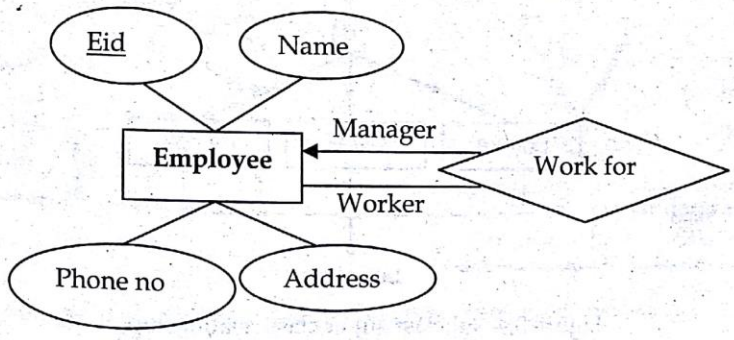


Figure 3.3 E-R diagram with role indicators

3.11 Extended E-R Model (EER Model)

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modeling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better. Hence, as part of the Enhanced ER Model, along with other improvements, three new concepts were added to the existing ER Model, they were:

- Subclasses and Super classes
- Specialization and Generalization
- Category or union type
- Aggregation

Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

Subclasses and super classes

Super class is an entity type that has a relationship with one or more subtypes. An entity cannot exist in database merely by being member of any super class. For example: Shape super class is having sub groups as Square, Circle, and Triangle.

Sub class is a group of entities with unique attributes. Sub class inherits properties and attributes from its super class. For example: Square, Circle and Triangle are the sub class of Shape super class.

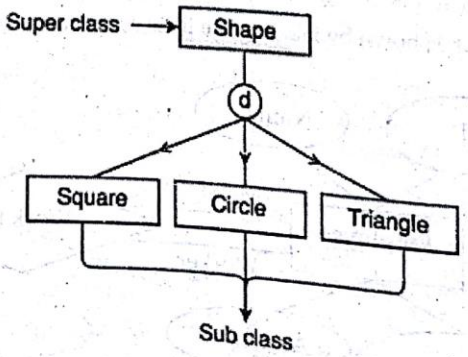


Figure 3.4 subclass super class relationship

Specialization and Generalization

Generalization

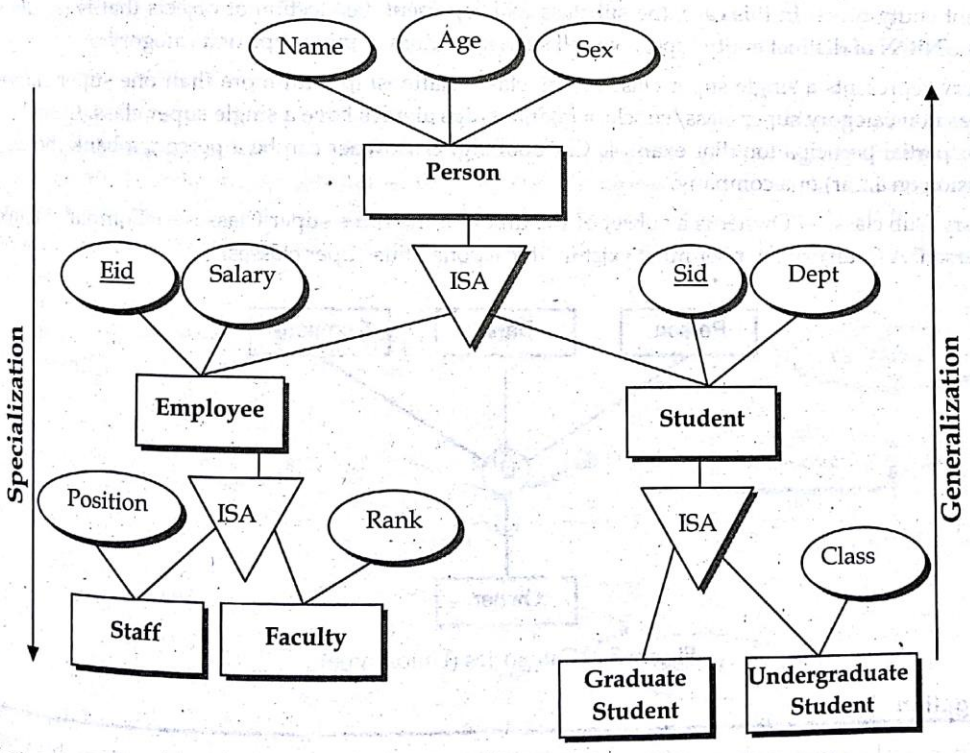
Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

It's more like Super class and Subclass system, but the only difference is the approach, which is bottom up. Hence, entities are combined to form a more generalized entity, in other words, sub-classes are combined to form a super-class.

Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible. It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member. It defines one or more sub class for the super class and also forms the super class/subclass relationship.

Let's take an example



Let's take separate examples of generalization and specializations as below;

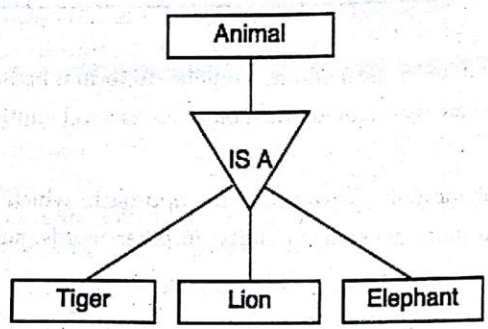


Figure 3.5 Generalization

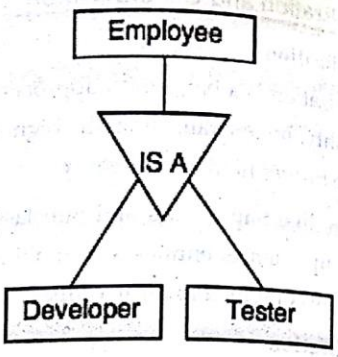


Figure 3.6 Specialization

In the above first example, Tiger, Lion, Elephant can all be generalized as Animals. Also in the above second example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

Category or Union

It is possible that single super class/subclass relationship has more than one super-class representing different entity types. In this case, the subclass will represent a collection of objects that is (a subset of) the UNION of distinct entity types; we call such a subclass a union type or a category.

Category represents a single super class or sub class relationship with more than one super classes whereas non-category super class/subclass relationships always have a single super class. It can be a total or partial participation. For example Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company.

Category (sub class) → Owner is a subset of the union of the three super Classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.

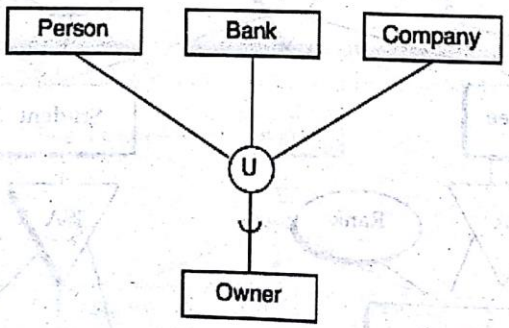
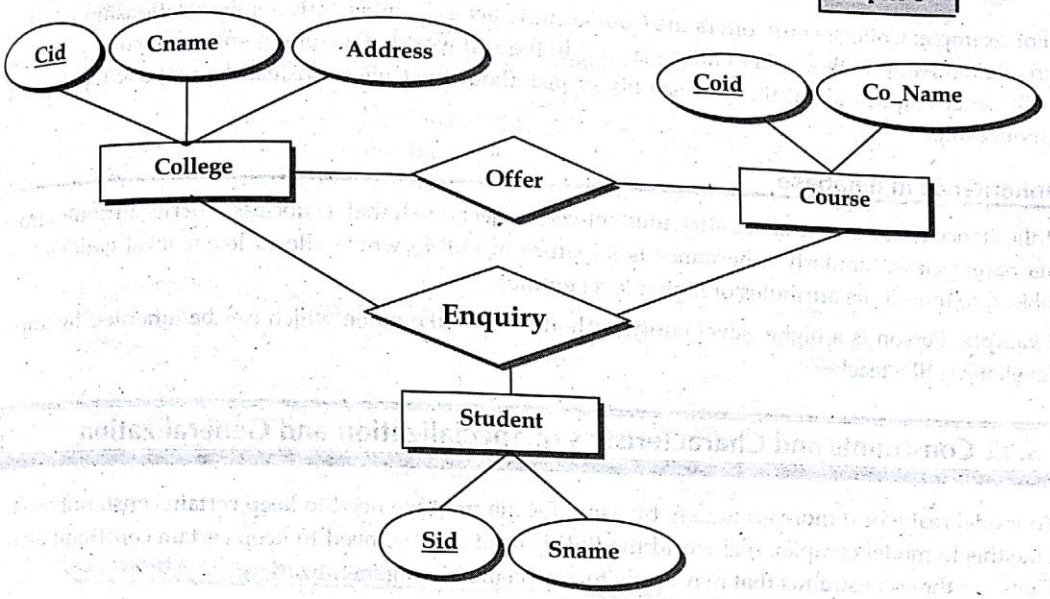


Figure 3.7: Categories (Union type)

Aggregation

The E-R model cannot express relationships among relationship sets and relationships between relationship sets and entity sets. Consider a DB with information about Colleges that offer courses and students enquiry colleges and courses offer by college. One alternative in this case is to define ternary relationship between {College, course, and Student} as in figure below.



Main problem with above ER diagram is that it has redundant relationships. Every {Employee, Project} combination that appears in uses relationship set also appears in works relationship set. The solution is to use aggregation.

Aggregation is a process that represents a relationship between a whole object and its component parts. Using aggregation we can express relationship among relationships. Aggregation shows 'has-a' or 'is-part-of' relationship between entities where one represents the 'whole' and other 'part'. It abstracts a relationship between objects and viewing the relationship as an object. It is a process when two entity and relationship with these entities is treated as a higher level single entity set.

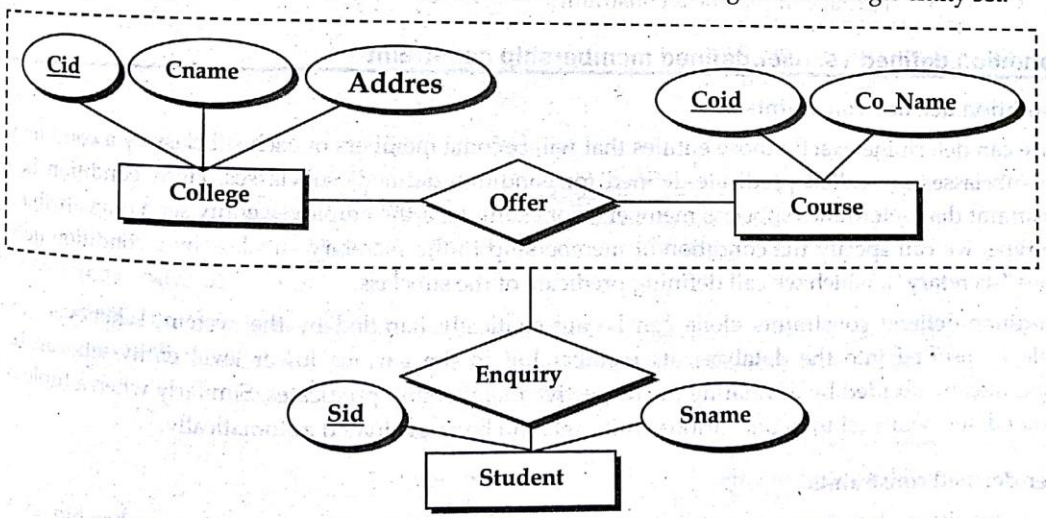


Figure 3.8 Aggregation

In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

For example: College entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity student. In the real world, if a student enquiry a college then he will never enquiry about the Course only or just about the College instead he will ask the enquiry about both.

Inheritance in database

Inheritance enables us to share attributes between objects such that a subclass inherits attributes from its parent class. Similarly Inheritance is a feature in DBMS which allows lower level entities (any object) to inherit the attributes of higher level entities.

Example: Person is a higher level entity with attributes like name which can be inherited by lower level object like teacher.

3.12 Constraints and Characteristics of Specialization and Generalization

To model real world more accurately by using ER diagram we need to keep certain constraints on it. Like this to model complex real world the EER is used and we need to keep certain constraint on it. There are three constraints that may apply to a specialization/generalization:

- **Membership constraints**
  - ✓ Condition defined membership constraint
  - ✓ User defined membership constraint
- **Disjoint constraints**
  - ✓ Disjoint constraint
  - ✓ Overlapping constraint
- **Completeness constraints**
  - ✓ Total completeness constraint
  - ✓ Partial completeness constraint

Condition defined vs. user defined membership constraint

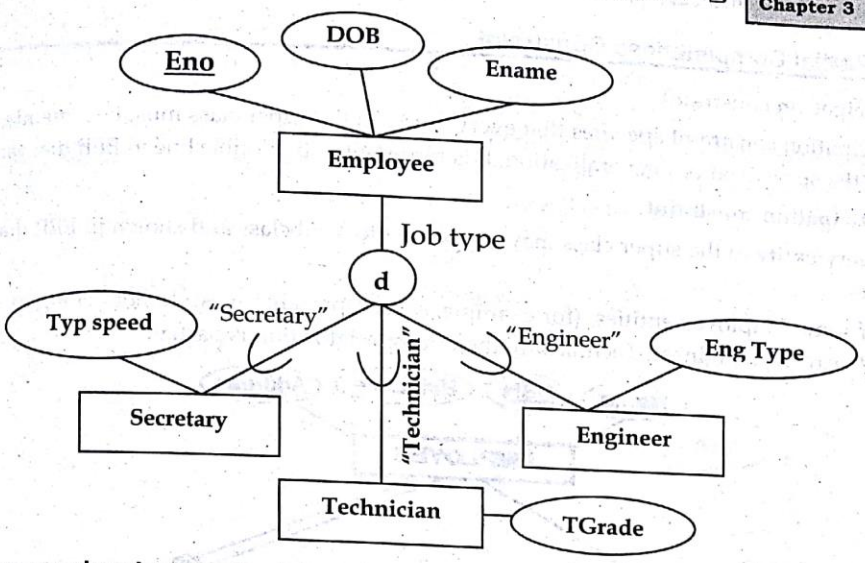
Condition defined constraints

If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses. Here, condition is a constraint that determines subclass members. For example, if the employee entity set has an attribute job-type, we can specify the condition of membership in the secretary subclass by a condition (job-type="secretary"), which we call defining predicate of the subclass.

Condition-defined constraints alone can be automatically handled by the system. Whenever any tuple is inserted into the database, its membership in the various lower level entity-sets can be automatically decided by evaluating the respective membership predicates. Similarly when a tuple is updated, its membership in the various entity sets can be re-evaluated automatically.

User-defined constraints

If no condition determines membership, the subclass is called user-defined. Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass. Membership in the subclass is specified individually for each entity in the super class by the user.



Disjoint vs. overlapping constraint

Another type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization.

Disjoint Constraint

It specifies that the subclasses of the specialization must be disjoint. Here an entity can be a member of at most one of the subclasses of the specialization and it is represented by d in EER diagram. Simply, if an entity can be a member of at most one of the subclasses of the specialization, then the subclasses are called disjoint.

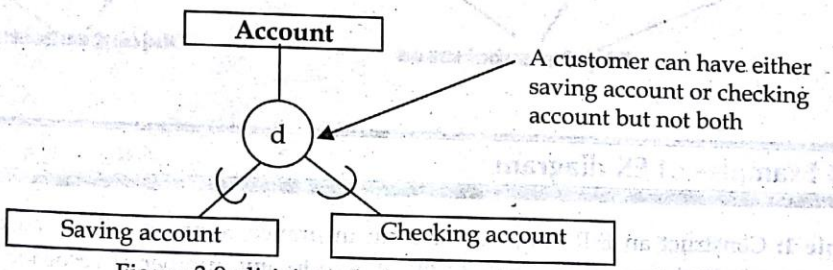


Figure 3.9: disjoint rule for account entity

Overlapping constraint

It specifies that the subclasses are not constrained to be disjoint, i.e., the same entity may be a member of more than one subclass of the specialization and it is represented by o in EER diagram.

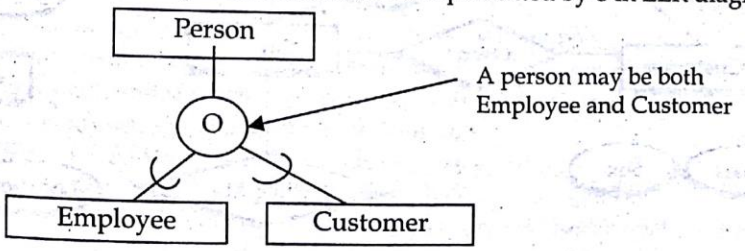


Figure 3.10: Overlap rule for person entity

Total vs. Partial Completeness Constraint

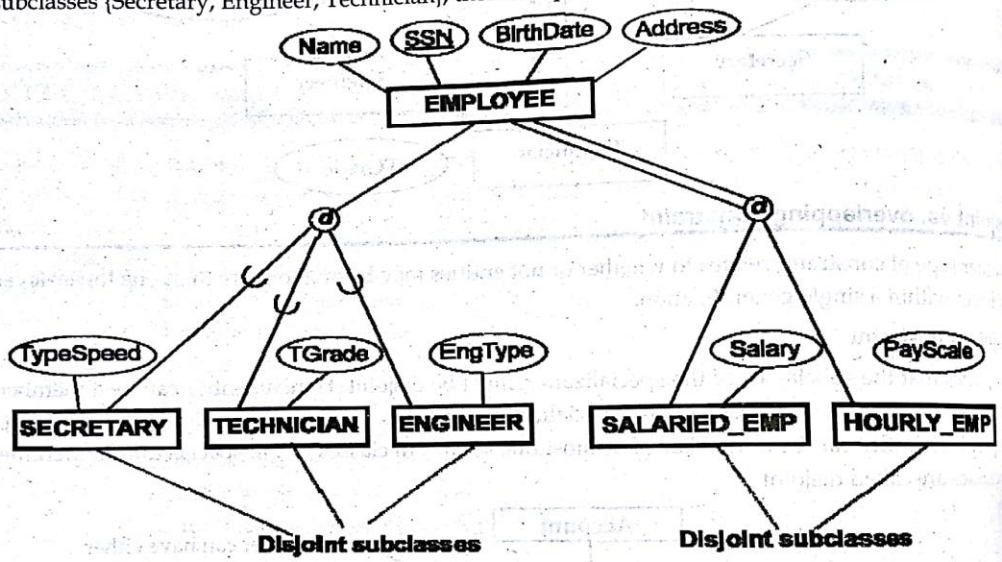
Total participation constraint

Total participation constraint specifies that every entity in the super class must be a member of some subclass in the specialization/generalization. It is represented by double line in EER diagram.

Partial participation constraint

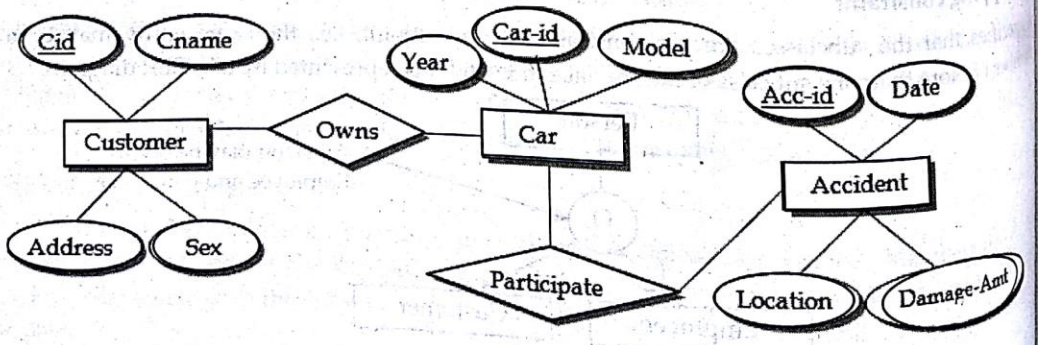
It allows every entity in the super class may not belong to a subclass and shown in EER diagrams by a single line.

Example: if some Employee entities, (for example, sales representatives) do not belong to any of the subclasses {Secretary, Engineer, Technician}, then the specialization is partial.

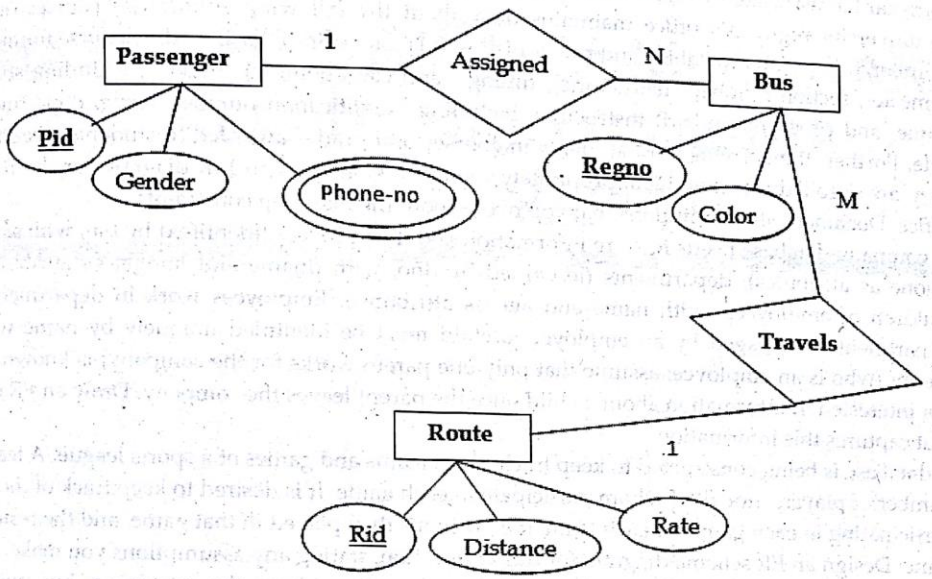


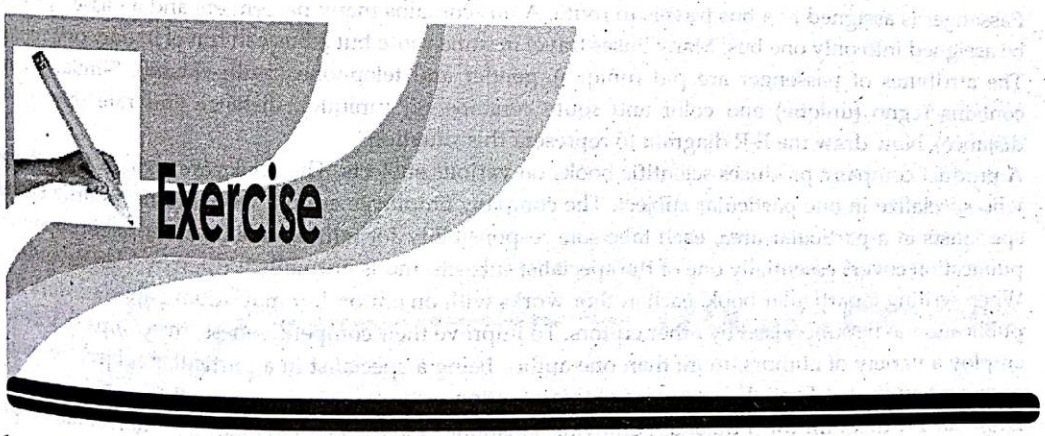
3.13 Examples of ER diagram

Example 1: Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Assume attributes of your own interest:



Example 2: Consider a bus ticketing system that records information about the passenger, bus and route. Passenger is assigned to a bus travels to route. A bus contains many passengers and a passenger can be assigned into only one bus. Many buses travel in same route but a bus can travel in only one route. The attributes of passenger are pid (unique), gender and telephone (multi-valued). Similarly bus contains regno (unique) and color and route contains rid (unique), distance and rate (based on distance). Now draw the E-R diagram to represent this situation.





1. What is the user of ER diagram? Explain various symbols used in ER diagram.
2. Explain the following terms briefly: attribute, domain, entity, relationship, entity set, relationship set, one-to-many relationship, many-to-many relationship, participation constraint, overlap constraint, covering constraint, weak entity set, aggregation, and role indicator.
3. What is an entity set? Explain the differences among an entity, and an entity set.
4. Explain the distinctions among the terms primary key, candidate key, and super key.
5. What is meant by a recursive relationship type? Give some examples of recursive relationship types.
6. Explain the difference between a weak and a strong entity set.

7. Define the concept of aggregation. Give two examples of where this concept is useful.
8. Explain the distinction between condition-defined and user-defined constraints. Which of these constraints can the system check automatically? Explain your answer.
9. Explain the distinction between disjoint and overlapping constraints and the distinction between total and partial constraints.
10. Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
11. A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.
12. A company database needs to store information about employees (identified by ssn, with salary and phone as attributes); departments (identified by dno, with dname and budget as attributes); and children of employees (with name and age as attributes). Employees work in departments; each department is managed by an employee; a child must be identified uniquely by name when the parent (who is an employee; assume that only one parent works for the company) is known. We are not interested in information about a child once the parent leaves the company. Draw an ER diagram that captures this information.
13. A database is being constructed to keep track of the teams and games of a sports league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Design an ER schema diagram for this application, stating any assumptions you make.
14. Consider a bus ticketing system that records information about the passenger, bus and route. Passenger is assigned to a bus travels to route. A bus contains many passengers and a passenger can be assigned into only one bus. Many buses travel in same route but a bus can travel in only one route. The attributes of passenger are pid (unique), gender and telephone (multi-valued). Similarly bus contains regno (unique) and color and route contains rid (unique), distance and rate (based on distance). Now draw the E-R diagram to represent this situation.
15. A product company produces scientific books on various subjects. The books are written by authors who specialize in one particular subject. The company employee editors who, not necessarily being specialists in a particular area, each take sole responsibility for editing one or more publications. A publication covers essentially one of the specialist subjects and is normally written by a single author. When writing a particular book, each author works with on editor, but may submit another work for publication to be supervised by other editors. To improve their competitiveness, the company tries to employ a variety of authors, more than one author being a specialist in a particular subject. Assume necessary attributes. Draw E-R diagram of this situation.
16. Draw an ER diagram for a database showing Hospital system. The Hospital maintains data about Affiliated Hospitals, type of Treatments facilities given at each hospital, and Patients.
17. Draw an ER diagram for database showing Bank. Each Bank can have multiple branches, and each branch can have multiple accounts and loans.
18. How supper key differ from Candidate key? Explain.
19. Describe types of relationship used in ER diagram with suitable example.
20. What is the user of ISA in EER diagram? Explain with suitable diagram.

❑❑❑

CHAPTER 4

THE RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

LEARNING OBJECTIVES

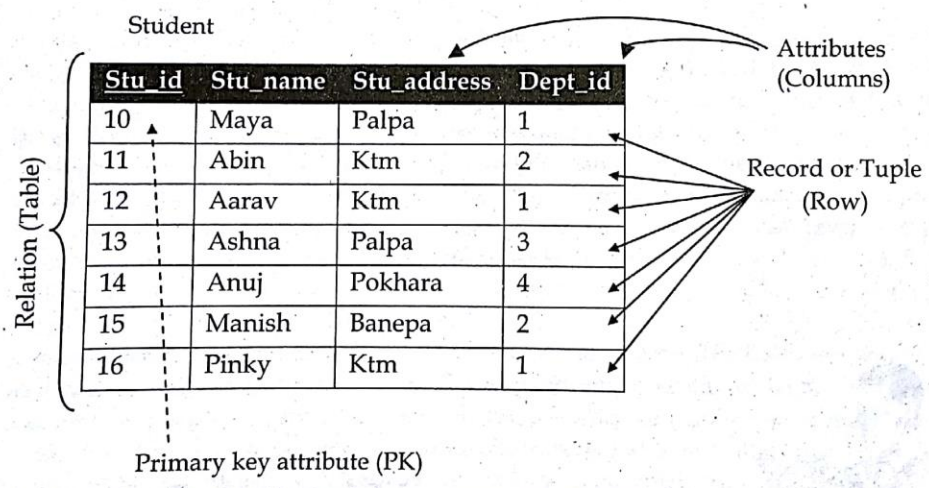
After comprehensive study of this Chapter, you will be able to:

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations, Transactions, and Dealing with Constraint Violations

4.1 Relational Model Concepts

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records. Relational data model represents the logical view of how data is stored in the relational databases. There exist some concepts related to this, which includes the following terms.

- **Table:** In relational data model, data is stored in the tables. The table consists of a number of rows and columns. Thus, table is used because it can represent the data in the simplest form possible making data retrieval very easy.
- **Attribute:** Any relation have definite properties that are called as attributes.
- **Tuple:** Rows of table represents the tuple which contains the data records.
- **Domain:** Domain is a set of values which is indivisible i.e. value for each attribute present in the table contains some specific domain in which the value needs to lie. For Example: The value of date of birth must be greater than zero. As, it cannot be negative. This is called domain of an attribute.
- **Relation:** A relation in relational data model represents the respective attributes and the correlation among them.



Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Abin	Ktm	2
12	Aarav	Ktm	1
13	Ashna	Palpa	3
14	Anuj	Pokhara	4
15	Manish	Banepa	2
16	Pinky	Ktm	1

4.2 Structure of Relational Databases

The relational model represents both data and the relationships among those data using relation. A relation is used to represent information about any entity (such as student, staff, Department, subject etc.) and its relationship with other entities in the form of attributes (or columns) and tuples (or rows). A relation comprises of a relation schema and a relation instance. A relation schema (also termed as relation intension) depicts the attributes of the table and a relation instance (also termed as relation extension) is a two-dimensional table with a time-varying set of tuples.

A relation schema consists of a relation name R and a set of attributes (or fields) A1, A2, ..., An. It is represented by R (A1, A2... An) and is used to describe a relation R. A set of relation schemas {R1, R2..... Rm} together with a set of integrity constraints in the database constitutes relational database schema. For example, the relational database schema CMS is a set of relation schemas, namely, Student, Staff, Subject, Marks, and College, which is shown below:

College Management System (CMS)

Department (Dept_id, Dept_name, Dept_block_no)

Student (Stu_id, Stu_name, Stu_address, Dept_id)

Staff (Staff_id, staff_name, Dept_id)

Subject (Sub_id, Sub_name, Sub_code, staff_id)

Marks (Marks_Obtain, Sub_id, Stu_id)

For example, a relational database instance corresponding to the **Student** schema is shown below:

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4
66	Umesh	3
77	Ramesh	1

Student

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Abin	Ktm	2
12	Aarav	Ktm	1
13	Ashna	Palpa	3
14	Anuj	Pokhara	4
15	Manish	Banepa	2
16	Pinky	Ktm	1

Marks

Marks_Obtain	Sub_id	Stu_id
60	20	10
55	21	12
58	22	10
49	20	13
61	23	14
67	22	13
60	24	16
55	26	15
33	25	11

Subject

Sub_id	Sub_name	Sub_code	Staff_id
20	DBMS	D-20	11
21	C++	C-21	22
22	NM	N-22	11
23	TOC	T-23	77
24	PHP	P-24	44
25	AI	A-25	33
26	ASP	A-26	55
27	CG	C-27	66
28	C-Prog	C-28	44

4.3 Relational Database Schemas

A relational database schema helps you to organize and understand the structure of a database. This is particularly useful when designing a new database, modifying an existing database to support more functionality, or building integration between databases.

A schema is the structure behind data organization. It is a visual representation of how different table relationships enable the schema's underlying mission business rules for which the database is created. In a schema diagram, all database tables are designated with unique columns and special features, e.g., primary keys, foreign keys or not null, etc. Formats and symbols for expression are universally understood, eliminating the possibility of confusion. The table relationships also are expressed via a parent table's primary key lines when joined with the child table's corresponding foreign keys. Schema diagrams have an important function because they force database developers to transpose ideas to paper. This provides an overview of the entire database, while facilitating future database administrator work.

Example: College Management System (CMS)

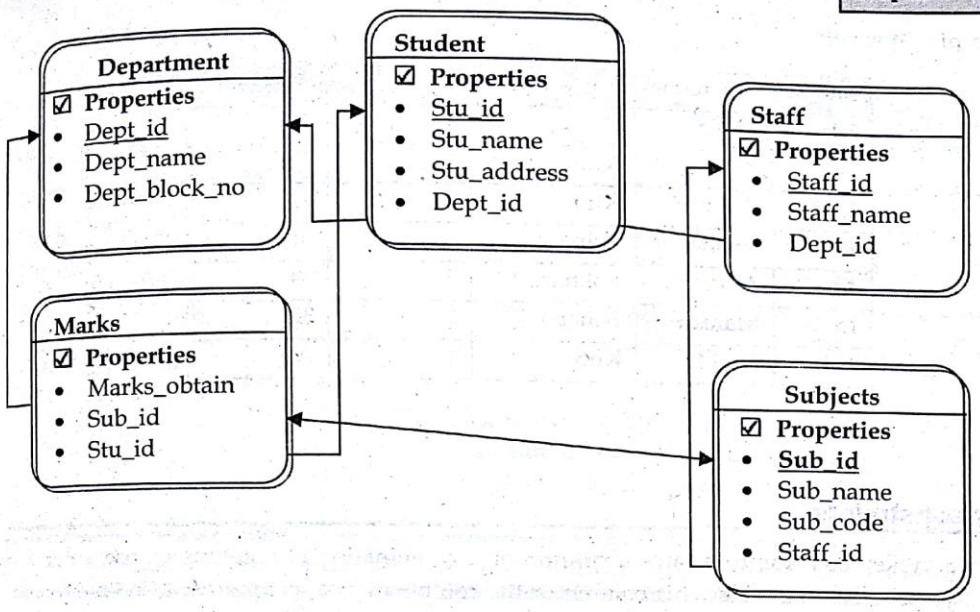
Department (Dept_id, Dept_name, Dept_block_no)

Student (Stu_id, Stu_name, Stu_address, Dept_id)

Staff (Staff_id, staff_name, Dept_id)

Subject (Sub_id, Sub_name, Sub_code, staff_id)

Marks (Marks_Obtain, Sub_id, Stu_id)



4.4 Relational Model Constraints

Relational Integrity constraints are referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents. There are many types of integrity constraints. A constraint on the Relational database management system is mostly divided into three main categories are:

- Domain constraints
- Key constraints
- Referential integrity constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type. A table is DBMS is a set of rows and columns that contain data. Columns in table have a unique name, often referred as attributes in DBMS. A domain is a unique set of values permitted for an attribute in a table. For example, a domain of month-of-year can accept January, February....December as possible values, a domain of integers can accept whole numbers that are negative, positive and zero.

Domain constraints can be defined as the definition of a valid set of values for an attribute. The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example: Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	22
11	Abin	Ktm	2	32
12	Aarav	Ktm	1	43
13	Ashna	Palpa	3	55
14	Anuj	Pokhara	4	34
15	Manish	Banepa	2	32
16	NULL	Ktm	1	AA

Not allowed because age is integer.

Key constraints

A primary key constraint declares a column or a combination of columns whose values uniquely identify each row in a table. This column or the combination of columns is also known as primary key of the table. If we insert or update a row that would cause duplicate primary key, database will issue an error message. In other words, a primary key constraint helps enforce the integrity of data automatically.

The Key Constraint specifies that there should be such an attribute in a table, which can be used to fetch data for any tuple. The Key attribute should never be NULL or same for two different rows of data.

Example: In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
1	Gmail	Active

Key constraint violets

Referential integrity constraints

A referential integrity constraint is specified between two tables. In the referential integrity constraints, if a foreign key in table 1 refers to the primary key of table 2, then every value of the foreign key in table 1 must be null or be available in table 2.

Student

Stu_id	Stu_name	Dept_id
10	Maya	1
11	Abin	2
12	Aarav	1
13	Ashna	3
14	Anuj	4
15	Manish	2
16	Pinky	6

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Not allowed as Dept_id 6 is not defined as Primary key of table Department, Dept_no is a foreign key defined

4.5 Update Operations, Transactions, and Dealing with Constraint Violations

There are three basic operations that can change the states of relations in the database. Insert, delete, and update (or modify). The insert new data, delete old data, or modify existing data records. Insert is used to insert one or more new tuples in a relation, delete is used to delete tuples and update is used to change the values of some attributes in existing tuples. Whenever these operations are applied, the integrity constraints specified on the relational database schema should not be violated. In this section we discuss the types of constraints that may be violated by each of these operations and the types of actions that may be taken if an operation causes a violation.

The insert operation

The insert operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation R. Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

Example

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4
66	Umesh	3
77	Ramesh	1

Here in the staff table we cannot insert new record {6, "English", 454.50} because value of attribute Dept_block_no is in floating point which violets the domain constraint.

Similarly, we cannot insert a new record {4, "English", 700} to department table because the key value 4 already exist in the Department table.

Also we cannot insert the new record {12, "Lalit", 6} to Staff table because their reference was not present at Department table.

The delete operation

The delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

From the above tables we cannot delete the record {1, "Computer", 100} from Department table because their reference is saved to their child table Staff so by deleting this it violets the foreign key constraint.

The update operation

The update or modify operation is used to change the values of one or more attributes in a tuple of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple to be modified. Here are similar issues as with Insert/Delete operations.

Examples

Here we cannot change the record {1, "Computer", 100} to {9,"Computer", 100} because their reference saved to their child table "Staff".

The Transaction Concept

A database application program running against a relational database typically executes one or more transactions. A transaction is an executing program that includes some database operations, such as reading from the database, or applying instructions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema. A single transaction may involve any number of retrieval operations and any number of update operations. These retrievals and updates will together from an atomic unit of work against the database, For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

A large number of commercial applications running against relational database in online transaction processing (OLTP) systems are executing transactions at rates that reach several hundred per second. Transaction processing concepts and concurrent execution of transactions and recovery from failures will be discussed in next chapters.

4.6 Advantages of using Relational model

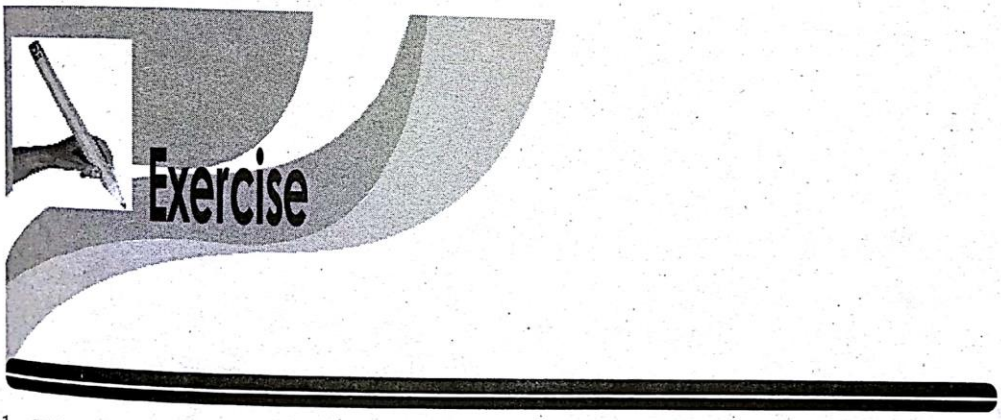
- **Simplicity:** A relational data model is simpler than the hierarchical and network model.
- **Structural Independence:** The relational database is only concerned with data and not with a structure. This can improve the performance of the model.

- **Easy to use:** The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability:** It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence:** The structure of a database can be changed without having to change any application.
- **Scalable:** Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

4.7 Disadvantages of using Relational model

There are a lot of advantages of relational model but some disadvantages of relational model are listed below:

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.



1. Differentiate between network and hierarchical model.
2. What do you understand by foreign key and referential integrity constraint?
3. Is it essential to have more than one table to define foreign key? If no, then explain using suitable examples.
4. Define RDBMS, what are the properties of the valid relation?
5. List and explain various database models in details with one example of each.
6. Explain why navigation is simpler in the relational data model than in any other data model with examples.
7. What do you mean by domain? Give example.
8. What is integrity rule in relational model?
9. Explain entity integrity constraint.

10. Discuss domain integrity constraint.
11. What do you mean by referential integrity constraint with suitable example?
12. Define the term 'data model'.
13. Explain different types of data model.
14. Define relation.
15. Differentiate between DBMS and RDBMS.
16. What are the advantages of using relational model?
17. Discuss the disadvantages of using relational model.
18. What are the three basic operations which changes the states of relations in the database?
19. Explain the insert, delete and update operation in relational database.
20. Discuss the transaction concept.

❑❑❑

CHAPTER 5

THE RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Unary Relational Operations: SELECT and PROJECT
- Relational Algebra Operations from Set Theory
- Binary Relational Operations: JOIN and DIVISION
- Additional Relational Operations
- The Tuple Relational Calculus; the Domain Relational Calculus

5.1. Query Languages

A query language is a language in which a user requests information from the database. Query languages can be categorized as either **procedural** or **non-procedural**. In a procedural language the user instructs the system to perform a sequence of operations on the database to compute the desired result. **Example**: Relational algebra.

In a non-procedural language, the user describes the desired information desired without giving a specific procedure for obtaining that information. **Example**: tuple relational calculus, domain relational calculus, SQL etc.

5.2. Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The main operations of the relational algebra are listed below;

a. **Fundamental operations**
   - Select operation
   - Project operation
   - Union operation
   - Set difference operation
   - Cartesian product operation
   - Rename operation

b. **Additional operations**
   - Set-intersection operation
   - Natural-join operation
   - Division operation
   - Assignment operation

c. **Extended Relational Algebra Operations**
   - Generalized projection
   - Aggregate functions
   - Outer join
   - Null values

Also relational algebra can be divided into various groups according to binary, unary and set operations listed below;

d. **Unary Relational Operations**
   - Select (symbol: σ)
   - Project (symbol: π)
   - Rename (symbol: ρ)

e. **Relational algebra operations from set theory**
- Union (U)
- Intersection (∩),
- Difference (-)
- Cartesian product (x)

f. **Binary Relational Operations**
- Join
- Division (÷)

g. **Additional Relational Operation**
- Assignment operation
- Generalized projection
- Aggregate functions
- Null values

5.3 Unary Relational Operations

The relational algebra operations that are uses single relation (table) are called unary relational operations.

Select (σ)

The Select operation is used for selecting a subset of the tuples according to a given selection condition. Sigma (σ) symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

Syntax: $\sigma_{<\text{selection condition}>}(R)$

Where, R stands for relation which is the name of the table

Comparison operators (<, >, ≤, ≥, =, ≠) can be used to specify conditions required for selection tuples from a relation. Furthermore, logical operators AND (Λ), OR (v) and NOT (¬) can be used to combine two or more conditions.

Example 1: Let's take a student relation.

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	22
11	Abin	Ktm	2	17
12	Aarav	Ktm	1	21
13	Ashna	Palpa	3	45
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	23
16	Pinky	Ktm	1	16

1. Find records of all students of address 'Palpa'.

   $\sigma_{\text{Stu\_address} = \text{"Palpa"}}$ (Student)

2. Find all students whose address "Ktm" and of department id 1

   $\sigma_{\text{Stu\_address} = \text{"Ktm"} \Lambda \text{Dept\_id} = 1}$ (Student)

3. Find all students of age greater than 20 or of address 'Ktm'

   $\sigma_{\text{Stu\_address} = \text{"Ktm"} v \text{Age} \geq 1}$ (Student)

Projection (π)

The Project operation used to selects certain columns from the table and discards the other columns. If we are interested in only certain attributes of a relation, we use the Project operation to project the relation over these attributes only.

Syntax: $\Pi_{\text{<attribute-list>}}$ **(R)**

Example: We must use above student table,

1. Display id and name of the student

   $\Pi_{\text{Stu\_id, Stu\_name}}$ (Student)

2. Display name and age of students

   $\Pi_{\text{Stu\_name, Age}}$ (Student)

Combining Selection and Projection Operations

The selection and projection operators can be combined to perform projection with selection operation. Here we use selection operation within projection operation. At first inner operation (selection operation) extract given specified rows and then outer operation i.e. projection operation extract only specified columns from selected rows.

Syntax: $\Pi_{\text{<attribute-list>}}(\sigma_{\text{<selection condition>}}(R))$

Example: We also use above Student table,

1. Find name age address of students whose age less than or equal to 25

   $\Pi_{\text{Stu\_name, Stu\_address}}(\sigma_{\text{age} \leq 25}(\text{Student}))$

2. Find name of the students whose age greater than 20 and of address "Palpa"

   $\Pi_{\text{Stu\_name}}(\sigma_{\text{age} > 25 \wedge \text{Stu\_address="Palpa"}}(\text{Student}))$

5.4 Relational algebra operations from set theory

All of these operations except Cartesian product set operation take two input relations, which must be union-compatible:

- Same number of fields
- Corresponding fields have the same type

Union Operation (U)

Consider the following relations R and S. The union of relations R and S is denoted by R U S and it is the set of tuples that are either in R or in S or in both. It returns the union of two compatible relations. For a union operation to be legal, we require that invoked relations must have the same number of attributes and corresponding attributes have same type. Formally, we can define union as below:

$R \cup S = \{t \mid t \in R \text{ or } t \in S\}$ Where, R and S are any two relations and t is a tuple.

Example: Let's take following two tables namely morning shift Employee as "Memplcyee" and Day shift employees as "Demplcyee".

Memployee

eid	name	salary
e1	Rajan	34000
e2	Aarab	45000
e3	Abin	55000
e4	Ashna	24000

Demployee

eid	name	salary
e1	Rajan	34000
e5	Umesh	78000
e8	Anisha	55000
e4	Ashna	33000

Memployee U Demployee

eid	name	Salary
e1	Rajan	34000
e2	Aarav	45000
e3	Abin	55000
e4	Ashna	24000
e4	Ashna	33000
e5	Umesh	78000
e8	Anisha	55000

Intersection (∩)

Set intersection is denoted by symbol ∩ and it returns a relation that contains tuples that are in both of its argument relations. The intersection of relations R and S is denoted by R ∩ S and it is defined as:

$$R \cap S = \{t \mid t \in R \text{ and } t \in S\}$$

Like a union operation to be legal, in intersection we also require that invoked relations must have the same number of attributes and corresponding attributes have same type.

Memployee ∩ Demployee

eid	name	Salary
e1	Rajan	34000

Difference (–)

Set difference is denoted by the minus sign (-). It finds tuples that are in one relation, but not in another. Thus results in a relation containing tuples that are in R but not in S.

The set difference of two relations R and S is denoted by R – S and is defined as,

$$R - S = \{t \mid t \in R \text{ and } t \notin S\}$$

Like a union and intersection operation to be legal, in set difference we also require that invoked relations must have the same number of attributes and corresponding attributes have same type.

Memployee - Demployee

eid	name	Salary
e2	Aarav	45000
e3	Abin	55000
e4	Ashna	24000

Cartesian Product (x)

This type of operation is helpful to merge columns from two relations. The Cartesian product operation does not require relations to union-compatible i.e. the involved relations may have different schemas. The Cartesian product of two relations R and S is denoted by R x S, is the set of all possible combinations of tuples of the two relations. Formally, we can define Cartesian product as below:

$$R \times S = \{t\ q \mid t \in R \text{ and } q \in S\}$$

Where, R and S are any two relations and t and q are tuples of relation R and S respectively.

Example

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4

Department x Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	22	Pratima	2
1	Computer	100	33	Madan	1
1	Computer	100	44	Kamala	3
1	Computer	100	55	Sandhya	4
2	Mathematics	200	11	Mohan	1
2	Mathematics	200	22	Pratima	2
2	Mathematics	200	33	Madan	1
2	Mathematics	200	44	Kamala	3
2	Mathematics	200	55	Sandhya	4
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1
3	Economics	300	44	Kamala	3
3	Economics	300	55	Sandhya	4
4	Account	400	11	Mohan	1
4	Account	400	22	Pratima	2
4	Account	400	33	Madan	1
4	Account	400	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics	500	11	Mohan	1
5	Physics	500	22	Pratima	2
5	Physics	500	33	Madan	1
5	Physics	500	44	Kamala	3
5	Physics	500	55	Sandhya	4

5.5 Binary Relational Operations

These operations are used to perform operations into multiple tables. Set relational algebra operations are also called binary relational operations but here explain join operation and division operations as binary relational operations.

Join operation

Join operation is essentially a Cartesian product followed by a selection criterion. Join operation denoted by ⋈. Join operation also allows joining variously related tuples from different relations.

Types of Join operations

Various forms of join operation are listed below;

a. **Inner Joins**
   - Theta join
   - Equi join
   - Natural join

b. **Outer join**
   - Left Outer Join
   - Right Outer Join
   - Full Outer Join

c. **Inner Join**

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta join

The general case of join operation is called a Theta join. It is denoted by symbol θ. The theta join operation is an extension to the natural join operation that allows us to specify the join condition. The theta condition consists one of the comparison operators {=, <, <=, >, >=, < >}.

Syntax: $A \bowtie_\theta B$ where, A and B are any two relations and θ is a join condition.

Example:

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4

Department ⋈ D.Dept_id > S.Dept_id (Staff)

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
2	Mathematics	200	11	Mohan	1
2	Mathematics	200	33	Madan	1
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1
4	Account	400	11	Mohan	1
4	Account	400	22	Pratima	2
4	Account	400	33	Madan	1
4	Account	400	44	Kamala	3
5	Physics	500	11	Mohan	1
5	Physics	500	22	Pratima	2
5	Physics	500	33	Madan	1
5	Physics	500	44	Kamala	3
5	Physics	500	55	Sandhya	4

Equi Join

When join condition is '=' i.e. θ is =, the operation is called an equijoin. Thus the theta join with equality condition is called **Equi join** operation.

Syntax: A ⋈ = B where, A and B are any two relations and = is a join condition.

Equi join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

Example: Let's take above two relations Department and Staff

Department ⋈ D.Dept_id = S.Dept_id (Staff)

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4

Natural Join (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same. The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the join

symbol ⋈. The natural join operation performs the Cartesian product of given relations together with remove the duplicate attributes. The natural join thus performs a join by equating the attributes with the same name and then eliminates the replicated attributes.

In brief the result of the natural join of two relations R and S is the set of all combinations of tuples in R and S that are equal on their common attribute names.

Example: Let's take above two relations Department and Staff with Dept_id act as primary key for Department relation and it acts as foreign key for Staff relation.

Department ⋈ Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name
1	Computer	100	11	Mohan
1	Computer	100	33	Madan
2	Mathematics	200	22	Pratima
3	Economics	300	44	Kamala
4	Account	400	55	Sandhya

Outer Join

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria. Thus the Outer join operation is an extension of the **Join** operation to deal with missing information. There are three types of outer joins:

Left Outer Join (⟕)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with NULL values.

Example: Let's take following two relations Department and staff

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4
23	Ramesh	7

Department ⟕ Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics	500	NULL	NULL	NULL

Right Outer Join (⟖)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with NULL values.

Example: Let's take above two relations Department and staff

Department ⟖ Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
NULL	NULL	NULL	23	Ramesh	7

Full Outer Join (⟗)

It includes all tuples in the left hand relation and from the right hand relation. In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

Example: Let's take above two relations Department and staff

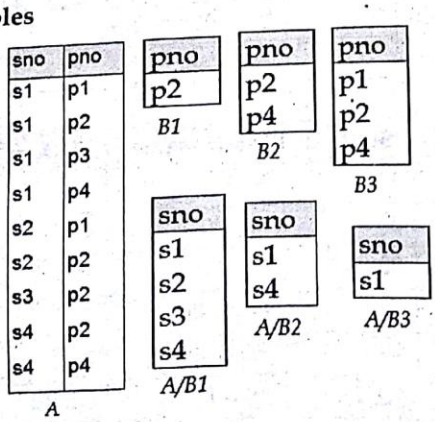
Department ⟗ Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics	500	NULL	NULL	NULL
NULL	NULL	NULL	23	Ramesh	7

Division (÷)

It is denoted by symbol ÷ and is suited to queries that include the phrase "for all". It takes two relations and builds another relation, consisting of values of an attribute of one relation that match all the values in the other relation.

Examples



sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

5.6 Additional Relational Operation

Additional operations are defined in terms of the fundamental operations. They do not add power to the algebra, but are useful to simplify common queries.

The Assignment Operation

The assignment operation (←) provides a convenient way to express complex queries.

Write query as a sequential program consisting of

- A series of assignments
- Followed by an expression whose value is displayed as a result of the query Assignment must always be made to a temporary relation variable.

The assignment operation (←) provides a convenient way to express complex queries. It helps human beings with writing out complex relational expressions in steps so that they can be more easily understood. The assignment operation works like assignment in a programming language.

Syntax: Variable ← E, Where E is any relational algebra expression.

Example: Let's take two relations borrower and depositor in bank database then write RA expression to the following query "Find all customer who taken loan from bank as well as he/she has bank account" by using assignment operation.

Temp1←Πcustomer_name (borrower)

Temp2←Πcustomer_name (depositor)

result← Temp1∩ temp2

Generalized projection

It extends the projection operation by allowing arithmetic functions to be used in the projection list. Generalized projection is enhanced version of project operation. It allows us to write aromatic operations containing attribute names and constants in projection list. General form of generalized projection is given below:

$$\prod_{F1, F2, F3, \ldots, Fn} (E)$$

Where, E is a relational algebra expression and Fi (i=1, 2....n) is an attribute or arithmetic expression containing attributes and constants.

Example: Let's take an employee relation

Employee

eid	name	Age	Salary
e1	Rajan	33	34000
e2	Aarav	17	45000
e3	Abin	22	55000
e4	Ashna	19	24000
e5	Manmohan	30	33000
e6	Deepak	26	78000
e7	Samjana	28	55000

1. Find name and salary of all employees by increasing their salary by 15%

$$\prod_{name,\ salary=salary+salary*0.15} (Employee)$$

2. Increase the salary of all employees whose age greater than 20 by 5%.

$$\prod_{eid,\ name,\ age,\ salary=salary+salary*0.05} (\sigma_{age>20} (Employee))$$

5.6.3 Aggregate functions

Aggregate functions are algebraic functions that take a collection of values as input and return a single value as a result. It is denoted by symbol $\mathcal{G}$ read it as "calligraphic G". General form of aggregate operation in relational algebra is;

$$G1, G2 \ldots Gn\ \mathcal{G}_{F1(A1),\ F2(A2) \ldots Fn(An)} (E)$$

Where, E is any relational-algebra expression

G1, G2 ..., Gn is a list of attributes on which to group (can be empty)

Each Fi is an aggregate function and each Ai is an attribute name.

All aggregate functions are deterministic. This means aggregate functions return the same value any time that they are called by using a specific set of input values. We cannot express aggregate functions by using basic relational algebra expressions. There are five aggregate functions that are included with most relational database systems. These operations are:

- SUM: sum of values
- AVG: average value
- MIN: minimum value
- MAX: maximum value
- COUNT: number of values

Example: Let's take an employee relation

Employee

Eid	Name	Address	Age	Salary
e1	Rajan	Ktm	33	34000
e2	Aarav	Pokhara	17	45000
e3	Abin	Palpa	22	55000
e4	Ashna	Ktm	19	24000
e5	Manmohan	Pokhara	30	33000

3. Find total number of employees

$$\mathcal{G}_{COUNT\ (Eid)} (Employee)$$

4. Find average age of employees of address 'Ktm'

$$\mathcal{G}_{AVG\ (Age)} (\sigma_{Address="Ktm"} (Employee))$$

5. Find minimum and maximum age of the student

$$\mathcal{G}_{MIN\ (Age),\ MAX\ (Age)} (Employee)$$

6. Find average salary of employee in each address level

$$Address\ \mathcal{G}_{AVG\ (Salary)} (Employee)$$

7. Find total salary of the employees

$$\mathcal{G}_{SUM\ (Salary)} (Employee)$$

NULL values

Null represents value for an attribute that is currently unknown or not applicable for tuple. It deals with incomplete or exceptional data. It represents the absence of a value and is not the same as zero or spaces, which are values. The result of any arithmetic expression involving null is null. Aggregate functions simply ignore null values. For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same. Comparisons with null values return the special truth value unknown. Two-valued logic is straight forward and we are well known about it. We can use null values with different RA operations described as above. Some of operations with null values are listed below;

- **Union**: A ∪ NULL = A
- **Intersection**: A ∩ NULL =NULL
- **Set difference**: A - NULL = A
- **Joins**: Natural Join: A X NULL = NULL
- **Left Join**: A ⟕ NULL = NULL
- **Right join**: A ⟖ NULL = A

5.7 Database Manipulation

Until now we only did the extraction of information from the database. In this section we will perform some modification on the database. We will namely use three types of operations for the modification of the database; they are **insertion**, **deletion** and **modification**. All these operations can be expressed using the assignment operator.

Insertion operation

To insert data into a relation, we specify a tuple to be inserted. Here we use assignment operation and union operation in combine form.

Syntax: R←R U E

Where R is a relation and E is a relational algebra expression.

Example: suppose we have a relation employee

Employee (Name, Salary, Address)

Suppose we wish to insert an employee "Bhupi" of salary 50,000 and live in Kathmandu then we write,

Employee←Employee U {"Bhupi", 50000, Kathmandu}

Deletion operation

We can remove the selected tuples from the database. We cannot delete values of only particular attributes. Here we user assignment operator and set difference operator to perform deletion operation in RA.

Syntax: R←R-E

Where R is a relation and E is a relational algebra expression.

Example: Delete employee 'Anju' from Employee relation

Employee

e-id	e-name	Salary
11	Bhupi	3000
13	Anju	4000
43	Manju	5000
54	Nisha	6000
33	Anju	3400

Employee ← Employee - $\sigma_{\text{e-name="Anju"}}$ (Employee)

Result: Employee

e-id	e-name	Salary
11	Bhupi	3000
43	Manju	5000
54	Nisha	6000

Updating Operation

In some situation we may wish to change a value in tuple without changing all values in the tuple. We can use the generalized-projection operator to do this task.

Syntax: $\Pi_{<A1, A2...An>}$ (Relation)

Where {A1, A2... An} are attributes.

Example: All employees working in department "Computer" has increased their salary by 15%.

Employee

e-id	e-name	Department	Salary
11	Bhupi	Computer	3000
13	Anju	Math	4000
43	Manju	Physics	5000
54	Nisha	Computer	6000
33	Anisha	Math	6400

Employee ← ($\Pi_{\text{e-id, e-name, department, salary + salary*0.15}}$ ($\sigma_{\text{department="Computer"}}$ (Employee)) U $\Pi_{\text{e-id, e-name, department, salary}}$ ($\sigma_{\text{department} \neq \text{"Computer"}}$ (Employee)))

Result: Employee

e-id	e-name	department	Salary
11	Bhupi	Computer	3450
13	Anju	Math	4000
43	Manju	Physics	5000
54	Nisha	Computer	6900
33	Anisha	Math	6400

5.8 Summary of all Operations of RA

Operation	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	Minus (-) Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes an equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join( )	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join()	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join()	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Relational Algebra Examples

Example 1: Consider the following relational schema of a hospital where primary keys are underlined.

DOCTOR (name, age, address)
WORKS (name, deptno)
DEPARTMENT (deptno, floor, room)

Write down the relational algebra for the following

1. List the room of the doctors named 'KAROL'.

   $\Pi_{room}(\sigma_{name="KAROL"}(DEPARTMENT \bowtie WORKS \bowtie DOCTOR))$

2. Count the number of doctors working in top floor.

   $\mathcal{G}_{COUNT(name)}(\sigma_{Floor="top"})(DOCTORS \bowtie WORKS \bowtie DEPARTMENT)$

3. Delete all the department of ground floor.

   DEPARTMENT ←DEPARTMENT – ($\sigma_{Floor="Ground"}$) (DEPARTMENT))

4. Insert a new department in the hospital.

   → There is no hospital relation

5. Increase the age of doctor named 'KSROL' by 1.

   $\Pi_{name,\ age=age+1,\ address}$ ($\sigma_{name="KSROL"}$ (DOCTOR))

Example 2: Consider the following relational schema.

Student (StuName, StuId, Class, Major)

Course (CourseName, CourseNumber, CreditHours, Department)

Section (SectionId, CourseNumber, Semester, Year, Instructor-Name)

GradeReport (StuId, SectionId, Grade)

Prerequisite (CourseNumber, PrerequisiteNumber)

Write down the RA for the following:

1. Display name and Id of those students whose major is finance.

   $\Pi_{StuName,\ StuId}$ ($\sigma_{Major="Finance"}$ (Student))

2. Display the number of courses taught by 'Bhupi'

   $\mathcal{G}_{COUNT\ (CourseNumber)}$ ($\sigma_{Instructor\text{-}Name="Bhupi"}$(Section))

3. Change credit hours of course which course number equal to 11 to 4.

   $\Pi_{CourseName\ ,\ CourseNumber,\ CreditHours\ =\ 4,\ Department}$ ($\sigma_{CourseNumber\ =11}$ (Course))

4. Insert a new student {'Raju', 10, 5, 'DBMS'} in student relation.

   Student ← Student U {"Kamala", 21, "BBA 4th", "DBMS"}

5. Display course name and course number of those courses whose credit hour is 3.

   $\Pi_{CourseName,\ CourseNumber}$ ($\sigma_{CreditHours=3}$ (Course))

Example 3: Consider following relations and write RA statements for given queries:

Police (Pid, Name, Post)

Thief (Tid, name, age)

Catches (Pid, Tid, caught-date)

1. List name of all inspectors

   Π_{Name} ($\sigma_{post="Inspector"}$ (Police))

2. List name of all thieves who are below 20 years

   Π_{name} (σage < 20(Thief))

3. How many thieves are caught on 21 sept, 2012?

   $\mathcal{G}_{COUNT\ (Tid)}$ ($\sigma_{caught\text{-}date="21\ sept,\ 2012"}$ (Catches))

4. Remove all thieves who are above 55 years.

   Thief ←Thief – ($\sigma_{age>55}$) (Thief)

5. Remove record of police whose name is "Rahul"

   Police← Police – ($\sigma_{name="Rahul"}$ (Police))

Example 4: Consider the following relational schema of library where primary keys are underlined.

Employee (E-name, street, city)

Works (E-name, company-name, salary)

Company (company-name, city)

Manages (E-name, manager-name)

Write down RA for the following:

1. List out the names of all employees who work for ABC Company.

   $\Pi_{E\text{-name}}$ ($\sigma_{company\text{-}name="ABC"}$ (Employee ⋈ Works))

2. List out the names of all employees who do not work in any Company.

   $\Pi_{E\text{-name}}$ (Employee) - $\Pi_{E\text{-name}}$ (Employee ⋈ Works ⋈ Company))

3. Count the total no. of employee work in Microsoft Company.

   $\mathcal{G}_{COUNT\ (E\text{-}name)}$ ($\sigma_{company\text{-}name="Microsft"}$ (Works))

4. Insert a new record in Company table.

   Company← Company U {"ABC", "Ktm"}

5. List out the name of employee and salary those are managed by James manager.

   $\Pi_{E\text{-name, salary}}$ ($\sigma_{manager\text{-}name="James"}$ (Manages ⋈ Works))

Relational Calculus

Relational calculus is a non procedural query language. It uses mathematical predicate calculus instead of algebra. It provides the description about the query to get the result where as relational algebra gives the method to get the result. It informs the system what to do with the relation, but does not inform how to perform it. Relational Calculus exists in two forms:

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Relational Calculus is mainly based on the well-known Propositional Calculus, which is a method of calculating with sentences or declarations. Such sentences or declarations, also termed propositions, are ones for which a truth value (i.e. "True" or "false") may be assigned.

The Tuple Relational Calculus

Tuple relational calculus is used for selecting those tuples that satisfy the given condition. It is a logical language with variables ranging over tuples. In tuple relational calculus, we work on filtering tuples based on the given condition.

{t | P (t)} or {t | condition (t)}

Where t is the resulting tuples, P(t) is the condition used to fetch t.

In this form of relational calculus, we define a tuple variable; specify the relation name in which the tuple is to be searched for, along with a condition. We can also specify column name using a dot operator, with the tuple variable to only get a certain attribute (column) in result.

Example

- {t | EMPLOYEE (t) and t.Salary >10000} - implies that it selects the tuples from EMPLOYEE relation such that resulting employee tuples will have salary greater than 10000. It is example of selecting a range of values.
- {t | EMPLOYEE (t) AND t.Dept_Id = 10} – This selects all the tuples of employee name who work for Department 10.
- Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table Student, with age greater than 17, then, for T being our tuple variable,

  T.name | Student (T) AND T.age > 17

The Domain Relational Calculus

In contrast to tuple relational calculus, domain relational calculus uses list of attribute to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples. It is denoted as below:

$\{< a_1, a_2, a_3 \ldots a_n > \mid P(a_1, a_2, a_3 \ldots a_n)\}$

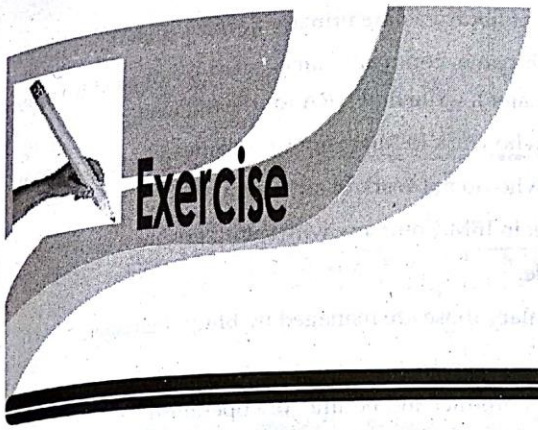
Where $a_1, a_2, a_3 \ldots a_n$ are attributes of the relation and P is the condition.

For example,

- {< name, age > | ∈ Student ∧ age > 17}: This query will return the names and ages of the students in the table Student who are older than 17.
- {<sid, name, address> | <sid, name, address> ∈ STUDENT}: This query return all records of student relation.
- {<name> | <sid, name, address> ∈ STUDENT ∧ sid=45}: Find the name of the student whose id is 45.

Difference between Relational algebra and relational calculus

Relational algebra	Relational calculus
Relational Algebra is a Procedural language.	Relational Calculus is Declarative language.
Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Relational Algebra is not domain dependent.	Relation Calculus can be domain dependent.
It is close to a programming language.	It is close to the natural language.



1. What is the user of RA in DBMS?
2. Mention the pros and cons of relational data model?
3. What so you understand by the term relational algebra? Explain?
4. Explain the fundamental operations of relational algebra.
5. Differentiate between Cartesian product and Join.
6. What is union compatibility? What are the relational algebra operators that require the relation on which they are applied be union compatible?
7. Bring out the reasons, why relational model became more popular?
8. Relational model was based on strong mathematical background.
9. Relational model used the power of mathematical abstraction. Operations do not require user to know storage structures used.
10. Strong mathematical theory provides tool for improving design.
11. Basic structure of the relation is simple, easy to understand and implement.
12. A union, intersection or difference can only be performed between two relations if they are type compatible. What is meant by type compatibility? Give an example of two type compatible and two non-type compatible relations?
13. Consider the following employee database, where primary keys are underlined.

    supplier (supplier-id, supplier-name, city)

    supplies (supplier-id, part-id, quantity)

    parts (part-id, par-name, color, weight)

    Write relation algebra and SQL expression for each of the following queries.

    a) Find the name of all suppliers located in city "Kathmandu" that supplies part 'P01'.
    b) Find the name of all parts supplied by "RD Traders".
    c) Find the name of all parts that are supplied in quantity greater than 300.
    d) Find the number of parts supplied by "S02".
    e) Find number of parts supplied by each supplier.

14. Consider the following relational schema of library where primary keys are underlined.
15. Employee (E-name, street, city), works (E-name, company-name, salary), company (company-name, city), manages (E-name, manager-name), write down RA for the following:
    a) List out the names of all employees who work in Microsoft Company and Intel Company.
    b) List out the names of all employees who do not work in any Company.
    c) Count the total no. of employee work in IBM Company with salary greater than 50000.
    d) Insert a new record in Company table.
    e) List out the name of employee and salary those are managed by Bhupi manager.
16. What is the user of join operation in RA?
17. What is main difference between Cartesian product and natural join operation?
18. What is the use of Relational calculus? How it is differ from RA? Explain.
19. Differentiate between domain relational calculus and tuple relational calculus.
20. What is theta join? How it is differ from equi join? Explain.

❑❑❑

CHAPTER 6

STRUCTURED QUERY LANGUAGE (SQL)

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Data Definition and Data Types
- Specifying Constraints
- Basic Retrieval Queries
- Complex Retrieval Queries
- INSERT, DELETE, and UPDATE Statements
- Views

6.1 Introduction to SQL

Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's Relational model of database. Today almost all RDBMS (MySQL, Oracle, Infomix, Sybase, MS. Access) use SQL as the standard database query language. SQL is used to perform all types of data operations in RDBMS. SQL can also perform administrative tasks on database such as database security, backup, user management etc.

6.2 Types of Structured Query Language (SQL)

In the above section, we learned what we do with the database using SQL. SQL is basically combination of four different languages, they are:

1. DQL (Data Query Language)
2. DML (Data Manipulation Language)
3. DDL (Data Definition Language)
4. DCL (Data Control Language)

DQL (Data Query Language)

DQL is used to fetch the information from the database which is already stored there. The SQL commands used to retrieve the data from database is known as Data Query Language (DQL). One such SQL command in Data Query Language used is SELECT statement which is used as information retrieval from the database.

SQL SELECT syntax

```
SELECT [ALL | DISTINCT] fieldnames
FROM <table name>;
```

The information from the database using SELECT statement can be retrieved column wise or row wise making use of following operations

- Project operation
- Selection

Project operation

To select certain fields from the table or to retrieve the information column wise, we make use of project operation. The resultant table will have the subset of available fields (columns).

Syntax:

```
SELECT [ALL | DISTINCT] FieldName
FROM TableName;
```

By default ALL is used in the SELECT statement.

Let's take following set of tables;

Student

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Abin	Ktm	2
12	Aarav	Ktm	1
13	Ashna	Palpa	3
14	Anuj	Pokhara	4
15	Manish	Banepa	2
16	Pinky	Ktm	1

Subject

Sub_id	Sub_name	Sub_code	Staff_id
20	DBMS	D-20	11
21	C++	C-21	22
22	NM	N-22	11
23	TOC	T-23	77
24	PHP	P-24	44
25	AI	A-25	33
26	ASP	A-26	55
27	CG	C-27	66
28	C-Prog	C-28	44

Subject

Sub_id	Sub_name	Sub_code	Staff_id
20	DBMS	D-20	11
21	C++	C-21	22
22	NM	N-22	11
23	TOC	T-23	77
24	PHP	P-24	44
25	AI	A-25	33
26	ASP	A-26	55
27	CG	C-27	66
28	C-Prog	C-28	44

Marks

Marks_Obtain	Sub_id	Stu_id
60	20	10
55	21	12
58	22	10
49	20	13
61	23	14
67	22	13
60	24	16
55	26	15
33	25	11

Marks

Marks_Obtain	Sub_id	Stu_id
60	20	10
55	21	12
58	22	10
49	20	13
61	23	14
67	22	13
60	24	16
55	26	15
33	25	11

Example: Consider the table or schema Student with the attributes Stu_id, Stu_name, Stu_address, and Dept_id as above.

1. Retrieving the Stu_id column or field;

```
SELECT Stu_id
FROM Student;
```

2. Retrieving the Stu_id and Stu_name columns or fields from student table;

Here we user comma separator for each fields as below

```
SELECT Stu_id, Stu_name
FROM Student;
```

3. To retrieve all fields of the table we use apostrophe operator (*)

```
SELECT *
FROM Student;
```

Selection Operation

It is used to select a subset of tuple (rows) from the relation or schema, or to retrieve the selected rows from the table or schema we use selection operation with certain conditions.

```
SELECT <attribute list>
FROM Tables
WHERE <predicate>;
```

Example

1. Find name of all students whose address is "Palpa"

```
SELECT Stu_name
FROM Student
WHERE Stu_address="Palpa";
```

2. Find id and name of the students whose department id less than 4

```
SELECT Stu_id, Stu_name
FROM Student
WHERE Dept_id<4;
```

SQL SELECT DISTINCT Statement

The SQL SELECT DISTINCT Statement is used to retrieve unique records (by removing the duplicates) from the specified Column in the SELECT Statement. In a table, some of the columns may contain duplicate values. This is not a problem; however, sometimes you will want to list only the different (distinct) values in a table. The DISTINCT keyword can be used to return only distinct values.

Syntax

```
SELECT DISTINCT <Column names>
FROM tables
WHERE <Predicate>;
```

Example

1. Display distinct address of the students,

```
SELECT DISTINCT Stu_address
FROM Student;
```

Output

Stu_address
Palpa
Ktm
Pokhara
Banepa

The WHERE clause

The WHERE clause is optional. When specified, it always follows the FROM clause. The WHERE clause filters rows from the FROM clause tables. Omitting the WHERE clause specifies that all rows are used. The = (equals) comparison operator compares two values for equality. Additional comparison operators and other additional operators used in WHERE clauses are listed below:

WHERE Clause Operators	
Operator	Description
=	Equality
<>	Non-equality
!=	Non-equality
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
!<	Not less than
!>	Not greater than
AND	Logical AND for predicates
OR	Logical OR for predicates
BETWEEN	Between two specified values
NOT BETWEEN	Not between two specified values
IN	Used for multiple OR
NOT IN	
IS NULL	Is a NULL value
LIKE	Used for pattern matching

Use of various comparison operators

1. Find name and age of students whose address is "Ktm" and of department id not equal to 1.

```
SELECT Stu_id, Stu_name
FROM Student
WHERE Stu_address="Ktm" AND Dept_id <>1;
```

2. Find name of the students whose student id less than 4 and of department id greater than 3.

```
SELECT Stu_name
FROM Student
WHERE Stu_id < 4 AND Dept_id > 3;
```

Use of BETWEEN / NOT BETWEEN operators

The SQL BETWEEN operator is used along with WHERE clause for providing a range of values. The values can be the numeric value, text value, and date. The values are defined as part of the BETWEEN range are inclusive i.e. the values that are mentioned in the range are included at the start and end values. SQL BETWEEN operator is almost like SQL IN operators used in a sequential manner.

Syntax of BETWEEN operators

```
SELECT Columns
FROM table_name
WHERE column BETWEEN value1 AND value2;
```

The SQL NOT BETWEEN operator is used for getting the values as part of result set which is outside of the range specified by the BETWEEN operator.

Syntax of NOT BETWEEN operator

```
SELECT Columns
FROM table_name
WHERE column NOT BETWEEN value1 AND value2;
```

Example: Let's take a student relation

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

1. Find id and name student whose age is between 20 and 35.

```
SELECT Stu_id, Stu_name
FROM Student
WHERE Age BETWEEN 20 AND 35;
```

2. Find id and name student whose age is not between 20 and 35.

```
SELECT Stu_id, Stu_name
FROM Student
WHERE Age NOT BETWEEN 20 AND 35;
```

IN operator

It is used in place of multiple OR. If we need to test multiple conditions with a lot of OR then in such a case we can user IN operator

Syntax,

```
SELECT Columns
FROM table_name
WHERE column IN (value1, value2 ... valueN);
```

Example: Let's take a student relation,

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

1. Find name and id of all students whose age is either 20 or 21 or 22 or 23.

```
SELECT Stu_id, Stu_name
FROM Student
WHERE age IN (20, 21, 22, 23);
```

2. Find name of all students whose age is not 21 or 22 or 23 or 24.

```
SELECT Stu_name
FROM Student
WHERE age NOT IN (21, 22, 23, 24);
```

SQL Wildcards (LIKE Operator)

Pattern matching is one of the important operations with strings. SQL allows pattern matching with strings by using LIKE operator. It matches a string value against a pattern string containing wild-card characters. The wild-card characters for LIKE are percent -- '%' and underscore -- '_'. Underscore matches any single character. Percent matches zero or more characters.

Syntax

```
SELECT column-names
FROM table-name
WHERE column-name LIKE <Pattern>;
```

With SQL wildcards, we can create search patterns that can be compared against given data. The wildcards themselves are actually characters that have special meanings within the SQL WHERE clauses. When searching for partial string matches, we might be interested in:

- Strings that begins or ends with a certain character
- Strings that begins or ends with a substring
- Strings containing a certain character anywhere within the string
- Strings containing a substring anywhere within the string
- Strings with a specific format, regardless of individual characters

Wildcard Operators

Operator	Matches
%	A percent sign matches any string of zero or more characters.
_	An underscore matches any one character.

Examples of % and _ Patterns

Pattern	Matches
'A%'	Matches a string of length ≥ 1 that begins with A, including the single letter A. Matches 'A', 'Anonymous', and 'AC/DC'.
'%s'	Matches a string of length ≥ 1 that ends with s, including the single letter s. A string with trailing spaces (after the s) won't match. Matches 's', 'Victoria Falls', and 'DBMSs'.
'%in%'	Matches a string of length ≥ 2 that contains in anywhere. Matches 'in', 'inch', 'Pine', 'linchpin', and 'lynchpin'.
'----'	Matches any four-character string. Matches 'ABCD', 'I am', and 'Abin'.
'Qua__'	Matches any five-character string that begins with Qua. Matches 'Quack', 'Quaff', and 'Quake'.
'_re_'	Matches any four-character string that has're' as its second and third characters. Matches 'Tree', 'area', and 'fret'.
'_re%'	Matches a string of length ≥ 3 that begins with any character and has re as its second and third characters. Matches 'Tree', 'area', 'fret', 'are', and 'fretful'.
'%re_'	Matches a string of length ≥ 3 that has're' as the second and third characters from its end and ends with any character. Matches 'Tree', 'area', 'fret', 'red', and 'Blood red'.
'200%'	Finds any values that start with 200
'%200%'	Finds any values that have 200 in any position
'_00%'	Finds any values that have 00 in the second and third positions.
'2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
'_2%3'	Finds any values that have a 2 in the second position and end with a 3.
'2___3'	Finds any values in a five-digit number that start with 2 and end with 3.

Example: Let's take a student relation

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

1. Find name of the students whose address beginning with letter P and of age less than 25.

```
SELECT Stu_name
FROM Student
WHERE Stu_address LIKE 'P%'AND Age<25;
```

2. Find id of the students whose address end with letter 'a' and of name of 4 letters.

```
SELECT Stu_id
FROM Student
WHERE Stu_address LIKE '%a' AND Stu_name LIKE '____';
```

3. Find id, and name of all Students whose name contains characters 'a' and 'n' such.

```
SELECT Stu_id, Stu_name
FROM Student
WHERE Stu_name LIKE '%a%n%' OR Stu_name LIKE '%n%a%';
```

Ordering the Display of Tuples

SQL Order By clause is used for sorting the result set. The sorting is done in either ascending or in descending order. The default order of sorting is sorting in ascending order. SQL Order By clause is used with SQL Select queries. Sorting is done after the result set is obtained. We can use multiple columns with order by clause, sorting will happen from left side columns first and then towards the right side columns.

Syntax;

```
SELECT <attribute list>
FROM tables
WHERE predicate
ORDER BY column-1 [ASC | DESC] [column-2 [ASC | DESC]];
```

Column-1, column-2 ... are column names specified, the new name is used in the ORDER BY list. ASC and DESC request ascending or descending sort for a column. ASC is the default. ORDER BY sorts rows using the ordering columns in left-to-right, major-to-minor order. The rows are sorted first on the first column name in the list. If there are any duplicate values for the first column, the duplicates are sorted on the second column in the order by list, and so on. Database nulls require special processing in ORDER BY. A null column sorts higher than all regular values; this is reversed for DESC.

Example

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

1. Find id and name of all students in the order of age.

```
SELECT Stu_id, Stu_name
FROM Student
ORDER BY age ASC;
```

Here ASC is optional. Since by default order is ascending order

2. Find id and name of all students whose address is 'Ktm' in ascending order of their name.

```
SELECT Stu_id, Stu_name
FROM student
WHERE Stu_address='Ktm'
ORDER BY Stu_name;
```

3. Find id and age of all students in the order of their address and then in descending order of age.

```
SELECT Stu_id, age
FROM Student
ORDER BY Stu_address, age DESC;
```

Aggregate Functions

Aggregate functions operate against a collection of values, but return a single value. It is mainly used in summarizing the information stored in tables. Aggregate functions supported by SQL are listed below:

Function	Description
AVG(column)	Returns the average value of a column
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
SUM(column)	Returns the total sum of a column

The result of the COUNT function is always integer. The result of all other functions is the same data type as the argument. The Aggregate Functions skip columns with nulls, summarizing non-null

values. COUNT counts rows with non-null values; AVG averages non-null values, and so on. COUNT returns 0 when no non-null column values are found; the other functions return null when there are no values to summarize. The DISTINCT and ALL specifiers are optional. ALL specifies that all non-null values are summarized; it is the default. DISTINCT specifies that distinct column values are summarized; duplicate values are skipped. Note that DISTINCT has no effect on MIN and MAX results. Aggregate functions have the following general format:

Aggregate-function ([DISTINCT | ALL] column-1)

In this syntax;

- First, specify the name of an aggregate function that you want to use such as AVG, SUM, and MAX.
- Second, use DISTINCT if you want only distinct values are considered in the calculation or ALL if all values are considered in the calculation. By default, ALL is used if you don't specify any modifier.
- Third, the expression can be a column of a table or an expression that consists of multiple columns with arithmetic operators.

Example Queries

Student

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Abin	Ktm	2
12	Aarav	Ktm	1
13	Ashna	Palpa	3
14	Anuj	Pokhara	4
15	Manish	Banepa	2
16	Pinky	Ktm	1

Marks

Marks_Obtain	Sub_id	Stu_id
60	20	10
55	21	12
58	22	10
49	20	13
61	23	14
67	22	13
60	24	16
55	26	15
33	25	11

1. Find minimum and maximum id of students

```
SELECT max (Stu_id), min (Stu_id)
FROM student;
```

2. Find average marks of all students whose id less than 15.

```
SELECT AVG (Marks_Obtain)
FROM Marks
WHERE Stu_id<15;
```

3. Find total number of students.

```
SELECT COUNT (*)
FROM Student;
```

4. Find number of subjects enrolled by student 'Maya'

```
SELECT COUNT (Sub_id) AS NO_Of_Subject
FROM Student, Marks
WHERE Stu_name='Maya' AND Student.Stu_id=Marks.Stu_id;
```

GROUP BY Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax

```
SELECT column_name, function (column_name)
FROM table_name
WHERE condition
GROUP BY column_name;
```

Example of Group by in a Statement

Employee

eid	Name	age	Salary
401	Anu	22	9000
402	Abin	29	8000
403	Rohan	34	6000
404	Aarav	44	9000
405	Geeta	35	8000

Here we want to find name and age of employees grouped by their salaries or in other words, we will be grouping employees based on their salaries, hence, as a result, we will get a data set, with unique salaries listed, alongside the first employee's name and age to have that salary. GROUP BY is used to group different row of data together based on any one column.

1. Find name and age of all employees in each age level

```
SELECT name, age
FROM Employee GROUP BY salary;
```

Result will be,

Name	Age
Rohan	34
Abin	29
Anu	22

2. Find name and salary of all employees whose age greater than 25 in each salary level.

```
SELECT name, salary
FROM Employee
WHERE age > 25
GROUP BY salary;
```

Result will be

Name	Salary
Rohan	6000
Abin	8000
Aarav	9000

HAVING Clause

Having clause is used with SQL queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query. Simply the GROUP BY clause is used to check the condition in each group created by GROUP BY clause.

Syntax

```
SELECT column_name, function (column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
HAVING function (column_name) condition;
```

Example: Let's take a student relation as below,

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

1. Find average age for all departments of student that have average age more than 22.

```
SELECT Dept_id, AVG (age)
FROM student
GROUP BY Dept_id
HAVING AVG (age)>22;
```

2. Count number of students in each department that have students more than 1.

```
SELECT COUNT (Stu_id)
FROM student
GROUP BY Dept_id
HAVING COUNT (Stu_id)>1;
```

Differences between WHERE and HAVING clause

- WHERE clause is employed in row operations and applied on a single row whereas HAVING clause is used in column operations and can be applied to summarized rows or groups.
- In WHERE clause the desired data is fetched according to the applied condition. In contrast, HAVING clause fetch whole data then separation is done according to the condition.

- Aggregate functions like min, sum, max, avg can never appear along with WHERE clause. As against, these functions can appear in HAVING clause.
- HAVING clause cannot use without a SELECT statement. Conversely, WHERE can be used with SELECT, UPDATE, DELETE, etc.
- WHERE clause behaves as a pre-filter while HAVING clause acts as a post-filter.
- WHERE clause when used with GROUP BY, comes before GROUP BY. This signifies that WHERE filter rows before aggregate calculations are performed. On the other hand, HAVING comes after GROUP BY, which means it filters after aggregate calculations are performed.

SQL Join

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables. JOIN Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is (n-1) where n, is number of tables. A table can also join to itself, which is known as, Self Join.

Types of JOIN in SQL

Following are the types of JOIN that we can use in SQL:

- Cross join
- Inner join
- Natural join
- Outer join
  - Left outer join
  - Right outer join
  - Full outer join

Cross JOIN or Cartesian product

This type of JOIN returns the Cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
FROM table-name1 CROSS JOIN table-name2;
```

Example: Let's take two relations department and staff

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4

```
SELECT *
FROM Department CROSS JOIN Staff;
```

We get following result

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	22	Pratima	2
1	Computer	100	33	Madan	1
1	Computer	100	44	Kamala	3
1	Computer	100	55	Sandhya	4
2	Mathematics	200	11	Mohan	1
2	Mathematics	200	22	Pratima	2
2	Mathematics	200	33	Madan	1
2	Mathematics	200	44	Kamala	3
2	Mathematics	200	55	Sandhya	4
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1
3	Economics	300	44	Kamala	3
3	Economics	300	55	Sandhya	4
4	Account	400	11	Mohan	1
4	Account	400	22	Pratima	2
4	Account	400	33	Madan	1
4	Account	400	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics	500	11	Mohan	1
5	Physics	500	22	Pratima	2
5	Physics	500	33	Madan	1
5	Physics	500	44	Kamala	3
5	Physics	500	55	Sandhya	4

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

```
SELECT column-name-list
FROM table-name1 INNER JOIN table-name2
WHERE table-name1.column-name = table-name2.column-name;
```

Example:

```
SELECT *
FROM Department INNER JOIN Staff
WHERE Department.Dept_id = Staff.Dept_id;
```

By running this code we get following output

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same data type present in both the tables to be joined.

The syntax for Natural Join is,

```
SELECT attribute list
FROM table-name1 NATURAL JOIN table-name2;
```

Example

```
SELECT *
FROM Department NATURAL Staff;
```

By running above query we get following result

Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name
1	Computer	100	11	Mohan
1	Computer	100	33	Madan
2	Mathematics	200	22	Pratima
3	Economics	300	44	Kamala
4	Account	400	55	Sandhya

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

LEFT Outer Join

The left outer join returns a result set table with the matched data from the two tables and then the remaining rows of the left table and null from the right table's columns.

Syntax for Left Outer Join is,

```
SELECT column-name-list
FROM table-name1 LEFT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

To specify a condition, we use the ON keyword with Outer Join.

Example: Let's take following two relations Department and staff

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1
44	Kamala	3
55	Sandhya	4
23	Ramesh	7

```
SELECT *
FROM Department LEFT OUTER JOIN Staff
ON Department.Dept_id = Staff.Dept_id;
```

By running above query we get following output

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics	500	NULL	NULL	NULL

RIGHT Outer Join

The right outer join returns a result set table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

Syntax for Right Outer Join is,

```
SELECT column-name-list
FROM table-name1 RIGHT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Example: Let's find right outer join to above department and staff table as,

```
SELECT *
FROM Department RIGHT OUTER JOIN Staff
ON Department.Dept_id = Staff.Dept_id;
```

By running above query we get following output

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
NULL	NULL	NULL	23	Ramesh	7

Full Outer Join

The full outer join returns a result set table with the matched data of two table then remaining rows of both left table and then the right table.

Syntax of Full Outer Join is,

```
SELECT column-name-list
FROM table-name1 FULL OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Example: Let's find right outer join to above department and staff table as,

```
SELECT *
FROM Department FULL OUTER JOIN Staff
ON Department.Dept_id = Staff.Dept_id;
```

By running above query we get following output

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics	500	NULL	NULL	NULL
NULL	NULL	NULL	23	Ramesh	7

SQL Alias - AS Keyword

Alias is used to give an alias name to a table or a column, which can be a result set table too. This is quite useful in case of large or complex queries. Alias is mainly used for giving a short alias name for a column or a table with complex names.

Syntax of Alias for table names,

```
SELECT column-name FROM table-name AS alias-name
```

SET Operations in SQL

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions. In this book, we will cover 4 different types of SET operations, along with example:

- UNION
- UNION ALL
- INTERSECT
- MINUS

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables, on which UNION operation is being applied. Union query will be like,

```
SELECT * FROM First table
UNION
SELECT * FROM Second table;
```

Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are ignored in case of UNION but not ignored in case of UNION ALL during duplicate checking.
- The IN operator has a higher precedence than the UNION operator.
- By default, the output is sorted in ascending order of the first column of the SELECT clause.

UNION ALL

This operation is similar to Union. But it also shows the duplicate rows. Union All query will be like,

```
SELECT * FROM First table
UNION ALL
SELECT * FROM Second table;
```

INTERSECT

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of Intersect the number of columns and data type must be same. Intersect query will be,

```
SELECT * FROM First table
INTERSECT
SELECT * FROM Second table;
```

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result. Minus query will be,

```
SELECT * FROM First table
MINUS
SELECT * FROM Second table;
```

Nested Query in SQL (Sub query)

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc. There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Sub query Syntax

```
SELECT  field_name
FROM    table_name
WHERE   field_name expression operator
            (SELECT  field_name FROM table_name);
```

A sub query is a SELECT statement that is embedded in a clause of another SELECT statement. We can build powerful statements out of simple ones by using sub queries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself. We can place the sub query in a number of SQL clauses, including the following:

- WHERE clause
- HAVING clause
- FROM clause

In the syntax: operator includes a comparison condition such as >, =, or IN

Note: Comparison conditions fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL). The ANY and ALL operators are used with a WHERE or HAVING clause. The ANY operator returns true if any of the sub query values meet the condition. The ALL operator returns true if all of the sub query values meet the condition.

Example: Display name of all students whose age is greater than a student of address 'Ktm'.

```
SELECT Stu_name
FROM  Student
WHERE age > (SELECT age FROM  Student WHERE Stu_address = 'Ktm');
```

Types of Sub queries

Single-row sub queries: Queries that return only one row from the inner SELECT statement. These types of sub query use a single-row operator.

Example: Display the students whose id is the same as that of student of id 14:

```
SELECT Stu_name, Stu_id
FROM  Student
WHERE Stu_id =
        (SELECT Stu_id FROM Student WHERE Stu_id = 14);
```

Multiple-row sub queries: Queries that return more than one row from the inner SELECT statement. We use multiple-row operator, instead of a single-row operator, which a multiple-row sub query. The multiple-row operator expects one or more values:

```
SELECT Stu_name, Stu_id, Stu_address
FROM Student
WHERE age IN
    (SELECT  MIN (age) FROM Student GROUP BY Stu_address);
```

The ANY operator (and its synonym, the SOME operator) compares a value to each value returned by a sub query.

```
SELECT Stu_id, Stu_name, age
FROM Student
WHERE age < ANY
      (SELECT age FROM Student WHERE  Stu_age >20)
          AND   Stu_address <> 'Ktm';
```

The ALL operator compares a value to every value returned by a sub query. The NOT operator can be used with IN, ANY, and ALL operators.

```
SELECT Stu_id, Stu_name
FROM  Student
WHERE age < ALL
        (SELECT age FROM Student WHERE Stu_address = 'Ktm')
AND   Stu_id <> 10;
```

DML (Data Manipulation Language)

DML is short name of Data Manipulation Language which deals with data manipulation, and includes most common SQL statements such INSERT, UPDATE, DELETE etc, and it is used to store, modify, delete and update data in database.

INSERT Statement

The INSERT INTO statement of SQL is used to insert a new row in a table. There are two ways of using INSERT INTO statement for inserting rows:

Only values: First method is to specify only the value of data to be inserted without the column names.

Syntax:

INSERT INTO table_name VALUES (value1, value2, value3...);

Column names and values both: In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:

Syntax:

INSERT INTO table_name (column1, column2, column3...) VALUES (value1, value2, value3...);

Example: Let's take a student table

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age

INSERT INTO Student VALUES (1, "Aarav", "Kathmandu", 11, 12);

Also we can insert more than one record at a time without details of column names as below,

INSERT INTO Student VALUES (1, "Aarav", "Kathmandu", 11, 12), (2, "Abina", "Palpa", 10, 22), (5, "Kamala", "Kanchanpur", 21, 32);

Also in second way we can inset the data to the table by specifying their column names as below,

INSERT INTO Student (Stu_id, Stu_name, Stu_address, Dept_id, Age)
VALUES (1, "Aarav", "Kathmandu", 11, 12);

UPDATE Statement

The UPDATE statement in SQL is used to update the data of an existing table in database. We can update single columns as well as multiple columns using UPDATE statement as per our requirement.

Syntax

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

Example: Let's take a Student relation as below,

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

1. Change address of all students whose age less than 25 to "Kanchanpur"

```
UPDATE Student
SET address = 'Kanchanpur'
WHERE Age<25;
```

2. Update name of student to 'Aayan' and address to "Kanchanpur" whose age less than 25 and of id less than 15.

```
UPDATE Student
SET Stu_name = 'Aayan', Stu_assress='Kanchanpur'
WHERE Age<25 AND stu_id<15;
```

DELETE Statement

The DELETE Statement in SQL is used to delete existing records from a table. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

Syntax:

```
DELETE FROM table_name
WHERE some_condition;
```

Example: Let's take above student relation,

1. Delete student of name 'Anuj'

```
DELETE FROM Student
WHERE Stu_name='Anuj';
```

2. Delete all students whose age less than 20

```
DELETE FROM Student
WHERE Age<20;
```

3. Delete all students of address 'Ktm' and of age greater than 20.

```
DELETE FROM Student
WHERE Stu_address='Ktm' AND Age>20;
```

DDL (Data Definition Language)

Data Definition Language (DDL) is a standard for commands that define the different structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

Data Types in SQL

Each column in a database table is required to have a name and a data type. An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data. The SQL standard supports a variety of built-in domain types. This subsection describes data type specifications included in SQL as below:

Text data types

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 bytes
BLOB	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number data types

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size, d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size, d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size, d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

Date data types

Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS Note: The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

CREATE Statement

There are two CREATE statements available in SQL:

1. CREATE DATABASE
2. CREATE TABLE

CREATE DATABASE

A Database is defined as a structured set of data. So, in SQL the very first step to store the data in a well structured manner is to create a database. The CREATE DATABASE statement is used to create a new database in SQL.

Syntax:

```
CREATE DATABASE database_name;
```

Example: Suppose we need to create a database College_MIS then we can create such a database in SQL as below

```
CREATE DATABASE College_MIS;
```

CREATE TABLE

We have learned above about creating databases. Now to store the data we need a table to do that. The CREATE TABLE statement is used to create a table in SQL. We know that a table comprises of rows and columns. So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data etc. Let us now dive into details on how to use CREATE TABLE statement to create tables in SQL.

Syntax:

```
CREATE TABLE table_name
(
    column1 data_type(size),
    column2 data_type(size),
    column3 data_type(size),
    ....
);
```

Example: Let's create a student relation as below

Stu_id	Stu_name	Stu_address	Dept_id	Age

```
CREATE TABLE Students
(
        Stu_id int(2) PRIMARY KEY,
        Stu_name varchar(20),
        Stu_address varchar(20),
        Dept_id int(2),
        Age int(3)
);
```

This query will create a table named Students. The Stu_id field is of type int and can store an integer number of size 2. The next two columns name and address are of type varchar and can store characters and the size 20 specifies that these two fields can hold maximum of 20 characters. Similarly next two fields are department id and age that can store an integer number of size 2 and 3 respectively.

ALTER Statement

Allow us to modify a given table. The structure of given table can be changed either of the following:

- By adding new column in existing table
- By deleting some columns from an existing table and
- By modifying some columns of given table

A new column can be added to the table as follows:

Syntax:

```
ALTER TABLE <table name>
ADD (<column_name>  <datatype>);
```

Example: suppose we want to add a new column 'addresses' to an existing table Student,

Student

Sid	Sname	Level	Age	Sex

```
ALTER TABLE Student
ADD (addresses VARCHAR (15));
```

If we execute the above query then we get following table,

Student

Sid	Sname	Level	Age	Sex	address

An existing column can be removed from the table as,

ALTER TABLE <table_name>

DROP (column_name);

Example: suppose we want to remove an existing column 'addresses' from the table Student as,

ALTER TABLE Student

DROP (addresses);

If we execute the above query then we get following table,

Student

Sid	Sname	Level	Age	Sex

An existing column can be modified as,

ALTER TABLE <table_name>

MODIFY (<column name> <data type>);

Also we can change the name of the field of given table by using CHANG keyword as below,

ALTER TABLE <table_name>

CHANGE <Existing field> <New name of existing field>

Example 1: modify Student relation by changing the range of the name of Student by 30,

ALTER TABLE Student

MODIFY (sname VARCHAR (30));

Example 2: modify Student relation by changing the field sname to Student_name.

ALTER TABLE friends CHANGE Sname, Student_name;

Student

Sid	Student_name	Level	Age	sex

DCL (Data Control Language)

The data control language (DCL) is used for user & permission management. It controls the access to the database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

- **System:** This includes permissions for creating session, table, etc and all types of other system privileges.
- **Object:** This includes permissions for any command or query to perform any operation on the database tables.

In DCL we have two commands,

- **GRANT:** Used to provide any user access privileges or other privileges for the database.
- **REVOKE:** Used to take back permissions from any user.

Allow a User to create session

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user. Following command can be used to grant the session creating privileges.

```
GRANT CREATE SESSION TO username;
```

Allow a User to create table

To allow a user to create tables in the database, we can use the below command,

```
GRANT CREATE TABLE TO username;
```

Provide user with space on table space to store table

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with privileges to use the available table space for their table and data.

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

The above command will alter the user details and will provide it access to unlimited table space on system.

Grant all privilege to a User

sysdba is a set of privileges which has all the permissions in it. So if we want to provide all the privileges to any user, we can simply grant them the sysdba permission.

```
GRANT sysdba TO username
```

Grant permission to create any table

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

```
GRANT CREATE ANY TABLE TO username
```

Grant permission to drop any table

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

```
GRANT DROP ANY TABLE TO username
```

To take back Permissions

If we want to take back the privileges from any user, use the REVOKE command.

```
REVOKE CREATE TABLE FROM username
```

Specifying Constraints

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

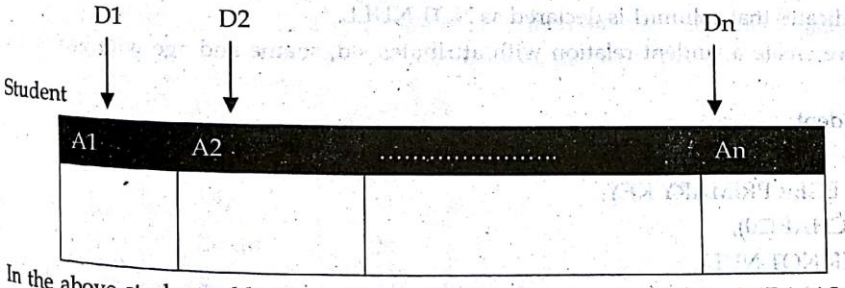
Following are some of the most commonly used constraints available in SQL. The following constraints are commonly used in SQL:

1. Domain constraints
   - DEFAULT constraint
   - NOT NULL constraints
   - UNIQUE constraints
   - PRIMARY KEY constraints
   - CHECK constraints etc.
2. Referential constraints
3. Triggers
4. Assertion

Domain constraint

Domains are used in the relational model to define the characteristics of the columns of a table. Domain refers to the set of all possible values that attribute can take. The domain specifies its own name, data type, and logical size. The logical size represents the size as perceived by the user, not how it is implemented internally. For example, for an integer, the logical size represents the number of digits used to display the integer, not the number of bytes used to store it. The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values. Strictly speaking, only values from the same domain should ever be compared or be integrated through a union operator. The domain integrity constraint specifies that each attribute must have values derived from a valid range.

Example: The age of the person cannot have any letter from the alphabet. The age should be a numerical value.



In the above student table, the attribute A1 draws value from domain D1, A2 from D2 and so on. Domain integrity means the definition of a valid set of values for an attribute.

The CREATE DOMAIN clause can be used to define new domains. For example, to ensure that age must be an integer in the range 1 to 100, we could use:

```
CREATE DOMAIN <Domain Name> INTEGER DEFAULT 0
CHECK (VALUE >= 1 AND VALUE <= 100)
```

The other basic constraints related to domain constraints are described as below:

DEFAULT Constraint

It is used to sets a default value for a column when no value is specified. The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table. Let's see how to specify this constraint and how it works.

Specify DEFAULT constraint while creating a table

Here we are creating a table **Student** we have a requirement to set the exam fees to 10000 if fees is not specified while inserting a record (row) into the Student table. We can do so by using DEFAULT constraint. As you can see we have set the default value of Exam_Fee column to 10000 using DEFAULT constraint.

```
CREATE TABLE Students
(
    Roll_No  INT          NOT NULL PRIMAY KEY,
    Stu_Name              VARCHAR (35)  NOT NULL,
    Stu_Age               INT NOT NULL,
    Exam_Fee              INT DEFAULT 10000,
    Stu_Address                   VARCHAR (35)
);
```

NOT NULL Constraint

If one is very much particular that the column is not supposed to take NULL value then we can impose NOT NULL constraint on that column. The syntax of NOT NULL constraint is:

```
CREATE TABLE <table name>
(
        Column name1, data-type of the column1, NOT NULL
        Column name2, data-type of the column2,
        Column nameN, data-type of the column
);
```

The above syntax indicates that column1 is declared as NOT NULL.

Example: suppose we create a student relation with attributes sid, sname and age with age is not null.

```
CREATE TABLE Student
(
    Sid      INTEGER PRIMARY KEY,
    sname VARCHAR(20),
    age INTEGER NOT NULL
)
```

If we execute the following query,

INSERT INTO Student VALUES (1, "Binek", 05); then one record is successfully inserted into the student table and if we display such a table then we get following table,

Sid	Sname	Age
1	Bines	05

But if we execute the query,

INSERT INTO Student VALUES (2, "Geeta", NULL); then we get age cannot be null message and insertion is failed.

UNIQUE Constraint

The UNIQUE constraint imposes that every value in a column or set of columns be unique and null values are allowed. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

Syntax of UNIQUE is,

```
CREATE TABLE <table name>
(
        Column name1, data-type of the column1 UNIQUE,
        Column name2, data-type of the column2,
        Column nameN, data-type of the column
);
```

The above syntax indicates that column1 is declared as UNIQUE.

Example: Let's create a student relation with attributes sid, sname, age with age is unique.

```
CREATE TABLE Student
(
        Sid     INTEGER PRIMARY KEY,
        sname VARCHAR(20),
        age INTEGER UNIQUE
)
```

If we execute the following query,

INSERT INTO Student VALUES (1, "Abin", 04);

INSERT INTO Student VALUES (2, "Aayan", 11);

INSERT INTO Student VALUES (3, "Bindu", 26); then three records are successfully inserted into the student table and if we display such a table then we get following table,

Sid	Sname	Age
1	Abin	04
2	Aayan	11
3	Bindu	26

But if we insert the duplicate aged record like below

INSERT INTO Student VALUES (4, "Umesh", 11); then we get duplicate entry for age error message and insertion is failed.

Primary Key Constraint

When an attribute or set of attributes is declared as the primary key, then the attribute will not accept NULL value moreover it will not accept duplicate values. It is to be noted that "only one primary key can be defined for each table."

Syntax,

CREATE TABLE <table name>
(
Column name1, data-type of the column1 PRIMARY KEY,
Column name2, data-type of the column2,
Column nameN, data-type of the column
);

The above syntax indicates that column1 is declared as PRIMARY KEY constraint.

Example

```
CREATE TABLE Student
(
        Sid     INTEGER PRIMARY KEY,
        sname VARCHAR(20),
        age INTEGER UNIQUE
)
```

If we execute the following query,

INSERT INTO Student VALUES (1, "Abin", 04);

INSERT INTO Student VALUES (2, "Aayan", 11);

INSERT INTO Student VALUES (3, "Bindu", 26); then three records are successfully inserted into the student table and if we display such a table then we get following table,

Sid	Sname	Age
1	Abin	04
2	Aayan	11
3	Bindu	26

But if we insert the duplicate aged record like below

INSERT INTO Student VALUES (4, "Umesh", 11); then we get duplicate entry for primary key attribute error message and insertion is failed. Also if we leave primary keys value for a particular record like below,

INSERT INTO Student VALUES ("Geeta", 25); then we get primary key cannot be null error message and insertion is failed.

CHECK Constraints

CHECK constraint is added to the declaration of the attribute. The CHECK constraint may use the name of the attribute or any other relation or attribute name may in a sub-query. Attribute value check is checked only when the value of the attribute is inserted or updated. CHECK constraints allow users to prohibit an operation on a table that would violate the constraint. It is a local constraint.

Example: let's create a student table with attributes student id, student name, age and address. If we need to allow only those students in the table whose age must be an integer range 20 to 45, we could use the CHECK constraint during the creation of table as below:

```
CREATE TABLE Student
(
        sid     INTEGER,
        sname VARCHAR(20),
        age     INTEGER,
        PRIMARY KEY (sid),
        CHECK (age>=20 AND age<=45)
)
```

In the above student table if we are trying to insert a new record as

```
INSERT INTO Student
VALUES (5, "Rajesh", 15);
```

We get insertion is rejected message since value of age attribute violated the check condition.

Referential Integrity

Referential Integrity is set of constraints applied to foreign key which prevents entering a row in child table (where you have foreign key) for which you don't have any corresponding row in parent table i.e. entering NULL or invalid foreign keys. Referential Integrity prevents your table from having incorrect or incomplete relationship.

Referential Integrity in SQL

Example of Referential Integrity is Student and Marks relationship. If we have Stu_id as foreign key in marks table than by using referential integrity constraints we can avoid creating Marks table without Student or non existing Marks table. In short Referential Integrity makes primary key foreign key relationship viable. Let's first create student and marks table with primary key, foreign key and referential Integrity constraints.

Student

Stu_id	Stu_name	Stu_address	Dept_id

Marks

Marks_Obtain	Sub_id	Stu_id

Primary keys and foreign keys can be specified as parts of the SQL create table statement as below:

```
Create table Student
(
        Stu_id INT PRIMARY KEY,
        Stu_name        VARCHAR (20),
        Stu_address VARCHAR (15),
        Dept_id INT
);
```

```
Create table Marks
(
        Marks_obtain  INT,
        Sub_id  INT,
        Stu_id          INT,
        FOREIGN KEY (Stu_id)
        REFERENCES Student (Stu_id)
);
```

Advantage of Referential Integrity in RDBMS and SQL

There is several benefit of Referential Integrity in relational database and maintaining integrity of data among parent and child tables. Here are some of the most noticed advantages of Referential Integrity in SQL:

- Referential Integrity prevents inserting records with incorrect details in table. Any insert or update operation will fail if it doesn't satisfy referential integrity rule.
- If a record from parent table is deleted, referential integrity allows deleting all related records from child table using cascade-delete functionality.
- Similar to second advantage if a record i.e. Stu_id of a Student is updated on parent table (Student), Referential Integrity helps to update Stu_id in child table (Marks) using cascade-update.

Assertions

Assertions are general purpose checks that allow the enforcement of any condition over the entire database. Similar to CHECK but they are global Constraints. When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion. This testing may introduce a significant amount of overhead; hence assertions should be used with great care. An assertion in SQL takes the form:

Create assertion <assertion-name> check <predicate>

Example: the department id of manager relation is always not null since each manager works at least one department.

```
CREATE ASSERTION NoBench CHECK
        (NOT EXISTS
                (SELECT * FROM MANAGER
                        WHERE DeptId IS NULL));
```

Above assertion ensures that there is no manager who is not assigned any department at any time. Let's take a manager relation in which some records are inserted as

Manager

Mid	Mname	Address	DeptId
M01	Aayan	Pokhara	D11
M02	Bhupi	lalitpur	D22
M03	Arjun	Kathmandu	D11
M05	Ramesh	Palpa	Null

In the above table the department id of manager 'Ramesh' is null due to which assertion is violated and we cannot further modify the database.

Assertions can be dropped using the DROP ASSERTION command.

```
DROP ASSERTION NoBench;
```

Triggers

A trigger is a procedure (statement) that is automatically invoked by the DBMS in response to specified changes to the database. A database that has a set of associated triggers is called an active database. Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met. It is the most practical way to implement routines and granting integrity of data. Unlike the stored procedures or functions, which have to be explicitly invoked, these triggers implicitly get fired whenever the table is affected by the SQL operation. For any event that causes a change in the contents of a table, a user can specify an associated action that the DBMS should carry out. Trigger follows the Event-Condition-Action scheme (ECA scheme). To design a trigger mechanism, we must meet following three requirements:

1. **Event:** A change to the database that activates the trigger.
2. **Condition:** Trigger performs some action only if a specified condition matches at the occurrence of the event
3. **Action:** A procedure that is executed when the trigger is activated and its condition is true.

Privileges Required for Triggers

Creation or alteration of a TRIGGER on a specific table requires TRIGGER v privileges as well as table privileges. They are:

1. To create TRIGGER in one's own schema, he must have CREATE TRIGGER privilege. To create a trigger in any other' s schema, one must have CREATE ANY TRIGGER system privilege.
2. To create a trigger on table, one must own the table or should have ALTER privilege for that table or should have ALTER ANY TABLE privilege.
3. To ALTER a trigger, one must own that trigger or should have ALTER ANY TRIGGER privilege. Also since the trigger will be operating on some table, one also requires ALTER privilege on that table or ALTER ANY TABLE table privilege.
4. To create a TRIGGER on any database level event, one must have ADMINISTER DATABASE TRIGGER system privilege.

General form of trigger is

```
DEFINE TRIGGER <trigger-name>
        <Time events>
            ON <list-of-tables>
                WHEN <Predicate>
                    <Action-name>
```

Three event types

- Insert
- Update
- Delete

Two triggering times

- Before the event
- After the event

Two granularities

- Execute for each row
- Execute for each statement

Example

```
CREATE TRIGGER overdraft AFTER UPDATE ON pre-paid
        REFERENCING NEW ROW AS nrow
                FOR EACH ROW
                        WHEN nrow.balance <= 0
                                UPDATE pre-paid
                                        SET BLOCKED = 'T';
```

For every pre-paid account whose balance is less than or equal to 0, the account is automatically marked as "blocked".

Let's take a pre-paid relation as

Pre-paid

PNO	Balance	Blocked
1	4000	False
2	5000	False
3	7000	False
4	2000	False

```
UPDATE TABLE pre-paid
SET Balance = Balance-4000;
```

If we execute this query then we get following modified Pre-paid table

PNO	Balance	Blocked
1	4000	T
2	5000	False
3	7000	False
4	2000	T

Views

A database view is a logical table. It does not physically store data like tables but represent data stored in underlying tables in different formats. A view does not require desk space and we can use view in most places where a table can be used. Since the views are derived from other tables thus when the data in its source tables are updated, the view reflects the updates as well. They also can be used by DBA to enforce database security.

Advantages of Views

- Database security: view allows users to access only those sections of database that directly concerns them.
- View provides data independence.
- Easier querying
- Shielding from change
- Views provide group of users to access the data according to their criteria.
- Vies allow the same data to be seen by different users in different ways at the same time.

When the column of a view is directly derived from the column of a base table, that column inherits any constraints that apply to the column of the base table. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations using that view are subject to the same referential constraints as the base table.

Syntax for creating view is:

CREATE VIEW <view name> AS <query expression>

Where, <query expression> is any legal query expression.

Example: Following view contains the id, name, and address of those Students whose age is greater than 25.

Student

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	21
11	Abin	Ktm	2	22
12	Aarav	Ktm	1	19
13	Ashna	Palpa	3	33
14	Anuj	Pokhara	4	23
15	Manish	Banepa	2	22
16	Pinky	Ktm	1	43

```
CREATE VIEW Student_view AS
SELECT Stu_id, Stu_name, Stu_address
FROM Student
WHERE age>25;
```

Now by executing this query we get following view (logical table);

Student_view

Stu_id	Stu_name	Stu_address
13	Ashna	Palpa
16	Pinky	Ktm

Now any valid database operations can be performed in this view like in that of general table. It is a named specification of a result table. The specification is a SELECT statement that is executed whenever the view is referenced in an SQL statement. Consider a view to have columns and rows just like a base table. For retrieval, all views can be used just like base tables.

Embedded SQL

SQL can be used in conjunction with a general purpose programming language such as PASCAL, C, C++, etc. The programming language is called the host language. Embedded SQL statements are SQL statements written within application programming languages such as C and Java. The

embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor can extract the SQL statements. These statements are preprocessed by an SQL pre-compiler before the application program is compiled. There are two types of embedded SQL, Static SQL, and Dynamic SQL. Embedded SQL provides the 3GL (Third Generation Language) with a way to manipulate a database. Embedded SQL supports highly customized applications. It also supports background applications running without user intervention.

Static SQL

The source form of a static SQL statement is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program. A source program containing static SQL statements must be processed by an SQL pre-compiler before it is compiled. The pre-compiler turns the SQL statements into host language comments, and generates host language statements to invoke the database manager. The syntax of the SQL statements is checked during the pre-compile process. The preparation of an SQL application program includes pre-compilation, the binding of its static SQL statements to the target database, and compilation of the modified source program.

Dynamic SQL

Programs containing embedded dynamic SQL statements must be precompiled like those containing static SQL, but unlike static SQL, the dynamic SQL statements are constructed and prepared at run time. The SQL statement text is prepared and executed using either the PREPARE and EXECUTE statements, or the EXECUTE IMMEDIATE statement. The statement can also be executed with cursor operations if it is a SELECT statement.

How to Create a New User

Up to now, we did all of the editing in MySQL as the root user, with full access to all of the databases. However, in cases where more restrictions may be required, there are ways to create users with custom permissions. Let's start by making a new user within the MySQL shell:

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
```

We can review a user's current permissions by running the following:

```
SHOW GRANTS username;
```

We can also delete user by using following command,

```
DROP USER 'username'@'localhost';
```

Grant / Revoke Privileges in MYSQL

Granting Privileges

We have already learned about how to create user in MySQL using MySQL | create user statement. But using the Create User Statement only creates a new user but does not grant any privileges to the user account. Therefore to grant privileges to a user account, the GRANT statement is used.

Syntax:

GRANT privileges_names ON object TO user;

Parameters Used

- **privileges_name:** These are the access rights or privileges granted to the user.
- **Object:** It is the name of the database object to which permissions are being granted. In the case of granting privileges on a table, this would be the table name.
- **User:** It is the name of the user to whom the privileges would be granted.

Different ways of granting privileges to the users are listed below

1. **Granting SELECT Privilege to a User in a Table:** To grant Select Privilege to a table named "users" where User Name is BHUPI, the following GRANT statement should be executed.

   GRANT SELECT ON Users TO' BHUPI '@'localhost;

2. **Granting more than one Privilege to a User in a Table:** To grant multiple Privileges to a user named "BHUPI" in a table "users", the following GRANT statement should be executed.

   GRANT SELECT, INSERT, DELETE, UPDATE ON Users
   TO ' BHUPI '@'localhost;

3. **Granting All the Privilege to a User in a Table:** To Grant all the privileges to a user named "BHUPI" in a table "users", the following Grant statement should be executed.

   GRANT ALL ON Users TO ' BHUPI '@'localhost;

4. **Granting a Privilege to all Users in a Table:** To Grant a specific privilege to all the users in a table "users", the following Grant statement should be executed.

   GRANT SELECT ON Users TO '*'@'localhost;

   In the above example the "*" symbol is used to grant select permission to all the users of the table "users".

5. **Granting Privileges on Functions/Procedures**: While using functions and procedures, the Grant statement can be used to grant users the ability to execute the functions and procedures in MySQL.

Revoking Privileges from a Table

The Revoke statement is used to revoke some or all of the privileges which have been granted to a user in the past.

Syntax:

REVOKE privileges ON object FROM user;

Parameters Used

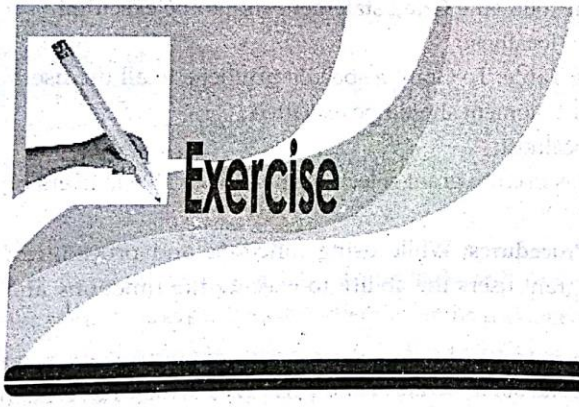
- **Object:** It is the name of the database object from which permissions are being revoked. In the case of revoking privileges from a table, this would be the table name.
- **User:** It is the name of the user from whom the privileges are being revoked.

Different ways of revoking privileges from a user

1. **Revoking SELECT Privilege to a User in a Table:** To revoke Select Privilege to a table named "users" where User Name is 'BHUPI', the following revoke statement should be executed.

   REVOKE SELECT ON users TO 'BHUPI'@localhost';

2. **Revoking more than Privilege to a User in a Table:** To revoke multiple Privileges to a user named "'BHUPI'" in a table "users", the following revoke statement should be executed.
   REVOKE SELECT, INSERT, DELETE, UPDATE ON
   Users TO 'BHUPI'@'localhost;
3. **Revoking All the Privilege to a User in a Table:** To revoke all the privileges to a user named 'BHUPI' in a table "users", the following revoke statement should be executed.
   REVOKE ALL ON Users TO 'BHUPI'@'localhost;
4. **Revoking a Privilege to all Users in a Table:** To revoke a specific privilege to all the users in a table "users", the following revoke statement should be executed.
   REVOKE SELECT ON Users TO '*'@'localhost;
5. **Revoking Privileges on Functions/Procedures**: While using functions and procedures, the revoke statement can be used to revoke the privileges from users which have been EXECUTE privileges in the past.
   Syntax:
   REVOKE EXECUTE ON [PROCEDURE |FUNCTION]
   Object FROM user;



1. What is basic structure of SQL? explain
2. What is relationship between SQL and relational algebra?
3. How you count the number of unique entries in a column?
4. What is main difference between WHERE and HAVING clause?
5. Define aggregate function.
6. What is view? How is view defined?
7. What is trigger? How they are differing from assertion?
8. What do you mean by domain integrity constraint? Why they are required in database?
9. What is use of check constrain?
10. Define referential integrity constraint with a suitable example.
11. How trigger is implemented in database explain with a suitable example.

12. List possible constraints used in the database.
13. What is the main use of primary key constraint in database?
14. How assertion is implemented in database explain with suitable example.'
15. What is the difference between the two SQL commands DROP TABLE and TRUNCATE TABLE?
16. What is integrity constraint? List different types of integrity constrains. Discuss domain constraint with suitable example.
17. What is mapping cardinality? Explain mapping cardinality constraint with suitable example.
18. Consider the following schema:

    Suppliers (sid: integer, sname: string, address: string)

    Parts (pid: integer, pname: string, color: string)

    Catalog (sid: integer, pid: integer, cost: real)

    a. Find the pnames of parts for which there is some supplier.
    b. Find the snames of suppliers who supply every part.
    c. Find the snames of suppliers who supply every red part.
    d. Find the pnames of parts supplied by Acme Widget Suppliers and by no one else.
    e. Find the sids of suppliers who charge more for some part than the average cost of that Part (averaged over all the suppliers who supply that part).
    f. Find the sids of suppliers who supply only red parts.
    g. Find the sids of suppliers who supply a red part and a green part.
    h. Find the sids of suppliers who supply a red part or a green part.

19. Write the Following Queries in SQL

    Department (dept_no,d_name, city)

    Employee (emp_Id, e_name, salary)

    Works (dept_no, emp_Id)

    a. Create the above tables by specifying the Primary key, Not NULL , Foreign key Constraints DDL Statement in MySQL database
    b. Write DML Statement to Insert any five records in each tables
    c. Display the name of the employees.
    d. Find the name of the employees whose salary is greater than 10000.
    e. Find the department (d_name) of the employee 'Binek'.

20. Write the Following Queries in SQL:

    Teacher (Tid, Tname, Address, Age)

    Student (Sid,Sname, Age, sex)

    Takes (sid, course-id)

    Course (course-id,course_name, text_book)

    Teaches (Tid, Cousrse-id)

    Taught-by{Sid, Tid}

Construct the following RA expressions for this relational database

a. Fine name, age and sex of all students who takes course "DBMS"

b. Find total number of students who are taught by the teacher "T01".

c. List all course names text books taught by teaher "T16"

d. Find average age of teachers for each course.

e. Insert the record of new teacher "T06" named "Bhupi" of age 27 into database who lives in "Balkhu" and takes course "DBMS"

❑❑❑

CHAPTER 7

RELATIONAL DATABASE DESIGN

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Relational Database Design Using ER-to-Relational Mapping
- Informal Design Guidelines for Relational Schemas
- Functional Dependencies
- Normal Forms Based don Primary Keys
- General Definitions of Second and Third Normal Forms; Boyce-Codd Normal Form;
- Multivalued Dependency and Fourth Normal Form
- Properties of Relational Decomposition

7.1 Relational Database Design Using ER-to-Relational Mapping

ER diagrams can be mapped to relational schema, that is, it is possible to create relational schema using ER diagram. We cannot import all the ER constraints into relational model, but an approximate schema can be generated. After designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL etc. In this unit we will discuss how to convert ER diagram to Relational Model for different scenarios.

To reduce given ER diagram into table simply we create a table for each entity set and for each relationship sets. And that assigned the name of the corresponding entity set or relationship set as table name. Generally the number of attribute of an entity set or relationship set equal to the degree of a corresponding table (fields of a table). To reduce given ER diagram into tables normally we divide ER diagram into following sections:

1. Mapping Strong entity sets to ER
2. Mapping Weak entity sets to ER
3. Mapping Relation sets to ER
   - Mapping of Binary 1:1 relation types to ER
   - Mapping of Binary 1:N relationship types to ER
   - Mapping of Binary M:N relationship types to ER
4. Mapping of multivalve attributes to ER
5. Mapping composite attributes to ER
6. Mapping of N-ary relationship types to ER
7. Mapping specialization/generalization to ER
8. Mapping Aggregation to ER

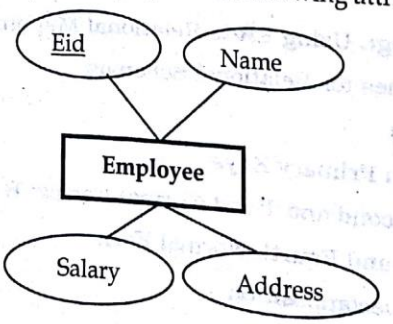
Mapping Strong entity sets to ER

For each regular (strong) entity type E in the ER schema, create a table T that includes all the simple attributes of E. Choose one of the key attributes of E as the primary key for T. Simply, an entity set is mapped to a relation in a straightforward way: Each simple attribute of the entity set becomes an attribute (column) of the table and primary key of the entity set becomes primary key of the relation.

Mapping Process

- Create table for each of the strong entity.
- Entity's attributes should become fields of tables with their respective data types.
- Declare key attribute of strong entity set as primary key of the table.

Example: Let's take a strong entity set Employee with following attributes;



Here we simply create a table "Employee" with fields Eid, Name, Salary and Address and make Eid as primary key of the created table as below,

Employee

Eid	Name	Salary	Address

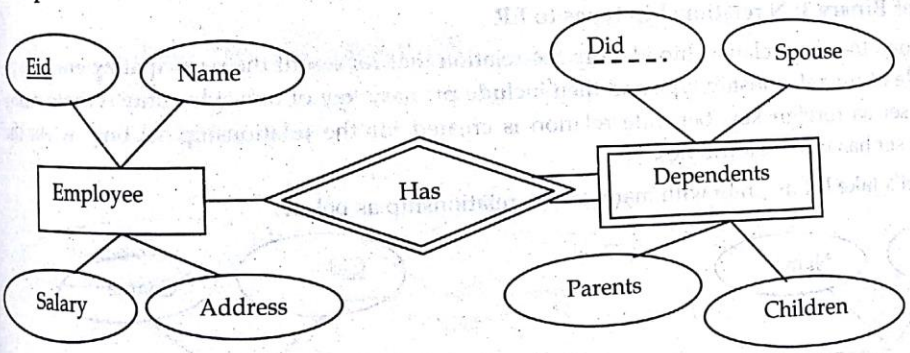
Mapping Weak entity sets to ER

A weak entity set does not have its own primary key and always participates in one-to-many relationship with owner entity set and has total participation. For a weak entity set create a relation that contains all simple attributes (or simple components of composite attributes). In addition, relation for weak entity set contains primary key of the owner entity set as foreign key and its primary key is formed by combining partial key (discriminator) and primary key of the owner entity set.

Mapping Process

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.
- Declare all foreign key constraints

Example: Let's take a weak entity set Dependents as shown in ER diagram below;



Here to draw table of weak entity set 'Dependents' we simply set all their attributes i.e. Did, spouse, Parents, children and also set primary key of Employee table to the Dependents table as below,

Employee

Eid	Name	Salary	Address

Dependents

Did	Did	Parents	Spouse	Children

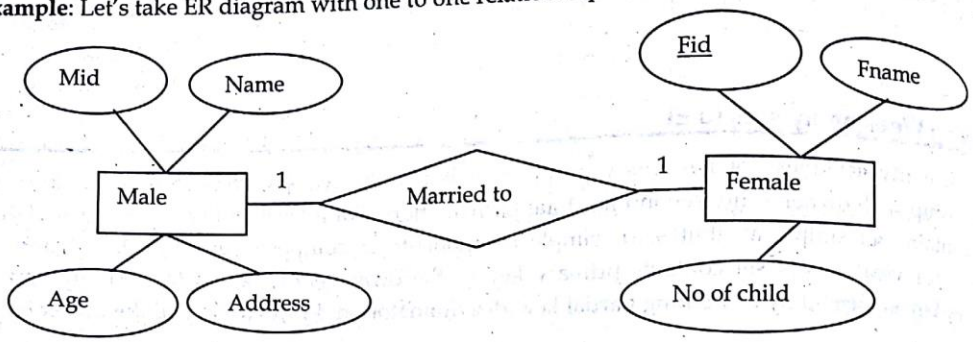
Mapping Relation sets to ER

To construct tables from given relationships we may have following types of relationships.

Mapping of Binary 1:1 relation types to ER

For constructing table from a binary one to one relationship we set primary key of any one of the entity set as foreign key. It is better to include foreign key into the entity set with total participation in the relationship.

Example: Let's take ER diagram with one to one relationship as below,



Here we can set mid as foreign key to Female table or Fid to Male table as their foreign key.

Male

Mid	Name	Age	Address

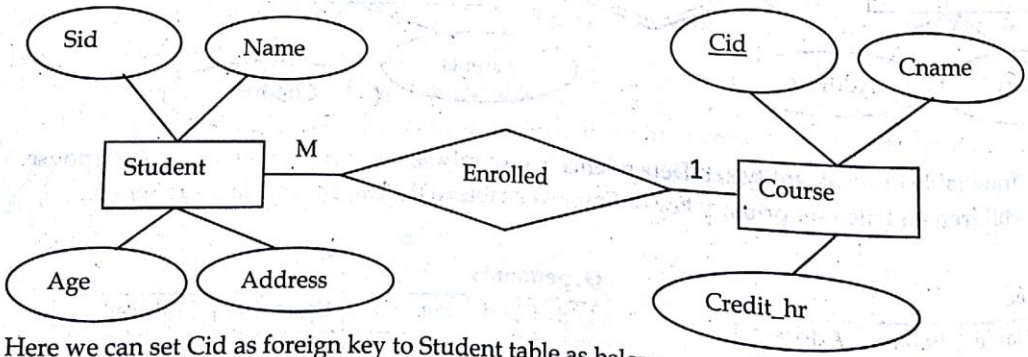
Female

Fid	Mid	Fname	No of child

Mapping of Binary 1: N relationship types to ER

For binary one-to-many relationship identify the relation that represent the participating entity type at the N-side of the relationship type and then include primary key of one side entity set into many side entity set as foreign key. Separate relation is created for the relationship set only when the relationship set has its own attributes.

Example: Let's take ER diagram with many to one relationship as below,



Here we can set Cid as foreign key to Student table as below,

Student

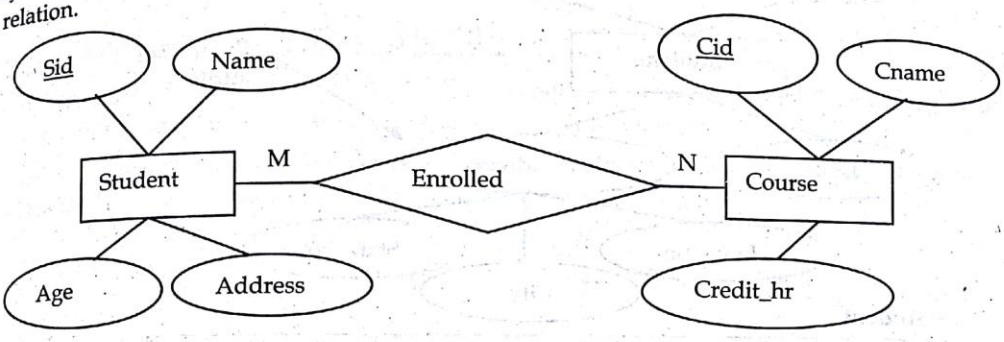
Sid	Name	Age	Address	Cid

Course

Cid	Cname	Credit_hr

Mapping of Binary M: N relationship types to ER

For a binary Many-to-Many relationship type, separate relation is created for the relationship type. Primary key for each participating entity set is included as foreign key in the relation and their combination will form the primary key of the relation. Besides this, simple attributes of the many-to-many relationship type (or simple components of composite attributes) is included as attributes of the relation.



Here we create a new table Enrolled that contains the primary keys of entities student and course entities as below,

Student

Sid	Name	Age	Address

Course

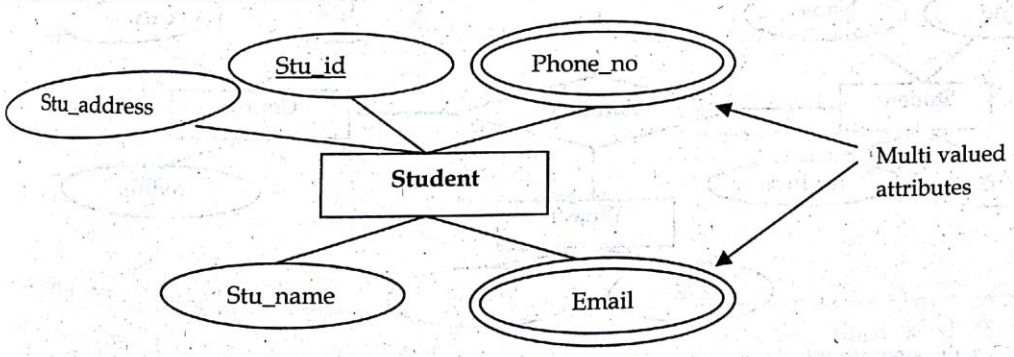
Cid	Cname	Credit_hr

Enrolled

Cid	Cid

Mapping of multivalve attributes to ER

If an entity has multivalued attribute, separate relation is created with primary key of the entity set and multivalued attribute itself.



Here we create separate table for multivalued attributes i.e. Email table and Phone table that contains the stu_id as their foreign key as below:

Student

Stu_id	Stu_name	Stu_address

Email

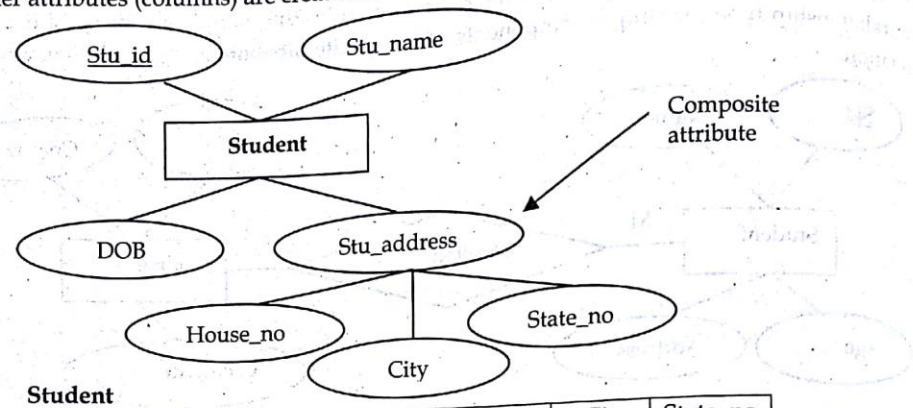
Stu_id	Email

Phone

Stu_id	Phone_no

Mapping composite attributes to ER

If an entity has composite attributes, no separate attribute (column) is created for composite attribute itself rather attributes (columns) are created for component attributes of the composite attribute.

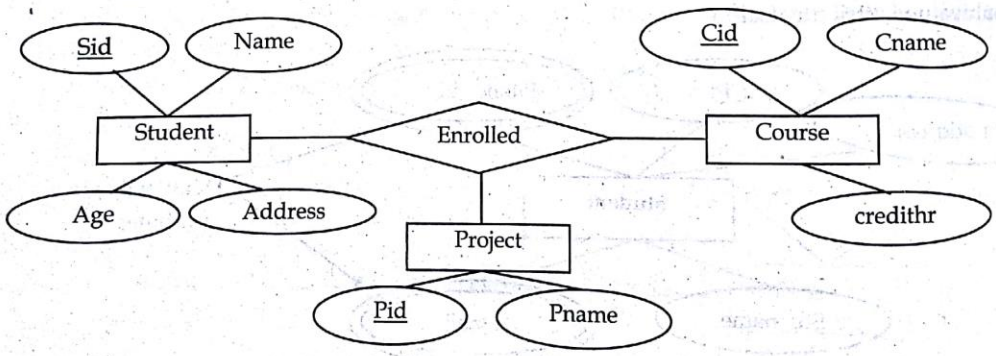


Student

Stu_id	Stu_name	DOB	House_no	Cit	State_no

Mapping of N-ary relationship types to ER

For each n-ary relationship set for n>2, a new relation is created. Primary keys of all participating entity sets are included in the relation as foreign key attributes. Besides this all simple attributes of the n-ary relationship set (or simple components of composite attributes) are included as attributes of the relation.



Here we create separate table for each entities and also create a single table for relationship Enrolled that contains the primary keys of each of the entities associated with this relationship.

Student

Sid	Name	Age	Address

Course

cid	Cname	cridithr

Proect

Pid	Pname

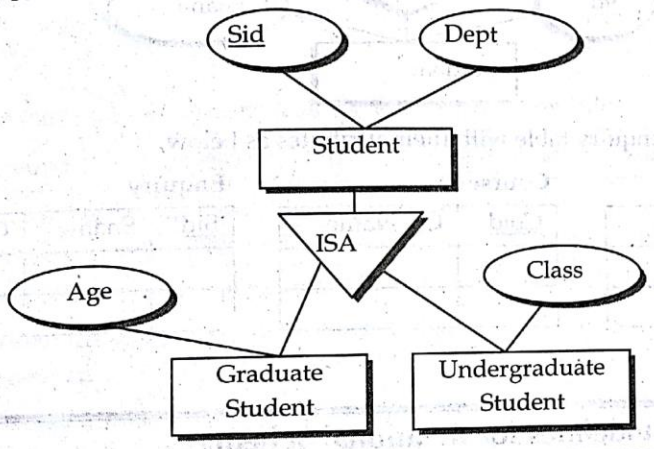
Enrolled

Sid	Cid	Pid

Mapping specialization/generalization to ER

To construct relational tables form given ER diagram with specialization/generalization we set primary key of the supper class to their sub classes as their foreign key.

If subclasses are disjoint and complete then relation for a subclass entity set includes all attributes of superclass entity set and all of its own attributes.



Here we set sid to Graduate student and Undergraduate Student tables as below

Student

Sid	Dept

Graduate student

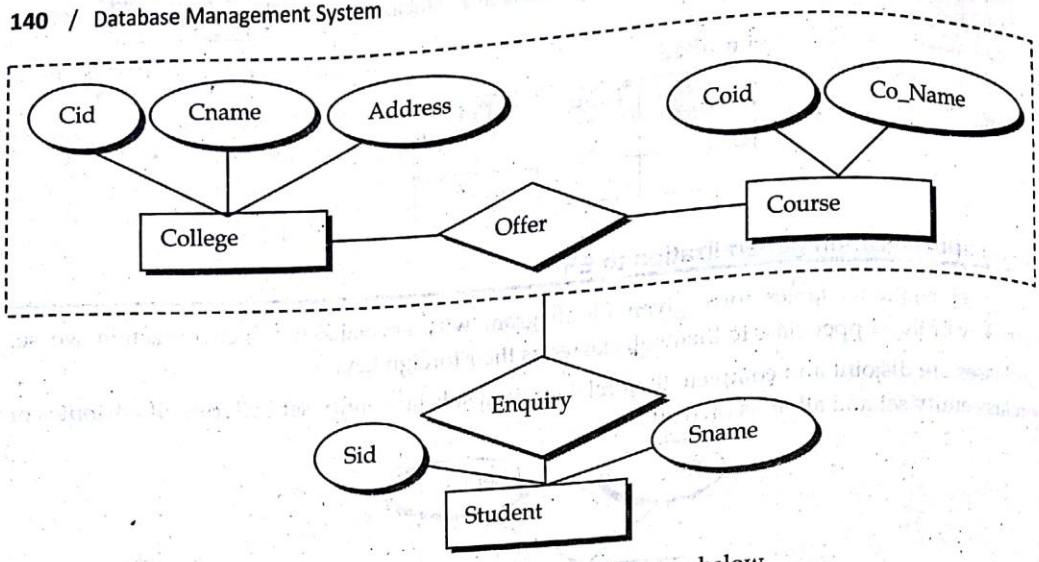
Sid	Age

Undergraduate students

Sid	Class

Mapping Aggregation to ER

Translating ER diagrams with aggregation into relation is easy because there is no distinction between entity sets and relationship sets in relational model. An EER diagram containing aggregation expresses relationship between a relationship set and an entity set. In relational model separate relation is created for this relationship and the relation contains primary key of associated entity set and the relationship set and its own attributes, if any.



Here we set cid and coid to enquiry table with their attributes as below,

College

Cid	cname	Address

Course

Coid	Co_Name

Enquiry

Sid	Sname	Cid	Coid

7.2 Informal Design Guidelines for Relational Schemas

Informal guidelines that may be used as measures to determine the quality of relation schema design are listed below:

- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

Making sure that the semantics of the attributes is clear in the schema

Whenever we are going to form relational schema there should be some meaning among the attributes. This meaning is called semantics. This semantics relates one attribute to another with some relation.

Guideline:

- Informally, each tuple in a relation should represent one entity or relationship instance.
- Attributes of different entities (relations) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible

Reducing the redundant information in tuples

If a table containing attributes of multiple entities may cause redundant information problems. Information is stored redundantly wasting storage. Problems with update anomalies

- Insertion anomalies
- Deletion anomalies
- Modification anomalies

We can reduce redundant information problem by using normalization. Normalization is the process of reducing a single table into a multiple simple tables.

Example: let's take two tables,

Student

Sid	Sname	Semester

Department

Dept_no	Dept_name	Address

If we integrate these two relations into a single table i.e. Student Info.

Sid	Sname	Semester	Dept_no	Dept_name	Address

- Here whenever if we insert the tuples there may be N students in one department, so Dept_no, Dept_name, Address values are repeated N times which leads to data redundancy.
- Another problem is update anomalies i.e. if we insert new department that has no students.
- If we delete the last student of a department, then whole information about the department will be deleted
- If we change the value of one of the attributes of a particular table then we must update the tuples of all the students belonging to that department else database will become inconsistent.

So design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

Reducing Null values in Tuples

In some schema designs we may group many attributes together into a "fat" relation. If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level.

Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied. SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable. Reasons for nulls:

- attribute not applicable or invalid
- attribute value unknown (may exist)
- value known to exist, but unavailable

As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation. Using space efficiently and avoiding joins with NULL values are

the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns (with the appropriate key columns). For example, if only 15 percent of employees have individual offices, there is little justification for including an attribute Office_number in the EMPLOYEE relation; rather, a relation EMP_OFFICES(Essn, Office_number) can be created to include tuples for only the employees with individual offices.

Relations should be designed such that their tuples will have as few NULL values as possible. Attributes that are NULL frequently could be placed in separate relations (with the primary key)

Disallowing the possibility of generating spurious tuples

Bad designs for a relational database may result in erroneous results for certain JOIN Operations the "lossless join" property is used to guarantee meaningful results for join operations. The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural-join of any relations.

Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

7.3 Problems without Normalization

If a table is not properly normalized and has data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, updating and Deletion Anomalies are very frequent if database is not normalized. To understand these anomalies let us take an example of a Student table.

Student

Sid	Sname	Branch	HOD	Office Phone
100	Abin	Pokhara branch	Mr. Nabin	4434564
101	Aarav	Pokhara branch	Mr. Nabin	4434564
102	Ashana	Pokhara branch	Mr. Nabin	4434564
103	Geeta	Pokhara branch	Mr. Nabin	4434564
104	Umesh	Banepa branch	Mr. Prabin	5212343
105	Ramesh	Banepa branch	Mr. Prabin	5212343

In the table above, we have data of 6 Computer Sci. students. As we can see, data for the fields branch, HOD and Office_Phone is repeated for the students who are in the same branch in the college, this is Data Redundancy.

Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.

Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students. These scenarios are nothing but Insertion anomalies.

Update Anomaly

What if Mr. Prabin leaves the college? Or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updating anomaly.

Deletion Anomaly

In our Student table, two different information are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

7.4 Functional Dependencies

Functional dependency plays a key role in differentiating good database design from bad database design. Functional dependency is a relationship that exists when one attribute uniquely determines another attribute. If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X→Y, which specifies Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute. Each value of X is associated with precisely one Y value.

Functional dependency in a database serves as a constraint between two sets of attributes. Defining functional dependency is an important part of relational database design and contributes to aspect normalization.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id→Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

7.5 Types of functional dependency

There are many types of functional dependencies, depending on several criteria

- Trivial and Non-trivial dependencies
- Fully functional dependency
- Partial functional dependency
- Transitive dependency
- Multivalued dependency

Trivial functional dependency

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute. Symbolically: A ->B is trivial functional dependency if B is a subset of A. The following dependencies are also trivial:

- A→A
- B→B

For example: Consider a table with two columns Student_id and Student_Name. {Student_Id, Student_Name} →S tudent_Id is a trivial functional dependency as Student_Id is a subset of

{Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined. Also, Student_Id→Student_Id & Student_Name→Student_Name are trivial dependencies too.

Non trivial functional dependency

If a functional dependency X→Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency

For example

An employee table with three attributes: {emp_id, emp_name, emp_address}

The following functional dependencies are non-trivial:

emp_id → emp_name (emp_name is not a subset of emp_id)

emp_id → emp_address (emp_address is not a subset of emp_id)

Fully Functional Dependency

An attribute is fully functionally dependent on a second attribute if and only if it is functionally dependent on the second attribute but not on any subset of the second attribute. Mathematically, for a relation schema R, in a functional dependency X→Y, Y is said to be fully functional dependent on X if Z→Y is false for all Z ⊂ X.

Partial Functional Dependency

An attribute is partial functionally dependent on a second attribute if and only if it is functionally dependent on the second attribute and also dependency occur on any subset of the second attribute. This is the situation that exists if it is necessary to only use a subset of the attributes of the composite determinant to identify its object. Mathematically, for a relation schema R, in a functional dependency X→Y, Y is said to be partial functional dependent on X if by removal of some attributes from X and the dependency still holds.

Example: The dependency {Emp-Id, Projecta-No} → {Emp-Name} is partial because Emp-Id→Emp-Name also holds.

Transitive dependency

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For example, X →Z is a transitive dependency if the following three functional dependencies hold true:

- X→Y
- Y→X
- Y→Z

Example: Let's take an example to understand it better

Movie_ID	Listing_ID	Listing_Type	DVD_Price ($)
M08	L09	Crime	180
M03	L05	Drama	250
M05	L09	Crime	180

Movie_ID →Listing_ID
Listing_ID →Listing_Type
Also, Movie_ID →Listing_Type

Multivalued dependency

Multivalued dependency occurs when there are more than one independent multivalued attributes in a table. For example: Consider a bike manufacture company, which produces two colors (Black and white) in each model every year.

Bike_model	Manuf_year	Color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

Here columns manuf_year and color are independent of each other and dependent on bike_model. In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:

bike_model ->> manuf_year

7.6 Introduction to Normalization

When developing the schema of a relational database, one of the most important aspects to be taken into account is to ensure that the duplication is minimized. This is done for two purposes:

- Reducing the amount of storage needed to store the data.
- Avoiding unnecessary data conflicts that may creep in because of multiple copies of the same data getting stored.

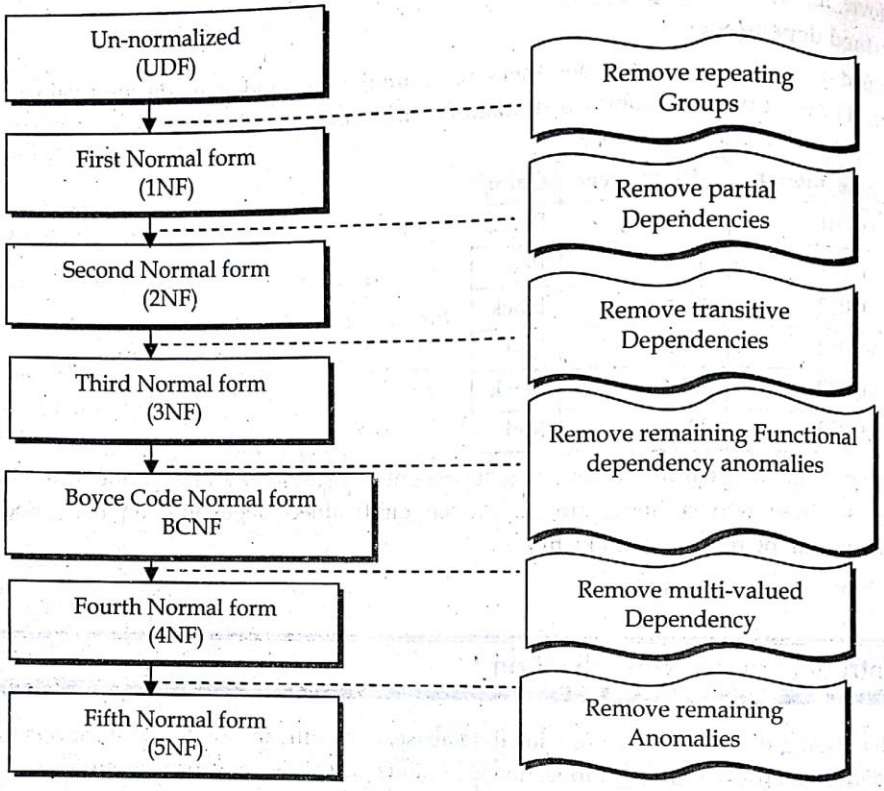
Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and update anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data. It divides larger tables to smaller tables and links them using relationships.

7.7 Types of Normalization

The degree of normalization is defined by normal forms. The normal forms in an increasing level of normalization are first normal form (1NF), second normal form (2NF), 3NF, Boyce-Codd normal form, 4NF and 5NF. Each normal form is a set of conditions on a schema that guarantees certain properties relating to redundancy and update anomalies. In general 3NF is considered good enough. In certain instances, a lower level of normalization, that is the instance where queries take enormous time to execute.

There are various database "Normal" forms. Each normal form has an importance which helps in optimizing the database to save storage and to reduce redundancies.



7.8 Normal Forms Based on Primary Keys

The normal forms first, second and third are called primary key based normal forms. Here each of the non-prime attributes of the relation depends upon the prime attribute of the relation.

First Normal Form

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute.

Simply, A relation R is in 1 NF if it holds following two properties;

- If R has no multivalued attributes
- If R has no composite attributes i.e. it must have simple attributes

Example: Relation **Student** in table below is not in 1NF because of multi-valued attribute Phone. Its decomposition into 1NF has been shown in table 2.

Sid	Sname	Phone	State	Age
1	Ram	9848372632, 8905454342	State 1	23
2	Hari	9851093482	State 2	33
3	Ramesh	9848372098 9851123032	State 1	22
4	Anand	9804094302	State 3	29

Now convert given relation into 1NF as below,

Student

Sid	Sname	State	Age
1	Ram	State 1	23
2	Hari	State 2	33
3	Ramesh	State 1	22
4	Anand	State 3	29

Phone_detail

Sid	Phone
1	9848372632
1	8905454342
2	9851093482
3	9848372098
3	9851123032
4	9804094302

Second normal form (2NF)

A relation R (table) is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attributes of R fully dependents upon the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute. Simply a relation is in 2 NF if there are no partial dependencies between the primary key and non-prime keys of given relation R.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

Teacher

Tid	Tsubject	Tage
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {Tid, Tsubject}

Non prime attribute: Tage

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute Tage is dependent on Tid alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "no non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:

teacher_details table

Tid	Tage
111	38
222	38
333	40

teacher_subject table

Tid	Tsubject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- No transitive functional dependency between the attributes of given table.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X→Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	State 1	Dhankuta	Jhapa
1002	Ajeet	222008	State 2	Janakpur	Sarlahi
1006	Lora	282007	State 2	Janakpur	Sarlahi
1101	Lilly	292008	State 3	Kathmandu	Ktm
1201	Steve	222999	State 7	Dhangadhi	Kailali

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}...so on
Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF. To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

Employee table

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

Employee_zip table

emp_zip	emp_state	emp_city	emp_district
282005	State 1	Dhankuta	Jhapa
222008	State 2	Janakpur	Sarlahi
282007	State 2	Janakpur	Sarlahi
292008	State 3	Kathmandu	Ktm
222999	State 7	Dhangadhi	Kailali

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X→Y, X should be the super key of the table.

Mathematically,

A relation R is said to be in BCNF with respect to all the functional dependencies of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, if at least one of the following holds:

- $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
- α is a candidate key for schema R.

Example: Suppose there is a company wherein employees work in more than one department. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Nepalese	Production and planning	D001	200
1001	Nepalese	Store department	D001	250
1002	Indian	Technical support	D134	100
1002	Indian	Purchasing department	D134	600

Functional dependencies in the above table are:

emp_id→emp_nationality

emp_dept→{dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table

emp_id	emp_nationality
1001	Nepalese
1002	Indian

emp_dept_mapping table

emp_id	emp_dept
1001	Production and planning
1001	Store department
1002	Technical support
1002	Purchasing department

emp_dept table

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
Store department	D001	250
Technical support	D134	100
Purchasing department	D134	600

Functional dependencies:

emp_id → emp_nationality

emp_dept → {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id

For second table: emp_dept

For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

Differences between 3NF and BCNF

3NF	BCNF
A table or a relation is considered to be in third normal form only if the table is already in 2NF and there is no non-prime attribute transitively dependent on the candidate key of a relation.	BCNF is considered to be the stronger than 3NF. The relation R to be in BCNF must be in 3NF. And wherever a non-trivial functional dependency A→B holds in relation R, then A must be a super key of relation R.
No non-prime attribute must be transitively dependent on the Candidate key.	For any trivial dependency in a relation R say X→Y, X should be a super key of relation R.
3NF can be obtained without sacrificing all dependencies.	Dependencies may not be preserved in BCNF.
Lossless decomposition can be achieved in 3NF.	Lossless decomposition is hard to achieve in BCNF.

Fourth Normal Form

A table R is in fourth normal form (4NF) if and only if it satisfied following two conditions simultaneously:

- R is already in BCNF or in 3NF
- If it contains no multi-valued attributes

Equivalently, we can that a relation schema R is in 4NF if, for every nontrivial multi-valued dependency X –>> Y, X is a super key for R. The goal of fourth normal form is to eliminate nontrivial multi-valued dependencies from a table by projecting them onto separate smaller tables, thus eliminating the update anomalies associated with the multi-valued dependencies.

Example 1: Consider the following relation subject which has the attributes course, instructor, and textbook.

Subject (course, instructor, Text-book)

Course	Instructor	Text-Book
CS1	Indra	DBMS
CS1	Bhupi	DBMS
CS2	Arjun	NM
CS2	Bhupendra	NM
CS3	Indra	CG
CS3	Bhupi	CG
CS3	Arjun	CG

In this relation for each course there are many instructors and also for each course there are many text-books. But instructor and text-book are independent of each other. Thus, the given relation Subject is not in fourth normal form because of multi-valued dependency between attributes.

Course –>> instructor

Course –>> textbooks

The relation subject can be converted to fourth normal form by splitting the relation Subject into two relations Teacher and Text as below

Teacher (Course, Instructor)

Text (Course, Textbook)

Example 2: Consider the relation Employee with the attributes employee number, project name, and department name as shown below:

Employee (Eno, Pname, Dname)

Where, Eno stands for Employee number, Pname for project name, and Dname for department name. The relation Employee has the following multi-valued dependencies:

Eno –>> Pname

Eno –>> Dname

Since, Eno is not the super key of the relation Employee so it is not in 4NF. To convert the relation to 4 NF, we can decompose Employee relation into two relations Emp-Proj and Emp- Dept as shown below.

Emp-proj (Eno, Pname)

Emp-dept (Eno, Dname)

7.9 What is decomposition?

Decomposition is the process of breaking down in parts or elements. It replaces a relation with a collection of smaller relations. It breaks the table into multiple tables in a database. It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations. If there is no proper decomposition of the relation, then it may lead to problems like loss of information.

Properties of Relational Decomposition

Following are the properties of Decomposition,

1. Lossless Decomposition
2. Dependency Preservation
3. Lack of Data Redundancy

Lossless Decomposition

Decomposition must be lossless. It means that the information should not get lost from the relation that is decomposed. It gives a guarantee that the join will result in the same relation as it was decomposed.

Example: Let's take 'E' is the Relational Schema, with instance 'e'; is decomposed into: E1, E2, E3 ... En; with instance: e1, e2, e3 . . . en, If e1 ⋈ e2 ⋈ e3 . . . . ⋈ en, then it is called as 'Lossless Join Decomposition'.

In the above example, it means that, if natural joins of all the decomposition give the original relation, then it is said to be lossless join decomposition.

Example: Employee_Department

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Ktm	20000	D001	Finance
E002	PQR	30	Ktm	30000	D002	Production
E003	LMN	25	Banepa	5000	D003	Sales
E004	XYZ	24	Banepa	4000	D004	Marketing
E005	STU	32	Pokhara	25000	D005	Human Resource

Decompose the above relation into two relations to check whether decomposition is lossless or lossy.

Now, we have decomposed the relation that is Employee and Department.

Relation 1: Employee

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

Employee Schema contains (Eid, Ename, Age, City, Salary)

Relation 2: Department

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- Department Schema contains (Deptid, Eid, DeptName).
- So, the above decomposition is a Lossless Join Decomposition, because the two relations contain one common field that is 'Eid' and therefore join is possible.
- Now apply natural join on the decomposed relations.

Employee ⋈ Department

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing
E005	STU	32	Bangalore	25000	D005	Human Resource

Hence, the decomposition is Lossless Join Decomposition.

If the Employee table contains (Eid, Ename, Age, City, Salary) and Department table contains (Deptid and DeptName), then it is not possible to join the two tables or relations, because there is no common column between them. And it becomes Lossy Join Decomposition.

Dependency Preservation

- Dependency is an important constraint on the database.
- Every dependency must be satisfied by at least one decomposed table.
- If {A → B} holds, then two sets are functional dependent. And, it becomes more useful for checking the dependency easily if both sets in a same relation.
- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.

Lack of Data Redundancy

- Lack of Data Redundancy is also known as a Repetition of Information.
- The proper decomposition should not suffer from any data redundancy.
- The careless decomposition may cause a problem with the data.
- The lack of data redundancy property may be achieved by Normalization process.

7.10 Functional Dependency Inference Rules (Armstrong's Axioms)

Reflexivity

If Y ⊆ X then X→Y. The axiom of reflexivity indicates that given a set of attributes the set itself functionally determines any of its own subsets.

Augmentation

If X→Y and Z is a subset of table R (i.e., Z is any set of attributes in R), then XZ→YZ. The axiom of augmentation indicates that we can augment the left side of the functional dependency or both sides conveniently with one or more attributes. The axiom does not allow augmenting the right-hand side alone. The augmentation rule can be represented as If X→Y then XZ→Y

Transitivity

If X→Y and Y→Z then X→Z. The axiom of transitivity indicates that if one attribute uniquely determines a second attribute and this, in turn, uniquely determines a third one, then the first attributes determines the third one. Consider three parallel lines X, Y, and Z. The line X is parallel to line Y. The line Y is parallel to line Z then it implies that line X is parallel to line Z. This property is called transitivity.

Pseudo transitivity

If X→Y and YW→Z then XW→Z. Transitivity is a special case of pseudo transitivity when W is null. The axiom of pseudo transitivity is a generalization of the transitivity axiom.

Union

If X→Y and X→Z then X→YZ. The axiom of union indicates that if there are two functional dependencies with the same determinant it is possible to form a new functional dependency that preserves the determinant and has its right-hand side the union of the right-hand sides of the two functional dependencies. The union rule can be illustrated with the example of PINCODE. The PINCODE is used to identify city as well as PINCODE is used to identify state. This implies that PINCODE determines both city and state.

Decomposition

If X→YZ then X→Y and X→Z. The axiom of decomposition indicates that the determinant of any functional dependency can uniquely determine any individual attribute or any combination of attributes of the right-hand side of the functional dependency. The decomposition can be illustrated with an example of Book ID. The Book ID determines the title and the author similar to (X→YZ) which implies that Book ID determines title (X→Y) and Book ID determines Author (X→Z)

Closure of Set of Functional Dependencies

The Closure of Functional Dependency means the complete set of all possible attributes that can be functionally derived from given functional dependency using the inference rules known as Armstrong's Rules. If "F" is a functional dependency then closure of functional dependency can be denoted using "{F}+". There are three steps to calculate closure of functional dependency. These are:

- **Step-1:** Add the attributes which are present on Left Hand Side in the original functional dependency.
- **Step-2:** Now, add the attributes present on the Right Hand Side of the functional dependency.

- **Step-3:** With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes which can be derived are added in the closure.

Algorithm: Determining X+, the closure of X under F.

Input: Let F be a set of FDs for relation R.

Steps:

1. X+ = X //initialize X+ to X
2. for each FD: Y → Z in F Do
   If Y ⊆ X+ Then //If Y is contained in X+
   X+ = X+ ∪ Z //add Z to X+
   End If
   End For
3. Return X+ //Return closure of X

Output: Closure X+ of X under F

Example 1: Let R = (A, B, C, D)

F= {A→B, A→C, BC→D}. List several member of F+

Solution:

Since A→B, A→C then A→BC [by union rule]

Since A→BC, BC→D then A→D [by transitive rule]

Since A→C, BC→D then AB→D [by pseudo transitivity rule]

Hence F+ = {A→B, A→C, BC→D, A→BC, A→D, AB→D}

Example 2: Let R = (A, B, C, G, H, I)

Given F= {A→B, A→C, CG→H, CG→I, B→H}, Compute closure of F (F+).

Solution:

Since A→B, B→H then A→H [by transitivity rule]

Since A→B, A→C then A→BC [by union rule]

Since A→C then AG→CG [by augmentation rule]

Since AG→CG, CG→I then AG→I [by transitivity rule]

Since CG →H, CG→I then CG→HI [by union rule]

Hence, F+ = {A→A, B→B, C→C, H→H, G→G, I→I, A→B, A→C, CG→H, G→I, CG→HI, B→H, A→H, AG→I, CG→HI}

Here, first six FDs obtain by reflexive axiom.

Example 3: Let R = (A, B, C, D) and F= {A→B, A→C, BC→D} then compute F+.

Solution:

Since A→B and A→C then by union rule A→BC.

Since BC →D, then by decomposition B→D, C→D. Again by transitivity A→B &B →D ⇒ A→D and A→C and C→D ⇒ A→D.

Hence, F+ = {A→A, B→B, C→C, D→D, A→B, A→C, BC→D, B→D, C→D, A→D}

7.11 Closure of sets attributes with functional dependency

The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set. Closure of attribute set {X} is denoted as {X}+.

Steps to Find Closure of an Attribute Set

Following steps are followed to find the closure of an attribute set-

Step 1: Add the attributes contained in the attribute set for which closure is being calculated to the result set.

Step 2: Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Example 1: Consider a relation R (A, B, C, D, E, F, G) with the functional dependencies

$A \rightarrow BC$

$BC \rightarrow DE$

$D \rightarrow F$

$CF \rightarrow G$

Now, let us find the closure of some attributes and attribute sets-

Closure of attribute A:

$A+ = \{A\}$

$= \{A, B, C\}$ (Using $A \rightarrow BC$)

$= \{A, B, C, D, E\}$ (Using $BC \rightarrow DE$)

$= \{A, B, C, D, E, F\}$ (Using $D \rightarrow F$)

$= \{A, B, C, D, E, F, G\}$ (Using $CF \rightarrow G$)

Thus,

$A+ = \{A, B, C, D, E, F, G\}$

Closure of attribute D:

$D+ = \{D\}$

$= \{D, F\}$ (Using $D \rightarrow F$)

We cannot determine any other attribute using attributes D and F contained in the result set.

Thus,

$D+ = \{D, F\}$

Closure of attribute set {B, C}:

$\{B, C\}+ = \{B, C\}$

$= \{B, C, D, E\}$ (Using $BC \rightarrow DE$)

$= \{B, C, D, E, F\}$ (Using $D \rightarrow F$)

$= \{B, C, D, E, F, G\}$ (Using $CF \rightarrow G$)

Thus,

$\{B, C\}+ = \{B, C, D, E, F, G\}$

Example 2: Let R = {A, B, C, D, E} and F= {AB→C, A→D, D→E, AC→B} then compute (AB) +

Solution:

result = {AB}

AB→C hence result = {ABC}

A→D hence result = {ABCD}

D→E hence result = {ABCDE}

AC→B [B is already in the result]

So result= {ABCDE}

Compute result for {A} + for above example:

Result = {A}

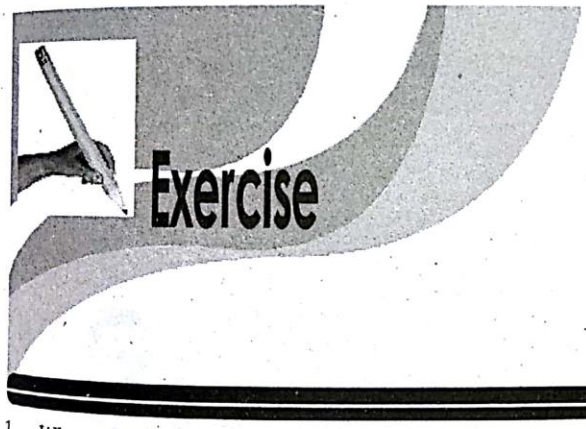
AB→C [no change] result = {A}

A→D hence result = {AD}

D→E hence result = {ADE}

AC→B [no change] result = {ADE}

So result= {ADE}



1. Why we need normalization? Explain.
2. Describe the purpose of normalizing the data? Describe the types of anomalies that may occur on a relation that has redundant data?
3. Given a relation R(S, B, C, D) with key={S B D, CBD} and F={S → C}. Identify the normal form of the relation R?
4. Give an example of a relation which is in 3NF but not in BCNF? How will you convert that relation to BCNF?
5. Discuss un-normalized form with example.
6. Explain types of functional dependency with example.
7. Compare trivial and non-trivial dependency
8. What do you mean by closure of a set of functional dependencies?
9. Compare fully and partially functional dependency.
10. Define non-loss decomposition.

11. What undesirable dependencies are avoided when a relation is in 3NF?
12. In what way do the generalized definitions of 2NF and 3NF extend the definitions beyond primary keys?
13. Define Boyce-Codd normal form. How does it differ from 3NF? Why is it considered a stronger form of 3NF?
14. What is multivalued dependency? When does it arise?
15. Does a relation with two or more columns always have an MVD? Show with an example.
16. Define fourth normal form. When is it violated? When is it typically applicable?
17. Define join dependency and fifth normal form.
18. Why 5NF is also called project-join normal form (PJNF)?
19. Why do practical database designs typically aim for BCNF and not aim for higher normal forms?
20. How can you mapping ER diagram with many to many relationship to relational model? Explain with suitable example.

❑❑❑

CHAPTER 8

INTRODUCTION TO TRANSACTION PROCESSING CONCEPTS AND THEORY

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Introduction to Transaction Processing
- Transaction and System Concepts
- Desirable Properties of Transactions
- Characterizing Schedules Based on Recoverability
- Characterizing Schedules Based on Serializability

8.1 Introduction

The concept of transaction provides a mechanism for describing logical units of database processing. **Transaction processing systems** are systems with large databases and hundreds of concurrent users executing database transactions. Examples of such systems include airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications. These systems require high availability and fast response time for hundreds of concurrent users. In this chapter, we present the concepts that are needed in transaction processing systems. We define the concept of a transaction, which is used to represent a logical unit of database processing that must be completed in its entirety to ensure correctness. A transaction is typically implemented by a computer program that includes database commands such as retrievals, insertions, deletions, and updates.

8.2 Introduction to Transaction Processing

Single user Versus Multiuser System

One criterion for classifying a database system is according to the number of users who can use the system concurrently. A DBMS is single-user if at most one user at a time can use the system, and it is multiuser if many users can use the system—and hence accesses the database—concurrently. Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser. For example, an airline reservations system is used by hundreds of travel agents and reservation clerks concurrently. Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems. In these systems, hundreds or thousands of users are typically operating on the database by submitting transactions concurrently to the system.

Multiple users can access databases—and use computer systems—simultaneously because of the concept of multiprogramming, which allows the operating system of the computer to execute multiple programs—or processes—at the same time. A single central processing unit (CPU) can only execute at most one process at a time. However, multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved.

If the computer system has multiple hardware processors (CPUs), **parallel processing** of multiple processes is possible. Most of the theory concerning concurrency control in databases is developed in terms of interleaved concurrency. In a multiuser DBMS, the stored data items are the primary resources that may be accessed concurrently by interactive users or application programs, which are constantly retrieving information from and modifying the database.

Transactions

A transaction is a unit of program execution that accesses and possibly updates various data items. A transaction includes one or more database access operations - these can include insertion, deletion, modification, or retrieval operations. Usually, a transaction is initiated by a user program written in a

high-level data-manipulation language (typically SQL), or programming language (for example, C++, or Java), with embedded database accesses in JDBC or ODBC. A transaction is delimited by statements (or function calls) of the form **begin transaction** and **end transaction**. The transaction consists of all operations executed between the **begin transaction** and **end transaction**. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**; otherwise it is known as a **read-write transaction**.

Database Items

A **data item** can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database. The transaction processing concepts are independent of the data item granularity (size) and apply to data items in general. Each data item has a unique name, but this name is not typically used by the programmer; rather, it is just a means to uniquely identify each data item. For example, if the data item granularity is one disk block, then the disk block address can be used as the data item name. Using simplified database model, the basic database access operations that a transaction can include are as follows:

- **read_item(X)**: Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
- **write_item(X):** Writes the value of program variable X into the database item named X.

Read and Write Operations and DBMS Buffers

Basic unit of data transfer from the disk to the computer main memory is one block. In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole block.

Executing read_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Executing write_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

It is step 4 that actually updates the database on disk. In some cases the buffer is not immediately stored to disk, in case additional changes are to be made to the buffer. Usually, the decision about when to store a modified disk block whose contents are in a main memory buffer is handled by the recovery manager of the DBMS in cooperation with the underlying operating system. The DBMS will maintain in the database cache a number of data buffers in main memory. Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed. When these buffers are all occupied, and additional database disk blocks must be copied into

memory, some buffer replacement policy is used to choose which of the current buffers is to be replaced. If the chosen buffer has been modified, it must be written back to disk before it is reused.

A transaction includes **read_item** and **write_item** operations to access and update the database. Figure 8.1 shows examples of two very simple transactions. The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that the transaction writes. For example, the read-set of T_1 in Figure 8.1 is {X, Y} and its write-set is also {X, Y}.

(a)

T_1
read_item(X);
X=X - N;
write_item(X);
read_item(Y);
Y=Y+N;
write_item(Y);

(b)

T_2
read_item(X);
X=X+M;
write_item(X);

Figure 8.1 Two sample transaction (a) Transaction T_1 (b) Transaction T_2

Concurrency control and recovery mechanisms are mainly concerned with the database commands in a transaction. Transactions submitted by the various users may execute concurrently and may access and update the same database items. If this concurrent execution is uncontrolled, it may lead to problems, such as an inconsistent database.

Why concurrency control is needed?

Several problems can occur when concurrent transactions execute in an uncontrolled manner. We illustrate some of these problems by referring to a much simplified airline reservations database in which a record is stored for each airline flight. Each record includes the number of reserved seats on that flight as a named (uniquely identifiable) data item, among other information. The types of problems we may encounter with these two simple transactions if they run concurrently.

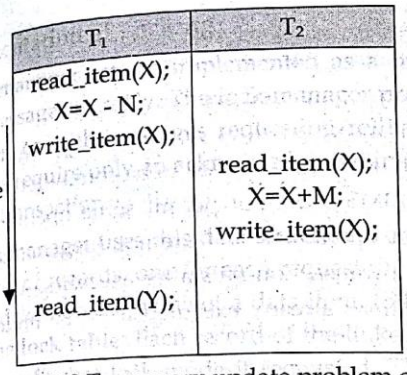
The Lost Update Problem: This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

Time ↓

T_1	T_2	
read_item(X);		
X=X - N;		
	read_item(X);	
	X=X+M;	
write_item(X);		
read_item(Y);		
	write_item(X);	← Item X has an incorrect value because its update by T_1 is **lost** (overwritten).
Y=Y+N;		
write_item(Y);		

Figure 8.2 Lost update problem occur during uncontrolled concurrent execution

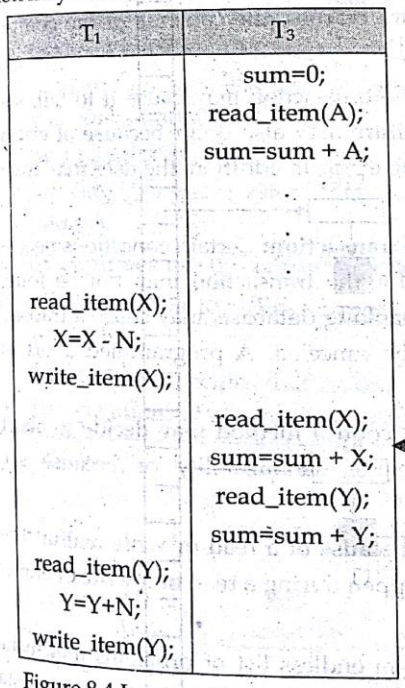
The Temporary Update (or Dirty Read) Problem: This occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.



Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the **temporary** incorrect value of X.

Figure 8.3 Temporary update problem occur during uncontrolled concurrent execution

The **Incorrect Summary Problem:** If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.



T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

Figure 8.4 Incorrect summary problem occur during uncontrolled concurrent execution

The **Unrepeatable Read Problem:** Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction between the two reads. Hence, T receives different values for its two reads of the same item. This may occur, for example, if during an airline reservation transaction, a customer inquires about seat availability on several flights. When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up reading a different value for the item.

Why Recovery is needed?

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or that the transaction does not have any effect on the database or any other transactions. In the first case, the transaction is said to be committed, whereas in the second case, the transaction is aborted. The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not, because the whole transaction is a logical unit of database processing. If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

Types of Failures

Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. **A computer failure (system crash):** A hardware or software error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.
2. **A transaction or system error:** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.
3. **Local errors or exception conditions detected by the transaction:** Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal from that account, to be canceled. A programmed abort in the transaction causes it to fail.
4. **Concurrency control enforcement:** The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.
5. **Disk failure:** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
6. **Physical problems and catastrophes:** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Failures of types 1, 2, 3, and 4 are more common than those of types 5 or 6. Whenever a failure of type 1 through 4 occurs, the system must keep sufficient information to quickly recover from the failure. Disk failure or other catastrophic failures of type 5 or 6 do not happen frequently; if they do occur, recovery is a major task. The concept of transaction is fundamental to many techniques for concurrency control and recovery from failures.

8.3 Transaction and System Concepts

Transaction States and Additional Operations

A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits or aborts. Therefore, the recovery manager of the DBMS needs to keep track of the following operations:

- **BEGIN_TRANSACTION:** This marks the beginning of transaction execution.
- **READ or WRITE:** These specify read or write operations on the database items that are executed as part of a transaction.
- **END_TRANSACTION:** This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates concurrency control or for some other reason.
- **COMMIT_TRANSACTION:** This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- **ROLLBACK (or ABORT):** This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Recovery techniques use the following operators:

- **UNDO:** Similar to rollback except that it applies to a single operation rather than to a whole transaction.
- **REDO:** This specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.

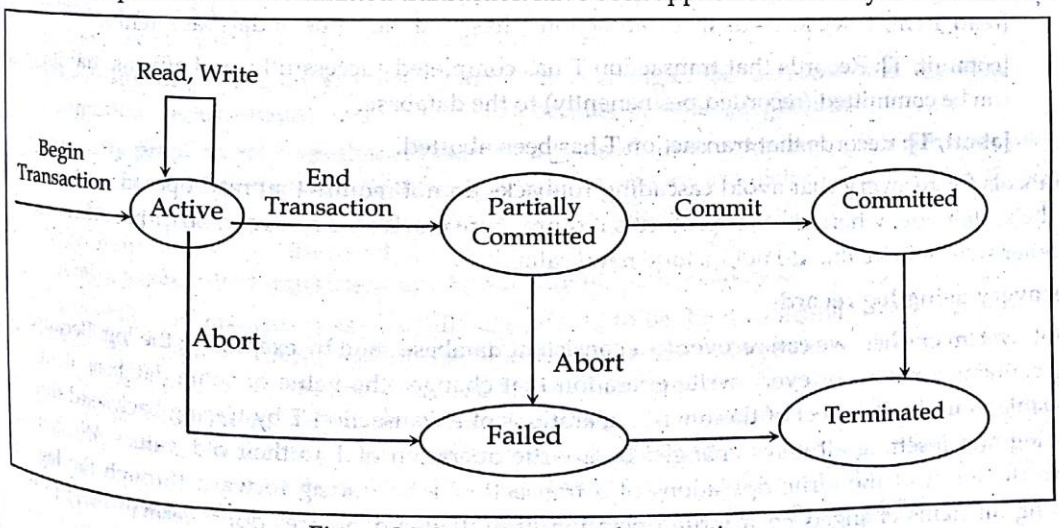


Figure 8.5 State diagram of transaction

Figure 8.5 shows a state transition diagram that illustrates how a transaction moves through its execution states. A transaction goes into an active state immediately after it starts execution, where it can execute its READ and WRITE operations. When the transaction ends, it moves to the partially committed state. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log, discussed in the next section). Once this check is successful, the transaction is said to have reached its commit point and enters the committed state. When a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database, even if a system failure occurs. However, a transaction can go to the failed state if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database. The terminated state corresponds to the transaction leaving the system. The transaction information that is maintained in system tables while the transaction has been running is removed when the transaction terminates. Failed or aborted transactions may be restarted later—either automatically or after being resubmitted by the user—as brand new transactions.

The System Log

The log (or journal) keeps track of all transaction operations that affect the values of database items. This information may be needed to permit recovery from transaction failures. The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure. In addition, the log is periodically backed up to archival storage (tape) to guard against such catastrophic failures. The following are the types of entries called **log records** that are written to the log file and the corresponding action for each log record. In these entries, T refers to a unique **transaction-id** that is generated automatically by the system for each transaction and that is used to identify each transaction:

1. **[start_transaction, T]:** Records that transaction T has started execution.
2. **[write_item, T, X, old_value, new_value]:** Records that transaction T has changed the value of database item X from old_value to new_value.
3. **[read_item, T, X]:** Records that transaction T has read the value of database item X.
4. **[commit, T]:** Records that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. **[abort, T]:** Records that transaction T has been aborted.

Protocols for recovery that avoid cascading rollbacks do not require that read operations be written to the system log, whereas other protocols require these entries for recovery. Strict protocols require simpler write entries that do not include new_value.

Recovery using log records

If the system crashes, we can recover to a consistent database state by examining the log. Because the log contains a record of every write operation that changes the value of some database item, it is possible to **undo** the effect of these write operations of a transaction T by tracing backward through the log and resetting all items changed by a write operation of T to their old_values. We can also **redo** the effect of the write operations of a transaction T by tracing forward through the log and setting all items changed by a write operation of T (that did not get done permanently) to their new_values.

Commit Point of a Transaction

A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log. Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database. The transaction then writes an entry [commit, T] into the log.

Roll Back of transactions: Needed for transactions that have a [start_transaction, T] entry into the log but no commit entry [commit, T] into the log.

Redoing transactions: Transactions that have written their commit entry in the log must also have recorded all their write operations in the log; otherwise they would not be committed, so their effect on the database can be redone from the log entries. (Notice that the log file must be kept on disk. At the time of a system crash, only the log entries that have been written back to disk are considered in the recovery process because the contents of main memory may be lost.)

Force writing a log: Before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called force-writing the log file before committing a transaction.

8.4 Desirable Properties of Transactions

Transactions should possess several properties, often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:

- **Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **Consistency preservation:** A correct execution of the transaction must take the database from one consistent state to another.
- **Isolation:** A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary.
- **Durability or permanency:** Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

The **atomicity property** requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity. If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effects of the transaction on the database. On the other hand, write operations of a committed transaction must be eventually written to disk.

The preservation of **consistency** is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints. Recall that a **database state** is a collection of all the stored data items (values) in the database at a given point in time. A **consistent state** of the database satisfies the constraints specified in the schema as well as any other constraints on the database that should hold. A database program should be written in a way that guarantees that, if the database is in a consistent state before executing the transaction, it will be in a consistent state after the complete execution of the transaction, assuming that no interference with other transactions occurs.

The **isolation property** is enforced by the concurrency control subsystem of the DBMS. If every transaction does not make its updates (write operations) visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks but does not eliminate all other problems. There have been attempts to define the level of isolation of a transaction. A transaction is said to have level 0 (zero) isolation if it does not overwrite the dirty reads of higher-level transactions. Level 1 (one) isolation has no lost updates, and level 2 isolation has no lost updates and no dirty reads. Finally, level 3 isolation (also called true isolation) has, in addition to level 2 properties, repeatable reads.

And last, the **durability property** is the responsibility of the recovery subsystem of the DBMS.

8.5 Characterizing Schedules Based on Recoverability

Transaction schedule or history

When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms what is known as a transaction schedule (or history). A schedule (or history) S of n transactions T_1, T_2, ..., T_n is an ordering of the operations of the transactions subject to the constraint that, for each transaction T_i that participates in S, the operations of T_1 in S must appear in the same order in which they occur in T_1. Note, however, that operations from other transactions T_j can be interleaved with the operations of T_i in S.

For some schedules it is easy to recover from transaction and system failures, whereas for other schedules the recovery process can be quite involved. In some cases, it is even not possible to recover correctly after a failure. Hence, it is important to characterize the types of schedules for which recovery is possible, as well as those for which recovery is relatively simple. These characterizations do not actually provide the recovery algorithm; they only attempt to theoretically characterize the different types of schedules.

- **Recoverable schedule:** One where no transaction needs to be rolled back. A schedule S is recoverable if no transaction T in S commits until all transactions T′ that have written an item that T reads have committed.
- **Cascadeless schedule:** One where every transaction reads only the items that are written by committed transactions.
- **Schedules requiring cascaded rollback:** A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.
- **Strict Schedules:** A schedule in which a transaction can neither read nor write an item X until the last transaction that wrote X has committed.

8.6 Characterizing Schedules Based on Serializability

In the previous section, we characterized schedules based on their recoverability properties. Now we characterize the types of schedules that are always considered to be correct when concurrent transactions are executing. Suppose that two users—for example, two airline reservations agents—submit to the DBMS transactions T_1 and T_2 in Figure 8.1 at approximately the same time. If no interleaving of operations is permitted, there are only two possible outcomes:

1. Execute all the operations of transaction T_1 (in sequence) followed by all the operations of transaction T_2 (in sequence).

2. Execute all the operations of transaction T_2 (in sequence) followed by all the operations of transaction T_1 (in sequence).

(a)

T_1	T_2
read_item(X);	
X=X - N;	
write_item(X);	
read_item(Y);	
Y=Y+N;	
write_item(Y);	
	read_item(X);
	X=X+M;
	write_item(X);

Time

Schedule A

(b)

T_1	T_2
	read_item(X);
	X=X+M;
	write_item(X);
read_item(X);	
X=X - N;	
write_item(X);	
read_item(Y);	
Y=Y+N;	
write_item(Y);	

Time

Schedule B

(c)

T_1	T_2
read_item(X);	
X=X - N;	
	read_item(X);
	X=X+M;
write_item(X);	
read_item(Y);	
	write_item(X);
Y=Y+N;	
write_item(Y);	

Time

Schedule C

(d)

T_1	T_2
read_item(X);	
X=X - N;	
write_item(X);	
	read_item(X);
	X=X+M;
	write_item(X);
read_item(Y);	
Y=Y+N;	
write_item(Y);	

Time

Schedule D

Figure 8.5 Example of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial Schedule A: T_1 followed by T_2 (b) Serial Schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations

A schedule S is **serial schedule** if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in schedule else the schedule is called non serial schedule. A schedule S is **serializable schedule** if the equivalent to some serial schedule of the same n transactions. Two schedules are called **result equivalent** if they produce the same final state of the database. Two schedules are said to be **conflict equivalent** if the order of any two conflicting operations is the same in both schedules. A schedule S is said to be **conflict serializable** if it is conflict equivalent to some serial schedule S'.

S_1
read_item(X);
X=X + 10;
write_item(X);

S_2
read_item(X);
X=X*1.1;
write_item(X);

Figure 8.6 Two schedules that are result equivalent for the initial value of X=100 but are not result equivalent in general.

Being serializable is not the same as being serial. Being serializable implies that the schedule is a correct schedule if it will leave the database in a consistent state and the interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution. Serializability is hard to check because interleaving of operations occurs in an operating system through some scheduler and difficult to determine beforehand how the operations in a schedule will be interleaved. Practical approach came up with methods (protocols) to ensure serializability. It's not possible to determine when a schedule begins and when it ends. Hence, we reduce the problem of checking the whole schedule to checking only a committed project of the schedule (i.e. operations from only the committed transactions.) Current approach used in most DBMSs is use of locks with two phase locking. A less restrictive definition of equivalence of schedules is **view equivalence** and a schedule is serializable if it is view equivalent t a serial schedule.

Two schedules are said to be view equivalent if the following three conditions hold:

1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
2. For any operation $R_i(X)$ of T_i in S, if the value of X read by the operation has been written by an operation $W_j(X)$ of T_j (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation $R_i(X)$ of T_i in S'.
3. If the operation $W_k(Y)$ of T_k is the last operation to write item Y in S, then $W_k(Y)$ of T_k must also be the last operation to write item Y in S'.

The premise behind view equivalence are as long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results and the read operations are said to see the same view in both schedules. The relationship between view and conflict equivalence are:

- The two are same under constrained write assumption which assumes that if T writes X, it is constrained by the value of X it read; i.e., new X = f(old X)
- Conflict serializability is stricter than view serializability. With unconstrained write (or blind write), a schedule that is view serializable is not necessarily conflict serializable.
- Any conflict serializable schedule is also view serializable, but not vice versa.

Consider the following schedule of three transactions T_1: $r_1(X)$, $w_1(X)$; T_2: $w_2(X)$; and T_3: $w_3(X)$; Schedule **Sa**: $r_1(X)$; $w_2(X)$; $w_1(X)$; $w_3(X)$; c_1; c_2; c_3;

In **Sa**, the operations $w_2(X)$ and $w_3(X)$ are blind writes, since T_1 and T_3 do not read the value of X. **Sa** is view serializable, since it is view equivalent to the serial schedule T_1, T_2, T_3. However, **Sa** is not conflict serializable, since it is not conflict equivalent to any serial schedule.

Testing for conflict serializability of a schedule

Algorithm for testing conflict serializability of a schedule S:

1. Looks at only read_Item (X) and write_Item (X) operations
2. Constructs a precedence graph (serialization graph) - a graph with directed edges
3. An edge is created from T_i to T_j if one of the operations in T_i appears before a conflicting operation in T_j

4. The schedule is serializable if and only if the precedence graph has no cycles.

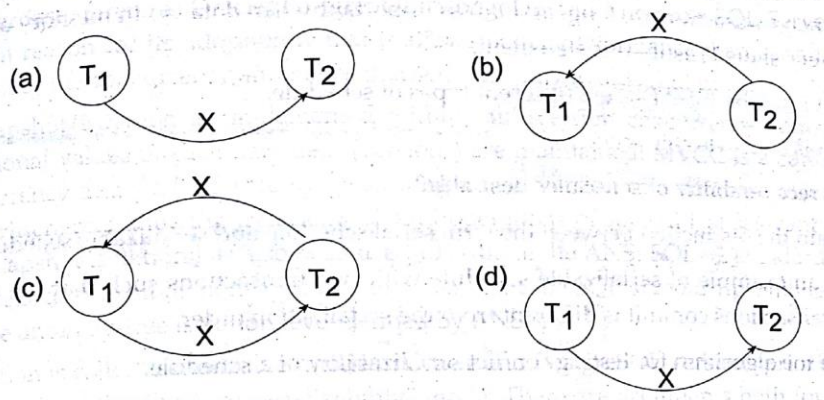


Figure 8.7 Constructing the precedence graph for schedules A to D from figure 8.5 to test for conflict serializability

1. What is meant by the concurrent execution of database transactions in a multiuser system?
2. Differentiate between single user and multiuser system.
3. Discuss why concurrency control is needed, and give informal examples.
4. Explain the types of problems which may encounter during concurrent transactions.
5. Discuss the different types of failures.
6. Why recovery is needed? Explain.
7. Explain the actions taken by the read_item and write_item operations on a database.
8. Draw a state diagram of transaction and explain the typical states briefly.
9. What is the system log used for?
10. What are the typical kinds of records in a system log?
11. What are transaction commit points and why are they important?
12. Discuss ACID properties of transactions along with usefulness of each.

13. "Concurrent execution of transactions is more important when data must be fetched from disk or when transaction are long, and is less important when data are in memory and transactions are very short." Justify this statement.
14. What is a schedule? Discuss different types of schedule.
15. What is a recoverable schedule?
16. Why recoverability of schedules desirable?
17. Explain the distinction between the term serial schedule and serializable schedule.
18. Give an example of serializable schedule with two transactions such that the order in which the transactions commit is different from the serialization order.
19. Write the algorithm for testing conflict serializability of a schedule.
20. Write short notes on:
    a. Transaction
    b. Database items
    c. ACID properties
    d. Recovery system
    e. Serial execution
    f. Precedence graph

❑❑❑

CHAPTER 9

CONCURRENCY CONTROL TECHNIQUES

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Two-Phase Locking Technique
- Timestamp Ordering
- Multiversion Concurrency Control
- Validation (Optimistic) Techniques and Snapshot Isolation Concurrency Control.

9.1 Introduction

In this chapter we discuss a number of concurrency control techniques that are used to ensure the noninterference or isolation property of concurrency executing transactions. Most of these techniques ensure serializability of schedules using concurrency control protocols (sets of rules) that guarantee serializability.

What is Concurrency Control?

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading and WRITE operations and hence the concurrency is a challenge.

Concurrency control is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases. Therefore, concurrency control is a most important element for the proper functioning of a system where two or multiple database transactions that require access to the same data, are executed simultaneously.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

For example: Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time. However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

9.2 Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

- Lock-Based Protocols
- Two Phase
- Timestamp-Based Protocols
- Validation-Based Protocols

Two-Phase Locking Technique

Some of the main techniques used to control concurrent execution of transactions are based on the concept of locking data items. A lock is a mechanism to control concurrent access to a data item. Generally, there is one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database items. There are various modes in which a data item may be locked. In this section, we restrict our attention to two modes:

1. **exclusive (X) mode:** Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
2. **shared (S) mode:** Data item can only be read. S-lock is requested using **lock-S** instruction.

Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

	S	X
S	True	False
X	False	False

Figure 9.1 Lock-compatibility Matrix

A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions. Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item. If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Example of a transaction performing locking:

```
T2: lock-S(A);
    read_item (A);
    unlock(A);
    lock-S(B);
    read_item (B);
    unlock(B);
    display(A+B)
```

Locking as above is not sufficient to guarantee serializability – if A and B get updated in-between the read of A and B, the displayed sum would be wrong. A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

T_3	T_4
lock-x(B)	
read_item(B)	
B=B-50	
write_item(B)	
	lock-s(A)
	read_item(A)
	lock-s(B)
lock-x (A)	

Figure 9.2 Partial Schedule

Neither T_3 nor T_4 can make progress – executing **lock-S(B)** causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X(A)** causes T_3 to wait for T_4 to release its lock on A. Such a situation is called a **deadlock**. To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released. The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil. **Starvation** is also possible if concurrency control manager is badly designed. For example: A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item. The same transaction is repeatedly rolled back due to deadlocks. Concurrency control manager can be designed to prevent starvation.

Table 9.1: Differences between starvation Vs deadlock

Starvation	Deadlock
Starvation happens if same transaction is always chosen as victim.	A deadlock is a condition in which two or more transaction is waiting for each other.
It occurs if the waiting scheme for locked items in unfair, giving priority to some transactions over others.	A situation where two or more transactions are unable to proceed because each is waiting for one of the other to do something.
Starvation is also known as lived lock.	Deadlock is also known as circular waiting.
Avoidance: • Switch priorities so that every thread has a chance to have high priority. • Use FIFO order among competing request.	**Avoidance:** • Acquire locks are predefined order. • Acquire locks at once before starting.
It means that transaction goes in a state where transaction never progress.	It is a situation where transactions are waiting for each other.

Guaranteeing Serializability by Two-Phase Locking

A transaction is said to follow the **two-phase locking protocol** if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction. Such a transaction can be divided into two phases: an **expanding** or **growing (first) phase**, during which new locks on items can be acquired but none can be released; and a **shrinking (second) phase**, during which existing locks can be released but no new locks can be acquired. If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase. Hence, a read_lock(X) operation that downgrades an already held write lock on X can appear only in the shrinking phase.

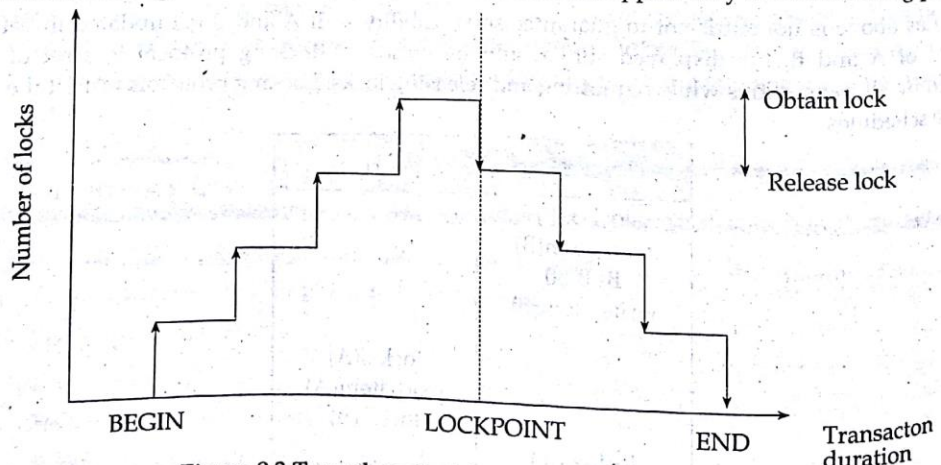


Figure 9.3 Two-phase Locking growing and shrinking.

Transactions T_1 and T_2 in Figure 9.4 (a) do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T_1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T_2. If we enforce two-phase locking, the transactions can be rewritten as T_1' and T_2', as shown in Figure 9.5. Now, the schedule shown in Figure 9.4 (c) is not permitted for T_1 and T_2 (with their modified order of locking and unlocking operations) under the rules of locking because T_1' will issue its write_lock(X) before it unlocks item Y; consequently, when T_2' issues its read_lock(X), it is forced to wait until T_1' releases the lock by issuing an unlock (X) in the schedule. It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules. The locking protocol, by enforcing two-phase locking rules, also enforces serializability.

(a)

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X=X+Y;	Y=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

(b)

initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result serial schedule T_2 followed by T_1: X=70, Y=50

(c)

T_1	T_2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y=X+Y;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
X=X+Y;	
write_item(X);	
unlock(X);	

Time

Result of schedule S: X=50, Y=50 (nonserializable)

Figure 9.4 Transactions that do not obey two-phase locking. (a) Two transactions T_1 and T_2. (b) Results of possible serial schedules of T_1 and T_2. (c) A nonserializable schedule S that uses locks.

T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X=X+Y;	Y=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Figure 9.5 Transactions T_1' and T_2', which are the same as T_1 and T_2 in figure 9.4, but follow the two-phase locking protocol

Two-phase policy generates two locking algorithms: (a) Basic and (b) Conservative

Conservative: Prevents deadlock by locking all desired data items before transaction begins execution.

Basic: Transaction locks data items incrementally. This may cause deadlock which is dealt with.

Strict: A stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

Two-phase locking does not ensure freedom from deadlocks. Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called **strict two-phase locking**. Here a transaction must hold all its exclusive locks till it commits/aborts. **Rigorous two-phase locking** is even stricter: here all locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

There can be conflict serializable schedules that cannot be obtained if two-phase locking is used. However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense: Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable.

Lock Conversion

Two-phase locking with lock conversion:

1. **First Phase:**
   - a. can acquire a lock-S on item
   - b. can acquire a lock-X on item
   - c. can convert a lock-S to a lock-X (upgrade)
2. **Second Phase:**
   - a. can release a lock-S
   - b. can release a lock-X
   - c. can convert a lock-X to a lock-S (downgrade)

This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

Implementation of Locking

A **lock manager** can be implemented as a process that receives messages from transactions and sends messages in reply. The lock-manager process replies to lock-request messages with lock-grant messages, or with messages requesting rollback of the transaction (in case of deadlocks). Unlock messages require only an acknowledgment in response, but may result in a grant message to another waiting transaction.

The lock manager uses this data structure: For each data item that is currently locked, it maintains a linked list of records, one for each request, in the order in which the requests arrived. It uses a hash table, indexed on the name of a data item, to find the linked list (if any) for a data item; this table is called the **lock table**. Each record of the linked list for a data item notes which transaction made the request, and what lock mode it requested. The record also notes if the request has currently been granted.

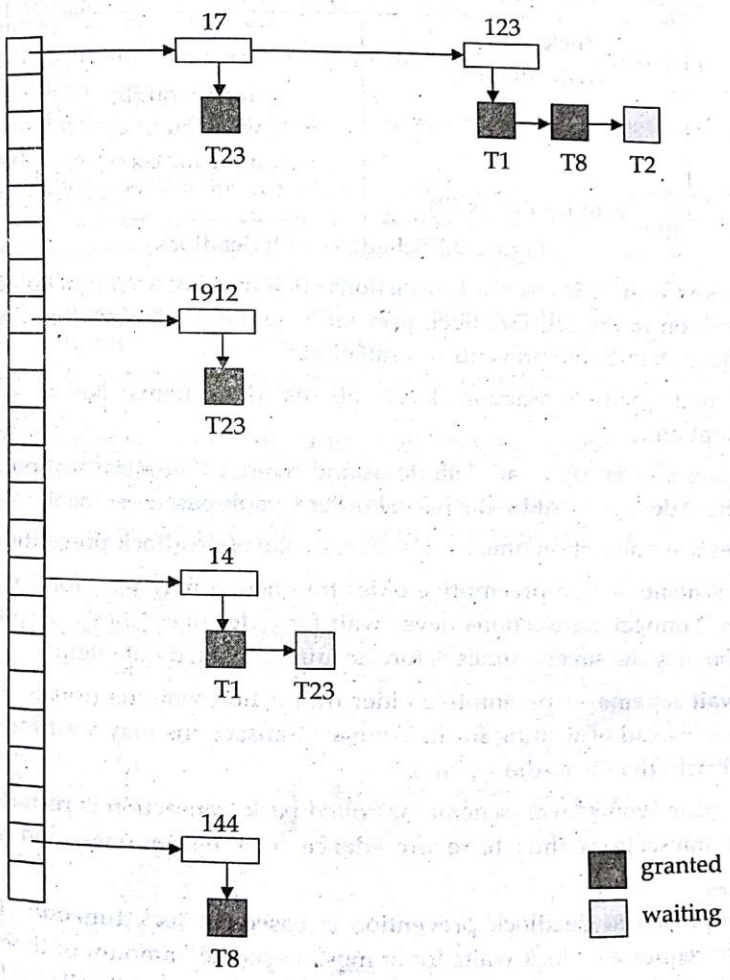


Figure 9.6 Lock Table

Figure 9.6 shows an example of a lock table. The table contains locks for five different data items, 14, 17, 123, 144, and 1912. The lock table uses overflow chaining, so there is a linked list of data items for each entry in the lock table. There is also a list of transactions that have been granted locks, or are

waiting for locks, for each of the data items. Granted locks are the rectangles filled in a darker shade, while waiting requests are the rectangles filled in a lighter shade. We have omitted the lock mode to keep the figure simple. It can be seen, for example, that T23 has been granted locks on 1912 and 17, and is waiting for a lock on 14.

9.4 Deadlock Handling

Consider the following two transactions:

T1: write_item (X)
write_item (Y)

T2: write_item (Y)
write_item (X)

T_1	T_2
lock-x on A write_item(A)	
	lock-x on (B) write_item(B)
wait for lock-x (B)	wait for **lock-x** on A

Figure 9.7 Schedule with deadlock

System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set. Deadlock prevention protocols ensure that the system will never enter into a deadlock state. Some prevention strategies:

- Require that each transaction locks all its data items before it begins execution (predeclaration).
- Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Following schemes use transaction timestamps for the sake of deadlock prevention alone.

- **wait-die scheme** — non-preemptive older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead. A transaction may die several times before acquiring needed data item.
- **wound-wait scheme** — preemptive older transaction wounds (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones. May be fewer rollbacks than wait-die scheme.

Both in wait-die and in wound-wait schemes, a rolled back transaction is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided.

Another simple approach to deadlock prevention is based on **lock timeouts**. In this approach, a transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, the transaction is said to time out, and it rolls itself back and restarts. If there was in fact a deadlock, one or more transactions involved in the deadlock will time out and roll back, allowing the others to proceed. This scheme falls somewhere between deadlock prevention, where a deadlock will never occur, and deadlock detection and recovery.

The **timeout scheme** is particularly easy to implement, and works well if transactions are short and if long waits are likely to be due to deadlocks. However, in general it is hard to decide how long a transaction must wait before timing out. Too long a wait results in unnecessary delays once a deadlock has occurred. Too short a wait results in transaction rollback even when there is no deadlock, leading to wasted resources. Starvation is also a possibility with this scheme. Hence, the timeout-based scheme has limited applicability.

Deadlock Detection and Recovery

If a system does not employ some protocol that ensures deadlock freedom, then a detection and recovery scheme must be used. An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred. If one has, then the system must attempt to recover from the deadlock. To do so, the system must:

- Maintain information about the current allocation of data items to transactions, as well as any outstanding data item requests.
- Provide an algorithm that uses this information to determine whether the system has entered a deadlock state.
- Recover from the deadlock when the detection algorithm determines that a deadlock exists.

Deadlock Detection

Deadlocks can be described precisely in terms of a directed graph called a **wait-for graph**. This graph consists of a pair G = (V, E), where V is a set of vertices and E is a set of edges. The set of vertices consists of all the transactions in the system. Each element in the set E of edges is an ordered pair $T_i \rightarrow T_j$. If $T_i \rightarrow T_j$ is in E, then there is a directed edge from transaction T_i to T_j, implying that transaction T_i is waiting for transaction T_j to release a data item that it needs. When transaction T_i requests a data item currently being held by transaction T_j, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when transaction T_j is no longer holding a data item needed by transaction T_i.

A deadlock exists in the system if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, the system needs to maintain the wait-for graph, and periodically to invoke an algorithm that searches for a cycle in the graph. To illustrate these concepts, consider the wait-for graph in Figure 9.8, which depicts the following situation:

- Transaction T_{17} is waiting for transactions T_{18} and T_{19}.
- Transaction T_{19} is waiting for transaction T_{18}.
- Transaction T_{18} is waiting for transaction T_{20}.

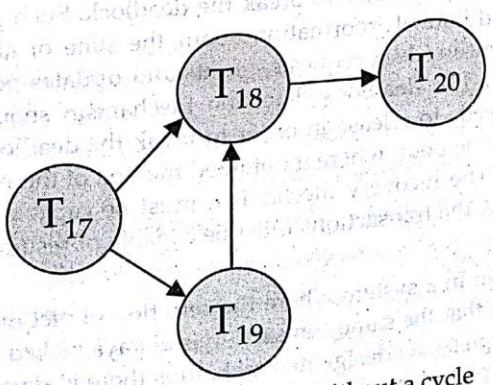


Figure 9.8 Wait-for graph without a cycle

Since the graph has no cycle, the system is not in a deadlock state. Suppose now that transaction T_{20} is requesting an item held by T_{19}. The edge $T_{20} \rightarrow T_{19}$ is added to the wait-for graph, resulting in the new system state in Figure 9.9. This time, the graph contains the cycle: $T_{18} \rightarrow T_{20} \rightarrow T_{19} \rightarrow T_{18}$

Implying that transactions T_{18}, T_{19}, and T_{20} are all deadlocked.

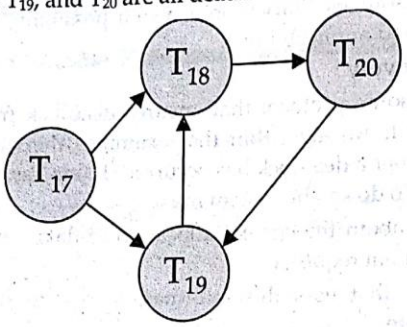


Figure 9.9 Wait-for graph with a cycle

Deadlock Recovery

When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution is to roll back one or more transactions to break the deadlock. Three actions need to be taken:

1. **Selection of a victim**: Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may determine the cost of a rollback, including:
   a) How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.
   b) How many data items the transaction has used.
   c) How many more data items the transaction needs for it to complete.
   d) How many transactions will be involved in the rollback?
2. **Rollback:** Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a total rollback: Abort the transaction and then restart it. However, it is more effective to roll back the transaction only as far as necessary to break the deadlock. Such partial rollback requires the system to maintain additional information about the state of all the running transactions. Specifically, the sequence of lock requests/grants and updates performed by the transaction needs to be recorded. The deadlock detection mechanism should decide which locks the selected transaction needs to release in order to break the deadlock. The selected transaction must be rolled back to the point where it obtained the first of these locks, undoing all actions it took after that point. The recovery mechanism must be capable of performing such partial rollbacks. Furthermore, the transactions must be capable of resuming execution after a partial rollback.
3. **Rollback in Starvation**: In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designated task, thus there is **starvation**. We must ensure that a transaction can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

9.5 Timestamp Ordering

The locking protocols that we have described thus far determine the order between every pair of conflicting transactions at execution time by the first lock that both members of the pair request that involves incompatible modes. Another method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a **timestamp-ordering scheme / protocol**.

The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation. The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter. The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created. Let's assume there are two transactions T_1 and T_2. Suppose the transaction T_1 has entered the system at 007 times and transaction T_2 has entered the system at 009 times. T_1 has the higher priority, so it executes first as it is entered the system first. The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Timestamps

Recall that a timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the transaction start time. We will refer to the timestamp of transaction T as TS(T). Concurrency control techniques based on timestamp ordering do not use locks; hence, **deadlocks cannot occur**.

Timestamps can be generated in several ways. One possibility is to use a counter that is incremented each time its value is assigned to a transaction. The transaction timestamps are numbered 1, 2, 3, ... in this scheme. A computer counter has a finite maximum value, so the system must periodically reset the counter to zero when no transactions are executing for some short period of time. Another way to implement timestamps is to use the current date/time value of the system clock and ensure that no two timestamp values are generated during the same tick of the clock.

Timestamp Ordering Algorithm

Basic Timestamp Ordering (TO)

Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with R_TS(X) and W_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new **timestamp**.

1. Check the following condition whenever a transaction T_i issues a **read_item(X)** operation:
   - If W_TS(X) >TS(T_i) then the operation is rejected. (Younger transaction has already written to the data item)
   - If W_TS(X) <= TS(T_i) then the operation is executed.
   - Timestamps of all the data items are updated.
2. Check the following condition whenever a transaction T_i issues a **write_item(X)** operation:

- If $TS(T_i) < R\_TS(X)$ then the operation is rejected. (Younger Transaction has already read the data item)
- If $TS(T_i) < W\_TS(X)$ then the operation is rejected (Younger Transaction has already read the data item) and Ti is rolled back otherwise the operation is executed.

Where,

TS(Ti) denotes the timestamp of the transaction T_i.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Strict Timestamp Ordering

A variation of basic TO called strict TO ensure that the schedules are both strict (for easy recoverability) and (conflict) serializable.

1. **Transaction T issues a write_item(X) operation:**
   - If TS(T) > R_TS(X), then delay T until the transaction T′ that wrote or read X has terminated (committed or aborted).
2. **Transaction T issues a read_item(X) operation:**
   - If TS(T) > W_TS(X), then delay T until the transaction T′ that wrote or read X has terminated (committed or aborted).

Thomas's Write Rule

A modification of the basic TO algorithm, known as Thomas's write rule, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows:

1. If R_TS(X) > TS(T) then abort and roll-back T and reject the operation.
2. If W_TS(X) > TS(T), then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
3. If the conditions given in 1 and 2 above do not occur, then execute write_item(X) of T and set W_TS(X) to TS(T).

9.6 Multiversion Concurrency Control

The concurrency-control schemes discussed thus far ensure serializability by either delaying an operation or aborting the transaction that issued the operation. For example, a read operation may be delayed because the appropriate value has not been written yet; or it may be rejected (that is, the issuing transaction must be aborted) because the value that it was supposed to read has already been overwritten. These difficulties could be avoided if old copies of each data item were kept in a system.

In **multiversion concurrency-control** schemes, each write_item(X) operation creates a new version of X. When a transaction issues a read_item(X) operation, the concurrency-control manager selects one of the versions of X to be read. The concurrency-control scheme must ensure that the version to be read is selected in a manner that ensures serializability. It is also crucial, for performance reasons that a transaction is able to determine easily and quickly which version of the data item should be read.

Multiversion technique based on timestamp ordering

Assume X_1, X_2, ..., X_n are the version of a data item X created by a write operation of transactions. With each X_i a R_TS (read timestamp) and a W_TS (write timestamp) are associated.

1. **R_TS:** The read timestamp of is the largest of all the timestamps of transactions that have successfully read version.
2. **W_TS:** The write timestamp of is the timestamp of the transaction that wrote the value of version.

Whenever a transaction T is allowed to execute a write_item(X) operation, a new version of item X is created, with both the W_TS and the R_TS set to TS(T). Correspondingly, when a transaction T is allowed to read the value of version X_i, the value of R_TS() is set to the larger of the current R_TS() and TS(T).

To ensure serializability, the following two rules are used:

1. If transaction T issues a write_item(X) operation, and version i of X has the highest W_TS() of all versions of X that is also less than or equal to TS(T), and R_TS() > TS(T), then abort and roll back transaction T; otherwise, create a new version of X with R_TS() = W_TS() = TS(T).
2. If transaction T issues a Read(X) operation, find the version i of X that has the highest W_TS() of all versions of X that is also less than or equal to TS(T); then return the value of to transaction T, and set the value of R_TS() to the larger of TS(T) and the current R_TS().

As we can see in case 2, a read_item(X) is always successful, since it finds the appropriate version to read based on the W_TS of the various existing versions of X. In case 1, however, transaction T may be aborted and rolled back. This happens if T is attempting to write a version of X that should have been read by another transaction T whose timestamp is R_TS(); however, T has already read version X_i, which was written by the transaction with timestamp equal to W_TS(). If this conflict occurs, T is rolled back; otherwise, a new version of X, written by transaction T, is created. Notice that, if T is rolled back, cascading rollback may occur. Hence, to ensure recoverability, a transaction T should not be allowed to commit until after all the transactions that have written some version that T has read have committed.

Multiversion Two-Phase Locking Using Certify Locks

In this multiple-mode locking scheme, there are three locking modes for an item: read, write, and certify, instead of just the two modes (read, write). Hence, the state of LOCK(X) for an item X can be one of read-locked, write-locked, certify-locked, or unlocked.

In the standard locking scheme, once a transaction obtains a write lock on an item, no other transactions can access that item. The idea behind multiversion 2PL is to allow other transactions T to read an item X while a single transaction T holds a write lock on X. This is accomplished by allowing two versions for each item X; one version must always have been written by some committed transaction. The second version X is created when a transaction T acquires a write lock on the item. Other transactions can continue to read the committed version of X while T holds the write lock. Transaction T can write the value of X as needed, without affecting the value of the committed version X.

However, once T is ready to commit, it must obtain a certify lock on all items that it currently holds write locks on before it can commit. The certify lock is not compatible with read locks, so the transaction may have to delay its commit until all its write-locked items are released by any reading transactions in order to obtain the certify locks. Once the certify locks—which are exclusive locks—are acquired, the committed version X of the data item is set to the value of version X, version X is discarded, and the certify locks are then released. In this multiversion 2PL scheme, reads can proceed concurrently with a single write operation—an arrangement not permitted under the standard 2PL schemes.

9.7 Validation (Optimistic) Techniques

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

1. **Start(T_i):** It contains the time when Ti started its execution.
2. **Validation (T_i):** It contains the time when Ti finishes its read phase and starts its validation phase.
3. **Finish(T_i):** It contains the time when Ti finishes its write phase.

This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.

Hence $TS(T_i)$ = validation(T_i).

The serializability is determined during the validation process. It can't be decided in advance. While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts. Thus it contains transactions which have less number of rollbacks.

9.8 Snapshot Isolation Concurrency Control

In databases, and transaction processing (transaction management), snapshot isolation is a guarantee that all reads made in a transaction will see a consistent snapshot of the database (in practice it reads the last committed values that existed at the time it started), and the transaction itself will successfully commit only if no updates it has made conflict with any concurrent updates made since that snapshot.

Snapshot isolation has been adopted by several major database management systems, such as SQL Anywhere, InterBase, Firebird, Oracle, PostgreSQL, MongoDB and Microsoft SQL Server (2005 and later). The main reason for its adoption is that it allows better performance than serializability, yet still avoids most of the concurrency anomalies that serializability avoids (but not always all).

In practice snapshot isolation is implemented within multiversion concurrency control (MVCC), where generational values of each data item (versions) are maintained: MVCC is a common way to increase concurrency and performance by generating a new version of a database object each time the object is written, and allowing transactions' read operations of several last relevant versions (of each object). Snapshot isolation has also been used to critique the ANSI SQL-92 standard's definition of isolation levels, as it exhibits none of the "anomalies" that the SQL standard prohibited, yet is not serializable (the anomaly-free isolation level defined by ANSI).

Snapshot isolation is called "serializable" mode in Oracle and PostgreSQL versions prior to 9.1, which may cause confusion with the "real serializability" mode. There are arguments both for and against this decision; what is clear is that users must be aware of the distinction to avoid possible undesired anomalous behavior in their database system logic.

1. What is concurrency control?
2. Why use concurrency method?
3. Discuss the problems of deadlock and starvation.
4. Differentiate between deadlock and starvation.
5. What is the two-phase locking protocol? How does it guarantee serializability?
6. List the types of two-phase locking? Why is strict or rigorous two-phase locking often preferred?
7. Explain the exclusive and shared locks along with lock-compatibility matrix.
8. Discuss the phases of lock conversion.
9. Explain how is the lock implemented?
10. What do you mean by deadlock handling?

11. Describe the wait-die and wound-wait protocols for deadlock prevention.
12. If deadlock is avoided by deadlock-avoidance schemes, is starvation still possible? Explain.
13. How deadlock is detected? Explain.
14. Explain the deadlock recovery process.
15. What is a timestamp? How does the system generate timestamps?
16. Write the timestamp ordering algorithm.
17. When transaction is rolled back under timestamp ordering, it is assigned a new timestamp. Why can it not simply keep its old timestamp?
18. Discuss Thomas's Write Rule.
19. Discuss multiversion techniques for concurrency control.
20. Explain the validation techniques and snapshot isolation concurrency control in detail.

❑❑❑

CHAPTER 10

DATABASE RECOVERY TECHNIQUES

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Recovery Concepts
- NO-UNDO/REDO Recovery Based on Deferred Update
- Recovery Technique Based on Immediate Update
- Shadow Paging
- Database Backup and Recovery from Catastrophic Failures

10.1 Introduction

A computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure, information may be lost. Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure. The recovery scheme must provide high availability; that is, it must minimize the time for which the database is not usable after a failure. In this chapter we discuss some of the techniques that can be used for database recovery from failures.

Failure Classification

There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner. In this chapter, we shall consider only the following types of failure:

- Transaction failure
- System crash
- Disk failure

1. **Transaction failure:** A transaction needs to abort once it fails to execute or once it reaches to any further extent from wherever it can't go to any extent further. This is often known as transaction failure wherever solely many transactions or processes are hurt. The reasons for transaction failure are:
   - Logical errors
   - System errors
     a. **Logical errors:** Where a transaction cannot complete as a result of its code error or an internal error condition.
     b. **System errors:** the database system must terminate an active transaction due to an error condition (e.g., deadlock)
2. **System crash:** There are issues external to the system – that will cause the system to prevent abruptly and cause the system to crash. For instance, interruptions in power supply might cause the failure of underlying hardware or software package failure. Examples might include OS errors.
3. **Disk failure:** In early days of technology evolution, it had been a typical drawback wherever hard-disk drives or storage drives accustomed to failing oftentimes. Disk failures include the formation of dangerous sectors, unreachability to the disk, disk crash or the other failure, that destroys all or a section of disk storage.

Storage Structure

Classification of storage structure is as explained below:

1. **Volatile storage:** As the name suggests, a memory board (volatile storage) cannot survive system crashes. Volatile storage devices are placed terribly near to the CPU; usually, they're embedded on the chipset itself. For instance, main memory and cache memory are samples of the memory board. They're quick however will store a solely little quantity of knowledge.

2. **Non-volatile storage:** These recollections are created to survive system crashes. They are immense in information storage capability, however slower in the accessibility. Examples could include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

10.2 Recovery Concepts

Recovery Outline and Categorization of Recovery Algorithms

Recovery from transaction failures usually means that the database is restored to the most recent consistent state just before the time of failure. To do this, the system must keep information about the changes that were applied to data items by the various transactions. This information is typically kept in the system log. A typical strategy for recovery may be summarized informally as follows:

1. If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was backed up to archival storage (typically tape or other large capacity offline storage media) and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure.
2. When the database on disk is not physically damaged, and a noncatastrophic failure has occurred, the recovery strategy is to identify any changes that may cause an inconsistency in the database. **For example**, a transaction that has updated some database items on disk but has not been committed needs to have its changes reversed by undoing its write operations. It may also be necessary to redo some operations in order to restore a consistent state of the database; for example, if a transaction has committed but some of its write operations have not yet been written to disk. For noncatastrophic failure, the recovery protocol does not need a complete archival copy of the database. Rather, the entries kept in the online system log on disk are analyzed to determine the appropriate actions for recovery.

Log Based Recovery

In log based recovery system, a log is maintained, in which all the modifications of the database are kept. A log consists of log records. For each activity of database, separate log record is made. Log records are maintained in a serial manner in which different activities are happened. There are various log records. A typical update log record must contain following fields:

a) **Transaction identifier:** A unique number given to each transaction.
b) **Data-item identifier:** A unique number given to data item written.
c) **Date and time of updation.**
d) **Old value:** Value of data item before write.
e) **New value:** Value of data item after write.

Logs must be written on the non-volatile (stable) storage. In log-based recovery, the following two operations for recovery are required:

a) **Redo:** It means, the work of the transactions that completed successfully before crash is to be performed again.
b) **Undo:** It means, all the work done by the transactions that did not complete due to crash is to be undone.

The redo and undo operations must be idempotent. An idempotent operation is that which gives same result, when executed one or more times.

For any transaction Ti, Various log records are:

[T_i start]: It records to log when T_i starts execution.

[T_i, A_j]: It records to log when T_i reads data item A_j.

[Ti, A_j, V_1, V_2]: It records to log when T_i updates data item A_j, where V_1 refer to old value and V_2 refers to new value of A_j.

[T_i Commit]: It records to log when Ti, successfully commits.

[T_i aborts]: It records to log if T_i aborts.

There are two types of log based recovery techniques and they are: **(a) Recovery based on deferred update** and **(b) Recovery based on immediate update**

Caching (Buffering) of Disk Blocks

The recovery process is often closely intertwined with operating system functions— in particular, the buffering of database disk pages in the DBMS main memory cache. Typically, multiple disk pages that include the data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. The caching of disk pages is traditionally an operating system function, but because of its importance to the efficiency of recovery procedures, it is handled by the DBMS by calling low-level operating systems routines.

Two main strategies can be employed when flushing a modified buffer back to disk. The first strategy, known as **in place updating**, writes the buffer to the same original disk location, thus overwriting the old value of any changed data items on disk. Hence, a single copy of each database disk block is maintained. The second strategy, known as **shadowing**, writes an updated buffer at a different disk location, so multiple versions of data items can be maintained, but this approach is not typically used in practice.

In general, the old value of the data item before updating is called the before image (BFIM), and the new value after updating is called the after image (AFIM). If shadowing is used, both the BFIM and the AFIM can be kept on disk; hence, it is not strictly necessary to maintain a log for recovering.

Write-Ahead Logging

Every recovery procedure involves flushing its logs store in data buffers into the disk. In place updating & showing in place updating writes the data items that are updated at the same location in the disk every time by overwriting the content whereas shadowing places the logs records at different locations in disk, which helps the log record of the same data item updated at different time.

Now, if a failure occurs, the RAM buffer that stores database information as well as the log may lost, which will cause loosing of information write-ahead logging protocols is derived in protect the system in such case. In case of in-place uploading where old data values has been replaced by new data values, we need to implement write-ahead logging protocol, which states that:-

1. The old value cannot be replaced by its new value until and undo type logging record has been permanently stored in the disk.
2. Prior to commit operation of a transaction, the redo portion and undo portion of the log have been written permanently in the disk. To make the recovery process more efficient DBMS recovery subsystem may maintain a list of transaction details, which include the list of active transactions that are not committed yet as well as the list of all the committed & aborted transactions until the last checkpoints.

Checkpoint

Another type of entry in the log is called a checkpoint. The **checkpoint** is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk. The

checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created. When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on. The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:

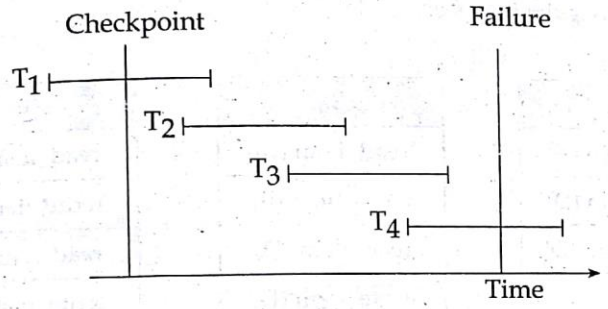


Figure 10.1 Recovery using Checkpoint

The recovery system reads log files from the end to start. It reads log files from T_4 to T_1.

Recovery system maintains two lists, a redo-list, and an undo-list. The transaction is put into redo state if the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs. For example: In the log file, transaction T_2 and T_3 will have <T_n, Start> and <T_n, Commit>. The T_1 transaction will have only <T_n, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T_1, T_2 and T_3 transaction into redo list. The transaction is put into undo state if the recovery system sees a log with <T_n, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed. For example: Transaction T_4 will have <T_n, Start>. So T_4 will be put into undo list since this transaction is not yet complete and failed amid.

Transaction Rollback and Cascading Rollback

If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to roll back the transaction. If any data item values have been changed by the transaction and written to the database, they must be restored to their previous values (BFIMs). The undo-type log entries are used to restore the old values of data items that must be rolled back.

If a transaction T is rolled back, any transaction S that has, in the interim, read the value of some data item X written by T must also be rolled back. Similarly, once S is rolled back, any transaction R that has read the value of some data item Y written by S must also be rolled back; and so on. This phenomenon is called cascading roll-back, and can occur when the recovery protocol ensures recoverable schedules but does not ensure **strict** or **cascadeless** schedules. Understandably, cascading rollback can be quite complex and time-consuming. That is why almost all recovery mechanisms are designed so that cascading rollback **is never required.**

Figure 10.2 shows an example where cascading rollback is required. The read and write operations of three individual transactions are shown in Figure 10.2 (a). Figure 10.2 (b) shows the system log at the point of a system crash for a particular execution schedule of these transactions. The values of data items A, B, C, and D, which are used by the transactions, are shown to the right of the system log entries. We assume that the original item values, shown in the first line, are A = 30, B = 15, C = 40, and D = 20. At the point of system failure, transaction T_3 has not reached its conclusion and must be rolled back. The WRITE operations of T_3, marked by a single * in Figure 10.2 (b), are the T_3 operations that are undone during transaction rollback. Figure 10.2 (c) graphically shows the operations of the different transactions along the time axis.

(a)

T_1
read_item (A)
read_item (D)
write_item (D)

T_2
read_item (B)
write_item (B)
read_item (D)
write_item (D)

T_3
read_item (C)
write_item (B)
read_item (A)
write_item (A)

(b)

		A	B	C	D
		30	15	40	20
	[start_transaction, T_3]				
	[read_item, T_3, C]				
*	[write_item, T_3, B, 15, 12]		12		
	[start_transaction, T_2]				
	[read_item, T_2, B]				
**	[write_item, T_2, B]		18		
	[star_transaction, T_1]				
	[read_item, T_1, A]				
	[read_item, T_1, D]				
	[write_item, T_1, D, 20, 25]				25
	[read_item, T_2, D]				
**	[write_item, T_2, D, 25, 26]				26
	[read_item, T_3, A]				

← System Crash

*T_3 is rolled back because it did not reach its commit point.

*T_2 is rolled back because it reads the value of item B written by T_3.

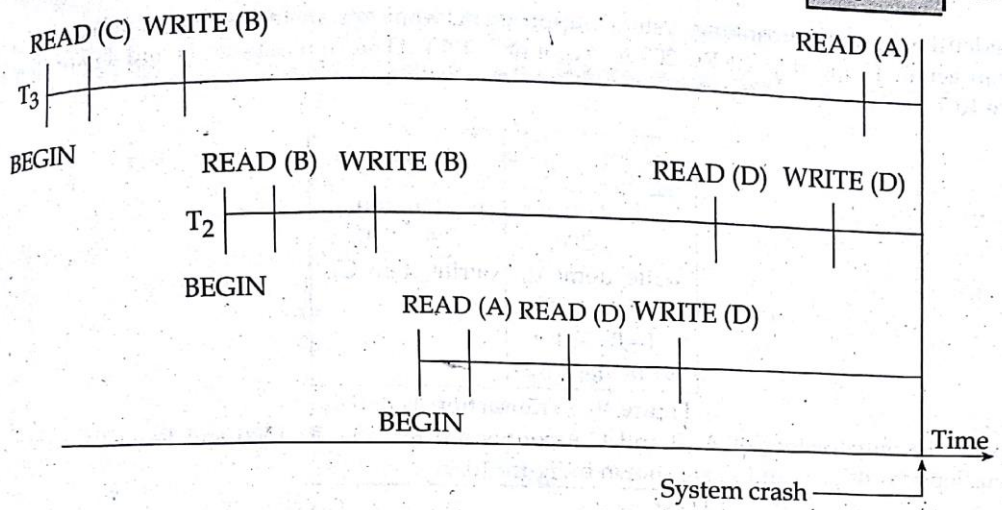


Figure 10.2 Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operatins before the crash.

We must now check for cascading rollback. From Figure 10.2(c) we see that transaction T_2 reads the value of item B that was written by transaction T_3; this can also be determined by examining the log. Because T_3 is rolled back, T_2 must now be rolled back, too. The WRITE operations of T_2, marked by ** in the log, are the ones that are undone. Note that only write_item operations need to be undone during transaction rollback; read_item operations are recorded in the log only to determine whether cascading rollback of additional transactions is necessary.

In practice, cascading rollback of transactions is never required because practical recovery methods guarantee cascadeless or strict schedules. Hence, there is also no need to record any read_item operations in the log because these are needed only for determining cascading rollback.

Transaction Actions That Do Not Affect the Database

In general, a transaction will have actions that do not affect the database, such as generating and printing messages or reports from information retrieved from the database. If a transaction fails before completion, we may not want the user to get these reports, since the transaction has failed to complete. If such erroneous reports are produced, part of the recovery process would have to inform the user that these reports are wrong, since the user may take an action based on these reports that affects the database. Hence, such reports should be generated only after the transaction reaches its commit point. A common method of dealing with such actions is to issue the commands that generate the reports but keep them as batch jobs, which are executed only after the transaction reaches its commit point. If the transaction fails, the batch jobs are canceled.

10.3 Recovery based on Deferred Update

In deferred update technique, deferred (stops) all the write operations of any Transaction T_i until it practically commits. It means modify real database after T_i partially commits. All the activities are recorded in log. Log records are used to modify actual database. Suppose a transaction T_i wants to write on data item A_j, then a log record $[T_i, A_j, V_1, V_2]$ is saved in log and it is used to modify database after actual modification Ti enters in committed state. **In this technique, the old value field is not needed.**

Consider the example of banking system. Suppose you want to transfer Rs. 200 from Account A to B in Transaction T_1 and deposit Rs. 200 to Account C in T_2. The transactions T_1 and T_2 are shown in figure 10.3.

T_1	T_2

Read_item(A);
A=A-200;
write_item(A);
read_item(B);
B=B+200;
write_item(B);
	read_item(C);
	C=C+200;
	write_item(C);

Figure 10.3 Transaction T_1 and T_2

Suppose, the initial values of A, B and C accounts are Rs. 500, Rs.1000 and Rs. 600 respectively, various log records of T_1 and T_2 are shown in figure 10.4:

[T_1 start]
[T_1, A]
[T_1, A, 300]
[T_1, B]
[T_1, B, 1200]
[T_1 commit]
[T_2 start]
[T_2, C]
[T_2, C, 800]
[T_2 commit]

Figure 10.4 Log records for transaction T_1 and T_2

For a **redo** operation, log must contain [T_i start] and [T_i commit] log records.

(a)	(b)
[T_1 start] [T_1, A] [T_1, A, 300]	[T_1 start] [T_1, A] [T_1, A, 300] [T_1, B] [T_1, B, 1200] [T_1 commit] [T_2 start] [T_2, C] [T_2, C, 800]

Figure 10.5 Log of transaction T_1 and T_2 in case of crash

Crash will happen at any time of execution of transactions. Suppose crash happened

a) **After write_item(A) of T_1:** At that time log records in long are shown in figure 10.5 (a). There is no need to redo operation because no commit record appears in the log. Log records of T_1 can be deleted.

b) **After write_item(C) of T_2:** At that time log records in log are shown in figure 10.5 (b). In this situation, you have to redo T_1 because both [T_1 start] and [T_1 commit] appears in log. After redo operation, value of A and B are 300 and 1200 respectively. Values remain same because redo is idempotent.

c) **During recovery:** If system is crashed at the time of recovery, simply starts the recovery again.

10.4 Recovery Based on Immediate Update

In immediate update technique, database is modified by any transaction T_i during its active state. It means, real database is modified just after the write operation but after log record is written to stable storage. This is because log records are used during recovery. Use both Undo and Redo operations in this method. Old value field is also needed (for undo operation). Consider again the bank transaction of figure 10.3. Corresponding log records after successful completion of T_1 and T_2 are shown in figure 10.6.

[T_1 start]
[T_1, A]
[T_1, A, 500, 300]
[T_1, B]
[T_1, B, 1000, 1200]
[T_1 commit]
[T_2 start]
[T_2, C]
[T_2, C, 600, 800]
[[T_2 commit]

Figure 10.6 Log records for transactions T_1 and T_2.

For a transaction T_i to be redone, log must contain both [T_i start] and [T_i commit] records. For a transaction T_i to be undone, log must contain only [T_i start] record.

(a)	(b)
[T_1 start]	[T_1 start]
[T_1, A]	[T_1, A]
[T_1, A, 500, 300]	[T_1, A, 500, 300]
	[T_1, B]
	[T_1, B, 100, 1200]
	[T_1 commit]
	[T_2 start]
	[T_2, C]
	[T_2, C, 600, 800]

Figure 10.7 Log of Transaction T_1 and T_2 in case of crash.

Crash will happen at any time of execution of transaction. Suppose crash happened.

a) **After write_item(A) of T_1:** At that time log records in long are shown in figure 10.7 (a). Here only [T_1 start] exists so undo transaction T1. As a result, Account A restores its old value 500.

b) **After write_item(C) of T_2:** At that time log records in log are shown in figure 10.7 (b). During back-up record [T_2 start] appears but there is no [T_2 commit], so undo transaction T2. As a result, Account C restores its old value 600. When you found both [T_1 start] and [T_1 commit] records in log, redo transaction T1 and Account A and B both keep their new value.

c) **During recovery:** If system is crashed at the time of recovery, simply starts the recovery again.

10.5 Shadow Paging

This is the method where all the transactions are executed in the primary memory or the shadow copy of database. Once all the transactions completely executed, it will be updated to the database. Hence, if there is any failure in the middle of transaction, it will not be reflected in the database. Database will be updated after all the transaction is complete.

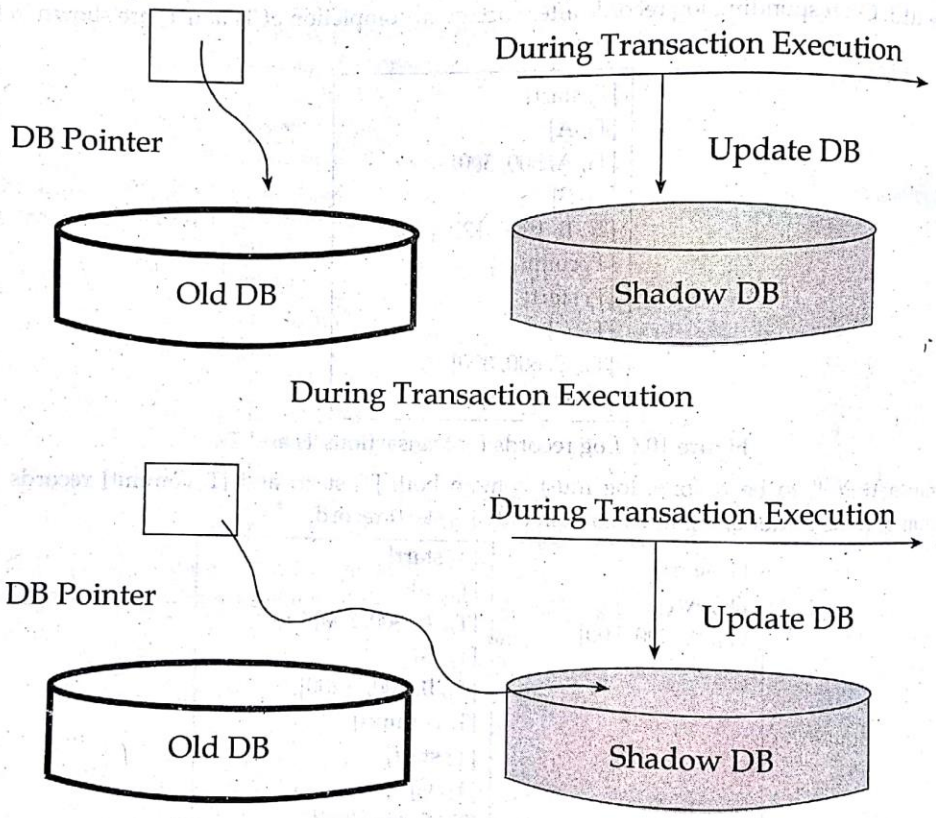


Figure 10.8 Shadow Paging

A database pointer will be always pointing to the consistent copy of the database, and copy of the database is used by transactions to update. Once all the transactions are complete, the DB pointer is modified to point to new copy of DB, and old copy is deleted. If there is any failure during the transaction, the pointer will be still pointing to old copy of database, and shadow database will be deleted. If the transactions are complete then the pointer is changed to point to shadow DB, and old DB is deleted.

As we can see in above diagram, the DB pointer is always pointing to consistent and stable database. This mechanism assumes that there will not be any disk failure and only one transaction executing at a time so that the shadow DB can hold the data for that transaction. It is useful if the DB is comparatively small because shadow DB consumes same memory space as the actual DB. Hence it is not efficient for huge DBs. In addition, it cannot handle concurrent execution of transactions. It is suitable for one transaction at a time.

Advantages

- No Overhead for writing log records.
- No Undo / No Redo algorithm.
- Recovery is faster.

Disadvantages

- Data gets fragmented or scattered.
- After every transaction completion database pages containing old version of modified data need to be garbage collected.
- Hard to extend algorithm to allow transaction to run concurrently.

10.6 Database Backup and Recovery from Catastrophic Failures

So far, all the techniques we have discussed apply to noncatastrophic failures. A key assumption has been that the system log is maintained on the disk and is not lost as a result of the failure. Similarly, the shadow directory must be stored on disk to allow recovery when shadow paging is used. The recovery techniques we have discussed use the entries in the system log or the shadow directory to recover from failure by bringing the database back to a consistent state.

The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as disk crashes. The main technique used to handle such crashes is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices. In case of a catastrophic system failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.

Data from critical applications such as banking, insurance, stock market, and other databases is periodically backed up in its entirety and moved to physically separate safe locations. Subterranean storage vaults have been used to protect such data from flood, storm, earthquake, or fire damage. Events like the 9/11 terrorist attack in New York (in 2001) and the Katrina hurricane disaster in New Orleans (in 2005) have created a greater **awareness of disaster recovery of business-critical databases.**

To avoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape. The system log is usually substantially smaller than the database itself and hence can be backed up more frequently. Therefore, users do not lose all transactions they have performed since the last database backup. All committed transactions recorded in the portion of the system log that has been backed up to tape can have their effect on the database redone. A new log is started after each database backup. Hence, to recover from disk failure, the database is first recreated on disk from its latest backup copy on tape. Following that, the effects of all the committed transactions whose operations have been recorded in the backed-up copies of the system log are reconstructed.

1. Discuss recovery. What is the need of performing recovery?
2. Discuss the various methods of performing recovery.
3. Describe log-based and checkpoint schemes for data recovery.
4. Explain log-based recovery methods. What are the roles of checkpoints in it? Discuss.
5. Explain anyone of the methods to recover the lost data in a database system.
6. Which type of failures can exist in a system?
7. Compare the deferred and immediate update versions of the log-based recovery in terms of overhead and implementation.
8. Explain techniques of recovery of database in detail.
9. Discuss various data recovery issues.
10. What are the techniques available for data recovery?
11. What is a log file? What does it contain? How can a log file be used for recovery? Describe this with the help of an example that includes all recovery operations (UNDO, REDO, etc).
12. How can you recover from media failure on which your database was stored? Describe the mechanism of such recovery.
13. Why is recovery needed in database? How can log be used to recover from a failure? How is log-based recovery different from that of checkpoint based recovery?
14. What do you understand by recovery and reliability in the context of database systems? Explain with the help of an example.
15. When is a system considered to be reliable? Explain two major types of system failures. List any two recovery schemes.
16. Describe the shadow paging recovery scheme along with a diagram.
17. Compare shadow paging scheme with the log-based recovery scheme in terms of implementation.
18. Compare and contrast the features of log-based recovery mechanism with checkpointing based recovery. Suggest applications where you will prefer log-based recovery schemes over checkpointing. Give an example of checkpointing based recovery scheme.
19. What is checkpoint? Why is it needed? What are the actions which are taken by DBMS at a checkpoint? Explain with the help of an example.
20. Write short notes on:
    a. Transaction Rollback
    b. Cascading Rollback
    c. Write-Ahead Logging
    d. Storage Structure
    e. Data Backup
    f. Recovery from Catastrophic Failures

❑❑❑

CHAPTER 11

DISTRIBUTED DATABASE MANAGEMENT SYSTEM

LEARNING OBJECTIVES

After comprehensive study of this Chapter, you will be able to:

- Introduction to Distributed Database Management System
- Characteristics of DBMS
- Advantages/Disadvantages of DDBMS
- Components of Distributed Database system
- Distributed Database Design

11.1 Introduction to Distributed Database Management System

A major motivation behind the development of database systems is the desire to integrate the operational data of an organization and to provide controlled access to the data. Although integration and controlled access may imply centralization, this is not the intention. In fact, the development of computer networks promotes a decentralized mode of work. This decentralized approach mirrors the organizational structure of many companies, which are logically distributed into divisions, departments, projects, and so on, and physically distributed into offices, plants, factories, where each unit maintains its own operational data. The share ability of the data and the efficiency of data access should be improved by the development of a distributed database system that reflects this organizational structure, makes the data in all units accessible, and stores data proximate to the location where it is most frequently used.

Distributed DBMSs should help resolve the islands of information problem. Databases are sometimes regarded as electronic islands that are distinct and generally inaccessible places, like remote islands. This may be a result of geographical separation, incompatible computer architectures, incompatible communication protocols, and so on. Integrating the databases into a logical whole may prevent this way of thinking.

What is Distributed Database system?

A **distributed database (DDB)** is a collection of multiple, logically interrelated databases distributed over a computer network. A **distributed database management system** (DDBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users. The distributed database (DDB) and distributed database management system (DDBMS) together is called **Distributed database system (DDBS).**

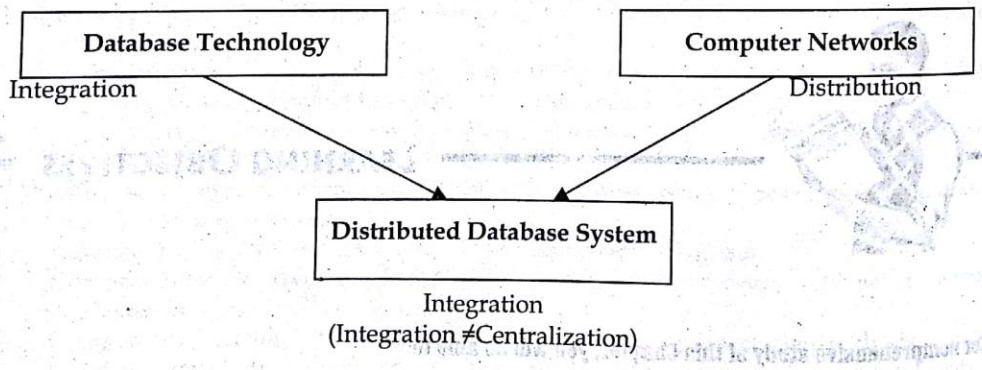


Figure 11.1 Distributed Database System

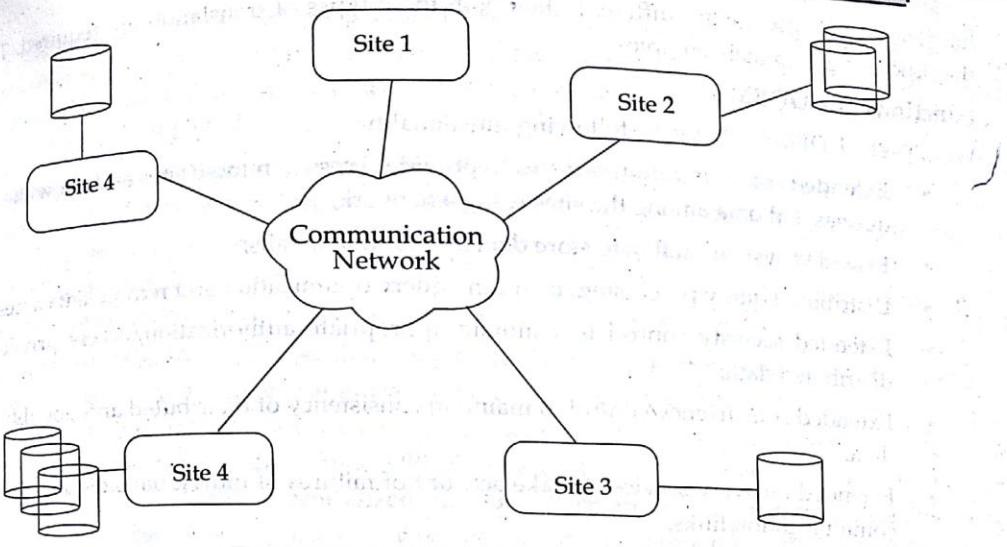


Figure 11.2: Distributed Environment

Types of Distributed DBMS

Distributed databases can be broadly classified into homogeneous and heterogeneous distributed database environments.

Homogeneous and Heterogeneous DDBMS

In a **homogeneous** system, all sites use the same DBMS product. In a **heterogeneous** system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs. Homogeneous systems are much easier to design and manage. This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.

Heterogeneous systems usually result when individual sites have implemented their own databases and integration is considered at a later stage. In a heterogeneous system, translations are required to allow communication between different DBMSs. To provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site. The system then has the task of locating the data and performing any necessary translation. Data may be required from another site that may have:

- Different hardware
- Different DBMS products
- Different hardware and different DBMS products.

If the hardware is different but the DBMS products are the same, the translation is straightforward, involving the change of codes and word lengths. If the DBMS products are different, the translation is complicated involving the mapping of data structures in one data model to the equivalent data structures in another data model. For example, relations in the relational data model are mapped to records and sets in the network model. It is also necessary to translate the query language used (for example, SQL SELECT statements are mapped to the network FIND and GET statements). If both the

hardware and software are different, then both these types of translation are required. This makes the processing extremely complex.

Functions of a DDBMS

We expect a DDBMS to have the following functionality:

- Extended communication services to provide access to remote sites and allow the transfer of queries and data among the sites using a network;
- Extended system catalog to store data distribution details;
- Distributed query processing, including query optimization and remote data access;
- Extended security control to maintain appropriate authorization/access privileges to the distributed data;
- Extended concurrency control to maintain consistency of distributed and possibly replicated data;
- Extended recovery services to take account of failures of individual sites and the failures of communication links.

11.2 Characteristics of DBMS

The characteristics of DDBMS are:

- A collection of logically related shared data;
- The data is split into a number of fragments;
- Fragments may be replicated;
- Fragments/replicas are allocated to sites;
- The sites are linked by a communications network;
- The data at each site is under the control of a DBMS;
- The DBMS at each site can handle local applications, autonomously;
- Each DBMS participates in at least one global application.

11.3 Advantages/Disadvantages of DDBMS

The distribution of data and applications has potential advantages over traditional centralized database systems. Unfortunately, there are also disadvantages. In this section we review the advantages and disadvantages of the DDBMS.

Advantages:

1. **Reflects organizational structure:** Many organizations are naturally distributed over several locations.
2. **Improved shareability and local autonomy:** The geographical distribution of an organization can be reflected in the distribution of the data; users at one site can access data stored at other sites. Data can be placed at the site close to the users who normally use that data. In this way,

users have local control of the data and they can consequently establish and enforce local policies regarding the use of this data. A global DBA is responsible for the entire system. Generally, part of this responsibility is devolved to the local level, so that the local DBA can manage the local DBMS.

3. **Improved availability:** In a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS or a failure of a communication link making some sites inaccessible does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. If a single node fails, the system may be able to reroute the failed node's requests to another site.

4. **Improved reliability:** Because data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible. Improved performance as the data is located near the site of "greatest demand," and given the inherent parallelism of distributed DBMSs, speed of database access may be better than that achievable from a remote centralized database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralized DBMS.

5. **Economics:** The potential cost saving occurs where databases are geographically remote and the applications require access to distributed data. In such cases, owing to the relative expense of data being transmitted across the network as opposed to the cost of local access, it may be much more economical to partition the application and perform the processing locally at each site.

6. **Modular growth:** In a distributed environment, it is much easier to handle expansion. New sites can be added to the network without affecting the operations of other sites. This flexibility allows an organization to expand relatively easily.

7. **Integration:** At the start of this section we noted that integration was a key advantage of the DBMS approach, not centralization. The integration of legacy systems is one particular example that demonstrates how some organizations are forced to rely on distributed data processing to allow their legacy systems to coexist with their more modern systems. At the same time, no one package can provide all the functionality that an organization requires nowadays. Thus, it is important for organizations to be able to integrate software components from different vendors to meet their specific requirements.

8. **Remaining competitive:** There are number of relatively recent developments that rely heavily on distributed database technology such as e-business, computer supported collaborative work and workflow management. Many enterprises have had to reorganize their businesses and use distributed database technology to remain competitive.

Disadvantages

1. **Complexity:** A distributed DBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability, and availability is inherently more complex than a centralized DBMS. The fact that data can be replicated also adds an extra level of complexity to the distributed DBMS. If the software does not handle data replication adequately, there will be degradation in availability, reliability, and performance compared with the centralized system, and the advantages we cited earlier will become disadvantages.

2. **Cost:** Increased complexity means that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralized DBMS. Furthermore, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional labor costs to manage and maintain the local DBMSs and the underlying network.

3. **Security:** In a centralized system, access to the data can be easily controlled. However, in a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but the network itself has to be made secure. In the past, networks were regarded as an insecure communication medium. Although this is still partially true, significant developments have been made to make networks more secure.

4. **Integrity control more difficult:** Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Enforcing integrity constraints generally requires access to a large amount of data that defines the constraint but that is not involved in the actual update operation itself. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be prohibitive.

5. **Lack of standards:** Although distributed DBMSs depend on effective communication, we are only now starting to see the appearance of standard communication and data access protocols. This lack of standards has significantly limited the potential of distributed DBMSs. There are also no tools or methodologies to help users convert a centralized DBMS into a distributed DBMS.

6. **Lack of experience:** General-purpose distributed DBMSs have not been widely accepted, although many of the protocols and problems are well understood. Consequently, we do not yet have the same level of experience in industry as we have with centralized DBMSs. For a prospective adopter of this technology, this may be a significant deterrent.

7. **Database design more complex:** Besides the normal difficulties of designing a centralized database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication.

11.4 Components of Distributed Database system

The different components of DDBMS are as follows:

- **Computer workstations or remote devices (sites or nodes)** that form the network system. The distributed database system must be independent of the computer system hardware.
- **Network hardware and software components** that reside in each workstation or device. The network components allow all sites to interact and exchange data. Because the components—computers, operating systems, network hardware, and so on—are likely to be supplied by different vendors, it is best to ensure that distributed database functions can be run on multiple platforms.

- **Communications media** that carry the data from one node to another. The DDBMS must be communications media-independent; that is, it must be able to support several types of communications media.
- **The transaction processor (TP),** which is the software component found in each computer or device that requests data. The transaction processor receives and processes the application's data requests (remote and local). The TP is also known as the application processor (AP) or the transaction manager (TM).
- **The data processor (DP),** which is the software component residing on each computer or device that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS.

11.5 Distributed Database Design

Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments. Fragmentation can be of three types:

- Horizontal Fragmentation
  - Primary Horizontal Fragmentation
  - Derived Horizontal Fragmentation
- Vertical Fragmentation
- Hybrid Fragmentation

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "**reconstructiveness**".

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table.

Vertical fragmentation can be used to enforce privacy of data. For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema.

Student

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
11	Abin	Ktm	2
12	Arav	Ktm	1
13	Ashna	Palpa	3
14	Anju	Pokhara	4
15	Manish	Banepa	2
16	Pinky	Ktm	1

Figure 11.3: Student table before fragmentation

Now, the address details are maintained in the admin section. In this case, the designer will fragment the database as follows:

```
CREATE TABLE Std_address AS
    SELECT Stu_id, Stu_address
    FROM Student;
```

Std_address

Stu_id	Stu_address
10	Palpa
11	Ktm
12	Ktm
13	Palpa
14	Pokhara
15	Banepa
16	Ktm

Figure 11.4 Std_address Table (Vertical Fragment of Student Table)

Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

For example, in the student schema which is shown in figure 11.3, if the details of all students of department 1 need to be maintained at the respective faculty, then the designer will horizontally fragment the database as follows:

```
CREATE TABLE Department AS
        SELECT * FROM Student
        WHERE Dept_id=1;
```

Department

Stu_id	Stu_name	Stu_address	Dept_id
10	Maya	Palpa	1
12	Arav	Ktm	1
16	Pinky	Ktm	1

Figure 11.5 Department Table (Horizontal Fragment of Student Table)

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task. Hybrid fragmentation can be done in two alternative ways:

1. At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
2. At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

For example,

```
CREATE TABLE Hybrid AS
        SELECT Stu_id, Stu_name FROM Student
        WHERE Stu_id=12;
```

Hybrid

Stu_id	Stu_name
12	Arav

Figure 11.6 Hybrid Table (hybrid fragment of Student Table)

1. Define distributed database and distributed database management system.
2. Explain how distributed database and distributed database management system are related with distributed database system.
3. What are the motivation for DDBMS?
4. What are main reasons for and potential advantages of distributed database?
5. What are the additional functions does a DDBMS have over centralized DBMS?
6. What are the types of DDBMS?
7. Differentiate between homogenous and heterogeneous DDBMS.
8. Write the characteristics of DDBMS.
9. Explain the advantages of `DDBMS.
10. Explain the disadvantages of DDBMS.
11. Discuss the relative advantages of centralized and distributed databases.
12. Describe the different components of DDBMS.
13. Define fragmentation with example.
14. Write the advantages of fragmentation.
15. What are the disadvantages of fragmentation?
16. Explain the vertical fragmentation with example.
17. Discuss horizontal fragmentation with example.
18. What do you mean by hybrid fragmentation? Explain.
19. Why is fragmentation a useful concept in distributed database design?
20. Explain the fault tolerance nature of DDBMS.

❑❑❑

BIBLIOGRAPHY

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc..

Connolly, T. M., & Beg, C. E. (2015). *Database systems: a practical approach to design, implementation, and management*.

Date, C. J. (1977). *An introduction to database systems* (Vol. 1). Pearson Education India.

Elmasri, R., & Navathe, S. (2010). *Fundamentals of database systems*. Addison-Wesley Publishing Company.

Gupta, S. B., & Mittal, A. (2009). *Introduction to Database Management System*. Laxmi Publications, Ltd..

Kahate, A. (2004). *Introduction to Database Management Systems*. Pearson Education India.

Kroenke, D. M., Auer, D. J., Vandenberg, S. L., & Yoder, R. C. (2010). *Database concepts*. Prentice Hall.

Özsu, M. T., & Valduriez, P. (2011). *Principles of distributed database systems*. Springer Science & Business Media.

Ramakrishnan, R., & Gehrke, J. (2000). *Database management systems*. McGraw Hill.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). DatabaseSystem Concepts.

Teorey, T. J. (1999). *Database modeling and design*. Morgan Kaufmann.

❑❑❑

Name:	Bhupendra Singh Saud
Qualification:	M. Sc in Computer science and Information Technology
Position:	IT officer at Citizen Investment Trust (CIT)
Teaching Experience:	More than 8 years in under graduate and 2 years in graduate level
Faculty member:	New Summit College, Henrey Ford International College.
Co-Author:	Numerical Methods, Computer Networks, Data Structure with JAVA, Programming in C, Computer Graphics, Operating System

Name:	Indra Chaudhary
Academic Qualification:	M. Sc. in Computer Science
Global Certification Course: (Certified and Appeared)	Red Hat Certified System Administrator (RHCSA) and Red Hat Certified Engineer (RHCE) in Red Hat Enterprise Linux (RHEL) based on RHEL 6, RHCSA in RHEL OpenStack 6.0 based on RHEL 7, Red Hat Certified Specialist in Ansible Automation (Ansible 2.3) based on RHEL 7.3, DBA Oracle's 11g.
Teaching Experience:	More than decade in academic field including 5 years devotion in under graduate level.
Faculty Member at:	Academia International College, College of Applied Business, New Summit College, Sagarmatha College of Science and Technology
Co-Author:	Computer Graphics, Computer Networks

KEC
Publication and Distribution (P.) Ltd.
Kathmandu, Nepal, Tel.: 01-4168301, 01-4241777
E-mail: kecpublication14@gmail.com