Bangladesh Govt. & UGC Approved
UNIVERSITY OF GLOBAL VILLAGE (UGV), BARISHAL.
THE FIRST SKILL BASED HI-TECH UNIVERSITY IN BANGLADESH

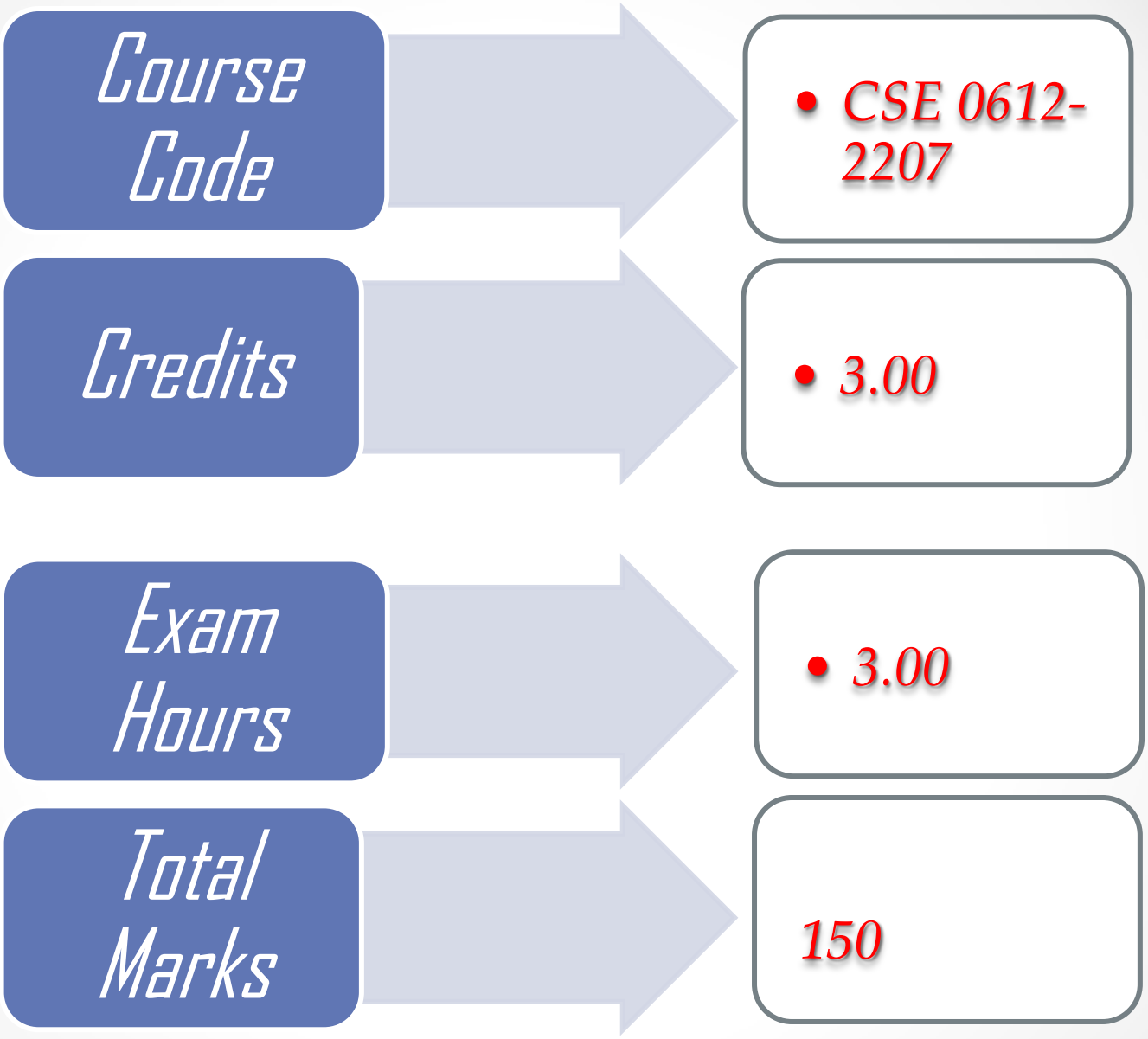
Database Management System

Prepared by:

Md. Abdur Razzak

Asst. Professor.

Department of CSE


Course Code
CSE 0612-2207
Credits
3.00
Exam Hours
3.00
Total Marks
150

CLO 1

- *Describe the fundamental concepts of databases, including data models, database architectures, and database management systems. Students will be able to explain the benefits and limitations of different types of databases, and identify use cases where each type would be appropriate.*

CLO 2

- *Understand and apply principles of database design, including entity-relationship modeling, normalization, and data modeling tools. Students will be able to create and optimize databases that are efficient, scalable, and maintainable.*

CLO 3

- *Create and execute SQL queries to retrieve and manipulate data in a relational database. Students will be able to design and implement SQL statements for a variety of use cases, including querying, updating, and aggregating data.*

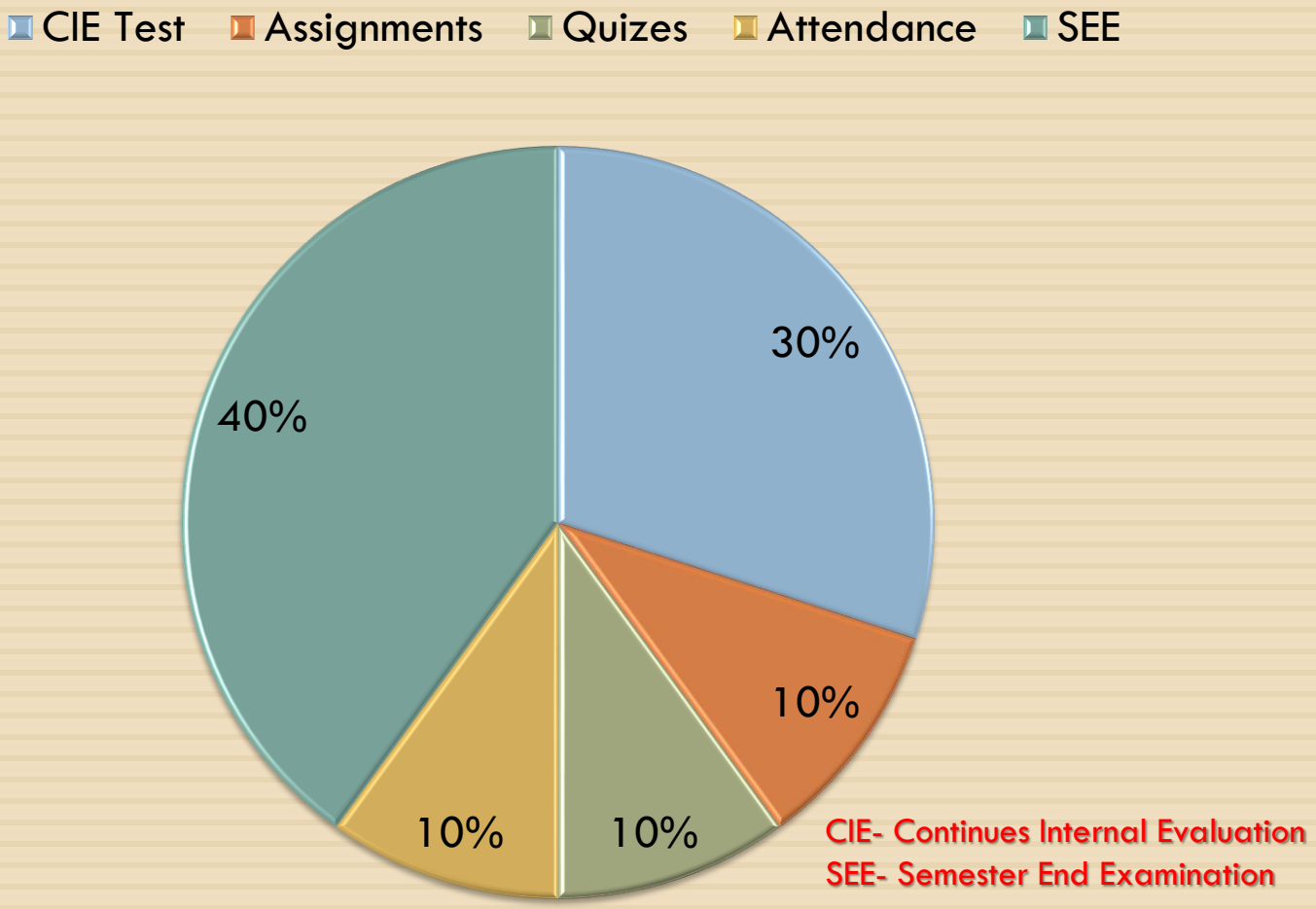
CLO 4

- *Apply database administration and security principles to manage and protect data in a database management system. Students will be able to configure database security settings, backup and restore data, and tune database performance.*

CLO 5

- *Identify and apply principles of NoSQL databases, including document, column family, graph, and key-value stores. Students will be able to analyze the benefits and limitations of different types of NoSQL databases, and identify use cases where each type would be appropriate. They will also be able to create and optimize NoSQL databases for specific use cases.*

Assessment Pattern



CIE- Continues Internal Evaluation
SEE- Semester End Examination

Detailed Assessment Pattern

CIE- Continuous Internal Evaluation (90 Marks)

Bloom's Category Marks (out of 90)	Tests (45)	Assignments (15)	Quizzes (15)	Attendance (15)
Remember	5	03		
Understand	5	04	05	
Apply	15	05	05	
Analyze	10			
Evaluate	5	03	05	
Create	5			

SEE- Semester End Examination (60 Marks)

Bloom's Category	Test
Remember	7
Understand	7
Apply	20
Analyze	15
Evaluate	6
Create	5

Week No	Topic	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
1	-Introduction to Database Systems -Understand the history and importance of database systems	Lecture, Reading Assignments	Quiz, Assignment	**CLO1**
2	Data Models Discuss different types of data models and their Applications	Lecture, Reading Assignments	QA, Quiz, Assignment	**CLO1, CLO2**
3	Database Design ,ER-Diagram and Unified Modeling Language Explore ER modeling and normalization techniques	Lecture, Case Studies	Quiz, Assignment	**CLO2**
4	The relational algebra and calculus Relational algebra: introduction, Selection and projection, set operations, renaming, Joins, Division, syntax Calculus: Tuple relational calculus, Domain relational Calculus.	Lecture, Case Studies	Quiz, Assignment	**CLO2**
5	**The relational algebra and calculus** **renaming, Joins, Division, syntax Calculus: Tuple relational calculus, Domain relational Calculus.**	**Lecture, Hands-on Labs**	**Quiz, Lab Reports**	**CLO3**

6	**SQL Basics** **Learn basic SQL syntax and commands**	**Lecture, Case Studies**	**Quiz, Assignment**	**CLO2**
7	SQL Basics (Continued) basic SQL syntax and commands	Lecture, Case Studies	Quiz, Case Study Reports	CLO4
8	Advanced SQL Understand advanced SQL queries and transaction management	Lecture, Problem Solving	Quiz, Problem Sets	CLO4
9	Constraints, What is constraints, types of constrains, Integrity constraints,	Lecture, Problem Solving	Quiz, Problem Sets	CLO5
10	Views Views: Introduction to views, data independence, security, updates on views, comparison between tables and views	Lecture, Guest Lectures	Quiz, Presentation	CLO5
11	Indexing Basic concepts, Ordered Indices,	Lecture, Discussions	Quiz, Assignment	CLO5

12	**Hashing** **Hash Based Indexing, Tree based Indexing**	**Lecture, Case Studies**	**Quiz, Case Study Reports**	**CLO4**
13	Transaction management and Concurrency control Transaction management: ACID properties, serializability and concurrency control, Lock based concurrency control (2PL, Deadlocks),Time stamping methods, optimistic methods, database recovery management.	Lecture, Problem Solving	Quiz, Problem Sets	CLO4
14	NoSQL Databases Discuss NoSQL databases and their applications	Lecture, Problem Solving	Quiz, Problem Sets	CLO5
15	Database Recovery techniques Discuss on database recovery techniques	Lecture, Guest Lectures	Quiz, Presentation	CLO5
16	Database Recovery techniques Discuss on database recovery techniques(advanced)			
17	Future Trends in Databases Study emerging trends and future directions in database technology	Lecture, Discussions	Quiz, Assignment	CLO5

Week 1

What is a Database?

Definition: A database is a structured collection of data that can be accessed, managed, and updated. It serves as a repository for organized data that supports efficient retrieval and management.

Examples:

- A customer directory in an e-commerce application.
- A hospital's patient records system.
- An educational institution's student database.

What is a Database?

Characteristics of Databases:

1. Organized and structured.
2. Persistent and stored electronically.
3. Accessible by multiple users or applications.

What is a Database Management System (DBMS)?

- **Definition:** A Database Management System (DBMS) is software designed to define, manipulate, retrieve, and manage data in a database.

- **Core Functions of DBMS:**
  - **Data Storage:** Efficiently stores large amounts of data.
  - **Data Retrieval:** Allows quick access to data using queries.
  - **Data Manipulation:** Supports operations like insertion, deletion, and updating of records.
  - **Data Security:** Ensures authorized access and protects data integrity.

Popular DBMS Examples:

- Relational DBMS: MySQL, PostgreSQL, Oracle Database.
- NoSQL DBMS: MongoDB, Cassandra, DynamoDB.
- Cloud DBMS: Google Cloud Spanner, Amazon RDS.

Key Differences Between Data and Databases

Aspect	Data	Database
Definition	Raw, unprocessed facts and figures.	Organized collection of structured data.
Example	“John, 25”	A table with columns: Name, Age.
Purpose	Basic unit of information.	Stores and manages related data.

Components of a DBMS

- **Database Engine:** Core software component that performs storage and retrieval tasks.
- **Query Processor:** Interprets user queries (e.g., SQL commands) and converts them into actionable tasks.
- **Data Definition Language (DDL):** Defines database schema and structure (e.g., CREATE, ALTER).
- **Data Manipulation Language (DML):** Performs data operations like insertion, deletion, and updates (e.g., INSERT, UPDATE, DELETE).
- **Transaction Manager:** Ensures data integrity and consistency during concurrent user access.

Types of DBMS

- **Relational DBMS (RDBMS):**
  - Data is organized in tables with rows and columns.
  - Follows a strict schema and supports SQL.
  - Example: MySQL, PostgreSQL.
- **NoSQL DBMS:**
  - Designed for unstructured or semi-structured data.
  - Uses document, key-value, or graph data models.
  - Example: MongoDB, Redis.

Types of DBMS(Cont.)

- **Hierarchical DBMS:**
  - Data is organized in a tree-like structure.
  - Example: IBM's IMS.
- **Network DBMS:**
  - Data is organized in a graph, with multiple parent-child relationships.
  - Example: Integrated Data Store (IDS).
- **Object-Oriented DBMS (OODBMS):**
  - Supports object-oriented data models.
  - Example: ObjectDB.

ADVANTAGES OF USING DBMS

- **Data Centralization:**
  - Provides a single location for data storage, reducing redundancy.
- **Data Integrity and Consistency:**
  - Enforces rules to ensure data remains accurate and consistent.
- **Data Security:**
  - Implements user authentication and access controls.

ADVANTAGES OF USING DBMS

- **Backup and Recovery:**
  - Ensures data is protected and can be restored in case of failure.
- **Concurrent Access:**
  - Supports multiple users accessing the database simultaneously without conflicts.

Real-World Applications of Databases

- **Banking:**
  - Customer accounts, transactions, and loan processing.
- **Healthcare:**
  - Patient records, prescriptions, and billing.
- **Retail:**
  - Inventory management, customer orders, and sales analytics.
- **Education:**
  - Student management, course registrations, and grading systems.
- **Social Media:**
  - User profiles, posts, and interactions.

Evolution of Database Systems

- **Early Systems:**
  - Flat-file systems and manual data storage techniques.
- **Hierarchical and Network Models:**
  - Emerged in the 1960s to handle more complex data relationships.
- **Relational Databases:**
  - Introduced by E.F. Codd in the 1970s; became the standard for structured data.
- **NoSQL and Cloud Databases:**
  - Developed in the 21st century to handle large-scale, unstructured data and distributed systems.

Week 2

Introduction to Data Models

- **Definition:** A data model defines the structure, organization, and constraints of data stored in a database. It provides a framework for how data is stored, accessed, and manipulated.
- **Purpose of Data Models:**
  - To represent real-world entities and their relationships.
  - To ensure data consistency and integrity.
  - To facilitate the design and implementation of databases.

Components of a Data Model:

- **Structure:** Defines how data is organized (e.g., tables, hierarchies).
- **Operations:** Specifies the methods to manipulate data (e.g., insert, update).
- **Constraints:** Enforces rules to maintain data integrity.

TYPES OF DATA MODELS

Hierarchical Data Model

- **Definition:** Data is organized in a tree-like structure with parent-child relationships.
- **Characteristics:**
  - Each child node has only one parent node.
  - Relationships are represented as a hierarchy.
- **Advantages:**
  - Simple to understand and implement.
  - Fast data access for parent-child relationships.

Types of Data Models(cont.)

- **Disadvantages:**
  - Rigid structure; difficult to handle many-to-many relationships.
  - Modifications require restructuring the hierarchy.
- **Example:**
  - An organizational structure where each employee reports to one manager.

Types of Data Models(cont.)

- **Network Data Model**
- **Definition:** Data is organized in a graph structure, allowing multiple parent-child relationships.
- **Characteristics:**
  - Data elements are connected by links, representing relationships.
  - More flexible than the hierarchical model.
- **Advantages:**
  - Supports many-to-many relationships.
  - Efficient data retrieval for complex relationships.

Types of Data Models(cont.)

- **Disadvantages:**
  - Complex to design and maintain.
  - Requires specialized knowledge to implement.
- **Example:**
  - A flight reservation system where a flight can connect to multiple airports.

Types of Data Models(cont.)

- **Relational Data Model**
- **Definition:** Data is organized in tables (relations) with rows and columns.
- **Characteristics:**
  - Each table represents an entity, and each row represents an instance of the entity.
  - Relationships between tables are established using keys (primary and foreign).
- **Advantages:**
  - Simple and widely used.
  - Supports SQL for data manipulation.
  - Ensures data consistency and integrity.

Types of Data Models(cont.)

- **Disadvantages:**
  - Performance issues with very large datasets.
  - Limited for complex relationships like hierarchical or network structures.
- **Example:**
  - A student database with tables for Students, Courses, and Enrollments.

TYPES OF DATA MODELS(CONT.)

- **Object-Oriented Data Model**
- **Definition:** Combines object-oriented programming concepts with database principles.
- **Characteristics:**
  - Data is represented as objects with attributes and methods.
  - Supports inheritance, encapsulation, and polymorphism.
- **Advantages:**
  - Ideal for applications with complex data relationships.
  - Supports multimedia data (images, videos).

Types of Data Models(cont.)

- **Disadvantages:**
  - Complex to implement and maintain.
  - Less popular compared to the relational model.
- **Example:**
  - A multimedia library where books, videos, and audio files are stored as objects.

Types of Data Models(cont.)

- **Entity-Relationship (ER) Model**
- **Definition:** Represents data as entities, attributes, and relationships between entities.
- **Characteristics:**
  - **Entities:** Objects or things in the real world (e.g., Student, Course).
  - **Attributes:** Properties of entities (e.g., Name, Age).
  - **Relationships:** Associations between entities (e.g., Enrolled).
- **Advantages:**
  - Easy to visualize data relationships using ER diagrams.
  - Useful for designing relational databases.

Types of Data Models(cont.)

- **Disadvantages:**
  - Limited to high-level design; needs to be converted into a physical model.
- **Example:**
  - An ER diagram showing Students enrolled in Courses taught by Professors.

Types of Data Models(cont.)

- **NoSQL Data Models**
- **Definition:** Designed for unstructured or semi-structured data, offering high scalability and flexibility.
- **Types of NoSQL Models:**
  - **Document Model:** Stores data in JSON or BSON format (e.g., MongoDB).
  - **Key-Value Model:** Data stored as key-value pairs (e.g., Redis).
  - **Column-Family Model:** Optimized for read/write operations in columns (e.g., Cassandra).
  - **Graph Model:** Focuses on relationships using nodes and edges (e.g., Neo4j).
- **Advantages:**
  - Highly scalable and suitable for big data.
  - Flexible schema design.

Types of Data Models(cont.)

- **Disadvantages:**
  - Lacks standardization.
  - Limited support for complex queries.
- **Example:**
  - Social media platforms storing user interactions in a graph database.

Comparison of Data Models

Model	Structure	Use Case	Example
Hierarchical	Tree	Organizational chart	IBM IMS
Network	Graph	Flight reservation system	IDS
Relational	Tables	Enterprise applications	MySQL, Oracle
Object-Oriented	Objects	Multimedia or CAD applications	ObjectDB
ER Model	Entities and Relations	Database design	ER Diagrams
NoSQL	Document, Key-Value	Big data and real-time analytics	MongoDB, Cassandra

Week 3

Introduction to Database Design

- **Definition:** Database design is the process of defining the structure, storage, and organization of data in a database to ensure efficiency, scalability, and data integrity.
- **Objectives of Database Design:**
  - To minimize data redundancy.
  - To ensure data consistency and integrity.
  - To enhance data retrieval and manipulation efficiency.

Introduction to Database Design(cont.)

- **Phases of Database Design:**
- **Conceptual Design**
- **Logical Design**
- **Physical Design**

Phases of Database Design

- **Conceptual Design**
- **Definition:** Focuses on high-level representation of the data structure using Entity-Relationship (ER) diagrams.
- **Key Components:**
  - **Entities:** Objects or things in the real world (e.g., Customer, Product).
  - **Attributes:** Properties of entities (e.g., Customer Name, Product Price).
  - **Relationships:** Associations between entities (e.g., Customer purchases Product).

Phases of Database Design

- **Tools Used:**
  - ER Diagrams
  - UML Diagrams
- **Example:** An ER diagram representing Customers and Orders where a Customer can place multiple Orders.

Phases of Database Design

- **Logical Design**
- **Definition:** Converts the conceptual design into a logical model that can be implemented in a database management system (DBMS).
- **Steps Involved:**
  - Transform entities into tables.
  - Define primary keys for each table.
  - Define relationships using foreign keys.
  - Normalize the database.

Phases of Database Design

- **Normalization:**
- Process of organizing data to minimize redundancy and improve integrity.
- **Normal Forms:**
  - **First Normal Form (1NF):** Eliminate repeating groups; ensure atomic values.
  - **Second Normal Form (2NF):** Eliminate partial dependencies; all non-key attributes depend on the whole primary key.
  - **Third Normal Form (3NF):** Eliminate transitive dependencies; non-key attributes depend only on the primary key.

Phases of Database Design

- **Physical Design**
- **Definition:** Focuses on the physical storage and performance optimization of the database.
- **Key Considerations:**
  - **Indexing:** Create indexes on frequently queried columns for faster data retrieval.
  - **Partitioning:** Divide large tables into smaller, more manageable pieces.
  - **Storage Optimization:** Ensure efficient use of disk space.
- **Example:**
  - Adding an index on the "CustomerID" column to speed up searches.

Steps in Database Design

- **Requirement Analysis**
  - Gather and analyze the requirements of the database from stakeholders.
  - Understand the data needs, use cases, and constraints.
- **Conceptual Modeling**
  - Use ER diagrams to visualize entities, attributes, and relationships.
- **Logical Modeling**
  - Convert the conceptual model into a relational schema.
  - Normalize the schema to eliminate redundancy.

Steps in Database Design

- **Physical Modeling**
  - Define how the data will be stored on the disk.
  - Optimize for performance and storage efficiency.
- **Implementation**
  - Use a DBMS to create the database schema and populate it with data.
- **Testing and Validation**
  - Verify that the database meets requirements and performs efficiently.

Key Principles of Good Database Design

- **Simplicity:** Keep the design as simple as possible while meeting requirements.
- **Scalability:** Ensure the database can handle growing amounts of data.
- **Integrity:** Enforce data integrity through constraints and normalization.
- **Flexibility:** Design for adaptability to future changes.
- **Performance:** Optimize for query speed and storage efficiency.

Challenges in Database Design

- Balancing normalization and performance.
- Handling complex relationships and large datasets.
- Addressing security and access control requirements.
- Accommodating changes in requirements.

Tools for Database Design

- **ERD Tools:**
  - Lucidchart, draw.io, Microsoft Visio
- **DBMS Tools:**
  - MySQL Workbench, Microsoft SQL Server Management Studio (SSMS), Oracle SQL Developer
- **Other Tools:**
  - dbdiagram.io, DbSchema, Navicat

Week 4

Relational Algebra and Relational Calculus

Relational Algebra and Relational Calculus are two core theoretical concepts in the field of database management systems (DBMS). These languages provide the foundation for querying and manipulating relational databases. While relational algebra is a procedural language, relational calculus is a non-procedural (or declarative) language. Understanding these concepts is crucial for anyone studying databases and query languages like SQL

Relational Algebra

- Relational Algebra is a procedural query language, meaning it describes a sequence of operations that need to be performed on the data. The result of each operation is a relation (a table), and these relations can be further manipulated by other operations.

Basic Operations in Relational Algebra

1. **Select (σ)**

- **Description:** The Select operation is used to filter rows based on a specified condition. It is equivalent to the WHERE clause in SQL.
- **Syntax:** σ (Condition)(Relation)
- **Example:** σ (Age > 30)(Employee)
  - This operation will return all rows from the Employee relation where the Age is greater than 30.

Basic Operations in Relational Algebra

2. Project (π)

- **Description:** The Project operation is used to select specific columns from a relation. It is similar to the SELECT clause in SQL.
- **Syntax:** π (Column1, Column2, ...)(Relation)
- **Example:** π Name, Age (Employee)
  - This operation will return only the Name and Age columns from the Employee relation.

Basic Operations in Relational Algebra

3. **Union** (U)

- **Description:** The Union operation combines the results of two relations and returns all distinct rows. It is similar to the UNION operator in SQL.
- **Syntax:** Relation1 U Relation2
- **Example:** Employee U Manager
  - This operation will return a new relation that includes all employees and managers, eliminating duplicates.

Basic Operations in Relational Algebra

4. Set Difference (−)

- **Description:** The Set Difference operation returns the rows that are in one relation but not in another. It is similar to the EXCEPT operator in SQL.
- **Syntax:** Relation1 − Relation2
- **Example:** Employee − Manager
  - This operation will return all employees who are not managers.

Basic Operations in Relational Algebra

5. Cartesian Product (×)

- **Description:** The Cartesian Product operation combines each row of one relation with each row of another relation. It produces a relation with every possible combination of rows.
- **Syntax:** Relation1 × Relation2
- **Example:** Employee × Department
  - This operation will combine each employee with each department, resulting in a relation where every employee is paired with every department.

Basic Operations in Relational Algebra

6. Rename (ρ)

- **Description:** The Rename operation changes the name of a relation or its attributes.
- **Syntax:** ρ (NewRelationName)(Relation)
- **Example:** ρ (Employee1)(Employee)
  - This operation renames the relation Employee to Employee1.

Basic Operations in Relational Algebra

7. Join (⋈)

- **Description:** The Join operation combines rows from two relations based on a common attribute. It is often used to combine data from two related tables.
- **Syntax:** Relation1 ⋈ Relation2
- **Example:** Employee ⋈ Department
  - This operation will join the Employee and Department relations based on the common attribute (for instance, Dept_ID).

Basic Operations in Relational Algebra (Extended Operations)

- In addition to the basic operations, relational algebra also includes more advanced operations:
- **Natural Join (⋈)**
  - **Description:** A Natural Join is a specific type of join where only columns with the same name in both relations are used as the basis for joining.
  - **Example:** Employee ⋈ Department
    - This will automatically join the relations based on common attributes like Dept_ID.

Week 5

Basic Operations in Relational Algebra (Extended Operations)

- **Division (÷)**
- **Description:** The Division operation is used when we need to find tuples in one relation that are related to all tuples in another relation.
- **Example:**
  - Suppose we have two relations: Employee(Name, Skill) and SkillRequired(Skill).
  - The division operation helps to find employees who possess all skills listed in the SkillRequired relation.

Relational Calculus

- Relational Calculus is a declarative query language used to describe what data to retrieve rather than how to retrieve it. It defines queries using predicates and logical formulas.
- There are two main types of relational calculus:

Tuple Relational Calculus (TRC)

- In Tuple Relational Calculus, we specify the desired tuples (or rows) of a relation using variables and conditions that must be satisfied. It uses tuple variables and predicates.
- **Syntax:** $\{t \mid P(t)\}$
  - t is a tuple variable (a row in the relation), and P(t) is the predicate that the tuple must satisfy.
- **Example:** $\{t.Name \mid \exists e\ (Employee(e) \land e.Age > 30 \land e.Name = t.Name)\}$
  - This query finds the names of employees whose age is greater than 30.

Domain Relational Calculus (DRC)

- In Domain Relational Calculus, we deal with domain variables rather than tuples. Each domain variable represents an individual attribute value.
- **Syntax:** {<x1, x2, ..., xn> | P(x1, x2, ..., xn)}
  - x1, x2, ..., xn are domain variables, and P(x1, x2, ..., xn) is the predicate.
- **Example:** {<Name> | ∃Age (Employee(Name, Age) ∧ Age > 30)}
  - This query retrieves the names of employees whose age is greater than 30.

Key Differences between TRC and DRC

- **TRC** uses tuples as variables, while **DRC** uses domain variables (attribute values).
- **TRC** is more expressive and intuitive in some cases because it works directly with tuples.
- **DRC** is often simpler when dealing with specific attributes, as it treats each attribute separately.

Relational Algebra vs. Relational Calculus

- **Procedural vs. Declarative:** Relational Algebra tells us how to get the result, whereas Relational Calculus tells us what the result should be, leaving the how to the DBMS.
- **Execution:** While relational algebra is more procedural and closely related to how queries are processed in a DBMS, relational calculus is more about expressing the logic of what is needed from the database.
- **Application:** SQL is closely based on relational algebra, but many features of SQL (such as subqueries) are more closely related to relational calculus.

Practical Applications and SQL

- Both relational algebra and relational calculus serve as the foundation for SQL. SQL queries are generally equivalent to relational algebra expressions. However, SQL also incorporates additional features that go beyond pure relational algebra or relational calculus, such as aggregation (e.g., COUNT, SUM), ordering, and nested queries.

Practical Applications and SQL

- For example:
- A query like SELECT Name, Age FROM Employee WHERE Age > 30 is based on the relational algebra expression π Name, Age (σ Age > 30 (Employee)).
- A subquery like SELECT Name FROM Employee WHERE Age > (SELECT AVG(Age) FROM Employee) is based on relational calculus.

Week 6

Basic SQL (Structured Query Language)

- SQL (Structured Query Language) is the standard language used for managing and manipulating relational databases. It provides a powerful and easy-to-understand syntax for querying, inserting, updating, and deleting data. SQL is the foundation for working with relational database management systems (RDBMS) like MySQL, PostgreSQL, SQL Server, and Oracle.

SQL Basics

- SQL operates on a set of relational tables (or relations) and uses **queries** to perform various operations like data retrieval, updates, and table management. SQL is divided into different categories based on the operations being performed.

Why SQL?

- **Data Management**: Efficiently organize and manage large datasets.
- **Querying Data**: Retrieve specific data from massive databases.
- **Data Analysis**: Perform aggregations, filtering, and data transformations.
- **Interoperability**: Works with all relational database systems (MySQL, PostgreSQL, SQLite, etc.).
- **Portability**: SQL can run across different platforms with minimal changes.
- **Standardization**: SQL is governed by standards, ensuring consistency.

Categories of SQL Commands

- **Data Definition Language (DDL)**: Used to define and modify database structures.
  - Examples: CREATE, ALTER, DROP
- **Data Manipulation Language (DML)**: Used to manipulate data within tables.
  - Examples: INSERT, UPDATE, DELETE
- **Data Query Language (DQL)**: Used to query and retrieve data.
  - Example: SELECT
- **Data Control Language (DCL)**: Used to manage access and permissions.
  - Examples: GRANT, REVOKE
- **Transaction Control Language (TCL)**: Used to manage database transactions.
  - Examples: COMMIT, ROLLBACK, SAVEPOINT

Basic SQL Commands

- Data Query Language (DQL)
- The most common and essential SQL command for querying data is SELECT.
- **SELECT**: Retrieves data from one or more tables in the database.
- **Syntax:**
- SELECT column1, column2, ... FROM table_name WHERE condition;

Basic SQL Commands

- **Example:**

- SELECT Name, Age FROM Employee WHERE Age > 30;

This query retrieves the Name and Age columns from the Employee table for all employees who are older than 30.

Basic SQL Commands

- **Selecting All Columns:**

- SELECT * FROM Employee;
- This query selects all columns (*) from the Employee table.

Basic SQL Commands

- **DISTINCT**: Removes duplicate records from the result.
- **Example:**

- SELECT DISTINCT Age FROM Employee;

- This query returns all distinct ages from the Employee table, eliminating duplicates.

Basic SQL Commands

- **ORDER BY**: Sorts the result set in ascending or descending order.
- **Syntax:**
- SELECT column1, column2, ... FROM table_name ORDER BY column_name ASC|DESC;

Basic SQL Commands

- **ORDER BY**: Sorts the result set in ascending or descending order.

- **Example:**
- SELECT Name, Age FROM Employee ORDER BY Age DESC;

- This query sorts employees by age in descending order.

Basic SQL Commands

- **LIMIT / OFFSET:** Limits the number of rows returned by a query (useful for pagination).

- **Syntax:**
- SELECT column1, column2 FROM table_name LIMIT number_of_rows;

Basic SQL Commands

- **LIMIT / OFFSET:** Limits the number of rows returned by a query (useful for pagination).

Example:

SELECT Name FROM Employee LIMIT 5;

This query retrieves the first 5 employee names.

Basic SQL Commands

- Data Definition Language (DDL).
- DDL is used to define and modify database structures.
- **CREATE TABLE:** Creates a new table in the database.
- **Syntax:**

- CREATE TABLE table_name (
- column1 datatype,
- column2 datatype,
- ...
-);

Basic SQL Commands

- Data Definition Language (DDL).
- **Example:**
- CREATE TABLE Employee (
- EmployeeID INT PRIMARY KEY,
- Name VARCHAR(100),
- Age INT,
- Department VARCHAR(50)
-);
- This query creates a new table named Employee with columns for EmployeeID, Name, Age, and Department.

Basic SQL Commands

- **ALTER TABLE:** Modifies an existing table (e.g., add, drop, or modify columns).
- **Syntax:**
- ALTER TABLE table_name ADD column_name datatype;
- ALTER TABLE table_name DROP COLUMN column_name;

Basic SQL Commands

- **Example:**

- ALTER TABLE Employee ADD Salary DECIMAL(10, 2);
- This query adds a Salary column to the Employee table.

Basic SQL Commands

- **DROP TABLE:** Deletes a table from the database.
- **Syntax:**

- DROP TABLE table_name;

Week 7

Basic SQL Commands

- **Example:**

- DROP TABLE Employee;

- This query deletes the Employee table from the database.

Basic SQL Commands

- Data Manipulation Language (DML)
- DML commands allow you to manipulate the data in the database.
- **INSERT INTO**: Adds new rows of data into a table.
  - **Syntax:**
- INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

Basic SQL Commands

- **Example:**

- INSERT INTO Employee (EmployeeID, Name, Age, Department) VALUES (1, 'John Doe', 28, 'HR');
- This query inserts a new employee record into the Employee table.

Basic SQL Commands

- **UPDATE:** Modifies existing rows of data in a table.
- **Syntax:**

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

Basic SQL Commands

- **Example:**

- UPDATE Employee SET Age = 29 WHERE EmployeeID = 1;
- This query updates the age of the employee with EmployeeID 1.

Basic SQL Commands

- **DELETE:** Removes rows of data from a table.
- **Syntax:**

- DELETE FROM table_name WHERE condition;

Basic SQL Commands

- **Example:**

DELETE FROM Employee WHERE Age < 30;

This query deletes all employees under the age of 30 from the Employee table.

Basic SQL Commands

- Data Control Language (DCL)

- DCL is used to control access to data in the database.
- **GRANT**: Gives users or roles permission to perform actions on database objects (e.g., tables, views).
  - **Syntax:**

GRANT privilege ON object TO user;

Basic SQL Commands

- **Example:** GRANT SELECT ON Employee TO user1;
- **REVOKE:** Removes previously granted permissions from a user or role.
- **Syntax:**
- REVOKE privilege ON object FROM user;

Basic SQL Commands

- **Example:** REVOKE SELECT ON Employee FROM user1;

Week 8

Joins in SQL

- **Example:** REVOKE SELECT ON Employee FROM user1;
- SQL joins allow you to combine rows from two or more tables based on a related column between them. Joins are essential for querying multiple related tables.
- **INNER JOIN:** Returns only the rows where there is a match in both tables.
- **Syntax:**

```
SELECT columns FROM table1 INNER JOIN table2 ON table1.column = table2.column;
```

Joins in SQL

- **Example:**

```
SELECT Employee.Name, Department.Name
FROM Employee
INNER JOIN Department
ON Employee.Department =
Department.DepartmentID;
```

Joins in SQL

- **LEFT JOIN (OUTER JOIN):** Returns all rows from the left table and matching rows from the right table. If there is no match, NULL values are returned for columns from the right table.
- **Syntax:**

```
SELECT columns FROM table1 LEFT JOIN table2 ON
table1.column = table2.column;
```

Joins in SQL

- **Example:**

```
SELECT Employee.Name, Department.Name
FROM Employee
LEFT JOIN Department
ON Employee.Department =
Department.DepartmentID;
```

Joins in SQL

- **RIGHT JOIN (OUTER JOIN):** Returns all rows from the right table and matching rows from the left table. If there is no match, NULL values are returned for columns from the left table.
- Syntax:

```
SELECT columns FROM table1 RIGHT JOIN table2 ON
table1.column = table2.column;
```

Joins in SQL

- **FULL OUTER JOIN:** Returns all rows when there is a match in one of the tables.

- Syntax:

```
SELECT columns FROM table1 FULL OUTER JOIN table2
ON table1.column = table2.column;
```

SQL Functions

- SQL provides several built-in functions to perform operations on data.
- **Aggregate Functions:**
  - COUNT(): Counts the number of rows.
  - SUM(): Calculates the sum of a numeric column.
  - AVG(): Calculates the average of a numeric column.
  - MAX(): Returns the maximum value.
  - MIN(): Returns the minimum value.

SQL Functions

- **Example:**

- SELECT AVG(Age) FROM Employee;
- **String Functions:**
- CONCAT(): Concatenates two or more strings.
- UPPER(): Converts a string to uppercase.
- LOWER(): Converts a string to lowercase.

SQL Functions

- **Example:**

- SELECT CONCAT(FirstName, ' ', LastName) AS FullName FROM Employee;

Subqueries

- Subqueries are queries nested inside another SQL query. They are used to perform operations that depend on results from another query.

Subqueries

- **Types of Subqueries**
- **Scalar Subquery:** Returns a single value.
- **Example:**

```
SELECT Name, Age
FROM Employee
WHERE Age > (SELECT AVG(Age) FROM Employee);
```

Retrieves employees older than the average age.

Subqueries

- **Correlated Subquery:** Depends on the outer query for its execution.
- **Example:**

```
SELECT e.Name
FROM Employee e
WHERE e.Salary > (SELECT AVG(Salary) FROM Employee
WHERE DepartmentID = e.DepartmentID);
```

Retrieves employees earning above the average salary in their department.

Subqueries

- **EXISTS Subquery**: Checks for the existence of rows.
- **Example:**

```
SELECT Name
FROM Employee e
WHERE EXISTS (SELECT 1 FROM Department d WHERE
e.DepartmentID = d.DepartmentID);
```

Window Functions

Window functions perform calculations across a set of rows related to the current row. They do not reduce the number of rows returned.

- **Common Window Functions**
- **ROW_NUMBER()**: Assigns a unique number to each row.

```
SELECT Name, Salary, ROW_NUMBER() OVER (ORDER BY Salary DESC) AS Rank
FROM Employee;
```

Window Functions

- **RANK()**: Assigns a rank to each row with tied values having the same rank.
- **Example:**

```
SELECT Name, Salary, RANK() OVER (ORDER BY Salary DESC)
AS Rank
FROM Employee;
```

Window Functions

- **PARTITION BY**: Divides the result set into partitions to apply the function to each partition.
- **Example:**

```
SELECT Name, DepartmentID, Salary, AVG(Salary) OVER
(PARTITION BY DepartmentID) AS AvgSalary
FROM Employee;
```

Transactions

- Transactions ensure that a sequence of SQL operations is executed as a single unit. Transactions help maintain data integrity.
- **START TRANSACTION**: Begins a new transaction.
- **COMMIT**: Saves all changes made in the transaction.
- **ROLLBACK**: Reverts changes made during the transaction.

Transactions

- Example:

```
START TRANSACTION;

UPDATE Employee SET Salary = Salary * 1.1 WHERE DepartmentID = 1;

IF (SELECT SUM(Salary) FROM Employee WHERE DepartmentID = 1) > 100000 THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

Common Table Expressions (CTEs)

- CTEs provide a way to simplify complex queries by creating temporary result sets.
- Syntax:

```
WITH CTEName AS (
    SELECT column1, column2
    FROM table
    WHERE condition
)
SELECT * FROM CTEName WHERE another_condition;
```

Common Table Expressions (CTEs)

- Example:

```
WITH DepartmentSalary AS (
    SELECT DepartmentID, AVG(Salary) AS AvgSalary
    FROM Employee
    GROUP BY DepartmentID
)
SELECT e.Name, e.Salary, d.AvgSalary
FROM Employee e
JOIN DepartmentSalary d ON e.DepartmentID = d.DepartmentID
WHERE e.Salary > d.AvgSalary;
```

Advanced SQL Use Cases

- **Pivot Tables:**
- Transform rows into columns.
- Example:

```
SELECT DepartmentID,
        SUM(CASE WHEN Gender = 'Male' THEN 1 ELSE 0 END)
AS MaleCount,
        SUM(CASE WHEN Gender = 'Female' THEN 1 ELSE 0 END)
AS FemaleCount
FROM Employee
GROUP BY DepartmentID;
```

Advanced SQL Use Cases

- **Recursive Queries:**
- Example:

```
WITH RECURSIVE EmployeeHierarchy AS (
    SELECT EmployeeID, ManagerID, Name
    FROM Employee
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.EmployeeID, e.ManagerID, e.Name
    FROM Employee e
    JOIN EmployeeHierarchy eh ON e.ManagerID = eh.EmployeeID
)
SELECT * FROM EmployeeHierarchy;
```

Week 9

Constraints in DBMS

- Constraints are predefined rules enforced on data in database tables to maintain the integrity, validity, and consistency of the data. Constraints restrict the type of data that can be entered into a table.

Constraints in DBMS

Types of Constraints:

- **NOT NULL Constraint**

  Ensures that a column cannot have NULL values, i.e., every row must have a value for this column.

  Used when a field is mandatory.

- **Syntax**

```
CREATE TABLE Employee (
    EmployeeID INT,
    Name VARCHAR(50) NOT NULL
);
```

Constraints in DBMS

- **Example:**A Name column in an Employee table must always have a value.

Constraints in DBMS

- UNIQUE Constraint
  - Ensures all values in a column or a combination of columns are unique.
  - Allows NULL values (but only one per column).
- **Syntax:**

```
CREATE TABLE Employee (
    EmployeeID INT UNIQUE,
    Name VARCHAR(50)
);
```

- **Example:**Employee ID must be unique to identify each employee.

Constraints in DBMS

- PRIMARY KEY Constraint:
  - Combines NOT NULL and UNIQUE.
  - Uniquely identifies each record in the table.
  - A table can only have one PRIMARY KEY.
- **Syntax:**

```
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50)
);
```

Constraints in DBMS

- PRIMARY KEY Constraint:
  - Composite Primary Key
  - A primary key can consist of multiple columns.

Example:

```
CREATE TABLE Enrollment (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID)
);
```

Constraints in DBMS

- FOREIGN KEY Constraint:
- Enforces referential integrity by linking columns in two tables.
- Ensures that the value in a column matches a value in another table.

Constraints in DBMS

- FOREIGN KEY Constraint:
- Syntax:

```
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);
```

Constraints in DBMS

- CHECK Constraint:
- Ensures that all values in a column meet a specific condition.
- **Syntax:**

```
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT CHECK (Age > 18)
);
```

- **Example:** Employees must be older than 18.

Constraints in DBMS

- DEFAULT Constraint:
- Provides a default value for a column if no value is provided during insertion.
- **Syntax:**

```
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    Status VARCHAR(20) DEFAULT 'Active'
);
```

- **Example:** New employees have a default status of "Active".

Constraints in DBMS

- **Constraints on Table Creation**
- Constraints can also be added to a table after its creation:
- **Syntax:**

```
ALTER TABLE Employee
ADD CONSTRAINT Unique_Name UNIQUE (Name);
```

Constraints in DBMS

- Benefits of Constraints:
- **Data Integrity**: Ensures only valid data is entered into the database.
- **Consistency**: Enforces uniformity across data entries.
- **Error Prevention**: Prevents invalid data modifications or deletions.
- **Reduces Code Complexity**: Handles data validation at the database level.

Week 10

Views in DBMS

- A **view** is a virtual table based on the result of a query. It does not store data itself but retrieves data from one or more base tables.

Views in DBMS

- **Characteristics of Views**
- **Logical Representation:** Views are used to represent subsets of data logically.
- **No Data Storage:** Views do not store data themselves.
- **Dynamic:** Always reflect the latest data in the underlying tables.
- **Read and Write Access:** Depending on the database, views may allow updates.

Views in DBMS

- Creating a View:
- Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

- Example:

Views in DBMS

Example

```
CREATE VIEW ActiveEmployees AS
SELECT EmployeeID, Name, Status
FROM Employee
WHERE Status = 'Active';
```

Views in DBMS

Querying a View:

Views can be queried like tables.

```
SELECT * FROM ActiveEmployees;
```

Updating Data Through Views

- Views allow data updates if:
  - They are created from a single table.
  - Do not involve aggregate functions or GROUP BY.
  - Do not involve complex joins.

Views in DBMS

Example:

```
UPDATE ActiveEmployees
SET Status = 'Inactive'
WHERE EmployeeID = 1;
```

Syntax:

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW ActiveEmployees;
```

Views in DBMS

Materialized Views:

- Unlike regular views, materialized views store query results physically on disk.
- They are useful for performance optimization when dealing with large datasets.

- Syntax:
  - CREATE MATERIALIZED VIEW materialized_view_name AS
  - SELECT column1, column2
  - FROM table_name;

Views in DBMS

Advantages of Views:

- **Data Security:**
  - Restrict access to specific rows or columns.
- Example

```
CREATE VIEW ManagerView AS
SELECT EmployeeID, Name
FROM Employee
WHERE Role = 'Manager';
```

Views in DBMS

- **Simplification:**
- Encapsulate complex queries.
- Example:

```
CREATE VIEW EmployeeStats AS
SELECT DepartmentID, COUNT(*) AS TotalEmployees
FROM Employee
GROUP BY DepartmentID;
```

- **Logical Independence:**
  - Abstract underlying table structure from users.
- **Data Consistency:**
  - Provide a consistent view of the database.

Views in DBMS

Constraints vs. Views

Aspect	Constraints	Views
Purpose	Ensure data validity and integrity during insert/update.	Simplify data retrieval and restrict access.
Data Storage	Exist as part of table metadata.	Virtual tables; may store data (materialized views).
Scope	Prevent invalid data from being entered into the table.	Provide customized data representations for queries.
Examples	NOT NULL, CHECK, FOREIGN KEY.	Filtered views, aggregations, or specific columns from multiple tables.

Views in DBMS

- **Simplification:**
- Encapsulate complex queries.
- Example:

```
CREATE VIEW EmployeeStats AS
SELECT DepartmentID, COUNT(*) AS TotalEmployees
FROM Employee
GROUP BY DepartmentID;
```

- **Logical Independence:**
  - Abstract underlying table structure from users.
- **Data Consistency:**
  - Provide a consistent view of the database.

Views in DBMS

- Practical Examples:
- Example: Constraints

```
CREATE TABLE Product (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(50) NOT NULL,
    Price DECIMAL(10, 2) CHECK (Price > 0),
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES
Category(CategoryID)
);
```

Views in DBMS

- Example: View

```
CREATE VIEW ExpensiveProducts AS
SELECT ProductID, ProductName, Price
FROM Product
WHERE Price > 1000;
```

Week 11

Indexing and Hashing in DBMS

- Indexing and hashing are techniques used in database management systems (DBMS) to optimize the performance of data retrieval. They help to reduce the time complexity of query execution by minimizing the number of disk I/O operations required to locate data.

Indexing and Hashing in DBMS

- Indexing in DBMS:
- **Indexing** is a data structure technique used to efficiently retrieve records from a database table. It creates an auxiliary structure that provides quick access to specific data rows based on a key.

Indexing and Hashing in DBMS

- What is an Index?

- An **index** is like a roadmap that allows the DBMS to locate rows in a table more efficiently, without scanning the entire table.
- It is typically implemented as a separate data structure (e.g., B-tree, hash table) that stores key-value pairs, where the key corresponds to the indexed column(s) and the value is the physical address of the data in the table.

Indexing and Hashing in DBMS

- Types of Indexes
- **Primary Index:**

  Built on a primary key, which uniquely identifies each row in the table.

  The table is sorted based on the primary key.
- **Example:** If a table has a StudentID as the primary key, the primary index will organize the data by StudentID.

Indexing and Hashing in DBMS

- **Secondary Index:**
- Built on non-primary attributes, which may not be unique.
- Allows quick access based on non-primary columns.
- **Example:** Creating an index on the Name column for faster retrieval of students by name.

Indexing and Hashing in DBMS

- **Clustered Index:**
- Organizes the table data to match the order of the index.
- Only one clustered index can exist per table because the data can only be physically sorted in one order.
- **Example:** A clustered index on EmployeeID ensures the data rows in the table are sorted by EmployeeID.

Indexing and Hashing in DBMS

- **Non-clustered Index:**
  - Does not affect the order of the data in the table.
  - Stores pointers to the actual data rows.
  - Multiple non-clustered indexes can exist on a single table.
  - **Example:** A non-clustered index on Name in an employee table.

Index Structures

- **Single-level Index:**
- A simple, one-level index structure where the index entries directly point to the table rows.
- **Types:**
  - Dense Index: Every table record has a corresponding index entry.
  - Sparse Index: Only some records have index entries (e.g., the first record of every block).

Index Structures

- **Multi-level Index:**
- An index on the index itself, reducing the size of the index structure at each level.
- Useful for large tables with a vast amount of data.

Index Structures

- **B-Tree Index:**
- A balanced tree structure where nodes contain keys and pointers to child nodes or data.
- Ensures logarithmic time complexity for search, insert, and delete operations.
- Widely used due to its ability to handle large datasets efficiently.

Index Structures

- **B+ Tree Index:**
  - A variation of B-tree where all data pointers are stored only at the leaf level.
  - Offers sequential access to data and is commonly used in DBMS.

Advantages of Indexing

- **Faster Data Retrieval:** Reduces the time complexity of queries by providing direct access to data.
- **Efficient Sorting:** Enables quick sorting of rows based on indexed columns.
- **Reduces Disk I/O:** Limits the number of disk blocks accessed during a query.

Disadvantages of Indexing

- **Storage Overhead:** Requires additional storage for maintaining index structures.
- **Maintenance Overhead:** Updates, inserts, and deletes require index modification, which can slow down operations.

Week 12

Hashing in DBMS

- **Hashing** is a technique used to map data to a fixed-size address space using a hash function. It is commonly used in situations where data retrieval is required in constant or near-constant time.

Hashing in DBMS

- **What is Hashing?**
- **Hash Function:** A mathematical function that converts input data into a fixed-size hash code or hash value.
- The hash value is used as an address to locate the data in a hash table.
- **Example:** If a StudentID is hashed to the value 5, the corresponding student record is stored at index 5 in the hash table.

Hashing in DBMS

- Types of Hashing:
- **Static Hashing:**
  - The hash table size is fixed, and the hash function maps data into a fixed set of buckets.
  - **Advantages:**
    - Simple to implement.
    - Works well for databases with a fixed amount of data.
  - **Disadvantages:**
    - Performance degrades as the table becomes full.
    - Poor handling of dynamic datasets.

Hashing in DBMS

- Types of Hashing:
- **Dynamic Hashing:**
  - The hash table grows or shrinks dynamically as the dataset changes.
  - **Techniques:**
    - **Extendible Hashing:** Uses a directory that grows dynamically as more data is added. Buckets can split when they overflow.
    - **Linear Hashing:** Buckets are split in a predefined sequence, eliminating the need for a directory.

Hashing in DBMS

- **Hashing Techniques**
  - **Division Method:**
    - The hash value is computed as h(k) = k % n, where k is the key and n is the number of buckets.
    - **Example:** For a key 101 and n = 10, the hash value is 101 % 10 = 1.

Hashing in DBMS

- **Hashing Techniques**
  - **Multiplication Method:**
  - The hash value is computed as h(k) = floor(n * (k * A % 1)), where A is a constant.
  - Reduces the likelihood of collisions compared to the division method.
  - **Universal Hashing:**
  - Uses a randomized hash function to minimize the chance of collisions.

Hashing in DBMS

- Collision Resolution in Hashing
  - When two keys hash to the same bucket, a **collision** occurs. Several methods are used to handle collisions:
  - **Chaining:**
    - Each bucket points to a linked list of entries that hash to the same bucket.
    - **Example**

      Bucket 0: Key1 -> Key2

Hashing in DBMS

- Collision Resolution in Hashing
  - **Open Addressing:**
  - All data is stored within the hash table itself, and collisions are resolved by probing (searching) for the next available slot.
  - **Techniques:**
  - **Linear Probing:** Search sequentially for the next empty slot.
  - **Quadratic Probing:** Search using a quadratic formula (e.g., 1^2, 2^2).
  - **Double Hashing:** Use a second hash function to determine the step size for probing.

Hashing in DBMS

- **Advantages of Hashing**
  - **Constant-Time Access:** Offers O(1) average time complexity for data retrieval.
  - **Dynamic Data Handling:** Suitable for dynamic datasets when using dynamic hashing techniques.
- **Disadvantages of Hashing**
  - **Collisions:** Handling collisions can add complexity and overhead.
  - **Storage Overhead:** Requires additional space for the hash table and collision resolution structures.

Indexing vs. Hashing

Aspect	Indexing	Hashing
Purpose	Optimizes range and point queries.	Optimizes point queries.
Structure	Tree-based (e.g., B-tree, B+ tree).	Hash-based (e.g., hash table).
Performance	Efficient for range queries.	Not suitable for range queries.
Collision Handling	Not applicable.	Requires collision resolution techniques.
Use Case	Querying large datasets with range conditions.	Exact match queries in large datasets.

Practical Applications

- **Indexing:**
  - Used for optimizing SQL queries.
  - Improves performance of range queries (e.g., SELECT * FROM Employee WHERE Age BETWEEN 25 AND 30).
  - Commonly implemented as B+ trees in RDBMS.
- **Hashing:**
  - Used for in-memory data structures like hash maps.
  - Optimizes equality-based queries (e.g., SELECT * FROM Employee WHERE EmployeeID = 123).

Week 13

Transaction Management and Concurrency Control

- Transaction management and concurrency control are crucial components of a Database Management System (DBMS) to ensure data consistency, integrity, and availability in multi-user environments.

Transaction Management and Concurrency Control

- Transactions in DBMS:
- **What is a Transaction?**
  - A **transaction** is a sequence of one or more database operations (such as insert, update, delete, or retrieve) executed as a single logical unit of work.
- A transaction must maintain the **ACID properties** to ensure data integrity.

Transaction Management and Concurrency Control

- **ACID Properties**
  - **Atomicity:**
  - Ensures that a transaction is executed completely or not at all.
  - If a transaction fails, all changes made during the transaction are rolled back.
  - Example: Transferring money from one bank account to another must either complete fully or not occur at all.
  - **Consistency:**
  - Ensures that the database transitions from one valid state to another after a transaction.
  - Example: The total balance in all accounts should remain unchanged after a transaction.

Transaction Management and Concurrency Control

- **ACID Properties**
  - **Isolation:**
  - Ensures that concurrently executing transactions do not interfere with each other.
  - Example: Two users withdrawing money from the same account simultaneously should not result in an inconsistent state.
  - **Durability:**
  - Ensures that once a transaction is committed, its changes are permanently recorded in the database, even in the case of system failure.
  - Example: Once money is transferred between accounts, the changes must persist even if the system crashes.

Transaction Management and Concurrency Control

- States of a Transaction:
  - **Active**: The transaction is being executed.
  - **Partially Committed**: All operations are completed, but the changes are not yet saved to the database.
  - **Committed**: The transaction has completed successfully, and the changes are saved.
  - **Failed**: An error occurred, and the transaction cannot proceed.
  - **Aborted**: The transaction is rolled back, undoing all changes made during its execution.

Transaction Management and Concurrency Control

- Concurrency in DBMS:
- Concurrency control is the process of managing simultaneous transaction execution in a multi-user environment while ensuring consistency and isolation.

Transaction Management and Concurrency Control

- **Problems in Concurrency**
  - **Lost Update Problem:**
  - Occurs when two transactions update the same data item, and one update overwrites the other.
  - Example:
    - Transaction T1 reads a value X = 100 and increments it by 10.
    - Transaction T2 reads the same X and decrements it by 5.
    - The final value of X should be 105, but it may incorrectly become 95 or 110.

Transaction Management and Concurrency Control

- **Problems in Concurrency**
  - **Dirty Read Problem:**
  - Occurs when a transaction reads uncommitted changes made by another transaction.
  - Example: T1 updates a value, and T2 reads the updated value before T1 commits. If T1 rolls back, T2 has used invalid data.
  - **Non-repeatable Read:**
  - Occurs when a transaction reads the same data item multiple times and gets different results.
  - Example: T1 reads a value, T2 modifies it, and T1 reads it again.

Transaction Management and Concurrency Control

- **Problems in Concurrency**
  - **Phantom Read:**
    - Occurs when a transaction retrieves a set of rows, and another transaction inserts or deletes rows, causing changes in subsequent reads.
    - Example: T1 reads a list of accounts, T2 adds a new account, and T1 re-reads the list, finding an extra account.

Transaction Management and Concurrency Control

- Concurrency Control Techniques:
  - **Lock-Based Protocols**
  - Locks prevent multiple transactions from accessing the same data item simultaneously.
  - **Types of Locks:**
    - **Shared Lock (S)**: Allows read access but prevents write access.
    - **Exclusive Lock (X)**: Allows write access and prevents both read and write access by other transactions.

Transaction Management and Concurrency Control

- Concurrency Control Techniques:
  - **Two-Phase Locking (2PL):**
  - Divides the execution of a transaction into two phases:
    - **Growing Phase:** Locks are acquired, but no locks are released.
    - **Shrinking Phase:** Locks are released, but no new locks are acquired.
  - Guarantees serializability but can lead to deadlocks.

  - **Deadlock Handling:**
  - **Detection and Recovery:** Detect deadlocks and abort one or more transactions.
  - **Prevention:** Ensure that transactions do not enter a deadlock state by acquiring all necessary locks in advance.

Transaction Management and Concurrency Control

- **Timestamp-Based Protocols:**
  - Transactions are assigned timestamps at the start.
  - Transactions are executed in the order of their timestamps.
  - **Rules:**
  - If a transaction tries to access a data item and its timestamp is older than the data's last modified timestamp, the transaction is aborted.
  - Ensures serializability without locks.

Transaction Management and Concurrency Control

- Multiversion Concurrency Control (MVCC):
  - Maintains multiple versions of data items to allow concurrent read and write operations.
  - Transactions read the version of the data item that was committed at the start of the transaction.
  - Used in modern databases like PostgreSQL and MySQL.

Transaction Management and Concurrency Control

- **Optimistic Concurrency Control:**
  - Assumes that conflicts are rare.
  - Transactions execute without restrictions, and conflicts are checked at the commit stage.
  - If a conflict is detected, one transaction is rolled back.

Transaction Management and Concurrency Control

- Transaction Recovery:
- Transaction recovery ensures that the database remains consistent in the case of system failures.
- **Types of Failures**
  - **Transaction Failure**: Caused by logical errors (e.g., division by zero) or system errors (e.g., deadlocks).
  - **System Failure**: Caused by hardware or software crashes.
  - **Media Failure**: Caused by physical damage to storage media.

Transaction Management and Concurrency Control

- **Recovery Techniques:**
  - **Undo Logging:**
  - Records old values of data items before they are modified.
  - Rollback can restore the original values in case of failure.
  - **Redo Logging:**
  - Records new values of data items after they are modified.
  - Replay can apply the changes after a crash.
  - **Undo-Redo Logging:**
  - Combines undo and redo logging to handle failures at any stage.
  - **Checkpointing:**
  - Periodically saves the current state of the database to reduce recovery time.

Transaction Management and Concurrency Control

- **Isolation Levels in SQL:**
  - DBMSs provide different isolation levels to balance concurrency and consistency.
  - **Read Uncommitted:**
    - Transactions can read uncommitted data.
    - May cause dirty reads.
  - **Read Committed:**
    - Transactions cannot read uncommitted data.
    - Prevents dirty reads but allows non-repeatable reads.

Transaction Management and Concurrency Control

- **Isolation Levels in SQL:**
  - **Repeatable Read:**
    - Ensures that data read by a transaction remains consistent during its execution.
    - Prevents dirty and non-repeatable reads but allows phantom reads.
  - **Serializable:**
    - Ensures complete isolation between transactions.
    - Prevents dirty reads, non-repeatable reads, and phantom reads.
    - Most restrictive isolation level.

Transaction Management and Concurrency Control

- **Practical Applications:**
  - **Banking Systems:** Ensure atomic money transfers and prevent concurrency issues.
  - **E-commerce:** Handle multiple users accessing the same inventory simultaneously.
  - **Reservation Systems:** Prevent overbooking by managing simultaneous bookings.

Week 14

NoSQL Databases

- NoSQL (Not Only SQL) databases are designed to handle large volumes of unstructured, semi-structured, or structured data. Unlike traditional relational databases, NoSQL databases offer flexible schemas, scalability, and high performance, making them ideal for modern applications.

What is NoSQL?

- A class of databases that provides mechanisms for storage and retrieval of data beyond the traditional table structures of relational databases.
- Built for distributed data stores, NoSQL databases can handle big data and real-time web applications.

Characteristics of NoSQL

- **Schema Flexibility**:No fixed schema; data can evolve dynamically.
- Suitable for handling unstructured and semi-structured data.
- **Scalability**:Horizontal scaling (adding more servers) instead of vertical scaling (upgrading server capacity).

Characteristics of NoSQL

- **High Availability:**
  - Ensures continuous availability even during system failures.
- **CAP Theorem:**
  - Consistency: Every read gets the most recent write or an error.
  - Availability: Every request receives a response without guarantee of the most recent data.
  - Partition Tolerance: The system continues to function despite network partitions.
  - NoSQL databases often trade off between these properties based on the use case.

Types of NoSQL Databases

- Key-Value Stores
- Document Stores
- Column-Family Stores
- Graph Databases

Types of NoSQL Databases

- Types of NoSQL Databases:
  - **Key-Value Stores**
  - **Structure:**
    - Data is stored as key-value pairs.
    - Example: { "userID": "12345" }
  - **Use Cases:**
    - Caching, session storage.
  - **Examples:**
    - Redis, DynamoDB.

Types of NoSQL Databases

- **Document Stores**
  - **Structure:**
    - Data is stored as documents, often in JSON, BSON, or XML format.
    - Example:

```
{
  "userID": "12345",
  "name": "John Doe",
  "orders": [
    {"orderID": "001", "amount": 100},
    {"orderID": "002", "amount": 200}
  ]
}
```

Types of NoSQL Databases

- **Document Stores**
  - **Use Cases:**Content management systems, catalogs, user profiles.
  - **Examples:**MongoDB, CouchDB.

Types of NoSQL Databases

- **Column-Family Stores**
  - **Structure:**
    - Data is stored in columns grouped into families.
    - Example:
      RowKey: 12345
      Name: John Doe
      OrderID: 001, 002
      Amount: 100, 200
  - **Use Cases:**Analytics, time-series data.
  - **Examples:**Cassandra, HBase.

Types of NoSQL Databases

- **Graph Databases**
  - **Structure:**
    - Data is represented as nodes, edges, and properties.
    - Example:

      (User) -- [LIKES] --> (Product)

      (User) -- [FRIEND] --> (User)
  - **Use Cases:**
    - Social networks, recommendation engines, fraud detection.
  - **Examples:**
    - Neo4j, ArangoDB.

Advantages of NoSQL

- **Scalability**:Easily scales horizontally to handle massive data.
- **Flexibility**:No rigid schema allows changes without downtime.
- **High Performance**:Optimized for high-speed read and write operations.

Advantages of NoSQL

- **Distributed Architecture:**
  - Built to operate across multiple servers or clusters.
- **Handles Big Data:**
  - Suitable for real-time data processing and large-scale datasets.

Disadvantages of NoSQL

- **Lack of Standardization**:No standardized query language like SQL.
- **Eventual Consistency**:Some NoSQL databases prioritize availability and partition tolerance over consistency.
- **Complexity in Relationships**:Managing relationships between entities is more complex than in relational databases.
- **Limited Maturity**:Newer than relational databases, with less extensive tools and community support.

NoSQL vs. SQL

- Applications of NoSQL:
- **Social Media:**
  - Storing user data, relationships, and interactions.
  - Example: Facebook using Cassandra for messaging.
- **E-commerce:**
  - Managing product catalogs and user profiles.
  - Example: Amazon DynamoDB for handling shopping cart data.
- **Real-Time Analytics:**
  - Processing massive datasets with low latency.
  - Example: Redis for caching.
- **IoT and Big Data:**
  - Storing and querying sensor data efficiently.
  - Example: MongoDB and Cassandra.
- **Content Management:**
  - Managing unstructured data like videos, images, and documents.
  - Example: Couchbase for media applications.

Querying NoSQL Databases

- NoSQL databases do not use SQL but offer their query mechanisms:

- **MongoDB Query Example (Document Store):**

  - db.users.find({ "name": "John Doe" })

Querying NoSQL Databases

- **Cassandra Query Example (Column-Family Store):**

- SELECT * FROM users WHERE userID = '12345';

Querying NoSQL Databases

- **Neo4j Query Example (Graph Database):**

```
MATCH (u:User)-[:LIKES]->(p:Product)
RETURN u, p;
```

Week 15

Database Recovery Techniques

- Database recovery ensures that a database remains consistent and operational after system failures, crashes, or errors. Recovery mechanisms restore the database to a correct state while preserving data integrity.

What is Database Recovery?

- Database recovery is the process of restoring a database to a previous consistent state after a failure.

- Failures can occur due to system crashes, power outages, hardware issues, software bugs, or user errors.

Goals of Database Recovery

- Maintain **ACID properties** (Atomicity, Consistency, Isolation, Durability).
- Ensure that committed transactions persist, and incomplete transactions are rolled back.
- Minimize downtime and data loss.

Types of Failures

- **Transaction Failure:**Logical errors (e.g., division by zero, invalid data).
- System errors (e.g., deadlocks, resource unavailability).
- **System Failure:**Hardware or software crashes affecting the database system.
- **Media Failure:**Physical damage to storage media (e.g., disk corruption).
- **Application or User Errors:**Mistakes like accidental deletion or incorrect data updates.

Types of Database Recovery

- **Immediate Update**
  - Changes made by a transaction are immediately written to the database.
  - Requires undo and redo logs to handle incomplete transactions.
- **Deferred Update**
  - Changes made by a transaction are written to a log and applied to the database only after the transaction is committed.
  - Eliminates the need for undo operations but requires redo operations during recovery.

Types of Database Recovery

- **In-Place Update**
  - Updates are directly applied to the original data.
  - Requires a combination of undo and redo logs for recovery.
- **Shadow Paging**
  - Instead of updating the original data, updates are made to a shadow copy.
  - Once the transaction commits, the shadow copy becomes the new database.

Database Recovery Techniques

- **Log-Based Recovery**
  - A log is a sequential record of all database operations.
  - Each log entry contains:
    - **Transaction ID**
    - **Data Item** (modified or accessed)
    - **Old Value** (before the update)
    - **New Value** (after the update)

Database Recovery Techniques

- **Undo Logging:**
  - Records old values of data items before they are updated.
  - If a transaction fails, the system uses the undo log to revert changes.

  - Example Log Entry:
  - <T1, X, Old_Value>

Database Recovery Techniques

- Recovery Steps:
  - Identify incomplete transactions.
  - Rollback changes using undo logs.
- **Redo Logging:**
  - Records new values of data items after updates.
  - Ensures that committed transactions are reapplied after a crash.
  - Example Log Entry:
  - <T1, X, New_Value>

Database Recovery Techniques

- Recovery Steps:
  - Identify committed transactions.
  - Reapply updates using redo logs.

- **Undo-Redo Logging:**
  - Combines both undo and redo logs to handle failures at any stage.

Database Recovery Techniques

- **Checkpointing**
  - A checkpoint is a snapshot of the database state at a particular point in time.
  - Helps reduce recovery time by limiting the log records that need to be processed.
- **Steps:**
  - Flush all logs and modified data to stable storage.
  - Write a checkpoint record to the log.
  - Example Log Entry:

    <CHECKPOINT>

Database Recovery Techniques

- Recovery Steps:
  - Start from the last checkpoint.
  - Process logs to undo uncommitted transactions and redo committed transactions.

Week 16

Database Recovery Techniques

- **Shadow Paging**
  - Uses two copies of the database:
    - **Current Page Table:** The current version of the database.
    - **Shadow Page Table:** A backup copy of the database.
  - **Process:**
    - Changes are made to a copy (shadow page table).
    - Once the transaction commits, the shadow table replaces the current table.

Database Recovery Techniques

- **Advantages:**
  - No logging required.
  - Ensures atomicity.
- **Disadvantages:**
  - High storage overhead.
  - Difficult to implement for large databases.

Database Recovery Techniques

- **ARIES (Algorithm for Recovery and Isolation Exploiting Semantics)**
  - ARIES is a popular recovery technique that combines logging, checkpoints, and a sophisticated recovery process.
  - **Key Features:**
    - Write-Ahead Logging (WAL): Log records must be written to stable storage before data changes.
    - Repeating History: During recovery, ARIES replays all operations from the logs.
    - Selective Undo: Only the operations of uncommitted transactions are undone.

Database Recovery Techniques

- **Phases:**
  - **Analysis:** Determine the state of all transactions at the time of failure.
  - **Redo:** Replay operations of committed transactions.
  - **Undo:** Rollback changes made by uncommitted transactions.

Recovery in Distributed Databases

- In distributed systems, transactions span multiple sites, requiring coordination for recovery.
- **Two-Phase Commit (2PC):**
  - Ensures atomicity across multiple sites.
  - Steps:
    - **Prepare Phase:** The coordinator asks participants if they are ready to commit.
    - **Commit Phase:** If all participants agree, the transaction is committed.

Recovery in Distributed Databases

- Disadvantages:
  - High communication overhead.
  - Blocking issues during failures.

- **Three-Phase Commit (3PC):**
  - Enhances 2PC by introducing a pre-commit phase to avoid blocking.

Practical Examples of Recovery

- **Banking System:**
  - If a power outage occurs during a money transfer, recovery ensures that either the transfer is fully completed or rolled back.
- **E-commerce:**
  - During a crash, recovery ensures that customer orders are not lost or duplicated.
- **Social Media:**
  - Recovery ensures that user actions (e.g., posting a comment) are consistently reflected even after system failures.

Challenges in Recovery

- **Performance Overhead:**
  - Logging and checkpointing add overhead to transaction processing.
- **Storage Requirements:**
  - Logs and shadow copies consume additional storage space.
- **Complexity:**
  - Advanced techniques like ARIES are complex to implement and maintain.
- **Distributed Recovery:**
  - Coordinating recovery across multiple nodes or sites is challenging.

Week 17

Future Trends in Database Management Systems (DBMS)

- The field of Database Management Systems (DBMS) continues to evolve rapidly, driven by the increasing demands of big data, cloud computing, AI integration, and real-time analytics. Emerging trends are reshaping how data is stored, accessed, and managed.

What Drives DBMS Evolution?

- Increasing data volume, variety, and velocity (3Vs of Big Data).
- Growing demand for real-time processing and analytics.
- Advances in technology like AI, IoT, and cloud computing.

Objectives of Modern DBMS Trends

- Enhance performance, scalability, and reliability.
- Improve flexibility for handling diverse workloads.
- Support integration with modern applications and frameworks.

Future Trends in DBMS

- **Cloud-Based Databases**
  - **Trend**: Migration to cloud platforms for scalable and cost-effective database solutions.
  - **Features**:
    - On-demand scalability.
    - Managed services for reduced administrative overhead.
    - Pay-as-you-go pricing.
  - **Examples**:
    - Amazon RDS, Google BigQuery, Microsoft Azure SQL Database.

Future Trends in DBMS

- **Benefits:**High availability with minimal downtime.
- Seamless integration with other cloud-native services.

Future Trends in DBMS

- Multi-Model Databases:
- **Trend**: Support for multiple data models (e.g., relational, document, graph) in a single system.
- **Features:**
  - Flexibility to store and query different data types.
  - Unified access to diverse data models.
- **Examples:**
  - ArangoDB, Couchbase, Oracle.
- **Benefits:**
  - Simplifies application development by reducing the need for multiple database systems.

NoSQL and NewSQL Evolution

- **NoSQL:**
  - Continued adoption for unstructured and semi-structured data.
  - Enhanced query capabilities in NoSQL databases.
  - Examples: MongoDB, Cassandra, Redis.

- **NewSQL:**
  - Combines the scalability of NoSQL with the ACID guarantees of traditional databases.
  - Examples: CockroachDB, Google Spanner.

NoSQL and NewSQL Evolution

- **Benefits:**Improved performance and consistency for distributed systems.

Artificial Intelligence and Machine Learning Integration

- **Trend**: AI-driven databases that leverage machine learning for optimization.
- **Applications**:
  - Query optimization using AI algorithms.
  - Predictive maintenance for database systems.
  - Automated indexing and schema design.

Artificial Intelligence and Machine Learning Integration

- **Examples:**Oracle Autonomous Database.
- **Benefits:**Reduces the need for manual tuning and management.
- Enhances performance and efficiency.

Real-Time Analytics and Stream Processing

- **Trend:** Increasing demand for real-time data analysis and decision-making.
- **Features:**
  - Support for real-time data ingestion and querying.
  - Integration with streaming platforms like Apache Kafka.
- **Examples:**
  - Apache Druid, Snowflake, Google BigQuery.
- **Benefits:**
  - Enables applications like fraud detection, IoT monitoring, and personalized recommendations.

Blockchain and Decentralized Databases

- **Trend**: Integration of blockchain technology into databases.
- **Features**:
  - Immutable ledgers for secure and transparent data storage.
  - Support for distributed and decentralized systems.
- **Examples**:
  - BigchainDB, Hyperledger Fabric.
- **Benefits**:
  - Enhanced data security and auditability.
  - Suitable for industries like finance, supply chain, and healthcare.

In-Memory Databases

- **Trend:** Use of in-memory databases for ultra-fast performance.
- **Features:**
  - Data stored in RAM instead of traditional storage.
  - Optimized for low-latency access.
- **Examples:**
  - SAP HANA, Redis, Memcached.
- **Benefits:**
  - Accelerates analytics and real-time processing.

Challenges in Adopting New Trends

- **Integration with Legacy Systems:**
  - Difficulty in integrating modern DBMS with existing infrastructure.
- **Cost:**
  - High cost of adopting and maintaining advanced systems.
- **Security Concerns:**
  - Increased attack surface with distributed and cloud-based databases.
- **Skill Gap:**
  - Need for specialized knowledge to manage advanced databases.