

CSBG028T

INTRODUCTION TO DATABASE



DR. C.V. RAMAN UNIVERSITY

Kargi Road, Kota, Bilaspur, (C. G.),

Ph. 07753-253801, 07753-253872

E-mail: info@cvru.ac.in, Website: www.cvru.ac.in

CSBG028T

INTRODUCTION TO DATABASE

Credit- 4

Subject Expert Team

Dr. Swarangini Sinha

Professor, Rabindranath Tagore University,
Bhopal

Dr. Ayush Kumar Agrawal

Assistant Professor, Dr. CV Raman
University, Bilaspur

Dr. Ashish Pandey

Assistant Professor, Dr. CV Raman
University, Bilaspur

Dr. Rajendra Gupta

Professor, Rabindranath Tagore
University, Bhopal

Course Edited By:

Mr. Vishwas Victor

Assistant Professor, Atal Bihari Vajpai Vishwavidyalaya, Bilaspur

Unit Written By:

Dr. Pritendra Malakar

Assistant Professor, Dr. CV Raman University, Bilaspur

Warning: All rights reserved, No part of this publication may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the publisher.

Edition: 2026

Publish by: DR. C.V. RAMAN UNIVERSITY

Kargi Road, Kota, Bilaspur, (C. G.),

Ph. 07753-253801, 07753-253872

STRUCTURE OF PROGRAM

Title of the Program	: BACHELOR OF COMPUTER APPLICATION
Program Type	: UNDERGRADUATE
Program Credits	: 120
Program Duration	: 3 YEARS
Eligibility Criteria	: 12th PASS

PROGRAMME OBJECTIVE

- To produce employable IT workforce, that will have sound knowledge of IT and business fundamentals that can be applied to develop and customize solutions for Small and Medium Enterprises (SME).
- To develop skilled manpower in the various areas of information technology like. Database management, Software Development, Computer-Languages, Software engineering, Web based applications etc.
- To Work professionally with positive attitude as an individual or in multidisciplinary teams and communicate effectively.

PROGRAMME OUTCOME

PO 1. Develop competence in basic technical subjects in computer applications like Programming Languages, Data Structures, Databases, Operating Systems, Software Engineering.

PO 2. Identify, analyze, formulate and develop computer applications.

PO 3. Map real life scenarios to various theoretical optimal solutions.

PO 4. Provide simplest automated solutions to various legacy systems.

PO 5. Use modern computing tools and techniques with confidence.

PROGRAMME SPECIFIC OUTCOME

PSO 1. Able to implement computer skill in the areas related to multimedia and website design, Data base, Hardware and networking.

PSO 2. Able to demonstrate basic knowledge in the areas such as, Software Engineering, Data communication and Networking, Data base management, Web Technology and Operating Systems for building IT applications.

PSO 3. Students will be able to understand, analyze and develop computer programs in the areas related to algorithm, system software, web design and networking for efficient design of computer-based system.

PSO 4. Apply standard software engineering practices and strategies in software project development using open-source programming environment to deliver a quality of product for business success.

PSO 5. Student will be able to know various issues, latest trends in technology development and thereby innovate new ideas and solutions to existing problems.

STRUCTURE OF COURSE

Course Title	:	INTRODUCTION TO DATABASE
Course Type	:	UNDERGRADUATE
Course Code	:	CSBG028T
Course Credits	:	4
Semester	:	IVth SEM

COURSE LEARNING OBJECTIVES:

- Explain the architecture and components of DBMS, including its structure, functions, and advantages over traditional file systems.
- Compare various data modeling techniques such as Hierarchical, Network, Relational, and ER models for effective database design.
- Differentiate and apply different types of keys in the relational model, including primary key, foreign key, candidate key, alternate key, composite key, and super key.
- Design and manage database structures using Data Definition Language (DDL) commands such as CREATE, ALTER, and DROP.
- Retrieve and manipulate data using Data Manipulation Language (DML) commands such as INSERT, UPDATE, DELETE, and SELECT.

COURSE LEARNING OUTCOMES:

- CO-1 Define the DBMS architecture.
- CO-2 Compare to various Data Modelling techniques.
- CO-3 Demonstrate the different types of keys.
- CO-4 Apply Data Definition Language in DBMS.
- CO-5 Utilize Data Manipulation Language (DML),

Course Content

BLOCK I	FUNDAMENTALS OF DATABASE SYSTEMS	01-66
Unit 1	Introduction to Database Systems	03
Unit 2	DBMS Architecture	29
Unit 3	Data Abstraction and Data Independence	47
BLOCK II	DATABASE ENVIRONMENT AND ADMINISTRATION	67-108
Unit 4	Database Users and DBA	69
Unit 5	Front-End Tools and Interfaces	87
BLOCK III	DATA MODELLING AND ER DESIGN	109-168
Unit 6	Basics of Data Modelling	111
Unit 7	Relationship Model	129
Unit 8	ER Diagrams and Database Design	147
BLOCK IV	RELATIONAL DATA MODEL	169-226
Unit 9	Relational Model Concepts	171
Unit 10	Relational Constraints	187
Unit 11	Keys in Relational Model	203
BLOCK V	STRUCTURED QUERY LANGUAGE (SQL)	227-284
Unit 12	Introduction to SQL and DDL	229
Unit 13	Table Operations using DDL	247
Unit 14	Data Manipulation Language (DML)	263

BLOCK I	FUNDAMENTALS OF DATABASE SYSTEMS	01-66
Unit 1	Introduction to Database Systems	03
Unit 2	DBMS Architecture	29
Unit 3	Data Abstraction and Data Independence	47



Block 1: Fundamentals of Database Systems introduces the basic concepts that form the foundation of database technology. It begins by explaining the difference between data (raw facts and figures) and information (processed and meaningful data). The block then defines a database as an organized collection of related data and a Database Management System (DBMS) as the software that manages, stores, retrieves, and controls access to that data. It highlights the limitations of traditional file systems—such as data redundancy, inconsistency, and poor security—and explains how DBMS overcomes these problems through centralized control. The block also discusses key concepts like data models, database architecture, data abstraction, schema and instance, database users (such as Database Administrator and end users), and the importance of data integrity, security, backup, and recovery. Overall, this block builds a strong conceptual understanding of how database systems work and why they are essential for modern organizations.

UNIT 1

INTRODUCTION TO DATABASE SYSTEMS

NOTE

UNIT STRUCTURE

- 1.1 Introduction
- 1.2 Objective
- 1.3 Introduction to Database
- 1.4 File Systems vs DBMS
- 1.5 Advantages of DBMS
- 1.6 Applications of DBMS
- 1.7 Summary
- 1.8 Glossary
- 1.9 Answer to Check Your Progress
- 1.10 Suggested Readings – OER
- 1.11 Reference
- 1.12 Exercise

1.1 INTRODUCTION

A database is an organized collection of related data that can be stored, accessed, managed, and updated efficiently. In modern organizations, enormous amounts of information are generated through business transactions, customer interactions, educational activities, banking operations, healthcare services, and online applications. Managing such information efficiently requires a systematic approach, which is provided by database systems. A Database Management System (DBMS) is software that enables users and applications to create, store, retrieve, modify, and manage data in an organized manner. This unit introduces the fundamental concepts of database systems and explains why they are preferred over traditional file-processing systems. It focuses on the basic characteristics of databases, the meaning and functions of DBMS, and the major components of a database environment. The unit also introduces concepts such as data sharing, data security, data consistency, data integrity, and controlled access. Students will also explore the advantages of database systems, including reduced data redundancy, improved accessibility, better security, and efficient data management. The unit provides a foundation for understanding more advanced database concepts discussed in subsequent units. By developing a clear understanding of database fundamentals, learners can appreciate the importance of DBMS in designing and managing reliable information systems.

NOTE

1.2 OBJECTIVES

After studying this unit, you will be able to:

- Define the basic concepts and terminology of database systems.
- Explain the major components and functions of a Database Management System.
- Differentiate between traditional file-processing systems and database systems.
- Analyze the advantages and limitations of using database management systems.
- Evaluate the role of database systems in different real-world applications.

1.3 INTRODUCTION TO DATABASE

A Database is an organized collection of related data stored in a structured format so that it can be easily accessed, managed, and updated.

Meaning of Data: Data refers to raw facts and figures that by themselves have no meaning.

Examples: Student Name: Ankur, Roll No: 101 and Marks: 85

When data is processed and organized, it becomes information.

Example: “Ankur scored 85 marks in Database Systems” → This is information.

Data refers to raw facts, figures, symbols, or observations that are collected from various sources. These facts do not carry any meaning by themselves unless they are processed. Data can exist in many forms, such as numbers, text, images, audio, or video. Examples of data include student roll numbers, marks obtained in subjects, dates of birth, employee IDs, product prices, or transaction amounts.

Data can be classified into different types based on its nature and usage:

- **Numeric Data:** Consists of numbers such as age, salary, marks, or quantities.
- **Textual (Alphanumeric) Data:** Includes characters, words, and symbols, such as names, addresses, and codes.

NOTE

- **Image Data:** Photographs, scanned documents, and diagrams.
- **Audio and Video Data:** Voice recordings, music files, and video clips.

In database systems, data is stored in structured formats such as tables, records, and fields. Each data item represents a single attribute or fact. For example, in a student database, fields like *Roll Number*, *Name*, and *Marks* represent individual data items.

Characteristics of Data: Data has the following key characteristics:

1. **Raw and Unprocessed** – Data is collected directly from sources and has not been analyzed.
2. **Large in Volume** – Organizations generate massive amounts of data daily.
3. **Context-Free** – Data alone does not explain its purpose or meaning.
4. **Input for Processing** – Data acts as input to computer systems and databases.

Because of these characteristics, data by itself cannot support decision-making. It must be converted into information through processing.

Concept of Information: Information is processed, organized, and structured data that provides meaning and value to users. When data is analyzed, summarized, classified, or interpreted, it becomes information. Information answers Qs such as *what*, *when*, *where*, and *how much*, making it useful for planning and decision-making.

For example:

- Data: Marks obtained by students in different subjects
- Information: Result sheet showing pass/fail status and average marks

In a business context, daily sales figures (data) are processed to generate monthly sales reports (information). In educational institutions, raw attendance data is converted into attendance percentage reports, which help in evaluating student performance.

Characteristics of Information

Good-quality information has the following characteristics:

1. **Accuracy** – Information should be free from errors.
2. **Relevance** – It must be related to the user's needs.
3. **Timeliness** – Information should be available when required.

NOTE

4. **Completeness** – All necessary data should be included.
5. **Clarity** – Information should be easy to understand.

Databases play a vital role in ensuring these characteristics by maintaining data integrity, consistency, and security.

Data Processing: From Data to Information: The transformation of data into information occurs through a process known as the data processing cycle, which includes the following stages:

1. **Data Collection** – Gathering raw facts from various sources.
2. **Data Input** – Entering data into a computer system or database.
3. **Processing** – Sorting, classifying, calculating, and analyzing data.
4. **Output** – Producing meaningful information such as reports or summaries.
5. **Storage** – Saving data and information for future use.

Database Management Systems automate this cycle, enabling fast and accurate processing of large volumes of data.



Figure 1.1: Introduction to Database

Relationship Between Data and Information: Data and information are closely related but not identical. Data acts as the raw material, while information is the finished product. Without data, information cannot exist, and without processing, data remains meaningless. In modern organizations, databases serve as repositories where data is stored

systematically so that meaningful information can be generated whenever required.

NOTE

Importance of Data and Information in Databases: In database environments, proper management of data and information leads to several benefits:

- Improved decision-making
- Reduced data redundancy
- Enhanced data security
- Quick retrieval of information
- Better organizational efficiency

Databases ensure that the same data can be used to generate different types of information for different users, such as managers, teachers, students, or administrators.

Real-Life Examples

- ⇒ **Education:** Student marks (data) → Result analysis (information)
- ⇒ **Banking:** Transaction records (data) → Account statements (information)
- ⇒ **Healthcare:** Patient details (data) → Medical reports (information)

Meaning of Database: Database concepts focus on how data is collected, processed, and transformed into useful information for decision-making. Databases are managed using specialized software known as Database Management Systems (DBMS). One of the primary goals of database systems is to reduce data redundancy and ensure data consistency. In a database environment, data is shared among multiple users and applications while maintaining accuracy and security. Database concepts also include data integrity, security, concurrency control, and backup mechanisms. Databases support multiple users simultaneously, allowing organizations to handle large volumes of data efficiently. They also provide flexibility by allowing data to be retrieved using queries based on specific conditions. These features make databases essential for modern information systems.

Key Characteristics:

- ⇒ Data is related
- ⇒ Data is stored in tables
- ⇒ Data can be shared
- ⇒ Data is controlled centrally

Example: A Student Database may contain:

NOTE

- ⇒ Student Table
- ⇒ Course Table
- ⇒ Marks Table

Meaning of DBMS: A Database Management System (DBMS) is software that allows users to create, maintain, and control access to a database.

It acts as an interface between Users, Database and Application programs.

Examples of DBMS Software:

- ⇒ Oracle Database
- ⇒ MySQL
- ⇒ Microsoft SQL Server
- ⇒ PostgreSQL

• Functions of DBMS

1. Data Storage

Data storage refers to the process of saving data in a structured and organized manner within a database. A DBMS stores data in the form of tables (relations), consisting of rows (records) and columns (fields/attributes).

The DBMS manages:

- Physical storage on disk
- File organization
- Indexing structures
- Data compression

It ensures efficient use of memory and disk space. Data storage includes managing data types, constraints, and storage formats so that data can be accessed and processed efficiently.

Example:

Student records stored in a Student table with attributes such as Student_ID, Name, and DOB.

2. Data Retrieval

Data retrieval refers to the process of accessing or extracting data from the database. This is typically done using query languages such as SQL (Structured Query Language).

The DBMS retrieves data based on:

- Conditions (WHERE clause)

- Sorting (ORDER BY)
- Grouping (GROUP BY)
- Filtering and joins

Efficient retrieval is supported through indexing and query optimization techniques.

Example:

```
SELECT Name FROM Student WHERE Student_ID = 101;
```

This query retrieves the name of a specific student.

3. Data Manipulation

Data manipulation involves modifying the data stored in the database. This includes:

- **Insertion (INSERT)** – Adding new records
- **Updating (UPDATE)** – Modifying existing records
- **Deletion (DELETE)** – Removing records

These operations are collectively known as DML (Data Manipulation Language) commands.

Example:

```
INSERT INTO Student VALUES (101, 'Ankit', '2002-05-10');
UPDATE Student SET Name = 'Ankur' WHERE Student_ID = 101;
DELETE FROM Student WHERE Student_ID = 101;
```

The DBMS ensures that these operations maintain consistency and integrity.

4. Data Security

Data security ensures that only authorized users can access or modify the database. It protects data from unauthorized access, misuse, and breaches.

Security mechanisms include:

- User authentication (login/password)
- Authorization (roles and privileges)
- Access control (GRANT and REVOKE)
- Encryption
- Audit logs

NOTE**Example:**

Only the administrator can delete records, while normal users may only view data.

Data security is crucial in sensitive systems like banking, healthcare, and government databases.

5. Backup and Recovery

Backup is the process of creating copies of database data to prevent data loss.

Recovery is the process of restoring data in case of failure.

Failures may occur due to:

- Hardware crashes
- Power failures
- Software errors
- Cyber-attacks

DBMS uses:

- Full backups
- Incremental backups
- Transaction logs
- Checkpoints

Recovery ensures that the database returns to a consistent state after failure.

6. Concurrency Control

Concurrency control ensures that multiple users can access the database simultaneously without causing inconsistency.

In multi-user environments:

- Several users may read/write data at the same time.
- Without control, conflicts like lost updates or dirty reads can occur.

DBMS uses techniques such as:

- Locking mechanisms
- Two-phase locking (2PL)
- Timestamp ordering
- Isolation levels

NOTE**Example:**

Two users updating the same bank account balance must not overwrite each other's transactions.

Concurrency control maintains data consistency and correctness.

7. Data Integrity Enforcement

Data integrity ensures that the data stored in the database is accurate, consistent, and reliable.

DBMS enforces integrity through constraints:

1. **Entity Integrity** – Primary key cannot be NULL.
2. **Referential Integrity** – Foreign key must match a primary key in another table.
3. **Domain Integrity** – Data must follow defined data types and formats.
4. **Check Constraints** – Custom conditions (e.g., age > 18).

Example:

A student cannot enroll in a course that does not exist (foreign key constraint).

Integrity enforcement prevents invalid or inconsistent data from entering the system.

- **Why DBMS is Needed**

In modern systems:

- ⇒ Data volume is huge
- ⇒ Multiple users access data
- ⇒ Security is essential
- ⇒ Fast retrieval is required

DBMS solves these problems efficiently.

1.4 FILE SYSTEM VS DBMS

A Database Management System (DBMS) offers numerous advantages that make it an essential component of modern information systems. One of the most significant advantages is the reduction of data redundancy. In traditional file-based systems, the same data is often stored in multiple files, leading to duplication and wastage of storage space. In contrast

NOTE

A File System and a Database Management System (DBMS) are both used to store and manage data, but they differ significantly in structure, functionality, control, and efficiency.

- **File System**

A File System is a method of storing data in files on a computer's storage device. Each application program manages its own files independently.

- **DBMS (Database Management System)**

A DBMS is software that manages a structured collection of related data (database) and provides controlled access to that data.

Examples of DBMS software:

- ⇒ MySQL
- ⇒ Oracle Database
- ⇒ Microsoft SQL Server
- ⇒ PostgreSQL

Table 1.1: Comparison Table

Basis of Comparison	File System	DBMS
Structure	Data stored in separate files	Data stored in tables (structured format)
Data Redundancy	High (duplicate data common)	Controlled and minimized
Data Consistency	Poor (duplicate copies may differ)	High (centralized control)
Data Security	Limited security	Strong security with authorization & authentication
Data Integrity	Not enforced automatically	Enforced using constraints
Data Sharing	Difficult	Easy and controlled
Concurrency Control	Limited or none	Supports multiple users simultaneously

NOTE

Basis of Comparison	File System	DBMS
Backup & Recovery	Manual process	Automatic backup & recovery mechanisms
Data Independence	No data independence	Logical & physical data independence
Query Processing	Complex programming required	Easy retrieval using SQL
Relationship Support	No relationship management	Supports relationships between tables
Scalability	Not suitable for large data	Suitable for large-scale applications
Maintenance Cost	Low initial cost	Higher initial setup cost
Performance	Slower for large datasets	Optimized performance with indexing

- **Explanation of Major Differences**

(1) Data Redundancy**File System:**

- ⇒ Same data stored in multiple files.
- ⇒ Example: Student address stored in admission file and exam file.

DBMS:

- ⇒ Data stored once and shared.
- ⇒ Reduces storage waste and errors.

(2) Data Consistency**File System:**

- ⇒ If one copy is updated and another is not, inconsistency occurs.

DBMS:

- ⇒ Updates reflect everywhere due to centralized storage.

(3) Data Security

NOTE

File System:

- ⇒ Security at file level only.
- ⇒ Limited access control.

DBMS:

- ⇒ Provides:
 - User authentication
 - Role-based access
 - Privilege control (SELECT, INSERT, UPDATE, DELETE)

(4) Data Integrity

File System:

- ⇒ No automatic validation.
- ⇒ Incorrect or invalid data may be stored.

DBMS:

- ⇒ Uses constraints:
 - Primary key
 - Foreign key
 - NOT NULL
 - UNIQUE

Ensures accuracy and correctness.

(5) Data Independence

File System:

- ⇒ Changes in file structure require modification in application programs.

DBMS:

- ⇒ Provides:
 - Logical data independence
 - Physical data independence

Applications do not need modification when structure changes.

(6) Concurrency Control

File System:

- ⇒ If two users update same file → conflict may occur.

DBMS:

- ⇒ Uses locking and transaction management.
- ⇒ Ensures safe multi-user access.

(7) Backup and Recovery**File System:**

- ⇒ Manual backup required.
- ⇒ Data loss risk is high.

DBMS:

- ⇒ Automatic backup.
- ⇒ Crash recovery mechanisms.
- ⇒ Transaction rollback.

(8) Query Processing**File System:**

- ⇒ Programmer must write detailed code for data retrieval.

DBMS:

- ⇒ Uses SQL (Structured Query Language).
- ⇒ Example:

SELECT * FROM Student;

Simple and efficient.

- **Diagrammatic View**

File System Architecture

User Program → File System → Files

DBMS Architecture

User → Application → DBMS → Database

DBMS acts as an intelligent layer between user and data.

- **Advantages of File System**

- ⇒ Simple to use
- ⇒ Low cost
- ⇒ Suitable for small applications
- ⇒ No need for complex software

NOTE

- **Advantages of DBMS Over File System**

- ⇒ Better data organization
- ⇒ High security
- ⇒ Reduced redundancy
- ⇒ Improved consistency
- ⇒ Multi-user support
- ⇒ Efficient data retrieval
- ⇒ Relationship handling
- ⇒ Data integrity enforcement

Table 1.2: Use of File System and DBMS

Situation	Suitable System
Small personal project	File System
Large organization	DBMS
Multi-user environment	DBMS
Banking / E-commerce	DBMS



Activity 1.1:

List five characteristics of File System and DBMS. Also describe why DBMS is needed?

Materials Needed:

Pens and paper or digital devices for writing
Access to the internet for research (optional)

1.5 ADVANTAGES OF DBMS

A Database Management System (DBMS) offers numerous advantages that make it an essential component of modern information systems. One of the most significant advantages is the reduction of data redundancy. In traditional file-based systems, the same data is often stored in multiple files, leading to duplication and wastage of storage space. In contrast, a DBMS stores data in a centralized and structured

manner, minimizing duplication through proper database design techniques such as normalization. This not only saves storage space but also improves data accuracy.

NOTE



Figure 1.2: Advantages of DBMS

Another major advantage of DBMS is improved data consistency. When data is duplicated in multiple places, there is a high risk that one copy may be updated while another remains unchanged, resulting in inconsistent information. A DBMS ensures that updates are reflected throughout the system because the data is centrally managed. As a result, users always access accurate and up-to-date information, which is especially important in banking, healthcare, and business environments.

Data security is also a crucial benefit of DBMS. Organizations often store sensitive information such as financial records, personal details, and confidential reports. A DBMS provides security mechanisms such as user authentication, password protection, and role-based access control. Only authorized users are allowed to view or modify specific data. For example, students may be allowed to view their marks but not change them, while administrators have full access to the database.

DBMS also ensures data integrity, meaning that the data stored in the database is accurate and reliable. Integrity constraints such as primary keys, foreign keys, and validation rules prevent invalid data from being entered. For instance, a primary key ensures that no two records have the same unique identifier, and a foreign key maintains proper

NOTE

relationships between tables. These constraints help maintain correctness and reliability in the database.

Another important advantage is data sharing and concurrency control. In modern organizations, multiple users often need to access the same data simultaneously. A DBMS allows concurrent access while preventing conflicts through transaction management and locking mechanisms. This ensures that even if many users are accessing or modifying data at the same time, the database remains consistent and accurate.

The backup and recovery features of DBMS provide protection against data loss. In case of system crashes, hardware failures, or accidental deletions, the DBMS can restore the database to its previous consistent state using backup files and transaction logs. This feature is extremely important for organizations where data loss can lead to serious financial or operational consequences.

DBMS also provides data independence, which means changes in the database structure do not require changes in application programs. This makes the system flexible and easier to maintain. For example, if a new column is added to a table, the existing programs may continue to function without modification. This separation between data and programs simplifies database management.

Finally, a DBMS supports efficient data retrieval and query processing through structured query languages such as SQL. Users can easily retrieve specific data without writing complex programs. Features such as indexing and query optimization further improve performance, enabling fast access even in large databases.

In conclusion, a Database Management System offers significant advantages including reduced redundancy, improved consistency, enhanced security, strong integrity, multi-user support, backup and recovery, data independence, and efficient querying. These benefits make DBMS a reliable and powerful solution for managing data in modern organizations.

1.6 APPLICATIONS OF DBMS

A Database Management System (DBMS) is widely used in almost every sector where large amounts of data need to be stored, managed, and retrieved efficiently. Its ability to handle structured data securely and support multiple users makes it essential for modern organizations. Below are several important applications of DBMS explained in detail.

NOTE**Figure 1.3: Applications of DBMS**

One of the most important applications of DBMS is in the banking sector. Banks handle millions of transactions daily, including deposits, withdrawals, fund transfers, loan processing, and account management. A DBMS stores customer information, account details, transaction history, and loan records in an organized manner. It ensures data security, consistency, and real-time updates. For example, when a customer withdraws money from an ATM, the database is immediately updated to reflect the new balance.

DBMS is also extensively used in the education sector. Schools, colleges, and universities maintain large amounts of data such as student records, faculty information, course details, examination results, attendance, and fee records. A DBMS helps in organizing and managing this information efficiently. It allows quick retrieval of student results, automatic generation of report cards, and proper maintenance of academic records. It also supports online admission and examination systems.

In the healthcare industry, DBMS plays a crucial role in managing patient information. Hospitals use database systems to store patient records, medical history, laboratory reports, prescriptions, billing details, and doctor schedules. This helps doctors access patient history quickly and provide better treatment. The database ensures that sensitive medical data remains secure and accessible only to authorized personnel.

NOTE

Another major application of DBMS is in e-commerce and online shopping platforms. Online stores maintain product details, customer information, order history, payment records, and inventory data in databases. Whenever a customer places an order, the database updates stock levels and records transaction details. DBMS ensures smooth functioning of online transactions and supports secure payment systems.

DBMS is widely used in government organizations for managing public records such as census data, tax information, voter records, vehicle registration, and national identity databases. It helps maintain accurate and secure data for administrative and legal purposes. Government portals rely heavily on database systems to provide online services to citizens.

In the telecommunication industry, DBMS is used to maintain customer data, call records, billing information, recharge details, and service plans. Telecom companies generate vast amounts of call data records daily, and DBMS helps store and analyze this data efficiently for billing and service improvement.

The railway and airline reservation systems are also strong examples of DBMS applications. These systems manage passenger details, ticket bookings, seat availability, schedules, and cancellations. The database updates seat status in real time whenever a ticket is booked or canceled, ensuring accurate information for customers.

In the business and corporate sector, companies use DBMS for inventory management, payroll processing, employee records, sales tracking, and customer relationship management. It helps businesses make data-driven decisions by analyzing stored information.

Finally, DBMS is used in library management systems to maintain records of books, authors, members, issue dates, and return dates. It simplifies book search, issue, and return processes while keeping track of overdue fines.

In conclusion, DBMS is applied in banking, education, healthcare, e-commerce, government services, telecommunications, transportation, corporate management, and libraries. Its ability to store large volumes of data securely, support multiple users, and provide fast retrieval makes it indispensable in today's digital world.

Check Your Progress 1.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is a DBMS? Explain its advantages.

Question 2: Compare DBMS and File System.

NOTE**1.7 SUMMARY**

This unit introduces the fundamental concepts of database systems and explains why databases are essential in modern information management. The unit begins by defining data as raw facts and figures, which when processed become meaningful information. A database is described as an organized collection of related data stored in a structured format for easy access, management, and updating.

The unit then explains the concept of a Database Management System (DBMS), which is software used to create, manage, and control access to databases. A DBMS acts as an interface between users and the database, ensuring efficient storage, retrieval, and manipulation of data. It provides essential services such as security, integrity, concurrency control, and backup and recovery.

A comparison between the traditional file system and DBMS highlights the limitations of file-based systems, such as high data redundancy, inconsistency, lack of security, and poor data sharing. In contrast, DBMS overcomes these issues through centralized control and structured data organization.

The unit also discusses the major advantages of DBMS, including reduced redundancy, improved consistency, enhanced security, data integrity, multi-user support, data independence, and efficient query processing. These advantages make DBMS suitable for handling large volumes of data in complex environments.

Various applications of DBMS are explained, showing its importance in sectors such as banking, education, healthcare, e-commerce, government, telecommunications, transportation, and business

NOTE

management. These real-world applications demonstrate how database systems support daily operations and decision-making processes.

Finally, the unit introduces the basic components of a database system, including hardware, software, data, procedures, and people (such as database administrators and users). It also covers fundamental terminology like tables, records, fields, schema, and instance.

In conclusion, Unit 1 provides a strong conceptual foundation for understanding database systems. It prepares students for advanced topics such as database architecture, data modeling, relational models, and SQL by establishing a clear understanding of the purpose, benefits, and real-world significance of DBMS.

1.8 GLOSSARY

Below is a glossary of important terms covered in this unit:

1. Data

Raw facts and figures that have no meaning by themselves, such as numbers, names, or dates.

2. Information

Processed and organized data that is meaningful and useful for decision-making.

3. Database

An organized collection of related data stored in a structured format for easy access and management.

4. Database Management System (DBMS)

Software that allows users to create, manage, retrieve, and control access to a database.

5. File System

A traditional method of storing data in separate files without centralized control.

6. Data Redundancy

Duplication of data in multiple locations within a system.

7. Data Consistency

The accuracy and uniformity of data across the database.

8. Data Integrity

The correctness and validity of data maintained through rules and constraints.

9. Data Security

Protection of data from unauthorized access or modification.

NOTE**10. Concurrency Control**

A mechanism that allows multiple users to access and modify data simultaneously without conflicts.

11. Backup

A copy of database data stored separately to prevent data loss.

12. Recovery

The process of restoring the database after a failure or crash.

13. Data Independence

The ability to modify the database structure without changing application programs.

14. Logical Data Independence

The ability to change the logical schema without affecting application programs.

15. Physical Data Independence

The ability to change the physical storage structure without affecting the logical schema.

1.9 ANSWERS TO CHECK YOUR PROGRESS

Q 1: What is a DBMS? Explain its advantages.

Answer: A Database Management System (DBMS) is software that enables users to create, manage, and control access to databases. It acts as an interface between the user and the database, ensuring that data is stored in an organized manner and can be retrieved efficiently.

Advantages of DBMS:**1. Reduces Data Redundancy**

DBMS minimizes duplication of data by storing it in a centralized system.

2. Improves Data Consistency

Since data is stored in one place, updates automatically reflect everywhere, ensuring uniformity.

3. Ensures Data Security

DBMS provides authentication and authorization to prevent unauthorized access.

4. Provides Data Integrity

Rules and constraints maintain the accuracy and correctness of data.

5. Supports Data Sharing

Multiple users can access data simultaneously.

NOTE**6. Backup and Recovery**

DBMS automatically provides backup and recovery mechanisms in case of failure.

7. Data Independence

Changes in database structure do not affect application programs.

Q 2: Compare DBMS and File System.

Answer:

Basis	DBMS	File System
Data Storage	Centralized database	Separate files
Redundancy	Controlled and minimized	High redundancy
Data Consistency	High	Low
Security	Strong security features	Limited security
Data Sharing	Supports multi-user access	Limited sharing
Backup & Recovery	Available	Difficult
Data Independence	Supported	Not supported

1.10 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13706	

1.11 REFERENCES (APA 7TH EDITION)

- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson Education.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems* (3rd ed.). McGraw-Hill Education.
- Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Pearson Education.
- Connolly, T., & Begg, C. (2014). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson Education.

1.12 EXERCISE**NOTE****PART 1****A. Multiple Choice Qs**

Q1. What is a Database?

- A) A collection of programs
- B) A structured collection of related data
- C) A type of operating system
- D) A hardware device

Q2. Which of the following is an advantage of DBMS over File System?

- A) Increased data redundancy
- B) Limited data sharing
- C) Improved data security
- D) No backup facility

Q3. Which of the following reduces data redundancy?

- A) File processing system
- B) Operating system
- C) Database Management System
- D) Compiler

Q4. Who is responsible for managing and maintaining the database?

- A) End user
- B) Database Administrator (DBA)
- C) Programmer
- D) System analyst

Q5. Data independence refers to:

- A) Data stored independently in files
- B) Ability to modify database structure without affecting application programs
- C) Data stored in different computers

NOTE

D) Data without security

Explanation of MCQs

Q1 – (B) A structured collection of related data

A database is defined as an organized or structured collection of related data stored in a systematic way. The data is arranged in tables, rows, and columns so that it can be easily accessed, managed, and updated. It is not a program, hardware device, or operating system. Therefore, option B is correct.

Q2 – (C) Improved data security

One of the major advantages of a DBMS is that it provides strong security mechanisms such as authentication (username/password), authorization (access rights), and role-based access control. These features protect data from unauthorized access or modification. File systems usually lack advanced security controls. Hence, option C is correct.

Q3 – (C) Database Management System

A Database Management System (DBMS) reduces data redundancy by storing data in a centralized database instead of separate files. It ensures that the same data is not unnecessarily duplicated in multiple places. This improves consistency and reduces storage waste. Therefore, option C is correct.

Q4 – (B) Database Administrator (DBA)

A Database Administrator (DBA) is responsible for managing and maintaining the database. The DBA handles tasks such as database design, security management, backup, recovery, performance tuning, and user access control. End users only use the database, while programmers develop applications. Hence, option B is correct.

Q5 – (B) Ability to modify database structure without affecting application programs

Data independence means that changes in the database structure (schema) do not require changes in application programs. For example, modifying the physical storage of data does not affect how users access it. This is an important feature of DBMS and is not available in traditional file systems. Therefore, option B is correct.

B. Short Answer Qs (100 Words)

1. What is a database? Explain its basic characteristics.
2. What is a database system? Explain its major components.
3. Define Database and DBMS.
4. What is Data Redundancy? Why is it a problem in file systems?
5. What is Data Independence? Mention its types.

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the concept of a database management system and discuss its major functions.
2. Describe the limitations of traditional file-processing systems and explain how DBMS overcomes them.
3. Explain the different components of a database system with suitable examples.
4. Explain the advantages of DBMS over the traditional file system in detail.
5. Describe the three-level architecture of DBMS with a neat diagram.

PART 3**D. Long Answer Type Questions (300 Words)**

1. Explain the evolution from traditional file-processing systems to modern database management systems.
2. Discuss the architecture, components, functions, and importance of a database management system.
3. Compare file-based systems and database systems in terms of data redundancy, security, consistency, sharing, and maintenance.
4. Explain the major objectives of a DBMS and discuss how it improves organizational data management.
5. Discuss different real-world applications of database systems and explain how databases support decision-making.

NOTE

UNIT 2

DBMS ARCHITECTURE

NOTE

UNIT STRUCTURE

- 2.1 Introduction
- 2.2 Objective
- 2.3 Three-level Architecture
- 2.4 Internal, Conceptual & External Levels
- 2.5 Types of DBMS Architecture (1-tier, 2-tier, 3-tier)
- 2.6 Summary
- 2.7 Glossary
- 2.8 Answer to Check Your Progress
- 2.9 Suggested Readings – OER
- 2.10 Reference
- 2.11 Exercise

2.1 INTRODUCTION

DBMS architecture describes the way a database system is structured and how different components interact to provide database services to users and applications. A well-designed architecture helps organize data processing, improve system performance, support security, and make database applications easier to maintain. Understanding database architecture is therefore an essential part of database system development. This unit introduces the fundamental concepts of DBMS architecture and explains how database systems are organized at different levels. It discusses the interaction between users, applications, database servers, and stored data. Particular attention is given to two-tier and three-tier architectures, which are commonly used for developing database applications. In a two-tier architecture, the client generally communicates directly with the database server, whereas a three-tier architecture introduces an application layer between the client and database server. The unit also explains the role of different architectural components and how they contribute to database operations. Students will learn how architectural decisions influence scalability, security, performance, and maintenance. The study of DBMS architecture provides learners with the conceptual foundation required to understand how database applications communicate with databases and how different system components work together to deliver efficient and reliable database services.

2.2 OBJECTIVE

After studying this unit, students should be able to:

NOTE**1. Understand the concept of DBMS architecture**

Explain why architecture is necessary in database systems.

After studying this unit, you will be able to:

- Define the concept and purpose of DBMS architecture.
- Explain the major levels and components of database architecture.
- Differentiate between two-tier and three-tier database architectures.
- Analyze the role of client, application server, and database server in DBMS architecture.
- Evaluate the suitability of different database architectures for application development.

2.3 THREE LEVEL ARCHITECTURE

The Three-Level Architecture is a framework proposed by the ANSI/SPARC committee to separate the user view, logical structure, and physical storage of a database. Its main purpose is to achieve data abstraction and data independence by dividing the database system into three distinct levels:

- External Level
- Conceptual Level
- Internal Level

The main objective of this architecture is to achieve data abstraction and data independence. By separating user views from the logical structure and physical storage, it ensures that changes made at one level do not directly affect the other levels. This layered approach improves flexibility, security, and maintainability of the database system.

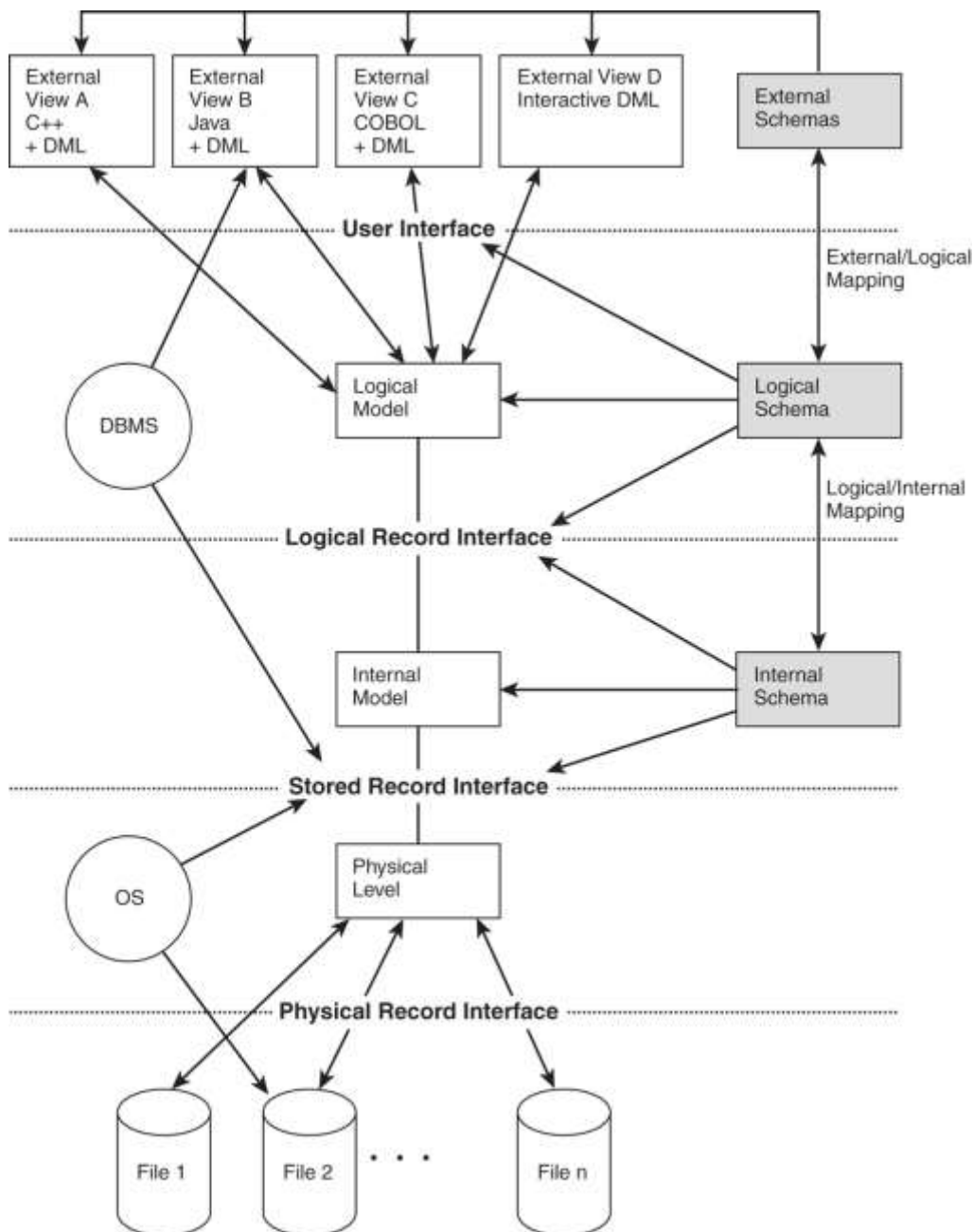
NOTE

Figure 2.1: Three Layer Architecture of DBMS

2.4 INTERNAL, CONCEPTUAL & EXTERNAL LEVELS

The Three-Level Architecture separates the database system into external, conceptual, and internal levels to ensure flexibility, security, and independence. It is a fundamental concept in DBMS architecture and helps in managing large and complex database systems efficiently.

1. External Level (View Level)

NOTE

The External Level is the highest level of abstraction. It describes how individual users view the database.

- Each user may have a different view of the same database.
- It shows only the required part of the database to a specific user.
- It hides the remaining details for simplicity and security.

For example:

- A student may see only personal details and marks.
- A teacher may see marks of all students.
- An administrator may see complete records.

This level ensures data security and customized access.

2. Conceptual Level (Logical Level)

The Conceptual Level describes the overall logical structure of the entire database.

- It defines what data is stored in the database.
- It specifies relationships between different data items.
- It includes entities, attributes, constraints, and relationships.

This level does not concern how data is physically stored.

For example:

It may define tables like:

- STUDENT (RollNo, Name, Class)
- SUBJECT (SubID, SubName)
- MARKS (RollNo, SubID, Marks)

The conceptual level acts as a bridge between the external and internal levels.

3. Internal Level (Physical Level)

The Internal Level is the lowest level of abstraction. It describes how data is physically stored in the system.

- Defines storage structures
- File organization methods
- Indexing techniques
- Data compression methods

NOTE

It deals with:

- Storage blocks
- Record placement
- Access paths

Users do not interact directly with this level.

Mapping Between Levels

There are two types of mappings:

1. External-Conceptual Mapping

Connects user views with the logical structure.

2. Conceptual-Internal Mapping

Connects the logical structure with physical storage.

These mappings help maintain data independence.

Advantages of Three-Level Architecture

1. Supports Data Independence (Logical & Physical)
2. Improves Security
3. Allows multiple user views
4. Simplifies database design
5. Makes system maintenance easier

2.5 TYPES OF DBMS ARCHITECTURE (1-TIER, 2-TIER, 3-TIER)

DBMS architecture can also be classified based on how the database system is distributed between client and server systems. The three main types are:

- **1-Tier Architecture**
- **2-Tier Architecture**
- **3-Tier Architecture**

These architectures describe how users interact with the database and where application logic and data processing occur.

1-Tier Architecture (Single-Tier Architecture)

NOTE

In 1-tier architecture, the user directly interacts with the database system. The presentation layer, application logic, and database are all on the same machine. In 1-tier architecture, all components of the database system are present on a single machine. The user directly interacts with the database without any network involvement. The presentation layer, business logic, and database are integrated into one system. This architecture is mainly used for personal or standalone applications such as locally installed database software. Since everything runs on the same computer, processing is fast and simple. However, it lacks scalability and security, making it unsuitable for large organizations or multi-user environments.

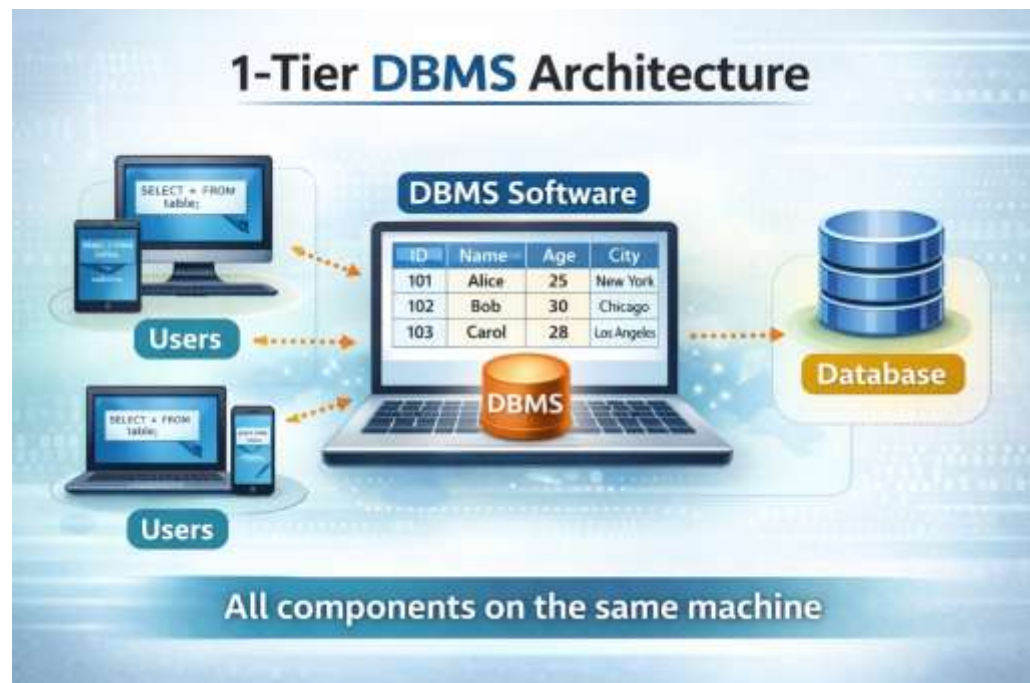


Figure 2.2: 1-Tier DBMS Architecture

Features:

- Database resides on the same computer as the user.
- No network communication required.
- Suitable for personal or standalone systems.

Example:

- MS Access running on a personal computer.
- Local MySQL installed and accessed directly.

NOTE**Advantages:**

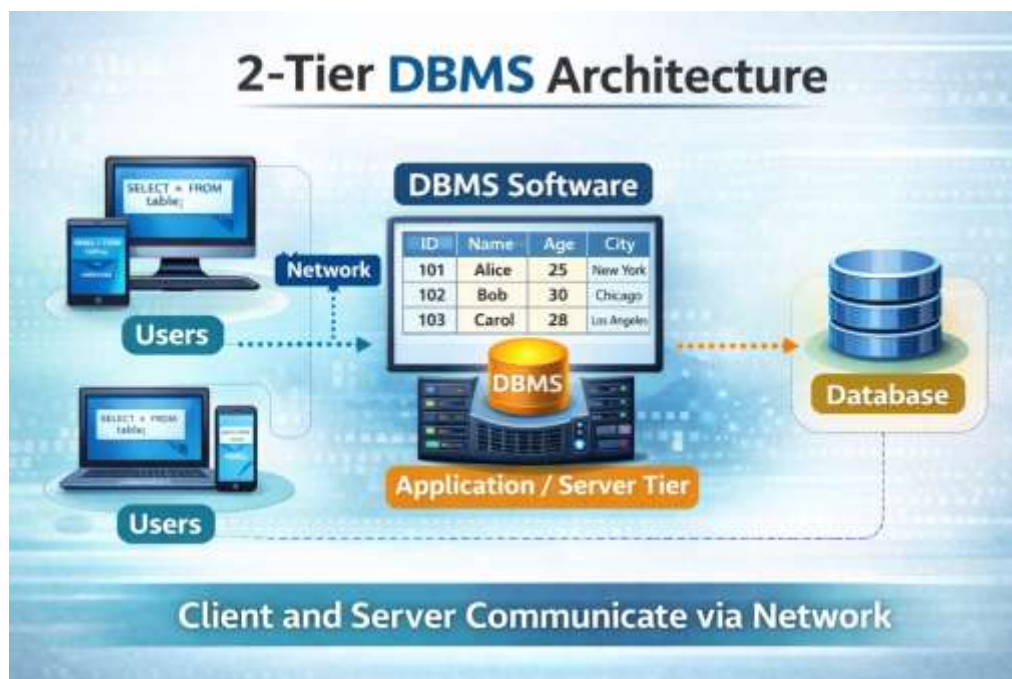
- Simple to design.
- Fast processing (no network delay).
- Easy to maintain for small systems.

Disadvantages:

- Not suitable for large organizations.
- Poor security.
- No scalability.
- Not ideal for multiple users.

2-Tier Architecture (Client–Server Architecture)

In 2-tier architecture, the database resides on a server, and the application runs on the client side. The client communicates directly with the database server. In 2-tier architecture, also known as client–server architecture, the database is stored on a server while the application runs on the client machine. The client sends requests directly to the database server using query languages such as SQL. This architecture supports multiple users and provides better security compared to the 1-tier system. It is commonly used in small to medium-sized organizations where desktop applications connect to a centralized database server. However, since business logic is usually handled at the client side, system maintenance can become complex, and heavy client requests may overload the server.



NOTE**Figure 2.3: 2-Tier DBMS Architecture****Structure:**

- Client: User interface + application logic
- Server: Database

Example:

- A desktop application connected to a centralized database server.
- Banking software connecting to a database server.

Advantages:

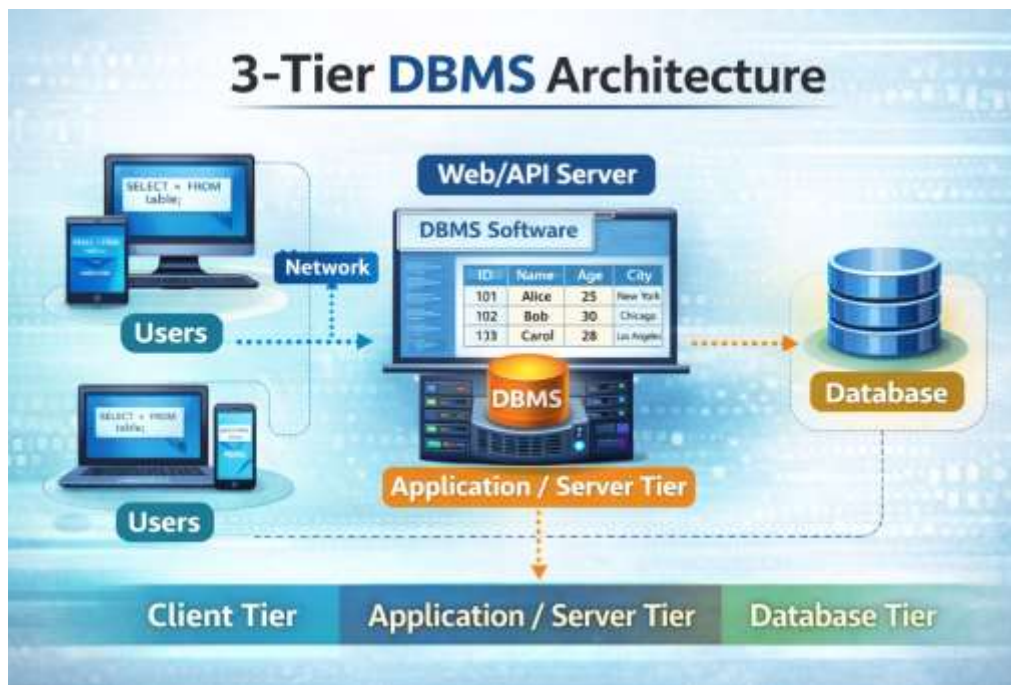
- Better security than 1-tier.
- Supports multiple users.
- Faster communication between client and database.

Disadvantages:

- Business logic stored on client side (less secure).
- Heavy load on database server.
- Not highly scalable.

3-Tier Architecture

In 3-tier architecture, the system is divided into three separate layers: the presentation layer (client), the application layer (application server), and the database layer (database server). The client interacts only with the application server, which processes requests and communicates with the database server. This separation improves security, scalability, and performance. It allows load distribution and easier maintenance because changes in one layer do not directly affect the others. Most modern web applications, such as online banking systems and e-commerce platforms, use 3-tier architecture due to its reliability and ability to handle large numbers of users efficiently.

NOTE**Figure 2.4: 3-Tier DBMS Architecture**

In 3-tier architecture, the system is divided into three layers:

1. Presentation Layer (Client)
2. Application Layer (Application Server)
3. Database Layer (Database Server)

Structure:

- Client → Application Server → Database Server

Example:

- Online shopping websites.
- Banking web applications.
- University management portals.

Advantages:

- High security.
- Better scalability.
- Easy maintenance.
- Load distribution.
- Suitable for large enterprises.

Disadvantages:

NOTE

- More complex to design.
- Higher cost.
- Requires more resources.

Table 2.1: Comparison of 1-Tier, 2-Tier, and 3-Tier Architecture

Feature	1-Tier	2-Tier	3-Tier
Number of Layers	1	2	3
Network Required	No	Yes	Yes
Scalability	Low	Medium	High
Security	Low	Moderate	High
Suitable For	Personal systems	Small/Medium organizations	Large enterprises

Overall, the choice of DBMS architecture depends on the size and requirements of the system. While 1-tier is simple and suitable for personal use, 2-tier is ideal for small organizations, and 3-tier is best suited for large-scale enterprise applications requiring high security and scalability.



Activity 2.1:

Take a table of students and define the three level of architecture.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

Check Your Progress 2.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: Explain the Three-Level Architecture of DBMS.

Question 2: Differentiate between 2-Tier and 3-Tier Architecture.

2.6 SUMMARY

Unit 2 focuses on the structure and organization of a Database Management System (DBMS). It explains how a DBMS is designed to manage data efficiently while ensuring security, flexibility, and data independence. The unit begins with an introduction to DBMS architecture and its importance in separating user interaction from physical data storage. This separation helps in simplifying database management and improving system performance.

A major topic covered in this unit is the three-level architecture (ANSI/SPARC architecture), which includes the external level, conceptual level, and internal level. The external level describes how users view the data, the conceptual level defines the logical structure of the entire database, and the internal level explains how data is physically stored. This layered design ensures logical and physical data independence, meaning changes at one level do not affect other levels.

The unit also explains the types of DBMS architecture based on system deployment: 1-tier, 2-tier, and 3-tier architecture. The 1-tier architecture is used for standalone systems, the 2-tier architecture follows a client-server model, and the 3-tier architecture separates presentation, application, and database layers for better security and scalability. These architectures are widely used in real-world database applications such as banking systems, e-commerce platforms, and enterprise software.

Overall, Unit 2 builds a clear understanding of how database systems are structured internally and externally. It highlights the importance of layered design, data abstraction, and deployment models in creating efficient, secure, and scalable database systems.

2.7 GLOSSARY

1. DBMS Architecture

The structural design of a Database Management System that defines how its components interact and how data is managed.

2. ANSI/SPARC Architecture

A standardized three-level database architecture model designed to achieve data abstraction and data independence.

3. External Level (View Level)

NOTE

The highest level of abstraction that describes how individual users view the database.

4. External Schema

The specific view of the database designed for a particular user or application.

5. Conceptual Level (Logical Level)

The middle level that defines the overall logical structure of the entire database.

6. Conceptual Schema

The complete logical description of the database, including entities, attributes, and relationships.

7. Internal Level (Physical Level)

The lowest level that describes how data is physically stored in storage devices.

8. Internal Schema

The detailed description of the physical storage structure of the database.

9. Data Abstraction

The process of hiding complex details and showing only essential information to users.

10. Data Independence

The ability to modify the database structure without affecting application programs.

11. Logical Data Independence

The ability to change the conceptual schema without altering external schemas.

12. Physical Data Independence

The ability to change the internal schema without affecting the conceptual schema.

NOTE**13. Mapping**

The process of connecting schemas between different levels (external–conceptual and conceptual–internal).

14. 1-Tier Architecture

A single-layer architecture where the user directly interacts with the database on the same system.

15. 2-Tier Architecture

A client–server architecture where the client communicates directly with the database server.

2.8 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: Explain the Three-Level Architecture of DBMS.**Answer:**

The Three-Level Architecture, also known as the ANSI/SPARC architecture, divides the database system into three levels: external level, conceptual level, and internal level.

The external level (view level) describes how individual users view the database. Different users can have different views depending on their requirements. It ensures security by hiding unnecessary details.

The conceptual level (logical level) describes the overall logical structure of the entire database. It defines entities, attributes, relationships, and constraints. It does not deal with physical storage details.

The internal level (physical level) describes how data is actually stored in the system. It includes file organization, indexing, and storage structures.

This architecture provides data abstraction and ensures logical and physical data independence.

Q 2: Differentiate between 2-Tier and 3-Tier Architecture.

NOTE**Answer:**

In 2-tier architecture, the client communicates directly with the database server. The client contains the presentation layer and business logic, while the server stores the database. It is suitable for small and medium-sized applications but may face scalability and security issues.

In 3-tier architecture, the system is divided into three layers: presentation layer (client), application layer (application server), and database layer (database server). The client interacts with the application server, which then communicates with the database. This architecture provides better security, scalability, and performance and is commonly used in large enterprise and web-based applications.

2.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13707	

2.10 REFERENCE (APA 7TH EDITION)

- Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson Education. (Sections on Three-Level Architecture and Client-Server Systems)
- Ramakrishnan, R., & Gehrke, J. (2003). Database Management Systems (3rd ed.). McGraw-Hill Education. (Chapters covering Database System Structure and Architecture)

2.11 EXERCISE**PART 1****A. Multiple Choice Qs**

1. The three-level architecture of DBMS was proposed by:

NOTE

- A) Oracle Corporation
- B) ANSI/SPARC Committee
- C) IBM
- D) Microsoft

2. Which level of the three-level architecture describes how data is physically stored?

- A) External Level
- B) Conceptual Level
- C) Internal Level
- D) View Level

3. In which architecture does the client directly communicate with the database server?

- A) 1-Tier Architecture
- B) 2-Tier Architecture
- C) 3-Tier Architecture
- D) Multi-Level Architecture

4. The main purpose of the conceptual level in DBMS architecture is to:

- A) Describe physical storage details
- B) Provide user-specific views
- C) Define the overall logical structure of the database
- D) Manage network communication

5. Which architecture provides better scalability and security for large web applications?

- A) 1-Tier Architecture
- B) 2-Tier Architecture
- C) 3-Tier Architecture
- D) File System Architecture

Explanation of MCQs

Q1 – (B) ANSI/SPARC Committee

NOTE**Explanation:**

The three-level architecture (External, Conceptual, Internal levels) was proposed by the ANSI/SPARC committee to provide data abstraction and data independence in database systems.

Q2 – C) Internal Level**Explanation:**

The internal level deals with the physical storage of data, including file organization, indexing, and storage structures. It focuses on how data is stored in memory and disks.

Q3 – B) 2-Tier Architecture**Explanation:**

In 2-tier architecture, the client application directly interacts with the database server without an intermediate application server. It is commonly used in small-scale systems.

Q4 – C) Define the overall logical structure of the database**Explanation:**

The conceptual level describes the complete logical structure of the database, including entities, attributes, relationships, and constraints, without considering physical storage details.

Q5 – C) 3-Tier Architecture**Explanation:**

3-tier architecture separates presentation, application, and database layers. This separation improves scalability, maintainability, and security, making it suitable for large enterprise and web-based applications.

B. Short Answer Qs (100 Words)

1. What is DBMS architecture?
2. Explain the concept of two-tier architecture.
3. What is the main purpose of the Three-Level Architecture in DBMS?
4. Define Data Independence.
5. What is 2-Tier Architecture in DBMS?

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the different levels of DBMS architecture.
2. Describe two-tier database architecture with its components and applications.

3. Explain three-tier database architecture and its advantages.
4. Explain the Three-Level Architecture of DBMS in detail.
5. Compare 1-Tier, 2-Tier, and 3-Tier Architecture in detail.

NOTE**PART 3****D. Long Answer Type Questions (300 Words)**

1. Explain DBMS architecture in detail and discuss the purpose of different architectural levels.
2. Discuss two-tier and three-tier architectures with suitable examples and explain their advantages and limitations.
3. Explain how three-tier architecture improves security, scalability, and maintainability of database applications.
4. Compare different DBMS architectures based on performance, security, scalability, and maintenance requirements.
5. Discuss the importance of database architecture in designing reliable and efficient database applications.

NOTE

UNIT 3

DATA ANSTRACTION AND DATA INDEPENDENCE

NOTE

UNIT STRUCTURE

- 3.1 Introduction
- 3.2 Objective
- 3.3 Data Abstraction
- 3.4 Physical & Logical Data Independence
- 3.5 Importance of Data Independence
- 3.6 Summary
- 3.7 Glossary
- 3.8 Answer to Check Your Progress
- 3.9 Suggested Readings – OER
- 3.10 Reference
- 3.11 Exercise

3.1 INTRODUCTION

Database systems manage large and complex collections of information, but users do not always need to know how data is physically stored or managed. Data abstraction provides a mechanism for hiding unnecessary implementation details while presenting users with only the information relevant to their requirements. It makes database systems easier to use and allows different users to interact with the same database at appropriate levels. This unit introduces the concept of data abstraction and explains the three major levels of database representation: physical, logical, and view levels. The physical level describes how data is actually stored, while the logical level focuses on the structure and relationships of the data. The view level provides customized representations of data for individual users or applications. The unit also introduces data independence, which refers to the ability to modify one level of the database system without requiring significant changes at another level. Physical data independence allows changes to storage structures without affecting the logical schema, while logical data independence allows changes to the logical schema without affecting user views or applications. Understanding these concepts helps students appreciate how DBMS architecture provides flexibility, security, and ease of maintenance. These concepts form an important

NOTE

foundation for studying advanced database design and management.

3.2 OBJECTIVE

After studying this unit, you will be able to:

- Define data abstraction and data independence in database systems.
- Explain the three levels of data abstraction.
- Differentiate between physical and logical data independence.
- Analyze the role of data abstraction in simplifying database management.
- Evaluate the importance of data independence in database system development and maintenance.

3.3 DATA ABSTRACTION

Data Abstraction is a fundamental concept in a Database Management System (DBMS) that focuses on hiding the complex details of how data is stored and maintained, while presenting only the necessary information to users. It simplifies interaction with the database by allowing users to work with data without understanding its physical storage structure. This makes the system easier to use, manage, and maintain.



Figure 3.1: Data Abstraction**NOTE**

In real-world applications, databases contain large amounts of data stored using complex file structures, indexing techniques, memory allocation methods, and storage devices. If users had to understand all these technical details, working with databases would become extremely difficult. Data abstraction solves this problem by separating user interaction from internal implementation details.

Purpose of Data Abstraction

The main purposes of data abstraction are:

- To reduce complexity in database systems
- To improve security by restricting access to sensitive data
- To simplify database design and maintenance
- To provide data independence
- To allow multiple users to interact with the database in different ways.

Levels of Data Abstraction

Data abstraction in DBMS is achieved through three levels, as defined in the ANSI/SPARC architecture:

1. Internal Level (Physical Level)

The internal level describes how data is physically stored in the database. It includes:

- File organization methods
- Indexing techniques
- Record storage format
- Data compression
- Storage allocation

At this level, the database administrator (DBA) manages storage structures and performance optimization. End users do not interact directly with this level.

For example, data may be stored using B-trees or hashing techniques, but users do not need to know this.

2. Conceptual Level (Logical Level)

The conceptual level describes the overall logical structure of the entire database. It includes:

NOTE

- Entities
- Attributes
- Relationships
- Constraints
- Data types

This level defines what data is stored in the database and the relationships among those data. It does not deal with physical storage details.

For example, a university database may contain entities such as Students, Courses, and Faculty, along with relationships like enrollment. This structure is defined at the conceptual level.

3. External Level (View Level)

The external level describes how individual users view the database. Different users may have different views depending on their requirements.

For example:

- A student may see only their grades.
- A faculty member may see student details and course information.
- An administrator may see complete records.

This level improves security by restricting access to sensitive information and allows customized views for different users.

Importance of Data Abstraction

Data abstraction plays a critical role in database systems because:

1. It hides unnecessary details from users.
2. It simplifies database interaction.
3. It improves system security.
4. It supports data independence.
5. It allows easier system modifications.
6. It enhances maintainability and scalability.

Without data abstraction, database systems would be complex, difficult to manage, and less secure.

Data abstraction is essential for efficient database management. By dividing the database system into internal, conceptual, and external levels, it ensures that users can interact with data easily without

worrying about technical storage details. It also enhances security, flexibility, and maintainability of the database system.

NOTE

3.4 PHYSICAL & LOGICAL DATA INDEPENDENCE

Data independence is a crucial feature of a Database Management System (DBMS). It refers to the ability to modify the database schema at one level without affecting the schema at the next higher level. It ensures flexibility, easier maintenance, and reduced application program changes.

Data independence is closely related to the three-level architecture (Internal, Conceptual, and External levels). There are two main types of data independence:

- **Physical Data Independence**
- **Logical Data Independence**

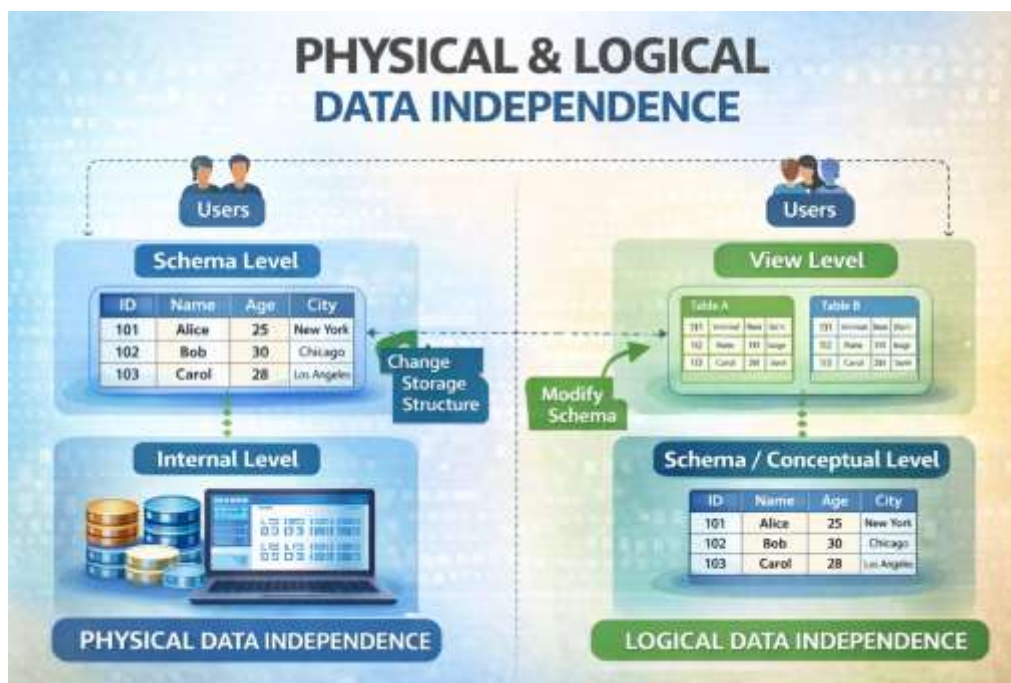


Figure 3.2: Physical & Logical Data Independence

1. Physical Data Independence

Definition

Physical Data Independence is the ability to change the internal (physical) schema without affecting the conceptual schema or application programs.

NOTE

In simple words, changes in how data is stored physically should not affect how users view or access the data.

What Can Be Changed?

At the internal level, the following changes can be made:

- Changing file organization (e.g., from sequential to indexed)
- Adding or removing indexes
- Changing storage devices (HDD to SSD)
- Modifying record storage formats
- Changing data compression techniques
- Reorganizing database files

Example

Suppose a university database stores student records using a sequential file organization. Later, the DBA decides to improve performance by adding indexing using B-tree structures.

This change happens at the internal level only. Users and application programs continue accessing student data in the same way.

Thus, the change in physical storage does not affect the logical structure or user views.

Advantages of Physical Data Independence

1. Improves system performance without affecting users.
2. Allows database tuning and optimization.
3. Reduces maintenance cost.
4. Provides flexibility in storage management.
5. Makes system upgrades easier.

Difficulty Level

Physical data independence is easier to achieve compared to logical data independence because changes at the physical level usually do not affect higher levels.

2. Logical Data Independence**Definition**

Logical Data Independence is the ability to change the conceptual schema without affecting the external schema or application programs.

NOTE

In simple words, changes in the logical structure of the database should not affect how users view the data.

What Can Be Changed?

At the conceptual level, the following changes can be made:

- Adding new attributes
- Adding new entities (tables)
- Modifying relationships
- Splitting or merging tables
- Changing constraints

Example

Suppose a student table contains:

Student (RollNo, Name, Course)

Later, the DBA decides to add a new attribute:

Student (RollNo, Name, Course, Email)

If application programs do not use the Email attribute, they should continue working without modification.

Another example: If a single table is divided into two related tables for better design, existing user views should remain unaffected.

Advantages of Logical Data Independence

1. Allows database structure expansion.
2. Supports evolving business requirements.
3. Reduces application reprogramming.
4. Improves system flexibility.
5. Helps in better database design and normalization.

Difficulty Level

Logical data independence is more difficult to achieve because changes in conceptual schema may affect multiple user views and application programs.

Table 3.1: Difference Between Physical and Logical Data Independence

NOTE

Basis	Physical Data Independence	Logical Data Independence
Level of Change	Internal Level	Conceptual Level
Affects	Physical storage details	Logical structure
Impact on Users	No impact	Should not impact user views
Ease of Implementation	Easier	More difficult
Example	Adding index	Adding new attribute

Importance of Data Independence

Data independence is important because:

- It reduces system maintenance cost.
- It increases database flexibility.
- It allows easy system modification.
- It improves scalability.
- It ensures long-term stability of application programs.

Without data independence, even small changes in the database would require rewriting application programs, which would be costly and time-consuming.

Physical and Logical Data Independence are essential features of DBMS. Physical data independence ensures that changes in storage do not affect logical structure, while logical data independence ensures that changes in database design do not affect user views or applications. Together, they provide flexibility, scalability, and maintainability in modern database systems.

**Activity 3.1:**

Create an employee table and describe the physical and logical data independence in that table.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

3.5 IMPORTANCE OF DATA INDEPENDENCE

NOTE

Data Independence is one of the most important features of a Database Management System (DBMS). It refers to the ability to modify the database schema at one level without affecting the schema at higher levels. It ensures that changes in database design or storage do not disturb application programs or user views. This feature makes DBMS more flexible, reliable, and easier to maintain.



Figure 3.3: Importance of Data Independence

The importance of data independence can be understood from several perspectives:

1. Reduces Application Program Changes

One of the biggest advantages of data independence is that it minimizes the need to modify application programs when changes are made to the database structure. In large organizations, application programs are complex and widely used. If every small change in the database required changes in programs, maintenance would become difficult and expensive. Data independence protects applications from such disruptions.

2. Simplifies Database Maintenance

NOTE

Database systems often require updates such as adding new attributes, modifying relationships, reorganizing files, or improving storage techniques. With data independence, these changes can be made without affecting end users. This simplifies maintenance and allows database administrators (DBAs) to improve the system efficiently.

3. Improves System Flexibility

Organizations grow and their data requirements change over time. New fields may need to be added, tables may need restructuring, or storage systems may need upgrades. Data independence provides flexibility to make these changes without redesigning the entire system.

4. Enhances Performance Optimization

Physical data independence allows changes in physical storage methods (such as adding indexes or changing file organization) without affecting logical design. This enables DBAs to optimize performance while keeping user interaction unchanged.

For example, if database performance is slow, an index can be added at the internal level to speed up queries without modifying application programs.

5. Supports Scalability

As organizations expand, databases grow larger and more complex. Data independence allows systems to scale smoothly. New modules, new tables, and additional data can be incorporated without affecting existing applications.

6. Increases System Reliability

If every change required rewriting application programs, the chances of errors and system failures would increase. Data independence reduces this risk by isolating changes at different levels, thereby improving system reliability.

7. Saves Time and Cost

Maintenance and modification of database systems can be expensive. By reducing the need for program changes, data independence lowers development time and operational costs. It is especially beneficial in large enterprise systems.

8. Enables Better Database Design

NOTE

Logical data independence allows database designers to improve the database structure (such as normalization or adding constraints) without affecting user views. This leads to better organization and integrity of data.

Data independence is a key feature that makes DBMS powerful and efficient. It provides flexibility, reduces maintenance efforts, improves performance, enhances scalability, and ensures system stability. Without data independence, database systems would be rigid, difficult to maintain, and costly to modify. Therefore, it plays a critical role in modern database management systems.

Check Your Progress 3.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: Explain Data Abstraction and its levels in DBMS.

Question 2: Differentiate between Physical and Logical Data Independence.

3.6 SUMMARY

Unit 3 focuses on two fundamental concepts of Database Management Systems (DBMS): Data Abstraction and Data Independence. These concepts are essential for designing flexible, secure, and efficient database systems.

Data Abstraction refers to the process of hiding complex storage and implementation details from users while showing only the necessary information. It simplifies interaction with the database and makes the system user-friendly. Data abstraction is achieved through three levels of architecture: the Internal Level (physical storage details), the Conceptual Level (overall logical structure of the database), and the External Level (user-specific views). These levels help in organizing the database system and separating user interaction from technical implementation.

NOTE

Data Independence is the ability to modify the database schema at one level without affecting the next higher level. It ensures that changes in database design or storage do not disturb application programs or user views. There are two types of data independence: Physical Data Independence, which allows changes in physical storage without affecting the logical structure, and Logical Data Independence, which allows changes in the logical structure without affecting user views or applications.

This unit emphasizes that data independence reduces maintenance cost, improves flexibility, enhances system performance, and supports scalability. Together, data abstraction and data independence form the foundation of a well-structured DBMS, ensuring that database systems remain stable, adaptable, and efficient even when changes are required.

3.7 GLOSSARY

1. Data Abstraction

The process of hiding complex storage and implementation details while showing only essential information to users.

2. Data Independence

The ability to modify the database schema at one level without affecting the schema at higher levels.

3. Internal Level (Physical Level)

The lowest level of database architecture that describes how data is physically stored in storage devices.

4. Conceptual Level (Logical Level)

The middle level that describes the overall logical structure of the entire database, including entities, attributes, and relationships.

5. External Level (View Level)

The highest level that describes how individual users view the database. Different users may have different views.

6. Physical Data Independence

The ability to change the internal schema (physical storage details) without affecting the conceptual schema.

NOTE**7. Logical Data Independence**

The ability to change the conceptual schema without affecting the external schema or application programs.

8. Schema

The overall design or structure of a database.

9. Internal Schema

The description of the physical storage structure of the database.

10. Conceptual Schema

The description of the logical structure of the entire database.

11. External Schema

The description of user-specific views of the database.

12. Database Administrator (DBA)

The person responsible for managing and maintaining the database system.

13. Data Security

Protection of data from unauthorized access or modification.

14. Data Integrity

Accuracy and consistency of data stored in a database.

15. Mapping

The process of transforming data between different levels of abstraction (e.g., conceptual-to-internal mapping).

3.8 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: Explain Data Abstraction and its levels in DBMS.

Answer:

NOTE

Data Abstraction is the process of hiding complex implementation and storage details of a database while presenting only essential information to users. It makes the database system simple, secure, and easy to use.

Data abstraction is achieved through three levels:

1. Internal Level (Physical Level):

This level describes how data is physically stored in the database. It includes file structures, indexing methods, and storage allocation. It focuses on performance and storage optimization.

2. Conceptual Level (Logical Level):

This level describes the overall logical structure of the entire database. It defines entities, attributes, relationships, and constraints without considering physical storage details.

3. External Level (View Level):

This level describes how individual users view the database. Different users may have different views according to their needs. It improves security by restricting access to sensitive information.

Thus, data abstraction simplifies database interaction and supports data independence.

Q 2: Differentiate between Physical and Logical Data Independence.

Answer:

Data independence refers to the ability to modify the database schema at one level without affecting the higher levels. It is of two types:

Physical Data Independence:

It is the ability to change the internal schema without affecting the conceptual schema. For example, adding an index or changing file organization does not affect user views or application programs. It is easier to achieve.

Logical Data Independence:

It is the ability to change the conceptual schema without affecting the external schema or application programs. For example, adding a new

NOTE

attribute to a table should not disturb existing applications. It is more difficult to achieve.

Physical data independence deals with storage changes, while logical data independence deals with structural changes.

3.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13708	

3.10 REFERENCE

- Ramakrishnan, R., & Gehrke, J. (2003). Database Management Systems (3rd ed.). McGraw-Hill Education.
- Connolly, T., & Begg, C. (2014). Database Systems: A Practical Approach to Design, Implementation, and Management(6th ed.). Pearson Education.

3.11 EXERCISE

PART 1

A. Multiple Choice Qs

1. Data abstraction in DBMS is used to:

- A) Increase data redundancy
- B) Hide complex storage details from users
- C) Delete unwanted data
- D) Improve internet speed

2. Which level of abstraction describes the overall logical structure of the database?

NOTE

A) Internal Level

B) Physical Level

C) Conceptual Level

D) External Level

3. Physical data independence refers to the ability to change:

A) User views without changing storage

B) Internal schema without affecting conceptual schema

C) Application programs without modifying database

D) Database users without changing schema

4. Logical data independence is more difficult to achieve because:

A) It deals with hardware changes

B) It affects physical storage only

C) Changes in conceptual schema may affect user views

D) It requires internet connectivity

5. Which of the following is an example of logical data independence?

A) Adding an index to improve performance

B) Changing file storage format

C) Adding a new attribute to a table without affecting existing applications

D) Changing disk storage device

Explanation of MCQs

Q1 – B) Hide complex storage details from users

Explanation:

Data abstraction hides the internal implementation and storage details of the database while presenting only essential information to users. This simplifies database usage.

NOTE**Q2 – C) Conceptual Level**

Explanation:

The conceptual level defines entities, attributes, relationships, and constraints. It describes what data is stored and how it is logically structured, without considering physical storage details.

Q3 – B) Internal schema without affecting conceptual schema

Explanation:

Physical data independence allows changes in physical storage methods (like indexing or file organization) without affecting the logical structure or application programs.

Q4 – C) Changes in conceptual schema may affect user views

Explanation:

Logical data independence involves changes in the conceptual schema. Since user views and application programs depend on the conceptual schema, achieving independence is more complex.

Q5 – C) Adding a new attribute to a table without affecting existing applications

Explanation:

Logical data independence allows modifications in the logical structure (such as adding new attributes) without disturbing existing user views or application programs.

B. Short Answer Qs (100 Words)

1. What is Data Abstraction in DBMS?
2. Define Physical Data Independence.
3. Why is Logical Data Independence difficult to achieve?
4. What is view level of data abstraction?
5. Define data independence and explain its importance.

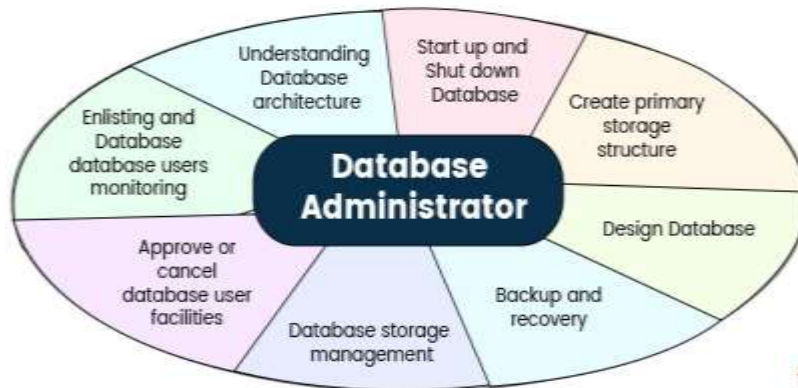
PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the three levels of Data Abstraction in detail.
2. Discuss the importance of Data Independence in DBMS.
3. Differentiate between physical and logical data independence.
4. Explain how data abstraction simplifies database management.
5. Explain the relationship between data abstraction and database architecture.

NOTE**D. Long Answer Type Questions (300 Words)**

1. Explain the three-level architecture of a DBMS and discuss how data abstraction is achieved at each level.
2. Discuss physical and logical data independence with suitable examples.
3. Explain the importance of data independence for database maintenance and system development.
4. Describe how data abstraction protects users from unnecessary details of database implementation.
5. Analyze the relationship between data abstraction, data independence, and database system flexibility.

BLOCK II	DATABASE ENVIRONMENT AND ADMINISTRATION	67-108
Unit 4	Database Users and DBA	69
Unit 5	Front-End Tools and Interfaces	87



Block II: Database Environment and Administration focuses on the overall working environment of a Database Management System (DBMS) and the responsibilities involved in managing it effectively. The database environment consists of several components such as hardware, software, data, procedures, and people. Hardware includes physical devices like servers, storage systems, and networking equipment. Software includes the DBMS itself, operating systems, and application programs. Data refers to the actual stored information along with metadata. Procedures are the rules and instructions for using and managing the database. People include database administrators (DBAs), designers, developers, and end users who interact with the system.

A major part of this block is database administration, which deals with the management and control of the database system. The Database Administrator (DBA) plays a central role in this environment. The DBA is responsible for database installation, configuration, security management, performance monitoring, backup and recovery, user authorization, and ensuring data integrity. Effective database administration ensures that the database system operates smoothly, securely, and efficiently.

This block also emphasizes data security, access control, backup strategies, recovery mechanisms, and performance tuning. In modern organizations, where databases support critical operations such as banking, healthcare, education, and e-commerce, proper administration is essential to prevent data loss, unauthorized access, and system failures.

Overall, Block II highlights how a database system functions within an organizational environment and explains the importance of proper administration for maintaining reliability, security, and performance of the DBMS.

UNIT 4

DATABASE USERS & DBA

NOTE

UNIT STRUCTURE

- 4.1 Introduction
- 4.2 Objective
- 4.3 Role of DBA
- 4.4 Responsibilities of DBA
- 4.5 Database Users
- 4.6 End Users
- 4.7 Summary
- 4.8 Glossary
- 4.9 Answer to Check Your Progress
- 4.10 Suggested Readings – OER
- 4.11 Reference
- 4.12 Exercise

4.1 INTRODUCTION

A database system involves several categories of users who interact with data according to their responsibilities and requirements. From ordinary users accessing information through applications to specialists responsible for designing and managing databases, each participant has a distinct role. Effective coordination among these users is essential for maintaining reliable, secure, and efficient database operations. This unit introduces different types of database users and explains their responsibilities within a database environment. Users may include database administrators, database designers, application programmers, system analysts, and end users. Each category contributes differently to the development, operation, and maintenance of a database system. The unit gives particular emphasis to the role of the Database Administrator, commonly known as the DBA. A DBA is responsible for several important activities, including database security, access control, backup and recovery, performance monitoring, storage management, and maintaining data integrity. The unit also explains the role of database designers in developing appropriate database structures and the responsibilities of application programmers who create software that interacts with databases. By studying these roles, students will gain an understanding of how responsibilities are distributed within a database environment. This knowledge is essential for appreciating the organizational and technical aspects involved in managing modern database systems.

NOTE

4.2 OBJECTIVE

After studying this unit, you will be able to:

- Identify different types of database users and their roles.
- Explain the functions and responsibilities of a Database Administrator.
- Differentiate between the roles of database designers, application programmers, and end users.
- Analyze the importance of database administration for database security and performance.
- Evaluate the role of a DBA in maintaining reliable and efficient database operations.

4.3 ROLE OF DBA

The Database Administrator (DBA) is the central authority responsible for managing, maintaining, and controlling the entire Database Management System (DBMS). The DBA ensures that the database operates efficiently, securely, and reliably. The role is both technical and managerial, requiring a deep understanding of database systems, organizational requirements, and security practices.



Figure 4.1: Role of DBA

The detailed roles and responsibilities of a DBA are explained below:

1. Database Installation and Configuration

One of the primary responsibilities of the DBA is installing the DBMS software and configuring it according to organizational requirements. This includes setting up the database environment, allocating storage, configuring memory settings, and ensuring proper connectivity between the database server and client systems.

2. Database Design and Implementation

The DBA participates in designing the database structure. This includes defining tables, relationships, constraints, indexes, and storage structures. The DBA ensures that the database schema is efficient, scalable, and aligned with business requirements.

3. Security Management and Access Control

Data security is one of the most critical responsibilities of a DBA. The DBA:

- Creates user accounts
- Grants and revokes access privileges
- Defines roles and permissions
- Implements authentication and authorization mechanisms

This ensures that only authorized users can access or modify data, protecting the database from unauthorized access and cyber threats.

4. Backup and Recovery Management

The DBA is responsible for protecting data against loss due to system failures, hardware crashes, or human errors. This includes:

- Creating regular backup schedules
- Performing full and incremental backups
- Testing recovery procedures
- Restoring databases when failures occur

A strong backup and recovery strategy ensures business continuity.

5. Performance Monitoring and Tuning

To ensure optimal performance, the DBA continuously monitors database activities. This includes:

- Analyzing query performance
- Optimizing indexes
- Managing memory allocation
- Identifying bottlenecks

NOTE

Performance tuning helps improve response time and system efficiency.

6. Data Integrity Management

The DBA ensures that data remains accurate and consistent. This involves implementing constraints such as primary keys, foreign keys, and validation rules. By enforcing integrity constraints, the DBA prevents invalid or duplicate data from entering the system.

7. Storage and Capacity Planning

The DBA manages storage allocation and plans for future growth. This includes monitoring disk usage, estimating future storage needs, and expanding database capacity when necessary.

8. Database Maintenance and Updates

Regular maintenance is essential for smooth operation. The DBA performs tasks such as:

- Applying patches and updates
- Reorganizing database structures
- Removing obsolete data
- Ensuring system compatibility

9. Disaster Recovery Planning

The DBA prepares strategies to handle major disasters such as server crashes, cyberattacks, or natural calamities. Disaster recovery planning ensures that systems can be restored quickly with minimal data loss.

10. Coordination with Developers and Users

The DBA works closely with application developers, system analysts, and end users. They provide technical support, resolve database-related issues, and guide developers in writing efficient queries.

The role of the DBA is crucial in any organization that relies on databases. The DBA ensures data security, integrity, availability, and performance. Without effective database administration, the entire information system of an organization can be at risk. Therefore, the DBA plays a vital role in maintaining the stability and success of the database environment.

4.4 RESPONSIBILITIES OF DBA

NOTE

The Database Administrator (DBA) is responsible for the overall management, control, and maintenance of a Database Management System (DBMS). The DBA ensures that the database remains secure, reliable, efficient, and available to users. The responsibilities of a DBA are extensive and require both technical expertise and administrative skills. The major responsibilities are explained below in detail.



Figure 4.2: Responsibilities of a DBA

1. Database Installation and Configuration

The DBA is responsible for installing the DBMS software and configuring it properly. This includes setting up the database server, defining system parameters, configuring memory and storage allocation, and ensuring proper connectivity. Proper configuration ensures smooth functioning of the database environment.

2. Database Design and Schema Definition

The DBA works with database designers and developers to define the database structure (schema). This includes creating tables, defining relationships, setting constraints, and establishing indexes. A well-designed schema ensures data consistency and efficient query performance.

3. User Management and Access Control

NOTE

The DBA manages user accounts and access rights. Responsibilities include:

- Creating and deleting user accounts
- Granting and revoking privileges
- Assigning user roles
- Implementing authentication mechanisms

This ensures that only authorized users can access or modify specific data, maintaining confidentiality and security.

4. Data Security Management

Data security is a critical responsibility of the DBA. The DBA protects the database from unauthorized access, cyber threats, and internal misuse. Security measures include encryption, firewall configuration, auditing user activities, and implementing security policies.

5. Backup and Recovery Management

The DBA develops and implements backup strategies to prevent data loss. This involves:

- Scheduling regular backups (full, incremental, differential)
- Monitoring backup processes
- Testing recovery procedures
- Restoring data after system failure

Effective backup and recovery plans ensure business continuity in case of hardware failure, software errors, or disasters.

6. Performance Monitoring and Tuning

The DBA continuously monitors database performance to ensure efficiency. This includes:

- Analyzing slow queries
- Optimizing indexes
- Managing memory allocation
- Identifying and resolving performance bottlenecks

Performance tuning improves response time and enhances user experience.

7. Data Integrity and Consistency Maintenance

The DBA ensures that data remains accurate and consistent. This is achieved by enforcing constraints such as primary keys, foreign keys, and validation rules. The DBA also ensures that transactions follow ACID properties (Atomicity, Consistency, Isolation, Durability).

8. Storage and Capacity Planning

The DBA monitors storage usage and plans for future expansion. This includes allocating disk space, managing tablespaces, and forecasting storage requirements to avoid system crashes due to insufficient space.

9. Database Maintenance and Updates

Regular maintenance tasks performed by the DBA include:

- Installing patches and updates
- Reorganizing database files
- Removing outdated or redundant data
- Ensuring compatibility with updated software

Maintenance ensures system stability and reliability.

10. Disaster Recovery Planning

The DBA prepares disaster recovery strategies to handle unexpected failures such as hardware crashes, cyberattacks, or natural disasters. A disaster recovery plan ensures quick restoration of services with minimal downtime.

11. Documentation and Policy Implementation

The DBA maintains documentation related to database configuration, security policies, backup schedules, and system procedures. Proper documentation ensures transparency and smooth knowledge transfer.

12. Supporting Developers and End Users

The DBA provides technical support to application developers and end users. They assist in query optimization, resolve database issues, and ensure that applications interact efficiently with the database.

The responsibilities of a DBA are broad and critical to the success of any organization that relies on data. The DBA ensures security, integrity, availability, and optimal performance of the database system. Through careful planning, monitoring, and management, the DBA maintains a stable and efficient database environment that supports organizational goals.

NOTE

4.5 DATABASE USERS

In a Database Management System (DBMS), different types of users interact with the database in various ways depending on their roles and responsibilities. These users range from technical experts who design and manage the database to ordinary users who simply retrieve or update data. Understanding the different categories of database users helps in organizing responsibilities and maintaining efficient database operations.



Figure 4.3: Database Users

The main types of database users are described below:

1. Database Administrator (DBA)

The Database Administrator (DBA) is the most important user in the database environment. The DBA is responsible for managing and controlling the entire database system. They handle installation, configuration, security management, backup and recovery, performance tuning, and user access control. The DBA ensures that the database remains secure, reliable, and efficient.

2. Database Designers (System Analysts)

Database designers are responsible for designing the overall structure of the database. They analyze the organization's requirements and create the database schema. Their tasks include:

- Identifying entities and relationships
- Defining attributes and constraints
- Designing ER diagrams
- Structuring tables and relationships

They ensure that the database design supports organizational goals and minimizes redundancy.

3. Application Programmers (Developers)

Application programmers write programs that interact with the database. They use programming languages such as Java, Python, or C#, along with SQL, to develop applications. These applications allow users to insert, update, delete, or retrieve data. Developers ensure that the database functions smoothly with software applications.

4. End Users

End users are individuals who directly use the database for their daily tasks. They are generally classified into the following types:

a) Naïve (Parametric) Users

These users perform routine tasks using predefined forms and applications. They do not need deep technical knowledge of the database.

Examples: Bank clerks, data entry operators, reservation staff.

b) Sophisticated Users

These users have knowledge of database query languages and interact with the database directly using SQL queries.

Examples: Engineers, researchers, analysts.

c) Casual Users

These users access the database occasionally to retrieve specific information. They use simple query interfaces.

d) Standalone Users

These users maintain personal databases using software like MS Access for individual purposes.

5. Specialized Users

NOTE

Specialized users develop advanced database applications for scientific research, artificial intelligence, data mining, or multimedia systems. They may use complex tools and programming techniques beyond traditional database applications. Database users play different roles depending on their responsibilities and level of expertise. From DBAs who manage the entire system to end users who perform daily transactions, each user category contributes to the effective functioning of the DBMS. Proper coordination among these users ensures data security, integrity, and efficient database operations within an organization.

**Activity 4.1:**

How to DBA perform patches installing and reorganizing database files? Check it using your system.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

4.6 END USERS

End users are the individuals who directly interact with a Database Management System (DBMS) to perform their daily tasks. They are typically not concerned with the technical structure or internal workings of the database. Instead, they focus on retrieving, updating, or managing data through user-friendly applications, forms, or query interfaces. End users play a vital role because they are the primary consumers of the data stored in the database.

End users are generally classified into different categories based on their level of expertise and frequency of database usage.

1. Naïve (Parametric) Users

Naïve users are individuals who use predefined database applications to perform routine and repetitive tasks. They do not require knowledge of database design or query languages. The system provides them with forms, menus, and graphical interfaces to perform operations.

For example, a bank clerk entering customer deposits, a railway reservation officer booking tickets, or a data entry operator updating student records are naïve users. They follow fixed procedures and use standard application programs designed by developers.

2. Casual Users

Casual users access the database occasionally when they need specific information. They may not use the system daily but require access from time to time. These users typically interact with the database using simple query tools or report-generating systems.

For example, a manager checking monthly sales reports or a teacher reviewing student performance data would be considered a casual user.

3. Sophisticated Users

Sophisticated users have a deeper understanding of the database system and may directly use query languages such as SQL to retrieve or analyze data. They are comfortable working with advanced tools and may perform complex queries.

Examples include business analysts, engineers, researchers, and financial analysts who need to extract detailed information for decision-making purposes.

4. Standalone Users

Standalone users maintain their own personal databases using desktop database software. They manage data for individual or small-scale purposes without connecting to large organizational database systems. For example, a small business owner using a personal database to manage inventory or accounts is a standalone user.

Importance of End Users

End users are important because they are the primary reason databases exist. The effectiveness of a DBMS depends largely on how efficiently end users can access and use the data. A well-designed database system ensures that end users can perform their tasks easily, accurately, and securely without needing technical expertise.

End users form the largest group of database users. Whether they are naïve, casual, sophisticated, or standalone users, they interact with the database according to their needs and skill levels. Understanding the different types of end users helps in designing better database systems that are user-friendly, secure, and efficient.

NOTE**Check Your Progress 4.1**

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: Who is a Database Administrator (DBA)? Explain any four responsibilities of a DBA.

Question 2: Explain the different types of end users in a DBMS.

4.7 SUMMARY

Unit 4 focuses on the different categories of users who interact with a Database Management System (DBMS) and highlights the important role of the Database Administrator (DBA). It explains that a database system is not only a collection of data and software but also a structured environment involving various people with specific responsibilities.

The unit describes different types of database users such as database designers, application programmers, and end users. End users are further classified into naïve users, casual users, sophisticated users, and standalone users based on their level of knowledge and interaction with the database. Each type of user performs specific tasks ranging from routine data entry to complex query analysis.

A major part of the unit emphasizes the role and responsibilities of the DBA. The DBA is responsible for database installation, configuration, security management, user authorization, backup and recovery, performance monitoring, data integrity, and disaster recovery planning. The DBA ensures that the database system operates efficiently, securely, and reliably.

Overall, this unit highlights the importance of coordination between different database users and the DBA. It helps in understanding how proper management, user roles, and administrative control contribute to the smooth functioning of a database environment

4.8 GLOSSARY

- **Application Programmer** – A person who writes programs that interact with the database using programming languages and SQL to develop applications.
- **Authorization** – The process of granting or denying access rights to users for specific database objects.
- **Backup** – A copy of database data stored separately to prevent data loss in case of system failure.
- **Casual User** – A user who accesses the database occasionally to retrieve specific information.
- **Data Integrity** – The accuracy and consistency of data stored in the database.
- **Database Administrator (DBA)** – The person responsible for managing, controlling, and maintaining the database system.
- **Database Designer** – A person who designs the database structure, schema, and relationships based on organizational requirements.
- **Disaster Recovery** – A plan and set of procedures to restore the database system after major failures or disasters.
- **End User** – An individual who directly interacts with the database to perform daily tasks.
- **Naïve (Parametric) User** – A user who performs routine tasks using predefined application programs without deep technical knowledge.
- **Performance Tuning** – The process of optimizing database operations to improve speed and efficiency.
- **Privileges** – Permissions granted to users to perform specific actions such as read, write, or modify data.
- **Recovery** – The process of restoring the database to a correct state after a failure.
- **Security** – Measures taken to protect the database from unauthorized access and threats.
- **Sophisticated User** – A knowledgeable user who interacts directly with the database using query languages like SQL.

NOTE

4.9 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: Who is a Database Administrator (DBA)? Explain any four responsibilities of a DBA.

Answer:

A Database Administrator (DBA) is a person responsible for managing, controlling, and maintaining the Database Management System (DBMS). The DBA ensures that the database operates efficiently, securely, and reliably.

Four responsibilities of a DBA are:

1. **User Management and Security:** The DBA creates user accounts, grants and revokes privileges, and ensures that only authorized users can access the database.
2. **Backup and Recovery:** The DBA schedules regular backups and restores the database in case of system failure or data loss.
3. **Performance Monitoring:** The DBA monitors database performance and optimizes queries, indexes, and memory usage to improve efficiency.
4. **Data Integrity Maintenance:** The DBA enforces constraints such as primary keys and foreign keys to maintain accuracy and consistency of data.

Q 2: Explain the different types of end users in a DBMS.

Answer:

End users are individuals who directly interact with the database to perform their daily tasks. They are classified into different types:

1. **Naïve (Parametric) Users:** These users perform routine tasks using predefined applications, such as bank clerks and data entry operators.
2. **Casual Users:** These users access the database occasionally to retrieve specific information, such as managers checking reports.
3. **Sophisticated Users:** These users have knowledge of query languages like SQL and perform complex queries, such as analysts and researchers.
4. **Standalone Users:** These users maintain personal databases using desktop database software for individual purposes.

These different types of users interact with the database according to their level of knowledge and requirements.

4.10 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13709	

4.11 REFERENCE

- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson Education.
- Korth, H. F., Silberschatz, A., & Sudarshan, S. (2019). *Database system concepts* (7th ed.). McGraw-Hill Education.

4.12 EXERCISE

PART 1

A. Multiple Choice Qs

1. Who is responsible for managing database security and user privileges?

- A) Application Programmer
- B) End User
- C) Database Administrator (DBA)
- D) System Analyst

2. Which type of user performs routine tasks using predefined application programs?

- A) Sophisticated User
- B) Naïve (Parametric) User

NOTE

C) Database Designer

D) Specialized User

3. Which of the following is NOT a responsibility of a DBA?

A) Backup and Recovery

B) Performance Tuning

C) Writing all application programs

D) User Authorization

4. A business analyst who writes SQL queries to analyze sales data is classified as a:

A) Casual User

B) Naïve User

C) Sophisticated User

D) Standalone User

5. Which responsibility ensures that the database can be restored after system failure?

A) Data Modeling

B) Backup and Recovery

C) Query Optimization

D) Data Entry

Explanation of MCQs

Q1 – C) Database Administrator (DBA)

Explanation:

The DBA is responsible for managing security policies, creating user accounts, and granting or revoking access privileges to ensure database security.

Q2 – B) Naïve (Parametric) User

Explanation:

Naïve users perform repetitive tasks such as data entry through predefined forms without needing technical knowledge of the database.

Q3 – C) Writing all application programs

Explanation:

Writing application programs is the responsibility of application programmers, not the DBA. The DBA focuses on management and administration tasks.

Q4 – C) Sophisticated User

Explanation:

Sophisticated users have knowledge of SQL and interact directly with the database to perform complex queries and analysis.

Q5 – B) Backup and Recovery

Explanation:

Backup and recovery procedures allow the DBA to restore lost data and maintain business continuity in case of hardware or software failure.

B. Short Answer Qs (100 Words)

1. Who are database users?
2. What is the role of a Database Administrator?
3. Who is a Database Administrator (DBA)?
4. Differentiate between naïve users and sophisticated users.
5. What is the importance of backup and recovery in database administration?

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the different types of database users.
2. Discuss the major responsibilities of a Database Administrator.
3. Explain the role of database designers in a database environment.
4. Explain in detail the roles and responsibilities of a Database Administrator (DBA).
5. Explain the importance of DBA in maintaining database security and performance.

PART 3

NOTE

D. Long Answer Type Questions (300 Words)

1. Discuss different categories of database users and explain their roles in a database environment.
2. Explain the responsibilities of a DBA with respect to database security, backup, recovery, and performance.
3. Discuss the role of database designers, application programmers, and end users in database development.
4. Explain why effective database administration is essential for reliable database operations.
5. Analyze the major challenges faced by database administrators in managing organizational databases.

UNIT 5

FRONT END TOOLS & INTERFACES

NOTE

UNIT STRUCTURE

- 5.1 Introduction
- 5.2 Objective
- 5.3 Front End Tools Overview
- 5.4 Database Interfaces
- 5.5 Forms and Report Tools
- 5.6 Application Connectivity Basics
- 5.7 Summary
- 5.8 Glossary
- 5.9 Answer to Check Your Progress
- 5.10 Suggested Readings – OER
- 5.11 Reference
- 5.12 Exercise

5.1 INTRODUCTION

Database systems are designed not only to store and manage information but also to provide convenient ways for users to interact with that information. Most users do not directly interact with database storage mechanisms. Instead, they use front-end applications, forms, menus, reports, graphical interfaces, and other tools to enter, retrieve, and manipulate data. These interfaces serve as a bridge between users and the underlying database system. This unit introduces the concept of front-end tools and database interfaces. It explains how front-end applications communicate with database systems and how user interfaces simplify database operations. Students will explore different types of interfaces, including command-based interfaces, graphical user interfaces, forms, menus, and report-based interfaces. The unit also discusses the relationship between front-end and back-end components. While the front end focuses primarily on user interaction and presentation, the back end generally handles data storage, processing, and database management. Effective coordination between these components is essential for developing reliable and user-friendly database applications. Understanding front-end tools is particularly important for BCA students because database applications are commonly developed as complete software systems rather than isolated databases. The unit therefore provides a foundation for understanding how database technology is integrated into practical application development and how interfaces can improve accessibility and usability.

NOTE

5.2 OBJECTIVE

After studying this unit, you will be able to:

- Define front-end tools and interfaces used in database applications.
- Explain the functions of forms, menus, reports, and graphical interfaces.
- Differentiate between front-end and back-end components of database applications.
- Analyze the role of user interfaces in improving database accessibility.
- Evaluate the effectiveness of different database interfaces for user-oriented applications.

5.3 FRONT END TOOLS AND INTERFACES

Front-end tools are software applications or interfaces that allow users to interact with a Database Management System (DBMS). They act as an intermediate layer between the user and the database (back-end). Instead of writing complex SQL queries directly, users perform actions such as filling forms, clicking buttons, or selecting options from menus. The front-end tool translates these actions into database commands that the DBMS can understand and execute.

In modern database systems, front-end tools are essential because they improve usability, efficiency, and accessibility. They help both technical and non-technical users interact with databases without needing deep knowledge of database structures or query languages.



Figure 5.1: Front End Tools and Interfaces

1. Purpose of Front-End Tools

The primary purpose of front-end tools is to provide a user-friendly interface for database operations. These tools:

- Simplify data entry and retrieval
- Reduce errors through validation checks
- Improve data visualization
- Enhance user experience
- Provide secure access to data

By hiding the complexity of SQL and database structures, front-end tools make database systems more accessible.

2. Types of Front-End Interfaces

Front-end tools can be categorized into several types:

a) Command-Line Interface (CLI)

This interface allows users to interact with the database by typing commands or SQL queries. It is mainly used by developers and database administrators. Although powerful, it requires technical knowledge.

b) Graphical User Interface (GUI)

GUI-based tools provide visual components such as windows, icons, menus, and buttons. Users interact through mouse clicks and forms rather than typing commands. Examples include database management software dashboards.

c) Form-Based Interfaces

These interfaces use predefined forms for data entry and updates. They are commonly used in banking systems, hospital management systems, and student information systems.

d) Web-Based Interfaces

Web applications act as front-end tools that connect users to databases through browsers. These are widely used in e-commerce websites, online banking, and cloud-based systems.

e) Reporting and Visualization Tools

NOTE

These tools generate reports, charts, dashboards, and graphs from database data. They help in decision-making by presenting data in an understandable format.

3. Key Features of Front-End Tools

Front-end tools usually include the following features:

- **Data Validation:** Prevents incorrect or incomplete data entry.
- **Security Controls:** Ensures only authorized users can access specific features.
- **User Authentication:** Requires login credentials.
- **Interactive Design:** Easy navigation and responsive design.
- **Error Handling:** Displays user-friendly error messages.

4. Interaction with Back-End (DBMS)

Front-end tools communicate with the database through APIs, middleware, or database connectors. When a user performs an action (e.g., submits a form), the front-end sends a request to the DBMS. The DBMS processes the request and returns the result, which is then displayed to the user.

This separation between front-end and back-end improves system flexibility and scalability.

5. Advantages of Front-End Tools

- Make databases easier to use
- Reduce dependency on technical staff
- Improve productivity
- Provide better data presentation
- Enhance system security through controlled access

Front-end tools play a crucial role in modern database systems. They bridge the gap between users and complex database operations, ensuring smooth interaction, better usability, and improved efficiency. Without front-end tools, database systems would be difficult for most users to operate effectively.

5.4 DATABASE INTERFACES

A Database Interface is the medium through which users communicate with a Database Management System (DBMS). It provides a structured way for users to interact with the database to perform operations such as inserting, updating, deleting, and retrieving data. The database interface acts as a bridge between the user and the back-

end database system, ensuring that user requests are correctly processed and results are displayed in an understandable format.

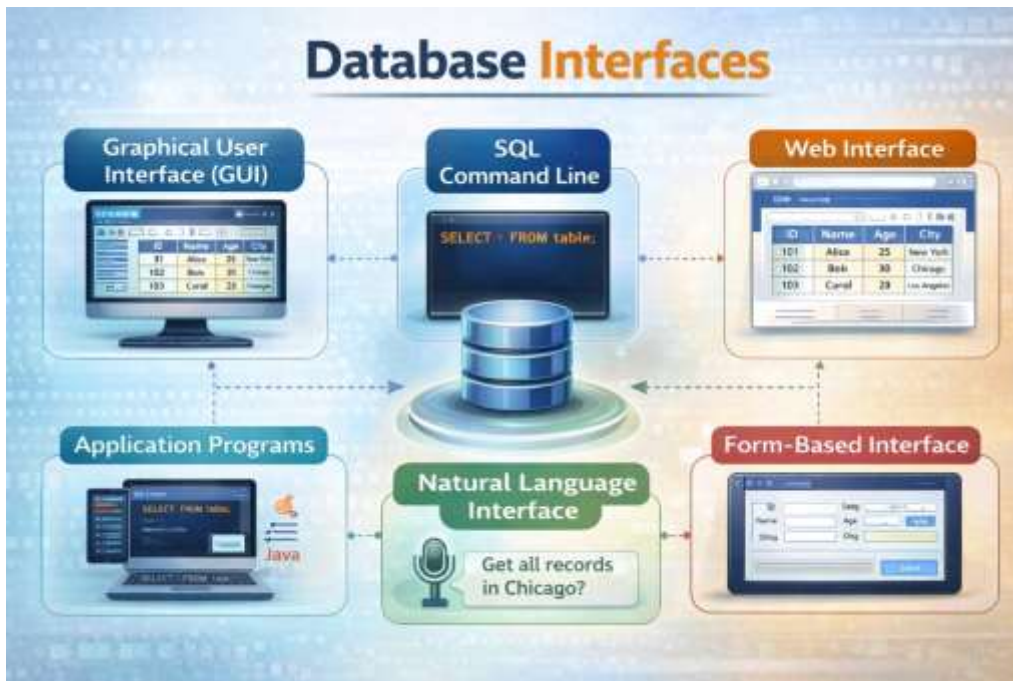


Figure 5.2: Database Interfaces

Database interfaces are designed to simplify complex database operations and make systems accessible to users with different levels of technical knowledge.

1. Purpose of Database Interface

The main purpose of a database interface is to:

- Provide a user-friendly environment for interacting with the database
- Convert user actions into database queries
- Ensure data accuracy through validation
- Maintain security by controlling user access
- Improve efficiency and productivity

By hiding the internal complexity of the DBMS, the interface allows users to focus on their tasks rather than technical details.

2. Types of Database Interfaces

Database interfaces can be classified into several types:

a) Command-Line Interface (CLI)

NOTE

In a command-line interface, users type SQL commands directly to interact with the database. This interface is mostly used by database administrators and developers who are familiar with query languages. It is powerful and flexible but requires technical knowledge.

b) Graphical User Interface (GUI)

A graphical user interface provides visual components such as windows, icons, buttons, menus, and dialog boxes. Users interact through mouse clicks instead of typing commands. GUI interfaces are easy to use and are suitable for beginners and non-technical users.

c) Form-Based Interface

Form-based interfaces allow users to enter and modify data through structured forms. Each field in the form corresponds to a database attribute. These interfaces are commonly used in banking systems, hospital systems, school management systems, and online registration portals.

d) Menu-Based Interface

In a menu-based interface, users select options from predefined menus. This type of interface is simple and user-friendly, often used in ATM machines and online ticket booking systems.

e) Web-Based Interface

Web-based interfaces allow users to access databases through web browsers. These are widely used in e-commerce websites, online banking, social media platforms, and cloud applications. They connect the front-end application to the database using web technologies and APIs.

f) Natural Language Interface

Some advanced systems allow users to interact with databases using natural language queries (e.g., "Show all students with marks above 80"). The system translates the request into SQL commands.

3. Components of a Database Interface

A database interface typically includes:

- **Input Module:** Accepts user commands or data
- **Validation Module:** Checks for errors and ensures correct data format

- **Query Processor:** Converts user requests into SQL queries
- **Output Module:** Displays results in tables, charts, or reports
- **Security Layer:** Controls authentication and authorization

4. Importance of Database Interfaces

Database interfaces are important because they:

- Improve user accessibility
- Reduce the need for technical knowledge
- Enhance data security
- Minimize data entry errors
- Increase system efficiency

A well-designed interface improves user satisfaction and ensures smooth database operations.

A database interface is an essential part of a database system. It provides an effective communication channel between users and the DBMS. By offering different types of interfaces such as CLI, GUI, form-based, and web-based systems, database interfaces make data management simple, secure, and efficient for various categories of users.

5.5 FORMS AND REPORT TOOLS

Forms and report tools are important front-end components of a Database Management System (DBMS). They help users interact with the database easily and present data in a meaningful format. While forms are mainly used for entering and updating data, report tools are used for displaying and analyzing data in structured formats.



Figure 5.3: Forms and Report Tools

NOTE

1. Forms in Database Systems

Meaning of Forms

A form is a user-friendly interface used to collect, enter, modify, or delete data in a database. Forms provide a structured layout where each field corresponds to a specific attribute in a database table. Instead of writing SQL commands, users can simply fill in text boxes, select options, or click buttons.

Purpose of Forms

- Simplify data entry
- Reduce user errors
- Improve data consistency
- Provide validation checks
- Enhance user experience

Forms are commonly used in banking systems, hospital management systems, school management systems, online registration systems, and e-commerce websites.

Components of a Form

A typical database form includes:

- **Text boxes** – For entering data (e.g., name, address)
- **Drop-down lists** – For selecting predefined values
- **Radio buttons / Checkboxes** – For selecting options
- **Command buttons** – For actions like Save, Update, Delete
- **Labels** – To describe fields
- **Validation controls** – To ensure correct data entry

Advantages of Forms

- Easy to use for non-technical users
- Reduces need for SQL knowledge
- Improves data accuracy
- Controls access to specific fields
- Enhances data security

2. Report Tools in Database Systems

Meaning of Reports

Reports are formatted outputs generated from database data. They present information in an organized, structured, and readable format.

Reports help users analyze and interpret data for decision-making purposes.

Purpose of Reports

- Summarize large amounts of data
- Present information clearly
- Support business decisions
- Provide statistical analysis
- Generate printable documents

Reports are widely used in businesses for sales summaries, financial statements, attendance records, inventory status, and performance analysis.

Types of Reports

- **Tabular Reports** – Display data in rows and columns
- **Summary Reports** – Show totals, averages, or counts
- **Graphical Reports** – Use charts and graphs
- **Grouped Reports** – Organize data by categories
- **Detailed Reports** – Provide complete records

Features of Report Tools

- Data filtering and sorting
- Grouping and aggregation functions
- Chart and graph generation
- Exporting to PDF, Excel, or print format
- Scheduled report generation

Table 5.1: Difference Between Forms and Reports

Forms	Reports
Used for data entry and modification	Used for data presentation and analysis
Interactive interface	Output-oriented format
Focus on input operations	Focus on output and summary
Used by clerks and operators	Used by managers and decision-makers

Forms and report tools are essential front-end tools in a DBMS. Forms make data entry simple, secure, and accurate, while report tools transform stored data into meaningful information for analysis and

NOTE

decision-making. Together, they improve efficiency, usability, and productivity in database systems.



Activity 5.1:

Create a form to enter a college data and add some radio buttons and text box in forms.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

5.6 APPLICATION CONNECTIVITY BASICS

Application connectivity refers to the process of connecting an application (such as a web app, desktop app, or mobile app) to a Database Management System (DBMS) so that the application can store, retrieve, update, and manage data. It forms the foundation of modern database-driven systems where front-end applications interact with back-end databases.

Without proper connectivity, an application cannot communicate with the database. Therefore, understanding application connectivity is essential for developing reliable, secure, and efficient database systems.

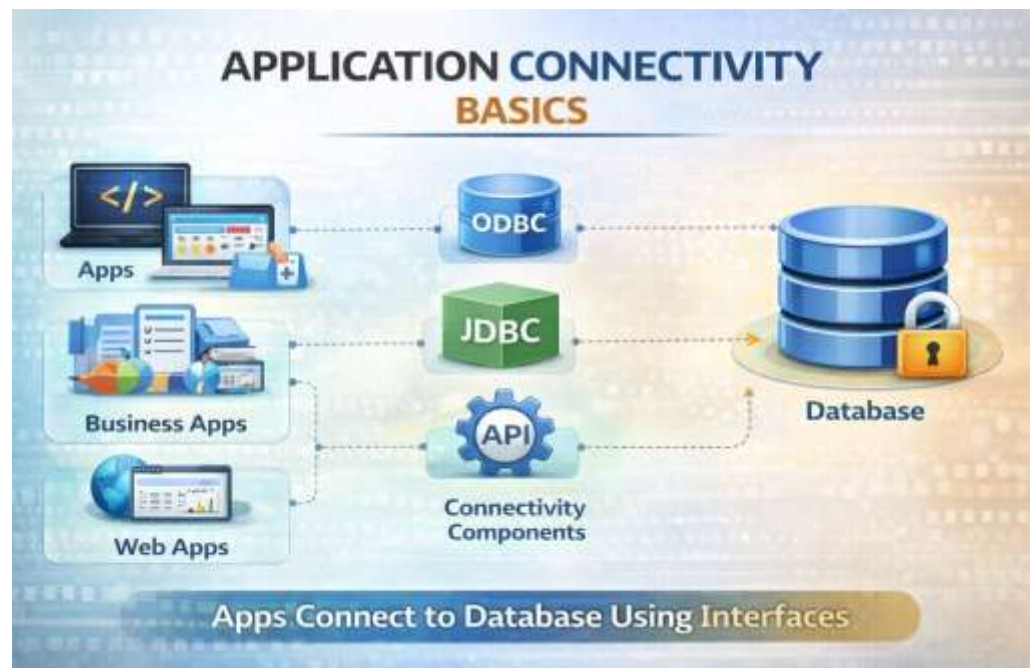


Figure 5.4: Application Connectivity Basics

1. Need for Application Connectivity

Most users do not directly interact with a database using SQL commands. Instead, they use applications like banking apps, e-commerce websites, hospital systems, or school management systems. These applications require connectivity to:

- Insert new data (e.g., customer registration)
- Retrieve stored data (e.g., account balance)
- Update existing records (e.g., change address)
- Delete unwanted data

Application connectivity ensures smooth communication between the user interface and the database server.

2. Basic Components of Application Connectivity

Application connectivity involves several important components:

a) Client Application

The software or interface used by the user (e.g., web browser, desktop application, mobile app).

b) Application Server

Acts as an intermediate layer that processes business logic and communicates with the database.

c) Database Server

Stores and manages data using a DBMS.

d) Database Driver / Connector

A software component that enables communication between the application and the database. It translates application requests into database-specific commands.

3. Connectivity Technologies

Several technologies are used to establish connectivity between applications and databases:

a) ODBC (Open Database Connectivity)

NOTE

ODBC is a standard API that allows applications to connect to different database systems using a common interface. It is widely used in Windows environments.

b) JDBC (Java Database Connectivity)

JDBC is an API used in Java applications to connect and execute queries on databases. It allows Java programs to interact with various DBMS platforms.

c) ADO.NET

Used in .NET applications to connect to databases. It supports data access and manipulation in Microsoft-based environments.

d) APIs and RESTful Services

Modern web applications use APIs to connect with databases through web services. These APIs handle communication between front-end and back-end systems.

4. Basic Steps in Database Connectivity

The general process of application connectivity involves:

1. **Load the database driver**
2. **Establish a connection** using connection string (database name, username, password, server address)
3. **Create and execute queries** (SQL commands)
4. **Process results** returned from the database
5. **Close the connection** after operations are complete

5. Connection String

A connection string contains necessary information to connect to a database, such as:

- Server name
- Database name
- Username
- Password
- Port number

It ensures secure and accurate connection between the application and the database.

6. Security in Application Connectivity

Security is critical in application connectivity. Important measures include:

- Authentication (username and password)
- Encryption of data transmission
- Use of secure APIs
- Protection against SQL injection
- Role-based access control

Proper security prevents unauthorized access and protects sensitive data.

7. Importance of Application Connectivity

Application connectivity is important because it:

- Enables real-time data processing
- Supports multi-user access
- Improves system scalability
- Ensures efficient data management
- Allows integration between different systems

Application connectivity is the backbone of modern database applications. It enables seamless communication between front-end applications and back-end databases using drivers, APIs, and connectivity technologies such as ODBC and JDBC. Proper understanding of connectivity basics ensures efficient, secure, and reliable database operations in real-world systems.

Check Your Progress 5.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What are Front-End Tools? Explain their importance in database applications.

Question 2: Explain Application Connectivity and its basic steps.

NOTE

5.7 SUMMARY

Unit 5 focuses on Front-End Tools and Interfaces used in database applications. It explains how users interact with databases through graphical interfaces instead of directly writing SQL queries. The unit highlights tools and technologies that make database systems user-friendly, efficient, and secure.

The unit begins with an overview of front-end tools, which are software applications used to design forms, reports, and user interfaces. These tools help in collecting input from users, displaying data in structured formats, and improving user experience. It discusses different front-end development environments and their importance in database-driven applications.

Next, the unit explains the Database Interface, which acts as a bridge between the user interface and the database. It describes how applications communicate with databases using queries and commands. The concept of structured interaction between front-end and back-end systems is emphasized.

The unit also covers Forms and Report Tools in detail.

- **Forms** are used for data entry, modification, and deletion. They provide a structured way for users to interact with the database.
- **Reports** are used to present stored data in an organized and formatted manner, often for analysis and decision-making purposes.

Another important topic in this unit is Application Connectivity Basics. It explains how applications connect to databases using technologies such as ODBC, JDBC, and other APIs. The steps involved in establishing database connections, executing queries, and ensuring secure communication are also discussed.

Overall, Unit 5 provides knowledge about designing user-friendly interfaces, connecting applications with databases, and managing data interaction efficiently. It emphasizes the importance of front-end tools in making database systems practical and accessible for real-world use.

5.8 GLOSSARY

1. **Front-End Tools:** Software used to design user interfaces that interact with databases.

2. User Interface (UI): The visual part of an application through which users interact with the system.
3. Database Interface: The connection layer that allows communication between the application and the database.
4. Form: A graphical interface used to enter, update, and delete data in a database.
5. Report: A formatted presentation of database data used for analysis and decision-making.
6. Application Connectivity: The process of connecting an application to a database system.
7. ODBC (Open Database Connectivity): A standard API that allows applications to connect to different database systems.
8. JDBC (Java Database Connectivity): An API used by Java applications to connect and execute queries on databases.
9. ADO.NET: A data access technology used in .NET applications for database connectivity.
10. Connection String: A string containing database connection details such as server name, database name, username, and password.
11. Query: A request sent to a database to retrieve or modify data.
12. SQL (Structured Query Language): A language used to manage and manipulate databases.
13. Database Driver: Software that enables communication between an application and a database.
14. API (Application Programming Interface): A set of rules and protocols that allows different software applications to communicate.
15. Client-Server Architecture: A computing model where the client requests services and the server provides them.

5.9 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What are Front-End Tools? Explain their importance in database applications.

NOTE

Answer:

Front-end tools are software applications used to design and develop user interfaces that interact with databases. They allow users to input, modify, and view data without directly writing SQL queries.

Importance of Front-End Tools:

1. Provide user-friendly graphical interfaces.
2. Simplify data entry and retrieval.
3. Reduce errors through validation controls.
4. Improve productivity and efficiency.
5. Enable better presentation of data using forms and reports.

Thus, front-end tools make database systems accessible and easier to use for non-technical users.

Q 2: Explain Application Connectivity and its basic steps.

Answer:

Application connectivity is the process of connecting an application (such as a web or desktop application) to a database system so that data can be stored, retrieved, and managed.

Basic Steps in Application Connectivity:

1. Load the database driver.
2. Establish a connection using a connection string.
3. Create and execute SQL queries.
4. Process the results returned by the database.
5. Close the connection after completing operations.

Application connectivity ensures smooth communication between the front-end interface and the database, enabling efficient data management.

5.10 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13710	

5.11 REFERENCE

- Connolly, T., & Begg, C. (2015). *Database systems: A practical approach to design, implementation, and management* (6th ed.). Pearson Education.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson Education.
- Date, C. J. (2019). *An introduction to database systems* (8th ed.). Pearson Education.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill Education.
- Groff, J. R., Oppel, A. J., & Hoberman, S. (2015). *SQL: The complete reference* (3rd ed.). McGraw-Hill Education.

5.12 EXERCISE

PART 1

A. Multiple Choice Qs

1. Which of the following is mainly used to design user-friendly data entry screens?

- A) SQL
- B) Forms
- C) Reports
- D) Database Driver

2. What is the primary purpose of a database interface?

- A) To store data permanently
- B) To design hardware components
- C) To connect the front-end application with the database
- D) To replace the database

3. Which technology is specifically used for database connectivity in Java applications?

- A) ODBC
- B) JDBC
- C) ADO.NET

NOTE

D) HTML

4. What does CRUD stand for in database operations?

- A) Create, Run, Update, Delete
- B) Create, Read, Update, Delete
- C) Copy, Read, Upload, Download
- D) Connect, Retrieve, Upload, Delete

5. Which of the following contains details required to connect to a database?

- A) Report
- B) Query
- C) Connection String
- D) Form Layout

Explanation of MCQs

Q1 – B) Forms

Explanation:

Forms are used to create user-friendly interfaces for entering, updating, and deleting data in a database. They help users interact with the system without writing SQL commands.

Q2 – C) To connect the front-end application with the database

Explanation:

A database interface acts as a bridge between the application and the database, enabling communication and data exchange.

Q3 – B) JDBC

Explanation:

JDBC (Java Database Connectivity) is an API that allows Java programs to connect and execute SQL queries on databases.

Q4 – B) Create, Read, Update, Delete

Explanation:

CRUD represents the four basic database operations used to manage data: creating new records, reading existing data, updating records, and deleting records.

Q5 – C) Connection String

Explanation:

A connection string contains necessary information such as server name, database name, username, and password to establish a connection between an application and a database.

B. Short Answer Qs (100 Words)

1. What are front-end tools in database applications?
2. What is a graphical user interface in a database system?
3. Explain the purpose of forms in database applications.
4. What is a database query interface?
5. Differentiate between front-end and back-end components.

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the role of front-end tools in database applications.
2. Describe different types of database user interfaces.
3. Explain how forms and menus improve interaction with databases.
4. Discuss the importance of graphical user interfaces in database systems.
5. Explain the relationship between front-end applications and database servers.

PART 3**D. Long Answer Type Questions (300 Words)**

1. Explain different front-end tools and interfaces used for developing database applications.
2. Discuss the role of forms, reports, menus, and graphical interfaces in database applications.
3. Explain how front-end and back-end components work together in a database application.
4. Discuss the advantages and limitations of graphical database interfaces.
5. Explain the importance of user-friendly interfaces in designing effective database applications.



DR. C.V. RAMAN UNIVERSITY

Chhattisgarh, Bilaspur AN A/SECT GROUP UNIVERSITY

Approved by: PCI | AICTE | NCTE | BCI | Member of: AU | Joint Committee: UGC | DB | AICTE | Recognized by: BGC | A NAAC Accredited University



DR. C. V. RAMAN UNIVERSITY (C.G.)

SLM FEEDBACK FORM

PROGRAMME : _____

COURSE : _____

BLOCK : _____

UNIT : _____

Dear Student,

While studying the units of this block, you may have found certain portions of the text difficult to comprehend. We wish to know your difficulties and suggestions, in order to improve the course. Therefore, we request you to fill up and send us the following questionnaire, which pertains to this block. If you find the space provided insufficient, kindly use a separate sheet.

Please mail to aisectcontent@aisect.org/ for online submission please scan QR code:

Questionnaire

1. Name : _____

2. Enrolment No. :

--	--	--	--	--	--	--	--	--	--

3. How many hours did you need for studying the units? :

--

4. Please give your reactions to the following items based on your reading of the block:

Items	Excellent	Very Good	Good	Poor	Give specific Examples (If poor)
Presentation Quality					
Language and Style					
Illustrations					
Used (diagrams, tables, etc.)					
Conceptual Clarity					
Check Your Progress Questions					
Suggested Readings - OER					

5. Any other comments:

BLOCK III	DATA MODELLING AND ER DESIGN	109-168
Unit 6	Basics of Data Modelling	111
Unit 7	Relationship Model	129
Unit 8	ER Diagrams and Database Design	147



Block III: Data Modelling and ER Design focuses on the conceptual design of a database system. It explains how real-world objects and their relationships are identified, structured, and represented before actual database implementation. Data modelling is the process of organizing data elements and defining how they relate to one another. The most important tool used in this block is the Entity-Relationship (ER) Model, which visually represents the structure of a database using entities (objects), attributes (properties), and relationships (connections between entities). Entities are shown as rectangles, attributes as ovals, and relationships as diamonds in an ER diagram. This block also covers concepts such as primary keys, foreign keys, cardinality (one-to-one, one-to-many, many-to-many), and participation constraints. By designing an ER diagram, developers can clearly understand data requirements, reduce redundancy, and ensure accuracy before converting the model into relational tables.

UNIT 6

BASICS OF DATA MODELLING

NOTE

UNIT STRUCTURE

- 6.1 Introduction
- 6.2 Objective
- 6.3 Entity Types and Entity Sets
- 6.4 Attributes (Simple, Composite, Multivalued, Derived)
- 6.5 Keys and Key Attributes
- 6.6 Summary
- 6.7 Glossary
- 6.8 Answer to Check Your Progress
- 6.9 Suggested Readings – OER
- 6.10 Reference
- 6.11 Exercise

6.1 INTRODUCTION

Data modelling is an important stage in database design because it provides a structured representation of the data requirements of an organization or application. Before creating database tables, designers need to identify what information must be stored, how different pieces of information are related, and what rules should govern the data. Data modelling provides a systematic method for representing these requirements. This unit introduces the fundamental concepts of data modelling and explains its importance in database development. Students will learn about essential modelling elements such as entities, attributes, relationships, and domains. An entity represents a distinguishable object or concept, while attributes describe its characteristics. Relationships represent associations between entities and help describe how different types of information are connected. The unit also introduces different approaches to data modelling and explains the distinction between conceptual and logical models. Conceptual modelling focuses on representing business requirements at a high level, while logical modelling provides a more structured representation suitable for database implementation. A clear understanding of data modelling helps reduce design errors, unnecessary data duplication, and inconsistencies. It also provides a foundation for creating Entity-Relationship diagrams and relational database structures. Through this unit, learners develop the ability to translate real-world information requirements into organized database models.

NOTE**6.2 OBJECTIVES**

After studying this unit, you will be able to:

1. Define data modelling and its fundamental concepts.
2. Explain entities, attributes, relationships, and domains in data modelling.
3. Differentiate between major types of data models.
4. Analyze the role of data modelling in database design.
5. Evaluate a data model based on its ability to represent real-world requirements effectively.

6.3 ENTITY TYPES AND ENTITY SETS

In data modelling, entities are the fundamental building blocks used to represent real-world objects or concepts that have a distinct existence and about which data is stored in a database. Understanding entity types and entity sets is essential for designing a clear and organized database using the Entity-Relationship (ER) model.



Figure 6.1: Entity Types & Entity Sets

1. Entity Types

An entity type is a category or class of objects in the real world that share common properties. It defines the structure of similar entities by specifying the attributes that describe them. In other words, an entity type is a blueprint or template for creating individual entities in a database.

Characteristics of Entity Types:

- Represent real-world objects or concepts (e.g., Student, Employee, Product).
- Defined by a set of attributes that describe their properties.
- Usually denoted by a rectangle in an ER diagram with the entity type name inside.

Example:

- Entity Type: Student
- Attributes: Student_ID, Name, Age, Course

This entity type represents the class of all students in an educational system.

2. Entity Sets

An entity set is the collection of all entities of a particular entity type that exist in the database at a given point in time. While an entity type is like a blueprint, an entity set is like the actual table of instances.

Characteristics of Entity Sets:

- Contains individual entities (instances) of the entity type.
- Each entity in the set has unique identifiers (primary key).
- Can be empty or populated, depending on the database content.

Example:

- Entity Type: Student
- Entity Set: {S1, S2, S3, S4}
 - S1: {Student_ID: 101, Name: Ankur, Age: 22, Course: B.Sc}
 - S2: {Student_ID: 102, Name: Priya, Age: 21, Course: BCA}

Here, the entity set contains actual student records stored in the database.

3. Types of Entities

Entities can be further classified into:

1. Strong Entity:

- Can exist independently in the database.
- Has a primary key that uniquely identifies each entity.

NOTE

- Example: Student, Employee

2. Weak Entity:

- Cannot exist without a strong entity.
- Does not have a complete primary key on its own; uses a partial key combined with the strong entity's key.
- Example: Dependent of an Employee

4. Representation in ER Diagram

- **Entity Type:** Represented by a rectangle labeled with the entity type name.
- **Attributes:** Represented by ovals connected to the entity rectangle.
- **Primary Key Attribute:** Underlined to indicate uniqueness.
- **Entity Set Instances:** Usually shown as examples under the entity type.

5. Importance

- Helps in structuring the database conceptually before implementation.
- Defines what data to store and how to organize it.
- Facilitates the creation of relationships with other entity types.
- Reduces redundancy and ensures data integrity.

Understanding entity types and entity sets is critical for building a clear ER model. Entity types provide a blueprint, while entity sets represent actual data instances. Proper identification and classification of entities form the foundation of effective database design.

6.4 ATTRIBUTES (SIMPLE, COMPOSITE, MULTIVALUED, DERIVED)

In database design, attributes describe the properties or characteristics of an entity. Each entity type has attributes that define what information is stored about it. Understanding different types of attributes is essential for accurately modeling a database in the Entity-Relationship (ER) model. Attributes help in uniquely identifying entities, storing data, and establishing relationships.



Figure 6.2: Database Attributes

1. Simple (Atomic) Attributes

A simple attribute is indivisible; it cannot be broken down into smaller components. These attributes store a single value for each entity instance.

Characteristics:

- Cannot be decomposed further.
- Stores only one piece of information.
- Typically used as basic properties of an entity.

Example:

- Entity Type: Student
- Simple Attributes: Student_ID, Age, Gender

Here, Student_ID is a simple attribute representing a unique identifier.

ER Diagram Representation: An oval connected to the entity rectangle.

2. Composite Attributes

NOTE

A composite attribute is an attribute that can be divided into smaller subparts, each having independent meaning. Composite attributes help in organizing complex data more effectively.

Characteristics:

- Can be split into sub-attributes.
- Each sub-attribute can store meaningful information.
- Often used for addresses, names, or contacts.

Example:

- Entity Type: Employee
- Composite Attribute:

Name = {First_Name, Middle_Name, Last_Name}

- Composite Attribute: Address = {Street, City, State, Zip_Code}

ER Diagram Representation:

An oval connected to the entity rectangle with smaller ovals for each sub-attribute.

3. Multivalued Attributes

A multivalued attribute can have multiple values for a single entity instance. These attributes often represent lists or repeated information.

Characteristics:

- One entity can have multiple values for this attribute.
- Requires special handling in database implementation (often using separate tables).

Example:

- Entity Type: Student
- Multivalued Attribute: Phone_Numbers = {+91-9876543210, +91-9123456789}
- Entity Type: Employee
- Multivalued Attribute: Skills = {Java, Python, SQL}

ER Diagram Representation: Double oval connected to the entity rectangle.

4. Derived Attributes

NOTE

A derived attribute is calculated from other attributes rather than stored directly in the database. These attributes are useful for dynamically generating values when needed.

Characteristics:

- Not physically stored in the database.
- Computed using other attribute values.
- Often used for age, total salary, or duration.

Example:

- Entity Type: Employee
- Attributes: Date_of_Birth
- Derived Attribute: Age = Current_Date - Date_of_Birth
- Entity Type: Order
- Attributes: Unit_Price, Quantity
- Derived Attribute: Total_Price = Unit_Price × Quantity

ER Diagram Representation: Dashed oval connected to the entity rectangle.

Table 6.1: Summary Table of Attribute Types

Attribute Type	Definition	ER Symbol	Example
Simple (Atomic)	Cannot be divided further	Oval	Student_ID, Age
Composite	Can be broken into sub-attributes	Oval with branches	Name → First_Name, Last_Name
Multivalued	Can have multiple values for a single entity	Double Oval	Phone_Numbers, Skills
Derived	Calculated from other attributes	Dashed Oval	Age (from DOB), Total_Price

Importance of Attributes in Data Modelling

1. Provide detailed information about entities.
2. Help in uniquely identifying entity instances (via primary keys).
3. Facilitate data retrieval, filtering, and sorting.
4. Ensure accurate representation of real-world information.
5. Support relationships and constraints in the ER model.

NOTE

6.5 KEYS AND KEY ATTRIBUTES

In database design, keys play a crucial role in uniquely identifying entities within a database and establishing relationships between different entities. Key attributes are those attributes of an entity that are used to form keys. Proper identification of keys ensures data integrity, reduces redundancy, and enables efficient retrieval of data.

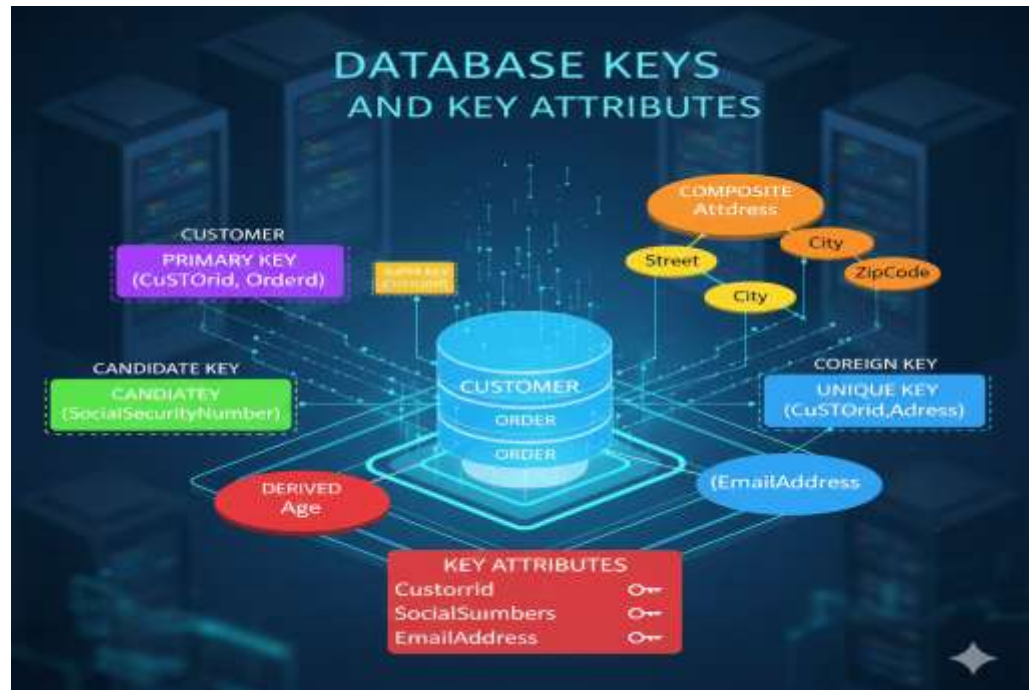


Figure 6.3: Database Keys and Key Attributes

1. Key Attributes

A key attribute is an attribute (or a combination of attributes) that helps in uniquely identifying an entity in an entity set. Every entity set must have a key attribute to distinguish each entity instance from others.

Example:

- Entity Type: Student
- Attributes: Student_ID, Name, DOB
- **Key Attribute:** Student_ID uniquely identifies each student, even if multiple students have the same name.

ER Diagram Representation: Key attributes are underlined in the ER diagram.

2. Types of Keys

Keys are classified based on their usage and properties in a database:

a) Primary Key (PK)

- A primary key is a key attribute (or a combination of attributes) that uniquely identifies each record in an entity set.
- Must have unique values and cannot be null.
- Only one primary key is allowed per entity.

Example:

- Entity: Student
- Primary Key: Student_ID

b) Candidate Key

- A candidate key is an attribute (or a combination of attributes) that could serve as a primary key.
- An entity can have multiple candidate keys, but only one is chosen as the primary key.
- Candidate keys must be unique and minimal.

Example:

- Entity: Employee
- Attributes: Employee_ID, Email_ID
- Both Employee_ID and Email_ID are candidate keys.
- Employee_ID might be selected as the primary key.

c) Alternate Key

- An alternate key is a candidate key that is not chosen as the primary key.
- Still has the uniqueness property.
- Used to enforce constraints and for referencing purposes.

Example:

- From the Employee example, if Employee_ID is chosen as PK, then Email_ID becomes an alternate key.

d) Composite Key (or Concatenated Key)

- A composite key consists of two or more attributes combined to uniquely identify an entity.
- Used when no single attribute is sufficient for uniqueness.

Example:

- Entity: Course_Enrollment

NOTE

- Attributes: Student_ID, Course_ID
- Composite Key: {Student_ID, Course_ID} uniquely identifies each enrollment record.

e) Super Key

- A super key is a set of attributes that can uniquely identify an entity, but it may contain unnecessary attributes.
- Every primary key is a super key, but not every super key is a primary key.

Example:

- Entity: Student
- Super Key: {Student_ID, Name}, {Student_ID}, {Student_ID, DOB}

f) Foreign Key (FK)

- A foreign key is an attribute in one table that refers to the primary key in another table.
- Establishes a relationship between two entities.
- Helps maintain referential integrity.

Example:

- Entity: Enrollment
- Attributes: Student_ID (FK referencing Student), Course_ID (FK referencing Course)

3. Properties of Keys

1. **Uniqueness:** Each key value must uniquely identify an entity.
2. **Minimality:** Only necessary attributes should be included (no extra attributes).
3. **Non-null:** Primary key values cannot be null.
4. **Stability:** Key values should not change frequently.
5. **Referential Integrity:** Foreign keys must match primary key values in referenced entities.

4. Importance of Keys

- Ensure uniqueness of records.
- Help in establishing relationships between entities.
- Support data retrieval, updates, and deletions efficiently.
- Maintain data integrity and consistency.

- Reduce redundancy and prevent duplicate entries.

5. ER Diagram Representation of Keys

- **Primary Key:** Underlined attribute
- **Composite Key:** Multiple attributes underlined together
- **Foreign Key:** Dashed underline or indicated with (FK)

Keys and key attributes are fundamental to database design. They ensure that each entity instance can be uniquely identified and that relationships between entities are properly maintained. Understanding primary keys, candidate keys, alternate keys, composite keys, super keys, and foreign keys is crucial for designing efficient, accurate, and reliable databases.



Activity 6.1:

Draw an ER diagram for University and define the keys and their attributes also.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

Check Your Progress 6.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is the difference between an entity type and an entity set?

Question 2: Explain the difference between a primary key and a foreign key.

6.6 SUMMARY

Unit 6 focuses on the fundamentals of data modelling, which is a crucial step in designing efficient and well-structured databases. Data

NOTE

modelling helps in understanding the data requirements of a system, identifying entities, their attributes, and the relationships among them before the actual database implementation.

The unit begins with an introduction to entity types and entity sets, which represent real-world objects and collections of those objects, respectively. It emphasizes the importance of attributes in defining the properties of entities, covering different types such as simple, composite, multivalued, and derived attributes.

Another core topic of this unit is keys and key attributes, which are essential for uniquely identifying entities and maintaining relationships between entity sets. The unit explains various types of keys, including primary key, candidate key, alternate key, composite key, super key, and foreign key, along with their significance in ensuring data integrity and minimizing redundancy.

Overall, Unit 6 provides a strong conceptual foundation for creating Entity-Relationship (ER) models, which are used to visually represent database structures and serve as the blueprint for relational database design. This unit is essential for understanding how to model real-world data accurately and efficiently for implementation in database systems.

6.7 GLOSSARY

- 1. Data Modelling:** The process of analyzing, organizing, and representing data to define its structure, relationships, and constraints before implementing a database.
- 2. Entity:** A real-world object or concept about which data is stored in a database.
- 3. Entity Type:** A category or class of entities with similar properties; serves as a blueprint for creating entities.
- 4. Entity Set:** A collection of all entities of a particular entity type present in the database at a given time.
- 5. Attribute:** A property or characteristic of an entity that describes it.
- 6. Simple (Atomic) Attribute:** An indivisible attribute that cannot be broken down into smaller parts.
- 7. Composite Attribute:** An attribute that can be divided into smaller subparts, each with independent meaning.

8. Multivalued Attribute: An attribute that can have multiple values for a single entity instance.

9. Derived Attribute: An attribute whose value is calculated from other attributes rather than stored directly.

10. Key Attribute: An attribute that is used to uniquely identify an entity in an entity set.

11. Primary Key (PK): A key attribute chosen to uniquely identify each entity in a set; cannot be null.

12. Candidate Key: An attribute (or set of attributes) that could serve as a primary key; must be unique.

13. Alternate Key: A candidate key that is not selected as the primary key.

14. Composite Key: A key composed of two or more attributes that together uniquely identify an entity.

15. Super Key: A set of one or more attributes that can uniquely identify an entity, possibly containing extra attributes.

6.8 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What is the difference between an entity type and an entity set?

Answer:

- Entity Type: A blueprint or category that defines a class of real-world objects sharing common properties. Example: Student entity type represents all students.
- Entity Set: The collection of actual entities (instances) of an entity type that exist in the database at a given time. Example: {S1, S2, S3} representing individual student records.

Q 2: Explain the difference between a primary key and a foreign key.

Answer:

NOTE

- Primary Key (PK): A key attribute that uniquely identifies each entity in an entity set. It cannot have null values. Example: Student_ID in the Student entity.
- Foreign Key (FK): An attribute in one entity that refers to the primary key of another entity, establishing a relationship between the two. Example: Student_ID in the Enrollment entity referencing Student_ID in the Student entity.

6.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13711	

6.10 REFERENCE

- Connolly, T., & Begg, C. (2015). *Database systems: A practical approach to design, implementation, and management* (6th ed.). Pearson Education.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson Education.
- Date, C. J. (2019). *An introduction to database systems* (8th ed.). Pearson Education.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill Education.

6.11 EXERCISE**PART 1****A. Multiple Choice Qs**

1. Which of the following represents a real-world object in a database?

- A) Attribute
- B) Entity
- C) Relationship

D) Key

2. What is a composite attribute?

- A) An attribute that can have multiple values
- B) An attribute whose value is derived from another attribute
- C) An attribute that can be divided into sub-parts
- D) An attribute that uniquely identifies an entity

3. Which key uniquely identifies each entity in an entity set?

- A) Foreign Key
- B) Primary Key
- C) Alternate Key
- D) Composite Key

4. A foreign key is used to:

- A) Uniquely identify an entity in its own table
- B) Establish a relationship between two entity sets
- C) Store multiple values for an attribute
- D) Derive an attribute from another attribute

5. Which of the following is a multivalued attribute?

- A) Student_ID
- B) Date_of_Birth
- C) Phone_Numbers
- D) Name

Explanation of MCQs

Q1 – B) Entity

NOTE

Explanation:

An entity is a real-world object or concept about which data is stored. Examples include Student, Employee, or Product.

Q2 – C) An attribute that can be divided into sub-parts

Explanation:

Composite attributes can be split into smaller, meaningful components. Example: Name → {First_Name, Last_Name}.

Q3 – B) Primary Key

Explanation:

A primary key is chosen to uniquely identify each entity instance in a set. It cannot be null and ensures uniqueness.

Q4 – B) Establish a relationship between two entity sets

Explanation:

A foreign key is an attribute in one table that references the primary key of another table, maintaining referential integrity.

Q5 – C) Phone_Numbers

Explanation:

A multivalued attribute can have multiple values for a single entity. For example, a student can have multiple phone numbers.

B. Short Answer Qs (100 Words)

1. What is data modelling?
2. Why is data modelling important in database design?
3. What are entities and attributes?
4. What is a relationship in data modelling?
5. Explain the difference between conceptual and logical data models.

PART 2

C. Long Answer Type Questions (200 Words)

1. Explain the basic concepts of data modelling.
2. Discuss entities, attributes, and relationships with suitable examples.
3. Explain the different types of data models.
4. Describe the importance of conceptual data modelling in database design.

5. Explain how data modelling helps in organizing complex data.

NOTE**PART 3****D. Long Answer Type Questions (300 Words)**

1. Explain the concept, objectives, and importance of data modelling in database development.
2. Discuss entities, attributes, relationships, and constraints as fundamental elements of data modelling.
3. Explain different types of data models and compare their characteristics.
4. Describe the process of developing a conceptual data model for a database application.
5. Analyze the role of data modelling in reducing redundancy and improving database design.

NOTE

UNIT 7

RELATIONSHIP MODEL

NOTE

UNIT STRUCTURE

- 7.1 Introduction
- 7.2 Objective
- 7.3 Relationship Types
- 7.4 Degree of Relationships
- 7.5 Cardinality and Participation Constraints
- 7.6 Summary
- 7.7 Glossary
- 7.8 Answer to Check Your Progress
- 7.9 Suggested Readings – OER
- 7.10 Reference
- 7.11 Exercise

7.1 INTRODUCTION

Relationships are a fundamental element of database modelling because information stored in a database is rarely independent. Different entities are usually associated with one another in meaningful ways. For example, students may enroll in courses, customers may place orders, and employees may work in departments. Representing these associations accurately is essential for developing an effective database. This unit introduces the concept of relationships and explains how associations between entities are represented in database models. A relationship describes how one entity is connected to another entity or to itself. The unit focuses on major relationship types, including one-to-one, one-to-many, and many-to-many relationships. Cardinality is an important concept in relationship modelling because it specifies the number of instances of one entity that can be associated with instances of another entity. Understanding cardinality helps database designers correctly represent real-world requirements. The unit also introduces the importance of identifying appropriate relationships during the database design process. Students will learn how relationships can be analyzed using practical examples and how different relationship types influence database structure. Proper relationship modelling helps avoid inconsistencies and supports effective data retrieval and management. This unit therefore provides an important foundation for studying Entity-Relationship diagrams, relational database design, and the conversion of conceptual models into relational structures.

NOTE**7.2 OBJECTIVES**

After studying this unit, you will be able to:

1. Define relationships and relationship types in database systems.
2. Explain one-to-one, one-to-many, and many-to-many relationships.
3. Differentiate between various relationship cardinalities.
4. Analyze relationships among entities in a database design.
5. Evaluate the suitability of different relationship types for real-world database applications.

7.3 RELATIONSHIP TYPES

In data modelling, a relationship defines how two or more entities are associated with each other. The type of relationship determines how instances of one entity are connected to instances of another entity. Understanding relationship types is critical for designing an accurate and efficient database.

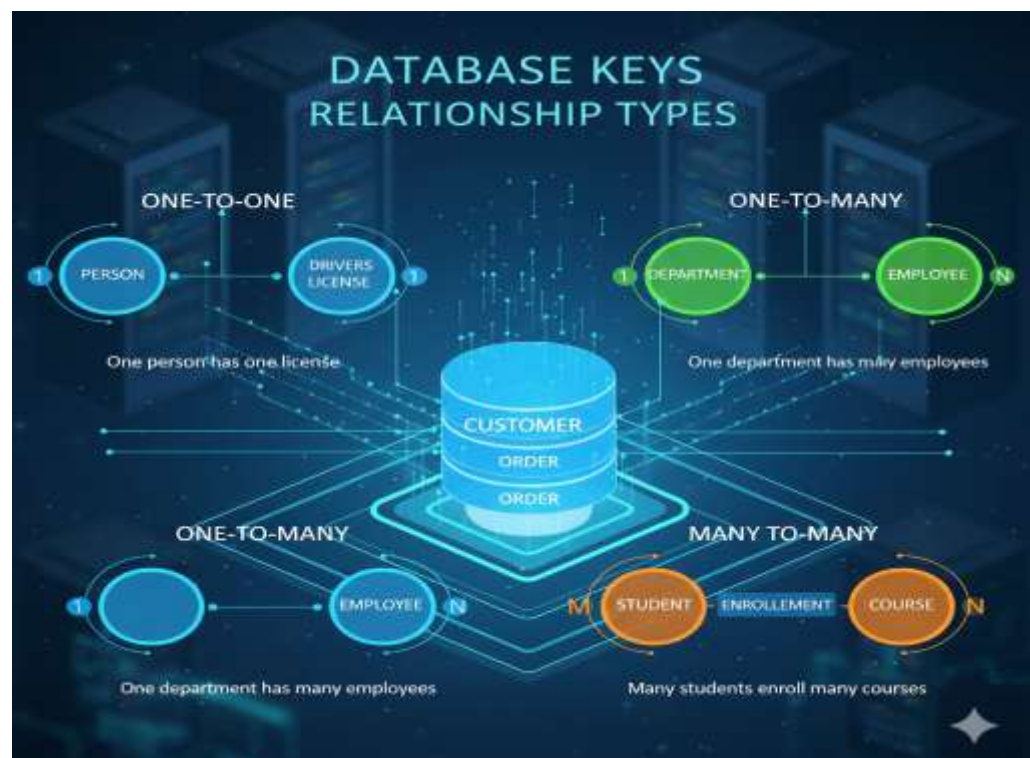


Figure 7.1: Relationship Types

1. One-to-One (1:1) Relationship

A one-to-one relationship exists when a single instance of an entity A is associated with at most one instance of entity B, and vice versa.

Characteristics:

- Each entity participates in exactly one relationship instance.
- Rare in practical databases but useful when splitting data for security or modular design.

Example:

- Entity A: Person
- Entity B: Passport
- Each person has exactly one passport, and each passport belongs to exactly one person.

2. One-to-Many (1:N) Relationship

A one-to-many relationship exists when a single instance of entity A can be associated with multiple instances of entity B, but each instance of entity B is associated with only one instance of entity A.

Characteristics:

- Most common type of relationship in databases.
- Useful to model hierarchical or parent-child relationships.

Example:

- Entity A: Department
- Entity B: Employee
- One department can have many employees, but each employee belongs to one department.

3. Many-to-Many (M:N) Relationship

A many-to-many relationship exists when multiple instances of entity A can be associated with multiple instances of entity B, and vice versa.

Characteristics:

- Requires an associative (junction) table or relationship entity in relational databases.
- Often includes attributes of the relationship itself, like enrollment date, quantity, etc.

Example:

- Entity A: Student
- Entity B: Course

NOTE

- A student can enroll in many courses, and a course can have many students.

4. Self-Referencing (Recursive) Relationship

A recursive relationship exists when an entity has a relationship with itself.

Characteristics:

- Useful to represent hierarchical or peer relationships.

Example:

- Entity: Employee
- Relationship: Manages
- An employee can manage other employees in the same entity set.

5. Ternary and Higher-Order Relationships

- Ternary Relationship: Involves three entities simultaneously.
- Higher-Order Relationship: Involves more than three entities.

Example of Ternary Relationship:

- Entities: Supplier, Part, Project
- Relationship: Supplies
- A supplier supplies a particular part for a specific project.

6. Relationship Attributes

Sometimes a relationship itself has attributes. These attributes describe the association between entities rather than the entities themselves.

Example:

In a Student-Course M:N relationship, Grade or Enrollment_Date is a relationship attribute.

Importance of Relationship Types

1. Ensures accurate representation of real-world associations.
2. Guides ER diagram design and table creation in relational databases.
3. Helps enforce data integrity via foreign keys.
4. Reduces redundancy and ensures proper normalization.
5. Determines how queries and joins are structured in the relational database.

7.4 DEGREE OF RELATIONSHIPS

In database design, the degree of a relationship refers to the number of entity types involved in a relationship. It helps to classify relationships based on how many entities participate in them. Understanding the degree is important for correctly modeling real-world interactions and for converting ER diagrams into relational schemas.

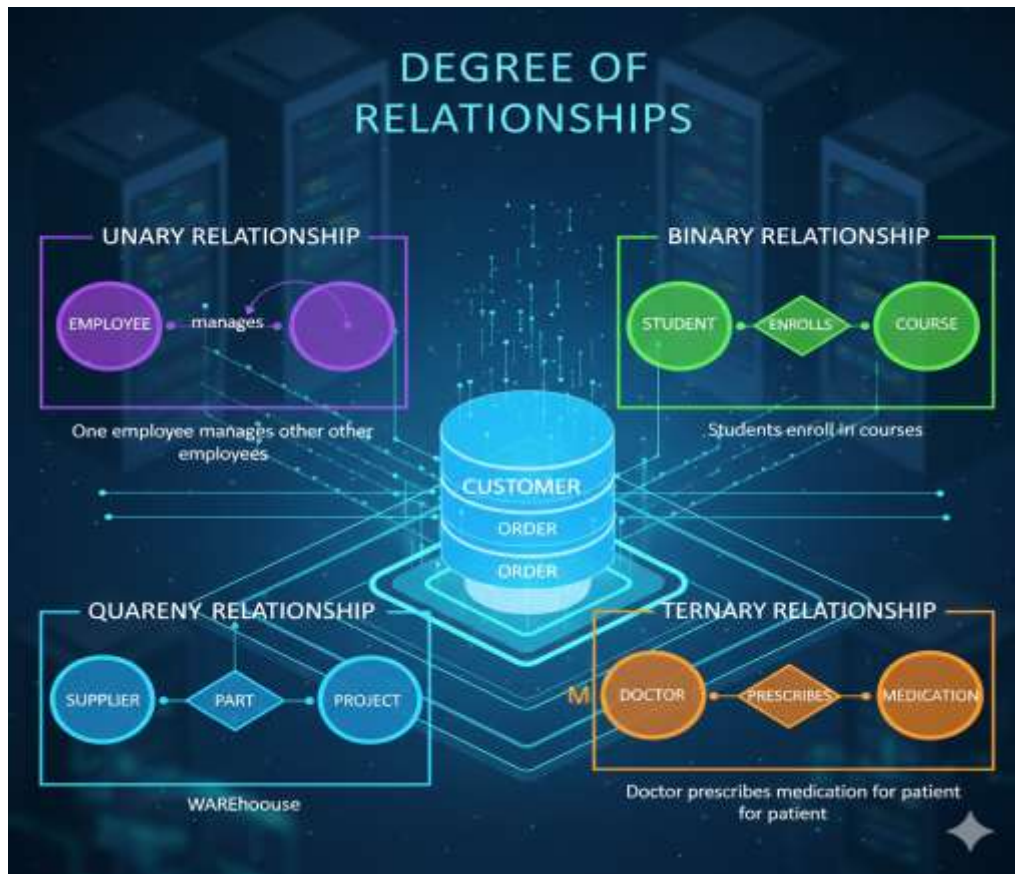


Figure 7.2: Degree of Relationships

1. Unary (Degree 1) Relationship

A unary relationship involves only one entity type. It is also called a recursive relationship, because the entity participates in a relationship with itself.

Characteristics:

- One entity is related to itself.
- Often used to model hierarchical or peer relationships.

Example:

NOTE

- Entity: Employee
- Relationship: Manages
- An employee can manage other employees.

Use Case: Organizational hierarchy, manager-subordinate relationships.

2. Binary (Degree 2) Relationship

A binary relationship involves two entity types, making it the most common type of relationship in databases.

Characteristics:

- Connects two entities.
- Can be 1:1, 1:N, or M:N.

Example:

- Entity A: Student
- Entity B: Course
- Relationship: Enrolls
- A student can enroll in many courses, and each course can have many students (M:N relationship).

3. Ternary (Degree 3) Relationship

A ternary relationship involves three entity types simultaneously.

Characteristics:

- Represents interactions that cannot be broken down into multiple binary relationships without losing information.
- Sometimes includes relationship attributes.

Example:

- Entities: Supplier, Part, Project
- Relationship: Supplies
- A supplier supplies a specific part for a specific project.

Use Case: Supply chain management, where a combination of three entities is required to fully describe the relationship.

4. Quaternary and Higher-Degree Relationships

- Quaternary Relationship: Involves four entity types.

NOTE

- Higher-Degree Relationships: Involve more than four entity types.
- Rare in practice because most real-world scenarios can be decomposed into ternary or binary relationships.
- Complexity increases with higher-degree relationships.

Example:

- Entities: Supplier, Part, Project, Warehouse
- Relationship: Delivers
- A supplier delivers a part for a project to a specific warehouse.

Table 7.1: Summary of Relationship Degrees

Degree	Number of Entities	Also Called	Example
Unary	1	Recursive	Employee-Manages-Employee
Binary	2	Most common	Student-Enrolls-Course
Ternary	3	-	Supplier-Supplies-Part-Project
Quaternary	4	-	Supplier-Delivers-Part-Project-Warehouse

Importance of Understanding Relationship Degrees

1. Accurate Modelling: Correctly represents real-world interactions.
2. ER Diagram Clarity: Helps in proper visualization of relationships.
3. Database Implementation: Guides how to map relationships into relational tables.
4. Data Integrity: Ensures that all constraints and participation rules are respected.
5. Normalization: Helps prevent redundancy and maintains consistency.

NOTE**Activity 7.1:**

Describe the different types of relationships with suitable examples.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

7.5 CARDINALITY AND PARTICIPATION CONSTRAINTS

In database design, understanding cardinality and participation constraints is crucial for accurately modeling the relationships between entities. These constraints define how many instances of one entity can or must be associated with instances of another entity, ensuring the database reflects real-world rules and business logic.

1. Cardinality Constraints

Cardinality specifies the number of instances of one entity that can be associated with instances of another entity in a relationship. Cardinality is usually represented as 1:1, 1:N, or M:N.

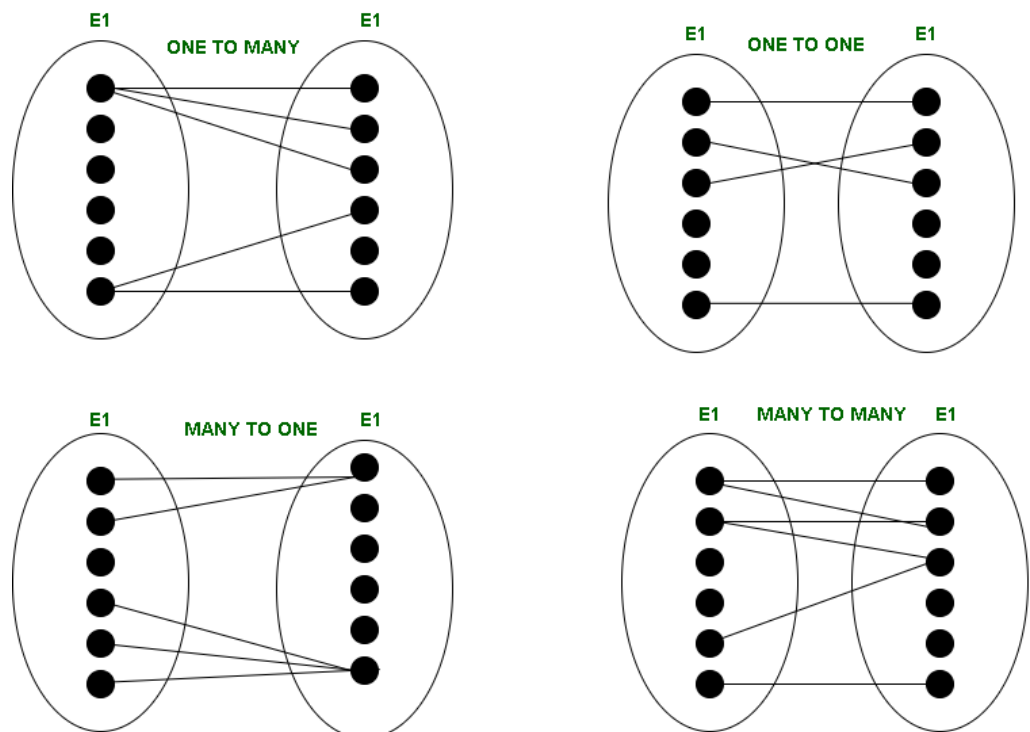


Figure 7.2: Cardinality Constraints

Types of Cardinality Constraints**a) One-to-One (1:1)**

- Each instance of Entity A is associated with at most one instance of Entity B, and vice versa.
- Rare in practice but useful for splitting data for security or modularity.

Example:

- Entities: Person and Passport
- Each person has exactly one passport, and each passport belongs to exactly one person.

b) One-to-Many (1:N)

- One instance of Entity A can be associated with multiple instances of Entity B.
- Each instance of Entity B is associated with only one instance of Entity A.

Example:

- Entities: Department and Employee
- One department can have many employees, but each employee belongs to one department.

c) Many-to-Many (M:N)

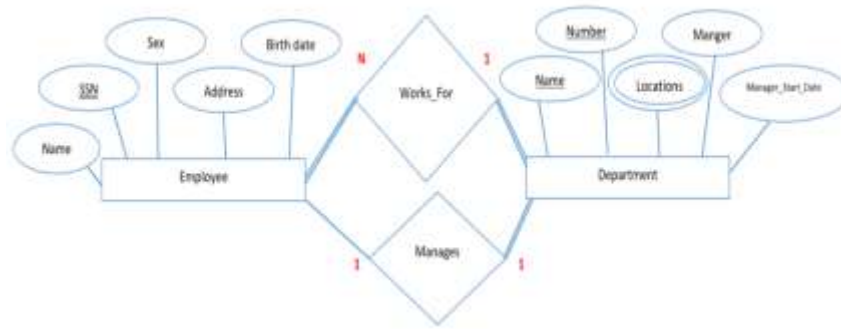
- Multiple instances of Entity A can be associated with multiple instances of Entity B, and vice versa.
- Usually implemented with a junction table in relational databases.

Example:

- Entities: Student and Course
- A student can enroll in many courses, and a course can have many students.

2. Participation Constraints

Participation constraints specify whether all or only some instances of an entity participate in a relationship.

NOTE**Figure 7.3: Participation Constraints****Types of Participation****a) Total Participation**

- Every instance of the entity must participate in the relationship.
- Denoted by a double line in ER diagrams.

Example:

- Entity: Employee
- Relationship: Works_In (Department)
- Every employee must belong to a department.

b) Partial Participation

- Some instances of the entity may or may not participate in the relationship.
- Denoted by a single line in ER diagrams.

Example:

- Entity: Customer
- Relationship: Purchases (Order)
- Some customers may not have placed any orders yet.

3. Combining Cardinality and Participation

- Cardinality defines how many entities can participate.
- Participation defines whether participation is mandatory or optional.

Table 7.2: Cardinality and Participation

Relationship	Entity	Cardinality	Participation
Works_In	Employee	1:N	Total
Purchases	Customer	0:N	Partial

Relationship	Entity	Cardinality	Participation
Passport_Owned	Person	1:1	Total

4. Importance

1. **Ensures data integrity:** Maintains valid associations between entities.
2. **Accurate real-world modeling:** Reflects business rules and constraints.
3. **Guides relational table design:** Determines primary keys, foreign keys, and constraints.
4. **Prevents anomalies:** Reduces data redundancy and inconsistency.
5. **Supports normalization:** Helps structure the database efficiently.

Table 5.3: ER Diagram Representation Summary

Symbol	Meaning
Single line	Partial participation
Double line	Total participation
1:N, 1:1, M:N	Cardinality ratio between entities

Check Your Progress 7.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: Define cardinality and explain the different types of cardinality in a database relationship?

Question 2: What are participation constraints? Explain total and partial participation with examples.

NOTE

7.6 SUMMARY

Unit 7, *Relationship Model*, focuses on how entities in a database interact with each other. It emphasizes the importance of accurately modeling relationships to represent real-world associations between entities in a system. The unit builds upon entity and attribute concepts and introduces methods for representing connections using ER diagrams.

Key topics covered in this unit include:

1. **Relationship Types** – Explains how entities can be associated as one-to-one (1:1), one-to-many (1:N), many-to-many (M:N), recursive (self-referencing), ternary, and higher-order relationships. Each type reflects different real-world scenarios.
2. **Degree of Relationships** – Describes the number of entity types involved in a relationship, such as unary (1), binary (2), ternary (3), and higher-degree relationships. Understanding degree helps in correctly modeling complex associations.
3. **Cardinality Constraints** – Specifies how many instances of one entity can be associated with instances of another entity (1:1, 1:N, M:N). Cardinality ensures relationships are represented accurately in the database.
4. **Participation Constraints** – Indicates whether all or some instances of an entity participate in a relationship, classified as total (mandatory) or partial (optional) participation.
5. **Relationship Attributes** – Some relationships themselves have attributes, such as Grade in a Student-Course enrollment scenario.

Overall, Unit 7 equips learners to:

- Identify and classify relationships between entities.
- Apply cardinality and participation rules to maintain data integrity.
- Represent relationships clearly in ER diagrams.
- Prepare for translating conceptual designs into relational database schemas.

Mastering these concepts is critical for creating accurate, normalized, and efficient database designs that mirror real-world systems.

7.8 GLOSSARY

- Relationship: A connection or association between two or more entities in a database.
- Relationship Type: A category or set of relationships that share the same attributes between entity sets.
- Unary Relationship (Recursive Relationship): A relationship where an entity is related to itself.
- Binary Relationship: A relationship involving two entity types. Most common in database design.
- Ternary Relationship: A relationship involving three entity types simultaneously.
- Higher-Degree Relationship: A relationship involving more than three entities (quaternary or higher).
- Cardinality: Specifies the number of instances of one entity that can or must be associated with instances of another entity (e.g., 1:1, 1:N, M:N).
- Participation Constraint: Specifies whether all instances of an entity participate in a relationship (total participation) or some instances may not participate (partial participation).
- Total Participation: When every entity instance must participate in a relationship. Represented by a double line in ER diagrams.
- Partial Participation: When some entity instances may or may not participate in a relationship. Represented by a single line in ER diagrams.
- Relationship Attribute: An attribute that belongs to a relationship rather than an entity, used to store information about the association (e.g., Grade in a Student-Course enrollment).
- One-to-One (1:1) Relationship: A relationship in which each instance of Entity A is associated with at most one instance of Entity B, and vice versa.
- One-to-Many (1:N) Relationship: A relationship where one instance of Entity A can be associated with multiple instances of Entity B, but each instance of B is associated with only one instance of A.
- Many-to-Many (M:N) Relationship: A relationship where multiple instances of Entity A can be associated with multiple instances of Entity B. Often implemented via a junction table.
- ER Diagram (Entity-Relationship Diagram): A visual representation of entities, attributes, and relationships, showing cardinality and participation constraints.

7.9 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: Define cardinality and explain the different types of cardinality in a database relationship.

NOTE**Answer:**

Cardinality specifies the number of instances of one entity that can or must be associated with instances of another entity in a relationship.

Types of Cardinality:

1. One-to-One (1:1): Each instance of Entity A is associated with at most one instance of Entity B, and vice versa.
 - Example: Person and Passport.
2. One-to-Many (1:N): One instance of Entity A can be associated with multiple instances of Entity B, but each instance of B is associated with only one instance of A.
 - Example: Department and Employee.
3. Many-to-Many (M:N): Multiple instances of Entity A can be associated with multiple instances of Entity B.
 - Example: Student and Course enrollment.

Q 2: What are participation constraints? Explain total and partial participation with examples.

Answer:

Participation constraints indicate whether all or some instances of an entity participate in a relationship.

1. Total Participation:
 - Every instance of the entity must participate in the relationship.
 - Denoted by a double line in ER diagrams.
 - Example: Every Employee must belong to a Department.
2. Partial Participation:
 - Some instances of the entity may not participate in the relationship.
 - Denoted by a single line in ER diagrams.
 - Example: Some Customers may not have placed any Orders.

7.10 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13712	

7.11 REFERENCES

- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson Education.
- Date, C. J. (2019). *An introduction to database systems* (8th ed.). Pearson Education.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill Education.
- Harrington, J. L. (2016). *Relational database design and implementation* (4th ed.). Morgan Kaufmann.

7.12 EXERCISE

PART 1

A. Multiple Choice Qs

1. Which of the following represents a relationship between two entity types in a database?

- A) Attribute
- B) Key
- C) Relationship
- D) Table

2. What type of relationship exists when one instance of an entity A is associated with many instances of entity B, but each instance of B is associated with only one instance of A?

- A) One-to-One (1:1)
- B) One-to-Many (1:N)
- C) Many-to-Many (M:N)
- D) Recursive

3. A ternary relationship involves how many entity types?

- A) 1
- B) 2
- C) 3
- D) 4

4. What does total participation in a relationship mean?

- A) All entity instances must participate in the relationship
- B) Some entity instances may participate in the relationship

NOTE

- C) The relationship has multiple attributes
- D) The relationship is recursive

5. Which of the following is a many-to-many relationship example?

- A) Each person has exactly one passport
- B) Each employee belongs to one department
- C) Students enroll in courses
- D) An employee manages another employee

Explanation of MCQs

Q1 – C) Relationship

Explanation:

A relationship defines an association between two or more entity types in a database. Attributes describe entities, and keys identify entities.

Q2 – B) One-to-Many (1:N)

Explanation:

In a 1:N relationship, one entity can be linked to multiple instances of another entity, but the reverse is restricted to one.

Example: Department → Employee.

Q3 – C) 3

Explanation:

A ternary relationship connects three entities simultaneously, such as Supplier, Part, and Project.

Q4 – A) All entity instances must participate in the relationship

Explanation:

Total participation means that every instance of the entity participates in the relationship. In ER diagrams, it is represented with a double line.

Q5 – C) Students enroll in courses

Explanation:

In a many-to-many (M:N) relationship, multiple instances of Entity A can be associated with multiple instances of Entity B. Each student can enroll in multiple courses, and each course can have multiple students.

B. Short Answer Qs (100 Words)

1. What is a relationship model?
2. What is a relationship in a database?
3. Explain one-to-one relationships.
4. What is a one-to-many relationship?
5. Explain many-to-many relationships.

PART 2

C. Long Answer Type Questions (200 Words)

1. Explain different types of relationships in database systems.
2. Describe one-to-one, one-to-many, and many-to-many relationships with examples.
3. Explain the importance of relationships in database design.
4. Discuss relationship cardinality and its significance.
5. Explain how relationships are represented in a database model.

PART 3

D. Long Answer Type Questions (300 Words)

1. Explain the concept of relationships in database modelling and discuss different relationship types.
2. Discuss one-to-one, one-to-many, and many-to-many relationships using suitable real-world examples.
3. Explain cardinality and participation constraints in relationship modelling.
4. Describe the process of identifying and defining relationships among entities during database design.
5. Analyze the importance of correctly defining relationships for maintaining data consistency and integrity.

NOTE

UNIT 8

ER DIAGRAMS AND DATABASE DESIGN

UNIT STRUCTURE

- 8.1 Introduction
- 8.2 Objective
- 8.3 ER Diagram Notation
- 8.4 Strong and Weak Entities
- 8.5 Database Design using ER Diagrams
- 8.6 Summary
- 8.7 Glossary
- 8.8 Answer to Check Your Progress
- 8.9 Suggested Readings – OER
- 8.10 Reference
- 8.11 Exercise

8.1 INTRODUCTION

Entity-Relationship modelling is a widely used approach for representing the structure and requirements of a database before its implementation. It provides a visual and conceptual method for identifying important objects, their characteristics, and the relationships among them. Entity-Relationship diagrams, commonly called ER diagrams, make complex database requirements easier to analyze and communicate. This unit introduces the fundamental concepts of ER diagrams and their role in database design. Students will learn about entities, entity sets, attributes, relationships, cardinality, and participation constraints. Entities represent objects or concepts of interest, while attributes describe their properties. Relationships represent associations between entities and help establish how information is connected. The unit also focuses on the process of developing ER diagrams from real-world requirements. Students will learn how to identify entities and attributes, determine relationships, and represent appropriate cardinalities. These concepts are important for transforming organizational requirements into a structured database design. ER diagrams also provide a useful communication tool between database designers, developers, and users because they represent database requirements visually. A carefully prepared ER model can help identify missing information, unnecessary relationships, and potential design problems before implementation begins. Therefore, this unit establishes an important foundation for

NOTE

developing well-structured relational databases and understanding the transition from conceptual design to database implementation.

8.2 OBJECTIVES

After studying this unit, you will be able to:

- Define Entity-Relationship modelling and its basic components.
- Explain entities, attributes, relationships, and constraints in ER diagrams.
- Construct ER diagrams for simple database applications.
- Analyze database requirements using Entity-Relationship modelling techniques.
- Evaluate an ER-based database design for accuracy, completeness, and consistency.

8.3 ER DIAGRAM NOTATION

An Entity-Relationship (ER) diagram is a visual representation of a database's entities, attributes, and relationships. ER diagram notations are standardized symbols used to represent these elements clearly and consistently. Understanding these notations is crucial for designing and interpreting ER diagrams accurately.

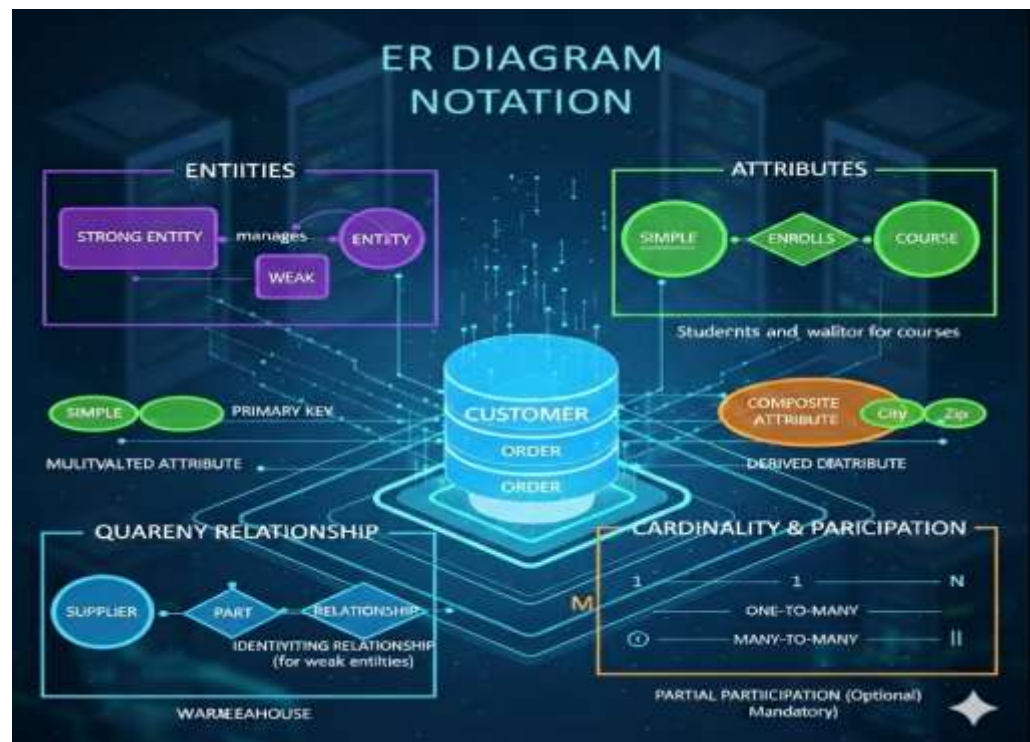


Figure 8.1: ER Diagram Notation

1. Entity Notation

- **Definition:** An entity represents a real-world object or concept about which data is stored.
- **Notation:** Rectangle
- **Types of Entities:**
 - **Strong Entity:** Exists independently and has a primary key.
 - Example: Student, Employee.
 - Notation: Single rectangle.
 - **Weak Entity:** Depends on a strong entity and does not have a complete primary key.
 - Example: Dependent of an employee.
 - Notation: Double rectangle.

Example:

[Student] (Strong Entity)

[[Dependent]] (Weak Entity)

2. Attribute Notation

- **Definition:** Attributes are properties or characteristics of an entity.
- **Notation:** Oval/Circle connected to the entity or relationship.

Types of Attributes:

1. **Simple Attribute:** Cannot be divided further.
 - Example: Student_ID, First_Name.
2. **Composite Attribute:** Can be divided into sub-parts.
 - Example: Full_Name → {First_Name, Last_Name}
3. **Multivalued Attribute:** Can have multiple values for a single entity.
 - Example: Phone_Numbers
 - Notation: Double oval
4. **Derived Attribute:** Can be derived from other attributes.
 - Example: Age derived from Date_of_Birth
 - Notation: Dashed oval

NOTE

5. **Key Attribute:** Uniquely identifies an entity.

- Example: Student_ID
- Notation: Underline the attribute name

Example Representation:

OVAL(Simple) OVAL(Derived-dashed)

3. Relationship Notation

- **Definition:** Represents association between two or more entities.
- **Notation:** Diamond shape connecting the participating entities.
- **Relationship Types:**
 1. **Unary (Recursive)** – Entity relates to itself.
 2. **Binary** – Two entities involved.
 3. **Ternary** – Three entities involved.

Example:

[Student] ---- Enrolls ---- [Course]

4. Cardinality Notation

- Specifies the number of instances of one entity associated with instances of another entity.

Table 8.1: Cardinality

Notation	Meaning
1	One instance
N or M	Many instances
0..1	Zero or one instance
0..N / 1..N	Zero or many / One or many

Placement: Near the entity on the line connecting it to the relationship.

Example:

Department 1 ----- N Employee

(One department has many employees)

5. Participation Constraint Notation

- Defines whether all or some instances of an entity participate in a relationship.
- **Total Participation:** Represented by a double line connecting entity to relationship.
- **Partial Participation:** Represented by a single line.

Example:

[[Employee]] == Works_In == Department

(Double line indicates every employee must belong to a department)

6. Weak Entity Relationship (Identifying Relationship)

- Weak entities rely on a strong entity for identification.
- **Notation:**
 - Weak entity: Double rectangle
 - Relationship: Double diamond

Example:

[[Dependent]] == DEPENDS_ON == [Employee]

7. Generalization and Specialization

- **Generalization:** Combines multiple entities into a higher-level entity.
 - Notation: Triangle pointing up, connected to parent entities.
- **Specialization:** Divides an entity into sub-entities based on distinguishing characteristics.
 - Notation: Triangle pointing down, connected to child entities.

8. Aggregation

- Treats a relationship as a higher-level entity when it participates in another relationship.
- Notation: Rectangle enclosing a diamond

Example:

[Project] ----- [[Assigned]] ----- [Employee] (Aggregation if Assigned participates in another relationship)

Importance of ER Diagram Notations

NOTE

1. Provides a clear visual representation of entities and relationships.
2. Helps in identifying data requirements and constraints.
3. Serves as a blueprint for converting ER diagrams into relational schemas.
4. Ensures consistency, data integrity, and minimal redundancy in the database.

8.4 STRONG AND WEAK ENTITIES

In database design, entities represent real-world objects or concepts about which data is stored. Based on their existence and identification requirements, entities are classified as strong entities or weak entities. Understanding the distinction is crucial for accurate ER modeling and relational database design.

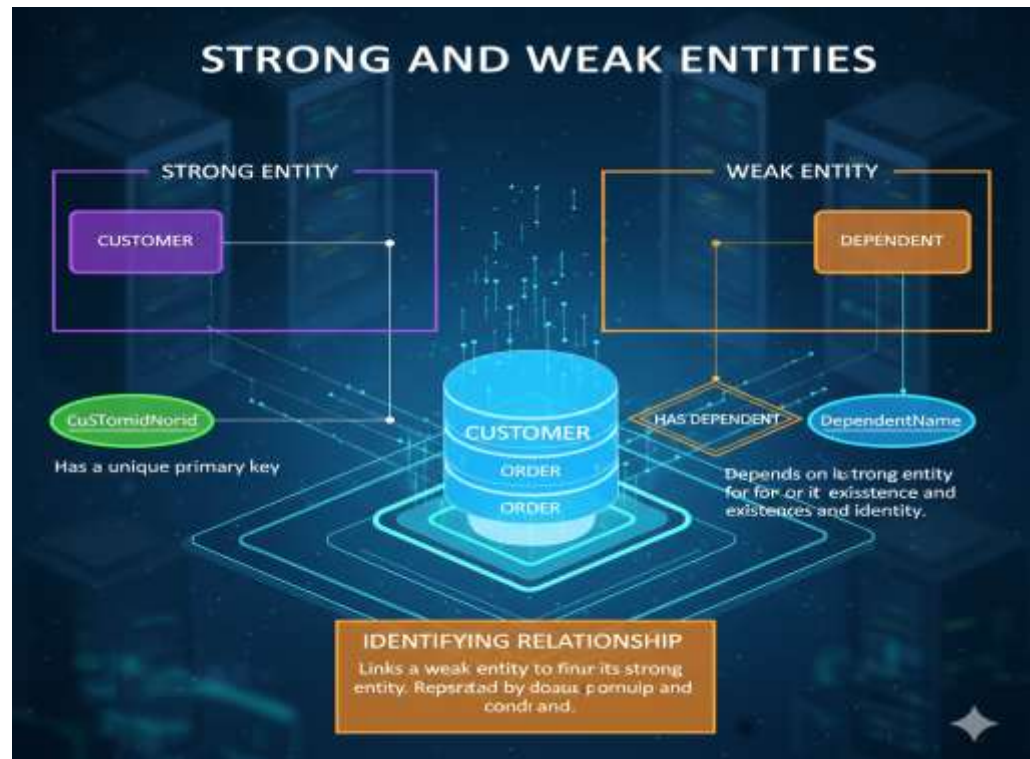


Figure 8.2: Strong and Weak Entities

1. Strong Entity

Definition:

A strong entity is an entity that exists independently and can be uniquely identified by its own primary key. It does not rely on any other entity for its identification.

Characteristics:

1. Has a primary key that uniquely identifies each instance.
2. Can exist without being associated with other entities.
3. Represented by a single rectangle in ER diagrams.

Example:

- Entities: Student, Employee, Department
- Student_ID uniquely identifies each student.
- Employee_ID uniquely identifies each employee.

ER Diagram Representation:

[Student] → Strong Entity

Importance:

- Forms the backbone of a database.
- Can participate in relationships but does not depend on others for identification.

2. Weak Entity**Definition:**

A weak entity cannot be uniquely identified by its own attributes alone and depends on a strong entity for its existence. It is linked to a strong entity through an identifying relationship.

Characteristics:

1. Does not have a sufficient primary key; uses a partial key (also called discriminator) along with the strong entity's primary key.
2. Existence depends on a strong entity.
3. Represented by a double rectangle in ER diagrams.
4. The relationship connecting it to the strong entity is called an identifying relationship, represented by a double diamond.
5. Typically, participation in the identifying relationship is total, meaning every weak entity instance must be associated with a strong entity instance.

Example:

- Strong Entity: Employee
- Weak Entity: Dependent (child or spouse of an employee)
- Dependent_Name alone cannot uniquely identify a dependent; it must be combined with Employee_ID.

NOTE**ER Diagram Representation:**

[[Dependent]] == DEPENDS_ON == [Employee]

- Double rectangle → Dependent (weak entity)
- Double diamond → DEPENDS_ON (identifying relationship)

Importance:

- Ensures dependent or subordinate data is properly linked to parent data.
- Prevents orphan records by enforcing mandatory relationships with strong entities.

Table 8.2: Comparison Table: Strong vs Weak Entity

Feature	Strong Entity	Weak Entity
Existence	Independent	Dependent on strong entity
Primary Key	Has its own primary key	Partial key, combined with strong entity key
ER Diagram Representation	Single rectangle	Double rectangle
Relationship	Can participate in any relationship	Must participate in identifying relationship
Example	Student, Employee, Department	Dependent, Order_Item

Key Points

1. Strong entities are the primary building blocks of a database.
2. Weak entities allow modeling real-world dependent data, such as dependents, invoices, or order items.
3. Correctly identifying strong and weak entities ensures data integrity and proper relational design.

**Activity 8.1:**

How to decide the entity is weak or strong? Give suitable example.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

8.5 DATABASE DESIGN USING ER DIAGRAMS

Database design is the process of creating a blueprint of how data is stored, organized, and accessed in a database. Using Entity-Relationship (ER) diagrams is a crucial part of the conceptual design phase, where real-world data requirements are represented visually to ensure clarity, consistency, and efficiency.

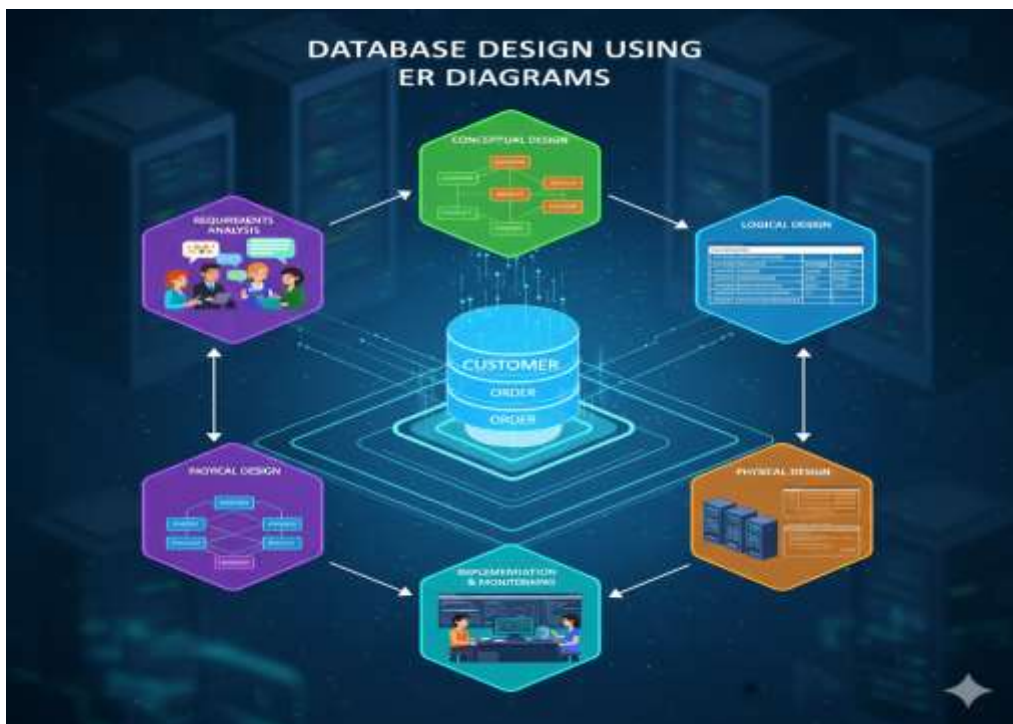


Figure 8.3: Database Design using ER Diagrams

1. Introduction to Database Design Using ER Diagrams

An ER diagram is a graphical representation of a database's entities, attributes, relationships, cardinality, and constraints. Designing a database using ER diagrams helps in:

- Visualizing the structure of data.

NOTE

- Communicating requirements between stakeholders, such as designers, developers, and end-users.
- Reducing redundancy and avoiding anomalies in data storage.
- Ensuring data integrity through clear definition of relationships, keys, and constraints.

ER diagrams form the foundation for converting conceptual designs into relational schemas during the logical design phase.

2. Steps in Database Design Using ER Diagram**Step 1: Requirement Analysis**

- Understand the real-world system to be modeled.
- Collect requirements from stakeholders such as data needs, business rules, and reports.
- Identify entities, attributes, and relationships from the requirements.

Example:

- A university system might include entities like Student, Course, Professor, and relationships like Enrolls and Teaches.

Step 2: Identify Entities and Attributes

- **Entities:** Represent objects or concepts about which data is stored.
- **Attributes:** Define properties of entities and relationships.
- Determine primary keys for strong entities and partial keys for weak entities.

Example:

- Student entity: Attributes
 - Student_ID (PK), Name, Date_of_Birth.
- Course entity: Attributes
 - Course_ID (PK), Course_Name, Credits.

Step 3: Define Relationships

- Identify how entities interact.
- Determine relationship types: unary, binary, ternary, or higher-order.

- Assign cardinality constraints (1:1, 1:N, M:N) and participation constraints (total or partial).

Example:

- Enrolls relationship between Student and Course is many-to-many (M:N) with attributes like Grade.

Step 4: Draw the ER Diagram

- Represent entities as rectangles, attributes as ovals, relationships as diamonds.
- Connect entities and relationships using lines and indicate cardinality and participation constraints.
- Use double rectangles for weak entities and double diamonds for identifying relationships.

Step 5: Review and Refine

- Validate ER diagram with stakeholders to ensure it meets requirements.
- Check for redundancy, missing entities, or improper relationships.
- Apply generalization or specialization if needed.

Step 6: Convert ER Diagram to Relational Schema

- Transform entities into tables.
- Map relationships to foreign keys or junction tables (for M:N relationships).
- Ensure primary keys, foreign keys, and constraints are defined.

Example:

- Student table: Student_ID (PK), Name, DOB
- Course table: Course_ID (PK), Course_Name, Credits
- Enrollment table: Student_ID (FK), Course_ID (FK), Grade

3. Advantages of Database Design Using ER Diagram

1. **Conceptual clarity:** Provides a clear and structured view of the system.
2. **Error reduction:** Identifies potential design flaws before implementation.
3. **Data integrity:** Clearly defines relationships, keys, and constraints.

NOTE

4. **Communication tool:** Facilitates discussions between technical and non-technical stakeholders.
5. **Foundation for normalization:** Helps in creating efficient relational schemas.

4. Best Practices in ER Diagram-Based Database Design

- Use consistent naming conventions for entities, attributes, and relationships.
- Avoid redundant attributes or relationships.
- Clearly define primary and foreign keys.
- Represent weak entities and identifying relationships accurately.
- Include relationship attributes where necessary.

In summary, ER diagrams are the cornerstone of conceptual database design, translating real-world requirements into a structured, visual model. They make it easier to create an efficient, normalized, and maintainable database before implementing it in a DBMS.

Check Your Progress 8.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is an ER diagram, and why is it important in database design?

Question 2: Differentiate between strong and weak entities with examples.

8.6 SUMMARY

Unit 8, *ER Diagrams and Database Design*, focuses on the conceptual design phase of database development, where real-world requirements are transformed into a structured visual model. This unit emphasizes the importance of ER diagrams as a tool to represent entities, attributes, relationships, cardinality, and participation constraints in a database system.

Key points covered in this unit include:

1. **Purpose of ER Diagrams** – ER diagrams provide a visual blueprint of a database, making it easier to understand, communicate, and validate requirements before implementation.
2. **Entities and Attributes** – Identification of strong and weak entities, their attributes (simple, composite, multivalued, derived), and appropriate primary or partial keys.
3. **Relationships** – Defining how entities interact, with types including unary, binary, ternary, and higher-order relationships.
4. **Cardinality and Participation Constraints** – Specifying how many instances of one entity are associated with another and whether participation is mandatory (total) or optional (partial).
5. **ER Diagram Notations** – Standard symbols for entities (rectangles), attributes (ovals), relationships (diamonds), weak entities (double rectangles), identifying relationships (double diamonds), and constraints.
6. **Database Design Process Using ER Diagrams** – Step-by-step methodology:
 - Requirement analysis
 - Identification of entities and attributes
 - Defining relationships
 - Drawing ER diagrams
 - Validation and refinement
 - Converting ER diagrams to relational schemas
7. **Advantages of ER-Based Design** – Ensures data integrity, reduces redundancy, facilitates normalization, and provides a clear communication tool between stakeholders.
8. **Best Practices** – Include consistent naming, accurate representation of weak entities, proper assignment of keys, and careful inclusion of relationship attributes.

Overall, Unit 8 equips learners with the knowledge to design well-structured, efficient, and normalized databases by translating real-world systems into ER diagrams, which then serve as the foundation for relational database implementation.

8.7 GLOSSARY

1. Database Design: The process of creating a structured blueprint for storing, organizing, and accessing data efficiently in a database system.

2. Conceptual Design: The phase of database design where real-world requirements are represented visually, usually through ER diagrams, without considering physical storage details.

NOTE

3. ER Diagram (Entity-Relationship Diagram): A graphical representation of entities, attributes, and relationships in a database, used for conceptual modeling.

4. Entity: A real-world object, person, or concept about which data is stored.

5. Strong Entity: An independent entity that can be uniquely identified by its own primary key. Represented by a single rectangle.

6. Weak Entity: An entity that cannot be uniquely identified by its own attributes and depends on a strong entity for identification. Represented by a double rectangle.

7. Attribute: A property or characteristic of an entity or relationship. Represented by an oval.

8. Simple Attribute: An indivisible attribute that cannot be broken into smaller components.

- Example: First_Name

9. Composite Attribute: An attribute that can be divided into smaller sub-attributes.

- Example: Full_Name → {First_Name, Last_Name}

10. Multivalued Attribute: An attribute that can have multiple values for a single entity instance. Represented by a double oval.

- Example: Phone_Numbers

11. Derived Attribute: An attribute whose value can be derived from other attributes. Represented by a dashed oval.

- Example: Age derived from Date_of_Birth

12. Key Attribute: An attribute (or set of attributes) that uniquely identifies an entity instance. Represented by underlined text.

13. Relationship: An association between two or more entities. Represented by a diamond in ER diagrams.

14. Relationship Attribute: An attribute that belongs to a relationship rather than an entity, providing additional information about the association.

- Example: Grade in Student-Course enrollment

NOTE

15. Cardinality: Specifies the number of instances of one entity that can or must be associated with instances of another entity (1:1, 1:N, M:N).

8.8 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What is an ER diagram, and why is it important in database design?

Answer:

An ER (Entity-Relationship) diagram is a graphical representation of entities, their attributes, and the relationships between them in a database.

Importance:

1. Provides a visual blueprint of the database structure.
2. Helps identify entities, attributes, and relationships clearly.
3. Reduces redundancy and ensures data integrity.
4. Serves as a communication tool between designers, developers, and end-users.
5. Forms the basis for converting conceptual design into a relational schema.

Q 2: Differentiate between strong and weak entities with examples.

Answer:

Feature	Strong Entity	Weak Entity
Definition	Independent entity uniquely identified by its own primary key	Dependent entity that cannot be uniquely identified by its own attributes
Existence	Can exist independently	Existence depends on a strong entity
ER Notation	Single rectangle	Double rectangle
Relationship	Can participate in any relationship	Must participate in identifying relationship
Example	Student, Employee	Dependent of an employee

Example Explanation:

NOTE

- Student is a strong entity with Student_ID as the primary key.
- Dependent is a weak entity; it needs the Employee_ID of the parent employee for unique identification.

8.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13713	

8.10 REFERENCES

- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson Education.
- Date, C. J. (2019). *An introduction to database systems* (8th ed.). Pearson Education.
- Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill Education.

8.11 EXERCISE**PART 1****A. Multiple Choice Qs**

1. Which symbol is used to represent a weak entity in an ER diagram?

- A) Single rectangle
- B) Double rectangle
- C) Diamond
- D) Oval

2. What does a diamond shape represent in an ER diagram?

- A) Entity
- B) Attribute
- C) Relationship

D) Weak Entity

3. Which of the following correctly represents a multivalued attribute?

A) Oval with dashed border

B) Double oval

C) Underlined oval

D) Rectangle

4. What does total participation of an entity in a relationship mean?

A) All instances of the entity must participate in the relationship

B) Some instances may or may not participate

C) The entity has a multivalued attribute

D) The entity has a derived attribute

5. What is the main purpose of using ER diagrams in database design?

A) To store data physically in the database

B) To visually represent entities, attributes, and relationships in a database

C) To create application interfaces

D) To write SQL queries

Explanation of MCQs

Q1 – B) Double rectangle

Explanation:

A weak entity depends on a strong entity for identification and cannot be uniquely identified by its own attributes. In ER diagrams, weak entities are represented by double rectangles, while strong entities use a single rectangle.

Q2 – C) Relationship

Explanation:

In ER diagrams, diamonds represent relationships between entities. Entities are rectangles, and attributes are ovals. Diamonds can also include attributes that belong specifically to the relationship.

Q3 – B) Double oval

NOTE**Explanation:**

A multivalued attribute can have multiple values for a single entity instance. It is represented in ER diagrams by a double oval. For example, a Student entity may have multiple Phone_Numbers.

Q4 – A) All instances of the entity must participate in the relationship

Explanation:

Total participation means every instance of the entity is involved in the relationship. It is depicted with a double line connecting the entity to the relationship in ER diagrams.

Q5 – B) To visually represent entities, attributes, and relationships in a database

Explanation:

The primary purpose of ER diagrams is to provide a conceptual, visual representation of the database structure. This helps in understanding data requirements, designing relationships, and ensuring data integrity before implementing the database physically.

B. Short Answer Qs (100 Words)

1. What is an Entity-Relationship diagram?
2. What are the basic components of an ER diagram?
3. What is an entity set?
4. Explain the purpose of attributes in an ER diagram.
5. What is cardinality in an ER diagram?

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the basic components of an ER diagram.
2. Describe the process of creating an ER diagram.
3. Explain different types of attributes used in ER modelling.
4. Discuss the role of ER diagrams in database design.
5. Explain cardinality and participation constraints in ER diagrams.

PART 3**D. Long Answer Type Questions (300 Words)**

1. Explain Entity-Relationship modelling and discuss its importance in database design.

NOTE

2. Describe the complete process of designing an ER diagram for a real-world database application.
3. Explain entities, attributes, relationships, cardinality, and participation constraints with suitable examples.
4. Discuss how an ER diagram can be converted into a relational database design.
5. Analyze the importance of ER diagrams in identifying database requirements and reducing design errors.



DR. C.V. RAMAN UNIVERSITY

Chhattingarh, Bilaspur AN A/SECT GROUP UNIVERSITY

Approved by: PCI / AICTE / NCTE / BCI / Member of: AMU / Joint Committee / UGC / SER / AICTE / Recognized by: UGC / A NAAC Accredited University



DR. C. V. RAMAN UNIVERSITY (C.G.)

SLM FEEDBACK FORM

PROGRAMME : _____

COURSE : _____

BLOCK : _____

UNIT : _____

Dear Student,

While studying the units of this block, you may have found certain portions of the text difficult to comprehend. We wish to know your difficulties and suggestions, in order to improve the course. Therefore, we request you to fill up and send us the following questionnaire, which pertains to this block. If you find the space provided insufficient, kindly use a separate sheet.

Please mail to aisectcontent@aisect.org/ for online submission please scan QR code:

Questionnaire

1. Name : _____

2. Enrolment No. :

--	--	--	--	--	--	--	--	--	--	--

3. How many hours did you need for studying the units? :

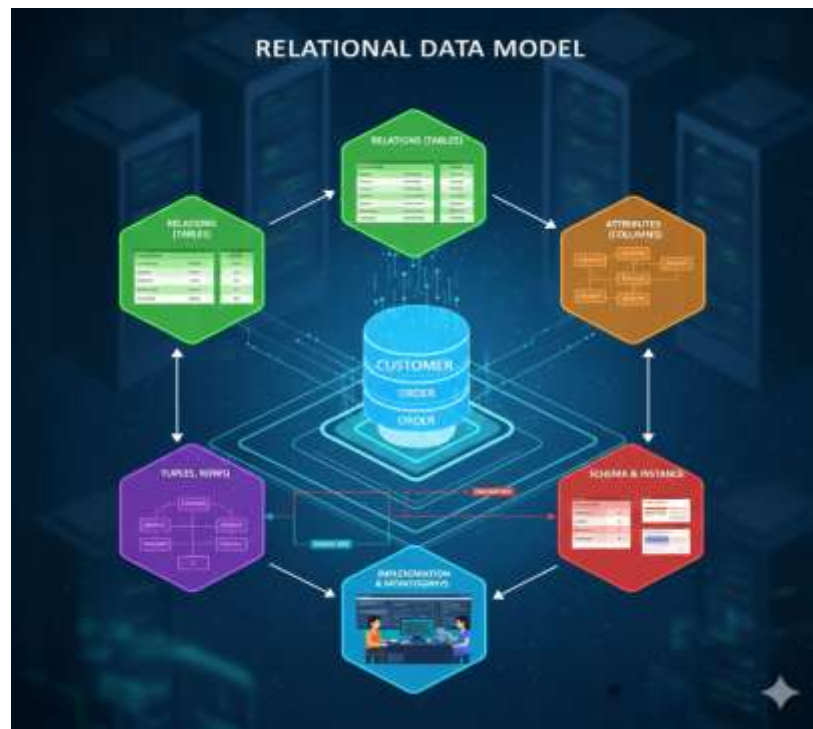
--

4. Please give your reactions to the following items based on your reading of the block:

Items	Excellent	Very Good	Good	Poor	Give specific Examples (If poor)
Presentation Quality					
Language and Style					
Illustrations					
Used (diagrams, tables, etc.)					
Conceptual Clarity					
Check Your Progress Questions					
Suggested Readings - OER					

5. Any other comments:

BLOCK IV	RELATIONAL DATA MODEL	169-226
Unit 9	Relational Model Concepts	171
Unit 10	Relational Constraints	187
Unit 11	Keys in Relational Model	203



BLOCK IV: Relational Data Model focuses on the fundamental principles of organizing data in a structured, table-based format to enable efficient storage, retrieval, and manipulation. The relational data model represents data as a collection of relations, commonly known as tables, where each table consists of rows (tuples) and columns (attributes). It emphasizes the use of keys, such as primary keys and foreign keys, to uniquely identify records and maintain relationships between tables, ensuring data integrity and consistency. This block also covers essential concepts like relational algebra and relational operations, which form the theoretical foundation for querying and manipulating relational databases. By understanding the relational data model, learners can design normalized databases that reduce redundancy, enforce constraints, and support efficient transactions, forming the backbone of modern database systems.

UNIT 9

RELATIONAL MODEL CONCEPTS

UNIT STRUCTURE

- 9.1 Introduction
- 9.2 Objective
- 9.3 Relational Model Basics
- 9.4 Relations and Tuples
- 9.5 Domain and Schema
- 9.6 Summary
- 9.7 Glossary
- 9.8 Answer to Check Your Progress
- 9.9 Suggested Readings – OER
- 9.10 Reference
- 9.11 Exercise

9.1 INTRODUCTION

The relational model is one of the most widely used approaches for organizing and managing structured data. It represents information using relations, commonly visualized as tables, in which data is organized into rows and columns. Because of its simplicity, flexibility, and well-defined mathematical foundation, the relational model has become a central concept in database management systems. This unit introduces the fundamental concepts of the relational model. Students will learn about relations, tuples, attributes, domains, degree, and cardinality. A relation represents a collection of related data, while a tuple represents an individual record within that relation. Attributes represent properties or characteristics of the data, and domains define the permissible values for attributes. The unit also explains the basic characteristics and properties of relational tables. Students will learn how data is structured within relations and how relational concepts support organized data storage and retrieval. Understanding these concepts is essential for working with relational databases and SQL. The relational model also provides a foundation for understanding keys, relational constraints, and database integrity. These concepts will be developed further in subsequent units. By studying the relational model, learners will gain the conceptual knowledge required to design tables, interpret relational database structures, and work effectively with database management systems based on the relational approach.

NOTE

9.2 OBJECTIVES

After studying this unit, you will be able to:

1. Define the fundamental concepts of the relational model.
2. Explain relations, tuples, attributes, domains, degree, and cardinality.
3. Differentiate between various components of a relational database.
4. Analyze the structure and properties of relational tables.
5. Evaluate the suitability of the relational model for database applications.

9.3 RELATIONAL MODEL BASICS

The relational model is a formal approach to managing and organizing data in databases, introduced by E.F. Codd in 1970. It provides a mathematical foundation for database systems, representing data in the form of relations, which are conceptually similar to tables. The relational model emphasizes data independence, consistency, and integrity, making it the foundation of modern relational database management systems (RDBMS). Here's a detailed breakdown of its basics:

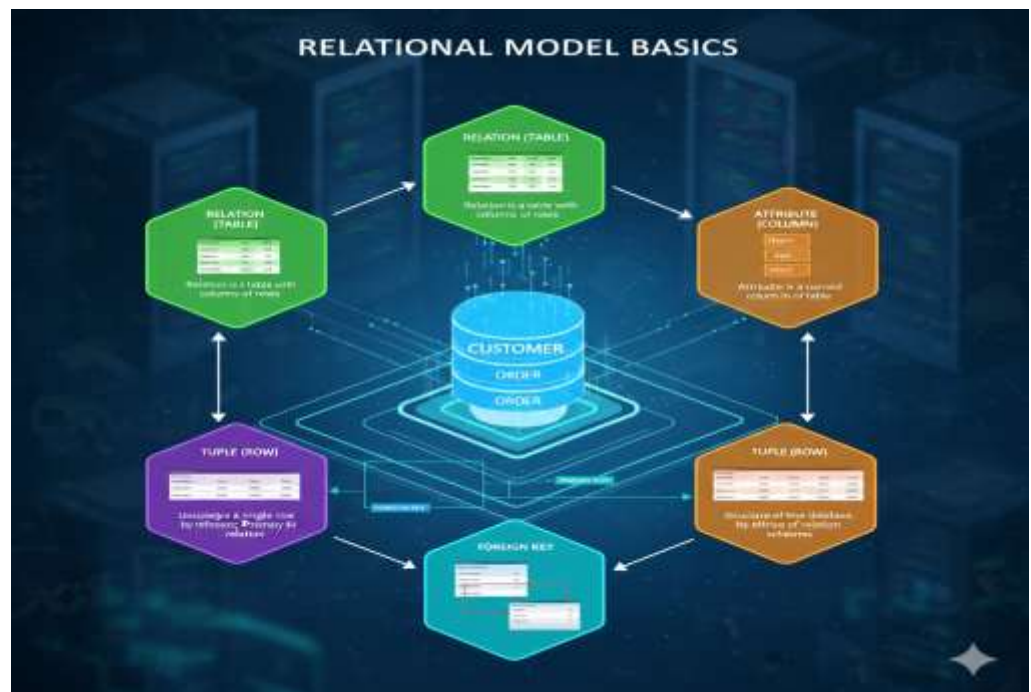


Figure 9.1: Relational Model Basics

1. Relation (Table)

- A relation is a set of tuples (rows), each representing a single record in the table.
- Relations have a name and consist of attributes (columns), which define the type of data stored.
- Each relation should have a unique name within a database.
- Example:

A Student relation might have attributes: StudentID, Name, Age, Course.

2. Tuple (Row)

- A tuple is a single record in a relation.
- It is an ordered set of values, one for each attribute.
- Tuples in a relation are distinct, meaning no two tuples are exactly the same.
- Example: (S101, Ankur, 21, B.Sc CS) is a tuple in the Student relation.

3. Attribute (Column)

- An attribute is a named column in a relation.
- Each attribute has a domain, which specifies the set of valid values.
- Example: Age might have a domain of integers from 18 to 100.

4. Domain

- A domain is a set of permissible values for an attribute.
- It ensures data consistency and type safety.
- Example: Domain of Course could be {B.Sc CS, B.Sc DS, BCA}.

5. Schema and Instance

- **Relation schema:** The structure of a relation, including its name, attributes, and domains.
 - Example: Student(StudentID: int, Name: string, Age: int, Course: string)
- **Relation instance:** The actual set of tuples stored in the relation at a given time.
 - Example:

{{S101, Ankur, 21, B.Sc CS}, {S102, Riya, 22, BCA}}

6. Keys

NOTE

Keys are special attributes used to identify tuples uniquely and maintain integrity.

- **Primary Key:** A unique attribute (or combination) that identifies each tuple in a relation.
 - Example: StudentID in Student table.
- **Candidate Key:** An attribute (or set) that could serve as a primary key.
 - Example: Both StudentID and Email could be candidate keys.
- **Foreign Key:** An attribute in one relation that refers to the primary key of another relation, establishing referential integrity.
 - Example:

CourseID in Enrollment table referring to CourseID in Course table.

7. Integrity Rules

- **Entity Integrity:** No primary key value can be NULL.
- **Referential Integrity:** A foreign key must either be NULL or match a value in the referenced primary key.
- **Domain Integrity:** Attribute values must conform to their defined domains.

8. Relational Operations (Brief)

- **Relational algebra** operations manipulate relations to retrieve data:
 - **Select (σ):** Extracts tuples satisfying a condition.
 - **Project (π):** Extracts specific attributes from a relation.
 - **Union ($\cup$), Intersection ($\cap$), Difference ($-$):** Set operations on relations.
 - **Join ($\bowtie$):** Combines related tuples from two relations.

9. Advantages of the Relational Model

- Simple, tabular representation makes it easy to understand.
- Strong mathematical foundation ensures correctness.
- Data independence allows changes in schema without affecting applications.
- Supports powerful querying using relational algebra or SQL.
- Enforces integrity constraints to maintain high-quality data.

In essence, the relational model organizes data in tables with rows and columns, uses keys and constraints to maintain integrity, and provides formal operations to manipulate and query data efficiently.

Mastery of these basics is essential for designing robust, scalable, and normalized databases.

9.4 RELATIONS AND TUPLES

In the relational data model, a relation is more than just a static table of data—it represents a dynamic collection of tuples that can be updated while maintaining the integrity and consistency of the database. Understanding how relations are updated is essential for managing a relational database effectively. Here's a detailed breakdown:

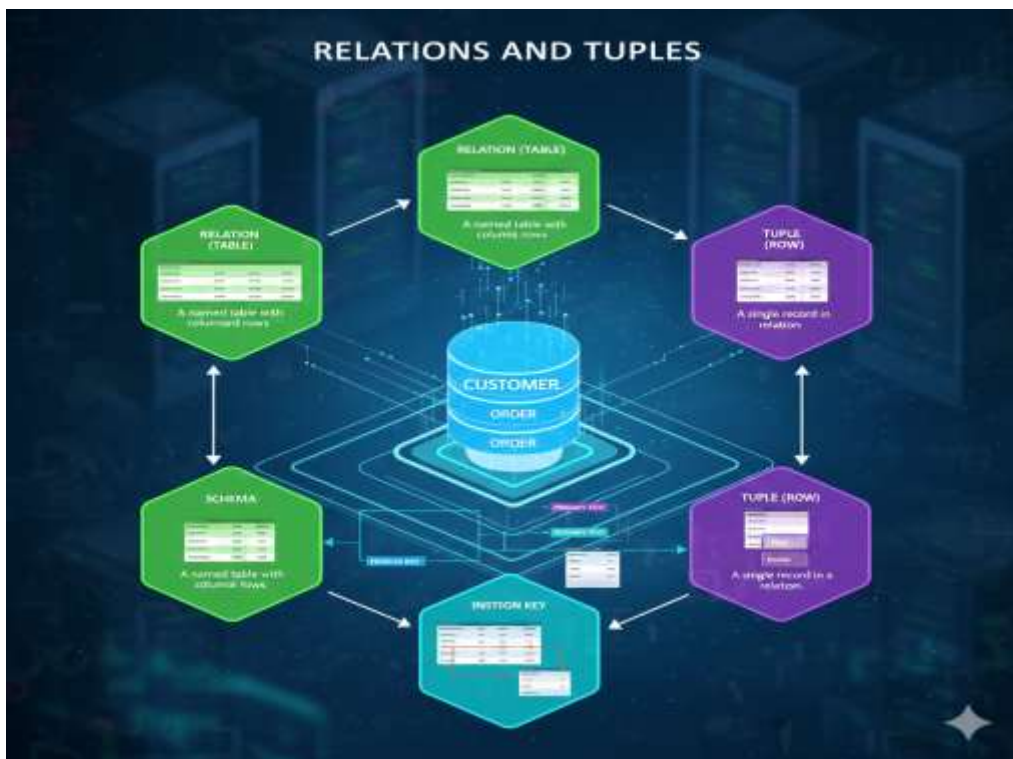


Figure 9.2: Relations and Tuples

1. Relation Recap

- A relation is a table with rows (tuples) and columns (attributes).
- Each relation has a schema, defining its structure, and an instance, representing the current set of tuples.
- Tuples in a relation must be unique, and attribute values must follow their domains.
- Keys (primary and foreign) enforce uniqueness and referential integrity.

2. Need for Updates

NOTE

Databases are rarely static; information changes over time. Updates are operations that modify the content of relations to reflect the current state of the real world. The three main types of updates are:

1. Insertion (Insert)

- Adds new tuples to a relation.
- Must comply with:
 - Domain constraints (values must be valid for each attribute).
 - Entity integrity (primary key cannot be NULL).
 - Referential integrity (foreign key values must exist in the referenced relation).
- Example: Adding a new student to the Student table:

```
INSERT INTO Student (StudentID, Name, Age, Course)
VALUES ('S103', 'Rohit', 20, 'B.Sc DS');
```

2. Deletion (Delete)

- Removes existing tuples from a relation.
- Must maintain referential integrity:
 - If a tuple referenced by a foreign key is deleted, dependent tuples may need to be updated or cascaded deleted.
- Example: Removing a student who has completed the course:

```
DELETE FROM Student WHERE StudentID = 'S101';
```

3. Modification / Update (Update)

- Changes the value(s) of one or more attributes in existing tuples.
- Must ensure:
 - Domain constraints are still valid.
 - Entity integrity and referential integrity are not violated.
- Example: Updating a student's course:

```
UPDATE Student
SET Course = 'BCA'
WHERE StudentID = 'S102';
```

3. Problems and Considerations in Updates

Updating relations requires careful attention to integrity constraints to avoid anomalies:

- **Insertion Anomaly:** Occurs when certain attributes cannot be inserted without other attributes.
 - Example: Cannot insert a course enrollment if the student doesn't exist.
- **Deletion Anomaly:** Unintended loss of data due to deleting a tuple.
 - Example: Deleting a student may also remove course enrollment info.
- **Update Anomaly:** Inconsistencies caused by updating a value in one place but not in others.
 - Example: Changing a course name in one tuple but not all related tuples.

These anomalies are minimized by proper database design and normalization.

4. Transaction and Atomic Updates

- Updates in relational databases are usually performed as transactions.
- A transaction is a sequence of operations (insert, delete, update) that must be atomic—either all succeed, or none succeed.
- This ensures consistency, especially in multi-user environments.

5. Relational Algebra Perspective

From the theoretical standpoint, updates can be represented using relational algebra operations:

- **Insertion:** Union of the existing relation and the new tuple(s).
- **Deletion:** Set difference of the existing relation and the tuples to remove.
- **Modification:** Combination of deletion (of old tuples) and insertion (of updated tuples).

This formalism ensures that updates maintain the mathematical properties of the relational model.

6. Best Practices for Updates

- Always check constraints before performing updates.
- Use transactions to maintain atomicity.
- Normalize relations to reduce redundancy and minimize anomalies.
- Use triggers and cascading rules to maintain referential integrity automatically.

NOTE

In summary, relations and updates highlight how dynamic data in a relational database is managed. Proper handling of inserts, deletes, and updates, while maintaining integrity constraints, is crucial for data consistency, reliability, and correctness. Understanding this ensures robust and error-free database operations.



Activity 9.1:

Create the relational tables and describe how to update the table by using insert, delete etc.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

9.5 DOMAIN AND SCHEMA

In the relational data model, domain and schema are foundational concepts that define the structure, constraints, and meaning of data in a database. They provide the rules that govern what values can be stored and how relations are organized. Understanding these concepts is crucial for designing reliable and consistent databases.

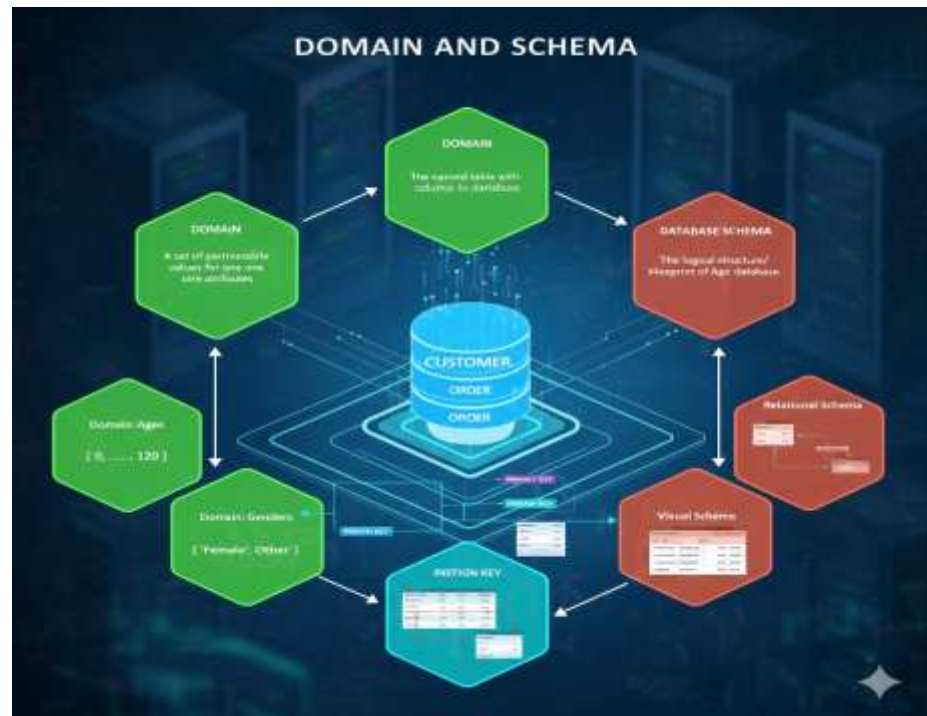


Figure 9.3: Domain and Schema

1. Domain

Definition:

A domain is the set of all permissible values that an attribute can take in a relation. It acts as a type constraint, ensuring that data stored in a database is valid and consistent.

Key Features of Domain:

1. **Value Type:** Each domain specifies the data type of the attribute, such as integer, string, date, boolean, or floating-point.
 - Example: Age $\rightarrow$ integer; Name $\rightarrow$ string.
2. **Value Range / Restrictions:** Domains may define a range or set of allowable values.
 - Example: Age must be between 18 and 100.
 - Example: Course $\in$ {B.Sc CS, B.Sc DS, BCA}.
3. **Atomicity:** Values in a domain **are atomic**, meaning indivisible.
 - Example: A PhoneNumber field should store a single number, not multiple numbers combined.
4. **Consistency:** Domains prevent storing invalid or inconsistent data.

Importance of Domain:

- Maintains data integrity by restricting input values.
- Ensures uniformity in attribute data across all tuples.
- Simplifies validation during insertion and updates.

2. Schema**Definition:**

A schema is the structural blueprint of a relation or an entire database. It defines:

- The name of the relation.
- The attributes (columns) of the relation.
- The domains of these attributes.
- Constraints like keys and referential integrity rules.

Types of Schemas:

1. **Relation Schema:**
 - Defines the structure of a single relation.
 - Example:

Student(StudentID: integer, Name: string, Age: integer, Course: string)

Here:

- Student is the relation name.

NOTE

- Attributes: StudentID, Name, Age, Course.
- Domains: integer for StudentID and Age, string for Name and Course.

2. Database Schema:

- Describes the entire database structure.
- Includes all relations and their attributes, domains, and constraints.
- Example:

Database containing Student, Course, Enrollment relations, each with its relation schema.

Key Features of Schema:

- **Static Definition:** The schema defines how data is stored, but it doesn't include actual data (that's the instance).
- **Constraint Specification:** Includes primary keys, foreign keys, uniqueness, and referential integrity rules.
- **Data Independence:** Changing the data values (tuples) doesn't affect the schema; only structural changes require schema modification.

3. Relationship Between Domain and Schema

- A schema uses domains to define the type and permissible values for each attribute.
- Domains ensure that the data stored in a relation conforms to the rules defined by the schema.
- Example: In the Student relation:
 - Schema defines attributes: StudentID, Name, Age, Course.
 - Domains restrict StudentID to integers, Name to strings, Age to 18–100, and Course to specific course names.

4. Benefits of Domain and Schema

- **Data Integrity:** Ensures only valid data is stored.
- **Consistency:** All tuples in a relation follow the same structure and rules.
- **Query Accuracy:** Helps relational operations and queries retrieve accurate results.
- **Ease of Maintenance:** Modifications are easier because structure and constraints are well-defined.
- **Domain:** Defines the set of permissible, atomic values for an attribute.

- **Schema:** Defines the structure of a relation or database, including attributes, domains, and constraints. Together, domains and schemas form the foundation of relational database design, ensuring accuracy, integrity, and consistency of data.

Check Your Progress 9.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is the difference between a primary key and a foreign key in a relational database?

Question 2: Explain the role of domains in a relational database.

9.6 SUMMARY

Unit 9 introduces the relational model, the fundamental framework for organizing, storing, and managing data in modern database systems. The unit begins with basic concepts of relations, explaining how data is represented in tables (relations) with rows (tuples) and columns (attributes). It emphasizes the importance of domains in defining permissible values for attributes and schemas in structuring relations and the overall database.

The unit explores the concepts of keys—primary, candidate, and foreign—and their role in maintaining entity and referential integrity. Students learn how constraints ensure the consistency and validity of data. The unit also addresses relational updates, including insertion, deletion, and modification, highlighting the importance of maintaining integrity during data changes. Finally, it briefly covers relational operations and algebra, providing the theoretical foundation for querying and manipulating relational data.

By the end of this unit, learners understand how to design robust, normalized relational databases, maintain data integrity, and perform safe updates and queries, which are essential for effective database management.

NOTE

9.7 GLOSSARY

- **Relational Model** – A formal framework for organizing data into relations (tables), introduced by E.F. Codd, emphasizing data integrity and mathematical rigor.
- **Relation** – A table consisting of tuples (rows) and attributes (columns) that represents an entity or relationship in a database.
- **Tuple** – A single row in a relation, representing one record or instance of the entity.
- **Attribute** – A named column in a relation, representing a property or characteristic of the entity.
- **Domain** – The set of all permissible values that an attribute can take, ensuring data consistency.
- **Schema** – The structural definition of a relation or database, including relation name, attributes, domains, and constraints.
- **Relation Instance** – The current set of tuples stored in a relation at a given point in time.
- **Primary Key** – An attribute (or set of attributes) that uniquely identifies each tuple in a relation.
- **Candidate Key** – An attribute (or set of attributes) that could serve as a primary key; one is chosen as the primary key.
- **Foreign Key** – An attribute in one relation that refers to the primary key of another relation, ensuring referential integrity.
- **Entity Integrity** – A rule stating that primary key values cannot be NULL to uniquely identify tuples.
- **Referential Integrity** – A rule that ensures foreign key values match primary key values in the referenced relation or are NULL.
- **Relational Update** – Operations that modify the content of a relation, including insertion, deletion, and modification of tuples.
- **Insertion Anomaly** – A problem where certain data cannot be inserted without adding other data.
- **Deletion Anomaly** – A problem where deleting one tuple causes unintended loss of related data.

9.8 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What is the difference between a primary key and a foreign key in a relational database?

Answer:

- **Primary Key:** An attribute (or combination of attributes) that uniquely identifies each tuple in a relation. It cannot have NULL values.

NOTE

- Foreign Key: An attribute in one relation that refers to the primary key of another relation, establishing a link between the two tables. It ensures referential integrity, meaning the value must exist in the referenced relation or be NULL.

Example: In a Student table, StudentID can be the primary key, while in an Enrollment table, StudentID can be a foreign key referencing the Student table.

Q 2: Explain the role of domains in a relational database.**Answer:**

A domain defines the set of permissible values for an attribute. It ensures that all values stored in the attribute are valid, consistent, and atomic. Domains help maintain data integrity by preventing invalid or incompatible data from being inserted.

Example:

- Attribute: Age
- Domain: Integer values from 18 to 100
This ensures that no invalid age values (like -5 or 200) can be stored in the database.

9.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13714	

9.10 REFERENCES

- Silberschatz, A., Korth, H., & Sudarshan, S. (2019). *Database System Concepts*. McGraw-Hill.
- Elmasri, R., & Navathe, S. (2017). *Fundamentals of Database Systems*. Pearson.
- Date, C. J. (2018). *An Introduction to Database Systems*. Pearson.

9.11 EXERCISE**PART 1**

NOTE**A. Multiple Choice Qs**

Q 1: Which of the following is a primary key constraint in a relational table?

- a) Ensures no duplicate tuples in the table
- b) Allows NULL values in the attribute
- c) References another table's attribute
- d) Ensures values are within a domain

Q 2: A foreign key is used to:

- a) Uniquely identify tuples in the same table
- b) Maintain a relationship between two tables
- c) Store multiple values in a single column
- d) Define the domain of an attribute

Q 3: What does a **domain** define in a relational database?

- a) The number of tuples in a relation
- b) The permissible values for an attribute
- c) The primary key of a relation
- d) The type of join operation

Q 4: Which of the following operations is part of relational algebra?

- a) Insert
- b) Update
- c) Select
- d) Transaction

Q 5: Which anomaly occurs when deleting a tuple causes unintended loss of other related data?

- a) Insertion anomaly
- b) Deletion anomaly
- c) Update anomaly
- d) Key anomaly

Explanation of MCQs

Q1 - a) Ensures no duplicate tuples in the table

Explanation: A primary key uniquely identifies each tuple and cannot contain duplicate or NULL values.

Q2 - b) Maintain a relationship between two tables

Explanation: A foreign key links one table's attribute to the primary key of another table, ensuring referential integrity.

Q3 - b) The permissible values for an attribute

Explanation: Domains restrict attribute values to valid, atomic, and consistent data.

Q4 – c) Select

Explanation: Select (σ) is a relational algebra operation used to retrieve tuples satisfying a condition.

Q5 – b) Deletion anomaly

Explanation: Deleting a tuple may inadvertently remove essential information stored in the same relation, causing a deletion anomaly.

B. Short Answer Qs (100 Words)

1. What is the relational model?
2. What is a relation in a relational database?
3. Define tuple and attribute.
4. What is the degree of a relation?
5. What is the cardinality of a relation?

PART 2

C. Long Answer Type Questions (200 Words)

1. Explain the basic concepts of the relational model.
2. Describe relations, tuples, attributes, and domains.
3. Explain the characteristics of a relational table.
4. Differentiate between degree and cardinality of a relation.
5. Discuss the advantages of the relational model.

PART 3

D. Long Answer Type Questions (300 Words)

1. Explain the relational model in detail and discuss its fundamental concepts.
2. Describe relations, tuples, attributes, domains, degree, and cardinality with suitable examples.
3. Explain the characteristics and properties of relations in a relational database.
4. Discuss the advantages and limitations of the relational model.
5. Analyze why the relational model is widely used for developing database applications.

NOTE

UNIT 10

RELATIONAL CONSTRAINTS

NOTE

UNIT STRUCTURE

- 10.1 Introduction
- 10.2 Objective
- 10.3 Domain Constraints
- 10.4 Entity Integrity Constraints
- 10.5 Referential Integrity Constraints
- 10.6 Summary
- 10.7 Glossary
- 10.8 Answer to Check Your Progress
- 10.9 Suggested Readings – OER
- 10.10 Reference
- 10.11 Exercise

10.1 INTRODUCTION

Maintaining accurate, consistent, and valid data is one of the most important objectives of a database management system. Databases frequently contain large volumes of information that are accessed and modified by different users and applications. Without appropriate rules, invalid or inconsistent information may enter the database. Relational constraints provide mechanisms for controlling the validity of data stored in relational tables. This unit introduces the concept of relational constraints and explains their importance in maintaining database integrity. Students will study different types of constraints, including domain constraints, entity integrity constraints, and referential integrity constraints. Domain constraints ensure that attribute values belong to an appropriate set or domain. Entity integrity ensures that important identifying attributes, particularly primary keys, maintain valid values. Referential integrity focuses on relationships between tables and ensures that references between related records remain valid. These constraints help prevent problems such as invalid references, duplicate identifying values, and inappropriate data entries. The unit emphasizes the role of constraints in database design and management. Students will learn how constraints support data accuracy and consistency and how they help protect databases from invalid operations. A strong understanding of relational constraints is essential for designing reliable databases and forms an important foundation for studying keys, SQL table definitions, and database integrity mechanisms.

NOTE

10.2 OBJECTIVES

After studying this unit, you will be able to:

- Define relational constraints and database integrity.
- Explain domain, entity integrity, and referential integrity constraints.
- Differentiate between various types of relational constraints.
- Analyze the role of constraints in maintaining data accuracy and consistency.
- Evaluate the effectiveness of integrity constraints in preventing invalid database operations.

10.3 DOMAIN CONSTRAINTS

In the relational data model, domain constraints are one of the most fundamental types of relational constraints. They ensure that the values stored in a database are valid, consistent, and meaningful according to the predefined rules for each attribute.

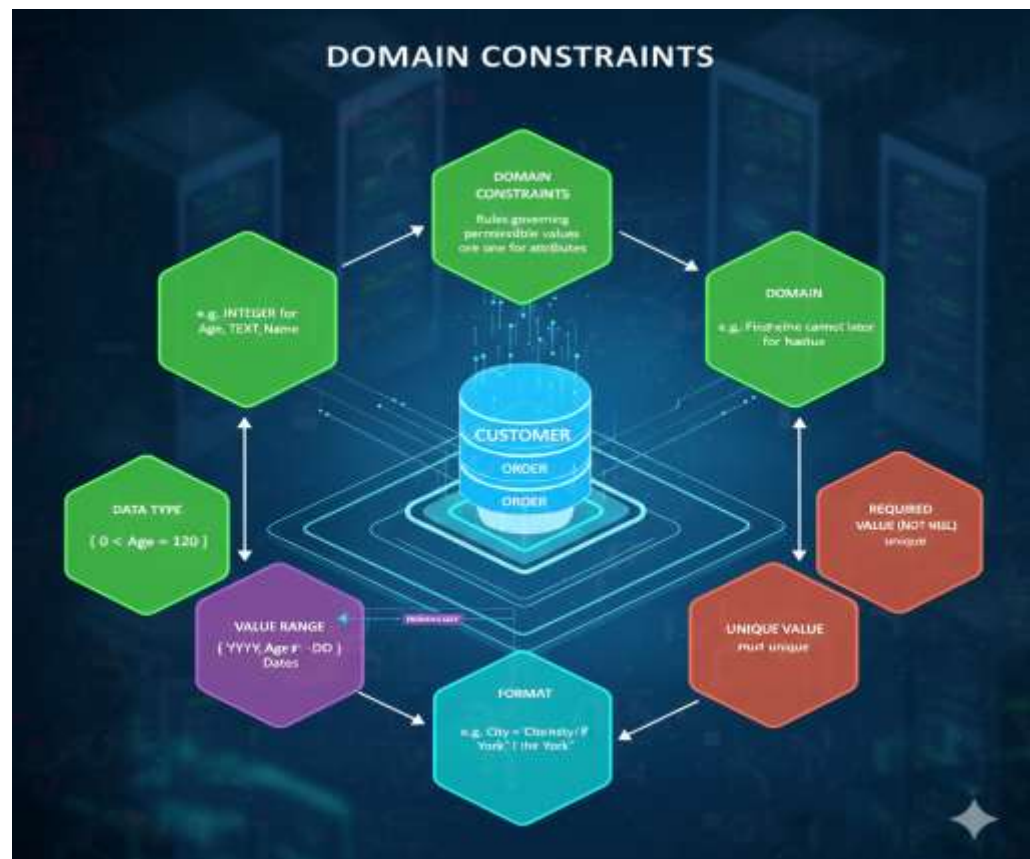


Figure 10.1: Domain Constraints

1. Definition of Domain Constraint

A domain constraint specifies that the values of an attribute must belong to a predefined set or domain. Each attribute in a relation is associated with a domain, which defines:

- The data type of the attribute (e.g., integer, string, date).
- The range or set of permissible values.
- The format or structure of the data (if applicable).

Mathematical Representation:

If A is an attribute in a relation R, then the value of A in any tuple must belong to the domain D(A):

$$\forall t \in R, t[A] \in D(A)$$

This ensures that every value for attribute A in relation R is valid according to the rules of the domain.

2. Key Features of Domain Constraints

1. Data Type Enforcement:

- Ensures that attribute values are of the correct type.
- Example: Age attribute must be an integer; Name must be a string.

2. Atomicity:

- Each value in a domain must be atomic, i.e., indivisible.
- Example: A PhoneNumber field should store a single number, not multiple numbers separated by commas.

3. Value Range / Restrictions:

- Domains can restrict values to a specific range or set.
- Example: Age $\in [18, 100]$; Gender $\in \{\text{'Male'}, \text{'Female'}, \text{'Other'}\}$.

4. Consistency:

- All tuples for a particular attribute must have values from the same domain, ensuring uniformity across the table.

5. Null Values:

- Domain constraints can also specify whether NULL values are allowed for an attribute.

Table 10.1: Domain Constraints

Attribute	Domain Specification	Example Values
StudentID	Integer, non-null	101, 102, 103

NOTE

Attribute	Domain Specification	Example Values
Name	String (max 50 characters), non-null	'Ankur', 'Riya'
Age	Integer between 18 and 100	21, 25, 30
Gender	{'Male', 'Female', 'Other'}, non-null	'Male', 'Female'
EnrollmentDate	Date format YYYY-MM-DD	'2026-02-16'

Example in SQL:

```
CREATE TABLE Student (
    StudentID INT NOT NULL,
    Name VARCHAR(50) NOT NULL,
    Age INT CHECK (Age >= 18 AND Age <= 100),
    Gender VARCHAR(10) CHECK (Gender IN ('Male', 'Female', 'Other')),
    EnrollmentDate DATE);
```

Here, CHECK clauses enforce the domain constraints for Age and Gender.

4. Importance of Domain Constraints

- **Data Integrity:** Prevents invalid data entry.
- **Consistency:** Ensures all tuples for an attribute follow the same rules.
- **Error Prevention:** Reduces the chances of anomalies during insertion, update, or deletion.
- **Simplifies Validation:** Application logic doesn't need to enforce these rules manually; the database ensures compliance.
- Domain constraints restrict attribute values to a valid set or type.
- They enforce data type, atomicity, and range rules.
- They are essential for database integrity and consistency, forming the first line of defense against invalid data entry.

10.4 ENTITY INTEGRITY CONSTRAINTS

In relational databases, entity integrity is a fundamental constraint that ensures each entity (tuple) in a relation is uniquely identifiable. It is one of the most critical rules for maintaining data correctness and reliability.



Figure 10.2: Entity Integrity Constraints

1. Definition of Entity Integrity Constraint

The entity integrity constraint states that the primary key of a relation cannot contain NULL values. This rule guarantees that every tuple (row) in a table is distinct and uniquely identifiable.

- **Primary Key:** A set of one or more attributes that uniquely identifies each tuple in a relation.
- **NULL Restriction:** None of the attributes that form the primary key can be NULL, because a NULL value represents an unknown or missing value, which cannot uniquely identify a record.

Formal Representation:

If R is a relation and PK is its primary key, then for every tuple t in R:

$$t[PK] \neq \text{NULL}$$

2. Key Features of Entity Integrity

1. **Uniqueness:**
 - Ensures that no two tuples in a relation have the same primary key value.
2. **Non-nullability:**

NOTE

- Primary key attributes cannot be NULL. This ensures every entity is uniquely identifiable.
- 3. **Essential for Identifying Entities:**
 - Without entity integrity, it would be impossible to distinguish between different tuples in a table reliably.
- 4. **Foundation for Relationships:**
 - Entity integrity is required for foreign key references, as foreign keys rely on primary keys to enforce referential integrity.

Table 10.2: Entity Integrity: Student Table

StudentID	Name	Age	Course
101	Ankur	21	B.Sc CS
102	Riya	22	BCA
103	Rohit	20	B.Sc DS

- StudentID is the primary key.
- Entity integrity ensures no NULL values in StudentID and no duplicates.

Example – SQL Implementation:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY, -- Enforces entity integrity
  Name VARCHAR(50) NOT NULL,
  Age INT,
  Course VARCHAR(50));
```

- PRIMARY KEY automatically enforces uniqueness and non-nullability for StudentID.

3. Importance of Entity Integrity

- **Prevents Duplicate Records:** Guarantees each entity can be uniquely identified.
- **Supports Referential Integrity:** Foreign keys reference primary keys; entity integrity ensures valid references.
- **Ensures Accurate Queries:** Unique identification allows precise retrieval and manipulation of data.
- **Essential for Normalization:** Entity integrity supports database normalization by preventing redundancy and ambiguity.
- Entity integrity constraint enforces that primary key attributes are unique and never NULL.

NOTE

- It is crucial for uniquely identifying tuples, supporting referential integrity, and ensuring reliable, consistent database operations.
- Without entity integrity, the relational database model would fail to guarantee correctness and uniqueness of entities.

10.5 REFERENTIAL INTEGRITY CONSTRAINTS

In relational databases, the referential integrity constraint is a fundamental rule that ensures consistency and correctness of relationships between tables. It prevents the creation of invalid references in foreign key relationships and maintains the logical connection between related data.



Figure 10.3: Referential Integrity Constraints

1. Definition of Referential Integrity

The referential integrity constraint states that:

“The value of a foreign key in one relation must either be NULL or match a value of the primary key in the referenced relation.”

This ensures that every reference from one table to another is valid. In other words, it is impossible to have a “dangling reference” where a foreign key points to a non-existent primary key.

NOTE**2. Components****1. Primary Key (PK):**

- Uniquely identifies a tuple in the referenced table (parent table).

2. Foreign Key (FK):

- An attribute in the referencing table (child table) that refers to the primary key in the parent table.

3. Parent-Child Relationship:

- Parent Table: The table containing the primary key.
- Child Table: The table containing the foreign key.

3. Rules of Referential Integrity**1. Matching Values:**

- Each foreign key value must either:
 - Match a value in the referenced primary key, or
 - Be NULL if the foreign key allows nulls.

2. No Dangling References:

- You cannot insert a foreign key value that does not exist in the parent table.

3. Cascading Actions (Optional):

- **ON DELETE CASCADE:** Deleting a parent row automatically deletes all child rows referencing it.
- **ON UPDATE CASCADE:** Updating a primary key in the parent table updates all corresponding foreign key values in the child table.

4. Examples of Referential Integrity**Example 1 – Student and Enrollment Tables:****Table 10.3: Parent Table: Student**

StudentID	Name	Age	Course
101	Ankur	21	B.Sc CS
102	Riya	22	BCA

Table 10.4: Child Table: Enrollment

EnrollmentID	StudentID	Subject
1	101	Database
2	102	Algorithms

NOTE

- Here, StudentID in Enrollment is a foreign key referencing StudentID in Student.
- Referential integrity ensures:
 - You cannot insert a row in Enrollment with StudentID = 105 if no such student exists.

Example 2 – SQL Implementation:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(50) NOT NULL,
  Age INT,
  Course VARCHAR(50));
```

```
CREATE TABLE Enrollment (
  EnrollmentID INT PRIMARY KEY,
  StudentID INT,
  Subject VARCHAR(50),
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
  ON DELETE CASCADE
  ON UPDATE CASCADE);
```

- FOREIGN KEY enforces referential integrity.
- Cascading rules automatically maintain consistency when a parent row is deleted or updated.

5. Importance of Referential Integrity

- **Maintains Data Consistency:** Prevents invalid foreign key references.
- **Supports Relationships:** Ensures that related tables remain synchronized.
- **Prevents Orphan Records:** Avoids “child” tuples referencing non-existent “parent” tuples.
- **Simplifies Database Maintenance:** Cascading updates and deletes automatically propagate changes.

6. Common Violations

1. **Dangling References:** A foreign key points to a primary key that does not exist.
2. **Deletion of Parent Without Cascading:** Removing a parent tuple leaves child tuples referencing non-existent data.
3. **Insertion of Invalid Foreign Key Value:** Attempting to insert a value in the foreign key column that does not exist in the parent table.

NOTE

- Referential integrity constraint ensures that foreign key values always reference valid primary key values or are NULL.
- It is crucial for maintaining logical relationships between tables in a relational database.
- Cascading actions (update/delete) help automatically maintain consistency across related tables.
- Together with entity integrity, it forms the core set of constraints that guarantee reliable, consistent, and accurate data.



Activity 10.1:

What is the difference between entity and referential integrity constraints. Take a table and describe.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

Check Your Progress 10.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is the main purpose of the entity integrity constraint in a relational database?

Question 2: Explain the concept of referential integrity constraint with an example.

10.6 SUMMARY

Unit 10 focuses on the relational constraints that enforce rules on data stored in a relational database, ensuring accuracy, consistency, and integrity. The unit emphasizes that while relations (tables) organize data in rows and columns, constraints prevent invalid, inconsistent, or meaningless data from being stored.

The major types of relational constraints covered in this unit include:

NOTE

1. **Domain Constraints** – Restrict attribute values to a predefined set or type, ensuring data validity, atomicity, and consistency.
2. **Entity Integrity Constraint** – Enforces that primary key attributes cannot be NULL, guaranteeing that each tuple in a relation is uniquely identifiable.
3. **Referential Integrity Constraint** – Ensures that a foreign key in a child table must match a primary key in the parent table or be NULL, maintaining consistent relationships between tables.
4. **Key Constraints** – Includes primary, candidate, and unique keys, which enforce uniqueness of tuples and support entity integrity.
5. **Advanced Constraints** – Concepts like assertions and triggers allow enforcement of complex business rules beyond standard key and referential constraints.

The unit also highlights the importance of cascading actions (ON DELETE/ON UPDATE) to automatically maintain integrity when related data changes.

Key Takeaways:

- Constraints are essential for data correctness, consistency, and reliability.
- Domain, entity, and referential constraints form the core of relational integrity enforcement.
- Proper use of constraints, combined with good database design and normalization, prevents anomalies, ensures valid relationships, and enables robust query operations.

In essence, this unit equips learners with the knowledge to design error-free, consistent, and reliable relational databases by applying the right constraints at the schema level.

10.7 GLOSSARY

- **Relational Constraint** – A rule applied to a relational database to enforce data integrity, consistency, and correctness.
- **Domain Constraint** – A restriction that ensures attribute values belong to a predefined set or domain, enforcing valid, atomic, and consistent data.
- **Entity Integrity Constraint** – A rule stating that primary key attributes cannot be NULL, ensuring each tuple in a relation is uniquely identifiable.
- **Referential Integrity Constraint** – A rule that requires foreign key values in a child table to either match a primary key in the

NOTE

parent table or be NULL, maintaining valid relationships between tables.

- **Primary Key** – An attribute or set of attributes that uniquely identifies each tuple in a relation. It cannot contain NULL values.
- **Candidate Key** – An attribute or set of attributes that could serve as a primary key. A table may have multiple candidate keys, but only one is chosen as the primary key.
- **Alternate Key** – A candidate key that is not selected as the primary key but can still uniquely identify tuples.
- **Foreign Key** – An attribute in a relation that references the primary key of another relation, enforcing referential integrity.
- **Unique Key Constraint** – Ensures that the values of an attribute or a combination of attributes are unique across all tuples in a relation.
- **Null Value** – Represents unknown or missing information in a relation. Certain constraints, like entity integrity, prohibit NULL values for primary keys.
- **Cascading Actions** – Actions applied to foreign keys such as ON DELETE CASCADE or ON UPDATE CASCADE to automatically maintain referential integrity when a referenced tuple is deleted or updated.
- **Assertion** – A constraint defined on the database as a whole to enforce complex business rules beyond individual relations.
- **Trigger** – A stored procedure that automatically executes when a specified event occurs (INSERT, UPDATE, DELETE), used to enforce additional integrity rules.

10.8 ANSWER TO CHECK YOUR PROGRESS- POSSIBLE ONE

Q 1: What is the main purpose of the entity integrity constraint in a relational database?

Answer:

The entity integrity constraint ensures that the primary key of a relation cannot be NULL, guaranteeing that each tuple is uniquely identifiable. This prevents duplicate or unidentifiable records and supports accurate data retrieval and relationships.

Example: In a Student table, StudentID is the primary key, so it cannot have NULL values.

Q 2: Explain the concept of referential integrity constraint with an example.

Answer:

The referential integrity constraint ensures that a foreign key in a child

NOTE

table must either match a primary key in the parent table or be NULL. This maintains valid relationships between tables and prevents “dangling references.”

Example:

- Parent table: Student(StudentID, Name)
- Child table: Enrollment(EnrollmentID, StudentID, Course)
- StudentID in Enrollment is a foreign key referencing StudentID in Student.
- Referential integrity ensures that Enrollment.StudentID cannot have a value that does not exist in Student.StudentID.

10.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13715	

10.10 REFERENCE

- Elmasri, R., & Navathe, S. B. (2021). *Fundamentals of Database Systems* (8th ed.). Pearson.
- Covers relational constraints including domain, entity, and referential integrity in detail.
- Connolly, T. M., & Begg, C. E. (2020). *Database Systems: A Practical Approach to Design, Implementation, and Management* (7th ed.). Pearson.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill.

10.11 EXERCISE

PART 1

A. Multiple Choice Qs

Q 1: Which of the following statements about the entity integrity constraint is TRUE?

- a) Primary key attributes can have NULL values.

NOTE

- b) Every tuple must have a unique primary key value.
- c) Foreign keys cannot reference primary keys.
- d) Entity integrity applies only to foreign keys.

Q 2: What is the purpose of a foreign key in a relational database?

- a) To uniquely identify each tuple in the same table.
- b) To store atomic values only.
- c) To establish a link between two tables and enforce referential integrity.
- d) To define the data type of an attribute.

Q 3: Which constraint restricts an attribute's values to a predefined set or type?

- a) Entity integrity
- b) Domain constraint
- c) Referential integrity
- d) Key constraint

Q 4: If a student's record is deleted from the Student table, and the corresponding enrollments in the Enrollment table are also automatically deleted, which feature is being used?

- a) Domain constraint
- b) ON DELETE CASCADE
- c) Entity integrity
- d) Unique key constraint

Q 5: Which of the following is TRUE about primary keys and candidate keys?

- a) A table can have multiple primary keys.
- b) Candidate keys are potential primary keys; only one is chosen as primary.
- c) Candidate keys can contain NULL values.
- d) Primary keys are always foreign keys.

Explanation of MCQs

Q1 – b) Every tuple must have a unique primary key value.

Explanation:

Entity integrity ensures that each tuple in a relation is uniquely identifiable by its primary key. Primary key attributes cannot be NULL and must be unique.

Q2 – c) To establish a link between two tables and enforce referential integrity.

Explanation:

A foreign key in a child table references a primary key in a parent table, ensuring that all references are valid and there are no “dangling references.”

Q3 – b) Domain constraint

Explanation:

Domain constraints enforce that the values of an attribute are valid, atomic, and consistent by restricting them to a specified domain or range.

Q4 – b) ON DELETE CASCADE

Explanation:

The ON DELETE CASCADE action ensures that related child table records are automatically deleted when a parent table record is removed, maintaining referential integrity.

Q5 – b) Candidate keys are potential primary keys; only one is chosen as primary.

Explanation:

Candidate keys are all possible keys that can uniquely identify tuples. A primary key is selected from among them, while other candidate keys can be alternate keys.

B. Short Answer Qs (100 Words)

1. What are relational constraints?
2. What is domain constraint?
3. Explain entity integrity constraint.
4. What is referential integrity?
5. Why are constraints important in relational databases?

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the different types of relational constraints.
2. Describe domain constraints with suitable examples.
3. Explain entity integrity and its importance in relational databases.
4. Discuss referential integrity constraints with suitable examples.
5. Explain how constraints maintain data accuracy and consistency.

NOTE**D. Long Answer Type Questions (300 Words)**

1. Explain relational constraints in detail and discuss their role in maintaining database integrity.
2. Discuss domain, entity integrity, and referential integrity constraints with suitable examples.
3. Explain how relational constraints prevent invalid and inconsistent data from entering a database.
4. Analyze the importance of integrity constraints in designing reliable relational databases.
5. Compare different types of relational constraints and explain situations in which each is applied.

UNIT 11

KEYS IN RELATIONAL MODEL

NOTE

UNIT STRUCTURE

- 11.1 Introduction
- 11.2 Objective
- 11.3 Primary Key
- 11.4 Foreign Key
- 11.5 Candidate Key
- 11.6 Alternate Key
- 11.7 Composite Key
- 11.8 Super Key
- 11.9 Summary
- 11.10 Glossary
- 11.11 Answer to Check Your Progress
- 11.12 Suggested Readings – OER
- 11.13 Reference
- 11.14 Exercise

11.1 INTRODUCTION

Keys are essential components of relational database design because they provide mechanisms for uniquely identifying records and establishing relationships among tables. A properly selected key helps prevent duplicate records, supports data integrity, and enables relationships between related tables. Consequently, understanding keys is fundamental for anyone working with relational databases. This unit introduces the concept of keys in the relational model and explains their importance in database design. Students will study different types of keys, including super keys, candidate keys, primary keys, alternate keys, composite keys, and foreign keys. Each type serves a specific purpose in identifying records or establishing relationships between tables. A primary key provides a unique identification mechanism for records within a table, while a foreign key establishes a connection between records in different tables. Candidate keys represent possible choices for primary key selection, whereas alternate keys are candidate keys that are not selected as the primary key. Composite keys use more than one attribute to uniquely identify a record. The unit also discusses how keys contribute to entity integrity and referential integrity. Proper key selection is important for avoiding ambiguity and maintaining consistent relationships between tables. By studying keys, learners will develop the knowledge required to design well-structured relational

NOTE

databases and understand how different database tables can be connected effectively.

11.2 OBJECTIVES

After studying this unit, you will be able to:

1. Define super keys, candidate keys, primary keys, alternate keys, and foreign keys.
2. Explain the purpose and characteristics of different types of keys.
3. Differentiate between primary keys, candidate keys, and foreign keys.
4. Analyze the role of keys in maintaining relationships and database integrity.
5. Select appropriate keys for designing relational database tables.

11.3 PRIMARY KEY

In the relational data model, the primary key is one of the most important types of keys. It plays a central role in ensuring data uniqueness, entity integrity, and efficient access to records in a table.

1. Definition of Primary Key

A primary key (PK) is an attribute or a set of attributes in a relation that uniquely identifies each tuple (row) in the table. No two tuples can have the same primary key value, and it cannot contain NULL values.

Formally, if R is a relation and PK is its primary key:

$$\forall t_1, t_2 \in R, t_1[PK] \neq t_2[PK] \text{ for } t_1 \neq t_2$$

This ensures uniqueness across all tuples in the relation.

2. Characteristics of a Primary Key

1. **Uniqueness:**
 - Every value of the primary key must be unique across all tuples.
 - This prevents duplicate records in the table.
2. **Non-nullability:**
 - Primary key attributes cannot have NULL values because a NULL would violate uniqueness.
3. **Minimality:**

NOTE

- A primary key contains only the necessary attributes required to uniquely identify a tuple.
- Extra attributes that do not contribute to uniqueness are excluded.

4. Immutable (Preferred):

- Ideally, primary key values should not change frequently because they may be referenced by foreign keys in other tables.

**Figure 11.1: Primary Key****3. Examples of Primary Key****Table 11.1: Example 1 – Student Table**

StudentID	Name	Age	Course
101	Ankur	21	B.Sc CS
102	Riya	22	BCA
103	Rohit	20	B.Sc DS

- Here, StudentID is the primary key because it uniquely identifies each student.

NOTE**Example 2 – SQL Implementation:**

```
CREATE TABLE Student (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(50) NOT NULL,  
    Age INT,  
    Course VARCHAR(50));
```

- The PRIMARY KEY constraint ensures uniqueness and non-nullability automatically.

4. Importance of Primary Key

- **Ensures Entity Integrity:** Guarantees that each tuple is uniquely identifiable.
- **Supports Relationships:** Primary keys are referenced by foreign keys in other tables to enforce referential integrity.
- **Prevents Data Duplication:** No two tuples can have the same primary key value.
- **Enables Efficient Querying:** Indexes are often created automatically on primary keys for faster data retrieval.
- **Foundation for Normalization:** Primary keys are essential for designing well-structured, normalized databases.
- The primary key is the main identifier of tuples in a relation.
- It cannot be NULL and must be unique for every row.
- Primary keys are central to entity integrity, relational operations, and database design.
- Choosing an appropriate primary key is critical for database efficiency, consistency, and reliability.

11.4 FOREIGN KEY

In the relational data model, a foreign key (FK) is a crucial concept that establishes relationships between tables and ensures referential integrity. It allows one table (child) to reference the primary key of another table (parent), enabling relational databases to model real-world associations between entities.



Figure 11.2: Foreign Key

1. Definition of Foreign Key

A foreign key is an attribute (or a set of attributes) in one relation that refers to the primary key in another relation. The foreign key ensures that every value in the referencing attribute matches a value in the referenced primary key or is NULL (if allowed).

Formally, if R1 is the child table with foreign key FK and R2 is the parent table with primary key PK:

$$\forall t \in R1, t[FK] \in \pi_{PK}(R2) \vee t[FK] = \text{NULL}$$

This ensures referential integrity, preventing invalid or “dangling” references.

2. Characteristics of a Foreign Key

1. Referential Integrity Enforcement:

- Ensures that foreign key values correspond to existing primary key values in the parent table.

2. Can Accept NULLs (Optional):

- A foreign key may allow NULLs if the relationship is optional, indicating that a child tuple does not currently reference any parent tuple.

NOTE**3. Establishes Parent-Child Relationships:**

- Parent Table: Contains the primary key.
- Child Table: Contains the foreign key referencing the parent.

4. Cascading Actions (Optional):

- **ON DELETE CASCADE:** Automatically deletes child tuples when the parent tuple is deleted.
- **ON UPDATE CASCADE:** Automatically updates child tuples when the parent primary key changes.

3. Examples of Foreign Key**Example 1 – Student and Enrollment Tables:****Table 11.2: Parent Table: Student**

StudentID	Name	Age	Course
101	Ankur	21	B.Sc CS
102	Riya	22	BCA

Table 11.3: Child Table: Enrollment

EnrollmentID	StudentID	Subject
1	101	Database
2	102	Algorithms

- StudentID in Enrollment is a foreign key referencing StudentID in student.
- Referential integrity ensures that every Enrollment.StudentID corresponds to an existing Student.StudentID.

Example 2 – SQL Implementation:

```
CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    Subject VARCHAR(50),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

- FOREIGN KEY enforces referential integrity.
- Cascading rules maintain consistency when parent records change.

4. Importance of Foreign Keys

- **Maintains Referential Integrity:** Prevents child records from referencing non-existent parent records.
 - **Supports Relational Design:** Models real-world relationships like one-to-many or many-to-many.
 - **Prevents Data Inconsistency:** Ensures that updates or deletions in parent tables are properly reflected in child tables.
 - **Enables Complex Queries:** Foreign key relationships allow joins and aggregations across tables.
- A foreign key is an attribute in a child table that references a primary key in a parent table.
 - It may allow NULLs if the relationship is optional.
 - Foreign keys are central to referential integrity, relational consistency, and relational queries.
 - Cascading actions like ON DELETE CASCADE and ON UPDATE CASCADE help maintain automatic consistency in relational databases.

11.5 CANDIDATE KEY

In the relational data model, a candidate key is an important concept that identifies all possible attributes or combinations of attributes in a relation that can uniquely identify each tuple (row). It serves as the foundation for choosing a primary key and ensures the uniqueness and integrity of data.



NOTE**Figure 11.3: Candidate Key****1. Definition of Candidate Key**

A candidate key (CK) is an attribute, or a minimal set of attributes, in a relation such that:

1. Uniqueness: Each tuple in the relation can be uniquely identified by the candidate key.
2. Minimality: No subset of the candidate key can uniquely identify a tuple; it contains no unnecessary attributes.

- Formally, for a relation R and candidate key CK:

$$\forall t_1, t_2 \in R, t_1[CK] = t_2[CK] \Rightarrow t_1 = t_2$$

- Multiple candidate keys can exist in a relation; one of them is chosen as the primary key, and the rest are alternate keys.

2. Characteristics of Candidate Keys**1. Uniqueness:**

- No two tuples have the same value for a candidate key.

2. Minimality:

- Only essential attributes are included; removing any attribute would violate uniqueness.

3. Potential Primary Key:

- Each candidate key is eligible to become a primary key.

4. Supports Integrity:

- Candidate keys help enforce entity integrity and support relational operations like joins.

3. Examples of Candidate Key**Table 11.4: Example 1 – Student Table**

StudentID	Email	Name	Age	Course
101	ankur@mail.com	Ankur	21	B.Sc CS
102	riya@mail.com	Riya	22	BCA
103	rohit@mail.com	Rohit	20	B.Sc DS

- **Possible candidate keys:**
 - StudentID (uniquely identifies each student)
 - Email (unique for every student)
- From these candidate keys, typically StudentID is chosen as the primary key, while Email can be an alternate key.

Example 2 – SQL Implementation:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,  -- Primary key from candidate keys
  Email VARCHAR(50) UNIQUE,   -- Alternate key
  Name VARCHAR(50),
  Age INT,
  Course VARCHAR(50));
```

- PRIMARY KEY enforces entity integrity on StudentID.
- UNIQUE enforces uniqueness on Email without making it primary.

4. Importance of Candidate Keys

- Ensures Uniqueness: Identifies all sets of attributes that can uniquely identify tuples.
- Supports Database Design: Helps choose a suitable primary key and alternate keys.
- Maintains Entity Integrity: Each candidate key is a potential enforcer of unique identification.
- Enables Efficient Querying: Candidate keys allow indexing for fast data retrieval.
- A candidate key is a minimal set of attributes that uniquely identifies each tuple in a relation.
- Multiple candidate keys can exist in a table; one is chosen as the primary key, and the rest are alternate keys.
- Candidate keys are essential for entity integrity, database normalization, and relational operations.

11.6 ALTERNATE KEY

In the relational data model, an alternate key is a type of key that comes from the set of candidate keys but is not chosen as the primary key. Alternate keys still maintain uniqueness for the tuples in a relation and can be used to enforce integrity constraints.

NOTE



Figure 11.4: Alternate Key

1. Definition of Alternate Key

An alternate key (AK) is a candidate key that is eligible to uniquely identify tuples in a relation but was not selected as the primary key.

Formally, if a relation R has candidate keys $CK1$, $CK2$, and $CK3$, and $CK1$ is chosen as the primary key, then $CK2$ and $CK3$ are alternate keys.

2. Characteristics of Alternate Keys

1. Derived from Candidate Keys:

- Every alternate key is a candidate key, but only candidate keys not selected as primary keys are considered alternate keys.

2. Uniqueness:

- Alternate keys must also uniquely identify each tuple in the relation.

3. Non-nullability (Optional):

- While primary keys cannot be NULL, alternate keys may allow NULLs, depending on the database design.

4. Supports Database Integrity:

- Alternate keys enforce uniqueness constraints on attributes other than the primary key.

3. Examples of Alternate Key

Table 11.5: Example 1 – Student Table:

StudentID	Email	Name	Age	Course
101	ankur@mail.com	Ankur	21	B.Sc CS
102	riya@mail.com	Riya	22	BCA
103	rohit@mail.com	Rohit	20	B.Sc DS

- **Candidate Keys:** StudentID, Email
- **Primary Key:** StudentID
- **Alternate Key:** Email

SQL Implementation:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,      -- Primary key
  Email VARCHAR(50) UNIQUE,      -- Alternate key
  Name VARCHAR(50),
  Age INT,
  Course VARCHAR(50));
```

Email ensures uniqueness, even though it is not the primary key.

4. Importance of Alternate Keys

- **Maintains Data Uniqueness:** Ensures attributes like Email or PassportNumber remain unique across tuples.
- **Supports Database Queries:** Alternate keys can be used to search, join, or index data efficiently.
- **Provides Backup Unique Identifiers:** If the primary key is unavailable or not used, alternate keys can uniquely identify tuples.
- **Essential for Data Integrity:** Helps prevent duplicate entries for non-primary attributes.
- An alternate key is a candidate key not chosen as the primary key but still uniquely identifies tuples in a relation.
- It enforces uniqueness on important attributes other than the primary key.
- Alternate keys are critical for data integrity, query optimization, and redundancy prevention.

NOTE

11.7 COMPOSITE KEY

In the relational data model, a composite key is a type of key that consists of two or more attributes that together uniquely identify a tuple (row) in a relation. Composite keys are used when a single attribute is not sufficient to ensure uniqueness across all tuples.



Figure 11.5: Composite Key

1. Definition of Composite Key

A composite key (CK) is a combination of two or more attributes in a relation that collectively act as a primary key.

- Formally, if a relation R has attributes A1, A2, ..., An, a composite key CK is:

$$CK = \{A1, A2, \dots, Ak\} \subseteq \{A1, A2, \dots, An\}$$

- The set of attributes together uniquely identifies each tuple, while individually they may not.

2. Characteristics of a Composite Key

1. Uniqueness:

- The combination of all attributes in the composite key uniquely identifies tuples.

2. Multiple Attributes:

- Consists of two or more attributes; no single attribute alone is sufficient.

3. Supports Entity Integrity:

- Composite keys enforce uniqueness and non-nullability for all attributes involved.

4. Used in Many-to-Many Relationships:

- Often used in junction/associative tables to represent many-to-many relationships.

3. Examples of Composite Key**Table 11.6: Example 1 – Enrollment Table (Student-Course Relationship)**

StudentID	CourseID	EnrollmentDate
101	C01	2026-01-10
101	C02	2026-01-12
102	C01	2026-01-11

- Here, neither StudentID nor CourseID alone can uniquely identify a record, because a student can enroll in multiple courses and a course can have multiple students.
- **Composite Key:** {StudentID, CourseID} uniquely identifies each enrollment record.

Example 2 – SQL Implementation:

```
CREATE TABLE Enrollment (
  StudentID INT NOT NULL,
  CourseID INT NOT NULL,
  EnrollmentDate DATE,
  PRIMARY KEY (StudentID, CourseID));
```

- The PRIMARY KEY is a combination of StudentID and CourseID.
- This ensures uniqueness across the table.

4. Importance of Composite Key

- Maintains Uniqueness: Ensures each tuple in a table is uniquely identifiable when single attributes are insufficient.
- Supports Many-to-Many Relationships: Commonly used in junction tables to relate two entities.
- Enforces Integrity: All attributes in the composite key collectively support entity integrity.

NOTE

- Optimizes Queries: Allows accurate joins and lookups based on combined key attributes.
- A composite key is a primary key made up of two or more attributes.
- It ensures uniqueness across tuples when a single attribute cannot uniquely identify records.
- Composite keys are essential in relational databases, especially for tables representing associative entities or complex relationships.

11.8 SUPER KEY

In the relational data model, a superkey is a fundamental concept that represents a set of attributes capable of uniquely identifying each tuple in a relation. It is the broadest form of key, from which candidate keys, primary keys, and alternate keys are derived.

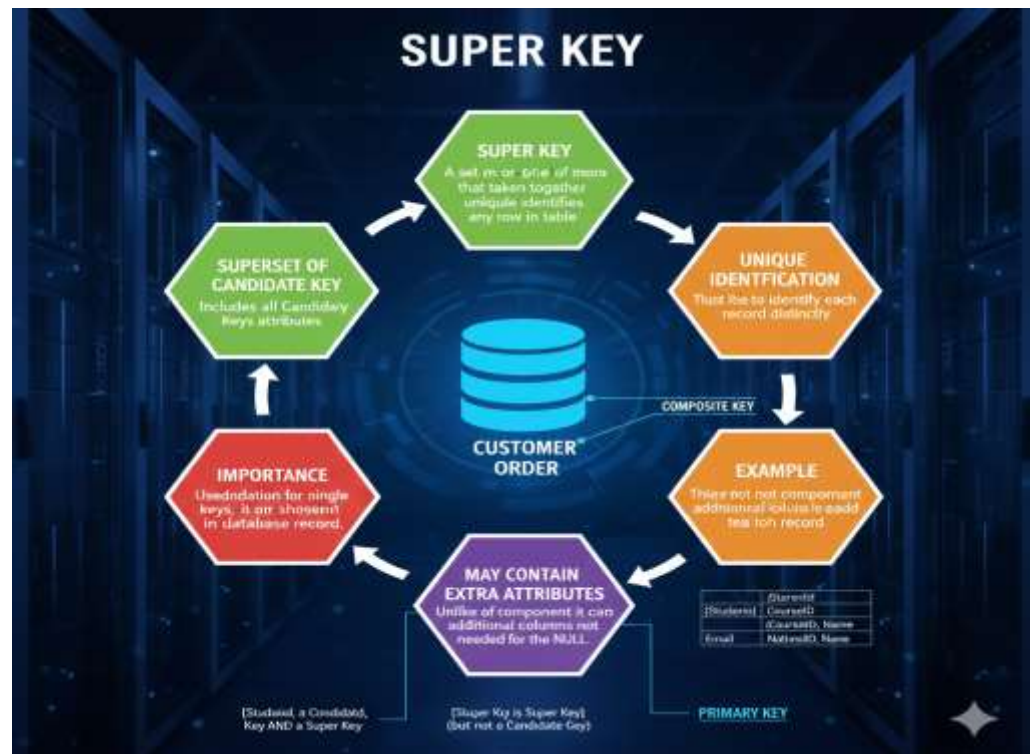


Figure 11.6: Super Key

1. Definition of Superkey

A superkey (SK) is a set of one or more attributes of a relation such that no two tuples in the relation have the same values for all attributes in the set.

- Formally, for a relation R with tuples t1 and t2:

$$t1[SK] = t2[SK] \Rightarrow t1 = t2$$

- Any set of attributes that satisfies uniqueness can be a superkey.

2. Characteristics of Superkeys

1. Uniqueness:

- Every superkey uniquely identifies tuples in the relation.

2. May Contain Extra Attributes:

- A superkey may include additional attributes that are not necessary for uniqueness.
- If a superkey has no unnecessary attributes, it is called a candidate key.

3. Superset of Candidate Key:

- Every candidate key is a minimal superkey, but not every superkey is a candidate key.

4. Supports Entity Integrity:

- Superkeys help in ensuring tuples are uniquely identifiable, supporting entity integrity.

3. Examples of Superkeys

Table 11.7: Example – Student Table:

StudentID	Email	Name	Age
101	ankur@mail.com	Ankur	21
102	riya@mail.com	Riya	22
103	rohit@mail.com	Rohit	20

- Possible superkeys:
 - {StudentID} → Candidate key and superkey
 - {Email} → Candidate key and superkey
 - {StudentID, Name} → Superkey (redundant attribute Name)
 - {StudentID, Email, Name} → Superkey (includes unnecessary attributes)
- Candidate keys are minimal superkeys with no extra attributes.

Example 2 – SQL Concept:

NOTE

- While superkeys are a theoretical concept in database design, primary key constraints enforce a minimal superkey in practical implementation.

4. Importance of Superkeys

- **Ensures Uniqueness:** Every superkey guarantees that tuples are uniquely identifiable.
- **Foundation for Other Keys:** Candidate keys, primary keys, and alternate keys are derived from superkeys.
- **Database Design:** Identifying superkeys helps in normalization and schema design.
- **Supports Integrity:** Superkeys maintain entity integrity by preventing duplicate records.
- A superkey is a set of attributes that uniquely identifies tuples in a relation.
- Superkeys may include extra, unnecessary attributes beyond what is required for uniqueness.
- Candidate keys are minimal superkeys used as primary or alternate keys.
- Understanding superkeys is essential for database design, normalization, and relational integrity.

**Activity 11.1:**

Define different types of keys with the help of a student table.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

Check Your Progress 11.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is the difference between a candidate key and a primary key in a relational database?

Question 2: Explain the purpose of a foreign key in relational database design?

11.9 SUMMARY

Unit 11 focuses on the concept of keys in relational databases, which are essential for ensuring data uniqueness, integrity, and efficient access. Keys are special attributes or sets of attributes that uniquely identify tuples (rows) in a relation and play a critical role in enforcing relational constraints and modeling relationships between tables.

The unit covers several types of keys:

1. **Superkey:** Any set of attributes that uniquely identifies tuples in a relation. Superkeys may contain extra attributes beyond what is necessary for uniqueness.
2. **Candidate Key:** A minimal superkey with no unnecessary attributes that can uniquely identify each tuple. A relation may have multiple candidate keys.
3. **Primary Key:** A candidate key chosen to uniquely identify tuples. Primary keys cannot contain NULL values and enforce entity integrity.
4. **Alternate Key:** A candidate key not selected as the primary key but still ensures uniqueness across tuples. Alternate keys can be used to enforce additional constraints.
5. **Composite Key:** A key composed of two or more attributes that together uniquely identify a tuple, often used in junction tables or many-to-many relationships.
6. **Foreign Key:** An attribute in a table that references the primary key in another table, enforcing referential integrity and supporting parent-child relationships.

NOTE**Key Takeaways:**

- Keys are fundamental for data integrity, uniqueness, and relational consistency.
- Proper selection of primary, candidate, and alternate keys is crucial for robust database design.
- Composite keys and foreign keys allow modeling of complex relationships between tables.
- Understanding superkeys provides a foundation for normalization, relational schema design, and efficient querying.

11.10 GLOSSARY

1. Key – An attribute or set of attributes in a relation used to uniquely identify tuples and enforce integrity.
2. Superkey (SK) – A set of one or more attributes that can uniquely identify each tuple in a relation. It may contain extra, unnecessary attributes.
3. Candidate Key (CK) – A minimal superkey with no redundant attributes, capable of uniquely identifying tuples. A table can have multiple candidate keys.
4. Primary Key (PK) – A candidate key chosen to uniquely identify tuples in a table. It cannot contain NULL values and enforces entity integrity.
5. Alternate Key (AK) – A candidate key not selected as the primary key but still ensures uniqueness in a relation.
6. Composite Key – A key made up of two or more attributes that together uniquely identify tuples. Often used in many-to-many relationships.
7. Foreign Key (FK) – An attribute in one table that references the primary key in another table, ensuring referential integrity between the tables.
8. Uniqueness – A property of keys that ensures no two tuples have the same key values.
9. Entity Integrity – The principle that primary key attributes cannot be NULL, guaranteeing that each tuple is uniquely identifiable.
10. Referential Integrity – The principle that foreign key values must match primary key values in the referenced table or be NULL, maintaining consistency between tables.
11. Minimal Superkey – A superkey from which no attribute can be removed without losing uniqueness, essentially the definition of a candidate key.
12. Parent Table – A table containing a primary key that is referenced by a foreign key in another table.

13. Child Table – A table containing a foreign key that references a primary key in a parent table.

11.11 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What is the difference between a candidate key and a primary key in a relational database?

Answer:

A candidate key is any minimal set of attributes that can uniquely identify tuples in a relation. A table may have multiple candidate keys. The primary key is the candidate key chosen to uniquely identify tuples in the table.

Example: In a Student table, both StudentID and Email can be candidate keys, but StudentID may be selected as the primary key, making Email an alternate key.

Q 2: Explain the purpose of a foreign key in relational database design.

Answer:

A foreign key is an attribute (or set of attributes) in one table that references the primary key of another table. Its purpose is to:

1. Establish relationships between tables (parent-child).
 2. Enforce referential integrity, ensuring that foreign key values either match existing primary key values or are NULL.
- Example:* In an Enrollment table, StudentID is a foreign key referencing StudentID in the Student table, ensuring all enrollments correspond to existing students.

11.12 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13716	

NOTE

11.13 REFERENCE

1. Elmasri, R., & Navathe, S. (2020). *Fundamentals of Database Systems*. Pearson.
2. Date, C. J. (2019). *An Introduction to Database Systems*. Pearson Education.
3. Silberschatz, A., Korth, H., & Sudarshan, S. (2021). *Database System Concepts*. McGraw-Hill.
4. Jain, S. K. (2018). *Database Management Systems*. BPB Publications.

11.14 EXERCISE

PART 1**A. Multiple Choice Qs**

Q 1: Which of the following statements about a primary key is TRUE?

- a) It can contain NULL values.
- b) It may not uniquely identify tuples.
- c) It uniquely identifies each tuple in a table.
- d) It is optional in a relational table.

Q 2: Which key is defined as a minimal superkey that can uniquely identify tuples in a relation?

- a) Superkey
- b) Candidate key
- c) Foreign key
- d) Alternate key

Q 3: In a table, which key is not chosen as the primary key but still ensures uniqueness?

- a) Primary key
- b) Alternate key
- c) Composite key
- d) Foreign key

Q 4: Which of the following is an example of a composite key?

- a) StudentID in a Student table
- b) Email in a Student table
- c) {StudentID, CourseID} in an Enrollment table
- d) DepartmentID in a Department table

Q 5: What is the purpose of a foreign key in a relational database?

- a) To allow NULL values in a table

- b) To enforce referential integrity between tables
- c) To create a composite key
- d) To identify candidate keys

Explanation of MCQs

Q1 c) It uniquely identifies each tuple in a table.

Explanation:

A primary key is a candidate key selected to uniquely identify each tuple in a table. It cannot contain NULL values and ensures entity integrity.

Q2 b) Candidate key

Explanation:

A candidate key is a minimal set of attributes that can uniquely identify tuples. All candidate keys are superkeys, but superkeys may contain extra attributes.

Q3 b) Alternate key

Explanation:

An alternate key is a candidate key that was not selected as the primary key, yet it maintains uniqueness for its attribute(s).

Q4 c) {StudentID, CourseID} in an Enrollment table

Explanation:

A composite key is a combination of two or more attributes that together uniquely identify a tuple. In the Enrollment table, neither StudentID nor CourseID alone is unique.

Q5 b) To enforce referential integrity between tables

Explanation:

A foreign key is an attribute in a child table that references the primary key of a parent table, ensuring that every foreign key value matches a primary key value or is NULL, maintaining referential integrity.

B. Short Answer Qs (100 Words)

1. What is a key in a relational database?
2. What is a primary key?
3. Explain the concept of a candidate key.
4. What is a foreign key?
5. Differentiate between primary key and foreign key.

NOTE**PART 2****C. Long Answer Type Questions (200 Words)**

1. Explain the different types of keys used in relational databases.
2. Differentiate between super key, candidate key, and primary key.
3. Explain the role of foreign keys in establishing relationships between tables.
4. Discuss the importance of keys in relational database design.
5. Explain composite keys with suitable examples.

PART 3**D. Long Answer Type Questions (300 Words)**

1. Explain different types of keys in the relational model and discuss their significance.
2. Compare super keys, candidate keys, primary keys, alternate keys, and foreign keys with suitable examples.
3. Explain how primary and foreign keys help maintain entity integrity and referential integrity.
4. Discuss the process of selecting appropriate keys while designing relational tables.
5. Analyze the consequences of poor key selection on database integrity, performance, and data management.



DR. C.V. RAMAN UNIVERSITY

Chhattisgarh, Bilaspur AN AICTE GROUP UNIVERSITY

Approved by: PCI / AICTE / NCTE / BCI / Member of: AUC / Joint Committee: DGC / DEB / AICTE / Recognized by: BGC / A NAAC Accredited University



DR. C. V. RAMAN UNIVERSITY (C.G.)

SLM FEEDBACK FORM

PROGRAMME : _____

COURSE : _____

BLOCK : _____

UNIT : _____

Dear Student,

While studying the units of this block, you may have found certain portions of the text difficult to comprehend. We wish to know your difficulties and suggestions, in order to improve the course. Therefore, we request you to fill up and send us the following questionnaire, which pertains to this block. If you find the space provided insufficient, kindly use a separate sheet.

Please mail to aisectcontent@aisect.org / for online submission please scan QR code:

Questionnaire

1. Name : _____

2. Enrolment No. :

--	--	--	--	--	--	--	--	--	--

3. How many hours did you need for studying the units? :

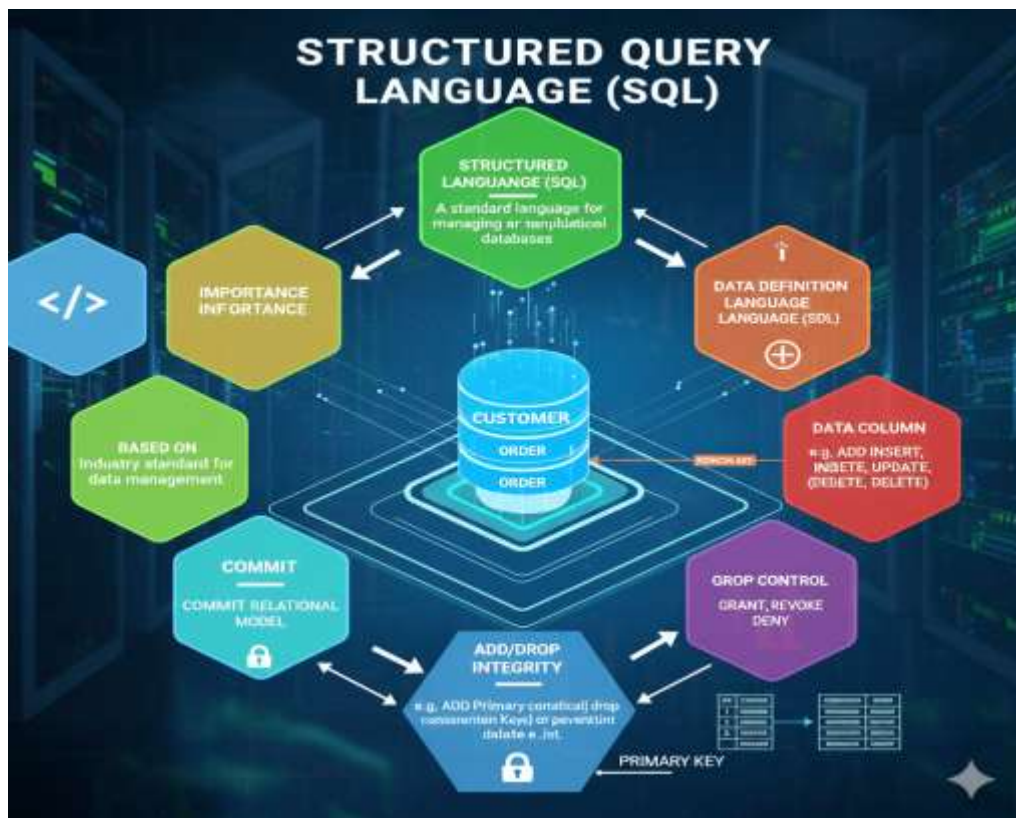
--

4. Please give your reactions to the following items based on your reading of the block:

Items	Excellent	Very Good	Good	Poor	Give specific Examples (If poor)
Presentation Quality					
Language and Style					
Illustrations					
Used (diagrams, tables, etc.)					
Conceptual Clarity					
Check Your Progress Questions					
Suggested Readings - OER					

5. Any other comments:

BLOCK V	STRUCTURED QUERY LANGUAGE (SQL)	227-284
Unit 12	Introduction to SQL and DDL	229
Unit 13	Table Operations using DDL	247
Unit 14	Data Manipulation Language (DML)	263



Structured Query Language (SQL) is the standard language for managing and manipulating relational databases. It allows users to define, access, update, and control the data stored in a relational database system. SQL provides a declarative approach, meaning users specify what they want from the database without detailing how to perform the operations. The language is divided into several categories: Data Definition Language (DDL) for creating and modifying database structures, Data Manipulation Language (DML) for inserting, updating, and deleting data, Data Query Language (DQL) for retrieving information using queries, and Data Control Language (DCL) for managing permissions and access. SQL supports powerful features such as joins, subqueries, grouping, sorting, and aggregate functions, enabling complex data analysis and reporting. With its standardization across platforms, SQL ensures that database operations are consistent, efficient, and secure, making it a cornerstone of modern database management and application development.

UNIT 12

INTRODUCTION TO SQL AND DDL

NOTE

UNIT STRUCTURE

- 12.1 Introduction
- 12.2 Objective
- 12.3 Introduction to SQL
- 12.4 Data Definition Language (DDL)
- 12.5 CREATE DATABASE
- 12.6 DROP DATABASE
- 12.7 Summary
- 12.8 Glossary
- 12.9 Answer to Check Your Progress
- 12.10 Suggested Readings – OER
- 12.11 Reference
- 12.12 Exercise

12.1 INTRODUCTION

Structured Query Language, commonly known as SQL, is a standard language used to interact with relational database systems. It provides commands for defining database structures, manipulating data, controlling access, and retrieving information. Knowledge of SQL is therefore essential for database developers, administrators, and application programmers. This unit provides an introduction to SQL and focuses particularly on Data Definition Language, or DDL. DDL consists of commands used to define and modify the structure of database objects such as tables. Students will be introduced to important SQL concepts and the major categories of SQL commands. The unit explains the purpose of commands such as CREATE, ALTER, DROP, and TRUNCATE and describes their role in database structure management. CREATE is used to define new database objects, while ALTER can be used to modify existing structures. DROP removes database objects, whereas TRUNCATE removes records from a table while retaining its structure. The unit also introduces the distinction between DDL and other categories of SQL commands, particularly Data Manipulation Language. By studying SQL and DDL, learners will gain the foundation required to create and manage database structures. This knowledge prepares students for practical database development and for performing table operations in subsequent units.

NOTE

12.2 OBJECTIVES

After studying this unit, you will be able to:

1. Define SQL and Data Definition Language.
2. Explain the major categories of SQL commands.
3. Differentiate between DDL and DML commands.
4. Analyze the role of DDL commands in defining database structures.
5. Evaluate the appropriate use of SQL commands for database definition tasks.

12.3 INTRODUCTION TO SQL

Structured Query Language (SQL) is the standardized language used to interact with relational databases. It was first developed in the 1970s by IBM researchers for querying and managing relational data, and it has since become the universal language for relational database management systems (RDBMS) such as MySQL, Oracle, SQL Server, PostgreSQL, and SQLite. SQL provides a declarative approach, meaning that users specify *what* they want from the database rather than *how* to retrieve it, allowing the database engine to determine the most efficient execution method.



Figure 12.1: Introduction to SQL**NOTE****Key Features of SQL**

1. **Data Definition:** SQL allows the creation and modification of database structures, including tables, views, indexes, and schemas.
2. **Data Manipulation:** SQL provides commands to insert, update, delete, and retrieve data from tables.
3. **Data Querying:** SQL supports complex queries using SELECT statements, filtering (WHERE), sorting (ORDER BY), grouping (GROUP BY), and joining multiple tables (JOIN).
4. **Data Control:** SQL manages user permissions and access rights, ensuring that data is secure.
5. **Transaction Management:** SQL ensures data consistency and integrity through transactions, supporting COMMIT and ROLLBACK operations.

Advantages of SQL

- **Platform-Independent:** Standardized and supported by most RDBMS systems.
- **Powerful Data Retrieval:** Can handle large volumes of data efficiently.
- **Ease of Learning:** SQL syntax is English-like, making it accessible for beginners.
- **Data Integrity and Security:** Supports constraints, keys, and permissions.
- **Flexible and Scalable:** Suitable for both small and enterprise-level databases.

Categories of SQL Commands

SQL commands are grouped into the following categories:

1. **Data Definition Language (DDL)**

Commands like CREATE, ALTER, DROP for defining and modifying database objects.

2. **Data Manipulation Language (DML)**

Commands like INSERT, UPDATE, DELETE for manipulating data.

3. **Data Query Language (DQL)**

SELECT statements for querying data.

NOTE**4. Data Control Language (DCL)**

Commands like GRANT and REVOKE for controlling access.

5. Transaction Control Language (TCL)

Commands like COMMIT, ROLLBACK, and SAVEPOINT for managing transactions.

SQL serves as the core interface for relational databases, enabling users to define, manipulate, query, and secure data. Its declarative nature, combined with powerful querying capabilities, makes it an indispensable tool for database administrators, developers, and data analysts. A strong understanding of SQL forms the foundation for database design, management, and application development.

12.4 DATA DEFINITION LANGUAGE (DDL)

Data Definition Language (DDL) is a subset of SQL used for defining, creating, modifying, and deleting database structures such as tables, views, indexes, and schemas. Unlike commands that manipulate data (DML), DDL focuses on the database schema and structure. DDL commands are essential for setting up a database, enforcing data integrity, and ensuring that the database is organized for efficient storage and retrieval.

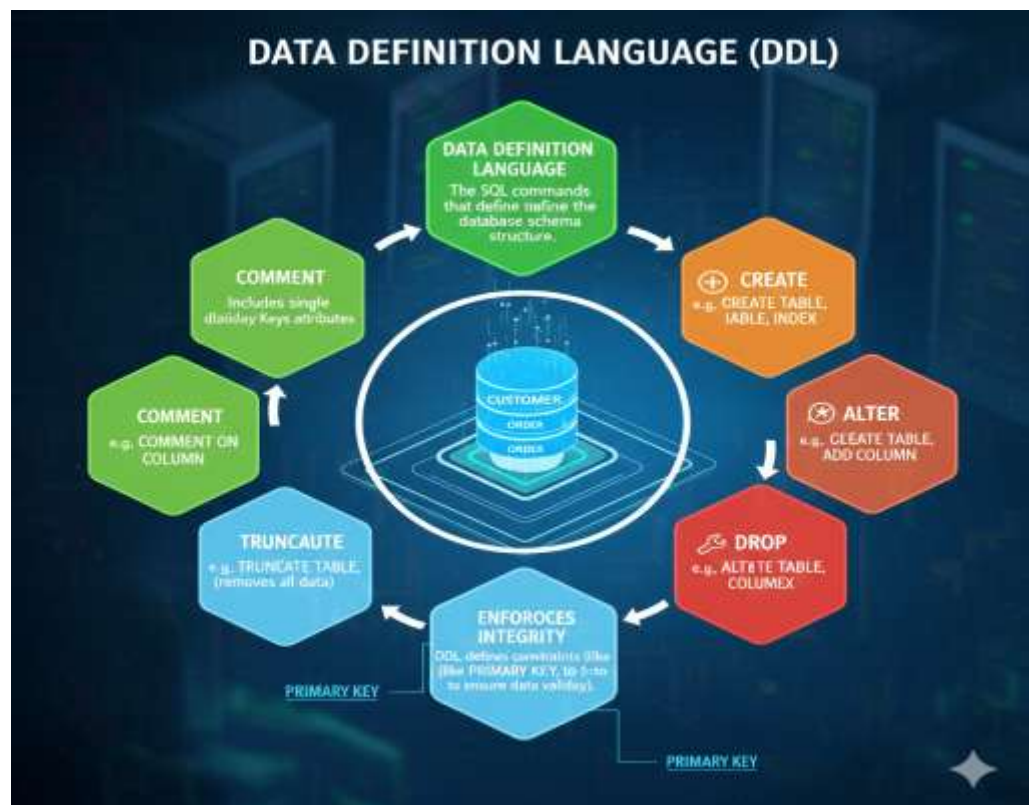


Figure 12.2: Data Definition Language (DDL)

Key Features of DDL

1. **Schema Definition:** DDL allows the creation of new tables, databases, and other objects.
2. **Structure Modification:** Existing database objects can be altered to add, remove, or modify attributes.
3. **Constraint Enforcement:** DDL supports primary keys, foreign keys, unique constraints, not null constraints, and check constraints to ensure data integrity.
4. **Object Deletion:** Tables, views, and other database objects can be removed using DDL commands.
5. **Automatic Commit:** DDL commands are usually auto-committed, meaning changes are permanent and cannot be rolled back unless explicitly using transaction features in some RDBMS.

Common DDL Commands**1. CREATE:**

- Used to create new database objects such as databases, tables, views, and indexes.
- **Syntax Example – Creating a Table:**

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(50) NOT NULL,
  Age INT,
  Course VARCHAR(50));
```

- This command creates a Student table with primary key and not null constraints.

2. ALTER:

- Used to modify an existing database object, such as adding, dropping, or modifying columns and constraints.
- **Syntax Example – Adding a Column:**

```
ALTER TABLE Student
ADD Email VARCHAR(50) UNIQUE;
```

- **Syntax Example – Modifying a Column:**

```
ALTER TABLE Student
MODIFY Age INT NOT NULL;
```

3. DROP:

NOTE

- Used to delete an existing database object permanently.

- **Syntax Example – Dropping a Table:**

DROP TABLE Student;

- Warning: All data in the table will be permanently deleted.

4. TRUNCATE:

- Removes all records from a table but retains the table structure for future use.

- **Syntax Example:**

TRUNCATE TABLE Student;

- Faster than DELETE because it does not log individual row deletions.

5. COMMENT (Optional in Some RDBMS):

- Adds descriptions or notes to tables or columns for documentation.

- **Example:**

COMMENT ON COLUMN Student.Age IS 'Age of the student in years';

Importance of DDL

- **Defines Database Structure:** Establishes the foundation for data storage and organization.
- **Supports Data Integrity:** Allows defining keys and constraints to prevent invalid data.
- **Enables Schema Evolution:** Tables and other objects can be modified as requirements change.
- **Efficient Management:** Provides a clear method for creating and managing database objects systematically.

DDL commands are the backbone of database design and management. They enable administrators and developers to create, modify, and delete database structures, enforce constraints, and ensure that the database is organized, consistent, and reliable. Mastery of DDL is essential for anyone working with relational databases and forms the foundation for learning data manipulation (DML) and queries (DQL) in SQL.

12.5 CREATE DATABASE

The CREATE DATABASE command in SQL is a DDL (Data Definition Language) statement used to create a new database in a relational

NOTE

database management system (RDBMS). This command is usually the first step in database creation, providing a container for all tables, views, indexes, stored procedures, and other database objects.

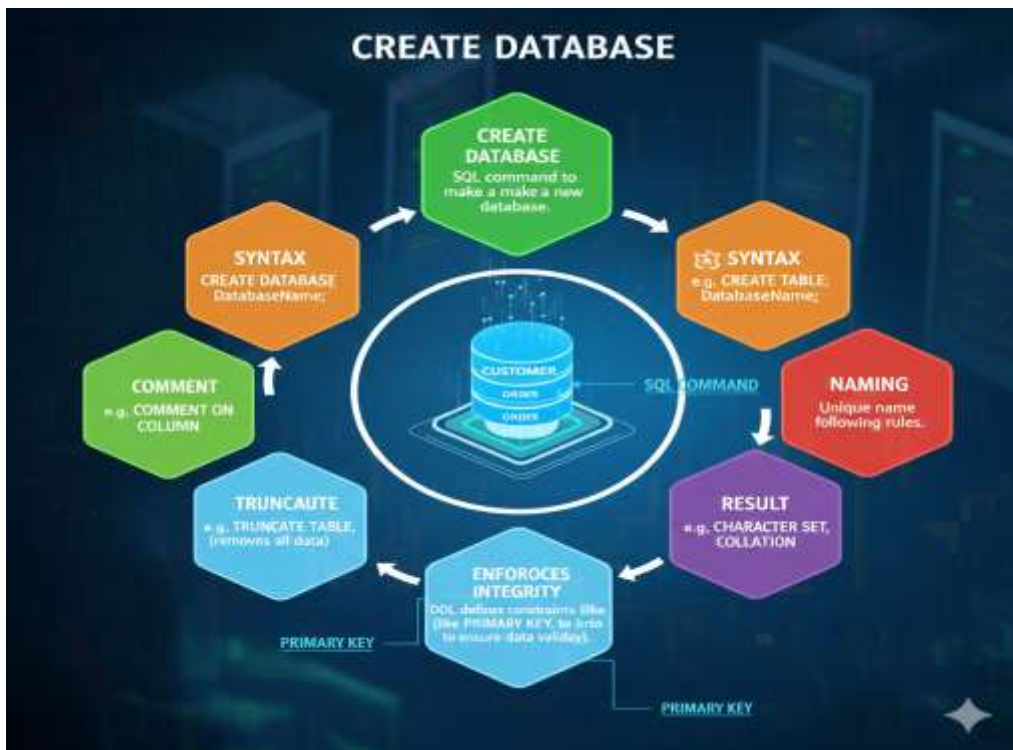


Figure 12.3: Create Database

Purpose of CREATE DATABASE

- To initialize a new database environment for storing related data.
- To define the database name and optionally specify storage parameters, character sets, and collations depending on the RDBMS.
- To provide a separate namespace to avoid conflicts with other databases on the same server.

Syntax

Basic SQL syntax for creating a database:

```
CREATE DATABASE database_name;
```

Optional advanced syntax with parameters (supported in many RDBMS like MySQL, SQL Server, Oracle):

STRUCTURED QUERY LANGUAGE (SQL)

NOTE

```
CREATE DATABASE database_name  
[CHARACTER SET charset_name]  
[COLLATE collation_name];
```

- CHARACTER SET specifies the character encoding for text data.
- COLLATE defines rules for string comparison, sorting, and case sensitivity.

Examples

Example 1 – Simple Database Creation:

```
CREATE DATABASE SchoolDB;
```

- Creates a new database named SchoolDB with default settings.

Example 2 – Database with Character Set and Collation (MySQL):

```
CREATE DATABASE SchoolDB  
CHARACTER SET utf8mb4  
COLLATE utf8mb4_general_ci;
```

- Creates SchoolDB using UTF-8 encoding and default case-insensitive sorting.

Example 3 – Database in SQL Server:

```
CREATE DATABASE SchoolDB  
ON PRIMARY (  
    NAME = 'SchoolDB_data',  
    FILENAME = 'C:\SQLData\SchoolDB.mdf',  
    SIZE = 10MB,  
    MAXSIZE = 100MB,  
    FILEGROWTH = 5MB)  
LOG ON (  
    NAME = 'SchoolDB_log',  
    FILENAME = 'C:\SQLData\SchoolDB_log.ldf',  
    SIZE = 5MB,  
    MAXSIZE = 50MB,  
    FILEGROWTH = 5MB);
```

- This example specifies data and log file paths, initial sizes, maximum sizes, and growth increments.

Important Points

NOTE

1. Unique Name: Each database must have a unique name on the server.
2. Privileges: Only users with CREATE DATABASE privileges can execute this command.
3. RDBMS Specific Features: Parameters like CHARACTER SET, COLLATE, and file storage options vary across RDBMS.
4. Default Settings: If optional parameters are omitted, the database is created with default character set, collation, and storage parameters.

Steps After CREATE DATABASE

1. Select the database for use:

USE SchoolDB;

- Ensures subsequent operations like creating tables are applied to the new database.
2. Create tables, indexes, and constraints using DDL commands.
 3. Grant privileges to users if multiple users will access the database.

The CREATE DATABASE command is the foundation of any relational database system, providing a container for all related data and objects. It allows developers and administrators to set up, configure, and organize databases before defining tables, relationships, and constraints. Mastery of this command is essential for database setup, management, and deployment.

12.6 DROP DATABASE

The DROP DATABASE command in SQL is a DDL (Data Definition Language) statement used to permanently delete an existing database along with all its contained objects, such as tables, views, indexes, stored procedures, and data. This command is powerful and irreversible, so it must be used with caution.

NOTE

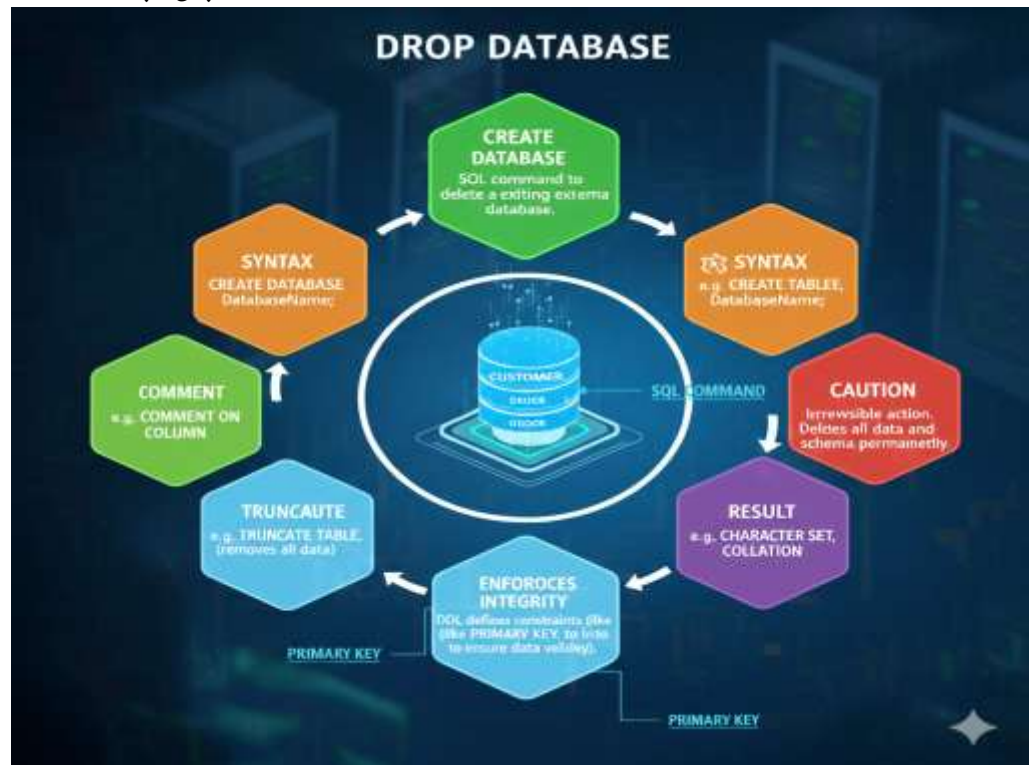


Figure 12.4: Drop Database

Purpose of DROP DATABASE

- To remove an entire database from the RDBMS server.
- To free up storage space occupied by the database.
- To permanently delete data and structures that are no longer needed.
- To reset or clean up a development or testing environment.

Syntax

Basic SQL syntax:

DROP DATABASE database_name;

Optional syntax (some RDBMS support conditional deletion):

DROP DATABASE IF EXISTS database_name;

- IF EXISTS ensures the command only executes if the database exists, preventing errors if the database does not exist.

Examples

Example 1 – Simple Database Deletion:

DROP DATABASE SchoolDB;

- Deletes the SchoolDB database along with all its tables, data, and other objects.

Example 2 – Conditional Deletion (MySQL / PostgreSQL):

DROP DATABASE IF EXISTS SchoolDB;

- Only deletes SchoolDB if it exists; otherwise, the command does nothing.

Example 3 – SQL Server Example:

USE master;

DROP DATABASE SchoolDB;

- Switches to the master database first because you cannot drop a database that is currently in use.

Important Points

1. Irreversible Operation: Once executed, all data and objects in the database are permanently deleted. Use with caution.
2. Permissions Required: Only users with DROP DATABASE privileges can execute this command.
3. Database in Use: A database cannot be dropped if it is currently in use. Switch to another database (e.g., master) first.
4. RDBMS Specifics: Some systems allow options like CASCADE to remove dependent objects automatically, while others require manual deletion.
5. No Undo: Unlike DELETE or TRUNCATE, DROP DATABASE cannot be rolled back in most RDBMS.

Steps Before Dropping a Database

1. Backup the database if data may be needed later.
2. Ensure no users are connected to the database.
3. Switch to another database to avoid “database in use” errors.
4. Execute the DROP DATABASE command carefully.

The DROP DATABASE command is a powerful SQL statement that completely removes a database and all its contents. It is primarily used for cleanup, maintenance, or reinitialization of databases. Proper privileges, caution, and backups are essential before using this command to prevent unintended data loss.

NOTE**Activity 12.1:**

How to use create and drop of database? Explain with the help of a table.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

Check Your Progress 12.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is the difference between DROP DATABASE and TRUNCATE TABLE in SQL?

Question 2: Explain the purpose of DDL (Data Definition Language) in SQL.

12.7 SUMMARY

Unit 12 introduces students to Structured Query Language (SQL), the standard language used for relational database management systems (RDBMS). The unit emphasizes the declarative nature of SQL, allowing users to specify *what* they want from the database without detailing *how* the operations are executed.

A key focus of the unit is Data Definition Language (DDL), a subset of SQL used to define and manage the structure of databases and database objects. Students learn how to create, modify, and delete databases and tables, and how to enforce integrity constraints such as primary keys, foreign keys, unique constraints, and not null constraints.

The unit also covers essential DDL commands, including:

- **CREATE DATABASE** – To create a new database.
- **DROP DATABASE** – To permanently remove an existing database and all its objects.

- **CREATE TABLE** – To define new tables and their columns with appropriate data types.
- **ALTER TABLE** – To modify table structures by adding, deleting, or changing columns and constraints.
- **DROP TABLE** – To delete a table and all its data.
- **TRUNCATE TABLE** – To remove all data from a table while retaining its structure.

Key Takeaways:

- SQL is the core language for interacting with relational databases.
- DDL commands are critical for database setup, schema definition, and enforcing data integrity.
- Proper understanding of SQL and DDL ensures efficient database design, management, and secure storage of data.
- Mastery of CREATE and DROP commands is essential for database lifecycle management, including initialization and cleanup.

In essence, Unit 12 equips learners with the foundational skills to define, organize, and manage databases, which is essential before moving on to data manipulation and querying using DML and DQL commands in SQL.

12.8 GLOSSARY

- **SQL (Structured Query Language)** – A standard language used to create, manipulate, query, and manage relational databases.
- **DDL (Data Definition Language)** – A subset of SQL commands used to define and manage database structures, such as databases, tables, and constraints.
- **CREATE DATABASE** – A DDL command used to create a new database in the RDBMS.
- **DROP DATABASE** – A DDL command used to permanently delete an existing database along with all its objects and data.
- **CREATE TABLE** – A DDL command used to define a new table with columns, data types, and constraints.
- **ALTER TABLE** – A DDL command used to modify an existing table, such as adding, modifying, or dropping columns and constraints.
- **DROP TABLE** – A DDL command used to delete a table permanently along with all its data.
- **TRUNCATE TABLE** – A DDL command used to remove all rows from a table while keeping its structure intact.

NOTE

- **Constraints** – Rules defined on table columns to ensure data integrity, including PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK constraints.
- **Primary Key** – A column or set of columns in a table that uniquely identifies each row and cannot contain NULL values.
- **Foreign Key** – A column or set of columns in a table that references the primary key of another table, enforcing referential integrity.
- **Entity Integrity** – Ensures that primary key values cannot be NULL, guaranteeing that each row is uniquely identifiable.

12.9 ANSWER TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What is the difference between DROP DATABASE and TRUNCATE TABLE in SQL?

Answer:

- **DROP DATABASE:** Permanently deletes an entire database along with all its tables, data, and objects. Once executed, it cannot be undone.
- **TRUNCATE TABLE:** Removes all rows from a table but keeps the table structure intact for future use. It is faster than DELETE but does not remove the table itself.

Q 2: Explain the purpose of DDL (Data Definition Language) in SQL.

Answer:

DDL is a subset of SQL used to define and manage the structure of databases and database objects. It includes commands such as CREATE, ALTER, DROP, and TRUNCATE. DDL allows users to create databases and tables, modify their structure, define constraints like primary and foreign keys, and maintain data integrity.

12.10 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13717	

12.11 REFERENCE

- Elmasri, R., & Navathe, S. (2019). *Fundamentals of Database Systems* (7th ed.). Pearson.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill.
- Date, C. J. (2015). *An Introduction to Database Systems*. Pearson India.
- Ullman, J. D. (2016). *Principles of Database Systems*. Galgotia.
- NPTEL DBMS Course Materials (India).

12.12 EXERCISE

PART 1

A. Multiple Choice Qs

Q 1: Which SQL command is used to create a new database?

- a) DROP DATABASE
- b) CREATE DATABASE
- c) CREATE TABLE
- d) ALTER DATABASE

Q 2: Which SQL command removes all rows from a table but keeps the table structure intact?

- a) DROP TABLE
- b) DELETE
- c) TRUNCATE TABLE
- d) ALTER TABLE

Q 3: Which of the following is a DDL command?

- a) INSERT
- b) UPDATE
- c) CREATE TABLE
- d) SELECT

Q 4: What happens when you execute the following command:

INTRODUCTION TO DATABASE

NOTE

DROP DATABASE SchoolDB;

- a) Deletes all rows in SchoolDB but keeps the tables
- b) Deletes the database SchoolDB and all its contents permanently
- c) Deletes only empty tables in SchoolDB
- d) Deletes the schema but preserves data

Q 5: Which clause is used in CREATE DATABASE to define text encoding and sorting rules?

- a) COLLATE and CHARACTER SET
- b) PRIMARY KEY
- c) FOREIGN KEY
- d) DEFAULT VALUE

Explanation of MCQs

Q1 b) CREATE DATABASE

Explanation:

The CREATE DATABASE command is a DDL statement used to initialize a new database in the RDBMS. Other commands like DROP DATABASE and ALTER DATABASE are used to delete or modify databases, not create them.

Q2 c) TRUNCATE TABLE

Explanation:

TRUNCATE TABLE removes all data in a table quickly but does not delete the table itself. In contrast, DROP TABLE deletes the table structure along with its data, and DELETE removes rows individually and can be rolled back if within a transaction.

Q3 c) CREATE TABLE

Explanation:

DDL commands define and manage database structures. CREATE TABLE creates a new table, whereas INSERT and UPDATE are DML commands used to manipulate data, and SELECT is a DQL command used for querying.

Q4 b) Deletes the database SchoolDB and all its contents permanently

Explanation:

DROP DATABASE permanently deletes the entire database, including all tables, views, and data. It cannot be undone, so caution is required before executing it.

Q5 a) COLLATE and CHARACTER SET

Explanation:

CHARACTER SET defines the encoding (e.g., UTF-8), and COLLATE specifies sorting and comparison rules for text data. These options are commonly used in MySQL and other RDBMS when creating a database.

B. Short Answer Qs (100 Words)

1. What is SQL?
2. What are the major categories of SQL commands?
3. What is Data Definition Language?
4. Explain the purpose of the CREATE command.
5. What is the role of DDL in database management?

PART 2

C. Long Answer Type Questions (200 Words)

1. Explain SQL and discuss its importance in relational database management.
2. Describe the major categories of SQL commands.
3. Explain the purpose and features of Data Definition Language.
4. Discuss the CREATE, ALTER, and DROP commands in SQL.
5. Explain the differences between DDL and DML commands.

PART 3

D. Long Answer Type Questions (300 Words)

1. Explain SQL in detail and discuss its role in managing relational databases.
2. Discuss the different categories of SQL commands and explain their major purposes.
3. Explain Data Definition Language and discuss commonly used DDL commands with suitable examples.
4. Compare DDL and DML in terms of purpose, operations, and database objects affected.

STRUCTURED QUERY LANGUAGE (SQL)

NOTE

5. Analyze the importance of SQL knowledge for database developers and database administrators.

UNIT 13

TABLE OPERATIONS USING DDL

UNIT STRUCTURE

- 13.1 Introduction
- 13.2 Objective
- 13.3 CREATE TABLE
- 13.4 DROP TABLE
- 13.5 ALTER TABLE
- 13.6 Summary
- 13.7 Glossary
- 13.8 Answer to Check Your Progress
- 13.9 Suggested Readings – OER
- 13.10 Reference
- 13.11 Exercise

13.1 INTRODUCTION

Tables are fundamental structures in relational databases because they provide an organized way to store related information. Database developers must be able to create tables, modify their structures, and remove them when they are no longer required. Data Definition Language provides SQL commands that support these structural operations.

This unit focuses on table operations using DDL commands. Students will learn how to create tables and define their attributes and appropriate constraints. The CREATE TABLE command forms the basis for establishing a new relational table. Once a table has been created, its structure may need to be changed to accommodate new requirements, which can be performed using the ALTER TABLE command. The unit also discusses DROP and TRUNCATE operations and explains their different purposes. DROP is used to remove a database object, whereas TRUNCATE removes records while retaining the table structure. These operations must be used carefully because they can significantly affect database contents or structures. Through this unit, students will develop practical knowledge of managing relational table structures using SQL. The concepts presented here are important for database development because database requirements may change over time. Understanding DDL-based table operations enables learners to create flexible database structures and make appropriate modifications while managing database objects effectively.

NOTE**13.2 OBJECTIVES**

After studying this unit, you will be able to:

1. Define the major DDL operations performed on database tables.
2. Explain the purpose of CREATE, ALTER, DROP, and TRUNCATE commands.
3. Demonstrate the creation and modification of tables using DDL commands.
4. Analyze the effects of different DDL operations on database tables.
5. Select appropriate DDL commands for creating, modifying, and removing database structures.

13.3 CREATE TABLE

The CREATE TABLE command in SQL is a DDL (Data Definition Language) statement used to define and create a new table in a relational database. Tables are the primary structures for storing data in rows (tuples) and columns (attributes), and CREATE TABLE allows you to specify the table structure, including columns, data types, and constraints that ensure data integrity and consistency.



Figure 13.1: Create Table

Purpose of CREATE TABLE

- To define a new table for storing related data in a structured format.
- To specify columns and their data types, determining the type of data each column can hold.
- To apply constraints like primary key, foreign key, unique, not null, and check to maintain data integrity.
- To provide a foundation for data insertion, querying, and manipulation.

NOTE**Basic Syntax**

```
CREATE TABLE table_name (
    column1_name data_type [constraints],
    column2_name data_type [constraints],
    ...
    table_constraints);
```

Components of CREATE TABLE**1. Table Name:**

- The identifier for the table (e.g., Student, Employee).
- Must be unique within the database.

2. Column Definitions:

- Each column must have a name and data type.
- Optional column-level constraints can be added to enforce rules.
- **Example Column Definition:**

```
StudentID INT PRIMARY KEY,
Name VARCHAR(50) NOT NULL,
Age INT,
Email VARCHAR(50) UNIQUE
```

3. Data Types:

Common SQL data types include:

- **INT / INTEGER** – Whole numbers
- **VARCHAR(n)** – Variable-length text (max n characters)
- **CHAR(n)** – Fixed-length text
- **DATE** – Calendar date
- **DECIMAL(p,s) / NUMERIC(p,s)** – Precise numeric values
- **BOOLEAN** – True/False values

4. Constraints:

NOTE

Constraints enforce data integrity. Examples include:

- **PRIMARY KEY:** Uniquely identifies each row; cannot be NULL
- **FOREIGN KEY:** References a primary key in another table to maintain relationships
- **NOT NULL:** Ensures a column cannot have NULL values
- **UNIQUE:** Ensures all values in a column are distinct
- **CHECK:** Ensures column values satisfy a condition
- **DEFAULT:** Assigns a default value if none is provided

Example 1 – Simple Table Creation

```
CREATE TABLE Student (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(50) NOT NULL,  
    Age INT,  
    Email VARCHAR(50) UNIQUE);
```

- Creates a Student table with primary key, not null, and unique constraints.

Example 2 – Table with Foreign Key

```
CREATE TABLE Enrollment (  
    EnrollmentID INT PRIMARY KEY,  
    StudentID INT,  
    CourseID INT,  
    EnrollmentDate DATE,  
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),  
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID));
```

- Enrollment table references Student and Course tables, enforcing referential integrity.

Important Points

1. **Table Name Uniqueness:** Table names must be unique within a database.
2. **Column Names:** Each column must have a distinct name within the table.
3. **Constraints:** Can be defined at column level (inline) or table level (after all columns).
4. **Order of Creation:** Tables referenced by foreign keys must exist before creating dependent tables.
5. **Persistence:** Once created, a table remains in the database until dropped explicitly.

NOTE

The CREATE TABLE command is a fundamental SQL operation that establishes the structure of relational data. By specifying columns, data types, and constraints, it ensures that data is stored efficiently, accurately, and consistently. Mastery of CREATE TABLE is essential for database design, application development, and enforcing data integrity in relational databases.

13.4 DROP TABLE

The DROP TABLE command in SQL is a DDL (Data Definition Language) statement used to permanently delete an existing table from a database. Unlike commands that remove only data (like DELETE or TRUNCATE), DROP TABLE removes the entire table structure, its data, indexes, constraints, and dependent objects. This operation is irreversible, so it must be executed with caution.



Figure 13.2: Drop Table

Purpose of DROP TABLE

- To completely remove a table and all its data from the database.
- To clean up unused or temporary tables from the database.
- To reorganize database structures when redesigning or updating the schema.

NOTE**Syntax**

DROP TABLE table_name;

Optional syntax in some RDBMS:

DROP TABLE IF EXISTS table_name;

- IF EXISTS ensures the command executes only if the table exists, preventing errors if the table is missing.

Examples**Example 1 – Simple Table Deletion**

DROP TABLE Student;

- Deletes the Student table along with all rows, indexes, and constraints.

Example 2 – Conditional Deletion (MySQL / PostgreSQL)

DROP TABLE IF EXISTS Enrollment;

- Deletes the Enrollment table only if it exists, preventing execution errors.

Important Points

1. **Irreversible Action:** All table data and structure are permanently deleted. Backup is recommended before dropping important tables.
2. **Permissions Required:** Only users with DROP TABLE privileges can execute this command.
3. **Dependent Objects:** Tables referenced by foreign keys in other tables may not be dropped unless those constraints are removed first or CASCADE is used (if supported).
4. **Database in Use:** The table must exist in the current database; you cannot drop a table from another database without specifying the database name in some RDBMS.

Table 13.1: Difference from TRUNCATE TABLE

Feature	DROP TABLE	TRUNCATE TABLE
Deletes table structure	Yes	No

Feature	DROP TABLE	TRUNCATE TABLE
Deletes all rows	Yes	Yes
Can be rolled back	Usually No	Sometimes No (depends on DB)
Preserves table for reuse	No	Yes
Affects constraints	Deletes constraints permanently	Constraints remain

Example – Dropping a Table with Foreign Key Dependency (SQL Server)

```
ALTER TABLE Enrollment DROP CONSTRAINT FK_StudentID;
DROP TABLE Student;
```

- First removes the foreign key constraint, then drops the Student table safely.

The DROP TABLE command is a powerful DDL operation for completely removing a table from a database. It permanently deletes both data and structure, so proper precautions such as backups and removing dependencies are essential. Mastery of DROP TABLE is crucial for database maintenance, schema redesign, and cleanup tasks.

13.5 ALTER TABLE

The ALTER TABLE command in SQL is a DDL (Data Definition Language) statement used to modify the structure of an existing table without losing its data. It is a versatile command that allows database administrators and developers to add, modify, or remove columns, constraints, and other table attributes. ALTER TABLE is essential for adapting table structures to changing requirements while preserving existing data.

NOTE**Figure 13.3: Alter Table****Purpose of ALTER TABLE**

- To add new columns to an existing table.
- To modify the data type or attributes of existing columns.
- To drop columns or remove unwanted columns.
- To add, modify, or drop constraints such as primary keys, foreign keys, unique, and check constraints.
- To rename tables or columns (supported in some RDBMS).

Syntax

The basic syntax varies depending on the operation:

1. Add Column

```
ALTER TABLE table_name
ADD column_name data_type [constraints];
```

2. Modify Column

```
ALTER TABLE table_name
MODIFY column_name new_data_type [constraints]; -- MySQL
ALTER TABLE table_name
ALTER COLUMN column_name TYPE new_data_type; -- PostgreSQL
```

3. Drop Column

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

4. Add Constraint

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name constraint_definition;
```

5. Drop Constraint

```
ALTER TABLE table_name  
DROP CONSTRAINT constraint_name;
```

Examples

Example 1 – Adding a Column

```
ALTER TABLE Student  
ADD PhoneNumber VARCHAR(15);
```

- Adds a new column PhoneNumber to the Student table.

Example 2 – Modifying a Column Data Type

```
ALTER TABLE Student  
MODIFY Age SMALLINT; -- MySQL
```

- Changes the Age column to a smaller integer type.

Example 3 – Dropping a Column

```
ALTER TABLE Student  
DROP COLUMN Email;
```

- Removes the Email column from the Student table permanently.

Example 4 – Adding a Constraint

```
ALTER TABLE Student  
ADD CONSTRAINT PK_StudentID PRIMARY KEY (StudentID);
```

- Adds a primary key constraint to the StudentID column.

Example 5 – Dropping a Constraint

NOTE

```
ALTER TABLE Student  
DROP CONSTRAINT PK_StudentID;
```

- Removes the primary key constraint from the StudentID column.

Important Points

1. **Preserves Data:** ALTER TABLE does not delete existing data unless columns are dropped.
2. **Constraint Handling:** Foreign key dependencies may need to be handled before modifying or dropping columns.
3. **RDBMS Variations:** Syntax for modifying columns or constraints may differ between MySQL, SQL Server, and PostgreSQL.
4. **Multiple Alterations:** Some RDBMS allow multiple alterations in a single statement, while others require separate statements.
5. **Table Locking:** Altering table structure may lock the table temporarily, affecting concurrent operations.

The ALTER TABLE command is a critical tool for maintaining and evolving database schemas. It provides the flexibility to adapt existing tables to new requirements by adding, modifying, or removing columns and constraints without losing existing data. Mastery of ALTER TABLE is essential for database administration, schema evolution, and maintaining data integrity.



Activity 13.1:

How to use create, drop and alter command on table? Explain taking an example.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

Check Your Progress 13.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is the difference between DROP TABLE and TRUNCATE TABLE in SQL?

Question 2: Explain the purpose of the ALTER TABLE command in SQL.

13.6 SUMMARY

Unit 13 focuses on managing and modifying tables in relational databases using Data Definition Language (DDL) commands in SQL. Tables are the core structures for storing data in rows and columns, and effective table management is essential for ensuring data integrity, consistency, and accessibility.

The unit covers the following key table operations:

1. **CREATE TABLE:** Defines a new table with specified columns, data types, and constraints (e.g., primary key, foreign key, unique, not null). This establishes the structure for storing and organizing data.
2. **ALTER TABLE:** Allows modification of an existing table by adding, modifying, or dropping columns, and by adding or removing constraints. This enables the database schema to adapt to changing requirements while preserving existing data.
3. **DROP TABLE:** Permanently deletes a table along with all its data, indexes, and constraints. It is a destructive operation, so careful use and backups are recommended.
4. **TRUNCATE TABLE:** Removes all rows from a table while retaining its structure. It is faster than DELETE because it does not log individual row deletions, making it ideal for clearing large tables without affecting the table schema.

Key Takeaways:

NOTE

- DDL commands are essential for table creation, modification, and deletion in relational databases.
- Constraints like primary keys, foreign keys, unique, and not null are critical for maintaining data integrity.
- ALTER TABLE provides flexibility for schema evolution without data loss.
- Proper understanding of table operations ensures efficient database design, maintenance, and secure data handling.

In essence, Unit 13 equips learners with the knowledge and practical skills to create, manage, and maintain tables, forming the foundation for data manipulation and querying in subsequent SQL operations.

13.7 GLOSSARY

1. **Table** – A structured collection of data in a relational database organized in rows (records) and columns (attributes).
2. **DDL (Data Definition Language)** – A subset of SQL commands used to define and manage database structures, including tables, columns, and constraints.
3. **CREATE TABLE** – A DDL command used to define and create a new table with specified columns, data types, and constraints.
4. **ALTER TABLE** – A DDL command used to modify the structure of an existing table, such as adding, deleting, or modifying columns and constraints.
5. **DROP TABLE** – A DDL command used to permanently delete a table along with all its data, indexes, and constraints.
6. **TRUNCATE TABLE** – A DDL command used to remove all rows from a table while keeping the table structure intact.
7. **Column** – A vertical division in a table that represents an attribute of the entity, with a specific data type and constraints.
8. **Row / Tuple** – A horizontal record in a table that represents a single instance of data.
9. **Primary Key** – A column or set of columns that uniquely identifies each row in a table and cannot contain NULL values.
10. **Foreign Key** – A column or set of columns that references a primary key in another table, enforcing referential integrity.
11. **Constraint** – A rule applied to a column or table to ensure the accuracy and integrity of data. Examples: PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK.
12. **Data Type** – Defines the type of data a column can store, such as INT, VARCHAR, DATE, or BOOLEAN.
13. **Schema Evolution** – The process of modifying table structures over time to accommodate changing business requirements, typically using ALTER TABLE.

NOTE

14. **Referential Integrity** – Ensures that foreign key values correspond to existing primary key values in the referenced table.
15. **Table Structure** – The definition of a table, including its columns, data types, and constraints.

13.8 ANSWERS TO CHECK YOUR PROGRESS

Q 1: What is the difference between DROP TABLE and TRUNCATE TABLE in SQL?

Answer:

- **DROP TABLE:** Permanently deletes the table along with all its data, indexes, and constraints. The table structure is also removed.
- **TRUNCATE TABLE:** Removes all rows from the table but retains the table structure for future use. It is faster than DELETE because it does not log individual row deletions.

Q 2: Explain the purpose of the ALTER TABLE command in SQL.

Answer:

The ALTER TABLE command is used to modify the structure of an existing table without losing its data. It allows you to:

- Add new columns to the table.
- Modify column data types or attributes.
- Drop columns that are no longer needed.
- Add, modify, or drop constraints such as primary keys, foreign keys, and unique constraints.

13.9 SUGGESTED READINGS – OER

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13718	

13.10 REFERENCES (APA 7TH EDITION)

- Elmasri, R., & Navathe, S. B. (2021). *Fundamentals of Database Systems* (8th ed.). Pearson.

NOTE

- Detailed explanations on table creation, modification, deletion, and constraints using DDL commands.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill.
- Covers table operations including CREATE TABLE, ALTER TABLE, DROP TABLE, and TRUNCATE TABLE with examples.

13.11 EXERCISE

PART 1**A. Multiple Choice Qs**

Q 1: Which SQL command is used to create a new table in a database?

- a) CREATE DATABASE
- b) CREATE TABLE
- c) ALTER TABLE
- d) DROP TABLE

Q 2: Which SQL command removes all data from a table but keeps the table structure intact?

- a) DROP TABLE
- b) DELETE
- c) TRUNCATE TABLE
- d) ALTER TABLE

Q 3: What is the purpose of the ALTER TABLE command?

- a) Delete a table and all its data
- b) Create a new table in the database
- c) Modify the structure of an existing table
- d) Query data from a table

Q 4: What happens when you execute the following command?

DROP TABLE Student;

- a) Deletes all rows but keeps the table structure
- b) Deletes the table Student along with all data and constraints permanently
- c) Renames the table
- d) Creates a backup of the table

Q 5: Which of the following is a correct way to add a new column PhoneNumber to an existing table Student?

- a) CREATE TABLE Student (PhoneNumber VARCHAR(15));

- b) ALTER TABLE Student ADD PhoneNumber VARCHAR(15);
- c) DROP TABLE Student ADD COLUMN PhoneNumber;
- d) TRUNCATE TABLE Student ADD PhoneNumber;

Explanation of MCQs**Q1** b) CREATE TABLE**Explanation:**

The CREATE TABLE command is a DDL statement used to define a new table with columns, data types, and constraints. CREATE DATABASE creates the database itself, while ALTER TABLE modifies an existing table, and DROP TABLE deletes a table.

Q2 c) TRUNCATE TABLE**Explanation:**

TRUNCATE TABLE deletes all rows from the table quickly without removing the table structure, making it ideal for clearing data while preserving columns and constraints. DROP TABLE removes the table entirely.

Q3 c) Modify the structure of an existing table**Explanation:**

ALTER TABLE is used to add, modify, or drop columns and add or remove constraints in an existing table without affecting the existing data.

Q4 b) Deletes the table Student along with all data and constraints permanently**Explanation:**

DROP TABLE permanently removes the table structure, data, and associated constraints from the database. This operation cannot be undone unless a backup exists.

Q5 b) ALTER TABLE Student ADD PhoneNumber VARCHAR(15);

NOTE**Explanation:**

ALTER TABLE with the ADD clause is used to add new columns to an existing table. CREATE TABLE defines a new table, and DROP TABLE or TRUNCATE TABLE are used for deleting tables or clearing.

B. Short Answer Questions (100 Words)

1. How is a table created using DDL?
2. What is the purpose of the ALTER TABLE command?
3. What is the DROP TABLE command?
4. Explain the purpose of the TRUNCATE command.
5. Differentiate between DROP and TRUNCATE commands.

PART 2**C. Long Answer Type Questions (200 Words)**

1. Explain the process of creating tables using SQL DDL commands.
2. Describe how the ALTER TABLE command is used to modify a table.
3. Explain the purpose and working of the DROP TABLE command.
4. Differentiate between DELETE, DROP, and TRUNCATE operations.
5. Explain how constraints can be defined while creating a table.

PART 3**D. Long Answer Type Questions (300 Words)**

1. Explain different table operations performed using DDL commands with suitable examples.
2. Discuss CREATE TABLE, ALTER TABLE, DROP TABLE, and TRUNCATE commands and their applications.
3. Explain how table structure can be modified using ALTER TABLE and discuss common modifications.
4. Compare DROP, TRUNCATE, and DELETE operations based on their purpose and effect on database tables.
5. Describe the complete process of creating and modifying a relational table using SQL DDL commands.

UNIT 14

DATA MANIPULATION LANGUAGE (DML)

NOTE

UNIT STRUCTURE

- 14.1 Introduction
- 14.2 Objective
- 14.3 INSERT Statement
- 14.4 UPDATE Statement
- 14.5 DELETE Statement
- 14.6 SELECT and Filtering Data
- 14.7 Summary
- 14.8 Glossary
- 14.9 Answer to Check Your Progress
- 14.10 Suggested Readings – OER
- 14.11 Reference
- 14.12 Exercise

14.1 INTRODUCTION

A database becomes useful when users and applications can insert, modify, delete, and manage the information stored within its tables. Data Manipulation Language, commonly known as DML, provides SQL commands that enable users to perform these essential data management operations. DML is therefore an important part of practical database usage and application development. This unit introduces the concepts and functions of Data Manipulation Language. Students will learn about major DML operations, particularly INSERT, UPDATE, and DELETE. The INSERT command is used to add new records to a table, while UPDATE modifies existing records according to specified conditions. DELETE removes records that are no longer required. The unit emphasizes the importance of performing DML operations carefully because incorrect commands can result in unintended changes to database information. Students will also learn how DML operations relate to relational tables and database integrity. Appropriate use of conditions and constraints is essential for ensuring that data remains accurate and consistent. Understanding DML provides a foundation for developing database-driven applications in which information is continuously added, modified, and removed. By the end of this unit, learners will have a conceptual foundation for working with data

NOTE

manipulation operations and will be better prepared to use SQL for practical database management tasks.

14.2 OBJECTIVES

After studying this unit, you will be able to:

- Define Data Manipulation Language and its major commands.
- Explain the purpose of INSERT, UPDATE, and DELETE operations.
- Demonstrate basic data manipulation operations on relational tables.
- Analyze the effects of DML operations on database records.
- Select appropriate DML commands for performing different data management tasks.

14.3 INSERT STATEMENT

The INSERT statement in SQL is a Data Manipulation Language (DML) command used to add new records (rows) into a table. It is one of the most fundamental operations for populating relational databases with data. The INSERT statement allows users to specify values for each column, ensuring that data is stored according to the table's structure and constraints.



Figure 14.1: Insert Statement

Purpose of INSERT

- To add new rows of data into a table.
- To ensure data adheres to defined data types and constraints (e.g., PRIMARY KEY, NOT NULL, UNIQUE).
- To populate tables with initial or ongoing data for database applications.

Basic Syntax

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

- table_name: Name of the table where data is to be inserted.
- (column1, column2, ...): Optional list of columns. If omitted, values must be provided for all columns in the table in the defined order.
- VALUES: The keyword used to specify the data values for the columns.

Examples**Example 1 – Inserting values into all columns**

```
INSERT INTO Student (StudentID, Name, Age, Email)
VALUES (101, 'Ankur Khare', 22, 'ankur@example.com');
```

- Adds a new row to the Student table with values for all columns.

Example 2 – Inserting values into specific columns

```
INSERT INTO Student (StudentID, Name)
VALUES (102, 'Riya Sharma');
```

- Adds a new row where only StudentID and Name are specified. Other columns will use NULL or default values if defined.

Example 3 – Inserting multiple rows at once

```
INSERT INTO Student (StudentID, Name, Age, Email)
VALUES
(103, 'Amit Singh', 21, 'amit@example.com'),
(104, 'Neha Verma', 23, 'neha@example.com');
```

- Inserts multiple rows in a single statement, improving efficiency.

Important Points**1. Data Types and Constraints:**

Values must match the column data types and satisfy constraints like PRIMARY KEY, UNIQUE, and NOT NULL.

NOTE**2. Default Values:**

If a column has a default value, it can be omitted in the INSERT statement.

3. Auto-increment Columns:

Columns defined with AUTO_INCREMENT (MySQL) or IDENTITY (SQL Server) do not require explicit values; the database generates them automatically.

4. Order Matters:

If column names are not specified, values must match the table's column order.

5. NULL Values:

Columns that allow NULL can be left out, but columns defined as NOT NULL must always receive a value.

The INSERT statement is a crucial DML operation that allows users to add data to relational tables efficiently and accurately. Understanding how to insert data while respecting constraints, data types, and default values is fundamental for database management and application development. Mastery of INSERT is the first step toward performing all other data manipulation operations in SQL.

14.4 UPDATE STATEMENT

The UPDATE statement in SQL is a Data Manipulation Language (DML) command used to modify existing records in a table. Unlike the INSERT statement, which adds new rows, UPDATE allows you to change one or more column values of one or multiple rows based on specified conditions. It is essential for maintaining accurate and up-to-date data in a relational database.

NOTE**Figure 14.2: Update Statement****Purpose of UPDATE**

- To modify existing data in a table without deleting it.
- To correct or change values in one or more columns.
- To apply bulk updates to multiple rows efficiently using conditions.

Basic Syntax

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

- **table_name:** The name of the table to update.
- **SET:** Specifies the columns and new values to be assigned.
- **WHERE condition:** Determines which rows are affected by the update.
 - If omitted, all rows in the table are updated.

Examples**Example 1 – Updating a single row**

```
UPDATE Student
SET Age = 23
WHERE StudentID = 101;
```

- Updates the Age of the student with StudentID = 101 to 23.

NOTE**Example 2 – Updating multiple columns**

```
UPDATE Student  
SET Age = 22, Email = 'ankur.k@example.com'  
WHERE StudentID = 101;
```

- Updates both Age and Email for the student with StudentID = 101.

Example 3 – Updating multiple rows

```
UPDATE Student  
SET Age = Age + 1  
WHERE Age < 25;
```

- Increments the Age by 1 for all students younger than 25.

Important Points

1. **WHERE Clause is Crucial:**
 - Without a WHERE clause, all rows in the table will be updated.
 - Always double-check conditions to avoid unintended data modification.
2. **Constraints:**
 - Updates must respect column constraints such as PRIMARY KEY, UNIQUE, NOT NULL, and data types.
3. **Expressions:**
 - You can use arithmetic or string expressions in SET. For example: Salary = Salary * 1.10.
4. **Transactions:**
 - In transactional databases, updates can be rolled back if wrapped in a transaction, providing data safety.
5. **Multiple Columns:**
 - You can update one or more columns simultaneously in a single UPDATE statement.

The UPDATE statement is a powerful DML command that allows precise modification of existing data. By using conditions, constraints, and expressions, users can efficiently maintain data accuracy and consistency. Mastery of UPDATE is critical for database maintenance, correcting errors, and managing dynamic data in relational databases.

**Activity 14.1:**

How to use insert and update statement in database? Explain with the help of an example.

Materials Needed:

Pens and paper or digital devices for writing

Access to the internet for research (optional)

14.5 DELETE STATEMENT

The DELETE statement in SQL is a Data Manipulation Language (DML) command used to remove existing rows from a table. Unlike the DROP TABLE command, which deletes the table structure along with its data, DELETE removes only selected data while preserving the table and its structure. This command is essential for maintaining data accuracy, cleaning outdated records, and managing dynamic datasets in relational databases.



Figure 14.3: Delete Statement

Purpose of DELETE

NOTE

- To remove specific rows from a table based on conditions.
- To maintain data consistency and relevance by deleting obsolete or incorrect records.
- To allow selective deletion, unlike TRUNCATE TABLE which removes all rows.

Basic Syntax

```
DELETE FROM table_name  
WHERE condition;
```

- table_name: The name of the table from which data is to be deleted.
- WHERE condition: Specifies which rows should be deleted.
 - If omitted, all rows in the table will be deleted.

Examples

Example 1 – Deleting a single row

```
DELETE FROM Student  
WHERE StudentID = 101;
```

- Deletes the record of the student with StudentID = 101.

Example 2 – Deleting multiple rows

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

- Deletes all students whose Age is less than 20.

Example 3 – Deleting all rows

```
DELETE FROM Student;
```

- Removes all rows from the Student table but keeps the table structure intact.

Important Points

1. **WHERE Clause is Crucial:**
 - Omitting the WHERE clause will delete all rows, which can lead to data loss.
 - Always verify conditions before executing a delete operation.
2. **Constraints:**

NOTE

- Deletions must respect foreign key constraints.
- If a row is referenced by a foreign key in another table, deletion may fail unless ON DELETE CASCADE is used.

3. Transactions:

- In transactional databases, DELETE operations can be rolled back if within a transaction.

4. Performance Considerations:

- DELETE operations can be slower than TRUNCATE TABLE for large tables because each row deletion is logged individually.

Table 14.1: Difference from TRUNCATE TABLE:

Feature	DELETE	TRUNCATE TABLE
Removes rows	Yes, based on WHERE condition	Yes, all rows
Preserves table structure	Yes	Yes
Can be rolled back	Yes (in transactions)	Usually yes/no depending on DB
Fires triggers	Yes	No

The DELETE statement is a core DML command for removing data selectively from relational tables. By using the WHERE clause and respecting constraints, it allows precise control over data removal while maintaining table structure. Mastery of DELETE is essential for data maintenance, cleanup, and managing dynamic datasets in real-world database applications.

14.6 SELECT AND FILTERING DATA

The SELECT statement in SQL is the primary DML command used to retrieve data from one or more tables in a relational database. Unlike INSERT, UPDATE, or DELETE, which modify data, SELECT allows users to view, query, and analyze stored data without altering it. This makes it fundamental for reporting, data analysis, and database interaction.

NOTE**Figure 14.4: Select and Filtering Data****Purpose of SELECT**

- To retrieve specific data from one or multiple tables.
- To filter and sort data according to user-defined conditions.
- To enable aggregation, calculation, and reporting on stored data.
- To allow joining multiple tables to extract meaningful information.

Basic Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
ORDER BY column [ASC | DESC];
```

- **SELECT column1, column2, ...** – Specifies the columns to retrieve. Use * to retrieve all columns.
- **FROM table_name** – Specifies the table(s) from which data is retrieved.
- **WHERE condition** – Filters rows that meet specific criteria.

- ORDER BY column – Sorts the result set ascending (ASC) or descending (DESC).

Filtering Data Using WHERE Clause

The WHERE clause is used with SELECT to filter rows based on specified conditions.

Operators Used in Filtering:

- **Comparison operators:** =, != (or <>), <, >, <=, >=
- **Logical operators:** AND, OR, NOT
- **Pattern matching:** LIKE (for partial text matches)
- **Range checks:** BETWEEN ... AND ...
- **Set membership:** IN (...)
- **Null checks:** IS NULL, IS NOT NULL

Examples

Example 1 – Select all columns

```
SELECT * FROM Student;
```

- Retrieves all rows and columns from the Student table.

Example 2 – Select specific columns

```
SELECT Name, Age FROM Student;
```

- Retrieves only the Name and Age columns for all students.

Example 3 – Filtering with WHERE

```
SELECT Name, Age
FROM Student
WHERE Age >= 21;
```

- Retrieves students whose age is 21 or older.

Example 4 – Using logical operators

```
SELECT Name, Age
FROM Student
WHERE Age >= 21 AND Age <= 25;
```

- Retrieves students whose age is between 21 and 25.

NOTE**Example 5 – Pattern matching with LIKE**

```
SELECT Name  
FROM Student  
WHERE Name LIKE 'A%';
```

- Retrieves students whose name starts with 'A'.

Example 6 – Sorting results

```
SELECT Name, Age  
FROM Student  
WHERE Age >= 21  
ORDER BY Name ASC;
```

- Retrieves students aged 21 or older, sorted by name in ascending order.

Important Points

1. **Column Selection:**
 - You can select one, multiple, or all columns using *.
2. **Filtering Rows:**
 - The WHERE clause allows precise data selection. Without it, all rows are returned.
3. **Logical Combinations:**
 - Use AND, OR, and NOT to create complex filtering conditions.
4. **Pattern Matching:**
 - LIKE with % (any number of characters) and _ (single character) enables flexible text search.
5. **Sorting Data:**
 - ORDER BY allows results to be sorted ascending (ASC) or descending (DESC) for easier analysis.
6. **Null Handling:**
 - IS NULL and IS NOT NULL are used to handle missing or undefined data.

The SELECT statement is the most frequently used DML command because it enables users to view, filter, and analyze data stored in relational databases. By combining column selection, filtering conditions, pattern matching, and sorting, SQL allows precise retrieval of meaningful information. Mastery of SELECT with filtering is essential for database querying, reporting, and analytics in both academic and real-world applications.

Check Your Progress 14.1

Notes: a) Space is given below for your answer.

b) Check your answer with the one given at the end of this unit.

Question 1: What is a DBMS? Explain its advantages?

Question 2: Explain the use of the WHERE clause in DML commands?

14.7 SUMMARY

Unit 14 introduces the Data Manipulation Language (DML), a core subset of SQL that allows users to interact with and manage data stored in relational databases. Unlike DDL, which defines database structures, DML focuses on managing the actual data within tables.

The unit covers the following key DML operations:

1. **INSERT Statement:**

- Used to add new rows to a table.
- Can insert values into all or selected columns, and allows multiple row insertion in a single command.
- Must respect data types and constraints such as PRIMARY KEY, NOT NULL, and UNIQUE.

2. **UPDATE Statement:**

- Modifies existing data in one or more columns of a table.
- Works with a WHERE clause to target specific rows.
- Supports arithmetic, string, and logical operations on column values.

3. **DELETE Statement:**

- Removes rows from a table based on conditions.
- Preserves the table structure, unlike DROP TABLE.
- Must be used carefully; omitting the WHERE clause deletes all rows.

NOTE**4. SELECT Statement and Filtering:**

- Retrieves data from one or more tables.
- Can filter, sort, and organize data using WHERE, ORDER BY, LIKE, BETWEEN, IN, and logical operators (AND, OR, NOT).
- Supports selecting specific columns or all columns and retrieving meaningful subsets of data.

Key Takeaways:

- DML commands are essential for daily database operations, including inserting, updating, deleting, and querying data.
- Proper use of conditions and filtering ensures data integrity and prevents unintended modifications or deletions.
- Mastery of DML enables efficient management and analysis of relational data, supporting real-world applications such as reporting, data cleaning, and business analytics.

Unit 14 equips learners with the practical skills to manipulate data effectively in relational databases.

Understanding INSERT, UPDATE, DELETE, and SELECT with filtering is fundamental for database operations, analytics, and application development.

14.8 GLOSSARY

1. **DML (Data Manipulation Language)** – A subset of SQL commands used to insert, update, delete, and retrieve data from database tables.
2. **INSERT Statement** – A DML command used to add new rows of data into a table.
3. **UPDATE Statement** – A DML command used to modify existing records in a table based on specified conditions.
4. **DELETE Statement** – A DML command used to remove one or more rows from a table while keeping the table structure intact.
5. **SELECT Statement** – A DML command used to retrieve data from one or more tables, optionally filtering, sorting, or aggregating results.
6. **WHERE Clause** – A condition used with DML commands (SELECT, UPDATE, DELETE) to filter rows based on specific criteria.
7. **Logical Operators** – Operators used to combine conditions in SQL, such as AND, OR, NOT.

NOTE

8. **Comparison Operators** – Operators used to compare values in SQL, including =, <>, >, <, >=, <=.
9. **LIKE Operator** – Used in the WHERE clause for pattern matching, with % for multiple characters and _ for a single character.
10. **BETWEEN Operator** – Filters rows where a column's value is within a specified range.
11. **IN Operator** – Filters rows where a column's value matches any value in a specified list.
12. **IS NULL / IS NOT NULL** – Checks whether a column contains NULL or non-NULL values.
13. **ORDER BY Clause** – Sorts the result set of a query in ascending (ASC) or descending (DESC) order.
14. **Row / Record** – A single data entry in a table representing a set of column values.
15. **Column / Attribute** – A vertical field in a table representing a specific type of data.

14.9 ANSWERS TO CHECK YOUR PROGRESS-POSSIBLE ONE

Q 1: What is the difference between DELETE and TRUNCATE TABLE commands in SQL?

Answer:

- **DELETE:** Removes selected rows from a table based on a WHERE clause. It preserves the table structure and can be rolled back in a transaction.
- **TRUNCATE TABLE:** Removes all rows from a table quickly and resets storage allocation, but the table structure remains. It is faster than DELETE for large tables, but usually cannot be rolled back.

Q 2: Explain the use of the WHERE clause in DML commands.

Answer:

The WHERE clause is used to filter rows for operations like SELECT, UPDATE, or DELETE. It allows the database to apply the operation only to rows that meet specific conditions, using comparison operators (=, <, >), logical operators (AND, OR, NOT), pattern matching (LIKE), range (BETWEEN), and set membership (IN). Without WHERE, all rows in the table may be affected.

NOTE**14.10 SUGGESTED READINGS – OER**

Sr. No.	Title	URL	QR Code
1.	Introduction to Database	http://books.aisectonline.com/User/UnitUrls?unitId=13719	

14.11 REFERENCES (APA 7TH EDITION)

1. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts*. McGraw-Hill.
2. Elmasri, R., & Navathe, S. (2021). *Fundamentals of Database Systems*. Pearson.
3. Balagurusamy, E. (2022). *Programming with SQL*. McGraw Hill Education (India).
4. Oracle. (2024). *SQL Language Reference*.
5. NPTEL. (2023). *Database Management Systems Course Materials*.

14.12 EXERCISE**PART 1****A. Multiple Choice Qs**

Q 1: Which SQL command is used to add new rows to a table?

- a) UPDATE
- b) INSERT
- c) DELETE
- d) SELECT

Q 2: What happens if you execute an UPDATE statement without a WHERE clause?

- a) Only the first row is updated
- b) No rows are updated
- c) All rows in the table are updated
- d) The database gives an error

Q 3: Which SQL command is used to remove specific rows from a table while keeping the table structure intact?

- a) DROP TABLE
- b) TRUNCATE TABLE
- c) DELETE
- d) ALTER TABLE

Q 4: Which clause is used with SELECT, UPDATE, or DELETE to filter rows based on conditions?

- a) ORDER BY
- b) GROUP BY
- c) WHERE
- d) HAVING

Q 5: How would you retrieve all students whose names start with the letter 'A'?

- a) SELECT * FROM Student WHERE Name = 'A%';
- b) SELECT * FROM Student WHERE Name LIKE 'A%';
- c) SELECT * FROM Student WHERE Name IN 'A%';
- d) SELECT * FROM Student WHERE Name BETWEEN 'A%';

Explanation of MCQs

Q1 b) INSERT

Explanation:

The INSERT statement is used to add new records into a table. UPDATE modifies existing records, DELETE removes records, and SELECT retrieves data without altering it.

Q2 c) All rows in the table are updated

Explanation:

If the WHERE clause is omitted, the UPDATE statement applies to every row in the table, which can lead to unintended data changes. Always use conditions to target specific rows unless a bulk update is intended.

Q3 c) DELETE

Explanation:

The DELETE command removes rows based on a WHERE clause while preserving the table structure. TRUNCATE

STRUCTURED QUERY LANGUAGE (SQL)

NOTE

TABLE removes all rows quickly, and DROP TABLE deletes the table structure entirely.

Q4 c) WHERE

Explanation:

The WHERE clause specifies conditions that determine which rows are affected or retrieved. ORDER BY sorts results, GROUP BY groups rows, and HAVING filters groups, not individual rows.

Q5 b) SELECT * FROM Student WHERE Name LIKE 'A%';

Explanation:

The LIKE operator with % is used for pattern matching. 'A%' matches all names starting with 'A'. = checks exact matches, IN checks a list, and BETWEEN is used for ranges.

B. Short Answer Ques (100 Words)

1. What is Data Manipulation Language?
2. What is the purpose of the INSERT command?
3. Explain the UPDATE command in SQL.
4. What is the purpose of the DELETE command?
5. Differentiate between DDL and DML.

PART 2

C. Long Answer Type Questions (200 Words)

1. Explain the purpose and importance of DML commands in SQL.
2. Describe the INSERT command with suitable examples.
3. Explain how the UPDATE command is used to modify existing records.
4. Discuss the DELETE command and explain its applications.
5. Explain how DML commands are used to manage data stored in relational tables.

PART 3

D. Long Answer Type Questions (300 Words)

1. Explain DML in detail and discuss the major commands used for manipulating data.
2. Describe INSERT, UPDATE, and DELETE commands with suitable examples and applications.
3. Explain how DML operations can be performed safely while maintaining data integrity.

4. Compare different DML operations based on their purpose, syntax, and effect on database records.
5. Discuss the importance of DML in database applications and explain how it supports day-to-day data management.

NOTE



DR. C.V. RAMAN UNIVERSITY

Chhattisgarh, Bilaspur AN A/SECT GROUP UNIVERSITY

Approved by: PCI | AICTE | NISTE | BCI | Member of: ANU | Joint Committee: DUGC | DEB | AICTE | Recognized by: BGC | A NAAC Accredited University



DR. C. V. RAMAN UNIVERSITY (C.G.)

SLM FEEDBACK FORM

PROGRAMME : _____

COURSE : _____

BLOCK : _____

UNIT : _____

Dear Student,

While studying the units of this block, you may have found certain portions of the text difficult to comprehend. We wish to know your difficulties and suggestions, in order to improve the course. Therefore, we request you to fill up and send us the following questionnaire, which pertains to this block. If you find the space provided insufficient, kindly use a separate sheet.

Please mail to aisectcontent@aisect.org/ for online submission please scan QR code:

Questionnaire

1. Name : _____

2. Enrolment No. :

--	--	--	--	--	--	--	--	--	--	--

3. How many hours did you need for studying the units? :

--

4. Please give your reactions to the following items based on your reading of the block:

Items	Excellent	Very Good	Good	Poor	Give specific Examples (If poor)
Presentation Quality					
Language and Style					
Illustrations					
Used (diagrams, tables, etc.)					
Conceptual Clarity					
Check Your Progress Questions					
Suggested Readings - OER					

5. Any other comments:
