Aktu | Database Management System | BCS-501

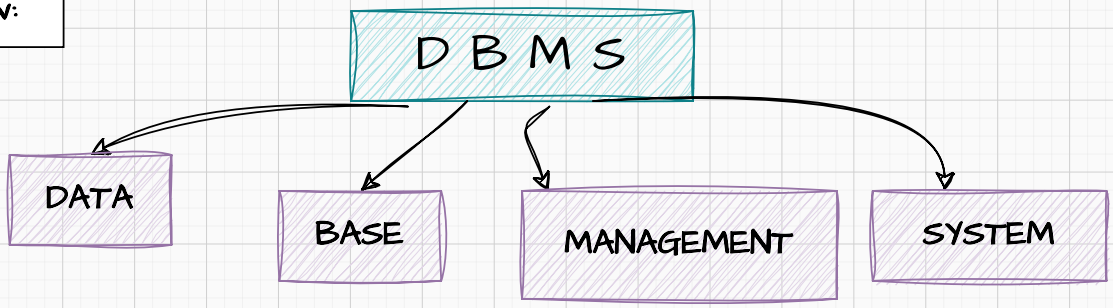
Syllabus

Introduction: Overview, Database System vs File System, Database System Concept and Architecture, Data Model Schema and Instances, Data Independence and Database Language and Interfaces, Data Definitions Language, DML, Overall Database Structure. Data Modeling Using the Entity Relationship Model: ER Model Concepts, Notation for ER Diagram, Mapping Constraints, Keys, Concepts of Super Key, Candidate Key, Primary Key, Generalization, Aggregation, Reduction of an ER Diagrams to Tables, Extended ER Model, Relationship of Higher Degree.

Topics

- Overview of DBMS
- DBMS vs. File System
- Database System Concept and Architecture (1-Tier, 2-Tier & 3-Tier)
- View of Data in DBMS
- Three-Level Architecture or Three Schema Architecture (2023-24 & 2022-23 AKTU (10Marks))
- Schema & Instances
- Data Models - (2021-22 , 2022-23 (Aktu-10Marks))
- Database Languages in DBMS
- Database Interfaces
- Overall Database Structure - (Aktu - 2021-22 & 2023-24)
- ER-Model
- Mapping Constraints
- Keys in DBMS
- DBA Roles
- Extended ER Model
- EER Features
- Reduction of an ER Diagrams to Table (2022-23 Aktu)

Overview:



DATA

It is a collection of raw, unprocessed facts, figures, or details that by themselves do not convey any specific meaning.

Example:
A list of numbers: "100, 200, 150."
Names, dates, and addresses: "John, 23rd Sept, 123 Main St."
These are just raw facts without context or interpretation.

Types of Data:

1. **Structured Data**: Organized in a defined format (e.g., tables in a database).
2. **Unstructured Data**: Unorganized and not easily interpretable (e.g., images, videos, plain text).

Information:

It is processed, organized, or interpreted data that conveys meaning or knowledge.

Examples:

- "John purchased items worth $100 on 23rd Sept at 123 Main St.

Key Idea: Information provides context and meaning to data, making it useful for decision-making or understanding.

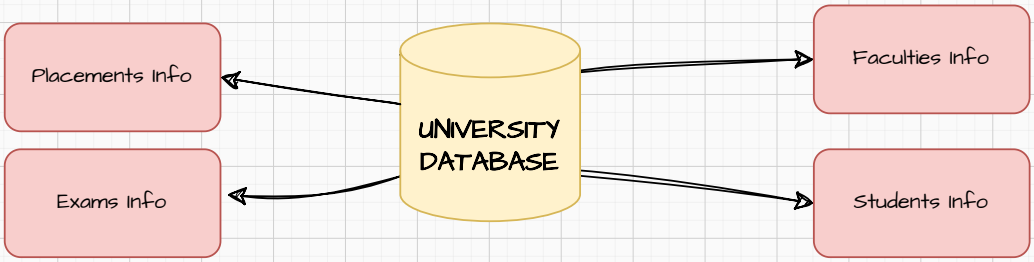
- Data is the raw input, and when processed or organized, it becomes information.

	DATA	INFORMATION
Definition	Raw, unprocessed facts and figures.	Processed or organized data that conveys meaning.
Nature	Unorganized and may not be useful on its own.	Organized, structured, and meaningful.
Example	"100, 200, 150," or "John, 23rd Sept, 123 Main St."	"John purchased items worth $100 on 23rd Sept at 123 Main St."
Representation	Simple observations, details, or statistics.	Contextualized facts that help with decision-making.
Processing	Data is the raw material that needs to be processed.	Information is the result of processing data.
Dependency	Data does not depend on information.	Information depends on data to exist.

What is Database?

A database is a **collection of interrelated data** that helps in the efficient retrieval, insertion, and deletion of data from the database and organizes the data in the form of **tables, views, schemas, reports**, etc.

For Example,
a university database organizes the data about students, faculty, admin staff, etc. which helps in the efficient retrieval, insertion, and deletion of data from it.



What is DBMS?

A Database Management System (DBMS) is a **software system** designed to create, manage, and manipulate databases. It allows users to efficiently **store, retrieve, update, and manage** data in a structured way.

- Ex:- MySQL, Oracle, etc

Key Functions of a DBMS:

1. Data Storage:

DBMS stores data in organized structures, typically in the form of tables with rows and columns. It ensures that data is stored efficiently and securely.

2. Data Retrieval:

DBMS allows users to query the database using languages like SQL (Structured Query Language) to fetch specific data quickly.

It uses indexes and optimized storage techniques to speed up the retrieval process, even when dealing with large amounts of data.

3. Data Manipulation:

With a DBMS, users can insert, update, and delete data using commands like INSERT, UPDATE, and DELETE in SQL.

It allows for batch processing and transactions, ensuring data integrity and consistency during operations.

4. Data Security and Integrity:

DBMS enforces access controls, ensuring only authorized users can view or manipulate certain data.

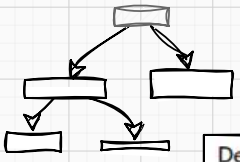
It ensures data integrity by enforcing constraints like primary keys, foreign keys, and unique constraints.

5. Backup and Recovery:

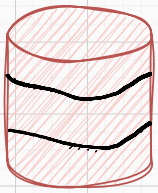
DBMS systems provide automatic backup and recovery features to protect data from accidental loss or corruption.

What is File System?

A File System is a method and structure used by an **operating system to organize, store, retrieve, and manage files** on a storage device such as a **hard drive, SSD, or USB**. It provides the foundation for storing data in files and directories, but **lacks the features** of a Database Management System (DBMS).



File System Vs DBMS



Definition	A system that manages files on a disk or storage device.	A software that manages databases, allowing storage, retrieval, and manipulation of data.
Data Storage	Stores data in files within directories.	Stores data in structured tables using rows and columns.
Data Organization	Unstructured; each file is independent.	Highly structured; data is related through tables, schemas, and relationships.
Redundancy	High redundancy due to lack of structure and relationships.	Minimizes redundancy by using relational models and enforcing constraints.
Data Integrity	No inherent mechanisms to ensure data accuracy or integrity.	Enforces data integrity through constraints like primary keys, foreign keys, and unique constraints.
Security	Basic file-level permissions (read, write, execute).	Fine-grained security controls; can restrict access based on user roles.
Concurrency Control	Minimal support for multiple users accessing files simultaneously.	Advanced concurrency control with mechanisms like locking, time-stamping, and transactions.
Data Querying	No querying capabilities; users access data by opening files.	Supports complex querying with SQL (Structured Query Language).

a **DBMS** is a more advanced system than a **File System**, especially when handling large and complex datasets that require consistency, security, and efficiency.

Data Relationships	No direct support for relationships between data files.	Supports relationships between tables via primary and foreign keys.
Data Independence	Low data independence; changes to file structure may affect applications.	High data independence; changes to database structure do not affect application code.
Backup and Recovery	Manual backup, with limited recovery options.	Automated backup and recovery mechanisms, including point-in-time recovery.
Transactions	Does not support atomic transactions.	Supports atomic transactions (ACID properties: Atomicity, Consistency, Isolation, Durability).
Performance	Slower when dealing with large amounts of data due to lack of indexing and optimization.	Optimized for performance with indexing, query optimization, and efficient storage mechanisms.
Example	File systems like FAT, NTFS, ext4 (e.g., in Windows or Linux).	DBMS like MySQL, Oracle, PostgreSQL, Microsoft SQL Server.

Database System Concept and Architecture

A database system refers to a **system that manages databases,** including the database itself and the Database Management System (DBMS). The architecture of database systems can be divided into **1-tier, 2-tier, and 3-tier architectures**, each offering different levels of abstraction and separation between users and the database.

1-Tier Architecture (Simple Architecture)

It is the **simplest architecture** of Database in which the **client, server, and Database all reside on the same machine**. A simple one tier architecture example would be anytime you install a Database in your system and access it to practice SQL queries. But such architecture is **rarely used in production.**

Advantages:

- Easy to manage.
- Suitable for local applications.

Disadvantages:

- Lacks scalability and security.
- Only suitable for small systems.

2-Tier Architecture (Client-Server Architecture)

In a 2-tier architecture, the system consists of **two layers**: the **client (frontend) and the server (backend)**. The client application **communicates directly with the database** server over a network.

Client: The user interface where users perform queries and operations (e.g., a desktop application).

Server: The database server, where the DBMS is installed and processes queries.

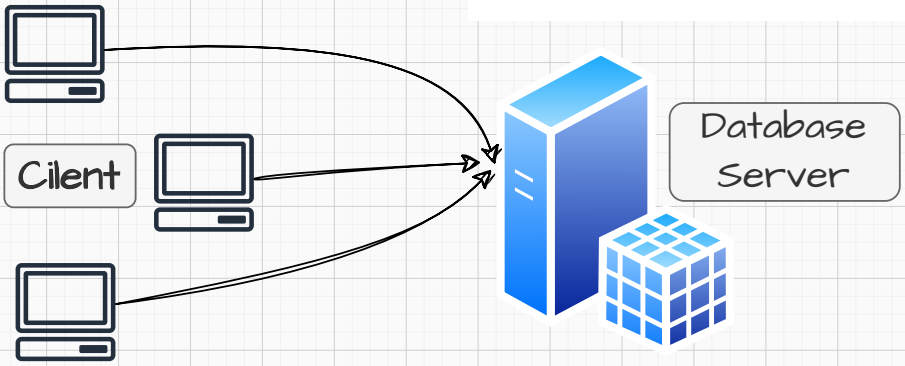
Use Case: This architecture is commonly used in desktop applications where multiple users connect to a central database server over a network.

Advantages:

-> Better performance for small to medium-sized systems.
-> Easier to manage security since the database is centrally located

Disadvantages:

-> Not very scalable for large systems.
-> Limited in handling multiple requests simultaneously, leading to performance issues.



3-Tier Architecture

Another layer exists between the client and server in the 3-Tier architecture. The **client cannot communicate directly** with the server with this design.
On the client side, **the program communicates with an application server**, which then communicates with the database system.
Beyond the application server, **the end-user has no knowledge of the database's existence**. Aside from the application, the database has no knowledge of any other users.

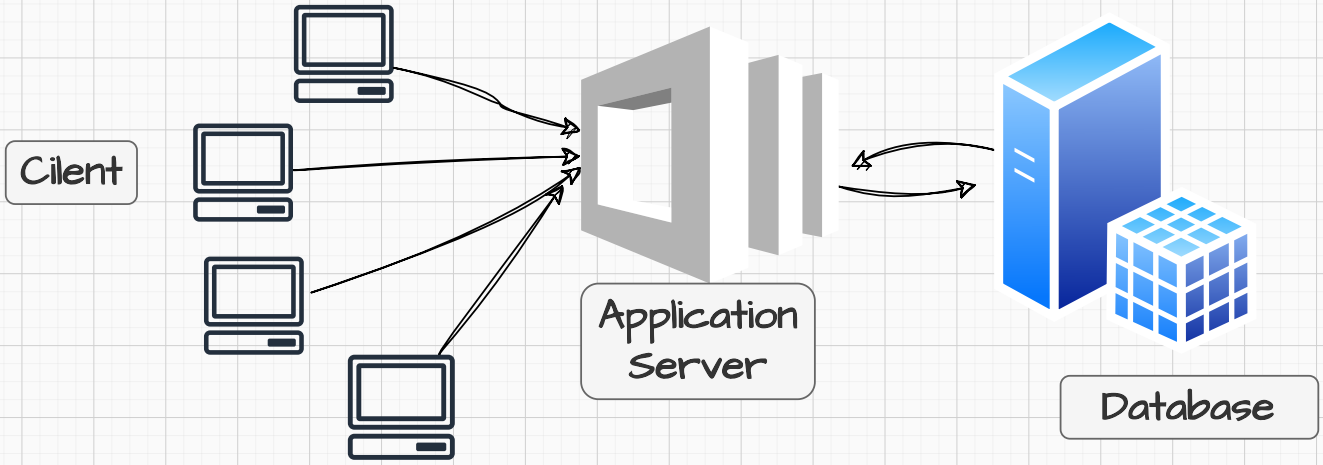
Client (UI): This is the user interface (e.g., web browser, mobile app) that users interact with.
Application Tier (Logic Layer): This is where the business logic resides. It processes user requests, communicates with the database, and returns data to the user interface.
Server: The database where data is stored and managed. The application tier communicates with the database using SQL queries.

Advantages

- **Scalable**: Easily handles large numbers of users and complex operations.
- **Security**: The application tier acts as a **middleware** that isolates the client from direct database access.
- **Maintainability**: Each layer can be updated or changed independently.

Disadvantages:

- More complex to set up and manage compared to 1-tier or 2-tier systems.
- Performance can be slower due to multiple layers of communication.



DBMS as Middleware:

In a 3-tier architecture, the DBMS functions as middleware between the application layer and the database. It provides the necessary services for querying, updating, and managing the database while abstracting the complexities of data storage from the application and the end user.

View of Data in DBMS

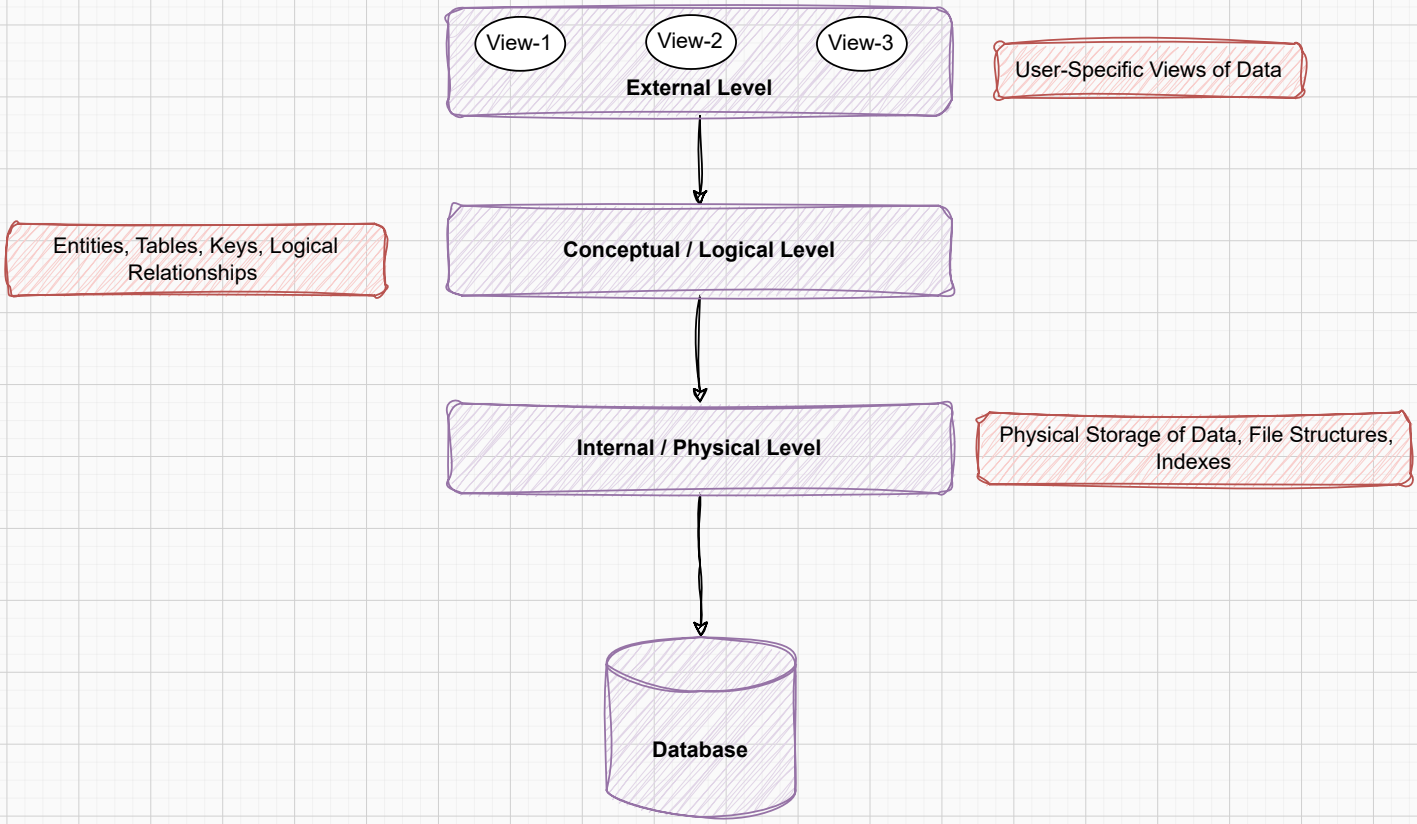
In a Database Management System (DBMS), the view of data refers to how data is presented and organized for users and applications. This view can be broken down into three levels of abstraction, which separates how data is stored from how it's viewed and accessed.

2023-24 & 2022-23 AKTU (10Marks)

Q. Discuss three level of abstractions or schemas architecture of DBMS.

Three-Level Architecture or Three Schema Architecture

The Three-Level Architecture (also known as Three-Schema Architecture) is a framework proposed by the **ANSI/SPARC** in the **1970s** for database management systems. It provides a way to separate user interactions with data from how the data is actually stored and maintained. The **three levels (schemas) of abstraction** help create flexibility, improve security, and ensure efficient data management.



1. Internal Level (Physical Schema)

Definition: The internal level represents the physical storage of data. It defines how data is stored on the storage medium, including file structures, indexes, and memory allocation strategies.
Purpose: This level focuses on the performance optimization and storage efficiency of data.
Responsibilities:

- Managing storage formats (binary, text, etc.).
- Handling the storage devices (e.g., disks, SSDs).
- Managing physical structures like indexes and block organization.

Example: A DBMS might store customer data in rows in a binary file and use indexing to make searching fast.

2. Conceptual Level (Logical Schema)

Definition: The conceptual level describes the overall structure of the database from a logical perspective. It defines the relationships, entities, attributes, and constraints.
Purpose: This level abstracts what data is stored in the database and the relationships among them.
Responsibilities:

-
- Defining entities (like Customer, Order) and relationships (e.g., each customer can place multiple orders).
- Creating tables, columns, data types, and keys (primary keys, foreign keys).
- Managing logical relationships between tables.

Example: A logical schema might define a Customers table with attributes like CustomerID, Name, and Email. It may also define the relationship between Customers and Orders.

3. External Level (View Schema)

Definition: The external level defines how the data is viewed by individual users or applications. It focuses on providing different views of the same data based on the needs of various user groups.

Purpose: This level provides security and simplicity by showing only the necessary parts of the database to different users while hiding irrelevant or sensitive data.

Responsibilities:

- Creating customized views for different user roles (e.g., a sales representative might only see customer names and sales data, while an HR person might see employee details).
- Restricting access to sensitive data by defining specific views.
- Simplifying complex database structures for end-users by hiding technical details.

Example: A SalesReport view might show only CustomerName and SalesAmount from the Customers and Orders tables, without exposing other sensitive information like customer addresses or credit card details.

Benefits of the Three-Level Architecture

1. **Data Abstraction:**
Separates the user's view from the physical structure of the database.
Provides users with different views of the same data without altering the underlying storage.

2. **Data Independence:**

- **Physical Data Independence:** Changes in the internal level (storage) don't affect the conceptual or external levels.
- **Logical Data Independence:** Changes in the conceptual level (e.g., modifying the schema) don't affect how users view the data at the external level.

3. **Improved Security:**
Users only access the part of the data they are authorized to see. This is managed through views at the external level.

4. **Flexibility:**
Enables multiple user views without impacting the logical structure of the database, allowing different departments or applications to interact with the database as needed.

Level	Description	Use Case
Physical Level	Describes how data is physically stored on storage devices.	Managed by DBAs to optimize performance and storage.
Logical Level	Describes what data is stored and the relationships between them.	Used by developers to design the schema and relationships.
View Level	Describes how data is viewed by users or specific applications.	Used to create customized views for different user groups.

Instances and Schema

In a database, instances and schemas are fundamental concepts that describe the structure and state of the database. They help define how data is stored, represented, and manipulated over time.

Schema:

Definition: A schema is the overall design or structure of the database. It defines the organization of data and the relationships between the different entities (tables) in the database.

Nature: The schema is static and usually does not change frequently. It is the blueprint of the database that outlines how the data is logically organized.

Components:

- **Tables**: Entities that store the data (e.g., Customers, Orders).
- **Columns**: Attributes of the tables (e.g., CustomerID, Name, OrderDate).
- **Relationships**: How tables relate to each other (e.g., foreign keys).
- **Constraints**: Rules applied to the data (e.g., primary keys, unique constraints, foreign keys)

Types of Schemas:

1. **Physical Schema**: Describes how data is stored physically (file systems, indexing).
2. **Logical Schema**: Describes the logical structure of the database (tables, columns, relationships).
3. **View Schema**: Describes the views that different users or applications have of the data.

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Address VARCHAR(255)
);
```

Instances:

Definition: An instance refers to the actual data in the database at a specific point in time. It is the content stored in the database, based on the schema.

Nature: Instances are dynamic and constantly change as data is inserted, updated, or deleted. The schema remains the same, but the instance evolves with time.

```
INSERT INTO Customers (CustomerID, Name, Email, Address)
VALUES (1, 'Alice', 'alice@example.com', '123 Elm St');
```

Aspect	Schema	Instance
Definition	The blueprint or structure of the database.	The actual data stored in the database.
Nature	Static (changes infrequently).	Dynamic (changes frequently).
Role	Describes how data is organized and structured.	Refers to the actual values and content.
Examples	Table definitions, column types, constraints.	Rows of data in the tables.
Changes	Typically updated when the database is modified (e.g., adding a new table).	Changes occur every time data is inserted, updated, or deleted.

Q. What are the different types of Data Models in DBMS? Explain them.

2021-22 , 2022-23 (Aktu-10Marks)

Data Models in DBMS

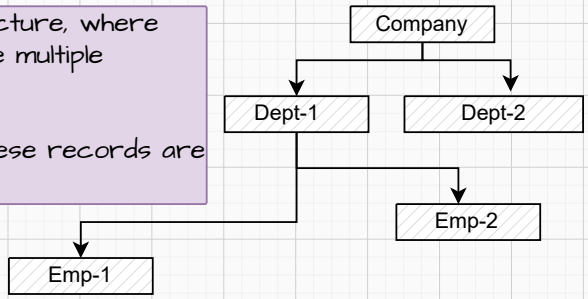
In a Database Management System (DBMS), data models define how data is stored, organized, and manipulated. They provide a structured framework for representing the relationships between data entities and offer rules for handling data operations.

There are several types of data models, each serving different purposes and providing various methods of organizing and accessing data. The most commonly used models in DBMS are:

1. Hierarchical Data Model

Definition: The hierarchical model organizes data in a tree-like structure, where each record (entity) has a single parent, and each parent can have multiple children.

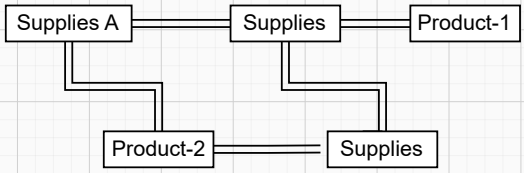
Structure: Data is represented as a collection of records, and these records are connected through parent-child relationships.



2. Network Data Model

Definition: The network model extends the hierarchical model by allowing more complex relationships, including many-to-many relationships. Data is organized as graphs, with records connected by pointers.

Structure: Data records are connected through sets, which define relationships between records.



3. Relational Data Model

Definition: The relational model organizes data into tables (relations), where each table consists of rows (records) and columns (attributes). Relationships between tables are defined through foreign keys.

Structure: Data is stored in relations (tables), and each table represents an entity or relationship. SQL is the standard query language used to interact with relational databases.

Customer Table

Customer ID	Name	Email
1	Sagar	--
2	King	--

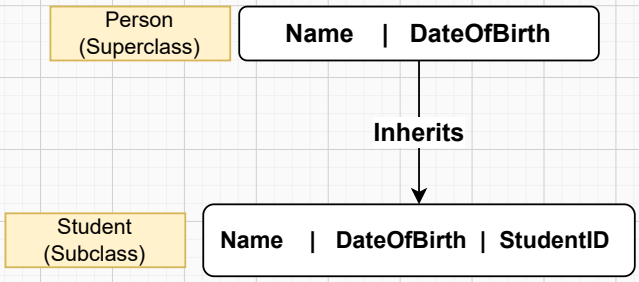
Order Table

Order ID	CustomerID	Date
101	1	--
102	2	--

4. Object-Oriented Data Model

Definition: The object-oriented model stores data as objects, similar to how object-oriented programming (OOP) languages like Java or C++ define objects. Each object contains attributes (data) and methods (behavior).

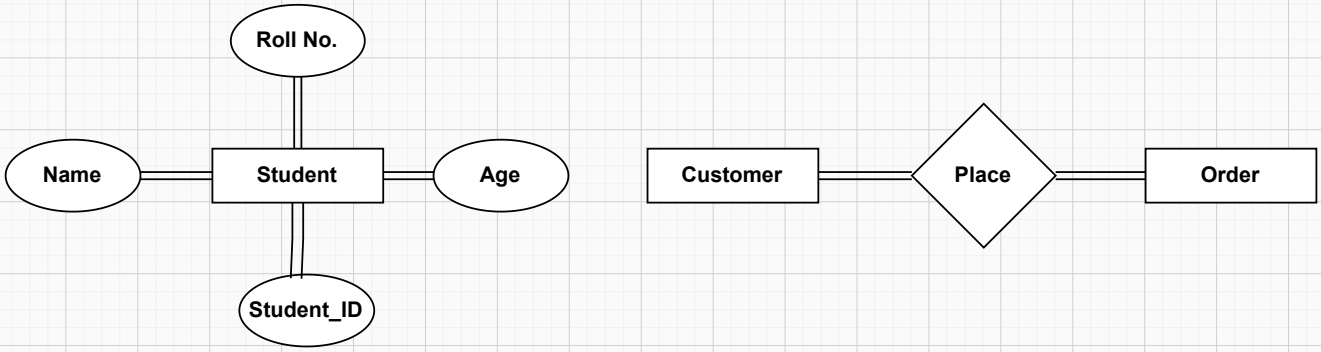
Structure: Objects are instances of classes, and classes define the properties and behaviors of those objects. Relationships are represented using inheritance and composition.



5. Entity-Relationship (ER) Model

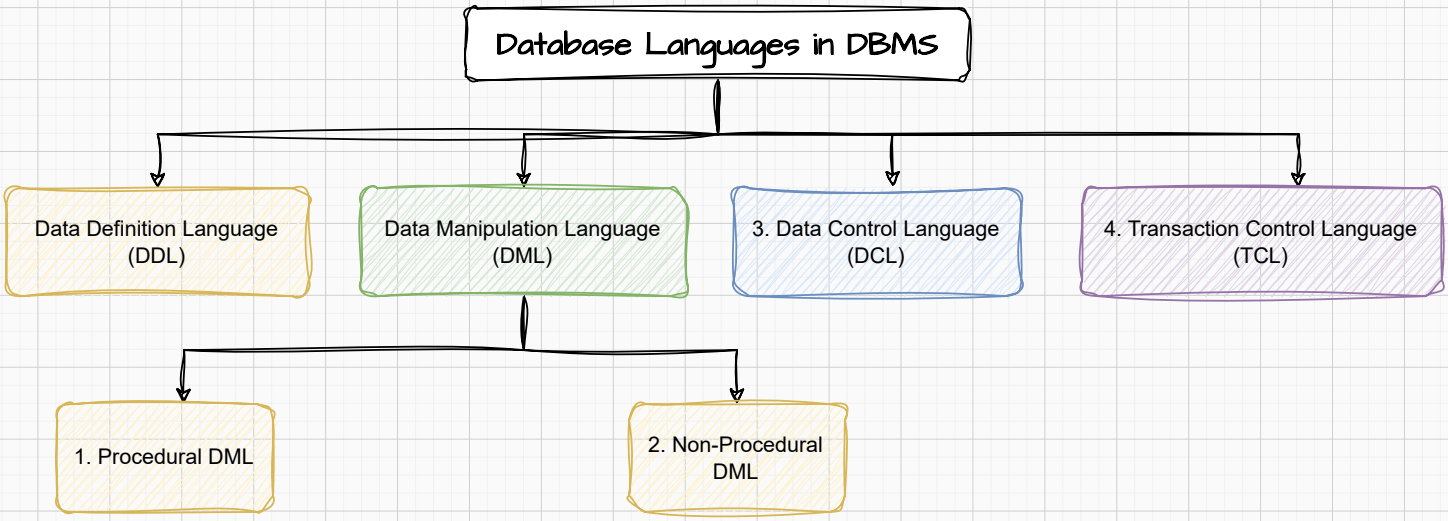
Definition: The ER model is used to visually represent the entities (objects) in the database and their relationships. It is often used during database design to model the logical structure.

Structure: Data is modeled using entities, attributes, and relationships. Entities are objects or concepts (e.g., Customer, Product), and relationships define how these entities interact with each other (e.g., a Customer places an Order).



Database Languages in DBMS

A Database Management System (DBMS) uses several languages to perform various operations like defining data structures and manipulating data. These languages form the backbone of interaction between users and the database. The most commonly used database languages are:



I. Data Definition Language (DDL)

DDL is used to define the structure of the database. It includes commands that allow users to create, modify, and delete database objects such as tables, indexes, views, and schemas.

Common DDL Commands:

- **CREATE**: Creates new database objects (e.g., tables, indexes).

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(50)
);
```

- **ALTER**: Modifies existing database objects (e.g., adding a new column to a table)

```
ALTER TABLE Employees ADD Salary DECIMAL(10, 2);
```

- **DROP**: Deletes existing objects from the database (e.g., tables, indexes).

```
DROP TABLE Employees;
```

- **TRUNCATE**: Removes all records from a table without deleting the table itself.

```
TRUNCATE TABLE Employees;
```

- **RENAME**: The RENAME command only changes the name of the object; it does not affect the data or structure of the table.

```
RENAME Employees TO Staff;
```

Purpose: DDL commands help manage the structure of the database by creating, altering, and deleting database objects.

2. Data Manipulation Language

DML (Data Manipulation Language) is a subset of SQL used to manipulate and interact with the data stored within database objects. DML commands allow you to retrieve, insert, modify, and delete data in tables.

Common DML Commands:

- **SELECT**: Retrieves data from the database.

```
SELECT * FROM Employees WHERE Department = 'IT';
```

- **INSERT**: Adds new records to a table.

```
INSERT INTO Employees (EmployeeID, Name, Department) VALUES (1, 'John Doe', 'HR');
```

- **UPDATE**: Modifies existing data in the table.

```
UPDATE Employees SET Department = 'Finance' WHERE EmployeeID = 1;
```

- **DELETE**: Removes data from a table.

```
DELETE FROM Employees WHERE EmployeeID = 1;
```

- **DML commands focus on manipulating the data inside tables, as opposed to DDL, which deals with database structure.**
- **These commands are typically part of day-to-day database operations.**
- **SELECT is the most commonly used DML command for retrieving data.**

State the procedural DML and nonprocedural DML with their differences

Aktu-(2021-22 & 2023-24) 10marks

DML can be categorized into two types based on how the data is accessed and manipulated:

1. Procedural DML:

Procedural DML requires the user to specify how to get the data. In other words, it involves describing the exact steps the system should follow to retrieve or manipulate data.

Characteristics:

- The user must specify what data is needed and how to retrieve it.
- It requires knowledge of the database structure and flow of the query.
- The user has more control over the execution process, which often involves looping or conditional operations.

Examples: Relational Algebra & PL/SQL

```
DECLARE
  totalSalary NUMBER(10);
BEGIN
  SELECT SUM(Salary) INTO totalSalary FROM Employees WHERE Department = 'IT';
  DBMS_OUTPUT.PUT_LINE('Total Salary: ' || totalSalary);
END;
```

2. Non-Procedural DML:

Non-Procedural DML requires the user to specify what data is needed, but not how to get it. The system determines the best way to execute the query to retrieve or manipulate data.

Characteristics:

- The user only specifies what data is needed, leaving the query execution process to the DBMS.
- The focus is on the result, not the method of retrieval.
- Easier to use for end-users since it does not require an understanding of the underlying data structure or query optimization.

Examples: SQL

```
SELECT SUM(Salary) FROM Employees WHERE Department = 'IT';
```

Aspect	Procedural DML	Non-Procedural DML
Definition	Specifies **how** to retrieve or manipulate data.	Specifies **what** data to retrieve, but not how.
Control	The user controls the exact steps and methods.	The DBMS controls how the query is executed.
Complexity	More complex; requires knowledge of the database structure.	Easier to use; focuses on results rather than process.
Example Languages	Relational Algebra, PL/SQL, T-SQL	SQL, Tuple Relational Calculus, Domain Relational Calculus
Flexibility	Offers more flexibility in query execution.	Less control over execution but simpler to write.
Use Case	Suitable for complex operations where specific execution details are important.	Suitable for general queries where the result is more important than the method.

3. Data Control Language (DCL)

DCL is used to control access to the database. It includes commands that manage user permissions and access controls.

Common DCL Commands:

- **GRANT**: Gives user access privileges to database objects.

```
GRANT SELECT, INSERT ON Employees TO 'user1';
```

- **REVOKE**: Removes user access privileges.

```
REVOKE INSERT ON Employees FROM 'user1';
```

4. Transaction Control Language (TCL)

TCL commands manage transactions within the database, ensuring that groups of operations are either completely executed or not executed at all.

Common TCL Commands:

- **COMMIT**: Saves all changes made during the transaction.

```
COMMIT;
```

- **ROLLBACK**: Reverts the database to its previous state before the transaction began.

```
ROLLBACK;
```

- **SAVEPOINT**: Sets a point within a transaction to which you can roll back.

```
SAVEPOINT savepoint1;
```

The database languages (DDL, DML, DCL, and TCL) are essential tools for managing and interacting with a database. Each language serves a distinct purpose, from defining and manipulating data structures to controlling user access and managing transactions.

Category	Full Form	Purpose	Key Commands
DDL	Data Definition Language	Defines or modifies database structures and schema.	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `RENAME`, `COMMENT`
DML	Data Manipulation Language	Manages and manipulates data within tables.	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `MERGE`
DCL	Data Control Language	Controls access to data and permissions.	`GRANT`, `REVOKE`
TCL	Transaction Control Language	Manages transactions in the database, ensuring data integrity.	`COMMIT`, `ROLLBACK`, `SAVEPOINT`, `SET TRANSACTION`

Interfaces in DBMS

A database management system (DBMS) interface is a user interface that allows for the ability to input queries to a database without using the query language itself. User-friendly interfaces provided by DBMS may include the following:

1. Command Line Interface (CLI):

- Uses command-line tools to execute SQL queries.
- Common for developers or administrators.
- Example: MySQL CLI, PostgreSQL psql.

2. Graphical User Interface (GUI):

- Provides a graphical interface to interact with the database.
- Often used by non-technical users for easier database operations.
- Example: phpMyAdmin, MySQL Workbench.
- Examples: Applications like Microsoft Access, Oracle SQL Developer, and MySQL Workbench.

3. Menu-Based Interfaces

- User-Friendly: Provides an intuitive point-and-click approach for easy database interaction.
- Predefined Commands: Offers a list of valid options to reduce errors during operations.
- Guided Workflow: Guides users through step-by-step processes for efficient task completion.

4. Application Programming Interface (API):

- A set of programming instructions that allow software applications to communicate with the database, enabling developers to perform database operations programmatically.
- RESTful APIs, GraphQL APIs, and database connectors/libraries in various programming languages.

5. Web-Based Interface:

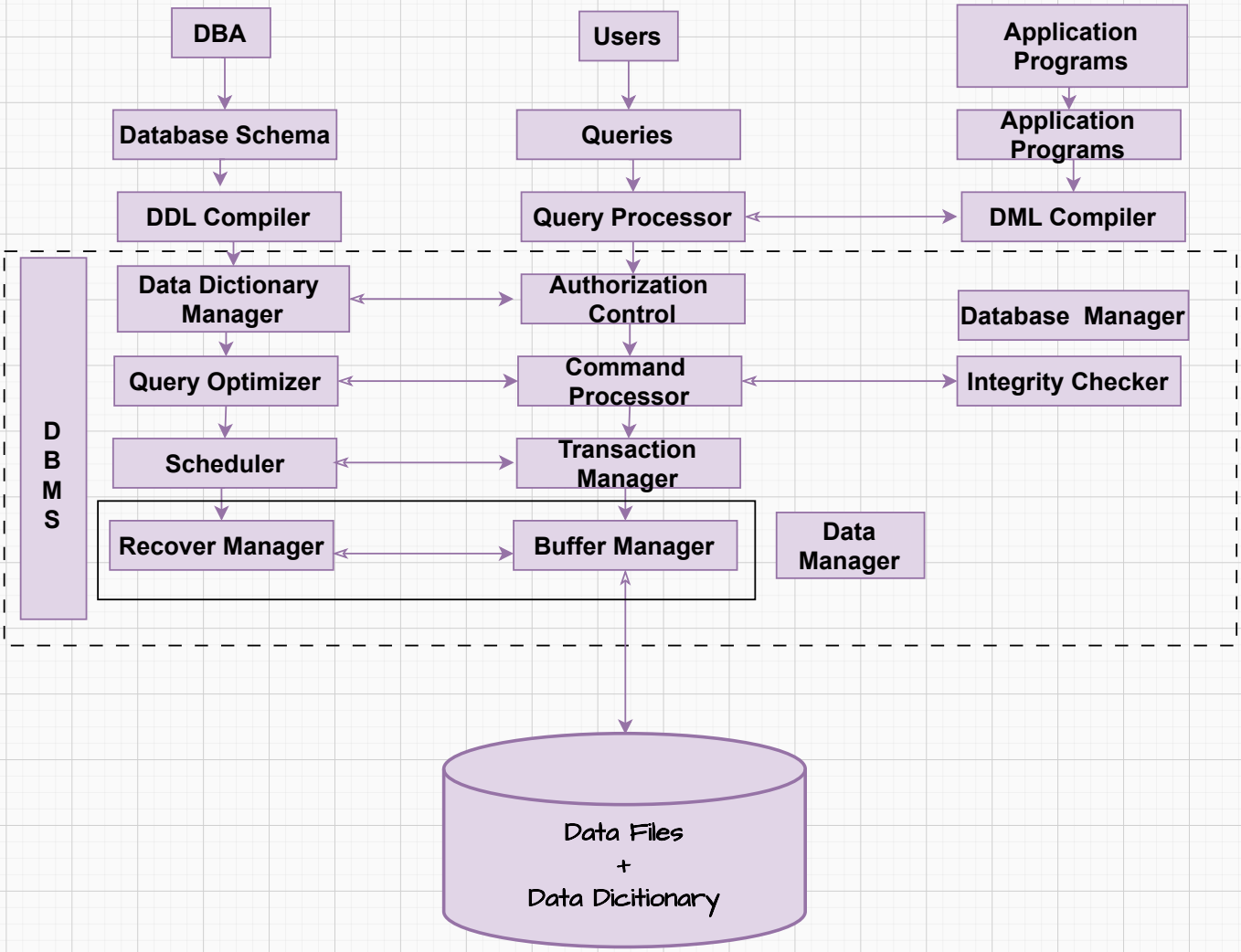
- An interface accessed through a web browser, allowing users to interact with databases via web forms and applications.
- Examples: phpMyAdmin, Adminer, and custom web applications using frameworks like Django or Flask.

A database interface plays a crucial role in enabling users and applications to efficiently interact with databases, ensuring that data can be managed and accessed effectively. The choice of interface often depends on user preferences, technical proficiency, and the specific requirements of the task at hand.

Structure of Database Management System

- Refers to the internal structure of a Database Management System (DBMS).
- It includes the components like the **Query Processor, Storage Manager, and Disk Storage**.
- Focuses on how the DBMS operates internally to manage and process data.

The Database Architecture focuses on internal DBMS components, while Tier Architecture emphasizes the structure of how a database is integrated into a broader application system.



Components of a Database System

1. Query Processor:

Role: Converts user queries into instructions that the system can execute.

Subcomponents:

- **DML Compiler:** Converts Data Manipulation Language (DML) statements into low-level machine instructions.
- **DDL Interpreter:** Converts Data Definition Language (DDL) statements into tables containing metadata.
- **Embedded DML Pre-compiler**: Transforms DML in application programs into procedural calls.
- **Query Optimizer:** Executes the instructions generated by the DML compiler, optimizing them for performance.

2. Storage Manager:

Role: Provides an interface between the queries and the physical data storage. It ensures data consistency, integrity, and proper data access control.

Subcomponents:

- **Authorization Manager:** Handles role-based access control to ensure only authorized users can access data.
- **Integrity Manager:** Ensures that integrity constraints (like primary keys and foreign keys) are enforced when data is modified.
- **Transaction Manager:** Manages transaction scheduling to maintain database consistency before and after transactions.
- **File Manager:** Handles file storage, managing file space, and structuring the data in the database.
- **Buffer Manager:** Manages cache memory and controls the transfer of data between secondary storage (e.g., hard drives) and primary memory.

3. Disk Storage:

Role: The physical layer where data is stored, organized, and managed.

Subcomponents:

- **Data Files:** Store the actual data in the database.
- **Data Dictionary:** Contains metadata, describing the structure and organization of database objects.
- **Indices:** Enable faster data retrieval by creating indexes on specific fields in the database tables.

Database Architecture deals with the internal workings of a DBMS, including query processing, storage management, and data integrity.

2021-22, 2023-24 Aktu

Role Of DBA (Database Administrator)

It is a professional responsible for managing, maintaining, and securing databases within an organization. They ensure that databases are properly designed, optimized, and safeguarded, while also managing user access, backups, and recovery processes to maintain the integrity and availability of data.

- **Database Design and Implementation**: Defining the structure of databases and implementing them to meet organizational needs.
- **Performance Monitoring**: Ensuring databases run efficiently and tuning them for optimal performance.
- **Security Management**: Implementing security measures to protect data from unauthorized access or breaches.
- **Backup and Recovery**: Ensuring regular backups and implementing recovery strategies in case of data loss.
- **Data Integrity**: Ensuring data accuracy, consistency, and preventing corruption.
- **User Management**: Managing database users, their access, and privileges.
- **Software and Hardware Management**: Installing, upgrading, and maintaining database software and hardware.

ER Model Concepts:

The **Entity-Relationship (ER) model** is a conceptual framework used to represent the data and relationships in a **database**. It helps in designing a database by defining **entities, their attributes, and the relationships** between them.

- ER diagrams represent database schema graphically and can be easily converted into relational tables.
- They model real-world objects, making it easy to visualize data structures.
- ER diagrams are easy to understand, even for non-technical users.
- They provide a standardized approach to visualizing data relationships and structure.

Components of an ER Diagram

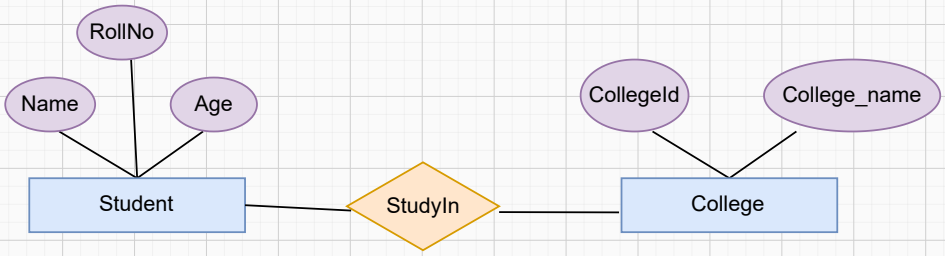
Entities: Objects or things in the real world that are distinguishable from other objects. Examples include "Student," "Course," and "Teacher."

Attributes: Characteristics or properties of an entity. For example, for a "Student" entity, attributes might include Student_ID, Name, and Age.

Relationships: Associations between entities. For example, a "Student" can enroll in a "Course," establishing a relationship between these entities.

Symbols Used in ER Model

1. **Rectangles**: Represent entities.
2. **Ellipses**: Represent attributes.
3. **Diamonds**: Represent relationships among entities.
4. **Lines**: Connect entities to attributes or relationships.



Entity Set: It is a collection of similar types of entities that share the same properties or attributes. In simpler terms, it's a group of entities that belong to the same type.

- If "Student" is an entity type, then each individual student, such as "John" or "Emily," is an entity.
- The collection of all students, such as John, Emily, and others, forms an Entity Set.

Types of Entities:

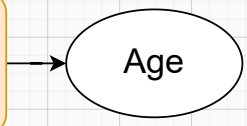
- **Strong Entity**: Can exist independently and has a primary key that uniquely identifies each instance.
- Represented by a rectangle.
- Example: Employee, Student.

- **Weak Entity**: Cannot exist independently and relies on a strong entity.
- Does not have its own primary key.
- Represented by a double rectangle.
- a ROOM can only exist in a BUILDING

Types of Attributes

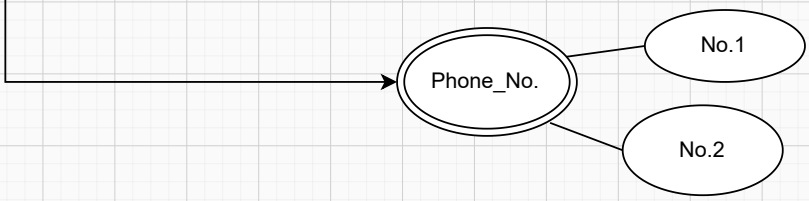
1. Single-Valued Attribute

Definition: An attribute that holds only a single value for a particular entity.
Example: Age of a person, Employee_ID of an employee.
Representation: In ER diagrams, it is represented by a single ellipse.



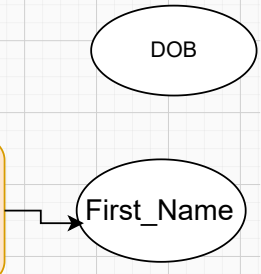
2. Multi-Valued Attribute

Definition: An attribute that can have multiple values for a particular entity.
Example: Phone_No of a person (since a person may have more than one phone no.).
Representation: In ER diagrams, it is represented by a double ellipse.



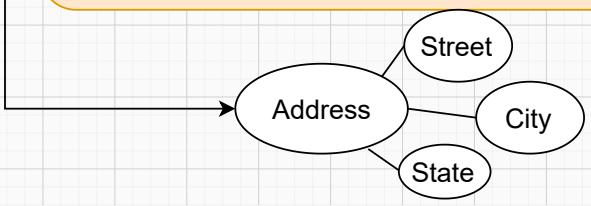
3. Simple (Atomic) Attribute

Definition: An attribute that cannot be divided further into smaller sub-attributes.
Example: First_Name, Last_Name, Date_of_Birth.
Representation: It is represented by a single oval in ER diagrams.



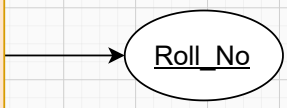
4. Composite Attribute

Definition: An attribute that can be divided into smaller sub-attributes, each representing a part of the whole attribute.
Example: Address which can be divided into Street, City, State, and Zip_Code.
Representation: In ER diagrams, it is represented by an oval with smaller ovals representing its sub-attributes.



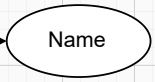
5. Key Attribute

Definition: An attribute that uniquely identifies an entity within an entity set. It serves as the primary key for an entity.
Example: Roll_No for a student, Employee_ID for an employee.
Representation: In ER diagrams, key attributes are represented by an oval with an underline.



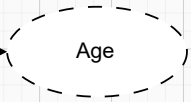
6. Non-Key Attribute

Definition: An attribute that does not uniquely identify an entity but provides additional information about the entity.
Example: Name, Address for a student (these do not uniquely identify a student but provide descriptive information).
Representation: Represented by a single oval without an underline.



7. Derived Attribute

Definition: An attribute whose value can be derived from other attributes within the entity.
Example: Age (can be derived from Date_of_Birth).
Representation: In ER diagrams, it is represented by a dashed oval.

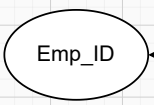


8. Stored Attribute

Definition: An attribute that holds a value stored directly in the database and is used to derive other attributes.
Example: Date_of_Birth (from which Age can be derived).
Representation: Represented by a single oval in the ER diagram (though there is no explicit differentiation from other attributes in representation).

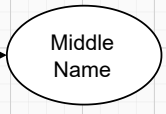
9. Required Attribute

Definition: An attribute that must have a value for each entity in the entity set.
Example: Employee_ID in the Employee entity (it must always be filled).
Representation: In ER diagrams, it is generally assumed that required attributes are present, but no specific notation differentiates them visually.



10. Optional Attribute

Definition: An attribute that may or may not have a value for a particular entity.
Example: Middle_Name for a person (not every person has a middle name).
Representation: No specific notation in ER diagrams for optional attributes, but often indicated in documentation.



Attribute Type	Description	Example	ER Diagram Representation
Single-Valued	Holds a single value for an entity.	`Age`	Single Ellipse
Multi-Valued	Holds multiple values for an entity.	`Phone_No`	Double Ellipse
Simple (Atomic)	Cannot be divided further.	`First_Name`	Single Ellipse
Composite	Can be divided into sub-attributes.	`Address`	Oval with sub-ovals
Key	Uniquely identifies an entity.	`Roll_No`	Oval with underline
Non-Key	Provides descriptive information, not unique.	`Name`, `Address`	Single Ellipse
Derived	Can be derived from other attributes.	`Age` (from DOB)	Dashed Oval
Stored	Attribute directly stored in the database.	`Date_of_Birth`	Single Ellipse
Required	Must have a value for every entity.	`Employee_ID`	Single Ellipse
Optional	May or may not have a value.	`Middle_Name`	No visual difference in ER diagram

Relationship Type and Relationship Set

Relationship Type:

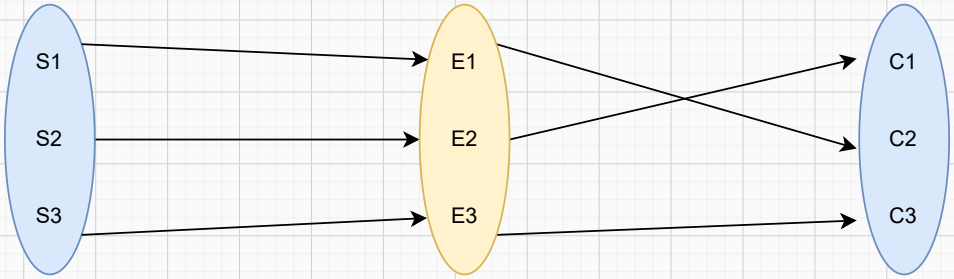
A Relationship Type represents the association or interaction between two or more entity types. It captures how entities are related to each other within a system or database. For example:

- In a university database, the relationship type "Enrolled in" can represent the association between the entity types Student and Course. This type shows that students enroll in courses.
- In an ER Diagram, a relationship type is represented by a diamond shape, with lines connecting it to the entities involved in the relationship.

Relationship Set:

A Relationship Set is a collection of the same type of relationships among entity instances. It is essentially a set of specific associations between instances of the related entity types.

- For example, in the "Enrolled in" relationship set, if S1 is enrolled in C2, S2 in C1, and S3 in C3, this forms the relationship set of the "Enrolled in" relationship type.



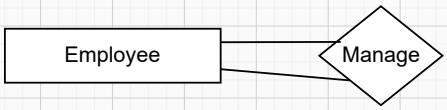
Degree of a Relationship Set

The degree of a relationship set refers to the number of entity sets involved in the relationship. It defines how many entities participate in a given relationship.

1. Unary Relationship (Degree = 1):

A unary relationship involves only one entity set. This is also known as a recursive relationship, where an entity is related to itself.

- **Example:** In an organization, an employee can supervise other employees. The relationship "Supervises" is a unary relationship, as it involves the Employee entity set relating to itself.
- **ER Diagram Representation:** A diamond labeled "Supervises" connects the Employee entity set to itself.



2. Binary Relationship (Degree = 2):

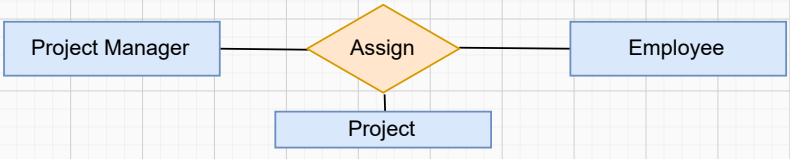
A binary relationship involves two entity sets. It is the most common type of relationship.

- **Example:** A Student enrolling in a Course. This is a binary relationship where the Student entity is associated with the Course entity.
- **ER Diagram Representation:** A diamond connecting the Student entity set with the Course entity set.

3. Ternary Relationship (Degree = 3):

A ternary relationship involves three entity sets.

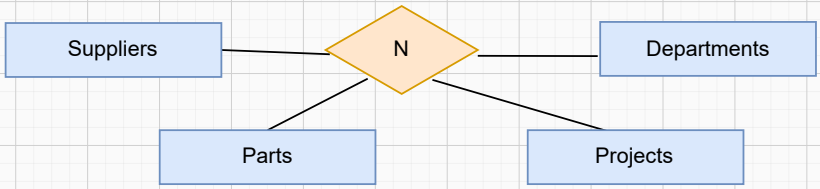
- **Example:** In a project management system, a Project Manager, an Employee, and a Project can be related through the relationship "Assigns." This connects a Project Manager to an Employee for a specific Project.
- **ER Diagram Representation:** A diamond labeled "Assigns" connects the Project Manager, Employee, and Project entity sets.



4. N-ary Relationship (Degree = n):

When there are n entity sets participating in a relationship, it is called an n-ary relationship.

- **Example:** A manufacturing process that involves Suppliers, Parts, Projects, and Departments can form a 4-ary relationship.
- **ER Diagram Representation:** A diamond connecting n entities



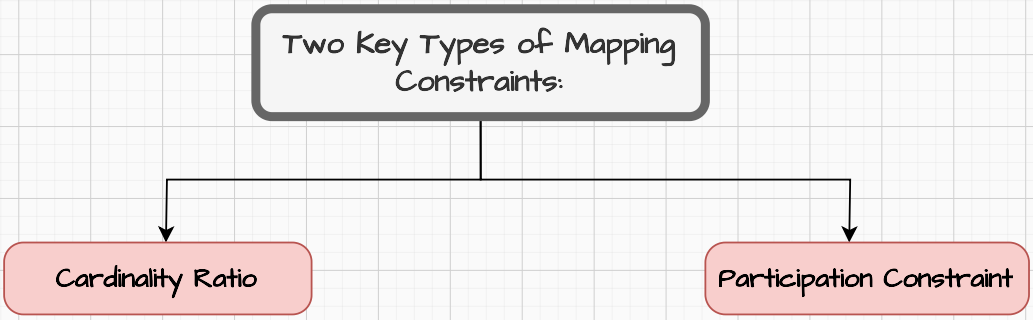
Mapping Constraints

Mapping constraints in DBMS define how many entities in one table (or entity set) can be associated with entities in another table (or entity set). They help us describe the relationship between two tables.

In real life, we have different kinds of relationships between objects or people. For example:

- One person can only have one passport.
- One teacher can teach many classes.
- Many students can enroll in many courses.

In DBMS, we describe these kinds of relationships using Mapping Constraints.



1. Cardinality Ratio (How many?)

This tells us how many entities from one set can be related to entities in another set.

Types of Cardinality Ratios:

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

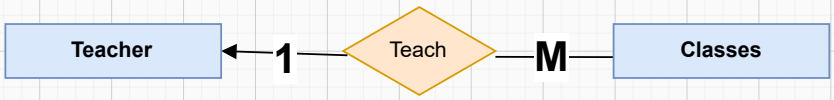
One entity in Set A can be related to only one entity in Set B, and vice versa.

- Example: A person has one passport, and a passport belongs to one person.

2. One-to-Many (1:M):

One entity in Set A can be related to many entities in Set B, but each entity in Set B is related to only one entity in Set A.

- Example: One teacher can teach many classes, but each class is taught by only one teacher.



3. Many-to-One (M:1):

Many entities in Set A are related to one entity in Set B.

- Example: Many students are assigned to one advisor, but each advisor manages many students.

4. Many-to-Many (M:N):

Many entities in Set A can be related to many entities in Set B, and vice versa.

- Example: Many students can enroll in many courses, and each course can have many students.

2. Participation Constraint (Who must participate?)

This tells us whether all entities or only some entities must participate in a relationship.

Types of Participation Constraints:

1. Total Participation:

Every entity in the entity set must participate in the relationship.

- **Example**: In a company, every employee must work in a department.
- **Diagram Representation**: Shown by a double line connecting the entity to the relationship.

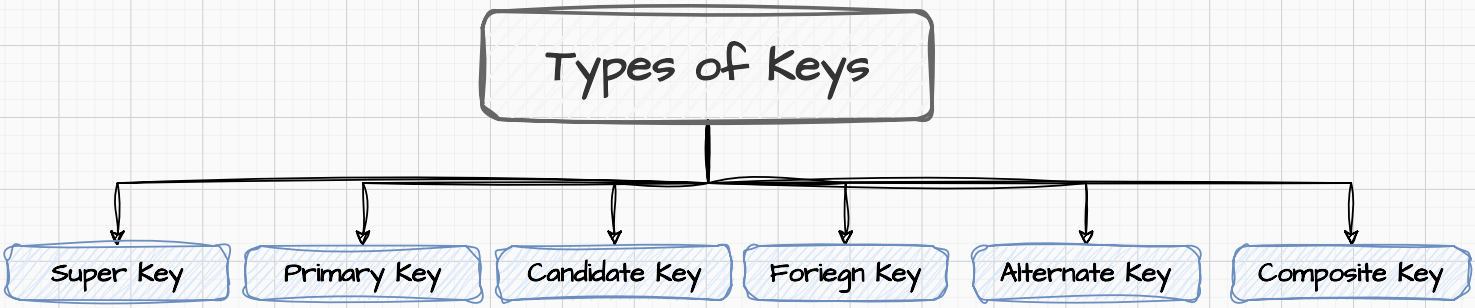
2. Partial Participation:

Some entities in the entity set participate in the relationship, while others do not.

- **Example**: Not every Employee Manages Department
- **Diagram Representation**: Shown by a single line connecting the entity to the relationship.

Keys in DBMS

Keys are **attributes or a combination of attributes** used to **uniquely identify** a tuple (row) in a relation (table). Let's explore the different types of keys:



Importance of Keys

Keys ensure data integrity, uniqueness, and efficient retrieval while establishing relationships between tables, preventing duplication, and maintaining database structure.

1. Super Key

A Super Key is a set of one or more attributes that can uniquely identify a row in a table. It can contain additional attributes that are not necessary for uniqueness.

Example: Let's say we have a Student table:

Roll_No	Name	Phone_Number	Email
101	Alice	9876543210	alice@gmail.com
102	Bob	8765432109	bob@gmail.com
103	Carol	7654321098	carol@gmail.com

Super Keys:

- {Roll_No}
- {Roll_No, Name}
- {Roll_No, Phone_Number}
- {Roll_No, Name, Email}

2. Candidate Key

A Candidate Key is a **minimal Super Key**. It's the smallest set of attributes that can uniquely identify a row.

Example: For the same Student table:

Roll_No	Name	Phone_Number	Email
101	Alice	9876543210	alice@gmail.com
102	Bob	8765432109	bob@gmail.com
103	Carol	7654321098	carol@gmail.com

Candidate Keys:

- **{Roll_No}** (since Roll_No alone uniquely identifies each student)
- **{Phone_Number}** (since every student has a unique phone number)
- {Roll__no, Phone__Number}

3. Primary Key

The Primary Key is a **special Candidate Key chosen by the database designer**. It uniquely identifies each row and cannot be null.

Example: Let's choose **Roll_No** as the Primary Key for the Student table:

Roll_No	Name	Phone_Number	Email
101	Alice	9876543210	alice@gmail.com
102	Bob	8765432109	bob@gmail.com
103	Carol	7654321098	carol@gmail.com

In this table, Roll_No is the Primary Key.

4. Alternate Key

An Alternate Key is any Candidate Key that is **not selected as the Primary Key**.

Example: In the Student table:

Roll_No	Name	Phone_Number	Email
101	Alice	9876543210	alice@gmail.com
102	Bob	8765432109	bob@gmail.com
103	Carol	7654321098	carol@gmail.com

Alternate Key:

- If Roll_No is the Primary Key, then Phone_Number could be an Alternate Key because it also uniquely identifies rows.

5. Composite Key

A Composite Key is made up of **two or more attributes that, together, uniquely identify** a row. Neither attribute alone can uniquely identify a row.

Example: Consider a **Course_Enrollment table** where students enroll in courses. A student can take multiple courses, and a course can have multiple students.

Student_ID	Course_ID	Enrollment_Date
101	CS101	2024-01-15
101	MA101	2024-02-10
102	CS101	2024-01-18

Composite Key:

- {Student_ID, Course_ID} because the combination of Student_ID and Course_ID uniquely identifies each enrollment.

6. Foreign Key

A Foreign Key is a field in one table that references the Primary Key in another table, establishing a relationship between the two.

Example: We have two tables: Students and Enrollments.

StudentID	Name	Age
1	John	21
2	Emma	22
3	Mike	20

EnrollmentID	Course	StudentID (Foreign Key)
101	Math	1
102	Science	2
103	History	1

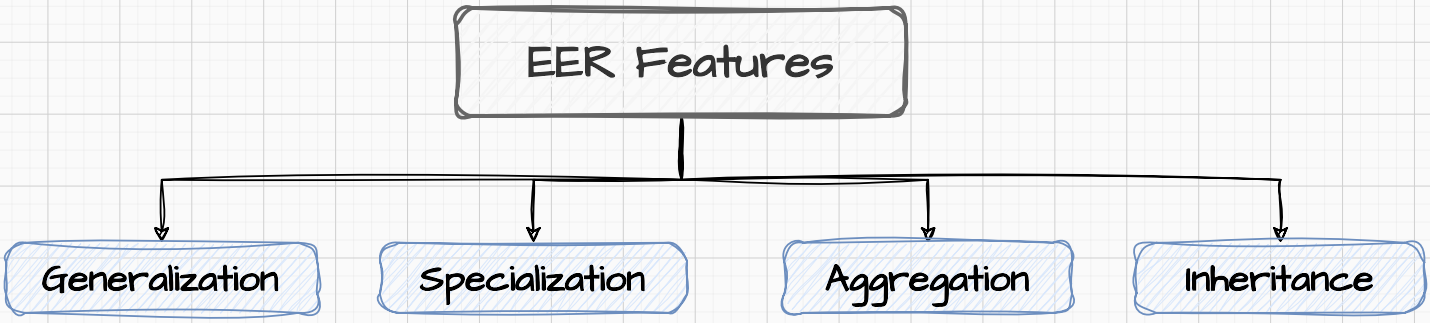
Foreign Key:

- StudentID in the Enrollments table is the Foreign Key.
- It references the StudentID in the Students table, meaning each entry in the Enrollments table must correspond to an existing StudentID in the Students table.

Extended ER Diagram (EER)

Today the complexity of the data is increasing so it becomes more and more difficult to use the traditional ER model for database modeling. To reduce this complexity of modeling we have to make improvements or enhancements to the existing ER model to make it able to handle the complex application in a better way.

The Extended ER Diagram (EER) is an enhancement of the basic Entity-Relationship (ER) model to better represent complex real-world scenarios. It introduces several new concepts and features, which are useful in modeling advanced database requirements. Here are the main features of the EER diagram:

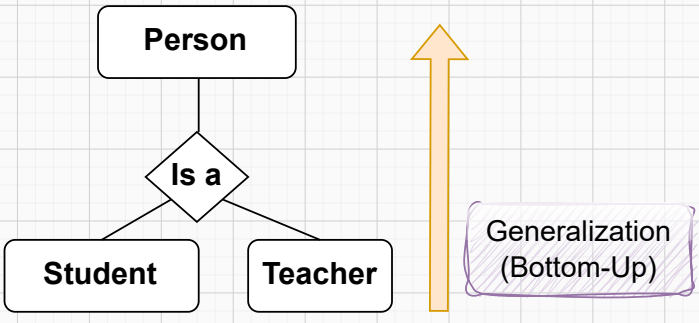


1. Generalization:

It is the process of **combining multiple specific entities into a more general**, higher-level entity. It identifies common features shared by multiple entities and groups them into a single generalized entity to avoid redundancy.

Example: Consider the entities Student and Teacher. Both can be generalized into a higher-level entity called Person because both share common attributes like Name, Address, and Date of Birth.

Why it matters: It simplifies the data model by reducing duplication of shared characteristics across multiple entities.

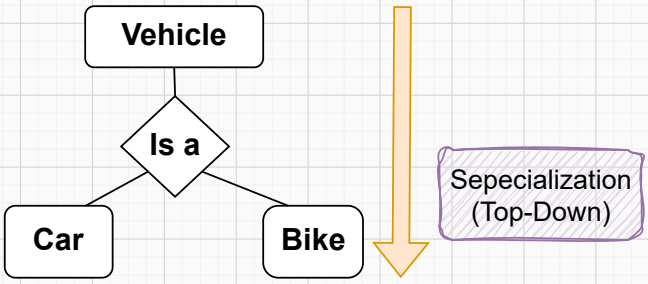


2. Specialization:

It is the **opposite of generalization**. It breaks down a general entity into more specific sub-entities. This allows us to represent entities with additional specific attributes and behaviors that differentiate them from the general entity.

Example: Starting with the general entity Vehicle, we can specialize it into Car and Bike. Both share general attributes like Brand and Model, but Car might have additional attributes like Number of Doors, and Bike might have Type of Handlebar.

Why it matters: It helps in representing more detailed and specific information about certain entities.

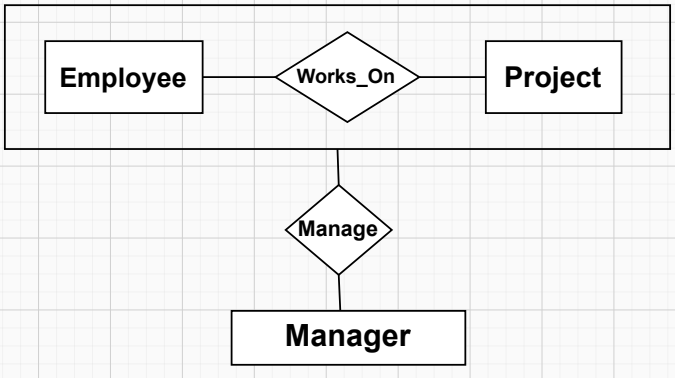


3. Aggregation:

Is used to **represent a relationship as an entity**. It allows you to treat a **relationship between entities as a higher-level entity**, which can then be related to other entities. This is useful when you have complex relationships that involve multiple entities, and you want to simplify or modularize the ER diagram.

Example: Suppose we have entities Employee and Project with a relationship Works_On. Now, if Manager is supervising the relationship between Employee and Project, we can aggregate the Works_On relationship and treat it as a single entity that Manager supervises.

Why it matters: It avoids repeating common information and ensures consistency in the model by allowing shared attributes to be reused across different entities.

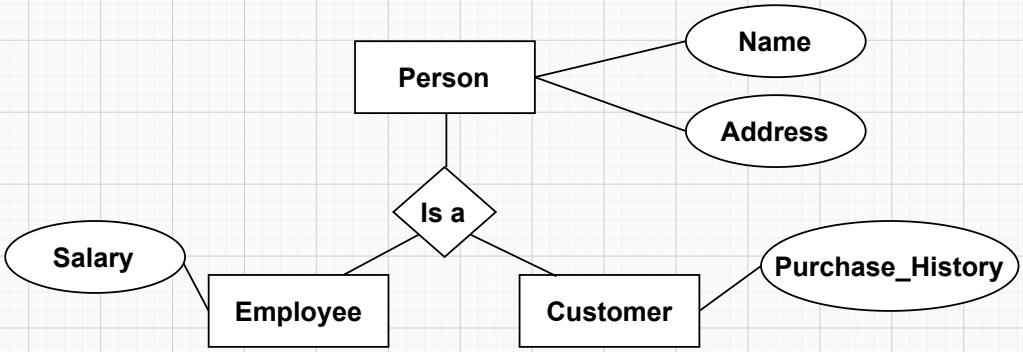


4. Inheritance:

It is a concept borrowed from object-oriented programming and refers to how a specialized entity inherits attributes and relationships from a generalized (parent) entity. The specialized entity can also have its own unique attributes.

Example: If we have a general entity Person, specialized entities like Employee and Customer can inherit common attributes like Name and Address from Person, while also having unique attributes like Salary (for Employee) and Purchase History (for Customer).

Why it matters: It simplifies complex relationships by grouping them and making them manageable.



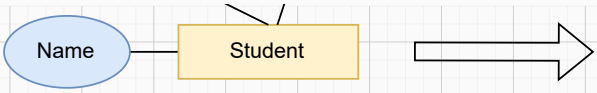
The **reduction of an ER diagram to tables** (also known as mapping an ER diagram to a relational schema) is a process that involves **converting the entities, relationships, and attributes from an ER diagram into tables** in a relational database.

This process ensures that all the information captured in the ER diagram is **properly stored in the database.**

I. Mapping Entities to Tables

- Each entity in the ER diagram becomes a table in the relational database.
- The attributes of the entity become the columns of the table.
- The primary key of the entity becomes the primary key of the table.

Example: If we have an entity **Student with attributes Student_ID, Name, and Age**, it is mapped to a table as follows:

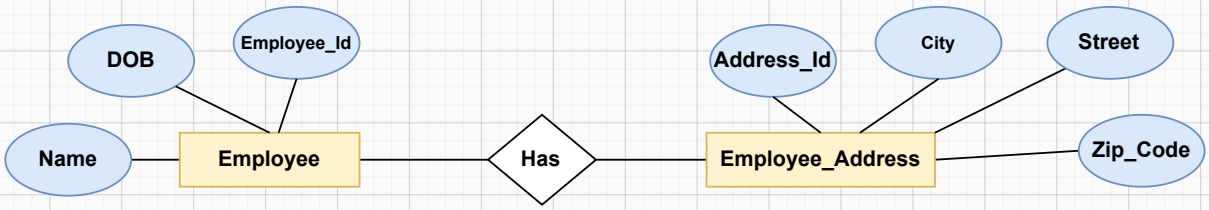


Student_ID (PK)	Name	Age
101	John	20
102	Alice	22

2. Mapping Relationships to Tables

- **One-to-One Relationship**: Can be merged into one table by combining the entities or by placing the foreign key in either of the tables.
- **One-to-Many Relationship**: The foreign key is added to the "many" side of the relationship.
- **Many-to-Many Relationship**: A new table is created with foreign keys from both entities to represent the relationship.

one-to-one relationship



Option 1: Merging the two entities into one table

Employee_ID (PK)	Name	DOB	Address_ID (PK)	Street	City	Zip_Code
101	John	1990-01-01	A101	Oak St	New York	10001
102	Alice	1992-05-12	A102	Pine St	Boston	02115

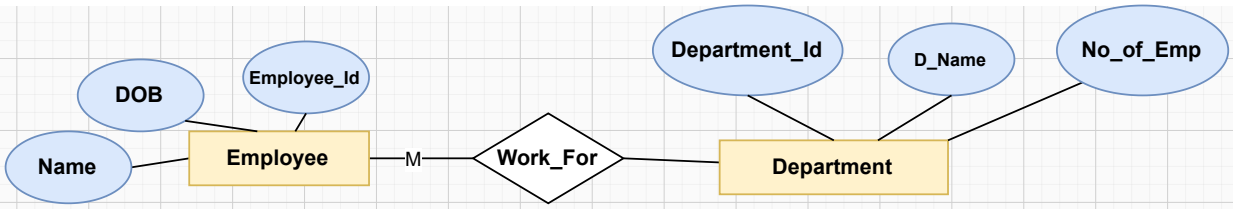
Option 2: Adding a foreign key in one of the tables

Address_ID (PK)	Employee_ID (FK)	Street	City	Zip_Code

A101	101	Oak St	New York	10001
A102	102	Pine St	Boston	02115

one-to-many relationship

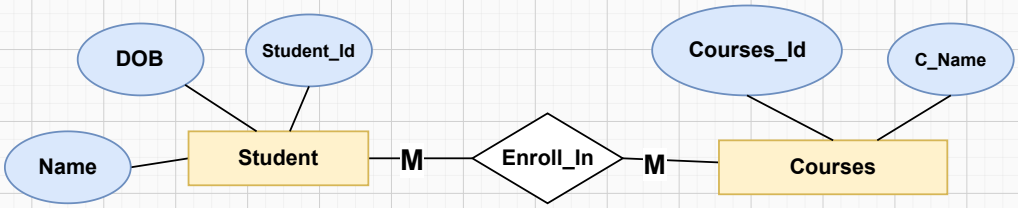
Example: A One-to-Many relationship between **Department** and **Employee** is mapped by adding a foreign key **Department_ID** in the Employee table.



Employee_ID (PK)	Name	Department_ID (FK)
1	John	101
2	Alice	102

Many-to-Many Relationship

Example: A new table **Student_Course** is created to represent the relationship, with **foreign keys from both Student and Course.**



Student_ID (FK)	Course_ID (FK)
101	CSE101
102	CSE101
101	MATH101

3. Mapping Attributes

- **Simple Attributes**: These are directly converted into columns of the corresponding table.
- **Composite Attributes**: Each sub-attribute of a composite attribute is treated as a column in the table.
- **Multi-valued Attributes**: A separate table is created with a foreign key from the original entity and the multi-valued attribute as columns.

Example: If a Student has multiple Phone Numbers, create a new table Student_Phone:

Student_ID (FK)	Phone_Number
101	1234567890
101	9876543210

4. Mapping Weak Entities

- A weak entity is represented as a table, but its primary key is a composite key consisting of its own attributes and the primary key of the related strong entity.

Aktu - 2022-23

(a) A database is being constructed to keep track of the teams and games of a sport league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of players participating in each game for each team, the positions they play in that game and the result of the game.
(i) Design an E-R schema diagram for this application.
(ii) Map the E-R diagram into relational model.

For a database that tracks the **teams, players, games, and participation details in a sports league**, we need to design an ER diagram and map it to a relational model. Let's break this down step by step.

(i) Design an E-R Schema Diagram

Key Entities and Relationships:

1. **Team**: Represents the different teams in the league.
2. **Player**: Represents the players who are part of the teams.
3. **Game**: Represents the games played between teams.
4. **Participation**: Represents the participation of players in specific games, along with their positions.

Attributes

1. **Team**:

- Team_ID (Primary Key)
- Team_Name

2. **Player**:

- Player_ID (Primary Key)
- Player_Name
- Position
- Team_ID (Foreign Key from Team)

3. **Game**:

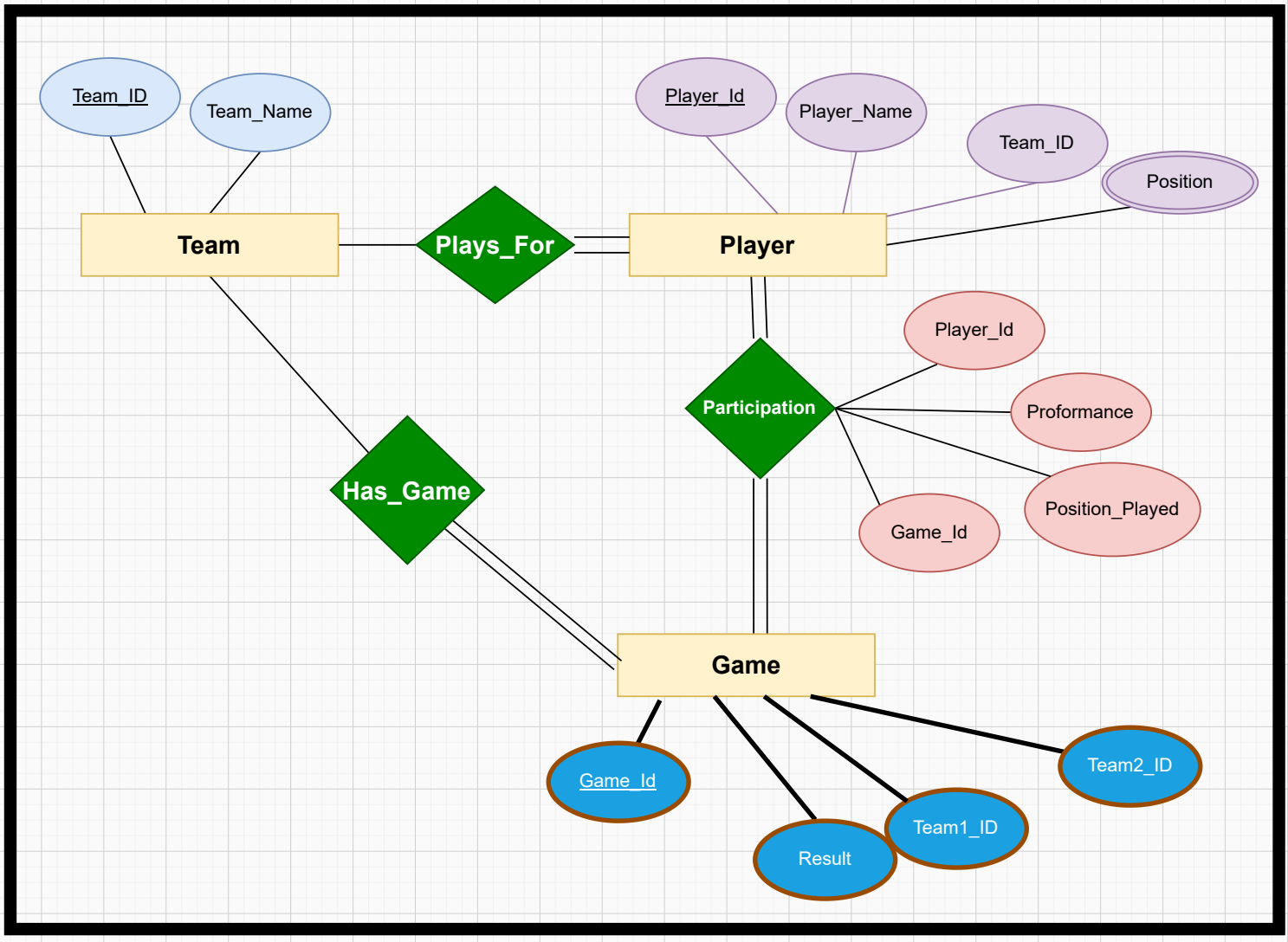
- Game_ID (Primary Key)
- Date
- Team1_ID (Foreign Key from Team)
- Team2_ID (Foreign Key from Team)
- Result (Stores the result of the game)

4. **Participation**:

- Game_ID (Foreign Key from Game)
- Player_ID (Foreign Key from Player)
- Position_Played (Position the player played in the game)
- Performance (Optional: Could store some stats or rating about their performance)

Relationships:

- **Plays_For**: Between Team and Player (one-to-many relationship).
- **Plays**: Between Game and Player (many-to-many relationship, since multiple players participate in a game).
- **Has_Game**: Between Team and Game (a team can participate in multiple games).



(ii) Map the E-R Diagram to a Relational Model

We will convert the entities and relationships from the ER diagram into tables.

1. Team Table:Team(Team_ID , Team_Name)

2. Player Table: (Player_ID , Player_Name ,Position ,Team_ID)

3. Game Table: (Game_ID, Date, Team1_ID, Team2_ID, Result)

4. Participation Table: (Game_ID, Player_ID, Position_Played, Performance)

1. Team Table:Team(Team_ID , Team_Name)

Team_ID (PK)	Team_Name
1	Team A
2	Team B

2. Player Table: (Player_ID , Player_Name ,Position ,Team_ID)

Player_ID (PK)	Player_Name	Position	Team_ID (FK)
101	John	Forward	1
102	Mike	Goalkeeper	1
103	David	Forward	2

3. Game Table: (Game_ID, Date, Team1_ID, Team2_ID, Result)

Game_ID (PK)	Date	Team1_ID (FK)	Team2_ID (FK)	Result
201	2024-10-12	1	2	2-1

4. Participation Table: (Game_ID, Player_ID, Position_Played, Performance)

Game_ID (FK)	Player_ID (FK)	Position_Played	Performance
201	101	Forward	9/10
201	102	Goalkeeper	8/10
201	103	Forward	7/10