KL (DEEMED TO BE UNIVERSITY) | CDOE CENTER FOR DISTANCE AND ONLINE EDUCATION

MCA

DATABASE SYSTEMS

How to Use Self-Learning Material?

The pedagogy used to design this course is to enable the student to assimilate the concepts with ease. The course is divided into modules. Each module is categorically divided into units or chapters. Each unit has the following elements:

Table of Contents: Each unit has a well-defined table of contents. *For example: "1.1.1. (a)" should be read as "Module 1. Unit 1. Topic 1. (Sub-topic a)" and 1.2.3. (iii) should be read as "Module 1. Unit 2. Topic 3. (Sub-topic iii).*

Aim: It refers to the overall goal that can be achieved by going through the unit.

Instructional Objectives: These are behavioural objectives that describe intended learning and define what the unit intends to deliver.

Learning Outcomes: These are demonstrations of the learner's skills and experience sequences in learning, and refer to what you will be able to accomplish after going through the unit.

Self-Assessment Questions: These include a set of multiple-choice questions to be answered at the end of each topic.

Did You Know?: You will learn some interesting facts about a topic that will help you improve your knowledge. A unit can also contain Quiz, Case Study, Critical Learning Exercises, etc., as metacognitive scaffold for learning.

Summary: This includes brief statements or restatements of the main points of unit and summing up of the knowledge chunks in the unit.

Activity: It actively involves you through various assignments related to direct application of the knowledge gained from the unit. Activities can be both online and offline.

Bibliography: This is a list of books and articles written by a particular author on a particular subject referring to the unit's content.

e-References: This is a list of online resources, including academic e-Books and journal articles that provide reliable and accurate information on any topic.

Video Links: It has links to online videos that help you understand concepts from a variety of online resources.

LEADERSHIP KLEF

President
Er. Koneru Satyanarayana

Vice Chancellor
Dr. G. Pardha Saradhi Varma

Pro-Vice Chancellor
Dr. N. Venkatram

Registrar
Dr. K. Subbarao

CREDITS

Author
R. Nagendra Babu

Director CDOE
C. Shanath Kumar

Instructional Designer
Nabina Das

Content Writer
J. Bharani

Graphic Designer
B. V. Satyanarayana

First Edition, 2023.

Author's Profile

R. Nagendra Babu
Assistant Professor

Nagendra Babu Rajaboina is currently working as an Assistant Professor in the Department of CSE at Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India. He graduated from Andhra University, Visakhapatnam, in 2009, with a B.Tech. in Information Technology. He obtained his M.Tech. Degree in Computer Science and Engineering from K L (Deemed to be) University in 2012. Furthermore, he is pursuing a PhD at Vellore Institute of Technology in Chennai. His research interests include Natural Language Processing, Machine Learning, and Computer Networks.

Database Systems

Course Description

This course introduces the fundamental concepts and advanced techniques of database systems. It is designed for students who want to interpolate how databases work and how to design, manage, and optimise them effectively. The course covers many topics, from basics of database management systems (DBMS) to more complex areas such as transaction management and data recovery.

At the beginning of the course, students will grasp the fundamentals of databases, including the characteristics and advantages of DBMS. You will explore different database environments, interpolate the roles of database users, and examine DBMS architecture. Key concepts like data independence and various types of DBMS will be discussed. This foundation is essential for interpolating how databases operate and why they are essential in modern information systems.

Next, the course focuses on data modelling using the Entity-Relationship (ER) model. Students will grasp representing data structures visually, interpolate different types of entities, relationships, and constraints, and apply this knowledge to real-world scenarios. Advanced modelling topics, such as the Enhanced ER (EER) model, will also be covered, providing you with the skills needed for comprehensive and detailed database design.

The course then explores the Relational Model, a widely used framework in modern databases. You will study relational concepts, constraints, and schema design and learn how to convert ER models into relational schemas. Structured Query Language (SQL) will be a significant focus, teaching you how to define and manipulate database data. You will practice writing advanced SQL statements for querying databases, creating tables, applying constraints, and using relational algebra operators for complex operations.

Database normalisation is another key topic that ensures that databases are efficient and free of redundancy. Students will explore different normal forms, including First (1NF), Second (2NF), Third Normal Forms (3NF), and Boyce-Codd Normal Form (BCNF). You will also learn about higher normal forms and techniques for effective data decomposition, ensuring your databases are optimised for performance.

The course will also introduce you to file and storage structures, discussing various file storage techniques and indexing methods to improve data retrieval efficiency. Techniques like hashing and query optimisation strategies will be covered to ensure that your databases can handle large-scale queries quickly and efficiently.

In the latter part of the course, transaction management becomes the focus. You will learn about the challenges of processing multiple transactions, ensuring that databases remain consistent and reliable. The course will cover the ACID properties of transactions (Atomicity, Consistency, Isolation,

Durability) and different concurrency control techniques like lock-based and timestamp-based methods to manage simultaneous transactions. The course will also introduce recovery techniques, such as shadow paging and the ARIES algorithm, which are essential for maintaining data integrity and ensuring that data can be recovered in case of system failures.

Overall, this course provides students with a solid foundation in database systems and equips them with the necessary skills to design, manage, and optimise databases for various applications.

The course **Database Systems** has **four** modules.

MODULE 1

INTRODUCTION TO DATABASE FUNDAMENTALS AND DATA MODELLING

Database fundamentals, including DBMS characteristics, advantages, environments, users, architecture, data independence, and various DBMS types. Focuses on data modeling with the ER model, including notation, constraints, types, and relationships. Advanced topics such as the Enhanced ER (EER) model and EER diagrams are explored for comprehensive database design.

MODULE 2

THE RELATIONAL MODEL AND SQL

Explores the Relational Model, covering fundamental concepts, constraints, and schema design, including ER to Relational Model conversion. SQL for data definition, manipulation, and complex querying, learning to create tables, define data types, apply constraints, and write advanced SQL statements. Relational algebra operators and comprehensive database design guidelines. Practical skills in views, compound statements, user-defined functions and procedures, cursors, and triggers are emphasised.

MODULE 3

DATABASE NORMALISATION AND FILE STORAGE STRUCTURES

Database normalisation, including First, Second, and Third Normal Forms, BCNF, and higher normal forms with multivalue and join dependencies. It explores decomposition algorithms for effective normalisation. Explores about file and storage structures, including file storage techniques, index structures, indexing, hashing, query processing and optimisation strategies for efficient data retrieval.

MODULE 4

TRANSACTION MANAGEMENT AND DATA RECOVERY

Transaction management covers transaction processing issues, states, and challenges in handling multiple transactions. It explores the ACID properties, system logs, and various concurrency control techniques such as lock-based, timestamp-based, and multiversion techniques. Students will study recovery techniques, including concepts like shadow paging and the ARIES algorithm, ensuring robust data recovery and system resilience in database environments.

Table of Contents

DATABASE SYSTEMS

MODULE 1

Introduction to Database Fundamentals and Data Modelling

Module Description

This module introduces the foundational concepts of databases and database management systems (DBMS). It covers essential DBMS characteristics, including data storage, retrieval, and management, along with key advantages such as reduced redundancy, improved data integrity, and multi-user access. Participants will explore different database environments, including centralised and distributed systems, and grasp the roles of database users like administrators and end-users. The module also covers important aspects of database architecture, focusing on data independence and the benefits of separating logical and physical data layers.

Most of the module is dedicated to data modelling, starting with the Entity-Relationship (ER) model. Students will grasp how to represent data relationships using ER diagrams and interpolate key components such as entities, attributes, and relationships. Additionally, the module explores constraints like cardinality and participation. The Enhanced ER (EER) model is introduced for more advanced database designs, along with EER diagrams to represent specialisation, generalisation, and inheritance. By the end of the module, participants will have a comprehensive interpolation of database fundamentals and data modelling techniques for designing efficient databases.

This module is divided into **two** units.

Unit 1.1 Database Fundamentals
Unit 1.2 Data Modelling with the ER Model

MODULE 1

Introduction to Database Fundamentals and Data Modelling

Unit 1

Database Fundamentals

Unit Table of Contents

Unit 1.1 Database Fundamentals

Aim

To provide foundational knowledge of database management systems (DBMS), including their characteristics, advantages, architecture, environments, users, data independence, and various DBMS types.

Instructional Objectives

This unit intends to:

- Define the key characteristics of a Database Management System (DBMS)
- Explain the different types of DBMS and their structures
- Describe the concept of data independence in a DBMS

Learning Outcomes

Upon completion of the unit, you will be able to:

- Explain the characteristics and functions of a Database Management System (DBMS)
- Differentiate between the advantages of DBMS and traditional file systems
- Discuss the significance of data independence in database management

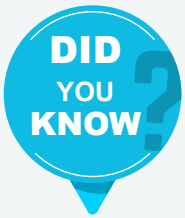
1.1.1 DBMS Characteristics and Advantages

Data: Any fact that can be recorded in the real world is called data.

- Data can be in text, numbers, Images, Audio, videos, etc.
- If data is in the format of text and numbers -> TRADITIONAL DATA
- If data is the format of Images, Audio, or Videos -> MULTIMEDIA DATA

Database: A Collections of information of related data of a particular organisation stored in an electronic container or “REPOSITORY” is called a database

- A database holds the information for the past 20-30 years of data, whereas a data warehouse will hold the information for the past 100years



The first-ever computerised database called the Integrated Data Store (IDS), was created in 1960 by Charles Bachman. It laid the foundation for modern database management systems and introduced the network model, which allowed for complex relationships between data—years before the popular relational model emerged! This innovation revolutionised how businesses store, manage, and access information.

DBMS: A DBMS is a piece of software which is mainly used to perform 4 types of activities:

- To insert data in the Database
- To delete data from the database
- To modify the data or to update the data in the database
- To access the data from the database.

Characteristics of DBMS:

There are so many characteristics of a database management system, which are as follows:

- A database management system can store any data in a database.
- The database management system must support ACID properties (atomicity, consistency, isolation, durability).
- The database management system allows many users to access databases simultaneously.
- Backup and recovery are the two main methods which allow users to protect their data from damage or loss.
- It also provides multiple views for different users in a single organisation.
- It follows the concept of normalisation, which helps minimise the redundancy of a relation.
- It also provides users with query language, which helps insert, retrieve, update, and delete data in a database.

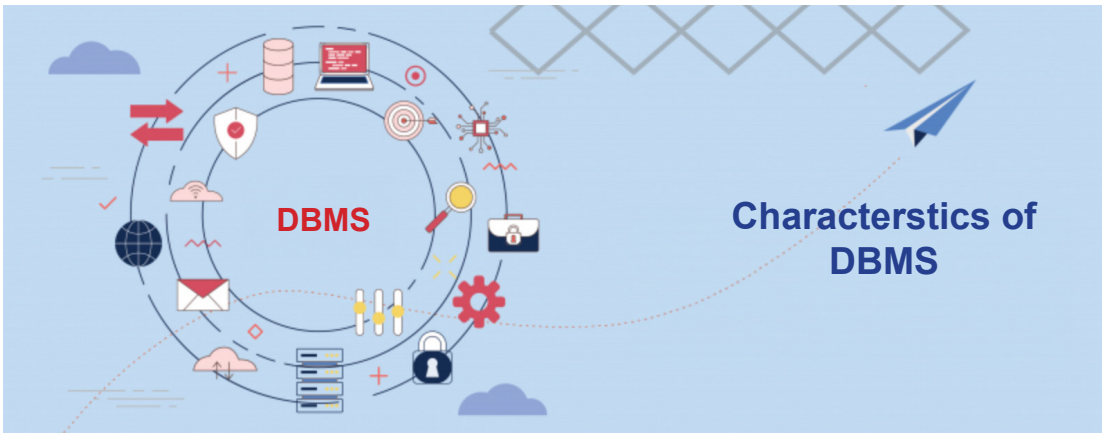


Fig. 1: Characteristics of DBMS

Advantages of DBMS:

The advantages of the DBMS are explained below

The redundancy problem can be solved

- In the File System, duplicate data is created in many places because all the programs have their files, creating data redundancy and wasting memory. In a DBMS, all the files are integrated into a single database, so there is no chance of duplicate data.
- For example, a student record in a library or examination can contain duplicate values, but when it is converted into a single database, all the duplicate values are removed.

It has a very high security level

- Data security is high because it protects your precious data from unauthorised access. Only authorised users should have the grant to access the database with the help of credentials.

Presence of Data Integrity

- Data integrity unifies many files into a single file. DBMS allows data integrity, which makes it easy to decrease data duplication, improve Data integration, and reduce redundancy and inconsistency.

Support multiple users

- DBMS allows multiple users to access the same database simultaneously without any conflicts.

Avoidance of inconsistency

- DBMS controls data redundancy and data consistency.
- In DBMS, data is stored in a single database, which makes it more consistent than file processing systems.

Shared data

- Data can be shared between authorised users of the database in DBMS. All users have the right to access the database. The admin has complete access to the database and can assign users access rights.

Enforcement of standards

- DBMS have central control of the database, so a DBA can ensure that all applications follow data format, document standards, etc. These standards help in data migrations or data exchange.

Any unauthorised access is restricted

- Unauthorised persons cannot access the database because of security credentials.

Provide backup of data

- Data loss is a big problem for all organisations. In the file system, users must back up the files regularly, wasting time and resources.

Tunability

- Tuning means adjusting something to get a better performance. It is the same in the case of DBMS, as it provides tunability to improve performance. DBA adjusts databases to get effective results.

File System vs. Database

File System	Database
We cannot store 500 GB of data in a file system because the system doesn't have enough main memory.	Whereas we can store 500 GB of data in a device called a "hard disk".
Even if you have 500 GB of main memory, we cannot access the data directly from the main memory of more than 4 GB	We can access more than 4 GB of data on a hard disk by writing simple SQL queries.
Data integrity (data quality) will be less in the file system.	However, data integrity is high in the database.
It will provide only a password mechanism for security.	Whereas the security of the database will lie in the hands of database administrators.

Users of Database:

The users of the database are mainly classified into 4 types:

Database administrator (DBA):

- A database administrator can be a single person or a group responsible for controlling and supervising the database.
- A database administrator must create the database structure for the organisation.
- They are responsible for modifying the structure of the database.
- Enforcing the security of a database.
- Maintaining data integrity.
- Executing immediate recovery procedures in case of failures.

Sophisticated users:

- The users who interact /access the database by writing simple SQL queries are called sophisticated users.

Application programmers:

- The users who interact with the database by writing application programs in 4GN languages are called application programmers.

Unsophisticated users:

- Users who interact with databases using application programs written by application programmers are called unsophisticated users.

Data Instance and Data Independence:

Generally, the data in a database changes from time to time. The data found in a database is called a **"data instance /snapshot**."

- Data independence is the ability to modify a scheme definition at one level without affecting the schema definition at the next highest level.

Three-Tier Schema Architecture (or) Data Abstraction (or) Views of Database

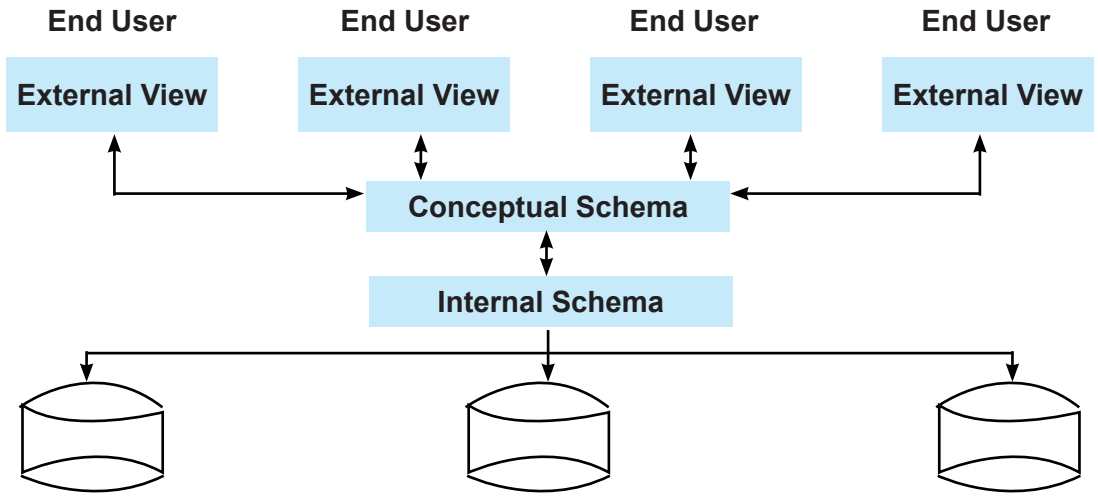


Fig. 2: Three-Tier Schema Architecture

- **Physical/internal Schema:** The physical schema describes the database's physical storage structure, i.e., it discusses the size of the database and access paths from the database.
- **Conceptual /logical level:** At this level, the DBA mainly concentrates on identifying the entities, attributes of the entities, datatypes, and relations btw the entities
- **External level:** It describes what part of the database the endues is accessing.

The architecture of DBMS:

The functional components of a DBMS are classified into two types:

Query Processor Component:

- **DML Compiler:** It translates all the DML statements into low-level instructions that the query evaluation or DB jet engine can understand.
- **DDL Interpreter (data definition language):** It translates all the DDL statements into low-level instructions that the query evaluation engine can understand.
- **Pre-Compiler:** It compiles the fourth-generation languages and interacts with the DML compiler to get low-level instructions that the query evaluation engine can understand.

Query Evaluation Engine: It executes all the low-level instructions that are generated by different compilers and processes the result

Storage Manager Components:

- **Buffer Manager:** It is responsible for processing the data from the disk and sending it to the main memory.
- **File Manager:** It allocates spaces for different files in the database.
- **Authorisation and Integrity Manager:** It checks whether a person is authorised or unauthorised to access the data and whether the database maintains high data quality.
- **Transaction Manager:** It checks whether the concurrent access occurs without conflicts.

Self-Assessment Questions

1. What type of data is considered “traditional data”?

   A). Text and images
   B). Text and audio
   C). Text and numbers
   D). Images and videos

2. A collection of related data stored in an electronic container is known as a:

   A). DBMS
   B). File system
   C). Database
   D). Data warehouse

3. Which of the following is NOT a key function of a DBMS?

   A). Insert data
   B). Delete data
   C). Create network connections
   D). Update data

4. Which of the following is NOT a characteristic of a DBMS?

   A). Backup and recovery support
   B). Multiple views for users
   C). Increased data redundancy
   D). ACID property support

Self-Assessment Questions

5. What is a key advantage of using a DBMS over a file system?

   A). Higher data redundancy
   B). Data duplication
   C). Better security and integrity
   D). Slower data access

6. In a DBMS, data can be shared between:

   A). Only administrators
   B). Authorised users
   C). All users without restrictions
   D). File systems

7. Who is responsible for creating and maintaining the database structure in an organisation?

   A). Sophisticated users
   B). Application programmers
   C). Database administrator (DBA)
   D). Unsophisticated users

8. Which DBMS types are structured in a parent-child relationship?

   A). Network DBMS
   B). Relational DBMS
   C). Hierarchical DBMS
   D). Object-Oriented DBMS

1.1.2 Types of DBMS

Various DBMS Types

Database Management Systems (DBMS) come in different types based on how they store, manage, and process data. Here are the most common types:

1. Hierarchical Database Management System

- **Structure:** Tree-like structure with parent-child relationships. Each parent can have multiple children, but each child has only one parent.
- **Example:** IBM Information Management System (IMS).
- **Use Case:** Banking and telecommunications applications where relationships are hierarchical.

2. Network Database Management System

- **Structure:** This is like a hierarchical structure but allows many-to-many relationships. For example, a child can have multiple parents, creating a more flexible data structure.
- **Example:** Integrated Data Store (IDS), CODASYL.
- **Use Case:** Complex data relationships like supply chains or organisational structures.

3. Relational Database Management System (RDBMS)

- **Structure:** Data is stored in tables (relations), and relationships are managed through foreign keys. SQL (Structured Query Language) is often used for queries.
- **Example:** MySQL, PostgreSQL, Oracle, Microsoft SQL Server.
- **Use Case:** General-purpose use, such as in web applications, banking systems, and customer relationship management (CRM) systems.

4. Object-Oriented Database Management System (OODBMS)

- **Structure:** Data is stored as objects, like in object-oriented programming. It supports inheritance, encapsulation, and polymorphism.
- **Example:** db4o, Object DB.
- **Use Case:** Applications requiring storing complex data types, such as CAD software, engineering designs, and multimedia databases.

5. Document-Oriented Database (NoSQL)

- **Structure:** This type stores data in documents (usually JSON or BSON format) rather than tables. It does not have a fixed schema, offering flexibility.
- **Example:** MongoDB, Couchbase.
- **Use Case:** Content management systems, real-time analytics, or applications with large volumes of unstructured data.

6. Columnar Database (NoSQL)

- **Structure:** Data is stored in columns rather than rows, which allows for fast retrieval of large datasets.
- **Example:** Apache Cassandra, HBase.
- **Use Case:** Data warehousing, real-time analytics, and Big Data applications.

7. Key-Value Store (NoSQL)

- **Structure:** Data is stored as a collection of key-value pairs, where a unique key maps to a value.
- **Example:** Redis, Amazon DynamoDB.
- **Use Case:** Caching, session management, and real-time recommendation engines.

8. Graph Database

- **Structure:** Data is stored in nodes, and relationships (edges) between nodes are first-class entities. Great for representing networks and complex relationships.
- **Example:** Neo4j, Amazon Neptune.
- **Use Case:** Social networks, fraud detection, recommendation systems, and network management.

9. Time-Series Database

- **Structure:** These databases are optimised for storing and querying time-stamped data. They are ideal for handling time-series data such as IoT sensor data, stock prices, etc.
- **Example:** Influx DB, Timescale DB.
- **Use Case:** Monitoring systems, financial market analysis, and IoT applications.

10. Multimodel Database

- **Structure:** Supports multiple data models (e.g., relational, graph, document) within the same database engine.
- **Example:** Arango DB, Couchbase.
- **Use Case:** Complex applications requiring flexible data representations, such as knowledge graphs or content management systems.

Each type of DBMS serves different needs depending on the data's structure, flexibility, scalability, and query complexity.

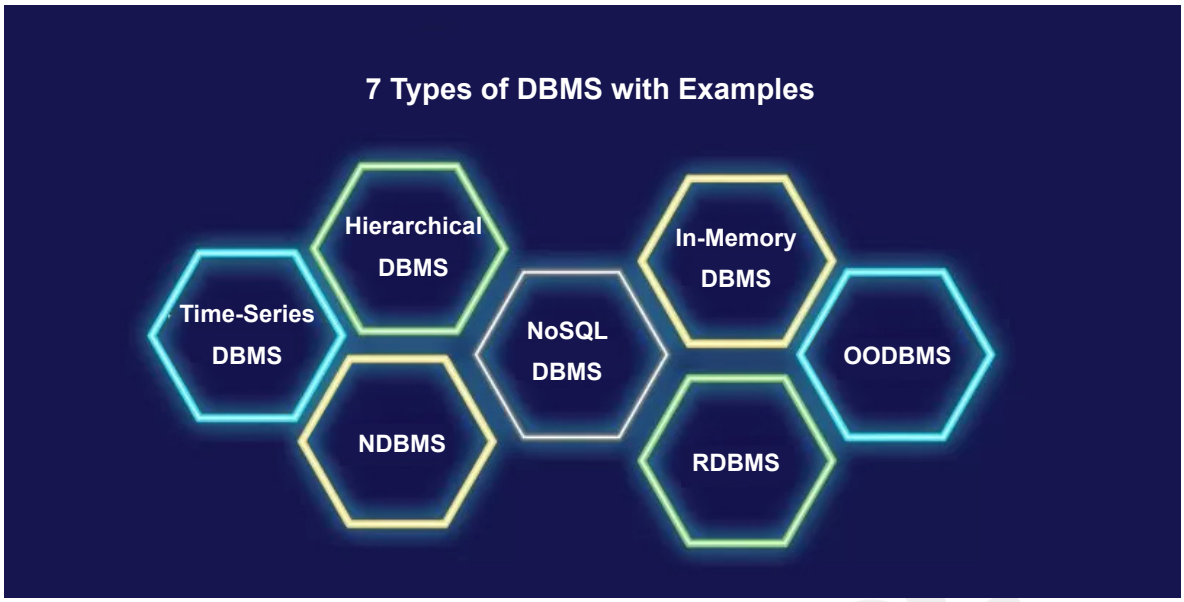


Fig. 3: Types of DBMS

Self-Assessment Questions

9. Which type of DBMS uses a parent-child relationship in a tree-like structure?

   A). Network DBMS
   B). Hierarchical DBMS
   C). Graph Database
   D). Relational DBMS

10. Which DBMS type is most suitable for complex many-to-many relationships?

    A). Hierarchical DBMS
    B). Relational DBMS
    C). Network DBMS
    D). Document-Oriented Database

11. What type of DBMS is best suited for storing data as objects, similar to object-oriented programming?

    A). Object-Oriented DBMS
    B). Relational DBMS
    C). Key-Value Store
    D). Columnar Database

12. Which DBMS is designed for handling unstructured data and typically stores information in a document format like JSON?

    A). Relational DBMS
    B). Document-Oriented Database
    C). Graph Database
    D). Time-Series Database

13. Which DBMS is ideal for representing networks and complex relationships, storing data in nodes and edges?

    A). Relational DBMS
    B). Key-Value Store
    C). Graph Database
    D). Time-Series Database

Summary

- Data can be in different forms, like text, numbers (traditional data), or images, audio, and video (multimedia data).
- A database stores related information in an organised way, often containing years of data.
- A DBMS (Database Management System) helps insert, delete, update, and retrieve data from a database.
- DBMS supports ACID properties, ensuring reliable and consistent data handling.
- It allows many users to access the database simultaneously without conflicts.
- DBMS reduces redundancy by storing data in a unified way and prevents data duplication.
- Data security in DBMS is high, with restricted access for unauthorised users.
- Unlike file systems, DBMS offers better data integrity, security, and handling of large datasets.
- Users of a database include administrators, programmers, and general users, each with different levels of access.
- Different DBMS types, like relational, hierarchical, and NoSQL, depend on data structure and application needs.

Terminal Questions

1. What are the differences between traditional data and multimedia data?
2. How does a database differ from a data warehouse regarding data storage?
3. What are the ACID properties, and why are they important in a DBMS?
4. How does a DBMS handle data redundancy compared to a traditional file system?
5. What is data independence, and why is it crucial in database systems?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	C
2	C
3	C
4	C
5	C
6	B
7	C
8	C
9	B
10	C
11	A
12	B
13	C

Activity

Activity type: Online **Duration: 1 Hour**

Design and implement a simple database using DBMS software (e.g., MySQL, SQLite).

- Create your database.

Glossary

- **ACID:** A set of properties that guarantee database transactions are processed reliably (Atomicity, Consistency, Isolation, Durability).
- **Normalisation:** The process of organising data to reduce redundancy and improve data integrity.
- **SQL (Structured Query Language):** A language used to communicate with a database to perform tasks such as retrieval, insertion, and deletion of data.
- **Schema:** The structure that defines how data is organised in a database, including tables and relationships.
- **Data Integrity:** The accuracy and consistency of data stored in a database.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **DBMS Characteristics:** https://www.interviewbit.com/blog/characteristics-of-dbms/
- **Types of DBMS:** https://databasetown.com/7-types-of-dbms-with-examples/

Video Links

Topic	Link
DBMS Characteristics	https://www.youtube.com/watch?v=wClEbCyWryI
Types of DBMS	https://www.youtube.com/watch?v=ibdb6XlbQBg

Image Credits

Fig.1: Characteristics of DBMS	https://www.interviewbit.com/blog/characteristics-of-dbms/
Fig. 2: Three-Tier Schema Architecture	Self generated
Fig. 3: Types of DBMS	https://www.techprofree.com/7-types-of-dbms-with-examples/

Keywords

- Database
- ACID properties
- Data redundancy
- Backup and recovery
- Data warehouse
- Data security

MODULE 1

Introduction to Database Fundamentals and Data Modelling

Unit 2

Data Modelling with the ER Model

Unit Table of Contents

Unit 1.2 Data Modelling with the ER Model

Aim

To interpolate data modelling with the ER and EER models for designing efficient databases.

Instructional Objectives

This unit intends to:

- Define the basic components of the ER model, including entities, attributes, and relationships
- Describe the different types of attributes in the ER model
- Recognise the symbols used in ER and EER diagrams

Learning Outcomes

Upon completion of the unit, you will be able to:

- Analyse the differences between simple, composite, and derived attributes in an ER model
- Summarise the concepts of one-to-one, one-to-many, many-to-one, and many-to-many relationships
- Discuss the significance of aggregation and participation constraints in ER and EER diagrams

1.2.1 Introduction to Data Modelling and ER Models

Data modelling is an essential aspect of database design, helping in the representation and structuring of data logically. One of the most popular techniques for data modelling is the Entity-Relationship (ER) model. This model focuses on defining entities, their attributes, and their relationships. The ER model is commonly used to design databases and ensures data is stored in an organised and efficient way. Advanced topics like the Enhanced Entity-Relationship (EER) model build on this basic ER model to handle more complex database design requirements.

This topic covers key concepts such as entities, attributes, entity sets, types of attributes, relationships, and mapping cardinalities. It also delves into more advanced areas, such as participation constraints, strong and weak entity sets, class hierarchies (specialisation and generalisation), and aggregation in ER models.

Entity, Attributes, and Entity Set

- Entity: An entity is any object that can be uniquely identified in the real world. For instance, objects like students, faculty, and courses are considered entities in a university database. In an ER diagram, entities are represented by rectangular shapes.
- Attributes: Attributes are an entity's properties or characteristics. In an ER diagram, attributes are represented by oval shapes. For example, a "Student" entity could have attributes like student name, ID, and age.
- Entity Set: An entity set is a collection of similar entities. For example, all students in a university database would make up the "Student" entity set.

Types of Attributes in the ER Model

Attributes in an ER model is classified into five types:

1. **Simple Attributes**: These attributes cannot be further divided. For example, a faculty ID or phone number would be considered simple attributes.
2. **Composite Attributes**: These attributes can be subdivided into smaller parts. For example, a "faculty name" can be divided into first, middle, and last names.
3. **Single-Valued Attributes**: These attributes hold only one value. For example, each member's faculty ID will have exactly one value.
4. **Multi-Valued Attributes**: These attributes can hold more than one value. For instance, a faculty member may have multiple phone numbers, making "phone number" a multi-valued attribute.
5. **Derived Attributes**: These attributes are derived from other attribute values. For example, the average salary could be derived from individual salary values.

Relationships and Relationship Sets

A relationship represents an association between two or more entities. An ER diagram shows a relationship using a diamond or rhombus shape.

Example:

- Faculty **works in** the Department
- **Since** it is a descriptive attribute of the “works in” relationship, it represents the date from when the faculty started working in that department.

The relationship is represented in an ER diagram using a diamond shape/rhombus shape.

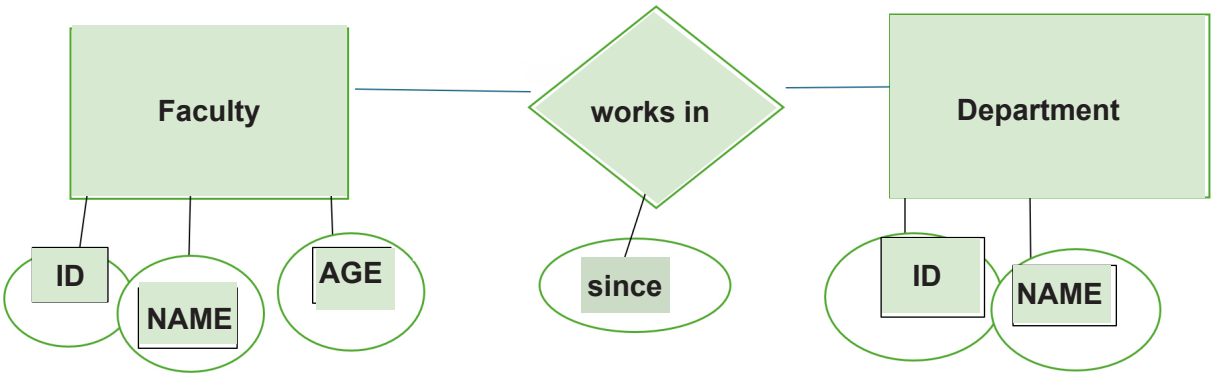


Fig. 1: Relationship set

In the above diagram, “since” is called a descriptive attribute.

A **relationship set** is a collection of similar relationships. For example, if multiple faculty members work in different departments, collecting all such relationships forms a relationship set.

Mapping Cardinalities

Mapping cardinality defines how entities from one set are associated with entities from another. There are four types of cardinalities:

1. **One-to-One Mapping**: Each entity in Set A is associated with exactly one entity in Set B, and vice versa.

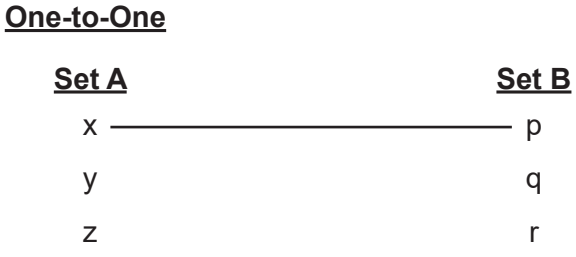


Fig . 2: One-to-one mapping

2. **One-to-Many Mapping**: One entity in Set A can be associated with multiple entities in Set B, but each entity in Set B is associated with only one entity in Set A.

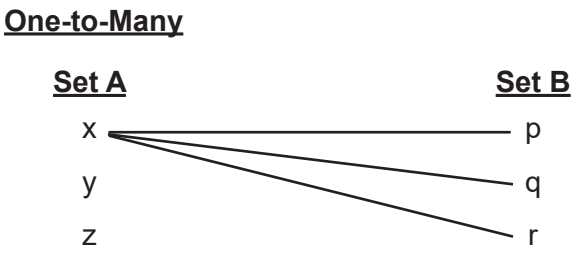


Fig . 3: One-to-many mapping

3. **Many-to-One Mapping:** Many entities in Set A can be associated with one entity in Set B, but each entity in Set A is associated with exactly one entity in Set B.

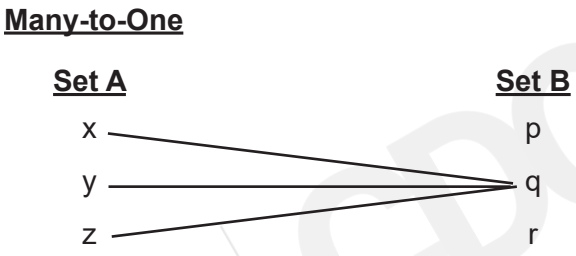


Fig . 4: Many-to-one mapping

4. **Many-to-Many Mapping:** Multiple entities in Set A can be associated with multiple entities in Set B, and vice versa.

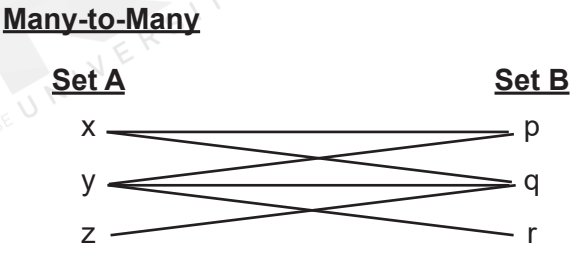


Fig . 5: Many-to-one mapping

Participation Constraints

Participation constraints define whether all or only some entities in an entity are set to participate in a relationship.

Participataion Constraints

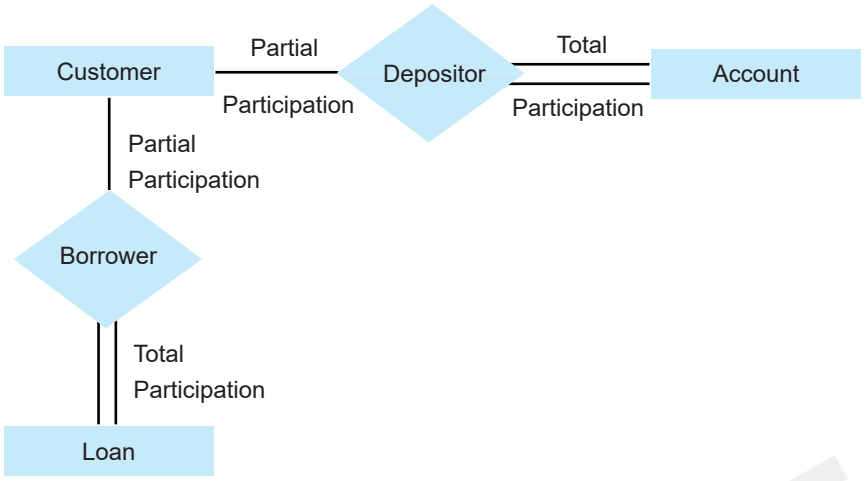


Fig . 6: Participation constraints

- Total Participation: When all entities in an entity set must participate in a relationship, it is called total participation. In ER diagrams, total participation is shown using double lines.
- Partial Participation: When only some entities in an entity set participate in a relationship, it is called partial participation. In ER diagrams, partial participation is shown using single lines.

Strong and Weak Entity Sets

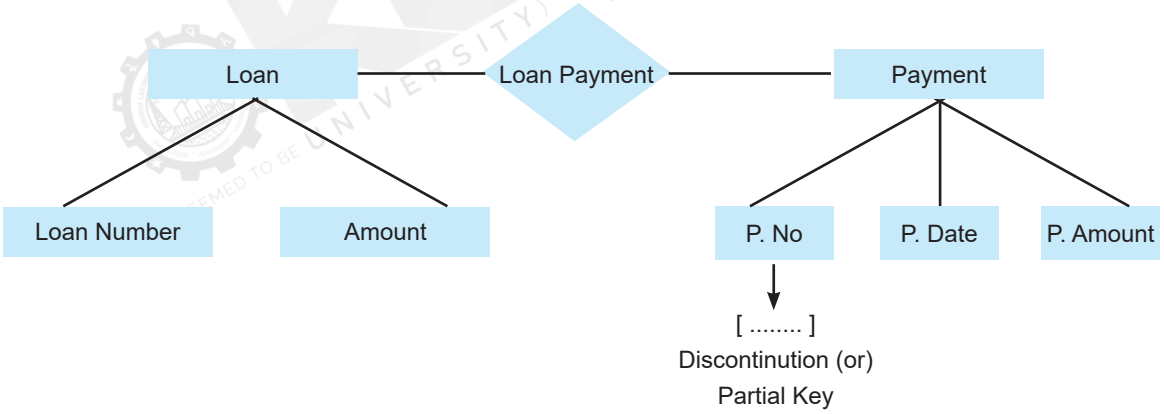


Fig . 7: Strong Entity sets

- **Strong Entity Set:** If an entity set has sufficient attributes to form a primary key, it is called a strong entity set. A primary key uniquely identifies each entity in the set.
- **Weak Entity Set:** A weak entity set lacks attributes to form a primary key. Weak entities rely on a strong entity set for identification. For example, a payment entity might rely on a loan entity because payment numbers could be repeated across different loans. In ER diagrams, weak entity sets are represented using a double rectangle.

Self-Assessment Questions

1. What is the purpose of the Entity-Relationship (ER) model in database design?

   A). To design the user interface for databases
   B). To define entities, attributes, and relationships for structuring data
   C). To create reports from the database
   D). To store unorganised and random data

2. Which of the following is an example of a multi-valued attribute?

   A). Student ID
   B). Faculty Name
   C). Faculty Phone Number
   D). Course Code

3. In an ER diagram, how are entities typically represented?

   A). By ovals
   B). By diamonds
   C). By rectangles
   D). By double lines

4. Which attribute type can be further divided into smaller parts, such as splitting a full name into first, middle, and last names?

   A). Simple Attribute
   B). Composite Attribute
   C). Single-Valued Attribute
   D). Derived Attribute

Self-Assessment Questions

5. What does “total participation” mean in an ER model?

   A). All entities in an entity set must participate in a relationship
   B). Only some entities in an entity set participate in a relationship
   C). Entities do not participate in any relationship
   D). Entities are completely independent

6. Which type of mapping cardinality allows one entity in Set A to be associated with multiple entities in Set B, but each entity in Set B is associated with only one entity in Set A?

   A). One-to-One
   B). Many-to-Many
   C). Many-to-One
   D). One-to-Many

1.2.2 Enhanced Entity-Relationship (EER) Model

As databases become more complex, the basic ER model may not be sufficient. The **Enhanced ER (EER) model** extends the basic ER model by introducing advanced concepts that provide greater flexibility in database design.

Class Hierarchies: Specialisation and Generalisation

Class hierarchies are used to organise entities into a hierarchy based on their properties.

- **Specialisation**: This is a top-down approach where a high-level entity is divided into two or more lower-level entities. The high-level entity is called the "superclass," and the lower-level entities are called "subclasses."

  For example, an "Employee" entity could be specialised into "Officer," "Teller," and "Secretary."

- **Generalisation**: This is a bottom-up approach where two or more lower-level entities are combined to form a high-level entity.

  For example, different employee roles (officer, teller, etc.) could be generalised into an "Employee" entity.

In both specialisation and generalisation, the "Is A" relationship specifies the connection between a superclass and a subclass.

EER Diagrams:

An **EER diagram** expands on the standard ER diagram by adding the following features:

- **Subclasses and Superclasses**: Entities can be divided into subclasses or combined into superclasses.
- **Inheritance**: Subclasses inherit attributes and relationships from their superclass, making it easier to organise and manage data.

Constraints in Specialisation and Generalisation

Two types of constraints are commonly found in specialisation and generalisation:

1. **Covering Constraints**: These constraints ensure that a superclass covers all subclasses. For example, "Officer," "Teller," and "Secretary" are all covered under the "Employee" entity set.

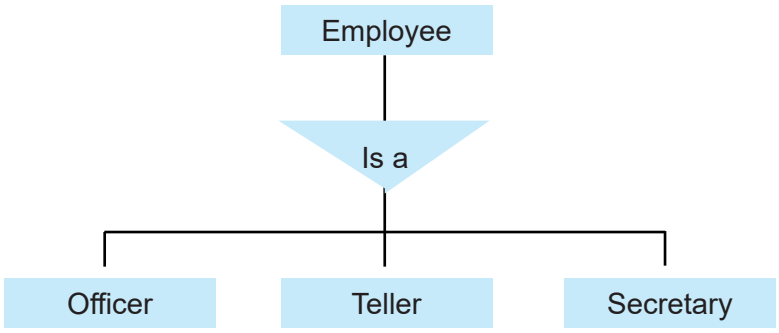


Fig . 8: Covering constraints

2. **Overlapping Constraints:** These constraints allow an entity to belong to multiple subclasses. For example, a senior employee can be an officer and a secretary, meaning the entity overlaps these roles.

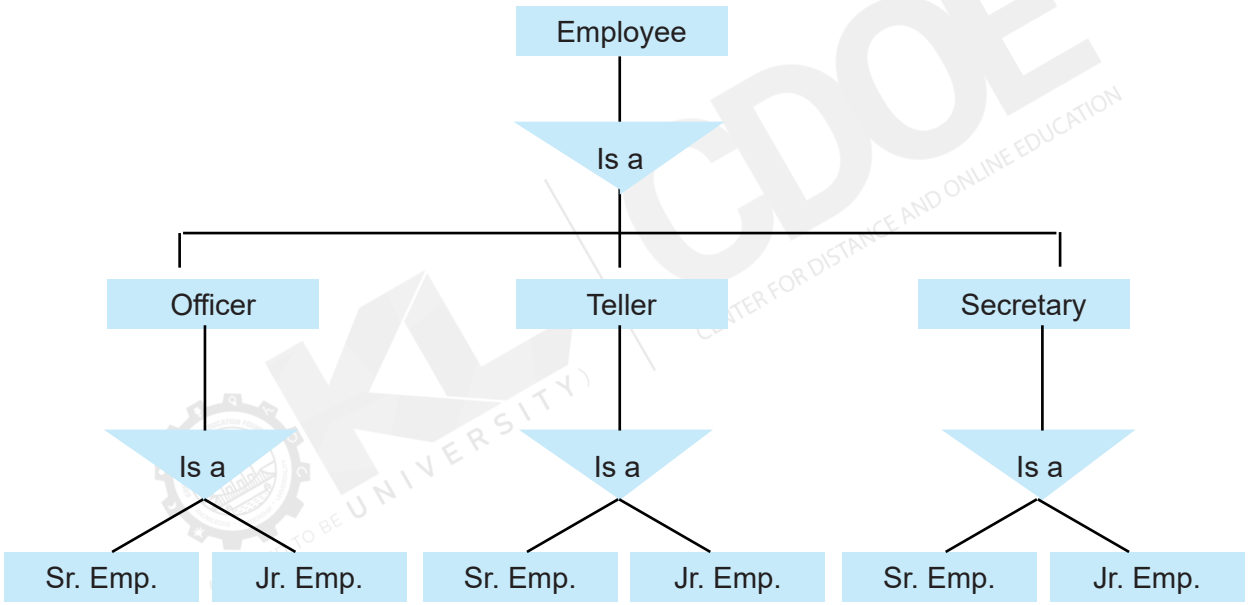


Fig . 9: Overlapping constraints

Aggregation

Aggregation is used when we need to express a relationship between relationships. The basic ER model limits this, but aggregation solves this by treating a group of entities and their relationships as a higher-level entity. This abstraction allows for more complex database designs.

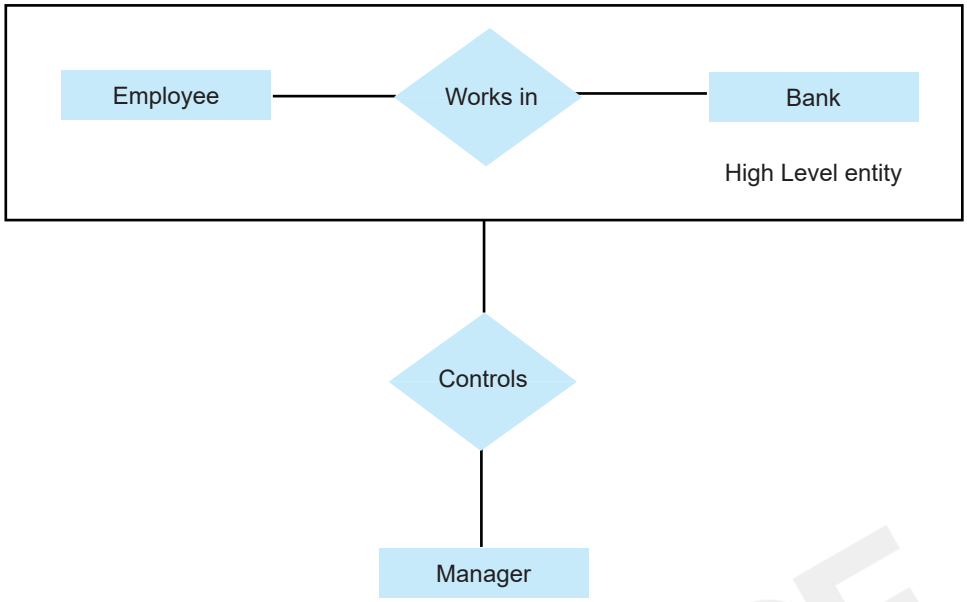


Fig . 10: Aggregation

Exercises:

To practice these concepts, the following exercises can help:

1. **Draw the ER diagram for the university database**.
   - Entities may include students, faculty, departments, and courses.
   - Relationships could include “enrols in” (between students and courses), “teaches” (between faculty and courses), and so on.
2. **Draw the ER diagram for the airline reservation system**.
   - Entities may include passengers, flights, and reservations.
   - Relationships could include “books” (between passengers and reservations) and “operates” (between airlines and flights).
3. **Draw the ER diagram for the library management system**.
   - Entities may include books, members, and librarians.
   - Relationships could include “borrows” (between members and books) and “manages” (between librarians and books).

The ER model is essential for creating database structures by focusing on entities, attributes, and relationships. The model can be extended with more complex features such as class hierarchies (specialisation and generalisation), aggregation, and participation constraints. Advanced models like the EER (Enhanced ER) model build upon these concepts to allow for more comprehensive and intricate database designs, making managing and representing complex data easier.

Exercise:

1. Draw the ER diagram for the university database.

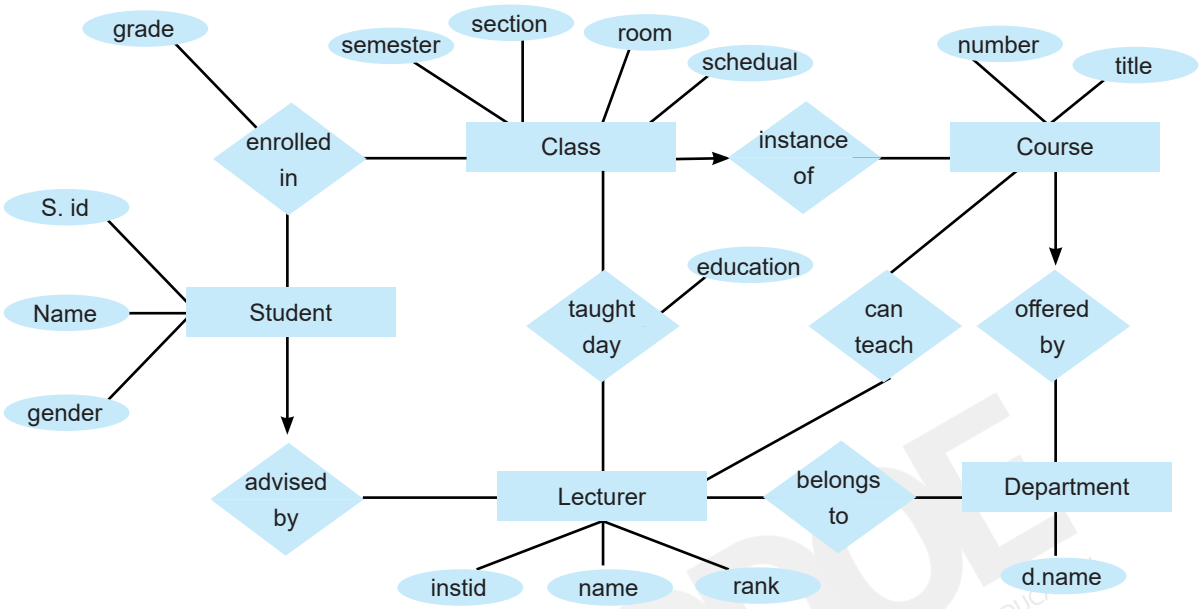


Fig . 11: University database

2. Draw the ER diagram for the airline's reservation system.

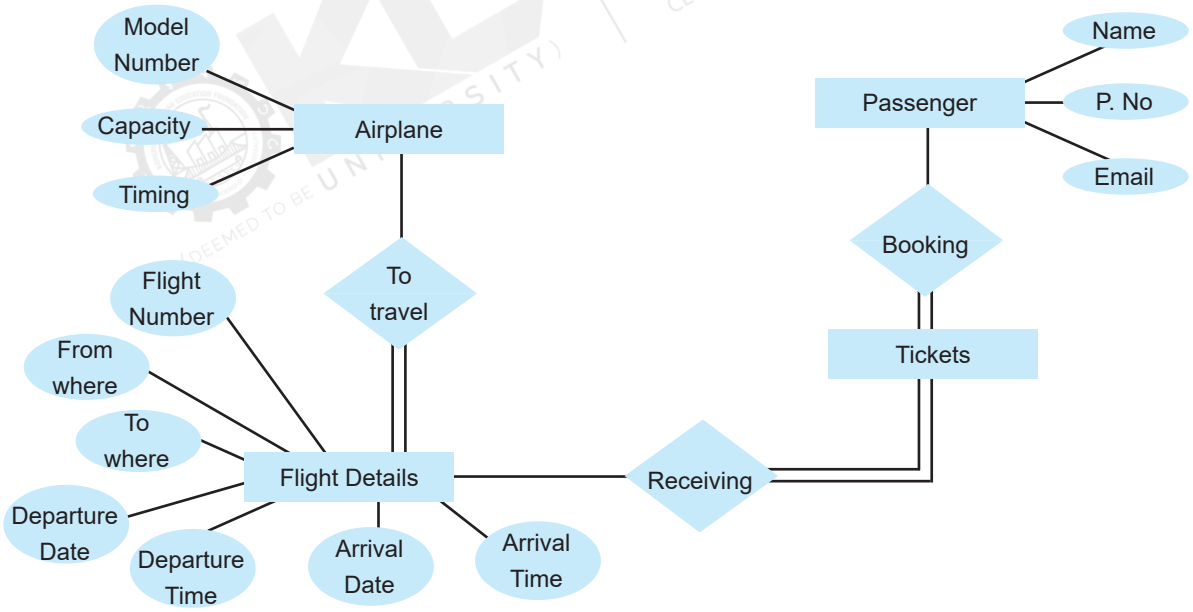


Fig . 12: Airline's reservation system

3. Draw the ER diagram for the library management system

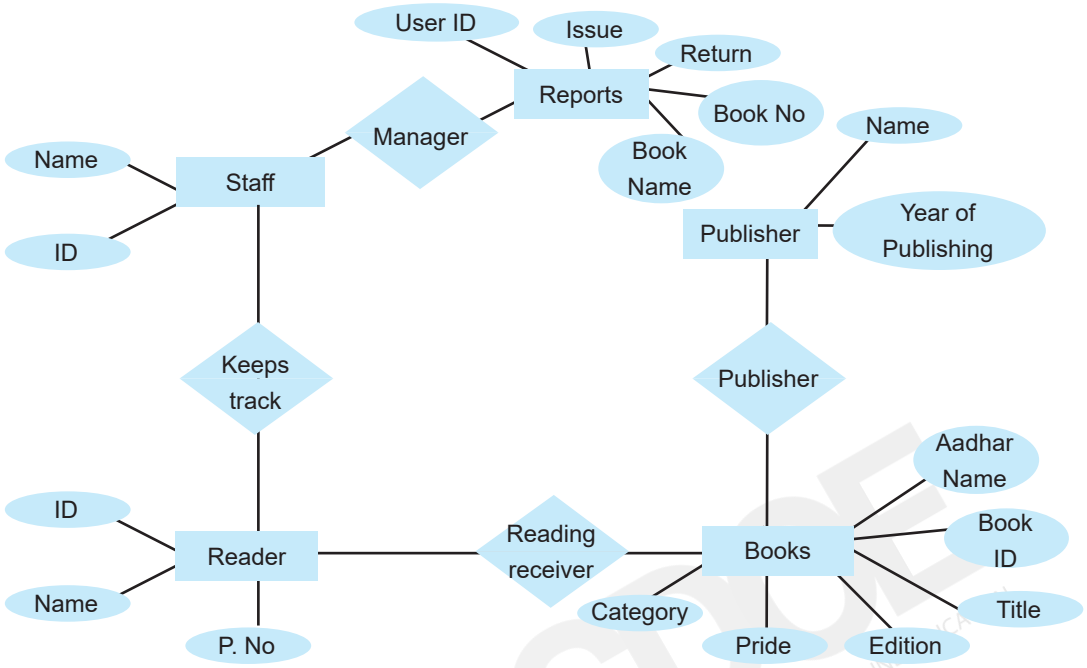


Fig . 13: Library management system

Comprehensive Database Design with ER and EER Models:

The ER and EER models provide a detailed and structured approach to database design, ensuring clarity and efficiency in organising data. The ER model offers a simple and clear method of modelling data through entities, attributes, and relationships. The EER model adds advanced capabilities such as specialisation, generalisation, and aggregation for more complex systems. These tools help design flexible, scalable, and robust databases that accommodate real-world complexities.

The combination of ER and EER models enables database designers to build comprehensive and scalable data structures, ensuring that real-world entities and their interactions are accurately represented and managed in the database.

Self-Assessment Questions

7. What is the main purpose of the Enhanced ER (EER) model in database design?

   A). To create a user-friendly interface for the database
   B). To add advanced features like class hierarchies, specialisation, and aggregation
   C). To simplify the basic ER model
   D). To represent only simple database structures

8. In the EER model, what does specialisation represent?

   A). Combining multiple entities into one
   B). Dividing a high-level entity into two or more lower-level entities
   C). Creating a new database system
   D). Eliminating all subclasses from a hierarchy

9. What is the relationship between a superclass and a subclass in an EER model?

   A). Aggregation
   B). Generalisation
   C). Inheritance
   D). “Is A” relationship

10. Which constraint ensures that a superclass covers all its subclasses?

    A). Overlapping Constraint
    B). Covering Constraints
    C). Participation Constraint
    D). Cardinality Constraint

Self-Assessment Questions

11. What does aggregation allow in an EER model?

A). Defining multiple subclasses
B). Creating simple relationships between entities
C). Expressing a relationship between relationships
D). Combining entities without using relationships

12. What is generalisation in an EER model?

A). A bottom-up approach where lower-level entities are combined into a high-level entity
B). A top-down approach where high-level entities are divided into subclasses
C). A method to create entities without attributes
D). A type of participation constraint

Summary

- The ER model designs databases by defining entities, attributes, and relationships.
- Entities are objects like students or courses and are represented by rectangles in diagrams.
- Attributes are characteristics of entities, shown as ovals in ER diagrams.
- Attributes can be simple, composite, single-valued, multi-valued, or derived.
- Entity sets are collections of similar entities, like all students in a university.
- Relationships link two or more entities, like diamonds in ER diagrams.
- Mapping cardinalities describes how entities from two sets relate (one-to-one, one-to-many, etc.).
- Total participation means all entities in a set must be part of a relationship; partial participation means only some are involved.
- Strong entity sets have primary keys, while weak entity sets depend on other entities for identification.
- Aggregation is used to simplify complex relationships by treating groups of entities and relationships as one entity.

Terminal Questions

1. What are the main components of the ER model, and how are they represented in an ER diagram?
2. Explain the difference between simple and composite attributes with examples?
3. What is the difference between total and partial participation in an ER model?
4. How are weak entity sets different from strong entity sets?
5. What is the purpose of specialisation and generalisation in database design?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	B
2	C
3	C
4	B
5	A
6	D
7	B
8	B
9	D
10	B
11	C
12	A

Activity

Activity type: Online **Duration: 1 Hour**

Create an ER diagram using online tools like Lucidchart or Draw.io.

Glossary

- **Entity:** A real-world object or concept that can be identified and described.
- **Attribute:** A property or characteristic of an entity.
- **Mapping Cardinality:** The number of associations between entities in one set and another.
- **Participation Constraints:** Rules that determine whether all or some entities participate in a relationship.
- **Specialisation:** Dividing a high-level entity into lower-level entities.
- **Aggregation:** Grouping entities and relationships to form a higher-level entity for modelling complex relationships.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **ER Model:** https://www.geeksforgeeks.org/introduction-of-er-model/
- **EER Model:** https://www.scaler.com/topics/eer-model-in-dbms/

Video Links

Topic	Link
Entity-Relationship Model	https://www.youtube.com/watch?v=wOD02sezmX8
Generalisation, Specialisation, Aggregation	https://www.youtube.com/watch?v=20WU801_KXw

Image Credits

Fig. 1: Relationship set	Self-generated
Fig. 2: One-to-one mapping	Self-generated
Fig. 3: One-to-many mapping	Self-generated
Fig. 4: Many-to-one mapping	Self-generated
Fig. 5: Many-to-one mapping	Self-generated
Fig. 6: Participation constraints	Self-generated
Fig. 7: Strong Entity sets	Self-generated
Fig. 8: Covering constraints	Self-generated
Fig. 9: Overlapping constraints	Self-generated
Fig. 10: Aggregation	Self-generated
Fig. 11: University database	Self-generated
Fig. 12: Airline's reservation system	Self-generated
Fig. 13: Library management system	Self-generated

Keywords

- Entity Set
- Relationship
- ER Diagram
- Total Participation
- Weak Entity
- Generalisation

DATABASE SYSTEMS

MODULE 2

The Relational Model and SQL

Module Description

This module focuses on the foundational and advanced aspects of the Relational Model and SQL, which are essential for effective database management. Students will explore the core principles of the Relational Model, focusing on essential concepts like constraints, schema design, and conversion from Entity-Relationship (ER) diagrams to relational schemas. It covers SQL's data definition (DDL) and data manipulation (DML) commands, guiding students in creating tables, defining data types, enforcing constraints, and executing complex queries.

The module also introduces relational algebra operators, providing a theoretical basis for query formulation and optimisation. Students will examine comprehensive database design principles and learn strategies for structuring efficient, flexible databases that meet user requirements.

The course emphasises advanced SQL techniques to build practical skills, including creating views, using compound statements, and developing user-defined functions and stored procedures. Topics like cursors and triggers are covered to give students a hands-on interpolating of automation and data control within SQL environments. By the end of the module, participants will have a solid grasp of relational database systems and be equipped with the knowledge and skills needed to design, implement, and manage efficient and secure databases.

This module is divided into two units.

Unit 2.1 The Relational Model
Unit 2.2 SQL and Advanced SQL

MODULE 2

The Relational Model and SQL

Unit 1

The Relational Model

Unit Table of Contents

Unit 2.1 The Relational Model

Aim

To interpolate the Relational Model, including schema design and ER conversion, and gain proficiency in SQL for data definition, manipulation, and complex queries.

Instructional Objectives

This unit intends to:

- Define key concepts of the Relational Model and SQL commands
- Identify SQL syntax for table creation and data constraints
- List types of SQL statements such as DDL, DML, DCL, and DQL

Learning Outcomes

Upon completion of the unit, you will be able to:

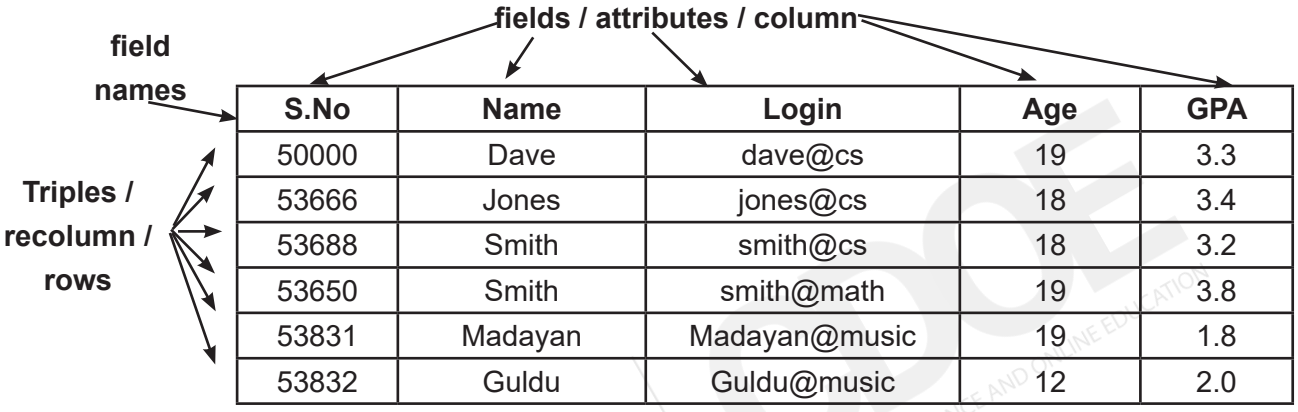
- Explain the basics of the Relational Model and schema design
- Describe the steps involved in converting an ER model to the Relational Model
- Demonstrate SQL use for creating tables and performing queries

2.1.1 Fundamentals of the Relational Model

Relational Model

The relational model is a data structure where data is organised in tables, known as “relations,” making it one of the most popular database models today. First introduced by E. F. Codd in 1970, this model underpins many modern databases due to its structured approach, which offers a straightforward way to manage data with rows and columns.

Antiquity constraints over a velation

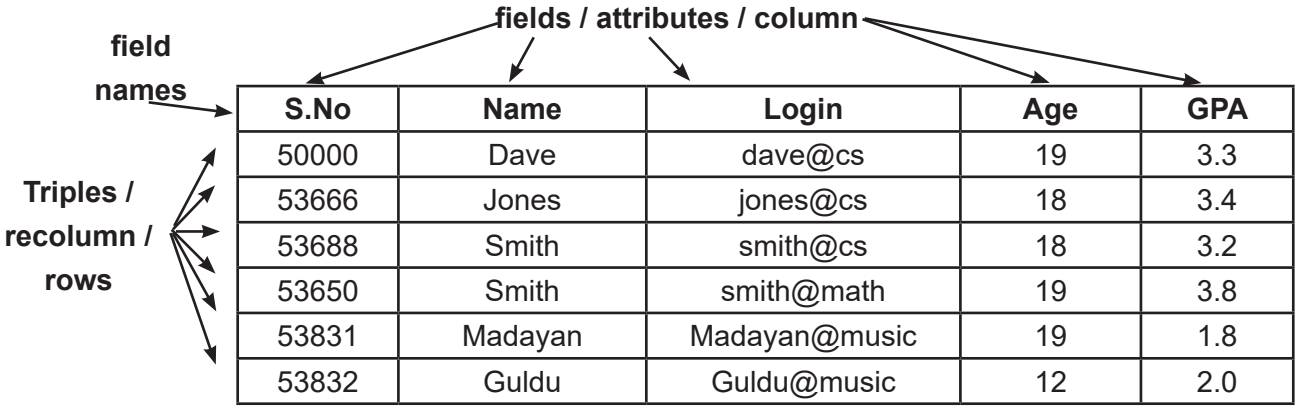


S.No	Name	Login	Age	GPA
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	Madayan@music	19	1.8
53832	Guldu	Guldu@music	12	2.0

Fig. 1: Relational Model

A relation in this model is defined through several key characteristics:

- Relational Schema: This specifies the structure of each table, detailing:
  - The name of each relation (or table).
  - The name of each field (or column).
  - The data type or domain of each field, specifying what kind of data can be stored (e.g., integer, string, date).
- Domain Constraints: These constraints ensure that the values in each field adhere to their respective domains. For instance, a field defined to store dates will only accept valid date formats.
- Degree of Relation: This term refers to the number of columns in a table. For example, a relation with five columns has a degree of five.
- Cardinality: This represents the total number of rows within a relation, such as a table with 100 rows having a cardinality of 100.



S.No	Name	Login	Age	GPA
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	Madayan@music	19	1.8
53832	Guldu	Guldu@music	12	2.0

Fig. 2: Cardinality

2. Integrity Constraints in a Relational Model

Integrity constraints are rules to ensure data accuracy and consistency within a database. In the relational model, three primary constraints are essential:

- **Super Key:** A super key is a set of attributes in a table that can uniquely identify each record or row.

  For example, if a table has attributes like sid, login, and age, any combination that includes unique values (like <sid, login> or <sid, login, age>) can be considered a super key.

- **Candidate Key:** Candidate keys are specific sets of attributes within a super key that uniquely identify records and contain unique values. In many cases, candidate keys allow null values.

  For instance, if <sid> and <login> can uniquely identify records in a table, they are candidate keys.

- **Primary Key:** This is a candidate key chosen to uniquely identify records in a table, ensuring there are no duplicates or null values.
  For instance, <sid> might be chosen as the primary key for a student table, meaning each sid is unique and not null.

3. Specifying a Candidate Key in a Relation

Candidate keys are crucial in maintaining data uniqueness. Candidate keys signify fields that uniquely identify records when defining a relation. Typically, a candidate key does not allow duplicate values but can allow null values under specific conditions.

4. Foreign Key (Referential Key)

Foreign keys link information in one table to another, creating relational connections between tables. A foreign key in a child table points to a primary key in a parent table, ensuring referential integrity. For example, in a database with a student table and an enrolled table, the enrolled

table may contain a student field, which is a foreign key referencing the student table's primary key. This linkage restricts operations that could violate relational dependencies (e.g., deleting a student record without removing all related enrolled records).

C.Id	Foreign Key grade	Stu id
Caronatic 101	C	53831
Reggac 203	B	53832
Topology 1120	A	53650
History 105	B	53666

Enrolled (Refferencing Relation)

↓

Child Table

S.Id	Name	Login	Age	GPA
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	Madayan@ music	11	1.8
53832	Guldu	Guldu@music	12	2.0

Students (Refference Relation)

↓

Parent Table

Fig. 3: Foreign Key

5. Specifying Foreign Key Constraints in SQL

In SQL, foreign key constraints enforce relationships between tables. When a foreign key constraint is added to a column, it references a primary key in another table, meaning any inserted or updated data must align with the data in the referenced table to maintain consistency.

6. Check Constraints

Check constraints ensure that data entered into a column meets specific criteria, adding an additional layer of validation. For example, if a salary field has a check constraint requiring values greater than 50,000, any insert or update operation that violates this condition will be rejected.

7. Logical Database Design

This design phase involves translating a conceptual model, often an Entity-Relationship (ER) model, into a logical structure suitable for relational databases.

Entity Set to Relation

Each entity set in the ER model corresponds to a relation (table) in the database. For instance:

- The entity name becomes the table name.
- Entity attributes become table columns.
- The primary key of the entity is used as the primary key of the table.

Relationship Set to Relation

Relationship sets in the ER model are also converted into relations, where:

- The relationship set name becomes the table name.
- Attributes from participating entities and any relationship-specific attributes become table columns.
- The primary keys of participating entities become part of the primary key and/or foreign key.

Weak Entity Set to Relation

A weak entity set relies on a strong entity set for its existence, meaning it has no primary key on its own. When converting weak entities:

- The relation is named after both the weak entity and its associated strong entity.
- The primary key of the strong entity and a partial key of the weak entity act as the composite primary key in the relation.
- The strong entity's primary key serves as a foreign key.

Aggregation to Relation

Aggregations are used to express relationships among relationships. When converting aggregations:

- The aggregation name becomes the relation name.
- Attributes from entities in the aggregation are added as columns in the table.
- The primary keys of these entities serve as the table's primary key and/or foreign key.

8. Enforcing Integrity Constraints

Integrity constraints are declared when relations are created and are enforced automatically by the database management system (DBMS) whenever data is modified. Every time a SQL statement is executed, the DBMS checks if the integrity constraints are satisfied. This automatic verification prevents data inconsistency and preserves the relational model's accuracy and reliability.

In a structured approach to defining tables, constraints, and relationships, the relational model forms the foundation for most modern databases. Proper implementation of these principles and constraints ensures data is accurate, consistent, and efficiently managed.

Self-Assessment Questions

1. Which of the following best describes a relational model?

   A). A model where data is represented in a tree structure
   B). A model where data is organised into tables with rows and columns
   C). A model where data is represented in the form of XML
   D). A model that uses a hierarchical structure to store data

2. In the relational model, what does “cardinality” refer to?

   A). The number of columns in a table
   B). The total number of tables in the database
   C). The number of rows in a table
   D). The data type of a field

3. Which constraint ensures that each record in a table can be uniquely identified?

   A). Foreign Key
   B). Domain Constraint
   C). Primary Key
   D). Check Constraint

4. what is a “foreign key” in a relational database?

   A). A field in a table that is linked to the primary key of another table
   B). A key that uniquely identifies a record within the same table
   C). A key that allows duplicate values within a table
   D). A field that contains non-unique data values

5. What is the purpose of a check constraint in a relational model?

   A). To restrict values entered in a column based on specific criteria
   B). To prevent duplicate records in a table
   C). To ensure each value in a column is unique
   D). To define relationships between tables

6. What is the “degree of relation” in the context of the relational model?

   A). The number of tables in a database
   B). The number of columns in a table
   C). The number of rows in a table
   D). The maximum number of relations in a database

2.1.2 SQL Basics

Introduction to SQL

Structured Query Language (SQL) is a powerful language for interacting with relational databases. It allows users to create, manage, manipulate, and retrieve data in a structured, organised way. Initially developed by IBM in the 1970s, SQL has become an essential tool for data handling across various database systems.

SQL's non-procedural nature allows users to specify what data they want to retrieve without detailing how it should be retrieved, which makes it user-friendly. The syntax is intuitive and, like English, helps beginners and experts alike easily execute complex database operations.

Characteristics of SQL

Non-Procedural Language: SQL allows users to specify what data is needed without detailing the method to fetch it. This abstraction simplifies database interaction.

User-Friendly Syntax: With commands that resemble English, SQL is intuitive, making it accessible for those new to programming.

Small Command Set: SQL has a relatively compact set of commands, allowing users to perform various operations with minimal instruction.

Flexible Formatting: SQL statements can be written across multiple lines or in a single line, allowing flexibility in coding style.

Data Types in SQL

SQL uses various data types to define the type of data a column can hold, which helps in managing the database efficiently. These types include:

I. Numeric Data Types

Numeric types handle integers and floating-point numbers. Some common numeric data types include:

The following are the different types in SQL:

DATATYPE	FROM	TO
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,648
Smallint	-32,768	32,768
Tinyint	0	225
Bit	0	1

II. Date and Time Data Types

These handle date, time, and combined date-time formats.

Datatypes	From	To
Date time	JAN 1,1753	DEC 31,9999
Small date time	JAN1,1900	JAN 6,2079
Time	Stores time of day like	12:30PM

III. Character String Data Types

Used for storing text.

Datatypes	Description
Char	Max length of 8000 characters (fixed length)
Varchar	Max length of 8000 characters (variable length)
Varchar(max)	Max length of 231 characters (variable length)(SQL server 2085 only)
Text	Max length of 2,147,483,647characters (variable length)

IV. Binary Data Types

Stores binary data, such as images or files.

Datatypes	Description
Binary	Max length of 8000 datatypes (fixed length binary data)
Varbinary	Max length of 8000 datatypes (variable length of binary data)
Varbinary(num)	Max length of 231 bytes (SQL server only 2085 only) (variable length binary data)
Image	Max length of 2,147,483,647 bytes (variable length binary data)

V. Appropriate Numeric Data Types

For floating-point values and real numbers:

Datatypes	From	To
Float	-1.79E+308	-1.79E+308
Real	-3.40E to 38	-3.40E to 38

Types of SQL Commands

SQL commands are grouped into five main categories:

1. Data Definition Language (DDL)

DDL commands define, modify, and remove database structures. Key commands include:

There are classified into 4 types:

1. Create
2. Alter
3. Drop
4. Truncate

The main purpose of DDL commands is to create an object in the database, modify the data from the database and delete the object from the database.

(i) Create:

Using the create command, we can create 2 types of tables.

Syntax: for creating parent table

SEMPID	SENAME	SSAL	SAGE

Create table name (COL1 DATATYPE, CLO2 DATATYPE……. COL N DATATYPE, unique (COL1, COL2) PRIMARY KEY(COL1));

Syntax for creating child table:
Create table name (COL1, COL2, COL3..COLN)
Select COL1.COL2, COLN from the parent table name

Example: Create Table Employee () Emp Id Number (4), E name Char (10), Sal Float (5,2), E age Number (2), Primary Key (Emp Id));

EMP ID	E NAME	SAL	E AGE

SYNTAX: for creating a child table:

Create Table E name (COL1, COL2, COL3……. COL N)

Example: Create Table Senior employee (S emp id, S name, S Sal, S age) As Select Emp id, E name, Sal, Age, From Employee.

SEMPID	SENAME	SSAL	SAGE

Fig: senior employee

(ii) Alter:

This command is mainly used to perform 4 major operations.

Syntax: for adding a new column to the existing table:

```
Alter Table Name Add (Column Datatype).
```

Example:

Alter table employee add (column phone number (10);

EMPID	ENAME	SAL	EAGE	PHONENUNMBER

Employee.

Syntax: for renaming a column name:

```
Alter Table Name Rename Column
Old Column Name to New Column Name.
```

Example: ALTER table employee rename column
EMPID to SSN:

SSN	ENAME	SAL	EAGE	PHONE NUMBER

Syntax for modifying a column datatype:

```
Alter Table Name: Modify Column Name Datatype:
```

Example: Alter Table Employee Modify Column Sal Number (10):

Syntax for dropping a particular column:

```
Alter Table Name
Drop Column Name
```

Example: Alter table employee
Drop column phone number

SSIN	ENAME	SAL	EAGE

3. Truncate:

IT IS mainly used to delete all the rows from a particular table, but the structure of the table will appear in the database.

Syntax:

```
Truncate Table Name:
```

O/P: TABLE truncated

4. DROP:

It is mainly used to delete all the rows from the table and destroy the tab structure from the database.

Syntax:

```
Drop table name
```

Example: Drop table employee.
O/p: table dropped.
`c2

2. Data Manipulation Language (DML)

DML commands manage data within tables, allowing users to insert, update, and delete records:

Insert: Adds new records.
Syntax

```
Insert into table name values(1011,'smith',50,000,35)
```

O/p: (1row inserted)

Sno	E name	Salary	E age	
1011	Smith	50000	35	
1012	Raju	35000	36	
1013	Ram	35000	40	
1014	Sandhya	25000	26	

Update: Modifies existing records.

Syntax:

To update a single column (without condition)

```
Update table name
    SET column name=arthexp
```

Example: update table employee

SET SAL=sal+5000.

O/p: updated successfully.

With condition:

```
Update table employee
    SET SAL=sal+5000
    Where empid=1011
```

For multiple columns (with condition):

```
Update table name
```

SET column1=arthexp, column2=arthexp….
Where clause.

Example: update table employee
SET SAL=SAL+5000, SSN =SSN+1.
O/p: updated successfully
With condition:
Update table employee
SET SAL=SAL+5000, SSN =SSN+1.
Where empid=1012.

O/p: updated successfully

Delete:

This command is mainly used to delete the data from the table.
It can also be specified within the 'where clause'.

Syntax:

```
Delete from table name
Where clause:
```

Example: 1. Delete table employee:
2. Delete table employee
where ssn=1102

3. Data Control Language (DCL)
DCL commands provide database access control:
These are mainly classified into:
(i) grant command (ii) revoke command

Grant: Gives a user access privilege.

(ı) Grant: This command is mainly used by users to access the database.

Syntax:

```
grant username
```

grant 2300520111

Revoke: Removes user access privileges.

Syntax:

```
Revoke username
```

Revoke 2300242301

4. Data Query Language (DQL)
The primary DQL command, Select, retrieves data from the database:
Syntax

```
Select [distinct] select list
From from-list
Were qualification
```

Every query must have a select class which specifies the columns to be retrieved.
The form class will specify the name of the table (or) group of table names.
'Where clause' will specify a condition on a given relation.

Emp. ID	E. Name	E. Sal	E. Age	Commission
1011	Turner	6000	22	200
1012	Mark	5000	23	150
1013	Blake	10000	25	300
1014	Mark	7500	21	100

Fig. 4: Data Query Language (DQL)

Different forms of using a select command:

Form 1:
To display all the column names.
Example: select distinct*from employee;
O/p: same table as above

Form 2: to display only particular column names (without where clause)

```
Example:  select distinct Emp id, E name
          from employee.
```

O/p:

Empid	Ename
1011	Tumer
1012	Mark
1013	Blake
1014	Mark

To display only particular column names (with where clause)

```
Example 2: select a distinct E name
        from employee
        where salary>8000.
```

O/p:

Emp. Name
Blake

Form2:
Implementing a like clause within a where clause

```
        select a distinct E name
         from employee
        where E name like 'm%'.
```

O/p: mark

Where clause allows records filtering, SQL supports complex querying through various select command forms, including distinct, group by, and having clauses for refined data retrieval.

5. Transaction Control Language (TCL)
TCL commands manage transactions to ensure data consistency:

Commit: Finalises a transaction and saves all changes.

Syntax: commit

This command is used to store the temporarily

Save point save point name

Syntax:

Rollback: Reverts to the previous state before changes.

Syntax:

Rollback to save point name

Union intersects and except clauses:

Sid	S.Name	Rating	Age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.5
64	Horatio	1	35.5
71	Zorba	10	16.0
74	Horatio	9	35.5
85	Art	3	25.5
95	Bob	3	63.3

An instance s3 of sailors

Sid	Bid	Day
22	101	10/10/98
22	102	10/10/98
22	103	10/08/98
22	104	10/07/98
31	102	11/10/98
31	103	10/06/98
31	104	11/12/98
64	101	09/05/98
64	102	09/08/98
74	103	09/08/98

An instance of r2 reserves.

Complex Queries with SQL

Nested Queries

Nested (or sub) queries are queries within other queries, allowing complex data selection.

Independent Nested Query: The subquery operates independently.

Correlated Nested Query: The subquery depends on data from the outer query.

SELECT Name FROM Employee WHERE employee id IN (SELECT employee id FROM Sales WHERE Amount > 5000).

Joins in SQL

Emp ID	Ename	Age	Dept.no
1011	Smith	45	10
1012	Allen	33	20
1013	Wearld	55	30
1014	Gones	25	10
1015	Martin	35	20
1016	Blake	35	30
1017	Raju	16	20
1018	Scort	35	40
1019	Turner	25	10
1020	Admas	40	30
1021	Games	28	40
1022	Ford	30	20
1023	Rives	26	40
1024	Dustin	42	30
1025	Andy	32	40

Dept. No	Dept. Name	Location
10	CSE	Newyork
20	ECE	Dallas
30	IT	London
40	EEE	Mexico

Fig. 5: Joins in SQL

In SQL, joins are mainly used to extract information from two or more tables.

Joins retrieve data across multiple tables:

- **Inner Join:** Returns matching records between tables.
- **Left Join:** Returns all records from the left table, with matching records from the right.
- **Right Join:** Returns all records from the right table, with matching records from the left.
- **Full Join:** Returns all records when a match is in either table.

Example:

Display the names of the employees and their salaries, where the employees' names must be in alphabetical order.

```
Select a distinct name, Sal
From employee
Order by name.
```

O/p:

Ename	Sal
A	18000
B	22000
C	16000
X	25000
Y	18000

Example 2: Display the names and sal of an employee where names must be in alphabetical order, and Sal must be greater than 18,000.

```
Select distinct E name, Sal
From employee
Where Sal>18000
Order by E name.
```

O/p:

Ename	Sal
B	22000
X	25000

Example 3: Display the names and sales of an employee, where employee names must be in reverse alphabetical order.

```
Select a distinct E name, Sal
From employee
Order by E name desc.
```

O/p:

Ename	Sal
Y	18000
X	25000
C	16000
B	22000
A	18000

Example 4: Display the names and salaries of employees where names must be in alphabetical order, and salaries must be in increasing order.

```
Select a distinct E name, Sal
From employee
Order by E name, Sal.
```

O/p:

Ename	Sal
A	18000
B	22000
C	16000
X	25000
Y	18000

Aggregation Operators

SQL supports aggregation functions to summarise data:

Count: Counts the number of rows or distinct values.
Sum: Adds up all values in a column.
Avg: Calculates the average of values.
Max/Min: Find the highest or lowest values.

Example:

SELECT dept id, COUNT (employee id) AS employee count FROM Employee GROUP BY dept id;

SQL Operators

SQL operators help compare or calculate data:

Set Comparison Operators: =, <>, >, <, etc., for comparing values.
Aggregation Operators: Aggregate data across groups, e.g., SUM (), AVG ().

SQL remains a fundamental tool for relational database management, with broad applications in data storage, retrieval, and manipulation. By mastering SQL, users can efficiently manage large datasets and build applications that depend on robust data-handling capabilities.

Self-Assessment Questions

7. What type of language is SQL considered to be?

   A). Procedural
   B). Object-Oriented
   C). Non-Procedural
   D). Functional

8. Which SQL command is used to create a new table in a database?

   A). ALTER
   B). CREATE
   C). INSERT
   D). UPDATE

9. Which of the following is NOT a data type commonly used in SQL?

   A). INTEGER
   B). VARCHAR
   C). IMAGE
   D). BOOLEAN

10. What is the primary purpose of the SELECT statement in SQL?

    A). To delete records from a table
    B). To retrieve data from a database
    C). To create a new database
    D). To grant user access

11. What does the ALTER command do in SQL?

    A). Adds or modifies table structure
    B). Deletes all rows from a table
    C). Commits a transaction
    D). Retrieves specific rows from a table

12. Which SQL commands are classified under Data Manipulation Language (DML)?

    A). TRUNCATE
    B). GRANT
    C). UPDATE
    D). CREATE

Summary

- The relational model organises data in tables, each with rows (records) and columns (attributes).
- It defines a table's structure, including table names, columns, and data types.
- Domain constraints ensure each column contains the correct data type, like dates only in a date field.
- Keys are crucial for organising data; primary keys uniquely identify rows without allowing nulls, while candidate keys can have unique values even with nulls.
- Foreign keys connect tables by linking a column in one table to the primary key in another.
- Integrity constraints help keep data accurate by enforcing rules on values in the database.
- Logical database design involves translating entity-relationship (ER) models into table structures and mapping entities and relationships to tables.
- SQL (Structured Query Language) creates, manages, and retrieves data from relational databases.
- It includes command types like DDL (for creating structure), DML (for manipulating data), DCL (for access control), DQL (for querying data), and TCL (for transactions).
- Complex queries use joins to link data across tables and nested queries to answer complex questions using one query inside another.

Terminal Questions

1. What is the primary purpose of a relational schema in a database?
2. How do domain constraints ensure data integrity within a relational model?
3. Why are integrity constraints essential for relational databases?
4. What is the difference between DDL and DML commands in SQL?
5. How does a check constraint work, and why is it used?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	B
2	C
3	C
4	A
5	A
6	B
7	C
8	B
9	D
10	B
11	A
12	C

Activity

Activity type: Online **Duration: 1 Hour**

Design a basic database schema for a hypothetical student enrollment system, including student, course, and enrollment tables. Specify primary and foreign keys.

Glossary

- **Relational Model:** A data model organising data in tables (relations) with rows and columns.
- **Schema:** Defines the structure of a database, detailing tables, columns, and data types.
- **Domain Constraint:** Limits the data type of each field (e.g., text, integer).
- **Foreign Key:** Field in one table that refers to the primary key in another, linking data across tables.
- **Candidate Key:** Unique identifier for records, among which one is chosen as the primary key.
- **SQL:** Structured Query Language used for managing and querying relational databases.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **Relational Model:** https://www.geeksforgeeks.org/relational-model-in-dbms/
- **SQL:** https://www.geeksforgeeks.org/structured-query-language/

Video Links

Topic	Link
Relational Model Constraints	https://www.youtube.com/watch?v=uPOGPL2C0_8
What is SQL	https://www.youtube.com/watch?v=5L7Tp-WkHw-o&t=2s

Image Credits

Fig. 1: Relational Model	Self-generated
Fig. 2: Cardinality	Self-generated
Fig. 3: Foreign Key	Self-generated
Fig. 4: Data Query Language (DQL)	Self-generated
Fig. 5: Joins in SQL	Self-generated

Keywords

- Domain Constraints
- Structured Query Language
- Primary Key
- Integrity Constraints
- Nested Queries
- Alter

MODULE 2

The Relational Model and SQL

Unit 2

SQL and Advanced SQL

Unit Table of Contents

Unit 2.2 SQL and Advanced SQL

Aim

To explore relational algebra operators and comprehensive database design guidelines. Emphasis is placed on practical skills in views, compound statements, user-defined functions, procedures, cursors, and triggers.

Instructional Objectives

This unit intends to:

- Define relational algebra operators such as selection, projection, and union
- List the key principles of effective database design guidelines
- Identify the components and purposes of views in SQL

Learning Outcomes

Upon completion of the unit, you will be able to:

- Explain the process of data retrieval using relational algebra operators
- Describe how database design guidelines minimise redundancy and improve data organisation
- Illustrate the creation and use of views to simplify complex SQL queries

2.2.1 Relational Algebra

Introduction to Relational Algebra

Relational algebra is a procedural query language that helps retrieve data from a database. Instead of asking, “What information do I want?” Relational algebra specifies how to get that information, outlining the steps needed to reach the result. It is a sequence of operations applied to tables (relations) to extract useful data.

It is a “procedural query language,” specifying what information is required and how it is obtained.

Fig. 1: Relational Algebra

Basic Operations of Relational Algebra

1. **Selection Operator (σ)**
    - **Purpose**: To filter rows based on a condition.
    - **Syntax**: σ_condition (Relation)
    - **Example**: Suppose we have a table Sailors with columns Sid, Sname, Rating, and Age. To get rows where Rating > 8, we use the selection operator, returning only sailors with high ratings.
    - **Output**: Rows where Rating is greater than 8.

2. **Projection Operator (π)**
    - **Purpose**: To select specific columns from a relation.
    - **Syntax**: π_columns (Relation)
    - **Example**: To see only the Sid and Sname columns from the Sailors table, we use the projection operator to exclude other columns.
    - **Output**: Only selected columns, like Sid and Sname, from the table.

3. **Set Operations**
    - Operations used to combine or compare data between two tables that have compatible structures.
    - **Union (U)**: Combines two tables, keeping unique rows from both.
        - **Example**: S1 U S2 merges two tables, S1 and S2, keeping only distinct rows. This requires both tables to have identical columns.
    - **Intersection (∩)**: Returns rows common to both tables.
        - **Example**: S1 ∩ S2 shows only rows that exist in both S1 and S2.
    - **Difference (-)**: Shows rows in one table but not the other.
        - **Example**: S1 - S2 displays rows in S1 that aren't found in S2.

4. **Cartesian Product (×)**
    - **Purpose**: Combining every row of one table with every row of another is useful in joining data.
    - **Syntax**: Relation1 × Relation2
    - **Example**: If Sailors and Boats tables are combined, each sailor is paired with each boat, creating a large table with all possible combinations.

5. **Division Operator (÷)**
    - **Purpose**: To find entries in one table (relation A) that match every entry in another table (relation B).
    - **Example**: Given a table A with columns x and y and a table B with only column y, division returns only those x values in A where all y values from B are present.

(a)selection:

Sid	S name	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0
Sid	S name	Rating	Age
28	Yuppy	9	35.0
31	Lubber	8	55.0
44	Guppy	5	35.0
58	Rusty	10	35.0

It's mainly used to extract the desired rows from the relation.
It is denoted as (pic)
Where: selection operator
predictable /condition
relation
For example, suppose we want to extract the rows from relation s2 where the rating is >8.
(pic) (s2);
(rating >8)

O/p:

Sid	S name	Rating	Age
28	Yuppy	10	35.0
58	Rusty	9	35.0

(b) projection operator:
It is mainly used to extract the desired columns from the given relation. It is denoted as:
where:
(pic of formula) projection operator
name of columns
name of relation

E xample: (pic)
O/p:

Rating
9
8
5
10

E xample:

Sid	Sname
28	Yuppy
31	Lubber
44	Guppy
58	Rusty

(c) set operations:
The following are the standard set operations which are available in relational algebra .

(ι) Union: let , s1 and s2 be two relations .
Then , s1 u s2 return a relation containing all the distinct rows that are present in either s1 or s2.

Mathematically it is denoted as:
(pic)

E xample: s1 u s2:

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0
28	Yuppy	9	35.0
44	Guppy	5	35.0

To perform a union operation on two tables, the two tables must satisfy the property union compatibility.
This means the 2 relationships must contain the same columns taken in order: left to right or right to left.

Intersection:

let s1 and s2 be two relations then s1 n s2 returns a relation containing only common tuples / rows from s1 and s2.
Both relations must satisfy union compatibility property to perform an intersection operation.
Mathematically, it is denoted as:

(pic)

E xample.:

S. Id	S. Name	Rating	Age

Difference:

- Let s1 and s2 be two relations: then s1-s2 returns a relation containing all the tuples present in s1 but not in s2.
- To perform a set difference operation on two relations, two relations must satisfy union compatibility property.

(maths formula)

Certation product / cross-product:

Let r and s be two relations. Then, r cross s returns a relation containing all the tuples of r followed by all the tuples of s.
If you want to perform the cross-product on two relations,
The two relations must not satisfy the union compatibility.

Example.: r relation: X s relation:

r x s:

S name	Rating	Age
Dustin	7	45.0
Lubber	8	55.0
Rusty	10	35.0
Sid	Bid	Day
22	101	10/10/96
58	103	11/12/96

S name	Rating	Age	Sid	Bid	Day
Dustin	7	45.0	22	101	10/10/12
Dustin	7	45.0	58	103	11/12/96
Lubber	8	55.0	22	101	10/10/12
Lubber	8	55.0	58	103	11/12/96
Rusty	10	35.0	22	101	10/10/12
Rusty	10	35.0	58	103	11/12/96

(c) division operator

A:

Sno	P no
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

Pno
P2

B2

Pno
P2
P4

B3

Pno
P1
P2
P3

A/B1=

Sno
S1
S2
S3
S4

A/B2=

Sno
S1
S4

A/B3=

Sno
S1

- Consider two relations a and b , in which a has exactly two fields x and y

And

- B has only one field with the same domain as in a.

We can define a division operation as the set of all the “x” values such that for every “y” value in a.

Self-Assessment Questions

1. What is the primary purpose of the selection operator (σ) in relational algebra?

   A). To combine two tables
   B). To filter rows based on a condition
   C). To select specific columns from a relation
   D). To perform a Cartesian product

2. Which operation in relational algebra combines two tables while keeping only distinct rows?

   A). Intersection
   B). Difference
   C). Union
   D). Cartesian Product

3. In relational algebra, which operator extracts specific columns from a relation?

   A). σ (Selection)
   B). π (Projection)
   C). U (Union)
   D). × (Cartesian Product)

4. What does relational algebra's division operator (÷) do?

   A). Combines two relations to return all distinct rows
   B). Filters rows based on a specified condition
   C). Returns entries from one table that match every entry in another table
   D). Extracts specific columns from a relation

5. Which of the following is true about relational algebra's Cartesian product (×)?

   A). It combines rows from two tables based on the condition.
   B). It retrieves rows that are common to both tables.
   C). It combines every row of one table with every row of another.
   D). It extracts specific columns from a relation.

2.2.2 Database Design Guidelines

Joins in Relational Algebra:

Join is one of the most powerful features in relational algebra.
These are mostly used to combine the information from two or more tables.

In relational algebra, the joins are mainly classified into three types:

- Condition join
- Equi join
- Natural join

(i) Condition join:

In a condition join, the operation accepts the join condition ‘c’ and a pair of relations and returns a relation instance.

The condition join operation is specified as below:

(pic of formula)

Here, the condition may be > or< operators.

Generally, in a condition join, we perform the cross product operation between the two relations first, and then we select the rows based on the conditions.

Let s and r be two relations:

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0

Symbol pic

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0

Example:

Sid	Sname	Rating	Age	Sid	Bid	Day
31	Lubber	8	55.0	22	101	10/10/96
58	Rusty	10	35.0	22	101	10/10/96

(ii) equi join:

In an equi join, the operation accepts the join condition ‘c’ and a pair of relations and returns a relation instance.

The condition join operation is specified as below:

(pic of formula)

Here, the condition may be ‘=’ operators.

Generally, in a condition join 1st we perform the cross-product operation b/w the two relations and we will select the rows based on the conditions.

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0

Symbol pic

Sid	Bid	Day
22	101	10/10/96
58	103	11/12/96

(iii) natural join:

A natural join simply performs a cross-product operation b/w two

Automatically joins tables on all columns with the same name, removing duplicate columns.

Example: If Sailors and Reservations have a common Sid column, a natural join combines the tables using this column without an explicit condition.

Database Design Guidelines and Practical SQL Skills

1. **Database Design Guidelines**
    - Good database design ensures that data is organised efficiently and without redundancy. Guidelines include normalising tables to minimise duplication and designing schemas that are easy to interpolate and maintain.
2. **SQL Practical Skills**
    - **Views**: Virtual tables created from query results. Views simplify complex queries and enhance security by restricting access to specific data.
    - **Compound Statements**: Multiple SQL statements grouped as a single unit. Useful in procedures or functions where several actions are performed together.
    - **User-Defined Functions (UDFs)**: Custom functions created to perform specific calculations or data transformations that standard SQL functions don't cover.
    - **Procedures**: Predefined SQL code blocks can be called to execute a sequence of commands, improving reusability.
    - **Cursors**: Database objects that allow row-by-row processing of query results, useful for handling large datasets in procedural tasks.
    - **Triggers**: Automated actions that execute in response to certain database events (e.g., inserting, updating, or deleting records), commonly used for enforcing data integrity.

Self-Assessment Questions

6. In relational algebra, which type of join automatically matches tables on all columns with the same name and removes duplicate columns?

   A). Condition Join
   B). Equi Join
   C). Natural Join
   D). Cartesian Join

7. Which join type in relational algebra uses conditions such as =, >, or < to combine rows from two tables?

   A). Condition Join
   B). Equi Join
   C). Natural Join
   D). Outer Join

8. What does an equi join specifically rely on to link rows between two tables?

   A). Cross-product followed by selection based on non-equal conditions
   B). Matching columns with identical names automatically
   C). A specified equality condition between columns
   D). Conditions involving greater than or less than

9. Which of the following SQL elements allows you to create a virtual table from a query, useful for simplifying complex queries and controlling data access?

   A). View
   B). Trigger
   C). Cursor
   D). Procedure

10. In SQL, which database object is designed to execute automatically in response to specific events such as inserting, updating, or deleting records?

    A). Procedure
    B). Cursor
    C). Trigger
    D). Compound Statement

Summary

- Relational algebra is a procedural query language that structures data retrieval by specifying the needed data and the steps to acquire it.
- Selection filters rows based on conditions, while projection extracts specific columns from a table.
- Set operations, such as union, intersection, and difference, enable the combination and comparison of tables with compatible structures, allowing retrieval of common or unique rows.
- The Cartesian product combines all rows from one table with all from another, which is useful for creating comprehensive table combinations.
- The division operator retrieves entries from one table that match every corresponding entry in another, ideal for queries needing exhaustive matches.
- Joins link data across tables, enhancing query flexibility with types like condition, equi, and natural joins, using different criteria to match and combine records.
- Condition joins use specified criteria to match rows, equi joins require exact matches in designated columns, and natural joins automatically match by column names.
- Proper database design involves organising tables to prevent redundancy, normalising data, and creating intuitive, maintainable schemas.
- SQL views allow for creating virtual tables based on query results, simplifying complex queries and controlling access to sensitive data.
- Advanced SQL elements, such as compound statements, user-defined functions, procedures, cursors, and triggers, enhance database functionality and support automation for managing complex tasks.

Terminal Questions

1. What is the primary purpose of the selection operator in relational algebra, and how is it used?
2. How does the projection operator differ from the selection operator? Provide an example for each.
3. Describe the union operation and its requirements for two relations to be union-compatible.
4. Explain the Cartesian product and provide a scenario where it might be applicable.
5. What does the division operator do in relational algebra, and in what situations is it particularly useful?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	B
2	C
3	B
4	C
5	C
6	C
7	A
8	C
9	A
10	C

Activity

Activity type: Online **Duration: 1 Hour**

Apply at least three different relational algebra operations (selection, projection, and a set operation) to derive meaningful insights from the data.

- Prepare a brief presentation summarising their findings and the operations used.

Glossary

- **Selection Operator (σ):** An operation that filters rows based on a specified condition.
- **Projection Operator (π):** An operation that selects specific columns from a relation.
- **Union (U):** A set operation that combines two relations into one, including all unique rows.
- **Cartesian Product (×):** An operation that combines every row of one relation with every row of another.
- **Division Operator (÷):** An operation that finds entries in one table that correspond to every entry in another.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **Database Design:** https://www.guvi.com/blog/database-design-principles-and-best-practices/
- **Relational Algebra:** https://www.guvi.com/blog/database-design-principles-and-best-practices/

Video Links

Topic	Link
Relational database design	https://www.youtube.com/watch?v=2ImhOvm-vp7w&list=PL4dfXPSoX1m7MFII32edVXhfwn_HOlt4L
Basics of relational algebra	Basics of relational algebra

Image Credits

Fig. 1: Relational Algebra	https://dasarpai.com/dsblog/relational-algebra

Keywords

- Relational Algebra
- Set Operations
- Triggers
- Cursors
- Cartesian Product
- Database Design
- SQL Skills

DATABASE SYSTEMS

MODULE 3

Database Normalisation and File Storage Structures

Module Description

This module covers the essentials of database normalisation and file storage structures. Participants will grasp the different levels of database normalisation, starting with the First, Second, and Third Normal Forms and progressing to the Boyce-Codd Normal Form (BCNF) and higher normal forms. The module explains how normalisation helps organise data effectively, remove redundancy, and ensure data consistency. It also introduces decomposition algorithms, which break down complex tables into simpler ones to achieve optimal normalisation.

In addition to normalisation, the module explores file and storage structures that support efficient data management. Topics include file storage techniques that help organise data on physical media, various index structures for fast data access, and indexing and hashing methods for quicker data retrieval. Students will also learn about query processing and optimisation strategies, essential for improving database performance. This module provides a solid foundation for organising data within a database and designing storage structures to support quick and efficient data access. By the end of the module, participants will interpolate how to implement effective database designs that improve data integrity and access speed.

This module can be divided into **two** units.

Unit 3.1 Database Normalisation
Unit 3.2 File and Storage Structures

MODULE 3

Database Normalisation and File Storage Structures

Unit 1

Database Normalisation

Unit Table of Contents

Unit 3.1 Database Normalisation

Aim

To grasp database normalisation up to higher normal forms, addressing data integrity and minimising redundancy. The unit covers decomposition algorithms for effective normalisation.

Instructional Objectives

This unit intends to:

- Define the concept of database normalisation and its importance
- Identify the characteristics of First, Second, and Third Normal Forms (1NF, 2NF, 3NF)
- Recall the definitions and principles of Boyce-Codd Normal Form (BCNF) and higher normal forms

Learning Outcomes

Upon completion of the unit, you will be able to:

- Explain the process of transforming a database schema to First, Second, and Third Normal Forms
- Illustrate examples of multivalued dependencies and join dependencies in a database
- Compare BCNF with 3NF and other higher normal forms to determine their application in normalisation

3.1.1 Database Normalisation and Normal Forms

Purpose of Normalisation

Database normalisation is a process in database design that organises data to reduce redundancy (repetition) and improve data integrity. It involves dividing and restructuring tables into smaller, more focused tables, each serving a clear purpose. Normalisation has several stages or "normal forms" (NF), each with rules that a table should follow to meet that level.

SSN	NAME	LOT	RATING	HOURLY WAGES	HOURLY WORKED
123-22-3666	ATTISHO	48	8	10	40
231-31-5368	Smily	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	GULDU	35	5	7	32
612-67-4137	MADAYAN	35	8	10	40

Hourly-Emps Relation

Sorting the same information redundantly, i.e., in more than one place, can lead to several problems

Redundant Storage:

Some information is stored repeatedly in the database, which consumes a lot of space on the disk.

Example: In the above table, sequences 8 and 10 are repeated thrice, consuming a lot of disk space.

Update Anomalies:

An inconsistency is created if one copy of repeated data is updated unless all the copies are similarly updated.

Insertion Anomalies:

It may not be possible to store certain information unless other unrelated information is known.

Deletion Anomalies:

It may be impossible to delete certain information without losing some other information. To overcome the above four problems, we use a normalisation concept.
"The decomposition of a table into 2 or more tables is called normalisation"

(or)

"The process of bringing back the table to some standard form or normal form is called normalisation"

Concept of functional dependencies (FD's):

	X	Y	Z
T1	A1	B1	C1
T2	A2	B2	C3
T3	A1	B1	C1
T4	A4	B3	C4

Let, R be a relation schema and x and y be 2 non-empty set of attributes then the relation R exists a FD's x->y if only if satisfies the following condition.

$$t_1(x) = t_2(x) \quad \& \quad t_1(y) = t_2(y)$$

Consider the above reaction R.

Case 1:

$t_1(x) = a$, ; $t_3(x) = a$,

$t_1(y) = b$, ; $t_1(y) = b$,

$$t_1(x) = t_3(x) \quad \& \quad t_1(y) = t_3(y)$$

There exists $a \in D$

$x \rightarrow y$

Case2: case3:

$t_1(x) = a$, ; $t_3(x) = a$,

$t_1(y) = c$, ; $t_1(y) = c$,

$$t_1(x) = t_3(x) \quad \& \quad t_1(y) = t_3(y)$$

There exists $a \in D$

$x \rightarrow y$

t_1 (y) = b, ; t_3 (y) = b,
t_1 (z) = c, ; t_3 (z) = c,

t_1 (y) = t_3 (y)
&
t_1 (z) = t_3 (z)

There exists $a \in D$

y→z

Closer set of functional dependencies:

The set of all the functional dependencies implied by the given set 'F' is called the "closer of a set of functional dependencies."

And it's denoted as 'F⁺'.

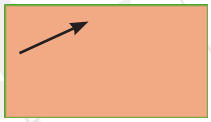
To calculate the closer of a set of functional dependencies, we use five rules called 'Armstrong axioms' (or) ' Rules of Inference.'

Rules are:

1. **Reflexivity Rule:**

   If alpha (α) and beta (β) are the set of attributes, then beta(β)is subset/equal ($\subseteq$) to alpha(α), then we can say there exist functional dependencies (FD's):

   α β

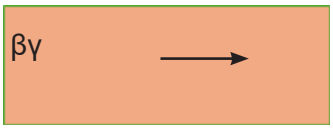


2. **Augumentation Rule:**

   If alpha α determines beta β holds and gamma (γ) is another set of attributes, then we can say there exists a FD'S:

   Alpha gamma determines beta gamma (γ).

   αγ βγ

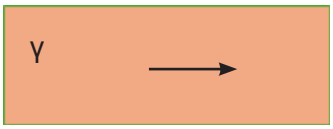


3. **Transitivity Rule:**

   If α -> β holds and β -> γ holds, then there exist FDs:

   Alpha α determines gamma γ:

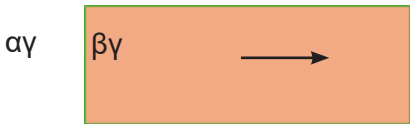
   α γ



4. **Union Rule:**

If α -> β holds and α -> γ holds, then there exist a FDs:

Alpha determines beta gamma.

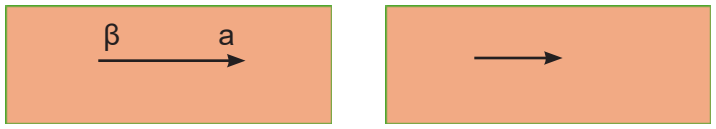


5. **Decomposition Rule:**

If α -> βγ holds

Then there exist a FD's:

Alpha determines beta, and alpha determines gamma



Problem:

Let's consider a relation R(A, B,C,G,H,I) and the set of FD's are F= { A->B
A->C
CG->H
CG->I
B->H}

Then calculate the closer set of FD'S(F+)

Solution: $\in^{+} = \{ A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H$

$A \rightarrow BC$ [According to union rule if $A \rightarrow B \ \& \ A \rightarrow C \Rightarrow A \rightarrow BC$]

$CG \rightarrow HI$ [According to union rule if $CG \rightarrow M \ \& \ CG \rightarrow I \Rightarrow CG \rightarrow HI$]

$A \rightarrow H$ [According to Transitivity rule if $A \rightarrow B \ \& \ B \rightarrow H \Rightarrow A \rightarrow H$]

$\}$

Attribute Closer of Functional Dependencies:

Steps for finding attributes closer to functional dependencies.

- **Step 1:** First, find the attribute closest to the attributes do not present on the right-hand side.
- **Step 2:** If the above step fails, calculate the attribute closest to the set of attributes that are not present on the left-hand side.

  If this step fails, then go to step 3.
- **Step 3:** Calculate the attribute closer to the combination of any attributes.

Problems:

1. Let's consider a relation R(A, B,C,G,H,I) and the set of FD's are F= { A->B
   A->C
   CG->H
   CG->I
   B->H}

Calculate the attribute closer to the relation R:

Solution : $(AG)^+ = AG$

$= ABG\ [A \rightarrow B]$

$= ABCG\ [A \rightarrow C]$

$= ABCGH\ [CG \rightarrow H]$

$= ABCGHI\ [CG \rightarrow I]$

A and G are called candidate keys / prime attributes

2. Let's consider a relation R (A, B,C,G,H,I) and the set of FD's are F= { A->B
   BC->E
   ED->A}

list all the keys for the given relation R.

Solution : $(C \in D)^+ = C \in D$

$= CDB\ [BC \rightarrow \in] = AC \in D\ (\in D \rightarrow A)$

$= CDB \in [\in D \rightarrow A] = ABC \in D\ (A \rightarrow B)$

$= CDB \in A\ [A \rightarrow B]$

CE and D are called candidate keys / prime attributes

$(ACD)^+ = ACD$

$= ABCD\ [A \rightarrow B]$

$= ABCD \in [BC \rightarrow B \in]$

A, C, D are called candidate keys

$(BCD)^+ = BCD$

$= BC \in D\ [BC \rightarrow \in]$

$= ABC \in D\ [\in D \rightarrow A]$

B, C, D are called cadidate keys

Normal forms based on functional dependencies:

Keeping the table in some standard forms is called normalisation.

If a table wants to keep in a 3NF, it must satisfy all the criteria of 1NF and 2NF.

Based on FDs, normal forms are categorised into four types.

- 1NF or 1st normal form
- 2NF or 2nd normal form
- 3NF or 3rd normal form
- BCNF or Boyce-cod.

1NF or 1st normal form:

1NF requires that:

- All data in a table should be atomic, meaning each column should contain only one value per row.
- There should be no repeating groups or arrays in any column.

For example, if you have a column storing multiple phone numbers, you would split each number into separate rows to meet 1NF.

Roll no	Name	Courses
101	Asif	DBMS, CN, CD, SE
102	Amit	CO, OS
103	Arpit	CD, DS, CN, DBMS

If a relation wants to be 1NF, it must satisfy the following specific criteria:

1. All the values in the column must be atomic. That is, composite values are not allowed.
2. No two rows should be identical.
3. The values in the column must be of the same kind.
4. Each column must have a unique name.

Roll no	name	Courses
101	Asif	DBMS
101	Asif	CN
101	Asif	CD
101	Asif	SE
102	Amit	CO
102	Amit	OS
103	Arpit	CD
103	Arpit	DS
103	Arpit	CN
103	Arpit	DBMS

(ii)2NF or second NF

2NF builds on 1NF and requires that:

- The table must be in 1NF.
- Each non-key column (a column that is not part of the primary key) must depend entirely on the primary key, not just part of it.

This means that for tables with a composite primary key (a primary key made of more than one column), all non-key columns must depend on both parts of the key, not just one.

SID	PID	SNAME	SCITY	STATUS	DNAME	QTY
S1	P1	avia	Mumbai	10	AC-engine	5
S1	P2	Avia	Mumbai	10	Generator	5
S2	P1	Aello	Delhi	20	AC-engine	2
S2	P3	Aello	Delhi	20	Altimeter	5
S3	P2	Air supp	Mumbai	10	Generator	10
S3	P3	Air supp	Mumbai	10	Altimeter	20

THE FDs for the above table is given as below

F= { sid -> sname,scity

Pid-> Pname

Scity->status

Sid,pid -> qty }

By applying the ARMSTRONG axioms to above FD's as specified below

F= { sid -> sname , scity , pname , status

Pid-> Pname

Sid , pid -> qty }

Identification of prime attributes and non-prime attributes:

$(sia, pid)^+$ = sid, pid

= sid, pid, frame, scity, status

[sid→ frame, scity, status]

= sid, pid, frame, frame, scity, status

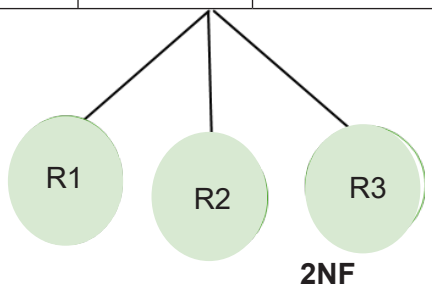
[pid → frame]

= sid, pid, qty, frame, frame, scity, status

[sid, pid → qty]

From the above 3 FDs, only the 3rd Satisfies the criteria of 2NF. Hence, we are decomposing the relations as R1, R2, and R3.

SID	PID	SNAME	SCITY	STATUS	FRAME	QTY



2NF

Fig. 1: Second NF

(iii) 3NF or third NF:

A relation R is said to be in 3NF if it satisfies the following specifications:

- A relation must be in 1NF and 2NF
- Transitive dependencies are not allowed.

SID	PID	SNAME	SCITY	STATUS	DNAME	QTY
S1	P1	Avia	Mumbai	10	AC-engine	5
S1	P2	Avia	Mumbai	10	Generator	5
S2	P1	Aelo	Delhi	20	Ac-engine	2
S2	P3	Aelo	Delhi	20	Altimeter	5
S3	P2	Air supp	Mumbai	10	Generator	10
S3	P3	Air supp	Mumbai	10	altimeter	20

The FDs for the above table are given as below:

F= { sid -> sname,scity(partial FD)

Pid-> Pname(Partial FD)

Scity->status(non FD)

Sid,pid -> qty }(Fully FD)

Identification of prime attributes and non-prime attributes:

$(sid,pid)^+$ = sid,pid

= sid,sname,scity,pid[sid->sname,scit]

= sid,sname,scity,status,pid[scity->status]

= sid,sname,scity,status,pid,pname

[pid->pname] = sid,sname,scity,status,pid,pname[sid,pid->qty]

->Prime attributes={sid,pid}

->Non-prime attributes= {sname,scity,status,pname,qty}

Hence, we are decomposing the R in to R1, R2,R3,R4 as specified below:

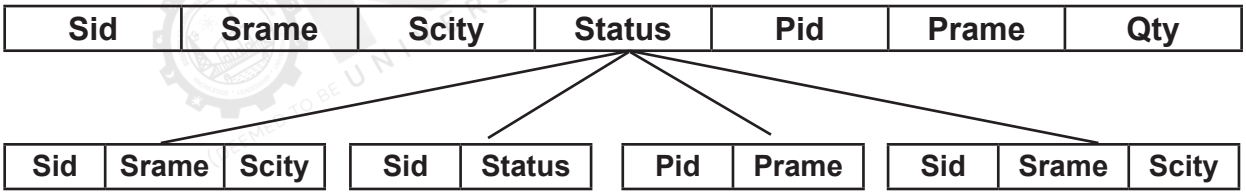


Fig. 2: Third NF

(iv) BCNF or Boyce-codd:

BCNF is an advanced version of 3NF with stricter rules:

- The table must be in 3NF.
- Every determinant (a column that uniquely identifies another column) must be a candidate key (a unique identifier for the table).

If there's a column that can determine other columns but isn't a candidate key, the table needs further normalisation to remove that dependency.

– R must be in 3NF
– If x->y is FD

Then x must act as a super key.

Example: R (A, B, C , D)

F={A->BCD

BC-> AD

D->B}

Identification of super keys:

(i) A-> BCD

$(A)^+$ =A

= ABCD[A->BCD]

-A is acting as a super key

(ii) BC - > AD

$(BC)^+$ =BC

= BCAD

(BC->AD)

-BC is acting as a super key

(iii) D->B

(D)+ =D

= DB(D->B)

-D is not acting as a key

>super keys = A, B, C

>non-super keys = D

A relation R is said to be in BCNF if it satisfies the following specific criteria:

a) R must be in 3NF
b) If x->y is FD then x must act as a super key:

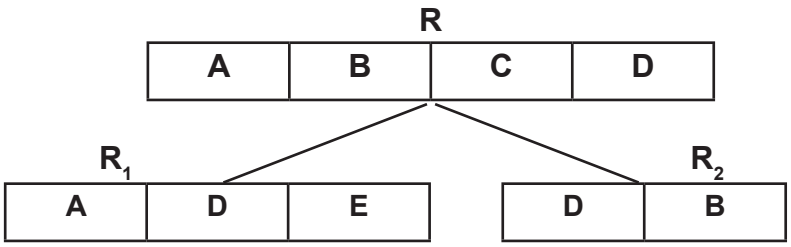


Fig. 3: BCNF

In the above decomposition, the super key B depends entirely on non-super key D. So, the attribute B should be in the R2 relation, and over, D is acting as a primary key in the R2 relation. Hence, D must act as a foreign key in the R1 relation.

Concept of surrogate key:

A surrogate key is an artificial primary key generated automatically by the system.
It is the numerical value which is automatically incremented for each row.

Minimal cover for the set of functional dependencies:

Minimal cover for the set of functional dependencies (FD's):

Problems:

Find the minimal cover for the following set of functional dependencies.

F={X->W, WZ->XY , Y->WXZ }

Sol:

STEP-1:

F={X->W

WZ->X

WZ->Y

Y->W

Y->X

Y->Z}

Step 2: Eliminate extraneous (additional) FDs:

(i)_ X->W

$(X)^+ = X$

= XW[X->W]

IGNORE X->W

(X)+ = X

HENCE, X->W is a wanted functional dependency.

(ii) wz->x

(wz)+ = wz

= wzx(wz->x)

= wzx(wz->y)

ignore wz-> x

(wz)+ = wz

= wzy(wz->x)

= wzxy(y->x)

hence, wz->x is an extraneous FD

(iii) wz->y

(wz)+ = wz

= wzy[WZ->Y]

=WZXY [Y->X]

IGNORE (WZ)->Y

(WZ)+ = WZ

HENCE, WZ->X IS AN wanted FD.

(iv) y -> w

(y)+ = y

= YW [y->w]

=YXW [Y->X]

=YXWZ [Y-> Z]

ignore Y-> W

(y)+ = y

= YX (Y->W)

=YXZ (Y->Z)

=YXZW [X-> W]

=Y-> W IS an extraneous FD.

(v) Y-> X

(Y)+ = Y

= YX (Y->X)

=YXZ(Y->Z)

=YXZW(X-> W)

ignore y->w

(Y)+ = Y

= YZ[Y->Z]

hence, Y->X is an wanted FD

(vi) Y-> Z

(Y)+ = Y

= YZ (Y->Z)

=YXZ(Y->X)

=YXZW(X-> W)

ignore y->Z

(Y)+ = Y

= YX[Y->X]

=YXW[X->W]

hence, Y->Z is an wanted FD.

FD ={X-> W

WZ->Y

Y->X

Y->Z }

STEP 3:

WZ-> Y

(WZ)+ = WZ

=WZY(WZ-> Y)

=XYZW(Y->X)

hence, (WZ)+ = {XYZW}

(W)+ = W

(Z)+ = Z

FINAL FD'S ARE F= {X-> W

WZ->Y

Y->X

Y->Z}

Properties of Decomposition:

The splitting of a relation into two or more relations is called decomposition.
While decomposing the relation 'R' into 2 or more relations, i.e. 'R1' and 'R2' and that decomposition should satisfy the below two properties.

1. Loss less joint decomposition
2. Dependency-preserving decomposition

Loss-Less Joint Decomposition:

Decomposition of a relation 'R' into 'R1' and 'R2' is said to be lossless joint if R1 ⋈ R2 = R (natural join - ⋈)

Should contain exact rows of R with no tuples added and no tuples deleted.
Let's consider the relation R:

A	B	C
A1	B1	C1
A2	B2	C1
A1	B1	C2
A3	B2	C3
A1	B1	C3
A2	B2	C4

Let's decompose R relation into two relations: R1(A,B) and R2(A,C)

Table R_1

A	B
A1	B1
A2	B2
A3	B2

Table R_2

A	C
A1	C1
A2	C1
A1	C2
A3	C3
A1	C3
A2	C4

Since R1⋈ R2=R, we can conclude that our decomposition is a loss-less join decomposition.

A	B	C
A1	B1	C1
A1	B1	C2
A1	B1	C3
A2	B2	C1
A2	B2	C4
A3	B2	C3

Dependency – Preserving Decomposition:

Getting loss less join is necessary, but of course

We must also keep our original FDs safe. Our decomposition is satisfying the property 'dependency preserving'.

Now let's consider a relation R (A, B, C) and the FD'S are F= {A->B
B->C}
Now let's decompose the relation R into two relations: R1(A, B) AND R2(B, C).

The possible FDs for the relation R1 are specified as below:
F1 = {A->A
A->B
B->A
B-> B}
Similarly, the possible FDs for the relation R2 are specified as below:
F2 = {B->B,
B->C,
C->B ,
C->C}

- Now perform the union operation between F1 and F2

  F1 U F2 = {A->A
  A->B
  B->A
  B->B
  B->C
  C->B
  C->C }

Since, F1 U F2 contains our original FD's A-> B
And B->C, so we can say our decomposition is dependent preservation.

Concept of multi-value dependencies:

X	Y	Z	
A	B1	C1	Tuple t1
A	B2	C2	Tuple t2
A	B1	C2	Tuple t3
A	B2	C1	Tuple t4

A relation R exists in a multi-value dependency

x->->y
If and only if t1(x)= t2(x)
t(xy)=t3(xy)
t2(z)=t3(z)

a=a hence, t1(x) = t2(x)
ab1=ab1 hence, t1(xy) = t3(xy)
c2=c2 hence, t2(z) = t3(z)
therefore, x →→y

4TH Normal Form:

Beyond BCNF, higher normal forms handle more complex cases:

4th Normal Form (4NF)

4NF addresses multivalued dependencies:

- A table is in 4NF if it's in BCNF and has no multivalued dependencies, where one attribute is linked to multiple independent values of another attribute. If this happens, it indicates that two separate tables are needed.

COURSE	TEACHER	BOOK
PHYSICS 101	GREEN	MECHANICAL
PHYSICS 101	GREEN	OPTICS
PHYSICS101	BROWN	MECHANICS
PHYSICS 101	BROWN	OPTICS
MATH 301	GREEN	MECHANICS
MATH 301	GREEN	VECTORS
MATH 301	GREEN	GEOMETRY

FIG: BCNF relation with redundancy that MVD'S reveals.

A relation R is said to be in 4NF if it satisfies the following specific criteria:

(i) R must be in BCNF

(ii) No multi-value dependencies are allowed

Consider the above relation with 3 attributes: course, teacher, and book. We have observed that multiple teachers, green and brown, teach the course Physics 101. So, we can conclude that the relation R exists as a multi-value dependency course-->teacher.

Similarly, we have observed that the course Physics 101 has multiple books, i.e., mechanics and optics.

So, we can conclude that relation R exists in a multi-value dependency course→ book.

Hence, we are decomposing our relation R into relations as specified below:

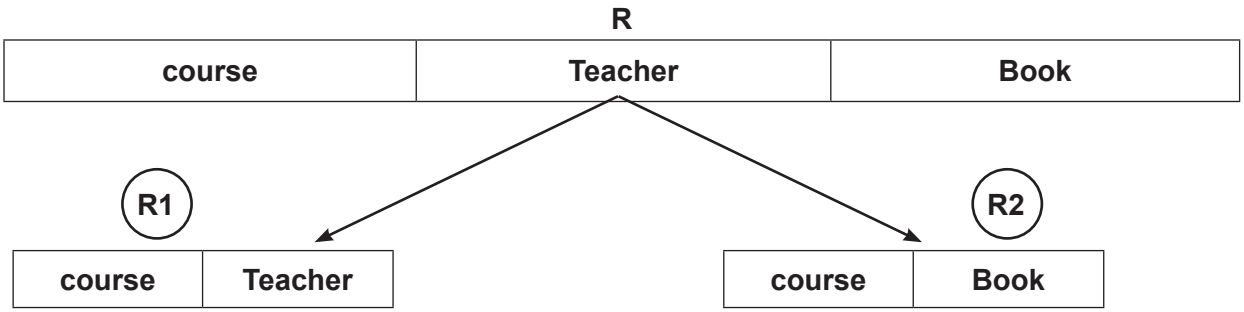


Fig. 4: Fourth NF

Join Dependencies:

A join dependency is a further generalisation of MVD. A join dependency is said to be held over R if R1, R2 …. RN satisfies lossless join decomposition property (R1 □ R2□ …...Rn).

5NF (or) Fifth Normal Form:

A relation R is said to be in 5NF if it satisfies the following specific criteria:

I. R must be in 4NF

II. Join dependencies are not allowed unless

R1 □ R2□ …...Rn= R

Decomposition Algorithms

Decomposition is the process of breaking down tables to meet specific normal forms. Effective decomposition ensures:

- Data consistency (data is stored only once where necessary).
- Data integrity (changes in one place don't create inconsistencies).

In decomposition, it's crucial to keep all original data intact and ensure that the tables can be recombined or joined to retrieve the original dataset without losing information.

Self-Assessment Questions

1. What is the primary goal of database normalisation?

   A). Increasing redundancy
   B). Reducing redundancy
   C). Expanding data storage
   D). Decreasing data security

2. Which normal form requires that each attribute contains atomic values only?

   A). 1NF
   B). 2NF
   C). 3NF
   D). BCNF

3. What kind of anomalies does normalisation help reduce?

   A). Sorting
   B). Scripting
   C). Update, insertion, and deletion
   D). Storage

4. What does the Augmentation Rule in Armstrong's Axioms imply?

   A). Alpha determines beta
   B). Adding attributes does not affect dependencies
   C). Attributes can be replaced with others
   D). Alpha and gamma are independent

5. Which of the following normal forms addresses multivalued dependencies?

   A). 1NF
   B). 2NF
   C). 4NF
   D). 3NF

Self-Assessment Questions

6. What is required for a relation to be in BCNF?

   A). It must be in 2NF
   B). Every determinant must be a candidate key
   C). It must contain multivalued dependencies
   D). It must have surrogate keys

7. Which rule allows us to decompose a table to retain all original dependencies?

   A). Lossless join
   B). Dependency preservation
   C). Surrogate key rule
   D). Functional dependency

8. What does ‘decomposition’ mean in database normalisation?

   A). Adding more tables
   B). Combining tables into one
   C). Splitting a relation into smaller relations
   D). Deleting duplicate data

9. What type of dependency exists if an attribute indirectly depends on a non-primary key?

   A). Transitive dependency
   B). Direct dependency
   C). Functional dependency
   D). Multivalued dependency

10. Which normal form is stricter than 3NF and ensures that each functional dependency is unique?

    A). 1NF
    B). 2NF
    C). BCNF
    D). 4NF

Summary

- Database normalisation organises data to reduce redundancy and improve data integrity by dividing tables into smaller, specific ones.
- Redundant storage occurs when the same data is stored multiple times, which can consume excess space.
- Update anomalies can create inconsistencies if repeated data copies are not uniformly updated.
- Insertion anomalies arise when adding data without needing unrelated information is impossible.
- Deletion anomalies lead to unintended data loss when deleting specific information.
- Functional dependencies (FDs) show relationships between attributes and are foundational to normalisation.
- The closure of a set of FDs is the complete set of all implied dependencies found using Armstrong's axioms.
- The normal forms (1NF, 2NF, 3NF, BCNF) define stages in normalisation, with stricter requirements for each stage.
- Higher normal forms, like 4NF, address complex issues like multivalued dependencies to refine data organisation further.
- Proper decomposition of relations maintains data integrity, ensuring lossless joins and dependency preservation.

Terminal Questions

1. What is database normalisation, and why is it important?
2. What is a functional dependency?
3. What are Armstrong's Axioms, and why are they used in normalisation?
4. Explain the difference between 1NF, 2NF, and 3NF.
5. What is BCNF, and how does it extend 3NF?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	B
2	A
3	C
4	B
5	C
6	B
7	B
8	C
9	A
10	C

Activity

Activity type: Online **Duration: 1 Hour**

Create a database in BCNF for a fictional business scenario (e.g., a library or inventory system), specifying tables, primary keys, and foreign keys.

Glossary

- **Normalisation:** Organising data to reduce redundancy and improve integrity.
- **Anomalies:** Problems in a database due to redundancy, including update, insertion, and deletion anomalies.
- **Armstrong's Axioms:** A set of rules for inferring functional dependencies.
- **Lossless Join:** A decomposition property ensuring no data loss when tables are joined.
- **Dependency Preservation:** Ensuring all original functional dependencies are maintained post-decomposition.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **Database Normalisation:** https://www.geeksforgeeks.org/introduction-of-database-normalization/
- **Normal forms 1NF, 2NF, 3NF:** https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/

Video Links

Topic	Link
Database Normalisation	https://www.youtube.com/watch?v=xoTyrdT9SZI&list=PLLGlmW7jT-nTr1ory9o2MgsOmmx2w8FB3
Normal forms 1NF, 2NF, 3NF	https://www.youtube.com/watch?v=J-drts33N8g

Image Credits

Fig. 1 Second NF	Self-generated
Fig. 2 Third NF	Self-generated
Fig. 3 BCNF	Self-generated
Fig. 4 Fourth NF	Self-generated

Keywords

- First Normal Form (1NF)
- Boyce-Codd Normal Form (BCNF)
- Dependency Preservation
- Surrogate Key
- Normalisation

MODULE 3

Database Normalisation and File Storage Structures

Unit 2

File and Storage Structures

Unit Table of Contents

Unit 3.2 File and Storage Structures

Aim

To introduce participants to the fundamentals of file and storage structures, including techniques and strategies for efficient data storage, retrieval, and query optimisation.

Instructional Objectives

This unit intends to:

- Define key terms such as file structures, indexing, hashing, and query optimisation
- Identify different file storage techniques and index structures
- Describe the basic concepts of hashing and its application in databases

Learning Outcomes

Upon completion of the unit, you will be able to:

- Explain how file and storage structures are used in managing database systems
- Compare different index structures and their suitability for specific database operations
- Illustrate the process of query optimisation with simple examples

3.2.1 Fundamentals of Indexing in DBMS

Indexing is a data structure technique to optimise the performance of database queries by minimising disk access.

An index is a data structure that allows quick and easy access to data in a database table.

Index structure:

Some of the columns of the database can be used to create Indexes.

Searched Key	Data Referece

Structure of Index

- The search key in the first column of the database is a replica of the table's primary or candidate key. The values of the primary key are stored in sorted order, making data retrieval easier.
- The second column in the database does reference the data. It consists of a list of pointers, which contains the address of a particular disk block in which the value for that specific key can be found.

Indexing Methods

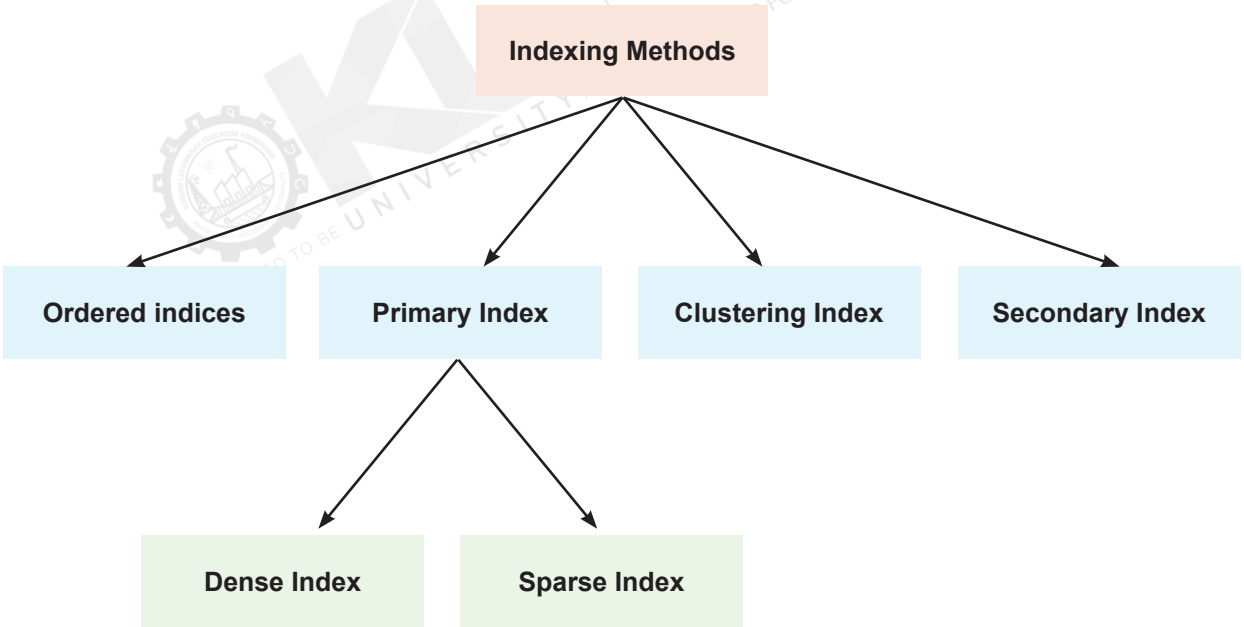


Fig. 1: Indexing Methods

Ordered indices

Indices are commonly ordered for efficient searching. These indexes that you sort is called order indices.

Consider an example where there might be thousands of records in the employee table, each 10 bytes. Assuming their IDs are starting from 1, 2, 3….and now we need to search student ID-543

For example, if this database does not have an index, we need to search from the beginning of the disk block until 543. The DBMS will read the record after having read 543*10=5430 bytes.

For an index, we will search through indexes, and the DBMS will then read the record after reading 542*2=1084 bytes, as opposed to huge sizes in the previous case.

Primary Index

- When an index is created based on the table's primary key, it is called primary indexing. There is a 1:1 relation between these records, and their primary keys are unique to each record.
- Since primary keys are stored in order, the time complexity of the searching operation is very well optimised.
- We also have two types of primary index: the dense index and the Sparse index.

Dense index

- In a dense index, an entry exists for each search key value in the data file, which speeds up searching.
- It means the record count is equal between the index and main tables.
- Additional storage is required for the index record itself. The index has a search key and a pointer to the actual record on disk.

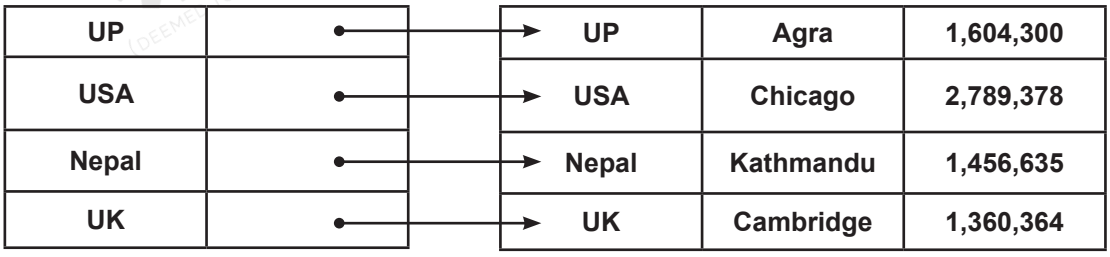


Fig. 2: Dense index

Sparse index

This happens because an index record is present for a few items in the data file. Each item points to a block.

Here, the index does not point to each record in the main table but to records in a gap from that of the main table.

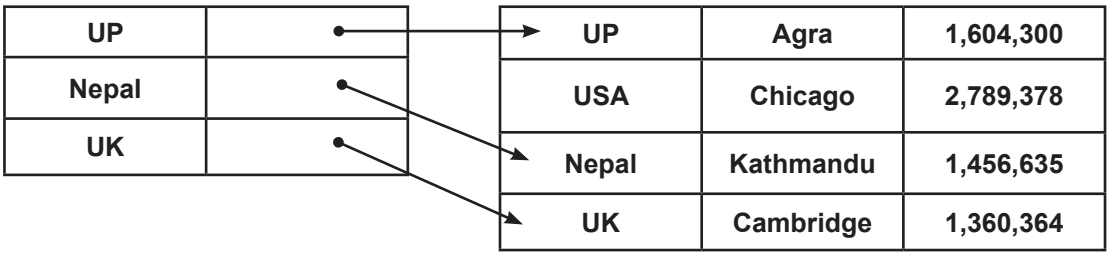


Fig. 3: Sparse index

Clustering Index

- A clustered index could be considered an ordering of the data file. Sometimes, the index is on non-primary key columns that do not uniquely correspond to each record.
- In this case, to find the record quickly, we combine two or more columns to obtain a unique value and create an index. This is referred to as a clustering index.
- This group consists of records with similar characteristics, and an index is created for this group.

For example, let's say we have some departments in a company, each with multiple employees. Imagine we use a clustering index whereby all employees belonging to the same Dept_ID are viewed as being stored together in one single cluster. Each employee is not an independent point; index pointers point to the cluster itself. In this case, Dept_Id is a non-unique key.

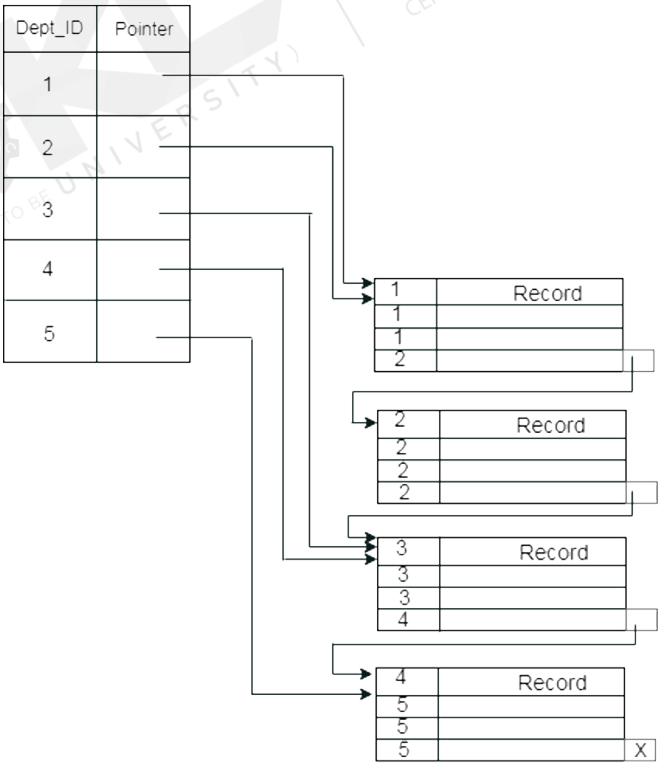


Fig. 4: Clustering Index 1

One disk block contains records belonging to different clusters, making the previous schema confusing. A better technique is to use separate disk blocks for separate clusters.

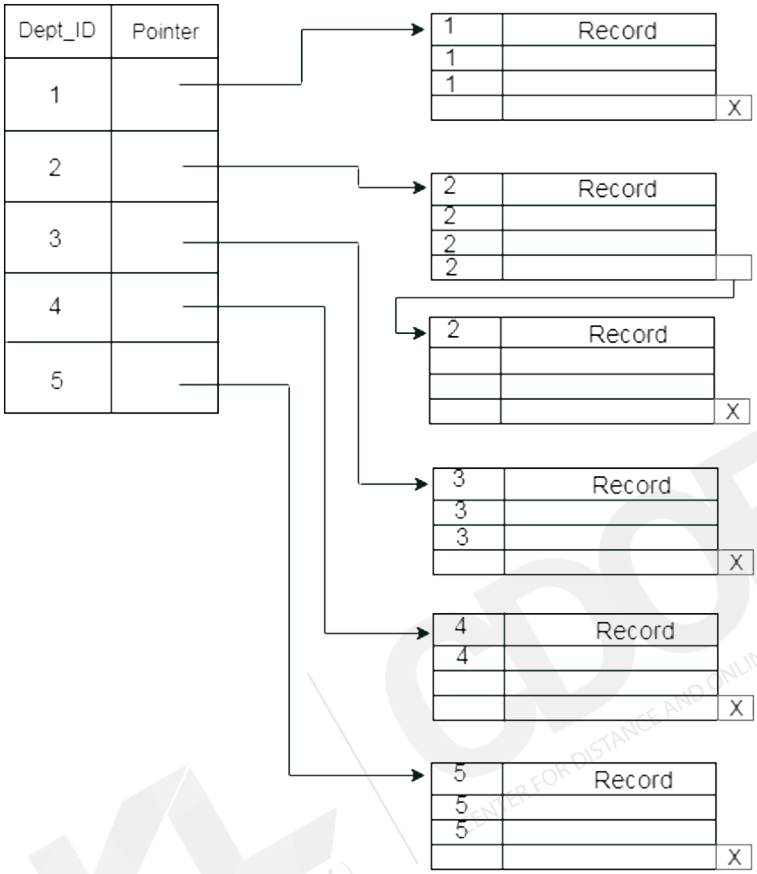


Fig. 5: Clustering Index 2

Secondary Index

Sparse Indexing Mapping Sparsely with Fewer Keys As the size of the table increases so does the size of the mapping. Usually, these mappings are stored in the main memory, so address fetch should be quick. After that, the secondary memory looks for real data by mapping —> Address. Then it takes longer to fetch the address itself if the size of that mapping grows. So here, the sparse index is not gonna work. Secondary indexing is proposed to address this issue.

Secondary indexing uses second-level indexing to minimise mapping. Here, an initial large range for the columns is chosen to make the level one mapping size small. Then, the ranges divide into smaller ranges. The mapping of this first level is kept in primary memory so that address fetch can be done quickly. The second-level mapping and real data are stored in secondary memory (hard disk).

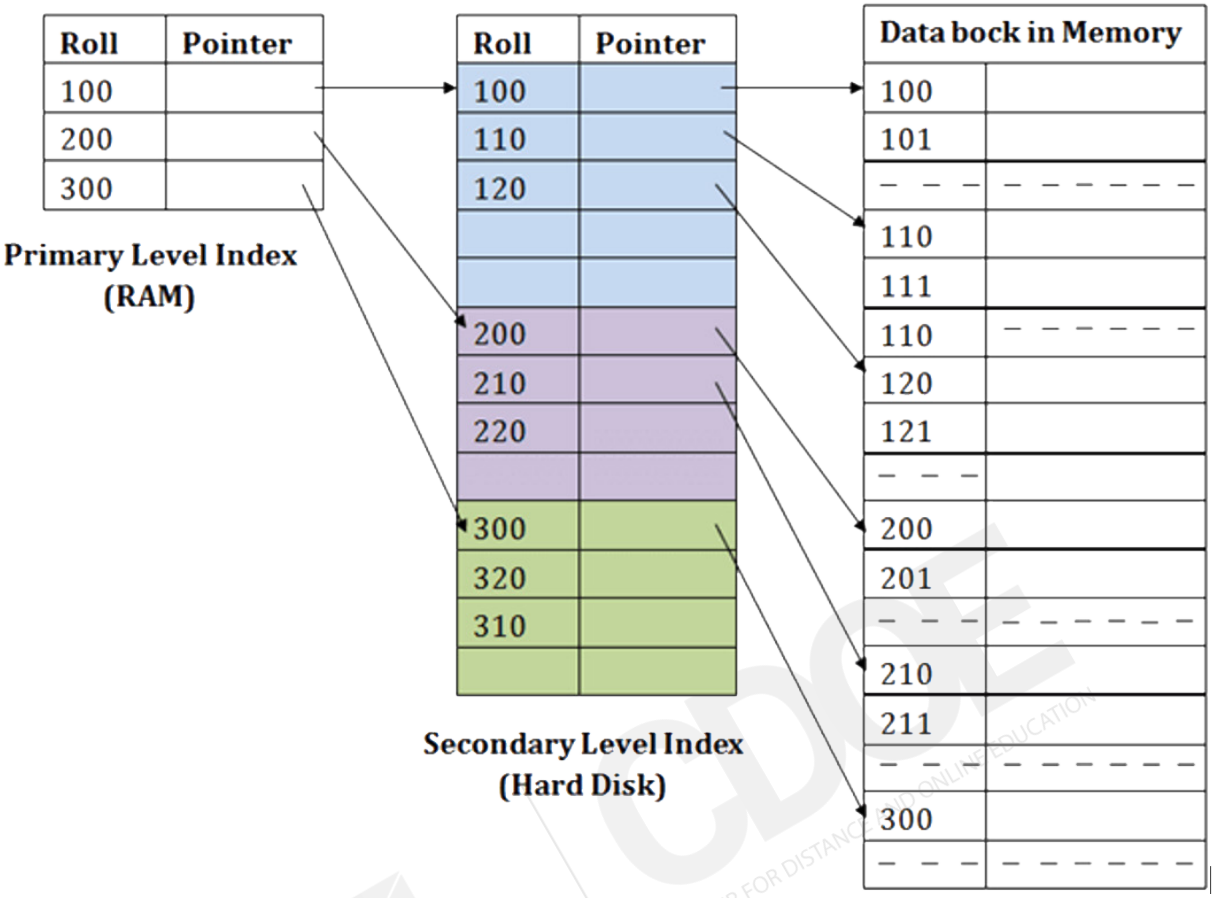


Fig. 6: Secondary Index

For example:

So, if you were searching for the record of roll 111, as shown in the above diagram, it would look for the highest entry, which is = 111 in a first-level index. At this level, it would certainly score 100.

So, in the second index level, similarly, it gets max (111) = 111 and receives 110. Now, it uses address 110, navigates to the data block, and checks each record until it reaches 111.

This is how you would search. Insert, update, or delete operations are also similar.

Self-Assessment Questions

1. Which of the following is true about a dense index?

   A). It has entries only for some of the records in the table.
   B). Additional storage is required for the index record.
   C). It is always created on non-primary key columns.
   D). It does not use pointers to the actual records.

2. What is the key difference between a dense and sparse index?

   A). The dense index is unordered, while the sparse index is ordered.
   B). A dense index contains pointers for all records, whereas a sparse index has pointers only for some records.
   C). Sparse index uses more storage than dense index.
   D). Sparse index can only be created on primary keys.

3. In clustering indexes, what happens when the index is created on non-primary key columns?

   A). The index is created for each record individually.
   B). Records with similar characteristics are grouped, and the index points to the group.
   C). The index is created only for the first column in the table.
   D). The table cannot have a primary key.

4. Which of the following statements about primary indexing is correct?

   A). Primary indexing is only applicable for non-unique keys.
   B). It creates an index based on a table's primary key.
   C). It does not require storage for additional pointers.
   D). Primary indexing does not guarantee ordered storage of keys.

5. Secondary indexing is primarily used to:

   A). Create an index on non-primary key columns without using additional storage.
   B). Reduce the mapping size in primary memory by using two-level indexing.
   C). Group similar records into a single cluster.
   D). Replace sparse indexing entirely in all database systems.

3.2.2 File and Storage Structures in DBMS

File Organisation

- A File is a set of records. We can access the records using the primary key. The type of file organisation used for a given group of records can determine how often and how access will be made.
- File organisation is a logical relationship between records. It describes how file records are mapped to blocks on disk.
- File organisation means how the records are stored in blocks and those blocks on storage.
- The first method to map the database onto the file is multiple files; only one fixed-length record can be stored in each file. Another way is to organise our files in a way that we will be able to put multiple lengths for records
- Fixed-length records: Files with fixed-length records are easier to implement than files with variable-length records.

Purpose of organising files

- It is the most suitable base of records, i.e. records can be chosen at top speed.
- Insert, delete or update transactions on records must be faster and easier.
- These duplicate records, once created, cannot be induced because of insert, update, or deletion.
- Records need to be stored at minimum cost.

Types of file organisation:

There are multiple ways to organise a file. These specific methods have advantages and disadvantages in terms of access or selection. He can consider the best file organisation mechanism according to his requirements and organise it within a file.

Types of All file organisation are:

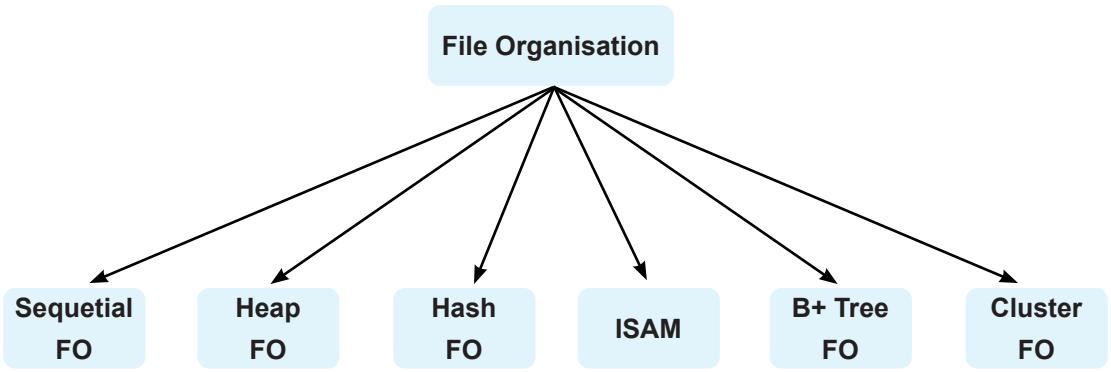


Fig. 7: File Organisation

- Sequential file organisation
- Heap file organisation
- Hash file organisation
- B+ file organisation
- Indexed Sequential Access Method (ISAM)
- Cluster file organisation

Sequential File Organisation

Organising files + This method is the easiest method of file storage. This way of storing files in a sequential format. There are two ways in which this method can be implemented:

Pile File Method:

It is a quite simple method. In this approach, we keep the records in sequence, one after another. You get the row inserted in a table as they insert the rows.

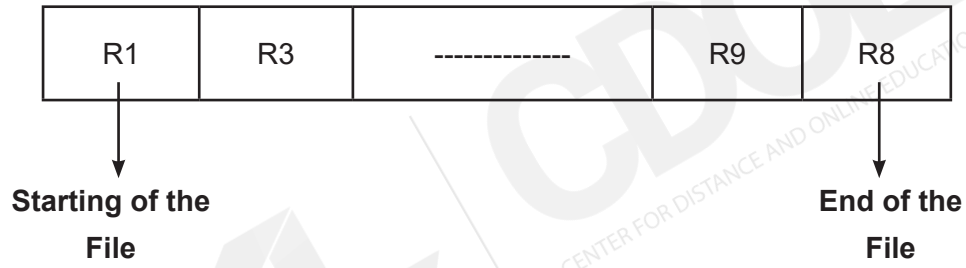


Fig. 8: Pile File Method

For example, while Updating or Deleting any record, it is searched in the Memory Block. When found, it marks the old record for deletion and inserts the new one.

Insertion of the new record:

Consider four records R1, R3 …… R9, and () in some order. So, records are rows in the table. If we want to insert a new record R2 in sequence, then it should be added at the end of the file. As far as records are concerned, they are just single-row bills on any Table.

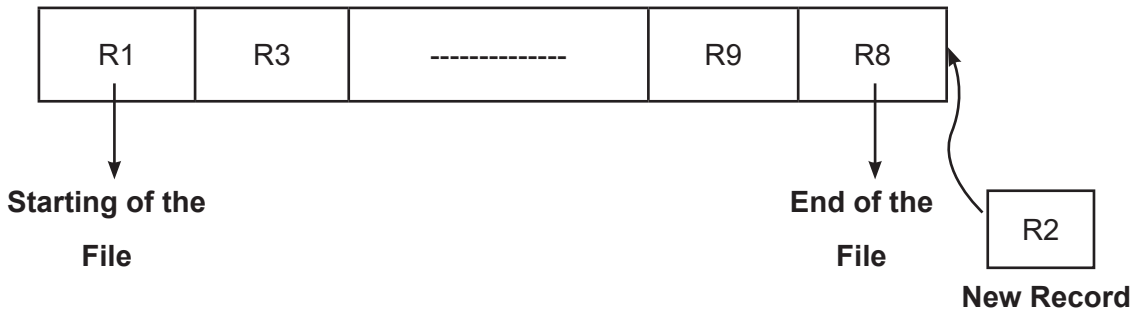


Fig. 9: Insertion of the new record

Sorted File Method:

In this method, the new record is inserted at the end of the file, and it sorts that sequence in ascending or descending order. Records can be sorted based on any primary key or any other key.

When any record is updated, it does three things: update the record, sort all records, and insert the updated record into its corresponding position.

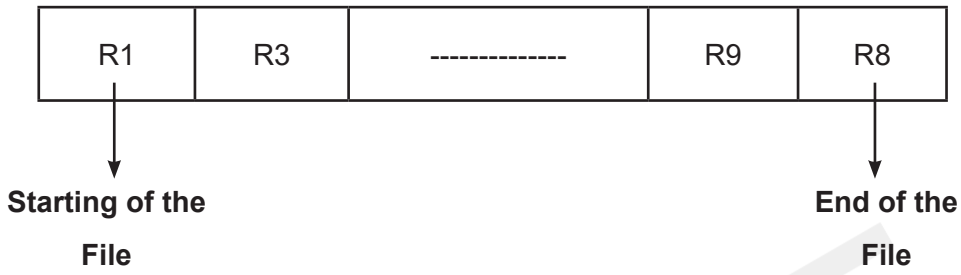


Fig. 10: Sorted File Method

Insertion of the new record:

Imagine a sorted run of four records, R1, R3, and so on, up to R6 and R7. A new record, R2, needs to be inserted in the sequence. It will be inserted at the end of the file, after which it will sort the sequence again.

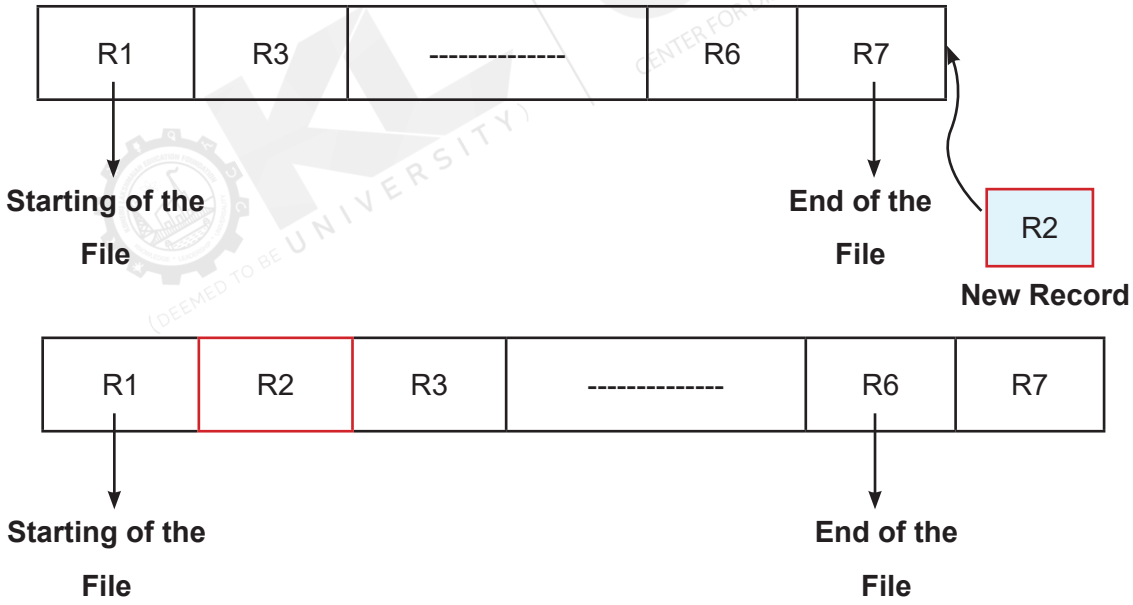


Fig. 11: Insertion of the new record

Heap file organisation

- This is the most plain and simple form of organisation. It works with data blocks. For example, in heap file organisation, the records are inserted at the end of the file. It does not Sort and order the records during record insertion.
- The new record will be saved in another block if the data block is filled. This new data block is not necessarily the immediate next data block, but it can be any free, unused data block available in memory to fill up with records. What is a heap file? − It is also called an unordered file.
- Each record has a unique ID, and each page in a file is the same size. The new records will be stored and managed by the DBMS.

Hash File Organisation

- Hash File Organisation computes a hash function on certain record fields. The hash function output determines the place to put the records, which tells us which disk block to use.

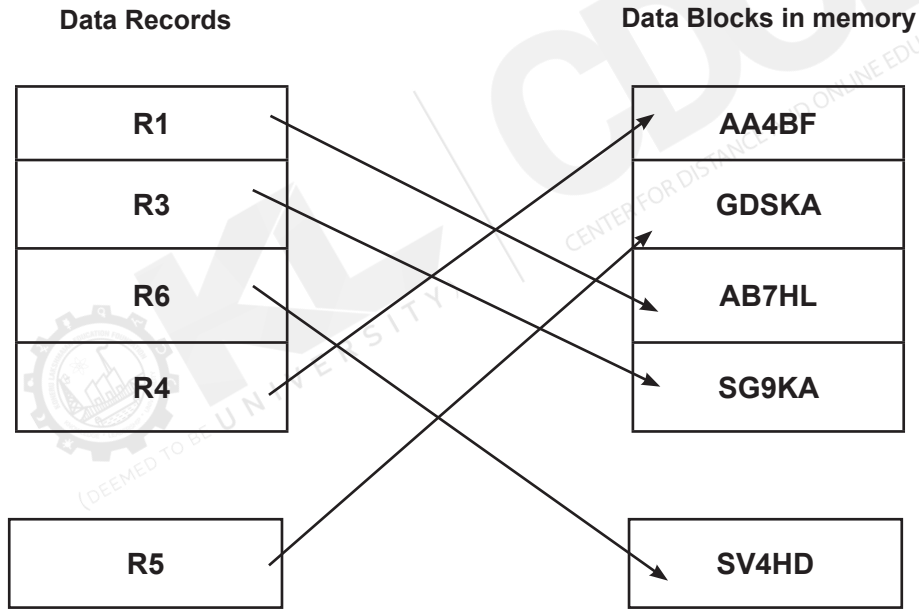


Fig. 12: Hash File Organisation 1

- When records are to be fetched using hash key columns, an address is generated, and then the entire record is accessed using this mysterious box log. Similarly, if a new record needs to be inserted, an address is generated using a hash key, and the record will be directly inserted at that place. This similar approach is used for deleting and updating.
- This method does not have to search and sort all the files. With this approach, every record is placed randomly in memory.

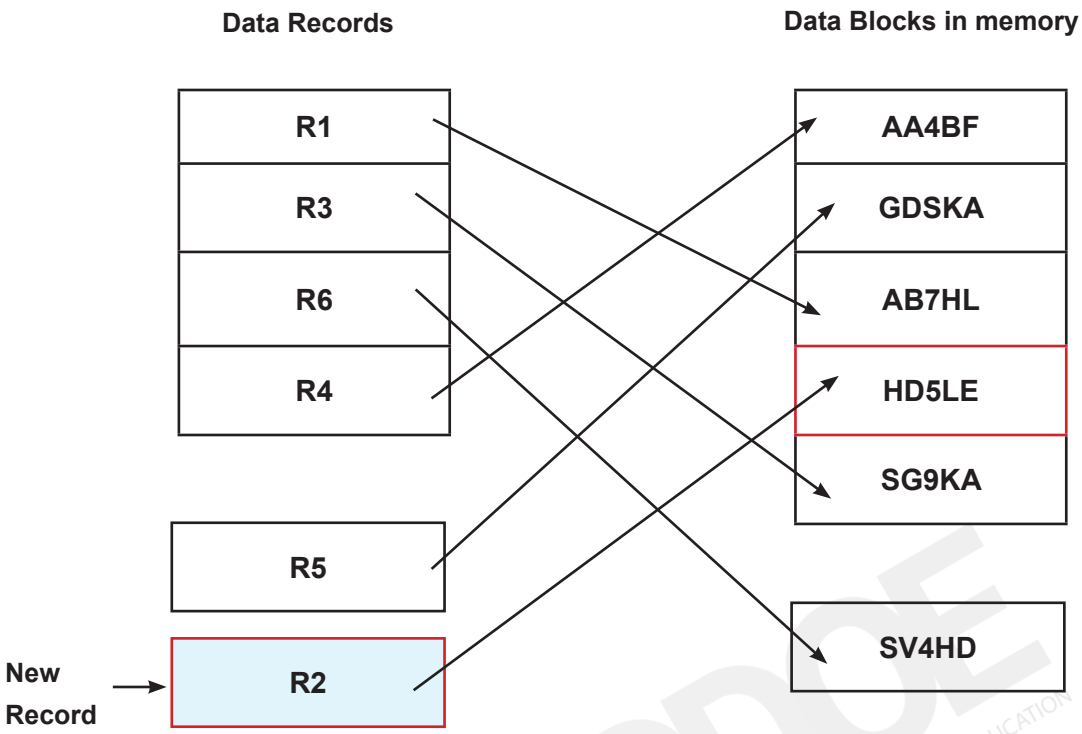


Fig. 13: Hash File Organisation 2

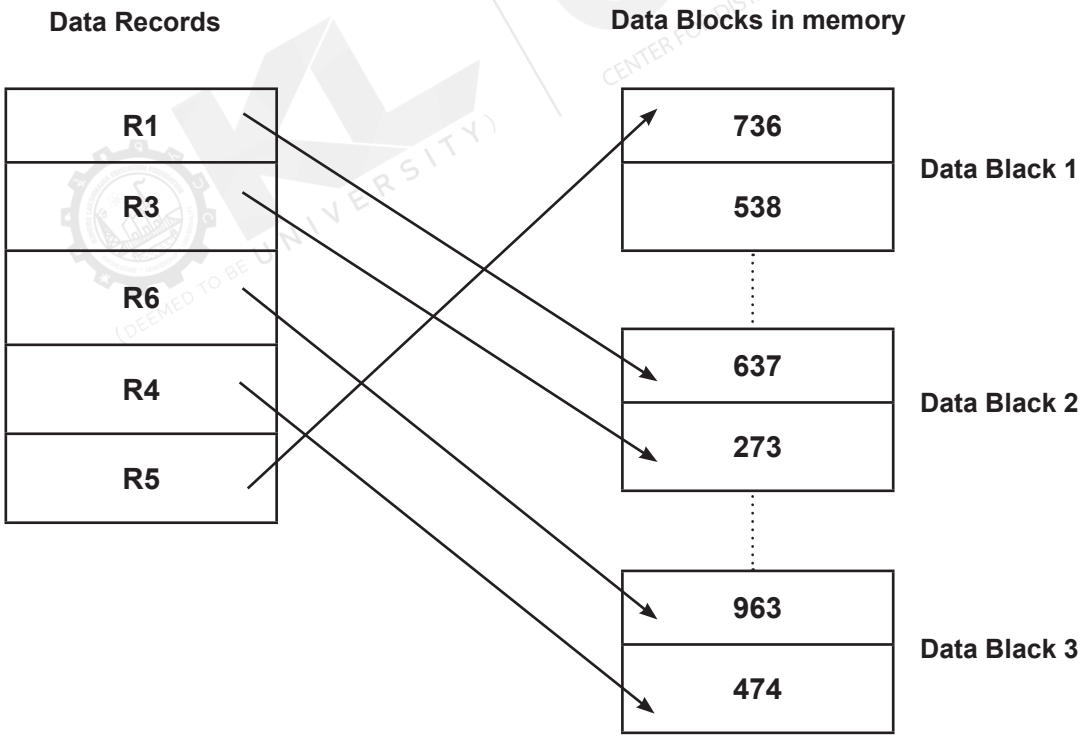


Fig. 14: Hash File Organisation 3

Insertion of a new record

Let there be 5 records, R1, R3,, R6,, R4,, and R5, in a heap. Suppose we must insert one record, R2, into the heap. When data block 3 is full, it will be placed in any database chosen by dbms, such as data block 1.

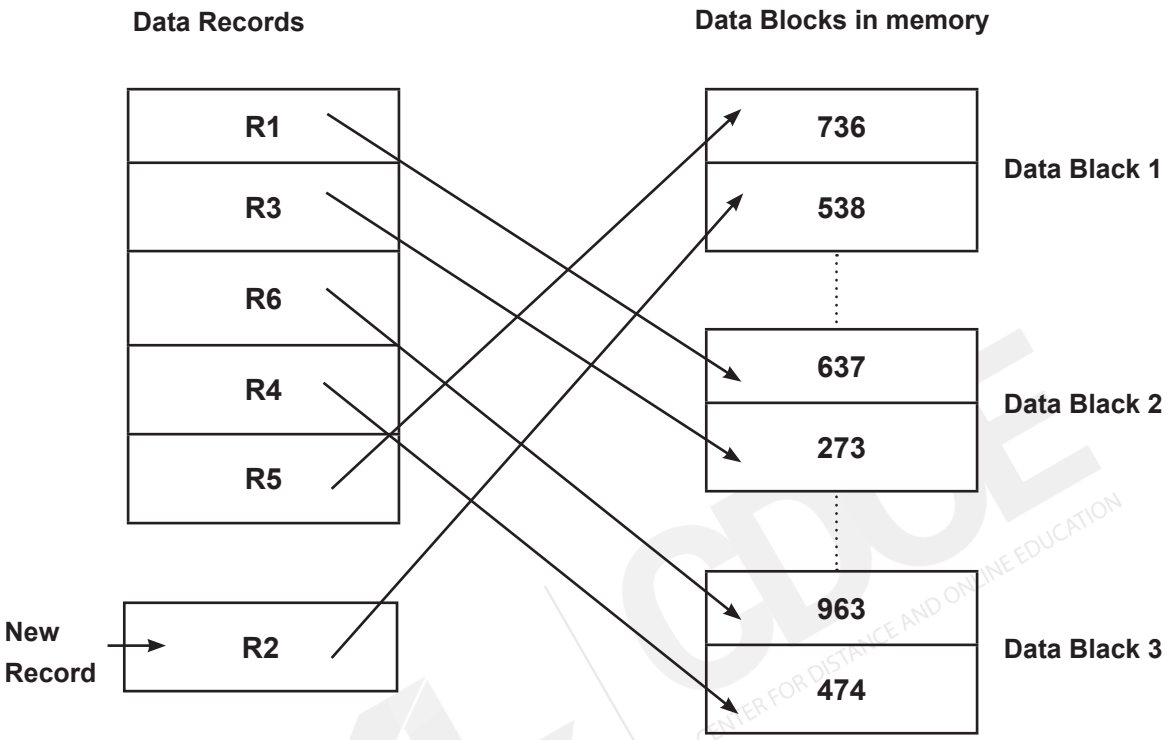


Fig. 15: Insertion of a new record

If we want to perform any search, update, or delete operation on heap file organisation, we need to go from the beginning of the data in the required file until we find our record.

If the database size is huge, then searching, updating or deleting records takes time because there is no sorting or ordering of records. In the case of a heap file organisation, we need to search through all the data until we reach the requested record.

B+ File Organisation

In other words, B+ tree file organisation is a newer version of an indexed sequential access method. It stores records in a File using a tree structure; let's forget the technical part.

It employs a key-index-like structure in which the primary key sorts the records relationally. An index value is created and aligned with each primary key.

A B+ tree is a BST (binary search tree) with at least two children. In this approach, the records are saved only at a leaf node. Intermediate nodes point out to the leaf nodes. There are no records regarding them.

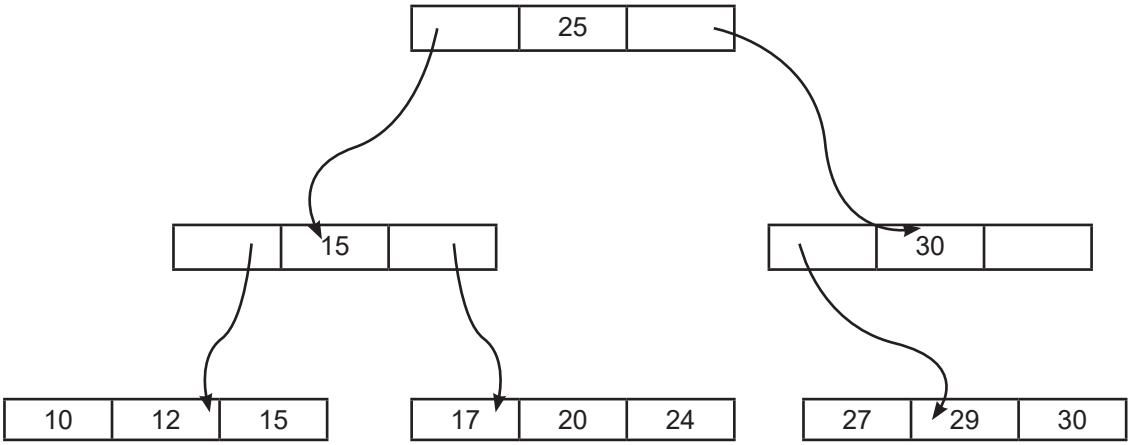


Fig. 16: B+ File Organisation

The B+tree above is such that:

- This tree has a root node only, whose key value is 25.
- There is an intermediate layer of nodes. It doesn't store records but serves as a pointer to the leaf node.
- The nodes to the left of the root node access the previous value to the root, and those on the right of the root generated give the next value, the right being 30, respectively.
- Only one leaf node is present, holding only the following values: 10, 12, 17, 20, 24, 27, and 29.
- With all leaf nodes balanced, a query for any record is very straight up.
- In this way, any records can be accessed via a single path and looked up very easily.

Indexed Sequential Access Method (ISAM)

ISAM is an advanced sequential file organisation method. The primary key store's records are in the file. A generated index value is established for each primary key, which is then linked with the record. This index holds the record's address in the file.

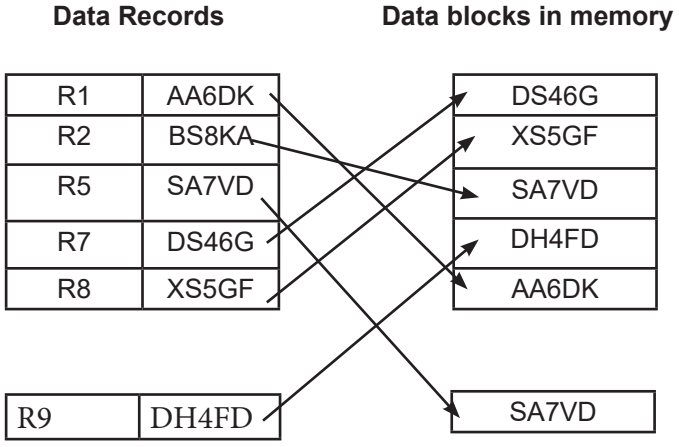


Fig. 17: Indexed Sequential Access Method (ISAM)

If a record must be extracted based on its index value, the data block address is first retrieved, and then the record is extracted from memory.

Cluster file organisation

The organisation of files into clusters When two or more records are stored together in a single file, the record is referred to as a cluster record. Key attributes would remain stored once, so these files have a single data block and two or more tables. This method reduces the cost of having to search on different files and to be able to retrieve for only certain conditions.

When we apply this clustering method to one table right down until it retrieves records from both tables based on the abovementioned condition, it will only provide a few records. For example, we are retrieving records for only particular departments in the given instance. This method cannot be used to retrieve the records for the entire department.

Employee

Emp_ID	Emp_Name	Address	Dep_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amella	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

Department

Dep_ID	Dep_Name
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

Dep_ID	Dep_Name	Emp_ID	Emp_Name	Address
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemstry	3	David	Mumbai

Fig. 18: Cluster file organisation

This method does not allow direct inserting, updating, or deleting records. The data is arranged according to the key we look for. The cluster key is a kind of key with which tables are joined.

Cluster file organisation types include:
There are two types of cluster file organisation:

1. Clusters with indexes: Records in an indexed cluster are stored together after being grouped according to the cluster key. An example of an indexed cluster is the relationship between the department and the employee mentioned above. In this case, all records are sorted according to the cluster key, DEP_ID.
2. **Clusters of Hashes:** It is comparable to the indexed cluster. In a hash cluster, we construct the hash key value for the cluster key and store the records with the same value rather than storing the records according to the cluster key.

Query Processing and Optimisation

Query processing involves transforming high-level queries into efficient execution plans.

1. Steps:

- **Parsing:** Checks syntax and converts the query into an internal format.
- **Query Optimisation:** Chooses the best execution strategy by analysing different possibilities.
- **Execution:** Executes the optimised query plan.

2. Optimisation Strategies:

- Use indexes to speed up searches.
- Reduce the number of disk accesses by limiting intermediate results.
- Reorder joins or conditions to minimise processing time.

Self-Assessment Questions

6. What is the main characteristic of a sequential file organisation?

   A). Records are stored in random order.
   B). Records are stored sequentially, one after another.
   C). Records are stored based on hash functions.
   D). Records are grouped using clustering techniques.

7. Which file organisation method stores records only at the leaf nodes of a tree structure?

   A). Heap file organisation
   B). Sequential file organisation
   C). B+ file organisation
   D). Cluster file organisation

8. In heap file organisation, what happens to a new record when a data block is full?

   A). It is inserted in the next sequential data block.
   B). It is discarded until space is available.
   C). It is placed in any available free data block.
   D). It is sorted and then inserted in the current block.

9. Which of the following is a key advantage of the Indexed Sequential Access Method (ISAM)?

   A). It does not use indexing for data retrieval.
   B). It allows quick access to records using an index value.
   C). It eliminates the need for a primary key.
   D). It does not require additional storage for the index.

10. What is the primary purpose of cluster file organisation?

    A). To sort records based on primary keys.
    B). To group related records from multiple tables into one file.
    C). To randomly place records in memory using a hash function.
    D). To store fixed-length records sequentially.

Summary

- Indexing in DBMS improves query performance by minimising disk access and using structures like search keys and pointers.
- Ordered indices store data in sorted order, enabling efficient searching compared to linear scans.
- Primary indexing uses primary keys with a 1:1 relationship, optimised for quick lookups using dense or sparse indexes.
- Dense indexes have entries for every key value, while sparse indexes use fewer entries and point to data blocks.
- Clustering indices group-related records based on non-primary keys, creating clusters for efficient retrieval.
- Secondary indexing adds a second-level index to handle large datasets by breaking mappings into smaller, manageable levels.
- File organisation determines how records are stored using sequential, heap, hash, and B+ trees.
- Sequential file organisation appends or sorts of records in a fixed order, while heap organisation stores records in any free space.
- Hash file organisation computes hash functions for key-based record placement, offering fast access but lacking order.
- B+ tree and ISAM use hierarchical structures or combined indexing methods for efficient multi-level data access.

Terminal Questions

1. What is indexing in DBMS, and how does it improve query performance?
2. Explain the differences between dense and sparse indexing.
3. What are the advantages of using a clustering index?
4. Describe the various types of file organisation in DBMS.
5. How does the B+ tree file organisation optimise data retrieval compared to a sequential file?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	B
2	B
3	B
4	B
5	B
6	B
7	C
8	C
9	B
10	B

Activity

Activity type: Online **Duration: 1 Hour**

Take data from the preloaded sample database (e.g., Employee Table with attributes: Employee_ID, Name, Dept_ID, Salary).

- Create a primary index on the Employee_ID column.
- Create a dense index for Dept_ID and a sparse index for Salary.

Glossary

- **Sparse Index:** Index entries for selected key values only.
- **Hash Function:** Algorithm that maps keys to specific memory locations.
- **B+ Tree:** A self-balancing tree structure for sorted data access.
- **Cluster Key:** Attribute(s) used to group related records in clusters.
- **ISAM:** A hybrid file organisation method combining sequential and index-based storage.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **Indexing:** https://www.geeksforgeeks.org/indexing-in-databases-set-1/
- **File organisation:** https://www.javatpoint.com/dbms-file-organization#:~:text=File%20organization%20is%20a%20logical,placed%20on%20the%20storage%20medium.

Video Links

Topic	Link
Indexing	https://www.youtube.com/watch?v=3G293is403I
File organisation	https://www.youtube.com/watch?v=2Dz7H-eMyJg

Image Credits

Fig. 1: Indexing Methods	Self-generated
Fig. 2: Dense index	Self-generated
Fig. 3: Sparse index	Self-generated
Fig. 4: Clustering Index 1	Self-generated
Fig. 5: Clustering Index 2	Self-generated
Fig. 6: Secondary Index	Self-generated
Fig. 7: File Organisation	Self-generated
Fig. 8: Pile File Method	Self-generated
Fig. 9: Insertion of the new record	Self-generated
Fig. 10: Sorted File Method	Self-generated
Fig. 11: Insertion of the new record	Self-generated
Fig. 12: Hash File Organisation 1	Self-generated
Fig. 13: Hash File Organisation 2	Self-generated
Fig. 14: Hash File Organisation 3	Self-generated
Fig. 15: Insertion of a new record	Self-generated
Fig. 16: B+ File Organisation	Self-generated
Fig. 17: Indexed Sequential Access Method (ISAM)	Self-generated
Fig. 18: Cluster file organisation	Self-generated

Keywords

- Indexing
- Dense Index
- File Organisation
- Hash File
- B+ Tree
- ISAM

DATABASE SYSTEMS

MODULE 4

Transaction Management and Data Recovery

Module Description

This module introduces participants to essential concepts and techniques in transaction management and data recovery within database systems. Transaction management is crucial for handling operations that involve multiple steps or processes, ensuring that all parts work together smoothly. In this module, participants will explore the ACID properties (Atomicity, Consistency, Isolation, and Durability) that help maintain database stability during transactions. These properties prevent data loss or corruption when multiple operations or users access the same data.

The module also covers various concurrency control techniques—methods that manage simultaneous transactions in a system. Participants will grasp lock-based, timestamp-based, and multi-version control techniques, ensuring that transactions don't interfere and allowing databases to handle multiple users or processes simultaneously without error.

To support system reliability, participants will study recovery techniques that restore data if an error occurs, such as during a system crash. Topics include Shadow Paging and the ARIES algorithm, key methods to recover databases to their last consistent state and ensure robust data protection. Shadow Paging creates database backups, allowing quick restoration, while ARIES provides detailed logging to reverse or repeat changes as needed.

By the end of this module, students will interpolate how to maintain consistent, reliable databases, even in complex or multi-user environments. These skills are fundamental for database administration, IT, and data management careers, where data integrity and availability are critical.

This module consists of **two** units.

Unit 4.1: Transaction Management
Unit 4.2: Data Recovery Techniques

MODULE 4

Transaction Management and Data Recovery

Unit 1

Transaction Management

Unit Table of Contents

Unit 4.1 Transaction Management

Aim

To enable students to interpolate transaction management, focusing on processing issues, states, ACID properties, system logs, and concurrency control techniques.

Instructional Objectives

This unit intends to:

- Define transaction management and its purpose in a database system
- List the ACID properties and describe each briefly
- Recognise the purpose of system logs in transaction management

Learning Outcomes

Upon completion of the unit, you will be able to:

- Explain the significance of ACID properties in maintaining database consistency
- Describe how system logs contribute to data recovery in transaction management
- Illustrate the transaction lifecycle, including states and transitions

4.1.1 Introduction to Transaction Management

Definition of Transaction:

A transaction is a set of operations that together complete a specific task. It includes all the steps needed to finish a logical action. Transactions often involve changes to data in a database. One important job of a database management system (DBMS) is to keep data safe in case of system crashes by saving and restoring data when the system restarts. Each time a user's program runs in a DBMS, it creates a new transaction, which has a set number of steps. Running the same program multiple times will result in multiple transactions.

Example of a Transaction:
Consider withdrawing cash from an ATM. Here's the sequence of steps in the transaction:

1. Start the transaction.
2. Insert your ATM card.
3. Choose your preferred language.
4. Select the "Savings Account" option.
5. Enter the withdrawal amount.
6. Enter your PIN.
7. Wait for processing.
8. Collect your cash.
9. Complete the transaction.

Transaction States in DBMS:

A transaction log is a file used to track all transaction activities. Once a transaction is successfully completed (committed), the log file is removed. In DBMS, transactions go through various states to keep data consistent. Here are the main states:

1. **Active State:** The transaction is active while instructions are running. If all operations (like reading and writing data) succeed, the transaction moves to the "Partially Committed" state. If an error happens, it goes to the "Failed State."
2. **Partially Committed State:** Once all operations are complete, the data changes are saved temporarily in memory. If these changes are successfully saved to the database, the transaction moves to the "Committed State." If there's a failure, it goes to the "Failed State."
3. **Failed State:** If any step in the transaction fails, or if data cannot be saved to the database, the transaction enters the "Failed State."

4. **Aborted State:** After a failure, the transaction goes to the "Aborted State," and any changes made are deleted or "rolled back" because they were only temporary.
5. **Committed State:** This state means all changes have been successfully saved to the database, and the transaction is fully complete.
6. **Terminated State:** When there is no rollback needed, or the transaction moves from the "Committed State," it ends in the "Terminated State." This indicates that the system is now ready for a new transaction.

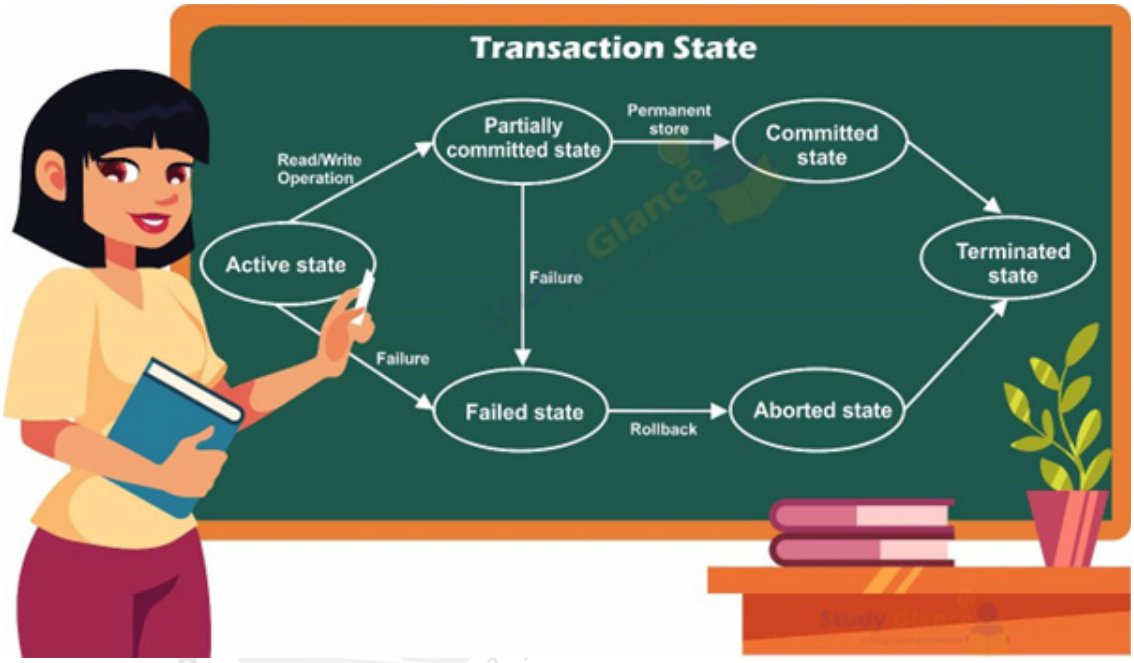


Fig. 1: Types of transaction states

ACID Properties of Transactions (Desirable Database Properties)

A database transaction has four main properties to keep data consistent before and after any operation. These are:

1. Atomicity

- Atomicity means that a transaction is all-or-nothing: it either fully or doesn't happen.
- A transaction can't partially occur—it's treated as a single unit.
- There are two outcomes:
  - **Abort:** If a transaction fails, all changes are erased as if it never happened.
  - **Commit:** If a transaction succeeds, all changes become permanent and visible.

2. Consistency

- Consistency ensures that the database remains valid before and after a transaction.
- After a transaction, the database is either in its original or new stable state.
- Transactions must follow integrity rules to keep data accurate and meaningful.

3. Isolation

- Isolation means that transactions run independently.
- If one transaction uses a data item, another cannot use it until the first one is finished.
- This ensures that each transaction is unaffected by others running simultaneously.

4. Durability

- Durability means that its changes are permanent once a transaction is complete.
- Even if there's a system crash or failure, the changes made by a committed transaction remain saved.
- The DBMS recovery system ensures that data remains unchanged despite system errors.

These ACID properties help databases maintain reliable and consistent performance, especially during concurrent and complex operations.

Self-Assessment Questions

1. Which of the following properties of a transaction ensures that a transaction either completes entirely or does not occur at all?

   A). Consistency
   B). Isolation
   C). Atomicity
   D). Durability

2. In which transaction state are all 'read and write' operations completed, but changes have not been made permanent in the database?

   A). Active
   B). Committed
   C). Partially Committed
   D). Terminated

3. The property that ensures that one transaction's changes are not visible to other transactions until completion is called:

   A). Consistency
   B). Atomicity
   C). Isolation
   D). Durability

4. If a transaction fails during execution, which state does it enter, necessitating that any temporary changes be discarded or rolled back?

   A). Partially Committed
   B). Failed
   C). Aborted
   D). Terminated

5. The "Durability" property in DBMS transactions ensures that:

   A). The database is consistent before and after the transaction.
   B). Transactions can be executed without interfering with each other.
   C). Changes made by a completed transaction remain permanent even after a system failure.
   D). The transaction can be aborted if any operation fails.

4.1.2 Lock-Based Protocols

A lock is a setting linked to a data item that shows if and how transactions can access it. Locks help manage how multiple transactions access the same data item simultaneously. This method ensures that only one transaction accesses a data item at a time. Here are two common types of locks and terms used in this method:

Types of Locks

1. **Shared Lock (S):**
    - Also called a Read-only lock.
    - Multiple transactions can share this lock because it only allows data to be read, not updated.
    - To request a shared lock, a transaction uses the “lock-S” command.
2. **Exclusive Lock (X):**
    - This lock allows both reading and writing of data.
    - Only one transaction can hold this lock on a data item.
    - To request an exclusive lock, a transaction uses the “lock-X” command.

These locks help organise transactions and avoid conflicts in accessing the same data items.

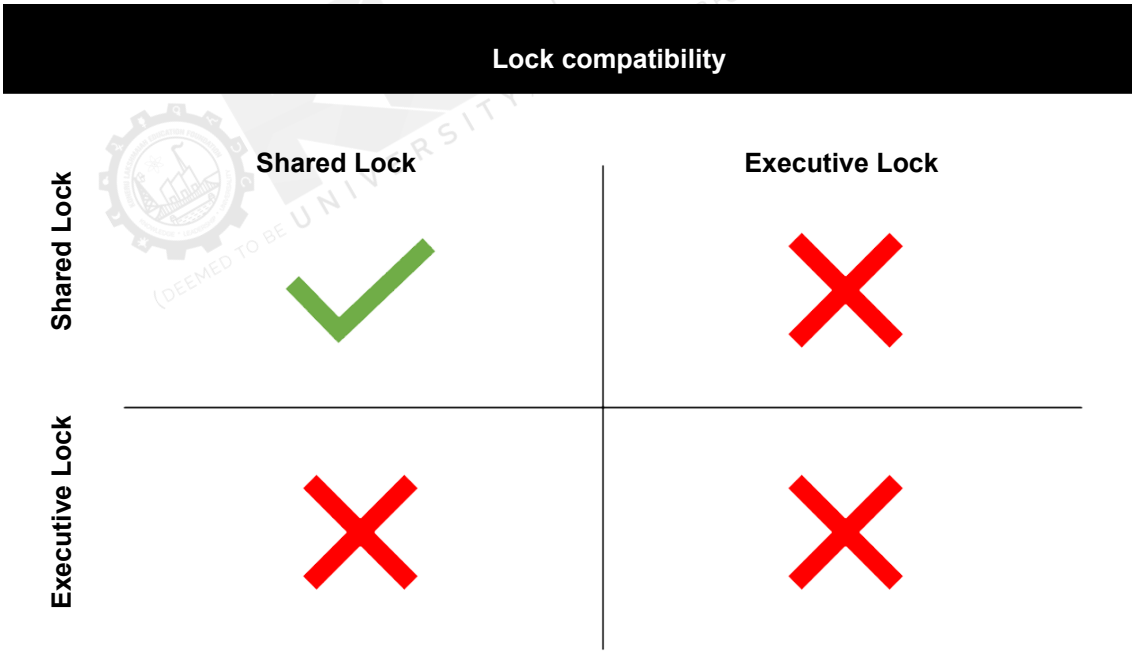


Fig. 2: Types of lock

Types of Lock-Based Protocols

1. **Simplistic Lock Protocol**
    - This is the simplest method for locking data during a transaction.
    - All transactions can get a lock on data before making any changes (like inserting, deleting, or updating).
    - The data item is unlocked after the transaction completes.
2. **Pre-Claiming Lock Protocol**
    - This protocol checks what data items need locks before the transaction starts.
    - It requests locks on all required data items from the DBMS before starting.
    - If all locks are granted, the transaction proceeds; if not, the transaction can be rolled back and retried once all locks are available.
    - Locks are released when the transaction finishes.
3. **Two-Phase Locking (2PL)**
    - Transactions following the Two-Phase Locking protocol have two phases:
        - **Growing Phase**: Locks can be acquired but not released.
        - **Shrinking Phase**: Locks can be released, but no new locks can be acquired.
4. **Strict Two-Phase Locking Protocol**
    - This is a stricter version of 2PL where all Exclusive (X) locks held by a transaction are only released after the transaction commits.

Timestamp-Based Concurrency Control

This method ensures safe and consistent transaction execution by using timestamps to organise the execution order of transactions, even when multiple transactions run simultaneously.

What is the Timestamp Ordering Protocol?

- Transactions are ordered by their timestamps, making sure they execute in a conflict-free sequence.
- Each transaction is assigned a unique timestamp when it begins, guiding the execution order for read and write operations.
- Two timestamps are used for each data item (X):
    - **W_TS(X)**: The highest timestamp of any transaction successfully written to X.
    - **R_TS(X)**: The highest timestamp of any transaction that is successfully read from X.

Basic Timestamp Ordering (TO) Protocol

- Each transaction gets a timestamp when it enters the system, determining its order.
- Conflicting read and write actions are managed in timestamp order:
    - **Write Operation (W_item(X))**: If the transaction's timestamp is older than either the R_TS(X) or W_TS(X), the transaction is aborted and rolled back. If not, the write is allowed, and W_TS(X) is updated.
    - **Read Operation (R_item(X))**: If W_TS(X) is newer than the transaction's timestamp, the transaction is aborted. Otherwise, the read proceeds and R_TS(X) is updated.
- This protocol ensures conflict-serialisability and prevents deadlock. However, cascading rollbacks can still occur if a transaction depends on data from a later aborted transaction.

Pros and Cons of the Basic TO Protocol

- **Advantages**: Ensures deadlock-free schedules and maintains consistent transaction order.
- **Disadvantages**: Cascading rollbacks can occur if a dependent transaction is aborted, affecting other transactions that rely on its data.

Timestamp Ordering protocol ensures serialisability since the precedence graph will be of the form:

Fig. 3: Timestamp Ordering

Precedence Graph for Timestamp Ordering

- **Deadlock-Free**: Timestamp protocol avoids deadlock since transactions don't wait on each other.
- **Limitations**: However, schedules created with timestamp order may not be cascade-free or fully recoverable.

Strict Timestamp Ordering

- In this variation, a transaction T that wants to perform a read or write (R_item(X) or W_item(X)) will be delayed if TS(T) > W_TS(X). This delay continues until the last transaction written to X has been committed or aborted.
- This method makes schedules both strict and conflict serialisable.

Advantages of Timestamp Ordering Protocol

- **High Concurrency**: Allows high concurrency, as transactions don't interfere with each other.
- **Efficient**: It doesn't require locks, making it efficient and scalable to handle many transactions.
- **No Deadlocks**: Since no locks are involved, deadlocks can't occur.
- **Improved Performance**: Concurrent execution boosts the database system's overall performance.

Disadvantages of Timestamp Ordering Protocol

- **Limited Precision**: The protocol's effectiveness depends on the precision of timestamps. If timestamp precision is insufficient, transactions may be blocked unnecessarily.
- **Order Management**: Correctly managing timestamps is crucial to avoid inconsistencies.
- **Clock Synchronisation**: Synchronisation is necessary for accurate timestamping. Unsynced clocks can lead to incorrect transaction ordering.
- **Timestamp Allocation**: Allocating unique timestamps in distributed systems, where transactions may start at various locations, can be challenging.

Self-Assessment Questions

6. What type of lock allows multiple transactions to read a data item but not modify it?

   A). Exclusive Lock (X)
   B). Shared Lock (S)
   C). Write Lock (W)
   D). None of the above

7. In which phase of the Two-Phase Locking (2PL) protocol are new locks acquired, but none are released?

   A). Growing Phase
   B). Shrinking Phase
   C). Active Phase
   D). Terminated Phase

8. The Strict Two-Phase Locking Protocol requires that all exclusive locks are released:

   A). Immediately after the transaction is completed
   B). Only after the transaction commits
   C). When a conflicting transaction requires access
   D). After the transaction aborts

9. Which of the following is an advantage of the Timestamp Ordering (TO) protocol?

   A). Requires fewer timestamps
   B). Ensures deadlock-free scheduling
   C). Allows cascading rollbacks
   D). Supports limited concurrency

10. In Basic Timestamp Ordering, if a transaction T tries to write an item X, which condition would cause it to be aborted?

    A). TS(T) < R_TS(X) and TS(T) < W_TS(X)
    B). TS(T) > R_TS(X) and TS(T) > W_TS(X)
    C). TS(T) < R_TS(X) only
    D). TS(T) > W_TS(X) only

Summary

- A transaction in a Database Management System (DBMS) is a collection of operations that execute a logical unit of work, often involving data changes.
- Transactions ensure data consistency and integrity despite system failures by adhering to ACID properties (Atomicity, Consistency, Isolation, and Durability).
- The stages a transaction can pass through are Active, Partially Committed, Failed, Aborted, Committed, and Terminated.
- Lock-based protocols, such as Shared and Exclusive locks, help manage concurrent transactions, ensuring that only one transaction can access data at a time.
- Protocols like Two-Phase Locking (2PL) and Timestamp Ordering manage concurrency by controlling the order and method of data access to prevent conflicts and ensure consistency.
- Timestamp-based Concurrency Control uses timestamps to manage transaction execution, offering high concurrency without deadlocks.
- However, it has limitations, such as managing precise timestamps and avoiding cascading rollbacks.

Terminal Questions

1. What is a transaction in DBMS, and how does it ensure data consistency?
2. List and explain the ACID properties of transactions with examples.
3. What is a lock in DBMS, and how does it help with transaction management?
4. What are cascading rollbacks, and why do they occur in timestamp ordering protocols?
5. How does Strict Two-Phase Locking differ from Basic Two-Phase Locking, and what benefits does it offer?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	C
2	C
3	C
4	C
5	C
6	B
7	A
8	B
9	B
10	A

Activity

Activity type: Online **Duration: 1 Hour**

Choose a common transaction scenario (e.g., booking tickets, online shopping, bank transactions).

- List steps required in each transaction, noting the ACID properties for each (e.g., Atomicity for purchase completion, Consistency for inventory updating).

Glossary

- ACID Properties: **Fundamental properties ensuring reliable transaction processing—Atomicity, Consistency, Isolation, and Durability.**
- Shared Lock (S): **A lock that allows multiple transactions to read a data item without making changes.**
- Exclusive Lock (X): **A lock allowing only one transaction to read and write a data item.**
- Two-Phase Locking (2PL): **This protocol ensures serialisability by dividing locking and unlocking into two distinct phases.**
- Timestamp: **A unique identifier assigned to each transaction based on its start time.**
- Concurrency Control: **DBMS techniques to manage simultaneous transaction execution, ensuring data consistency and integrity.**

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **Transaction Management:** https://www.geeksforgeeks.org/transaction-management/
- **Lock-Based Protocols:** https://www.geeksforgeeks.org/lock-based-concurrency-control-protocol-in-dbms/

Video Links

Topic	Link
Lock-Based Protocol	https://www.youtube.com/watch?v=MX_Gtn4l9ys
Transaction Management	https://www.youtube.com/watch?v=eYQwK-i7P8MM

Image Credits

Fig. 1: Types of transaction states	https://www.google.com/url?sa=i&url=https%3A%2F%2F-studyglance.in%2Fdbms%2Fdisplay.php%3Ftno%3D38%26topic%3DTrans-action-State-in-DBMS&psig=AOv-Vaw1PXOPAkjelkyWVJvRlJXl-N&ust=1731406946742000&source=imag-es&cd=vfe&opi=89978449&ved=0CBcQjhxqF-woTCLjFntOH1IkDFQAAAAAdAAAAABAe
Fig. 2: Types of lock	https://www.google.com/url?sa=i&url=https%3A%2F%2F-medium.com%2F%40diretnandom-nan%2Fthe-db-the-lock-and-the-in-dex-8f1955b6ed09&psig=AOv-Vaw1eK-mZVPbKvCSUYP7w_RoR&ust=1731407142058000&source=imag-es&cd=vfe&opi=89978449&ved=0CBcQjhxqF-woTCNC6grCI1IkDFQAAAAAdAAAAABAc
Fig. 3: Timestamp Ordering	Self-generated

Keywords

- ACID Properties
- Atomicity
- Concurrency Control
- Timestamp Ordering Protocol
- Cascading Rollbacks
- Strict Two-Phase Locking
- Transaction Log

MODULE 4

Transaction Management and Data Recovery

Unit 2

Data Recovery Techniques

Unit Table of Contents

Unit 4.2 Data Recovery Techniques

Aim

To enable learners grasp data recovery techniques, such as shadow paging and the ARIES algorithm, to build strong skills for keeping data safe and ensuring databases remain reliable.

Instructional Objectives

This unit intends to:

- Define basic recovery techniques used in databases
- Recall the purpose of shadow paging in data recovery
- Identify the components of the ARIES algorithm

Learning Outcomes

Upon completion of the unit, you will be able to:

- Explain how shadow paging contributes to data recovery
- Describe the steps involved in the ARIES algorithm for database recovery
- Discuss the role of recovery techniques in maintaining system resilience

4.2.1 Recovery Techniques: Shadow Paging and the ARIES Algorithm

In database management, recovery techniques are essential for protecting data and ensuring the system can return to a stable state if something goes wrong. Two important methods for recovery are Shadow Paging and the ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) algorithm. Here's a simplified explanation of how each works and their usefulness.

Shadow Paging

Concept: Shadow Paging is a method that helps keep data safe during transactions by creating "shadows" or copies of pages in the database. This system divides the database into small storage units called pages. Each page represents a piece of data in the database and is mapped onto physical storage blocks (i.e., sections on a disk).

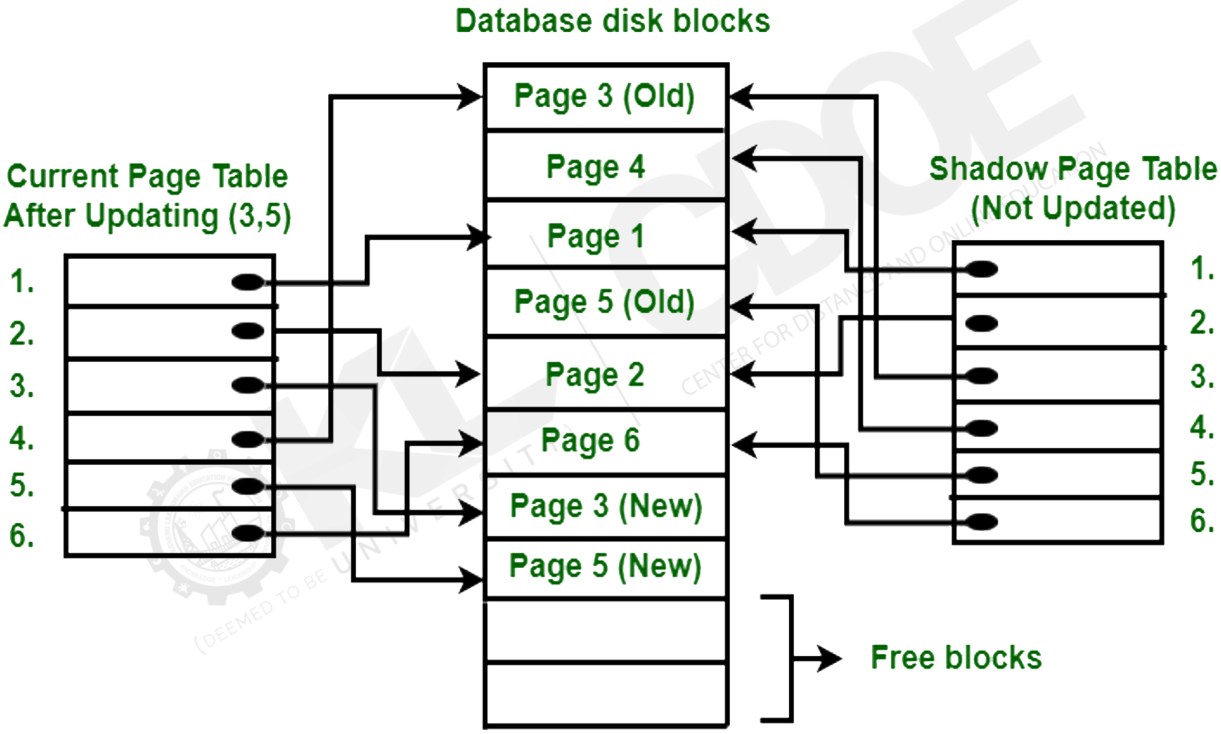


Fig. 1: Shadow Paging

How Shadow Paging Works:

- The technique uses two-page tables, called the Current Page Table and the Shadow Page Table:
  - **Current Page Table**: Tracks the database's most recent or updated pages.
  - **Shadow Page Table**: Holds a copy of the database's state at the beginning of a transaction. This means that it remains unchanged while the transaction is in progress.

Steps in Shadow Paging:

Start of Transaction: The system copies the current page table to create the shadow page table. This shadow page table is saved on disk as a backup of the original database state.

Performing Write Operations:

- If a change needs to be made (e.g., updating a page), the system first finds an empty block on the disk.
- It then copies the page to this free block to hold the new data while the current page table updates to point to this new page.
- The shadow page table still points to the old page version, keeping the old data safe until the transaction is completed.

After the Transaction:

Once the transaction is finished, both tables become identical again, meaning all data is now up to date and safe.

Example: Imagine you want to update two pages, page 3 and page 5. Shadow paging would follow these steps:

- When updating page 3, the system finds an empty block on the disk, copies the old page as a “new” page, and points the current page table to this new version. The shadow page table still points to the original page 3 until the transaction ends, which keeps it safe in case of errors.

ARIES Algorithm (Algorithm for Recovery and Isolation Exploiting Semantics)

The ARIES algorithm is another recovery method designed to restore the database to a stable state after a crash or unexpected shutdown. ARIES follows a protocol called Write Ahead Logging (WAL), which keeps a log of each operation that can be undone or redone as needed.

Key Features of ARIES:

Write Ahead Log (WAL): Before any changes are saved to the database, a log record describes the change. This log helps ensure data can be recovered.

Log Records:

- **Undo-only:** Keeps a “before” image of the data so it can be rolled back if necessary.
- **Redo-only:** Stores the “after” image, allowing the operation to be redone.
- **Undo-redo:** Stores both “before” and “after” images, offering more flexibility in recovery.

Log Sequence Number (LSN):

- Every log entry has a unique Log Sequence Number (LSN) to track each operation's order. Each data page also has a page LSN field, updated to match the LSN of the last change.
- A checkpoint record is periodically written to the log, capturing a snapshot of transactions in progress and any pages with unsaved changes.

Recovery Phases in ARIES: If the system crashes, ARIES follows three phases to restore it:

- **Analysis:** ARIES determines the earliest point in the log needed for recovery and scans forward from the checkpoint to recreate the database's state at the time of the crash.
- **Redo:** ARIES goes through the log from the earliest necessary LSN and reapplies (or "re-does") changes to ensure the database includes all updates completed before the crash.
- **Undo:** ARIES undoes any uncommitted changes by moving backwards through the log. This removes changes from "loser" transactions or those not completed before the crash.

Why Shadow Paging and ARIES are Important

- **Data Protection:** Both methods ensure data remains consistent and recoverable after a failure, protecting against data loss.
- **System Resilience:** They allow databases to recover quickly, vital for businesses and applications that rely on consistent data access.
- **Transaction Safety:** Each technique helps maintain a safe backup state by using shadow tables (Shadow Paging) or detailed logs (ARIES) to save only fully completed transactions.

Self-Assessment Questions

1. What is the primary purpose of recovery techniques in databases?

   A). To enhance transaction speed
   B). To ensure data integrity and system resilience
   C). To decrease data storage costs
   D). To increase user interaction

2. In Shadow Paging, what does the shadow page table represent?

   A). The most recent changes to the database
   B). The original database state at the start of a transaction
   C). Free blocks available on disk
   D). A log of all transaction steps

3. Which table in Shadow Paging is used to record changes during a transaction?

   A). Shadow page table
   B). Log table
   C). Current page table
   D). Checkpoint table

4. What is the first step in Shadow Paging when a transaction starts?

   A). Find free blocks on disk
   B). Create a checkpoint
   C). Copy the current page table to the shadow page table
   D). Write all data to disk

5. Which recovery method follows the Write Ahead Log (WAL) protocol?

   A). Shadow Paging
   B). Two-Phase Commit
   C). ARIES
   D). B-trees

Self-Assessment Questions

6. In ARIES, which log record type allows both undo and redo operations?

A). Undo-only
B). Redo-only
C). Undo-redo
D). Checkpoint

7. What is the purpose of the checkpoint in the ARIES algorithm?

A). To identify free pages in the database
B). To log completed transactions only
C). To capture a snapshot of active transactions and dirty pages
D). To remove old log records

8. During ARIES recovery, in which phase are uncommitted changes removed?

A). Analysis
B). Redo
C). Undo
D). Checkpoint

9. In Shadow Paging, what does the current page table point to?

A). The shadow page table
B). The original database pages
C). The most recent database pages on disk
D). Free blocks on disk

10. In ARIES, what is the function of the Log Sequence Number (LSN)?

A). To store page numbers
B). To track the order of log entries
C). To identify unused blocks on the disk
D). To monitor transaction sizes

Summary

- In database management, recovery techniques are essential to maintain data integrity and system resilience in case of unexpected crashes or failures.
- Two primary recovery methods are Shadow Paging and the ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) algorithm.
- Shadow Paging creates a snapshot of database pages at the beginning of a transaction using a current page table and a shadow page table.
- Changes are only made in the current page table, leaving the shadow page table unaltered until the transaction completes. This ensures a safe state to revert to if necessary.
- Based on Write-Ahead Logging (WAL), the ARIES algorithm records each change in a log before updating the database.
- In case of failure, ARIES restores the database in three phases: Analysis, Redo, and Undo. This ensures that completed transactions are saved and uncommitted transactions are discarded.
- Both methods are designed to safeguard data and ensure database recovery and stability.

Terminal Questions

1. What is the main goal of recovery techniques in database management?
2. Why is the Write-Ahead Log (WAL) important in the ARIES recovery method?
3. How does the checkpoint feature in ARIES contribute to the recovery process?
4. What is the difference between ARIES's undo-only, redo-only, and undo-redo log records?
5. What happens to the shadow page table when a transaction is completed in Shadow Paging?

Answer Keys

Self-Assessment Questions	
Question No.	**Answers**
1	B
2	B
3	C
4	C
5	C
6	C
7	C
8	C
9	C
10	B

Activity

Activity type: Online **Duration: 1 Hour**

Create a shared virtual spreadsheet and set up the current and Shadow Page Table.

- Label each entry in both tables with different Page IDs (e.g., Page 1, Page 2, etc.), representing different data points in a database.

Glossary

- **Shadow Paging:** A recovery technique that creates two tables, a current page table and a shadow page table, to track the database state during transactions.
- **Current Page Table:** The page table stores the most recent versions of database pages during a transaction.
- **Write-Ahead Log (WAL):** This protocol logs all changes before they are applied to the database, enabling recovery by undoing or redoing operations.
- **Log Sequence Number (LSN):** A unique identifier assigned to each log entry to maintain the order of operations.
- **Checkpoint:** A point in the transaction log that records the state of transactions, aiding in faster recovery.

Bibliography

Textbooks

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Database system concepts*. Tata McGraw-Hill.
- Fundamentals of Database Systems. (2001). *Fundamentals of database systems*. Pearson.
- Date, C. J. (2004). *An introduction to database systems*. Addison Publications Pvt Ltd.

Bibliography

External Resources

- Ullman, J. D. (1982). *Principles of database systems*. Galgotia Publications.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database management systems*. Raghu Ramakrishnan & Johannes Gehrke.
- Garcia-Molina, H. (2008). *Database systems: The complete book*. Philadelphia, PA: Pearson Education.

e-References

- **Shadow Paging:** https://www.geeksforgeeks.org/shadow-paging-dbms/
- **ARIES Algorithm:** https://www.geeksforgeeks.org/algorithm-for-recovery-and-isolation-exploiting-semantics-aries/

Video Links

Topic	Link
Shadow Paging	https://www.youtube.com/watch?v=495q-pDJ9zE
ARIES Algorithm	https://www.youtube.com/watch?v=-yIV_vSRE3I

Image Credits

Fig. 1: Shadow Paging	Self-generated

Keywords

- Recovery Techniques
- Shadow Page Table
- ARIES Algorithm
- Write Ahead Log (WAL)
- Log Sequence Number (LSN)
- System Resilience
- Data Integrity
- Database Resilience

DATABASE SYSTEMS

KL UNIVERSITY (DEEMED TO BE UNIVERSITY) | CDOE CENTER FOR DISTANCE AND ONLINE EDUCATION

https://kluonline.edu.in/

supportcdoe@kluniveristy.in