Bachelor of Science
(B.Sc. - PCM)

Database Management System
(DBSPDS201T24)

Self-Learning Material
(SEM -II)

Jaipur National University
Centre for Distance and Online Education

Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

EXPERT COMMITTEE

Prof. Sunil Gupta
(Dept. of Computer and Systems Sciences)
JNU, Jaipur

Dr. Shalini Rajawat
(Dept. of Computer and Systems Sciences)
JNU, Jaipur

COURSE COORDINATOR

Shish Kumar Dubey
(Dept. of Computer and Systems Sciences)
JNU, Jaipur

UNIT PREPARATION

Unit Writer(s)	**Assisting & Proofreading**	**Unit Editor**
Mr. Ram Lal Yadav Dept. of Computer and Systems Sciences, JNU, Jaipur (Unit 1-5)	Mr. Satender Singh (Dept. of Computer and Systems Sciences) JNU, Jaipur	Ms. Rachana Yadav (Dept of Computer and Systems Sciences) JNU, Jaipur

Secretarial Assistance
Mr. Mukesh Sharma

COURSE INTRODUCTION

Welcome to the Database Management Systems (DBMS) course! This course is designed to introduce you to the fundamental concepts and practices of managing databases effectively. You will explore how databases are structured, how data is stored, and how to retrieve and manipulate it using SQL, the standard query language for interacting with databases.

Throughout the course, you'll gain practical skills in designing database schemas, implementing data models, and ensuring data integrity and security. You'll also learn about different types of databases and their applications, from relational databases to newer NoSQL systems. By working on hands-on projects and real-world scenarios, you will develop a strong understanding of how to manage and utilize databases to support various applications and business needs.

By the end of this course, you will have a solid foundation in database management, preparing you to work with databases in various professional settings and laying the groundwork for more advanced topics in data management and analysis.

Course Outcomes:

At the completion of the course, a student will be able to:

1. Design and implement database schemas using relational database principles.
2. Write and optimize SQL queries for data retrieval, manipulation, and management.
3. Understand and apply key concepts such as normalization, indexing, and transaction management.
4. Implement data integrity and security measures to protect and manage database information.
5. Work with different types of databases, including relational and NoSQL systems, and understand their respective use cases.
6. Utilize database management tools and software to create, manage, and maintain databases effectively.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

Unit 1

Introduction to Database Management System

Learning Outcomes:

- Students will be capable of learning about the database and database users.
- Students will be able to understand the Characteristics of the database and database systems.
- Students will be able to understand the concepts and architecture of DBMS.
- Students will be able to understand about the schemas and Instances.

Structure

1.1 Database Management System

In today's data-driven world, the effective management of data is crucial for the success of businesses, organizations, and even individuals. Data, often referred to as the new oil, underpins decision-making processes, drives strategic initiatives, and fuels innovation. A Database Management System (DBMS) serves as the backbone of data management, providing the tools and frameworks necessary to store, retrieve, and manipulate data efficiently and securely.

A DBMS is “a software system designed to facilitate the creation, maintenance, and use of databases”. It acts as an intermediary between end-users and databases, ensuring that data is consistently organized and easily accessible. At its core, a DBMS manages data structures, enforces data integrity, and provides a suite of functionalities that enable users to perform complex queries and transactions. By abstracting the underlying hardware complexities, DBMSs allow users to interact with data using high-level query languages such as SQL (Structured Query Language).

It is impossible to exaggerate the significance of DBMSs in contemporary computing. Compared to conventional file-based data management systems, they provide a number of benefits, such as better data integration, increased data security, and better data sharing. Multiple users can access data concurrently with DBMSs, guaranteeing accurate and consistent updates to the data. Additionally, they uphold data integrity restrictions, including "primary keys" and "foreign keys," to guarantee the precision and dependability of the data that is saved.

1.2 File-Based Systems

A file-based system is an early method of managing data where information is stored in separate files, and each file is independently managed by the operating system. Before the advent of Database Management Systems (DBMS), file-based systems were the primary way to organize and store data. These systems involve the use of application programs to perform operations on data stored in flat files, such as reading, writing, updating, and deleting records.

1.2.1 Characteristics of File-Based Systems

1. Data Storage: Data is stored in flat files, typically organized in directories and subdirectories on a storage device.

2. Access Methods: Various access methods like sequential access and direct access are used to retrieve and manipulate data.
3. Data Independence: There is a lack of data independence, meaning “that changes to data structure often require changes to application programs that use the data”.
4. Redundancy and Duplication: Data redundancy is common as multiple copies of the same data may be stored in different files.
5. File Formats: Each application might have its own file format, leading to inconsistencies and difficulties in data sharing.
6. Limited Query Capabilities: Query capabilities are generally limited and depend heavily on custom application code.

1.3 Drawbacks of File-Based Systems

1. “Data Redundancy and Inconsistency”: One of the most significant drawbacks of file-based systems is “data redundancy”, where same piece of data is stored in multiple places. This redundancy can lead to inconsistencies, where different files contain conflicting versions of the same data. For example, if a customer’s address is stored in multiple files and only one file is updated when the address changes, the system ends up with conflicting information.
2. Difficulty in “Data Access”: File-based systems often require extensive programming to access and manage data. Since there is no uniform way to query data across different files, each application must have its own method for accessing the data it needs. This can make data retrieval cumbersome and time-consuming, especially as the volume of data grows.
3. “Data Isolation”: In file-based systems, data is isolated in separate files, making it difficult to retrieve related data across different files. For instance, extracting a comprehensive report that requires data from multiple files would necessitate significant programming effort to manually join and correlate data from these disparate sources.
4. “Integrity Problems”: Ensuring data integrity (i.e., accuracy and consistency of data) is challenging in a file-based system. There are no built-in mechanisms to enforce constraints (such as unique keys, referential integrity, or validation rules) across the files.

As a result, data integrity must be enforced at the application level, which is error-prone and difficult to maintain.

5. “Atomicity Issues”: File-based systems lack support for transaction management, which is essential for ensuring that multiple operations on data are completed successfully or rolled back in case of an error. This lack of atomicity control can lead to partial updates, leaving the data in an inconsistent state.
6. “Security and Access Control”: Implementing robust security measures in file-based systems is complex. These systems typically rely on the operating system’s file permissions, which offer limited granularity. Fine-grained access control, which is necessary to restrict access to specific parts of the data for different users, is challenging to implement and manage.
7. “Concurrency Control”: File-based systems do not provide built-in support for managing concurrent access to data by multiple users. This can lead to issues such as data corruption or loss if multiple users try to modify the same data simultaneously without proper synchronization mechanisms.
8. “Scalability and Performance”: As the volume of data grows, file-based systems often struggle with scalability and performance. Searching and retrieving data from large files can become slow and inefficient. Moreover, maintaining and managing a growing number of files adds to the complexity and can degrade system performance over time.
9. “Data Integration Difficulties”: Integrating data from different sources is inherently difficult in file-based systems. The lack of standardization in file formats and structures means that significant effort is required to merge and reconcile data from different files.
10. “Backup and Recovery Challenges”: Implementing effective backup and recovery strategies is more complex with file-based systems. Each file must be backed up independently, and restoring data to a consistent state after a failure can be cumbersome and error-prone.

1.4 Data and Information

Data: Data refers to raw, unprocessed facts that are collected and stored. These facts can be numbers, characters, symbols, or images and, on their own, have no inherent meaning. Data can be generated from various sources, such as transactions, observations, and measurements. For

example, a list of temperatures recorded every hour is data. It lacks context and does not provide any insights on its own.

Information: Information, on the other hand, is processed data that has been organized, structured, or presented in a given context to make it meaningful and useful. When data is analyzed and interpreted, it becomes information. For instance, if the temperature data is used to determine the average temperature over a week, this result is information. Information is crucial for decision-making processes as it provides insights and knowledge derived from data.

1.5 Database

A database is “an organized collection of data that is stored and accessed electronically”. Databases are designed to manage large volumes of data efficiently and to allow easy “retrieval, updating, and management of data”. The data in a database is typically structured in a way that makes it easily accessible, usually in the form of tables, which consist of rows and columns.

Databases are used in various applications, ranging from small personal projects to large enterprise systems. They support operations such as data insertion, querying, updating, and deletion, ensuring data integrity and security. Examples “MySQL”, “PostgreSQL”, and “Oracle”, as well as “NoSQL” databases like “MongoDB” and “Cassandra”, which cater to specific data management needs.

1.6 “Database Management System” (DBMS)

A “Database Management System (DBMS)” is a software system that provides tools and functionalities required to create, manage, and manipulate databases. It serves as an intermediary between database and the end-users or application programs, ensuring that data is consistently organized and remains easily accessible.

The primary functions of “DBMS” include:

1. **Data Storage and Retrieval:** A DBMS efficiently stores large volumes of data and provides mechanisms for fast retrieval through various querying techniques.

2. **Data Manipulation**: It allows users to perform operations such as “inserting, updating, and deleting data within database”.
3. **Data Security:** A DBMS enforces security measures to protect data from unauthorized access and breaches, including user authentication and access controls.
4. **Data Integrity:** It maintains the accuracy and consistency of data through integrity constraints and validation rules.
5. **Transaction Management:** DBMS ensures that all database transactions are processed reliably and adhere to the ACID (Atomicity, Consistency, Isolation, Durability) properties to maintain data integrity.
6. **Backup and Recovery:** It provides tools for data backup and recovery, ensuring data can be restored in case of hardware failures, software errors, or other disasters.

Relational DBMS (RDBMS): This type of DBMS is based on the relational model, where data is organized into “tables (relations)”. Each table has “rows (records) and columns (attributes)”. RDBMSs use “SQL” (Structured Query Language) for database management and are known for their robust transactional support and data integrity features. Examples “MySQL”, “PostgreSQL”, and “Oracle Database”.

NoSQL DBMS: “NoSQL” databases are designed to handle “unstructured or semi-structured data” and are optimized for large-scale data storage and real-time web applications. They do not use the traditional table-based relational model. Instead, they use various data models, such as key-value pairs, document stores, wide-column stores, and graph databases. Examples include “MongoDB (document store), Redis (key-value store), Cassandra (wide-column store), and Neo4j (graph database)”.

1.7 Characteristics of the Database Approach

The database approach represents a significant advancement over traditional file-based systems by offering a more structured and efficient way to manage data. This approach is defined by several key characteristics that distinguish it from other data management techniques:

1. Data Abstraction: Databases provide a higher level of data abstraction, separating the physical storage details from the logical data structure. This allows users to interact with data without needing to know the intricacies of where and how the data is physically stored.

2. Data Independence: One of the fundamental advantages of the database approach is data independence, which includes:

- Logical Data Independence: The capacity to change logical schema without altering external schema or “application programs”.
- Physical Data Independence: The ability to modify physical schema without impacting logical schema or application programs.

3. Minimized Data Redundancy: Databases are designed to reduce data redundancy by ensuring that each data item is stored only once. This reduces the risk of inconsistencies and saves storage space.

4. Data Integrity: Databases enforce data integrity constraints to ensure the accuracy and consistency of data. Integrity constraints include “primary keys, foreign keys, unique constraints, and check constraints”, which help maintain valid data across the database.

5. Data Security: The database approach includes robust security mechanisms to protect data from unauthorized access and breaches. This includes user authentication, access controls, and encryption methods to safeguard sensitive information.

6. Concurrent Access: Databases are designed to handle concurrent data access by multiple users efficiently. They implement mechanisms like locking, transactions, and isolation levels to ensure data consistency and integrity when multiple users access or modify the data simultaneously.

7. Data Consistency: By using transactions and ensuring ACID (Atomicity, Consistency, Isolation, Durability) properties, databases maintain data consistency even in the event of system failures or concurrent updates.

8. Data Sharing: Databases facilitate data sharing among multiple users and applications. This is achieved through centralized data management, where data can be accessed and modified by authorized users based on their roles and permissions.

9. Backup and Recovery: Databases provide tools for regular data backups and recovery mechanisms to restore data in case of system failures, ensuring data availability and reliability.

10. Standardized Query Language: Most databases use a standardized query language, such as SQL (Structured Query Language), which allows users to perform complex queries, data manipulations, and transaction management in a consistent manner.

1.8 Data Models

A data model is a "conceptual framework for organizing and defining the structure, relationships, and constraints of the data in a database". It serves as a blueprint for how data is "stored, accessed, and manipulated". Several data models are commonly used in database systems:

1. "Relational Data Model":

- Structure: Organizes data into "tables (relations)", each consisting of "rows (tuples)" and "columns (attributes)".
- Relationships: Relationships between tables are established through foreign keys.
- Example: A customer table linked to an orders table via a customer ID.
- Advantages: Simplicity, flexibility, strong theoretical foundation, and wide adoption.

2. "Hierarchical Data Model":

- Structure: Organizes data into a tree-like structure, with a single root and multiple levels of parent-child relationships.
- Relationships: Each parent can have multiple children, but each child has only one parent.
- Example: An organizational chart or file directory system.
- Advantages: Simple representation of 1:N relationships and fast access for hierarchically structured data.

3. “Network Data Model”:

- Structure: Similar to the hierarchical model but allows more complex relationships with multiple parent-child (many-to-many) connections.
- Relationships: Data is organized in a graph structure, enabling more flexible relationships.
- Example: A university course system where courses have multiple prerequisites and can be prerequisites for multiple courses.
- Advantages: More flexible than the hierarchical model and can represent complex relationships.

4. “Object-Oriented Data Model”:

- Structure: Combines database capabilities with object-oriented programming principles, representing data as objects.
- Relationships: Objects can inherit properties and behaviors from other objects.
- Example: A multimedia database where images, videos, and text are treated as objects with associated properties and methods.
- Advantages: Seamless integration with object-oriented programming languages and support for complex data types.

5. NoSQL Data Models:

- Document Model: Stores data as documents (e.g., JSON, BSON). Flexible schema design, suitable for semi-structured data.
- Key-Value Model: Stores data as key-value pairs, providing fast access for simple queries.
- Column-Family Model: Organizes data into columns and column families, ideal for handling large volumes of data with high read/write throughput.
- Graph Model: Represents data as nodes and edges, optimized for managing and querying highly interconnected data.
- Advantages: High scalability, flexibility, and performance for specific use cases like big data, real-time web applications, and complex relationships.

1.9 DBMS Architecture

Database Management System (DBMS) architecture refers to the design and structure of a DBMS, which determines how data is stored, accessed, and managed. A well-defined architecture ensures efficient data management, scalability, and ease of use. The architecture of a DBMS can be classified into several levels, each providing different views and functionalities to various users and applications. The most common architectures are the three-tier and two-tier models.

1.9.1 Three-Tier Architecture

Three-tier architecture is the most widely used DBMS architecture, offering a high level of abstraction and separation of concerns. It consists of three layers:

1. Presentation Tier (User Interface Layer):

- Function: This layer is responsible for interacting with the end-users. It provides a graphical user interface (GUI) or web-based interface for users to access the database.
- Components: It includes various tools and applications such as forms, reports, and web pages.
- Example: A web application interface where users input data and view query results.

2. Application Tier (Business Logic Layer):

- Function: This middle layer processes the business logic and handles the interactions between the presentation tier and the data tier. It interprets user commands, makes logical decisions, performs computations, and coordinates database operations.
- Components: It includes application servers, business logic components, and middleware.
- Example: A server-side script that validates user input, processes transactions, and constructs SQL queries.

3. Data Tier (Database Layer):

- Function: This layer is responsible for “storing, retrieving, and managing data”. It includes the actual database and the DBMS that manages the database operations.

- Components: It consists of the database engine, storage management, and data access APIs.
- Example: The DBMS like MySQL, Oracle, or MongoDB that stores user data and executes SQL queries.

1.9.2 Two-Tier Architecture

Two-tier architecture is simpler than three-tier architecture and involves direct communication between the client and the server. It consists of two layers:

1. Client Tier:

- Function: The client tier includes the user interface and the application that directly communicates with the database server. Users interact with the database through the client application, which sends requests to the server.
- Components: It includes client applications, such as desktop applications or front-end software.
- Example: A desktop application that connects directly to a database to perform queries and updates.

2. Server Tier:

- Function: The server tier manages the database and handles client requests. It processes queries, performs data manipulations, and returns results to the client.
- Components: It includes the DBMS, database engine, and data storage.
- Example: A DBMS like Microsoft SQL Server or PostgreSQL running on a server machine.

1.9.3 Components of DBMS Architecture

Regardless of the specific tier architecture, a DBMS typically includes several key components that work together to manage data efficiently:

1. Database Engine:

- Function: The core service for “storing, processing, and securing data”. It provides controlled access and rapid transaction processing.
- Components: Query processor, transaction manager, storage manager, and buffer manager.

2. Query Processor:

- Function: Interprets and executes SQL queries. It optimizes query execution plans for performance.
- Components: SQL parser, query optimizer, and execution engine.

3. Transaction Manager:

- Function: Ensures the ACID properties of transactions, managing concurrency control and recovery.
- Components: Lock manager, log manager, and recovery manager.

4. Storage Manager:

- Function: Manages the physical storage of data on disk. It handles file organization, indexing, and access methods.
- Components: Data files, indexes, and storage buffers.

5. Buffer Manager:

- Function: Manages memory buffers to improve data retrieval performance by caching frequently accessed data.
- Components: Buffer pools and cache algorithms.

6. Metadata Manager:

- Function: Maintains information about the database structure, such as schemas, tables, columns, and data types.
- Components: Data dictionary and system catalog.

7. Security Manager:

- Function: Controls access to the database, ensuring that only authorized users can perform specific operations.
- Components: Authentication, authorization, and encryption mechanisms.

1.10 Schema and Instances in database systems

In database management systems, the concepts of schema and instances are fundamental to understanding how data is organized, stored, and managed. These concepts help to distinguish between the structure of the data and the actual data stored within that structure.

1.10.1 Schema

A schema is blueprint or the logical structure that defines organization of data in a database. It outlines how data is organized and how the relationships among the data are managed. Schemas provide a high-level view of the database, abstracting the details of data storage and focusing on the structure and constraints.

Types of Schema:

1. Physical Schema:

- Describes how data is physically stored in database. This includes details about the storage structures, file formats, indexing methods, and physical locations of the data.
- Example: Defining how a table is stored on disk, whether it's in a heap file, clustered file, or indexed file.

2. Logical Schema:

- Describes the logical structure of the data, including tables, columns, data types, constraints, relationships, and indexes. It is a higher-level abstraction than the physical schema and focuses on how data is logically organized and interrelated.
- Example: A table definition with columns for customer information such as `CustomerID`, `Name`, `Address`, and `PhoneNumber`.

3. External Schema:

- Defines how different users or applications view the data. It provides customized views for different users based on their requirements and access privileges.
- Example: A sales department might have a view that only shows customer names and purchase history, while the accounting department has a view that includes financial details.

Schema Characteristics:

- Static Nature: Schemas are typically static, meaning they do not change frequently. They are defined during the database design phase and modified only when necessary.
- Integrity Constraints: Schemas include constraints such as “primary keys, foreign keys, unique constraints”, and check constraints to ensure data integrity and consistency.

1.10.2 Instances

An instance refers to the actual data stored in the database at any given point in time. While the schema is the blueprint, the instance is the realization of that blueprint with real data values.

Characteristics of Instances:

- Dynamic Nature: Unlike schemas, instances are dynamic and can change frequently as data is inserted, updated, or deleted.
- State of the Database: An instance represents the state of the database at a particular moment. Every update operation (insert, delete, update) can change the database instance.
- Variability: Multiple instances can exist over time for the same schema, reflecting the changes in the data.

Example of Schema and Instance:

- Schema Definition:

```sql
“CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
```

```
    Address VARCHAR(255),
    PhoneNumber VARCHAR(15)
);"
```

- Instance:

CustomerID	Name	Address	PhoneNumber
1	John Doe	123 Maple Street	555-1234
2	Jane Smith	456 Oak Avenue	555-5678
3	Emily Davis	789 Pine Road	555-8765

In this example, the schema defines the structure of the `Customers` table, including the columns and their data types. The instance is the actual data stored in the `Customers` table at a specific point in time.

1.11 Data Independence

Data independence refers to capacity to change schema at one level of a database system without having to alter schema at next higher level. This concept is fundamental in the design of database management systems (DBMS) as it ensures that data can be modified, updated, and maintained without affecting the overall structure and operation of the database. There are two primary types of data independence: "logical data independence" and "physical data independence".

1.11.1 "Logical Data Independence"

Logical data independence is "ability to change the conceptual schema without altering the external schema or application programs". The conceptual schema defines the logical structure of the database, including tables, views, and relationships. Changes at this level might include adding new fields to a table, creating new tables, or modifying existing relationships.

Examples of Logical Data Independence:

- Adding a new column to a table.

- Modifying the relationship between two tables, such as changing a one-to-many relationship to a many-to-many relationship.
- Creating or removing indexes to improve query performance.

Logical data independence is crucial because it allows the database to evolve and adapt to new requirements without necessitating changes to the application code that accesses the data. This reduces maintenance costs and minimizes the risk of introducing errors during modifications.

1.11.2 Physical Data Independence

Physical data independence is “ability to change physical schema without affecting conceptual schema”. The physical schema defines how data is stored in the database, including file structures, storage locations, and indexing mechanisms. Changes at this level might include moving data to a new storage device, changing the file organization method, or optimizing storage for performance improvements.

Examples of Physical Data Independence:

- Moving the database files from one disk to another.
- Changing the storage format of the data, such as from a heap file to a clustered file.
- Reorganizing the database to improve access times and reduce storage space.

Physical data independence ensures that changes to the physical storage mechanisms do not impact how users and applications interact with the data. This abstraction allows database administrators to optimize storage and improve performance without requiring changes to application programs.

Importance of Data Independence

1. Reduced Maintenance Costs: By allowing changes at one level without affecting other levels, data independence reduces the effort and cost required to maintain the database and related applications.

2. Improved Flexibility and Scalability: Databases can evolve and scale to meet changing requirements without necessitating extensive modifications to applications or user interfaces.
3. Enhanced Data Integrity: Consistent and independent schema definitions at different levels ensure that changes do not introduce inconsistencies or errors.
4. Ease of Data Migration and Upgrade: Physical data independence facilitates the migration of databases to new hardware or storage systems and allows for performance optimizations without impacting users.
5. Separation of Concerns: By separating the logical view of the data from its physical storage, developers and administrators can focus on their specific areas of responsibility, improving productivity and collaboration.

1.12 Summary

- Database users within a DBMS context encompass individuals benefiting from database usage, leveraging DBMS-provided applications and interfaces for data access and retrieval.
- Database users are granted access rights for reading, inserting, updating, and deleting specific objects, thereby defining a set of fields and business rules. The creation of database users is typically facilitated through the Database Access action in the Users application.
- The term schema denotes the overall description of a database, while an instance represents the collection of data and information stored within the database at any given point, with the schema remaining consistent throughout.
- Database programming languages enable the definition and manipulation of databases, with four primary types including “Data Definition Language (DDL)”, “Data Manipulation Language (DML)”, “Data Control Language (DCL)”, and “Transaction Control Language (TCL)”.
- “DDL” commands are utilized to alter or establish the schema and metadata of a database, while DML commands enable the manipulation of data stored within schema objects.

- A DBMS interface serves as a user-friendly platform allowing users to input queries into a database without necessitating the use of query languages.

1.13 Self-Assessment Questions

1 What is a Database?
2 What are the different types of Database users?
3 What are the Characteristics of the database?
4 What is Database Systems?
5 Explain the concept and architecture of the Database.
6 What is schemas and instances?
7 What is Data Independence?
8 What are the different types of DBMS Languages?

References

- Atkinson, M.P. (1981) *Database*. Maidenhead: PergamumInfoTech.
- Frank, L. and Helmersen, O. (1988) *Database theory and practice*. Wokingham, England: Addison-Wesley Publishing Company.
- *Database users: 2nd Toronto conference: Selected papers* (1990). Canadian Association for Information Science.
- *Database* (no date). Weston Ct.: Online, Inc.
- Kambayashi, Y. (1982) *Database*. Rockville, MD: Computer Science Press.

Unit - 2

Entity Relationship and Enhanced ER Modeling

Learning Outcomes:

- Students will be capable of learning about the Entity-Relationship Modelling.
- Students will be able to understand the entity types and relationships
- Students will be able to understand the concepts of Database design using ER.
- Students will be able to understand about the EER modeling, Schema and Constraints.

Structure

2.1 Entity-Relationship (ER) Modeling

Entity-relationship (ER) modeling is a foundational method used in database design to depict the relationships between entities within a system. It serves as a visual representation of how data is structured and interconnected, facilitating the understanding of complex data relationships. ER modeling involves several key concepts, including entity types, attributes, relationships, and cardinality constraints. In ER modeling, two principal elements are entity types and relationships.

2.2 Entity Types

Entity types in database management systems (DBMS) serve as categories for organizing data, representing groups of objects or entities sharing similar attributes. These types establish a structure for defining and managing information within a database, facilitating efficient data storage and retrieval.

Distinctive Characteristics of Entity Types:

1. Identification: Each entity type possesses a unique name within the database schema, enabling differentiation between different types of entities and aiding in data organization.
2. Attributes: Entity types comprise attributes defining the properties or characteristics of entities belonging to that type. These attributes, represented as columns in database tables, provide specific details about the entities.
3. Instances: Instances of an entity type represent individual occurrences or objects within the database. Each instance is characterized by the values of its attributes, delineating its unique properties.

Illustrative Example:

Consider a library system database:

- Book: Represents various books available in the library, with attributes like Book ID, Title, Author, Publisher, and Publication Year.
- Member: Signifies individuals registered as library members, with attributes such as Member ID, Name, Address, Email, and Phone.

- Transaction: Denotes borrowing or lending activities, featuring attributes like Transaction ID, BookID, Member ID, Borrow Date, and Return Date.

In this example, Book, Member, and Transaction are entity types, each defined by a set of attributes reflecting the properties of entities associated with that type.

Significance of Entity Types:

1. Data Organization: Entity types provide a structured framework for organizing data within a database, enhancing manageability and organization.
2. Data Integrity: By delineating specific entity types with well-defined attributes, data integrity is preserved, ensuring the storage of valid and relevant information.
3. Data Modeling: Entity types are fundamental in data modeling, enabling the creation of a logical representation of data requirements and relationships within the database.
4. Data Retrieval: Entity types facilitate efficient data retrieval by categorizing information into meaningful groups, facilitating streamlined access and retrieval.

In essence, entity types play a pivotal role in designing and implementing databases, offering a structured approach to managing and organizing data. Through the definition of entity types and their attributes, a logical and efficient representation of data can be established within the database environment.

Key Characteristics of Entity Types:

- Identification: Each entity type possesses a unique name for identification within the system. For instance, in a university database, entity types could include `Student`, `Professor`, and `Course`.
- Attributes: Entity types encompass attributes that portray entity properties. These attributes serve as the entity type's columns and represent specific entity details. For instance, attributes of a `Student` entity type might include `StudentID`, `Name`, `Address`, and `DateOfBirth`.
- Instances: Instances of an entity type represent individual occurrences or objects in the system. For instance, each student enrolled in the university would be an instance of the `Student` entity type.

2.3 Relationships

Relationships in database management systems (DBMS) delineate the associations between data entities, essential for structuring and organizing data effectively. These relationships facilitate efficient data retrieval and manipulation within the database. Typically, relationships in DBMS are classified into three main types: one-to-one, one-to-many, and many-to-many.

2.3.1 One-to-One Relationship

A one-to-one relationship signifies that one entity from one entity set corresponds to exactly one entity from another entity set, and vice versa. This straightforward relationship type is employed when two entities share a unique, singular relationship.

Example:

- An employee may be assigned to exactly one office, and each office is allocated to only one employee. Thus, the relationship between the Employee entity set and the Office entity set is one-to-one.

2.3.2 One-to-Many Relationship

In a one-to-many relationship, one entity from one entity set can be associated with one or more entities from another entity set, while an entity from the latter entity set is linked to only one entity from the former entity set. This type is prevalent when one entity can have multiple related entities, with each related entity having a single parent entity.

Example:

- A department can have numerous employees, yet each employee can belong to only one department. Hence, the relationship between the Department entity set and the Employee entity set is one-to-many.

2.3.3 Many-to-Many Relationship

A many-to-many relationship indicates that entities from both entity sets can be linked to multiple entities from the other entity set. This relationship necessitates the utilization of an

associative entity, such as a junction or linking table, to represent the associations between the two entity sets.

Example:
- A student can enroll in multiple courses, and conversely, a course can have several students enrolled in it. Therefore, the relationship between the Student entity set and the Course entity set is many-to-many. To represent this relationship, a third entity set, Enrollment, is introduced, containing attributes like StudentID and CourseID to link students to the courses they are enrolled in.

Importance of Relationships in DBMS

1. Data Integrity: Relationships uphold data integrity by enforcing referential integrity constraints, ensuring the storage of only valid and existing data.
2. Data Consistency: They maintain data consistency across different entity sets, mitigating the risk of redundancy and inconsistencies.
3. Query Optimization: Well-defined relationships facilitate efficient query optimization, leading to faster data retrieval and analysis.
4. Data Modeling: Relationships are crucial for data modeling, enabling the creation of a logical and structured representation of data requirements.
5. Normalization: They play a pivotal role in database normalization, organizing data to minimize redundancy and dependency, resulting in an efficient and scalable database design.

Key Characteristics of Relationships:

- Identification: Each relationship has a descriptive name that elucidates the association between involved entity types. For example, in a university database, a relationship between `Student` and `Course` might be termed `Enrollment`.
- Multiplicity: This defines the number of instances of one entity type associated with one instance of another entity type in a relationship. It specifies the cardinality or participation constraints of the relationship. For instance, a `Student` can enroll in multiple `Courses`, indicating a one-to-many relationship.

- Role: Role signifies the function each entity type plays in the relationship, clarifying its semantic meaning. For example, in a relationship between `Student` and `Course`, the `Student` entity type may play the role of "Enrollee," while the `Course` entity type may play the role of "Offering."
- Attributes: Relationships may possess attributes delineating relationship properties, distinct from entity type attributes.

2.4 Enhanced ER Modeling

Enhanced Entity-Relationship (ER) modeling extends traditional ER modeling by introducing additional constructs to represent complex data relationships and constraints within the system. These enhancements provide a more comprehensive and flexible approach to database design, allowing for a richer representation of data semantics and constraints.

Key Constructs of Enhanced ER Modeling:

1. **Subtypes and Supertypes**: Subtypes represent specialized categories of entities within a broader entity type, inheriting attributes and relationships from their supertypes. This modeling approach enables the representation of hierarchical relationships and allows for the definition of specific characteristics for each subtype.
2. **Specialization and Generalization**: Specialization defines the process of creating subtypes from a supertype, while generalization involves the abstraction of common characteristics from multiple entity types into a single generalized entity type. These concepts facilitate the modeling of inheritance hierarchies and promote reusability and abstraction in database design.
3. **Aggregation**: Aggregation represents a relationship between an entity and a higher-level entity composed of multiple lower-level entities. This construct allows for the modeling of complex relationships where one entity type is composed of or is associated with multiple smaller entities. Aggregation is particularly useful for representing part-whole relationships and composite objects.
4. **Union Types**: Union types represent a relationship between an entity type and a set of entity types, allowing for more flexible modeling of relationships where an entity can be

associated with entities of different types. This construct is beneficial for representing heterogeneous relationships and accommodating diverse data structures.

5. **Constraints**: Enhanced ER models can include various constraints such as participation constraints, cardinality constraints, and key constraints. These constraints enforce data integrity and specify the rules governing the relationships between entities, ensuring consistency and correctness in the database schema.

2.5 Schema Definition

In SQL, a schema defines the structure of the database, including tables, columns, data types, relationships, and constraints. It provides a blueprint for organizing and storing data in a structured format.

Example Schema Definition:

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT,
    CONSTRAINT FK_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);

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

In this example, we define two tables: Employees and Departments. The Employees table has columns for EmployeeID, FirstName, LastName, and DepartmentID, where DepartmentID is a foreign key referencing the Departments table.

2.6 Constraints

Constraints in SQL enforce rules and conditions on the data stored in the database, ensuring data integrity and consistency. Common constraints include primary key, foreign key, unique, not null, and check constraints.

Example Constraints:

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DepartmentID INT,
    CONSTRAINT FK_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);

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

In this example, we enforce the following constraints:

- The EmployeeID column in the Employees table is defined as the primary key.
- The FirstName and LastName columns in the Employees table are defined as not null.
- The DepartmentName column in the Departments table is defined as unique.

2.7 Object Modeling

Object modeling in SQL involves representing real-world entities and their relationships in the database schema. This includes defining tables, views, indexes, and other database objects to model the structure and behavior of the data.

Example Object Modeling:

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT,
    CONSTRAINT FK_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);

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

CREATE VIEW EmployeeDetails AS
SELECT e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName
FROM Employees e
JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

In this example, we define two tables: Employees and Departments, and create a view EmployeeDetails to display employee information along with their department names. The schema, constraints, and object modeling work together to create a structured and organized database system.

2.8 Summary

Entity-relationship (ER) modeling is a foundational technique used in database design to represent the relationships between entities within a system. It provides a visual means of understanding the structure and connections of data elements. Enhanced entity-relationship

(EER) modeling builds upon this foundation by introducing additional constructs to represent more complex relationships and constraints within the system.

In ER modeling, entities are represented as objects or concepts in the real world, while relationships depict the associations between these entities. Attributes describe the properties or characteristics of entities, providing further detail. This modeling technique is essential for organizing and structuring data, facilitating efficient data retrieval and manipulation.

EER modeling extends ER modeling by introducing constructs such as subtypes and supertypes, specialization and generalization, aggregation, and union types. These enhancements allow for the representation of hierarchical relationships, inheritance hierarchies, composite objects, and heterogeneous relationships. By incorporating these additional constructs, EER modeling provides a more expressive and flexible framework for database design.

2.9 Self-Assessment Questions

1 What are relationship in DBMS?

2 What are the different types of Entity?

3 What is Database Systems?

4 Explain the concept of ER modeling.

5 What is schemas and instances?

6 What is Constraints ?

Unit -3

Relational Data Model

Learning Outcomes:

- Students will be capable of learning about the Relational Data Models
- Students will be able to understand the Concepts and Relational database constraints.
- Students will be able to understand the closure rules and normalisation.
- Students will be able to understand about Relational Algebra.

Structure:

3.1 Relational Data Models

The relational model (RM) represents a database as a collection of relations, where each relation is essentially a table of values. Each row within these tables comprises relevant data values, symbolizing real-world entities like people, places, objects, or connections. The utilization of table and column names aids in comprehending the significance of the values in each row. Despite the physical storage method, the logical organization of data remains independent in the relational model.

The relational model for database management serves as a conceptual framework for representing and managing data within a database. Data within this model is structured into a series of two-dimensional tables or relations, proposed by E.F. Codd. Subsequently, after designing the conceptual database model using an Entity-Relationship (ER) diagram, it becomes imperative to translate it into a relational model feasible for implementation using various RDBMS languages such as Oracle SQL, MySQL, etc. Relational databases store data based on the relational model, where data is stored in relations or tables.

Think about the relation STUDENT, which has the Table 1 attributes ROLL NO, NAME, ADDRESS, AGE

ROLL NO	NAME	ADDRESS	AGE
1	RAHUL	RAIPUR	25
2	KIRTI	BILASPUR	22
3	SWATI	BHILAI	21
4	ROHAN	JAMSHEDPUR	24

Some of the popular Relational Database Management System are:

- DB2 and Informix Dynamic Server by IBM
- Oracle and RDB by Oracle
- SQL Server and Access by Microsoft

Characteristics of Relational Model

- Data is organized in rows and columns, typically represented as relations.
- The relational model describes the relationships between tables where data is stored.

- The relational model supports operations like data definition, manipulation, and transaction management.
- Each column is uniquely named and represents an attribute.
- Each row represents a single entity.
- Relation names are unique within the database.
- Each cell in a relation contains exactly one atomic (single) value.
- Attributes have unique names.
- The attribute domain defines the type of data a column can hold.
- Values within a tuple (row) are unique.
- Tuple order may vary across different instances.

Important terms related to the Relational Database

- Each column in a table represents an attribute, defining the characteristics of a relation.
- Examples of attributes include "Name," "Student Roll no," etc.
- Tables, also known as relations, store data in a tabular format within the relational model. Rows represent records, while columns represent attributes.
- A tuple refers to a single table row, representing a single record.
- Relation Schema encompasses the relation's name and its attributes.
- Degree indicates the total number of attributes in a relation.
- Cardinality refers to the total number of rows in a table.
- Columns define the range of values for a specific attribute.
- A relation instance in an RDBMS comprises a finite set of tuples.
- Duplicate tuples do not exist within relation instances.
- The relation key refers to one or more attributes that uniquely identify each row.
- Each attribute has a predefined value and domain, known as its attribute domain.

Concepts and Relational database constraints

Constraints are rules applied to a table's data columns, ensuring data integrity and reliability by filtering the types of data allowed into tables. These constraints can be imposed at both the column and table levels.

In the relational model, constraints are essential requirements that database data must meet to maintain validity. Before executing any database operation, such as insertion, deletion, or updating, these constraints are checked. If any constraint is violated, the operation will fail. Integrity constraints are of various types in DBMS, with three main categories:

1. **Domain Constraints**: These constraints ensure that attribute values adhere to the related domain and correct data type. Each attribute's value within a tuple must comply with domain requirements. Common data types such as integers, real numbers, characters, Booleans, and variable length strings are examples of this.
2. **Key Constraints**: Key constraints enforce uniqueness within a set of attributes, ensuring that each tuple has a unique identifier or key.
3. **Entity Integrity Constraints**: Entity integrity constraints guarantee the existence of a primary key in every table, preventing null values in primary key attributes.
4. **Referential Integrity Constraints**: These constraints maintain consistency between related tables, ensuring that foreign key values in one table correspond to primary key values in another, preventing orphaned records.

For example

Create DOMAIN Customer Name

CHECK (value not NULL)

For example

E-id	**E-name**	**Phone no**
1	Rohan	9994678990, 9690578999

Due to the fact that the Name in the relationship is a composite attribute and the Phone is a multi-valued attribute, the domain constraint is broken.

Key Constraints

Each relation in the database must contain at least one set of attributes that uniquely defines a tuple, known as keys. For instance, in a STUDENT relation, ROLL NO serves as an example of

a key. This ensures that there are no duplicate tuples within the relation, making it a uniqueness constraint.

A relation may have multiple keys or candidate keys, where a candidate key is the minimal super key. From these candidate keys, one key is chosen as the primary key. While there are no strict rules for selecting the primary key, it is generally recommended to choose the candidate key with the fewest attributes.

The primary key cannot include null values, and therefore, the Not Null constraint is a requirement for the key attributes. This ensures that each tuple in the relation has a valid and non-null value for the primary key attributes.

The following are the two characteristics of a key:

- For each tuple, it must be different.
- It cannot contain any NULL values.

For Example

EID	ENAME	AGE
1	Rahul	22
2	Rohan	21
3	Rohit	23
4	Amar	21
1	Armaan	20

The first and last tuples in the above table have the same value for EID, which is 01, therefore breaking the key constraint.

Entity Integrity Constraints:

No primary key can have a NULL value, according to entity integrity constraints, because we need primary keys to uniquely identify each tuple in a relation.

EID	ENAME	AGE
1	Rahul	20
2	Rohan	21
44	Karan	18
NULL	Prem	23

In the relationship mentioned above, EID is designated as the primary key. As the primary key cannot contain NULL values, it is violated in the third tuple.

Referential Integrity Constraints

Referential integrity ensures that an attribute in a relation can only contain values that exist in another attribute within the same relation or in another relation. This constraint is maintained through the use of foreign keys in DBMS.

A foreign key is a vital attribute within a relation that references attributes in other relations. It establishes a connection between different tables, enforcing relational integrity constraints. However, it is essential that the referenced attribute exists in the related table to maintain referential integrity.

EID	ENAME	DNO
1	Rohan	12
2	Rohit	22
4	Amit	14

Dno	Place
12	Jaipur
13	Bhilai
14	Raipur

The DNO of Table 1 is the foreign key, and the DNO of Table 2 is the primary key in the tables mentioned above.DNO = 22 cannot be used in the foreign key of Table 1 since it is not defined in Table 2's primary key. Referential integrity constraints are thus broken in this situation.

Features of Relational Database Model –

The relational database features are discussed briefly.

- Simplicity of Model The relational database model is simpler compared to other models, requiring straightforward SQL queries for data handling, without the need for complex processing or structuring.
- Ease of Use Users can efficiently retrieve required information using Structured Query Language (SQL) without facing the complexities often associated with databases.
- Accuracy Relational databases are well-structured, minimizing data duplication and ensuring accuracy in data representation.
- Data Integrity Relational Database Management Systems (RDBMS) ensure consistency across all tables, enhancing data integrity and reliability.
- Normalization Normalization breaks down complex data into manageable chunks, reducing storage size and ensuring uniformity in data structure, thus maintaining data integrity.
- Collaboration Multiple users can access and retrieve information simultaneously from the database, even during data updates.
- Security RDBMS systems offer robust security features, allowing only authorized users to access data directly, ensuring data confidentiality and integrity.

Advantages of the Relational Model

- It is simple than the network and the hierarchical model.
- It is very easy and very simple to understand.
- The structure of the relational Model can be changed according to the requirements.
- It is Adaptable, Safe.
- Data reliability
- Applications for operations are simple.

Disadvantages of the Relational Model

- Limited Scalability for Huge Databases: Relational databases may encounter scalability issues when handling large volumes of data, which can impact performance.

- Complex Relationship Management: Managing relationships between tables can sometimes be challenging, requiring careful design and maintenance.
- Complexity and Usability: Relational databases can be complex to set up and maintain, requiring developers and programmers to invest significant time and effort in database management tasks.
- Maintenance Challenges: As data volume increases over time, maintaining a relational database can become more challenging, leading to higher maintenance costs.
- Cost of Setup and Maintenance: Setting up and maintaining a relational database system can incur significant costs, including licensing fees, hardware expenses, and ongoing maintenance costs.
- Performance Issues with Large or Distributed Databases: Large or distributed databases may experience latency and availability issues, affecting overall performance.
- Limitation in Representing Complex Relationships: Relational databases store data in tabular form, which may not adequately represent complex relationships between objects, posing challenges for applications that require intricate data models.
- Performance Degradation with Increased Complexity: As the number of tables and data complexity grows, relational databases may experience slower response times and increased query complexity, leading to performance issues during peak usage times.

3.2 Operations in Relational Model

The relational database model supports the following four fundamental update operations:

- Data is inserted into the relation using insert.
- To remove tuples from a table, use delete.
- You can modify an existing tuple to change the values of certain of its attributes.
- Select enables you to pick a certain set of facts.

Integrity restrictions set in the relational database structure must never be broken when one of these operations is used. If the tuple being deleted is referred to by foreign keys from other tuples in the same database, the delete operation may not be referentially sound.

- **Knowledge Check 1**

  Fill in the blanks with the following sentences.

  1. Foreign Keys serve as the foundation for DBMS referential integrity constraints.
  2. _____are the requirements that must be met for database data in order for the Relational model to be valid.
  3. _____occurs when an attribute of a relation can only take values from an attribute of the same relation or from any other relation.

3.3 ACID Concept

In the realm of database management systems (DBMS), the concept of ACID (Atomicity, Consistency, Isolation, Durability) encapsulates a critical set of properties aimed at guaranteeing the reliability and consistency of transactions, which are the fundamental units of work performed on a database. These properties are pivotal in ensuring that database transactions are processed reliably, even in the face of system failures. Let's delve into each of these ACID properties:

1. Atomicity:

- Atomicity ensures that each transaction is treated as an indivisible unit of work. Transactions are either entirely completed and committed to the database or not executed at all. There is no room for partial execution or an interim state.
- In the event of a failure (such as a system crash or power outage) during transaction execution, the database management system ensures that either all changes made by the transaction are undone (rolled back) or, if the transaction is committed, all changes are applied to the database.

2. Consistency:

- Consistency mandates that the database remains in a valid state both before and after the execution of a transaction. It demands the preservation of all integrity constraints, including referential integrity, domain integrity, and entity integrity.
- Transactions must transition the database from one consistent state to another consistent state. Any breach of integrity constraints leads to the transaction being aborted.

3. Isolation:

- Isolation guarantees that when transactions are executed concurrently, the outcomes are identical to those produced by sequential execution, without any interference or reliance between transactions. Each transaction operates autonomously, oblivious to other transactions.
- Transactions are shielded from each other to forestall interference and uphold data integrity. This is achieved through different isolation levels (e.g., Read Uncommitted, Read Committed, Repeatable Read, Serializable), which determine the degree of isolation between concurrent transactions.

4. Durability:

- Durability assures that once a transaction is committed, its changes are permanently preserved and retained, even in the face of system failures like crashes or power losses. Committed transactions endure system failures and are not lost.
- The DBMS guarantees that all changes made by committed transactions are inscribed onto non-volatile storage (e.g., disk) and endure intact, even if the system undergoes a crash or restart. This ensures the database's durability and prevents data loss.

These ACID properties collectively ensure the dependability, integrity, and durability of database transactions, laying a sturdy groundwork for effective data management within DBMS. They are indispensable for upholding data consistency, thwarting data corruption, and guaranteeing the seamless execution of transactions in a multi-user, concurrent database environment.

3.4 CODD Rules

Codd's rules, formulated by Edgar F. Codd, the inventor of the relational model for database management, serve as a benchmark for evaluating the reliability and effectiveness of database management systems (DBMS). These rules establish the criteria that a system must meet to be considered a true relational database management system (RDBMS). Let's explore Codd's rules:

1. **Rule 0: The Foundation Rule:**
   - This rule lays the groundwork for all other rules and states that for a system to qualify as an RDBMS, it must be able to manage data entirely through its relational capabilities.

2. **Rule 1: The Information Rule:**
   - All data in the database, including metadata, must be represented in the form of tables (relations).

3. **Rule 2: Guaranteed Access Rule:**
   - Each individual data element (atomic value) must be accessible by specifying the table name, primary key value, and attribute name.

4. **Rule 3: Systematic Treatment of NULL Values:**
   - The DBMS must support a representation of missing information and distinguish between a NULL value and a zero value or an empty string.

5. **Rule 4: Dynamic Online Catalog Based on the Relational Model:**
   - The database schema, including metadata such as table definitions, must be stored in the database itself and accessible through the same query language used to access the data.

6. **Rule 5: Comprehensive Data Sublanguage Rule:**
   - The DBMS must support a data manipulation language (DML) that enables users to specify database queries and updates without the need for procedural programming.

7. **Rule 6: View Updating Rule:**
   - Any view that is theoretically updatable must also be updatable by the system.

8. **Rule 7: High-level Insert, Update, and Delete:**
   - The DBMS must support high-level insert, update, and delete operations. These operations should be capable of affecting multiple records in a single transaction.

9. Rule 8: Physical Data Independence:

- Changes to the physical storage structures or access methods must not require changes to the application programs.

10. Rule 9: Logical Data Independence:

- Changes to the logical structure (schema) of the database must not require changes to the application programs.

Codd's rules are intended to ensure that relational database management systems adhere to a consistent set of principles, promoting data integrity, reliability, and flexibility. Meeting these rules signifies that a DBMS is capable of effectively managing data in accordance with the relational model, thereby providing users with a robust and dependable platform for storing and manipulating data.

3.5 Concept of Key

In the realm of databases, a key serves as a fundamental concept, representing a unique identifier for entities within a dataset. It plays a crucial role in ensuring data integrity, facilitating efficient data retrieval, and establishing relationships between entities. Let's delve into the concept of a key:

1. Primary Key:

- A primary key is a unique identifier for each record (tuple) in a table. It ensures that no two records share the same key value, thereby uniquely identifying each entity in the dataset.
- Primary keys are typically composed of one or more attributes (columns) in a table. They serve as the primary means of referencing and accessing individual records.

2. Foreign Key:

- A foreign key is a column or set of columns in a table that establishes a relationship between that table and another table in the database. It represents a reference to the primary key of another table, known as the parent table.

- Foreign keys enforce referential integrity by ensuring that values in the referencing table (child table) correspond to valid values in the referenced table (parent table).

3. Candidate Key:

- A candidate key is a set of attributes that uniquely identifies tuples within a table. Unlike the primary key, a table may have multiple candidate keys.
- From the set of candidate keys, one key is selected as the primary key. The remaining candidate keys, if any, become alternate keys.

4. Alternate Key:

- An alternate key, also known as a secondary key, is a candidate key that is not selected as the primary key. While it uniquely identifies tuples, it is not designated as the primary means of identification for the table.

5. Composite Key:

- A composite key consists of multiple attributes (columns) that together serve as a unique identifier for records in a table. Unlike a single-column key, a composite key involves a combination of two or more attributes.
- Composite keys are often used when no single attribute can uniquely identify records, requiring a combination of attributes for uniqueness.

6. Super Key:

- A super key is a set of attributes that uniquely identifies each tuple within a table. It may include more attributes than necessary for uniqueness.
- Every primary key and candidate key is a super key, but not every super key is a candidate key or primary key.

Keys are integral to database design and play a pivotal role in establishing relationships between tables, enforcing data integrity, and facilitating efficient data retrieval operations such as querying and joining tables. They serve as the foundation for maintaining the integrity and coherence of the database structure, ensuring accurate and reliable data management.

3.6 Normalisation

Normalization is the systematic procedure of structuring data within a database. It entails the creation of tables and the establishment of relationships between them based on predetermined guidelines. The primary objectives of normalization are to protect data integrity and enhance database flexibility by removing redundant information and inconsistent dependencies.

Normalization primarily focuses on reducing redundancy within relationships or groups of relationships. Relational redundancy can lead to anomalies during data insertion, deletion, and updates. By adhering to normalization rules, relational redundancy is minimized or eliminated, thereby enhancing data integrity and consistency.

Normalization is typically carried out through a series of stages known as normal forms. These normal forms define specific criteria that a relation must meet to be considered normalized. Each normal form addresses different aspects of data redundancy and dependency.

Normalization helps in organizing data efficiently and ensures that databases are structured in a way that facilitates data management, retrieval, and maintenance. By eliminating redundancy and adhering to standard formats, normalization enhances the overall integrity and reliability of the database system.

The basic reason for normalisation is the removal of the data anomalies. Dataredundancy is the main reason for the failure of eliminating anomalies, and it results in the data integrity and others such problems as the database increases.

Normalization is the process of structuring the data in the database. Normalization is used to minimize redundancy from a relation or group of relations. Another use for it is the elimination of unwanted features such as Insertion, Update, and Deletion Anomalies. During normalisation, the bigger table is divided into smaller ones, which are linked by relationships.

- An insertion anomaly occurs when there is not enough data to allow the insertion of a new tuple into a relationship.
- A circumstance known as a "deletion anomaly" occurs when some significant data is mistakenly erased together with other data.
- Update Anomaly: An update anomaly is when modifying one data value requires modifying several data rows.

3.7 Types of Normal Form

3.7.1 First Normal Form

When a relation possesses atomic values, it conforms to the first normal form (1NF) in Database Management Systems (DBMS). Simply put, 1NF dictates that each attribute within a table should contain only a single value and should not accommodate multiple values.

In other words, a relation is considered to be in the first normal form when none of the following characteristics are present, or it violates the first normal form if it includes composite or multi-valued attributes. A relation is said to adhere to the first normal form if every attribute within it is a single-valued attribute.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Figure- 3.1 Conversion to 1NF

3.7.2 Second Normal Form

In Database Management Systems (DBMS), when a relation meets the criteria of the First Normal Form and contains no non-prime attribute that is functionally dependent on any proper subset of any candidate key, it achieves the Second Normal Form (2NF).

To qualify for the Second Normal Form, a relation must already adhere to the rules of the First Normal Form and should not display any partial dependencies. A relation attains the

2NF status when there are no partial dependencies observed between any non-prime attribute (attributes not included in any candidate key) and any proper subset of any candidate key in the table. Partial dependency occurs when a non-prime attribute is determined by only a portion of a candidate key.

3.7.3 Third Normal Form

"The Third Normal Form (3NF) is a schema design strategy in relational databases aimed at reducing data redundancy, preventing anomalies, maintaining referential integrity, and facilitating data management through normalization principles.

A relation achieves the third normal form if it is already in the second normal form and does not contain any transitive dependency for non-prime attributes. In 3NF, each non-trivial functional dependency must be explained by at least one of the following criteria:

- X → Y, where X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key)."

3.7.4 Fourth Normal Form

Fourth Normal Form, or 4NF in DBMS terminology, is the state in which a relation adheres to Boyce-Codd Normal Form and does not possess any multi-valued dependencies. In a relation A → B, a multi-valued dependency exists when there are multiple values of B for a single value of A.

A relation qualifies for Boyce-Codd Normal Form (BCNF) if it meets the criteria of the Third Normal Form and the left-hand side (LHS) is a super key for each functional dependency. In other words, if X is a super key in each non-trivial functional dependency X → Y, the relation satisfies BCNF.

3.7.5 Fifth Normal Form

Fifth Normal Form (5NF), also referred to as Projection-Join Normal Form (PJ/NF), represents an advanced level of database normalization aimed at resolving redundancy in relational databases housing multi-valued facts by isolating semantically related multiple relationships.

In 5NF, the focus is on ensuring that if a relation is already in the Fourth Normal Form (4NF) without any join dependencies, the joining process should be lossless. This means that when

combining multiple relations, no information should be lost, maintaining data integrity throughout the normalization process.

Advantages of Normalisation

Data redundancy is reduced with the aid of normalisation.

- A better general organization of the database.
- The internal data consistency of the database.
- A database architecture that is far more flexible.
- The concept of relationship integrity is upheld.

Disadvantages of Normalisation

- You need to determine the user's needs before building the database.
- The performance degrades as the relations are normalized to higher normal forms, like 4NF and 5NF.
- Restoring higher-degree connections to normalcy is a difficult and time-consuming task.
- Inadequate decomposition can lead to a badly designed database, which can lead to major problems.

3.8 Relational algebra

Relational algebra is a procedural query language that operates on instances of relations, accepting them as input and producing instances of relations as output. Queries in relational algebra are executed using a set of operators, which can be both unary and binary. These operators take relations as input and output, treating them recursively. The intermediate results obtained through the recursive application of relational algebra are also regarded as relations:

- Select
- Project
- Union
- Set difference
- Cartesian product
- Rename

“1. Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Symbol− $\sigma_p(r)$

Where **σ** stands for the selection predicate, and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$$\sigma_{subject = "database"}(Books)$$

1. Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An} (r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$$\prod_{subject, author} (Books)$$

2. Union Operation (∪)

It performs binary union between two given relations and is defined as −

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

3. Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author} (Books) - \prod_{author} (Articles)$

4. Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations, and their output will be defined as −

$r X s = \{ q t \mid q \in r \text{ and } t \in s\}$

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

5. Rename operation (ρ)

Even though the outcomes of relational algebra are relations, they lack names. We can rename the output relation using the rename operation. The little Greek character rho is used to indicate the "rename" procedure.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join”

3.9 Summary

The relational model within Database Management Systems (DBMS) is an abstract framework utilized for organizing and managing data stored in a database. It structures information into two-dimensional tables or relations, where each row represents an entity and each column denotes its attributes. Relational algebra is a procedural query language that accepts instances of relations as input and generates instances of relations as output. It employs operators, including both unary and binary, to execute queries effectively.

3.10 Self-Assessment Questions

1. What are Relational Data Models?
2. What are Relational Database Constraints?
3. What is Relational algebra?
4. Explain Normalisation.
5. Explain CODD Rules.
6. Overview of ACID property.

Unit 4

Structured Query Language

Learning Outcomes:

- Students will be capable of learning about the Structured Query Language
- Students will be able to understand the commands of Structured Query Language
- Students will be able to understand the DDL and DML.
- Students will be able to understand about DCL.

Structure

4.1 Introduction to “Structured Query Language” (SQL)

Structured Query Language (SQL) is a powerful tool for managing and manipulating relational databases. It provides the means to create, modify, and query databases, ensuring efficient data management and retrieval. SQL is the standard language for relational database management systems (RDBMS) and is used by various systems such as MySQL, PostgreSQL, Oracle, and SQL Server.

4.2 Commands in SQL

SQL commands can be broadly categorized into several types, each serving a specific purpose:

1. Data Definition Language (DDL):

- “CREATE”: Used to create databases, tables, indexes, and other database objects.
- “ALTER”: Used to modify existing database objects, such as adding or dropping columns in a table.
-“DROP”: Used to delete databases, tables, or other database objects.
- “TRUNCATE”: Used to remove all records from a table while keeping the table structure intact.

2. Data Manipulation Language (DML):

- “SELECT”: Used to query and retrieve data from databases.
- “INSERT”: Used to add new records to a table.
- “UPDATE”: Used to modify existing records in a table.
- “DELETE”: Used to remove records from a table.

3. Data Control Language (DCL):

- GRANT: Used to give users access privileges to the database.
- REVOKE: Used to remove access privileges from users.

4. Transaction Control Language (TCL):

- “COMMIT”: Used to save the current transaction's changes to the database.
- “ROLLBACK”: Used to undo changes made in the current transaction.

- “SAVEPOINT”: Used to set a savepoint within a transaction to which you can later roll back.

4.3 Data Types in SQL

SQL supports various data types to store different kinds of data:

- Numeric Types: INTEGER, FLOAT, DECIMAL, etc.
- Character Types: “CHAR, VARCHAR, TEXT”, etc.
- Date and Time Types: “DATE, TIME, TIMESTAMP”, etc.
- Boolean Type: “BOOLEAN”
- Binary Types: “BINARY, VARBINARY”

4.4Data Definition Language (DDL)

“Data Definition Language” (DDL) is a subset of “SQL” used to define, modify, and manage the structure of database objects such as “tables, indexes, and schemas”. DDL commands are critical for setting up and maintaining database schema, which is blueprint of the database structure. Here are primary “DDL” commands:

1. CREATE

The “CREATE” command is used to create new database objects, such as databases, tables, indexes, and views.

- Creating a Database:

```sql
CREATE DATABASE SchoolDB;
```

- Creating a Table:

```sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
```

```
    LastName VARCHAR(50),
    BirthDate DATE,
    EnrollmentDate TIMESTAMP
);
```

- Creating an Index:

```sql
CREATE INDEX idx_lastname ON Students(LastName);
```

- Creating a View:

```sql
CREATE VIEW ActiveStudents AS
SELECT StudentID, FirstName, LastName
FROM Students
WHERE EnrollmentDate IS NOT NULL;
```

2. ALTER

The “ALTER” command is used to modify structure of existing database objects, such as tables and indexes.

- Adding a Column:

```sql
“ALTER TABLE Students ADD COLUMN Email VARCHAR(100);”
```

- Modifying a Column:

```sql
“ALTER TABLE Students MODIFY COLUMN LastName VARCHAR(100);”
```

```

**- Dropping a Column:**

```sql
ALTER TABLE Students DROP COLUMN Email;
```

- Renaming a Table:

```sql
ALTER TABLE Students RENAME TO Pupils;
```

3. DROP

The `DROP` command is used to delete existing database objects like tables, databases, or indexes. This action is irreversible.

- Dropping a Database:

```sql
DROP DATABASE SchoolDB;
```

- Dropping a Table:

```sql
DROP TABLE Students;
```

- Dropping an Index:

```sql
DROP INDEX idx_lastname ON Students;
```
```

- Dropping a View:

```sql
DROP VIEW ActiveStudents;
```

4. TRUNCATE

The `TRUNCATE` command is used “to remove all rows from a table quickly, while keeping the table structure intact”. This command is faster than `DELETE` without a `WHERE` clause because it does not generate individual row delete actions.

- Truncating a Table:

```sql
TRUNCATE TABLE Students;
```

Example Usage of DDL Commands

Below is a sequence of DDL commands that demonstrate creating, modifying, and deleting database objects:

1. Create a Database and Table:

```sql
CREATE DATABASE SchoolDB;

USE SchoolDB;

CREATE TABLE Teachers (
    TeacherID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    HireDate DATE
```

);
```

**2. Alter the Table:**

```sql
ALTER TABLE Teachers ADD COLUMN Email VARCHAR(100);

ALTER TABLE Teachers MODIFY COLUMN LastName VARCHAR(100);
```

3. Create an Index:

```sql
CREATE INDEX idx_teacher_lastname ON Teachers(LastName);
```

4. Drop the Index:

```sql
DROP INDEX idx_teacher_lastname ON Teachers;
```

5. Truncate the Table:

```sql
TRUNCATE TABLE Teachers;
```

6. Drop the Table and Database:

```sql
DROP TABLE Teachers;

DROP DATABASE SchoolDB;
```
```

4.5 Data Manipulation Language (DML)

DML commands manipulate data within tables. Examples include:

- SELECT:

```sql
SELECT  FROM Students WHERE EnrollmentDate > '2023-01-01';
```

- INSERT:

```sql
INSERT INTO Students (StudentID, FirstName, LastName, EnrollmentDate) VALUES (1, 'Alice', 'Smith', '2023-06-01');
```

- UPDATE:

```sql
UPDATE Students SET Email = 'alice.smith@example.com' WHERE StudentID = 1;
```

- DELETE:

```sql
DELETE FROM Students WHERE StudentID = 1;
```

4.6 Data Control Language (DCL)

DCL commands manage access rights and permissions:

- GRANT:

```sql
GRANT SELECT, INSERT ON Students TO user_name;
```

- REVOKE:

```sql
REVOKE SELECT, INSERT ON Students FROM user_name;
```

```

## 4.7 Table Modification Commands

These commands modify the structure of existing tables:

- **ALTER TABLE**:

```sql
ALTER TABLE Students ADD COLUMN DateOfBirth DATE;
ALTER TABLE Students DROP COLUMN Email;
ALTER TABLE Students MODIFY COLUMN LastName VARCHAR(100);
```

4.8 Summary

DDL commands are fundamental for defining and managing the structure of databases. By using `CREATE`, `ALTER`, `DROP`, and `TRUNCATE`, database administrators and developers can efficiently establish and modify the schema, ensuring robust and scalable database systems. “Data Manipulation Language” (DML) is a crucial aspect of “SQL”, focusing on the manipulation and management of data within database tables. DML commands enable users to retrieve, insert, update, and delete data, thus playing a vital role in the day-to-day operations of database management.

4.9 Self-Assessment Questions

1. Discuss data types in SQL.
2. Explain data control language instruction.
3. Define SQL
```

UNIT-5

Database Design

Learning Outcomes:

- Students will be capable of learning about the database design
- Students will be able to understand the key aspects Structured Query Language
- Students will be able to understand the ER and EER.
- Students will be able to understand about OODBMS and ORDBMS.

Structure

5.1 What is Database Design?

Database design encompasses the systematic structuring and organization of data within a database to optimize storage, retrieval, and management processes. This comprehensive process involves the meticulous definition of the database schema, which comprises tables, relationships, and constraints tailored to accommodate the data requirements and operational guidelines of an organization. By adhering to established principles and methodologies, database designers aim to create robust frameworks that facilitate seamless data operations while ensuring data integrity and consistency.

5.2. Key Aspects of Database Design

1. **Requirements Analysis**: Understanding and documenting the data needs of the users and the organization. This step involves gathering detailed information on what data needs to be stored, how it will be used, and what relationships exist between different “data elements”.
2. **Conceptual Design**: Creating an abstract model of the database, often using entity-relationship (ER) diagrams. This phase focuses on identifying the main entities, attributes, and relationships without worrying about the specifics of the “database management system (DBMS)”.
3. **Logical Design**: Translating the conceptual model into a logical model that outlines the structure of the database in a way that can be implemented in a specific DBMS. This step involves defining tables, columns, data types, and primary and foreign keys to ensure data integrity and consistency.
4. **Normalization**: Organizing the data into tables and columns to minimize redundancy and dependency. Normalization involves dividing large tables into smaller, related tables and defining relationships between them. This process typically follows normal forms, such as the first, second, and third normal forms (1NF, 2NF, 3NF).
5. **Physical Design**: Determining how the logical model will be physically implemented in the database. This includes choosing the storage structures, indexes, and access paths that will optimize performance and storage efficiency.

6. **Implementation**: Creating the actual database using a DBMS. This step involves writing SQL commands to create tables, define relationships, and establish constraints based on the logical design.
7. **Testing and Evaluation**: Ensuring that the database meets the specified requirements and performs efficiently. This involves testing the database with real or simulated data to identify any issues or bottlenecks and making necessary adjustments.
8. **Maintenance**: Continuously monitoring and updating the database to accommodate changing requirements and to ensure it operates efficiently over time. Maintenance includes updating the schema, tuning performance, and ensuring data security and integrity.

5.3. Importance of Database Design

- **Data Integrity and Accuracy**: A well-designed database ensures that data is "accurate, consistent, and reliable". By defining clear relationships and constraints, database design helps maintain data integrity and prevent anomalies.
- **Efficiency and Performance**: Proper database design optimizes the storage and retrieval of data, ensuring that the database performs efficiently even as the amount of data grows. It reduces redundancy and enhances query performance.
- **Scalability**: A thoughtfully designed database can accommodate growth and changes in data requirements without requiring significant rework. This scalability is crucial for supporting the long-term needs of an organization.
- **Security**: Database design includes defining access controls and permissions to ensure that only authorized users can access or modify the data. This protects sensitive information and maintains confidentiality.
- **Usability**: A well-structured database makes it easier for users to understand and manipulate the data. It supports the development of user-friendly applications and reporting tools that can interact with the database seamlessly.

5.4 ER and EER to Relational Mapping

Entity-Relationship (ER) and Enhanced Entity-Relationship (EER) models are conceptual tools used to design and visualize database schemas. Once an ER or EER model is created, the next

step is to map it to a relational schema, which consists of tables, columns, and relationships. This process involves transforming the entities, attributes, and relationships in “ER/EER model” into “tables, attributes, and foreign keys” in the relational schema.

5.5 Steps for Mapping ER/EER to Relational Schema

1. **Entities to Tables**:
    - Each entity in ER/EER model corresponds to a table in relational schema.
    - The attributes of entity become columns in table.
2. **Attributes to Columns**:
    - Each attribute of an entity becomes a column in corresponding table.
    - The data type of the column is determined based on the domain of the attribute.
3. **Primary Keys**:
    - Identify the primary key attribute(s) of each entity.
    - In the relational schema, designate the corresponding column(s) as the primary key(s) for the table.
4. **Relationships to Foreign Keys**:
    - For each relationship between entities in the ER/EER model, identify the foreign key(s).
    - The foreign key(s) represent the primary key(s) of the related entity in the relational schema.
5. **Cardinality Constraints**:
    - Translate the cardinality constraints (e.g., one-to-one, one-to-many, many-to-many) into foreign key constraints in the relational schema.
    - Ensure that the foreign key columns appropriately reference the primary key columns of the related tables.
6. **Associative Entities** (for EER models):
    - If the EER model includes associative entities, create separate tables for them.
    - Include foreign keys referencing the primary keys of the related entities.
7. **Inheritance Hierarchies** (for EER models):

- Implement inheritance hierarchies using one of the available strategies, such as single-table inheritance, class-table inheritance, or shared-primary-key inheritance.
- Ensure that common attributes are stored in the superclass table, while specialized attributes are stored in subclass tables.

5.6 Example Mapping

Consider an ER diagram with entities "Student" and "Course" and a relationship "Enrollment" between them. The ER diagram also includes an attribute "Grade" for the Enrollment relationship.

- **ER Diagram**:
  - Entity: Student (Attributes: StudentID, FirstName, LastName)
  - Entity: Course (Attributes: CourseID, CourseName)
  - Relationship: Enrollment (Attributes: Grade)
- **Relational Schema Mapping**:
  - Table: Student (Columns: StudentID, FirstName, LastName)
    - Primary Key: StudentID
  - Table: Course (Columns: CourseID, CourseName)
    - Primary Key: CourseID
  - Table: Enrollment (Columns: StudentID, CourseID, Grade)
    - Foreign Keys: (StudentID references Student, CourseID references Course)
    - Composite Primary Key: (StudentID, CourseID)

5.7 Functional dependencies

Functional dependencies are a fundamental concept in the field of relational databases, governing the relationships between attributes in a relation. In essence, a “functional dependency” describes the relationship between two sets of attributes in a relation, where value of one set uniquely determines value of other set.

Formally, in a relation schema RRR, a functional dependency X→YX \rightarrow YX→Y holds if, for every valid instance rrr of RRR, the values of attributes in XXX uniquely determine the values of attributes in YYY. Here, XXX and YYY are subsets of attributes in RRR.

For example, consider a relation schema Student(StuID,Name,Age)Student(StuID, Name, Age)Student(StuID,Name,Age), where StuIDStuIDStuID uniquely identifies a student. In this case, StuID→NameStuID \rightarrow NameStuID→Name and StuID→AgeStuID \rightarrow AgeStuID→Age are functional dependencies because knowing the StuIDStuIDStuID uniquely determines the values of NameNameName and AgeAgeAge respectively.

Understanding functional dependencies is crucial for database normalization, as it helps identify redundancy and ensure data integrity by breaking down relations into smaller, well-structured forms. Additionally, it forms the basis for various database design techniques, such as normalization to higher normal forms and schema refinement.

5.8 Object-Oriented Database Management Systems (OODBMS)

An "Object-Oriented Database Management System (OODBMS)" is a specialized type of database management system that expands upon relational model to accommodate complex data structures like objects, classes, and inheritance. In an OODBMS environment, data is organized and accessed in a manner akin to object-oriented programming languages such as Java or C++. This approach allows for the representation of data in the form of objects, which can encapsulate both data and behavior, thus mirroring real-world entities and their relationships more naturally.

OODBMS differs from Object-Relational Database Management Systems (ORDBMS), which also enhance traditional relational databases but typically by integrating object-oriented features into the relational model. While both OODBMS and ORDBMS aim to handle intricate data types and relationships, they do so through distinct methodologies and approaches tailored to meet specific requirements and preferences.

Key Features of OODBMS:

1. **Object Identity**: Objects have unique identities, allowing them to be directly referenced and manipulated.
2. **Encapsulation**: Objects encapsulate both data and behavior, enabling more modular and reusable code.

3. **Inheritance**: Objects can inherit properties and methods from other objects, promoting code reuse and hierarchical organization.
4. **Complex Relationships**: OODBMS supports complex relationships between objects, including “one-to-one, one-to-many, and many-to-many” associations.
5. **Query Language**: OODBMS typically includes a query language that supports object-oriented concepts, such as inheritance and polymorphism.

5.9 Object-Relational Database Management Systems (ORDBMS)

ORDBMS is a hybrid database management system that combines the relational model with object-oriented features. Unlike OODBMS, which introduces a completely new data model, ORDBMS extends the relational model to support complex data types and relationships. It provides a middle ground between the simplicity of relational databases and the flexibility of object-oriented databases.

Key Features of ORDBMS:

1. **User-Defined Types (UDTs)**: ORDBMS allows users to define custom data types, including structured types, arrays, and abstract data types.
2. **Object-Relational Mapping (ORM)**: ORDBMS supports mapping between relational tables and object-oriented constructs, enabling seamless integration with object-oriented programming languages.
3. **Inheritance and Polymorphism**: ORDBMS supports inheritance and polymorphism through features like type hierarchies and method inheritance.
4. **Complex Data Structures**: ORDBMS allows for the storage and manipulation of complex data structures, such as nested tables and nested objects.
5. **SQL Extensions**: ORDBMS extends SQL with object-oriented constructs, such as user-defined methods, constructors, and destructors.

5.10 Summary:

OODBMS and ORDBMS represent two distinct approaches to database management, each with its own set of advantages and trade-offs. OODBMS offers a more natural representation of complex data and relationships, while ORDBMS provides a compromise between relational and object-oriented paradigms. Understanding the differences between these approaches is essential

for selecting the appropriate database management system for a given application and data model.

5.11 Self- Assessment Questions:

1. What is Database Design?
2. What do you understand by ER and ERR to relation mapping?
3. What are the key features of OODBMS and ORDBMS