DBMS

Syllabus

- Unit I: Introduction to DBMS: Introduction to Database Systems, Need and Limitations of DBMS, Users of DBMS, ANSI Three-Schema Architecture of database, Data Independence, Mapping of Conceptual-External Schema and Conceptual-Internal Schema.

- Unit 2: Data Models : Introduction to Entity-Relation Model. Basic Terminology, related to ER Model, Notations Used in ER Diagram, Generalization, Specialization and Aggregation. Solving Simple Case studies to draw ER, Diagram. Structure of Relational Model, Types of Keys, Referential Integrity, Codd"s rules, Relational Algebra Operators: Selection, Projection, Rename, Set Difference, Union, Intersection, Cross Product, Types of Joins, SQL Representation of Relational operators, Converting ER model to Relational Model.
- Unit 3: Data Normalization: Concept of Normalization, Functional Dependency, First Normal Form, Second Normal Form, Third Normal Form, and Solving Case Study based problems (Design ERD, Database Schema and Normalization up to 3NF).

- Unit 4: Transaction and Concurrency Control :Concepts of transaction processing, ACID Properties, States of Transaction, Concurrency Control, Problems in Concurrency Control, Serial Schedule and Serializability, Test Serializability, Deadlock, Locking & Time stamp Methods for Concurrency Control.

- Unit 5: Crash Recovery and Backup : Database Recovery Concept, Types of failures, and Types of database recovery: REDO & UNDO, Database Recovery Techniques: Deferred Update, Immediate Update, Shadow paging and Checkpoint. Database Security
- Database Security Issues, Discretionary Access Control Based on Grant &imp; Revoking Privilege, Mandatory Access Control and Role Based Access Control for multilevel security.

DBMS:- Introduction, Purpose and Importance

- ***A database management system is a computerized solution that helps store information in a manner that is easy to read, edit, delete, and scale, with the primary objective of drawing correlations, powering analysis, and supporting data-driven workflows.***

- A DBMS is software that allows users to create, store, and manage databases efficiently, acting as an interface between users, applications, and the data itself.

- It ensures data consistency, security, and integrity, and provides features for data manipulation, retrieval, backup, and recovery.

- Database Management Systems (DBMS) allowing users to create, read, update, and delete data in the database.

- A database management system (or DBMS) is a computerized data-keeping system. System users can carry out various actions on such a system to manipulate the data and manage the database structure.

Cont…

- The DBMS offers a centralized view of the data that may be accessed in a regulated manner by several users from numerous places.

Key features of DBMS

- **Data Organization:** Stores and organizes data in a structured format, often in tables or other structures.
- **Data Manipulation:** Allows users to insert, update, delete, and retrieve data.
- **Data Security:** Protects the database from unauthorized access.
- **Data Integrity:** Ensures the accuracy and consistency of the data.
- **Data Consistency:** Prevents conflicting data from being stored.
- **Concurrency Control:** Manages simultaneous access to the database by multiple users.
- **Backup and Recovery:** Provides mechanisms for backing up data and recovering it in case of failure.

Purpose of DBMS

- **Reduced Redundancy:** Centralized control minimizes data duplication.
- **Improved Efficiency:** Efficient storage, management, and retrieval of data.
- **Scalability:** Supports growing data and user demands.
- **Centralized Control:** Provides a single point for managing and accessing data.
- **Data Administration**
- **Crash Recovery**

Advantage of DBMS

- The advantages of using a Database Management System includes:
  - Enhanced data security,
  - Reduced data redundancy and inconsistency,
  - Improved data integrity and accuracy,
  - Efficient data sharing and access,
  - Reliable backup and recovery, and
  - Better data integration and decision-making.
  - A DBMS provides a centralized, structured environment for managing data, making it more accessible, secure, and reliable than traditional file-based

Limitation of Data base

- **High Cost**: The initial setup requires substantial investment in expensive hardware, software licenses, and skilled personnel (like Database Administrators), as well as ongoing maintenance.
- **Complexity**: Designing, implementing, and managing a DBMS is a complex process that demands specialized technical knowledge and skills.
- **Performance Overhead**: The features and general-purpose nature of DBMS can sometimes lead to performance bottlenecks, with slower response times for simple tasks.
- **Single Point of Failure**: If the central database fails, the entire system can become inoperable, leading to significant business disruption.
- **Security Risks**: Centralizing data makes the database a prime target for security breaches; a successful attack can compromise all data stored within.
- **Not Cost-Effective for Small Organizations**: The extensive resources and complexity required for a DBMS can be an unnecessary burden for small businesses or simple applications.
- **Vendor Lock-in**: Switching to a different DBMS can be difficult due to differences in data formats and proprietary query languages, creating dependency on a specific vendor.
- **Requires Regular Maintenance**: DBMS need frequent upgrades, patches, and maintenance to ensure optimal performance and security, which adds to the cost and complexity

Need of Database

- **Centralized Data Control**: A DBMS provides a single, central repository for all organizational data, leading to better management and consistency.
- **Improved Data Quality**: By enforcing data integrity rules and minimizing redundancy, A DBMS ensures data is accurate, consistent, and reliable.
- **Enhanced Data Security**: DBMS offers robust security features, protecting sensitive data from unauthorized access and cyber threats.
- **Efficient Data Access and Retrieval**: Users can access and query data efficiently using standardized languages like SQL, facilitating better decision-making.
- **Data Integration and Sharing**: A DBMS provides a unified view of data across an organization, allowing different departments to share and access information seamlessly.
- **Concurrency Control**: It allows multiple users to access and modify the same data simultaneously without causing conflicts, maintaining data integrity.
- **Disaster Recovery and Backups**: DBMS includes mechanisms for data backup and recovery, protecting data from loss due to hardware failures or other disasters.

Users of DBMS

- **Database Administrators (DBA):**
  - Responsible for the overall management and administration of the database system.
  - Tasks include schema definition, data access authorization, security management, performance monitoring, backup and recovery, and liaison with other users.
- **Database Designers:**
  - Responsible for identifying the data to be stored in the database and choosing appropriate structures to represent and store that data.
  - Focus on the logical and physical design of database, including defining tables, relationships and data types.
- **Application Programmers:**
  - Develop application program that interact with the database.
  - Write code in various programming language to create interfaces and functionalities for end-users to access and manipulate data.

Cont...

- **System Analyst:**
  - Analyze the requirements of the end-users and design the data system and application to meet those requirement.
  - Bridge the gap between user need and technical implementation.

- **End Users:** Individuals who directly interact with the database.
  - **Naïve/parametric users**: use pre-built application and interfaces without knowledge of the underlying database structure or Query language. Bank tellers or online shoppers.
  - **Sophisticated users**: interact with database directly using SQL to perform complex data analysis and retrieval. "Data Analyst, Researcher.
  - **Casual users**: access database occasionally for specific information or task.
  - **Specialized users:** Develop specialized database application that cater to specific need, often for scientific or engineering purpose.

Application of DBMS

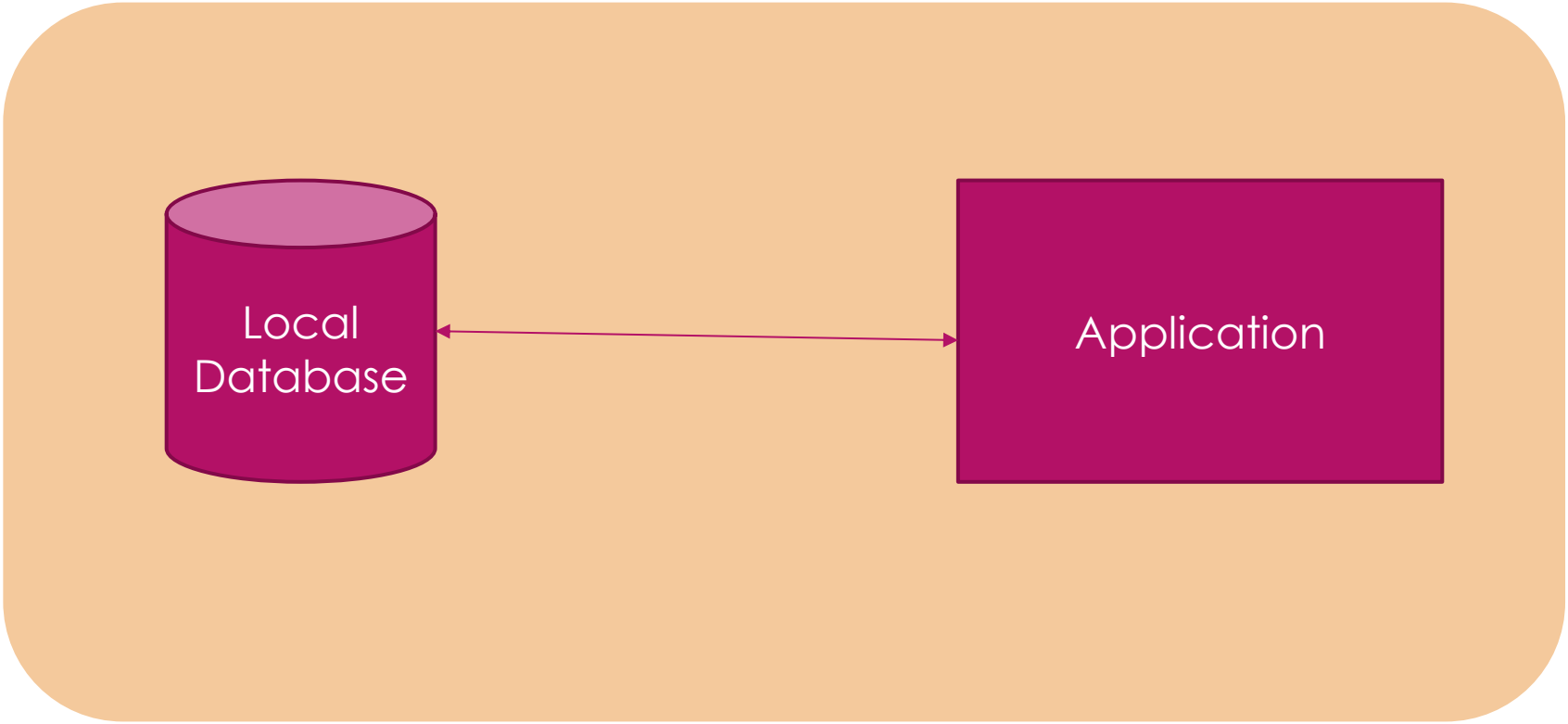
- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

Architect of DBMS

- The **architecture of a Database Management System (DBMS)** refers to the design and structure of the system that manages databases. It defines how data is stored, accessed, and managed, and how different components of the system interact with each other.

- A well-designed architecture and schema (a blueprint detailing tables, fields and relationships) ensure data consistency, improve performance and keep data secure.

One tier architecture

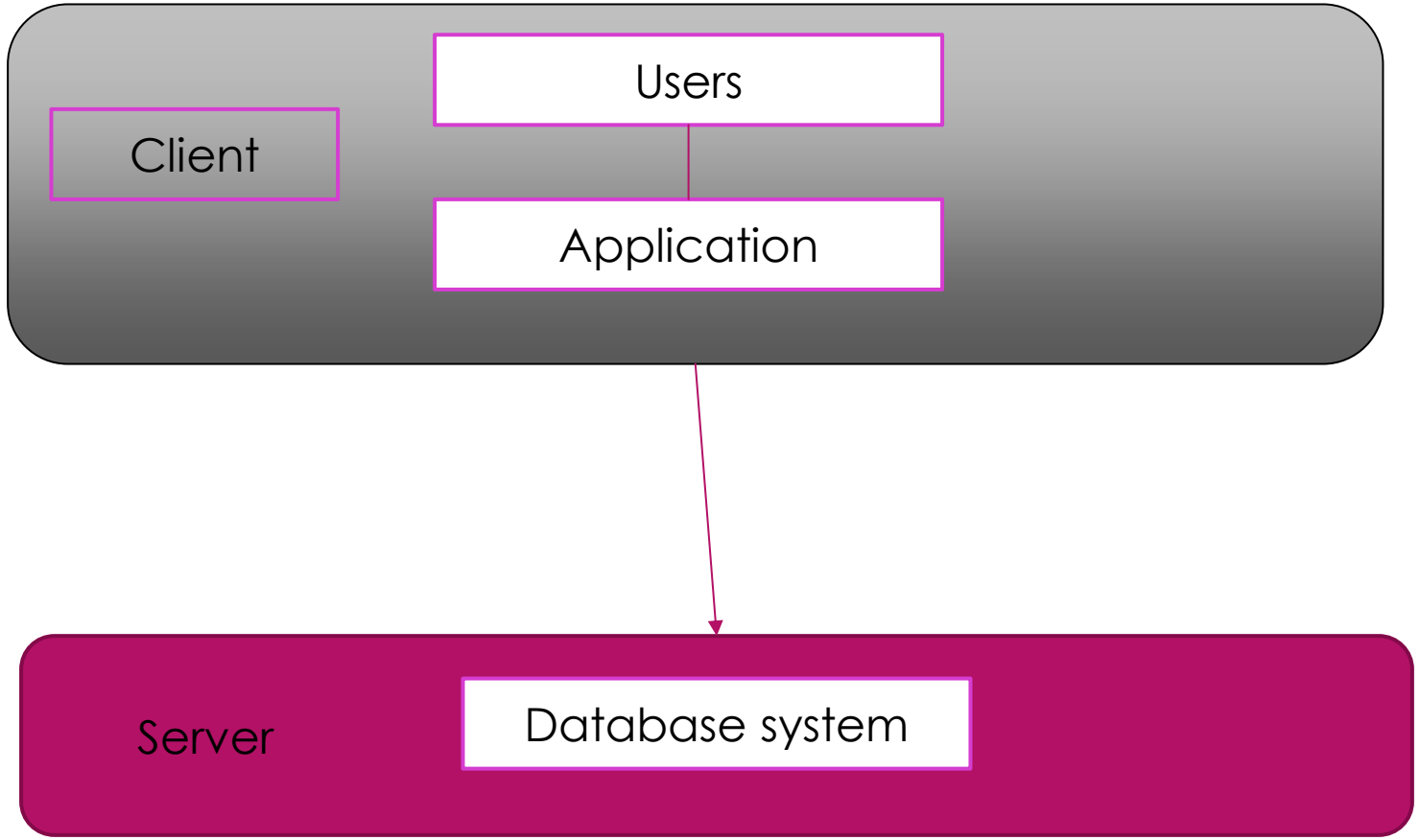


This means the client, server and database are all in one application. The user can open the application, interact with the data and perform tasks without needing a separate server or network connection.

Two Tier Architecture

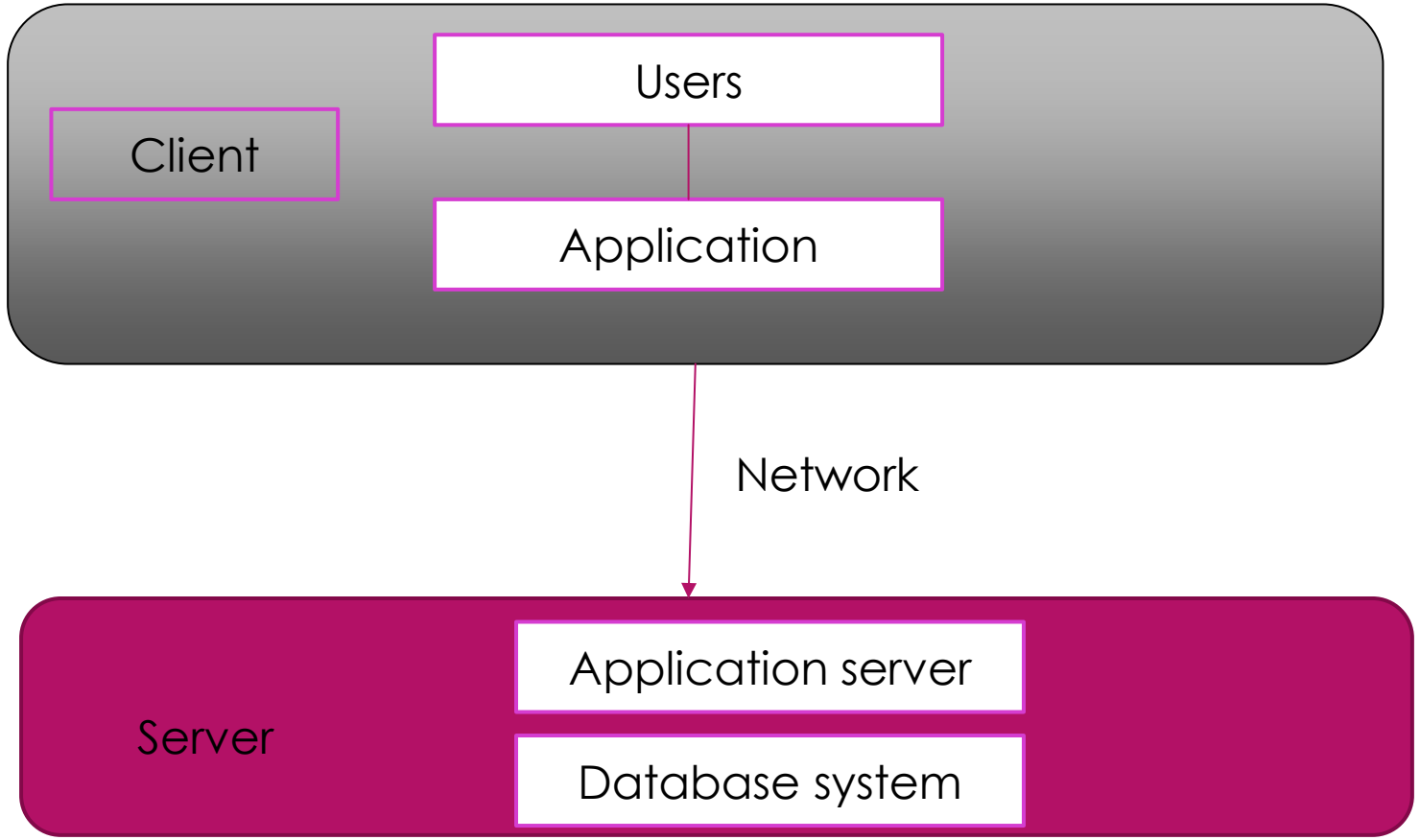
Client Layer (Tier 1): This is the user interface that library staff or users interact with. For example they might use a desktop application to search for books, issue them, or check due dates.

Database Layer (Tier 2): The database server stores all the library records such as book details, user information and transaction logs.

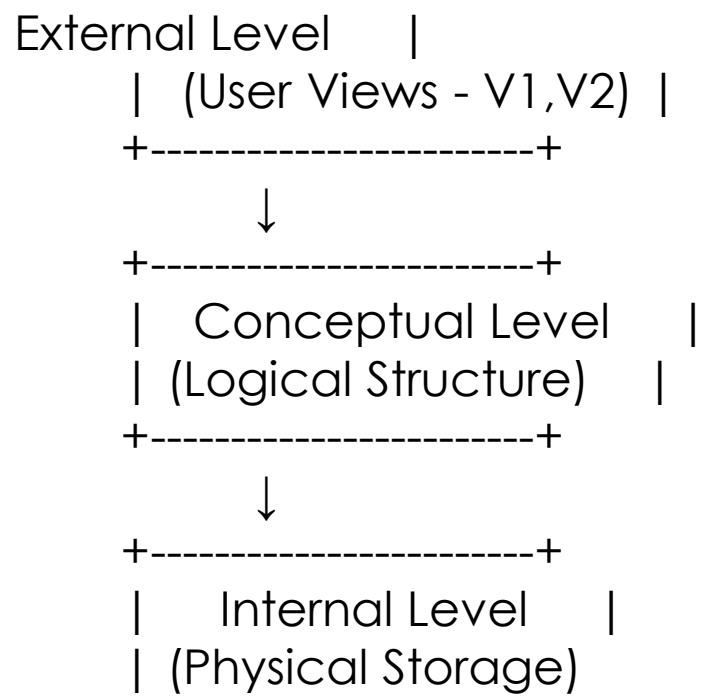


Three tier Architecture

The client does not directly communicate with the server. Instead, it interacts with an application server which further communicates with the database system and then the query processing and transaction management takes place.



ANSI Three-Schema Architecture of database



Data Types

Data Types	Description	
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1	
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 65535	
INTEGER(*size*)	Equal to INT(As per size)	
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615.	
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)	
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65.	
DATE DATETIME(*fsp*)	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time	

Data Independence

- Data independence is a feature of database systems that allows you to change the database schema at one level without affecting the schema at the next higher level.
- It separates data from the applications that use it, making databases more flexible, maintainable, and scalable over time.
- There are two types of data independence:
  - Physical Data Independence (changes to the internal/physical schema without affecting the conceptual schema) and
  - Logical Data Independence (changes to the conceptual schema without affecting the external/user view schema.)

Data Independence

- **Physical Data Independence:**
  - The ability to change the internal or physical schema (how the data is stored on disk) without affecting the conceptual or logical schema).
  - Example: changing in storage device (Hard disk to some other device), Modifying file organization techniques or indexing strategy, Disk partitioning.
- **Logical data independence:**
  - The ability to change the conceptual or logical schema (structure of database) without affecting external schema.
  - Example: adding new data to the conceptual schema, Modifying the logical structure of database.

Why data independence

- **Reduced Maintenance Effort:** Changes to the database don't require extensive updates to application programs, saving time and money.
- **Increased Flexibility:** The database can evolve and adapt to new requirements without disrupting existing functionalities.
- **Improved Scalability:** The system can handle growth and changes more easily, making it more scalable.
- **Enhanced Data Security and Integrity:** By separating different levels of data, it becomes easier to manage and secure the data

Mapping conceptual and External

- **Conceptual-External Mapping** (also called **View Mapping**) is part of the **three-level architecture** defined by the **ANSI/SPARC model**.

- **Conceptual-External Mapping** is the process of **mapping each user's external view (or schema) to the conceptual schema** of the database. It ensures that changes in the **conceptual schema** do not affect the **external schema**, maintaining **data independence**.
- **Purpose of Mapping:**

  •**Customization**: Different users can have customized views of the same database.
  •**Security**: Only authorized data is exposed to specific users.
  •**Independence**: Allows changes in the conceptual schema without affecting user applications.

Conceptual and Internal

- **Conceptual-Internal Mapping** defines **how the logical structures (tables, columns, relationships)** in the **conceptual schema** are stored and represented **physically** in the internal schema.
- It tells the DBMS **how to translate logical requests** into physical operations like file access, indexing, block storage, etc.

- Purpose:
  - **Data Abstraction**: Hides details of data storage from the logical model.
  - **Physical Data Independence**: Changes in storage structure (e.g., file organization, indexes) don't affect the logical schema.
  - **Optimization**: Enables better performance by mapping logical structures to optimized storage.

Unit-2
Entity Relation Model

Introduction

- An entity-relationship (ER) model is a conceptual data model that describes the logical structure of a database by representing data as entities (things of interest), their attributes (properties), and the relationships (connections) between them.

- The ER model is visualized in an ER diagram, which serves as a blueprint for database design, helping to plan, understand, and implement complex data structures in a clear, high-level way

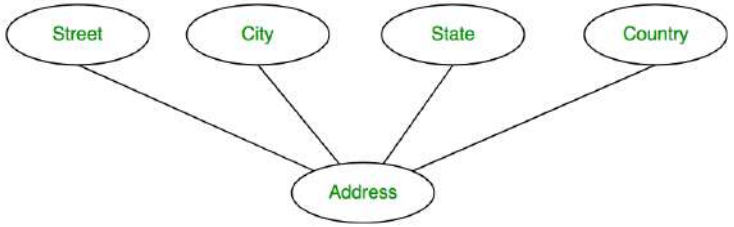
Component of ER model

- **Entity:**
  - A real-world object or concept with an independent existence that is stored as data, such as a "Student" or a "Course".
  - **Types of entity: Strong and weak entity**
  - **Real-World Objects:** Person, Car, Employee etc. **Concepts:** Course, Event, Reservation etc. **Things:** Product, Document, Device etc.
- **Attribute:**
  - A property or characteristic that describes an entity, like a "StudentID" for a student or "CourseName" for a course.
- **Relationship:**
  - The association or connection between two or more entities, such as a "Student enrolls in a Course".

Types of Attributes

- **Key attributes:** The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with an underline.

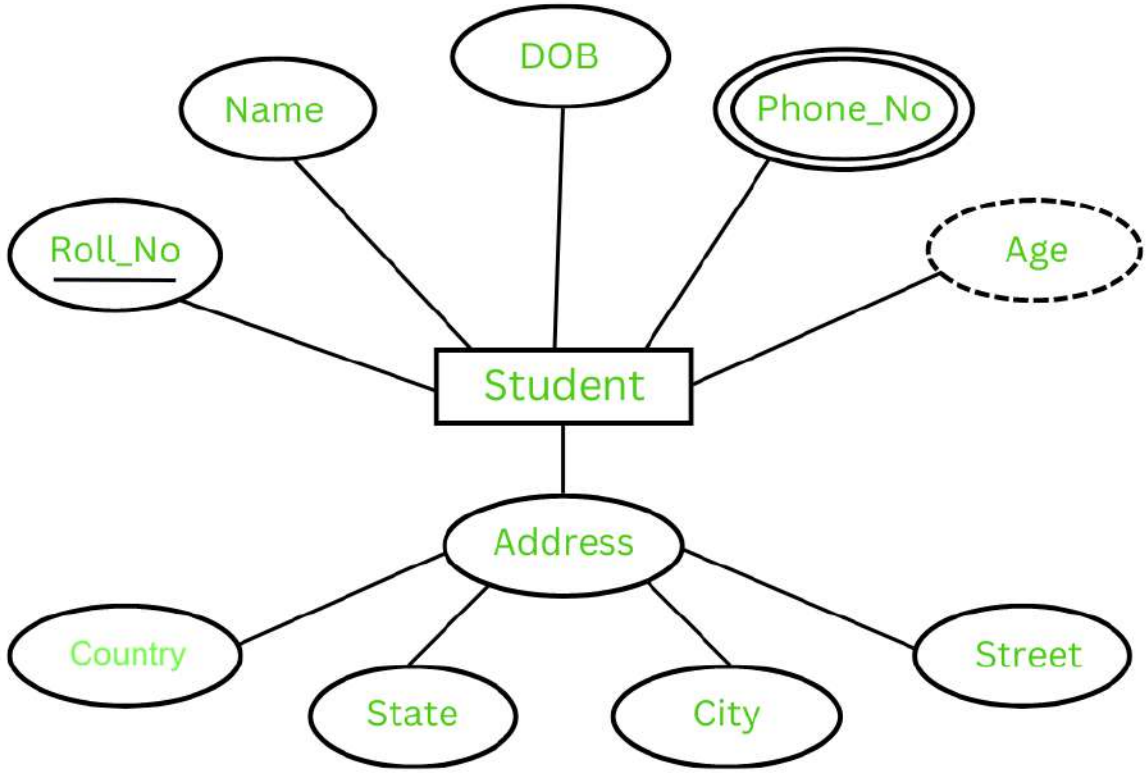
- **Composite Attributes:** An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



- **Multivalued Attributes:** An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.

- **Derived Attributes:** An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

Attributes


DOB
Name
Phone_No
Roll_No
Age
Student
Address
Country
State
City
Street

Attributes

- Attributes are used to describe the entity. The attribute is nothing but a piece of data that gives more information about the entity.

- These attributes are used to describe the entity in more detail.

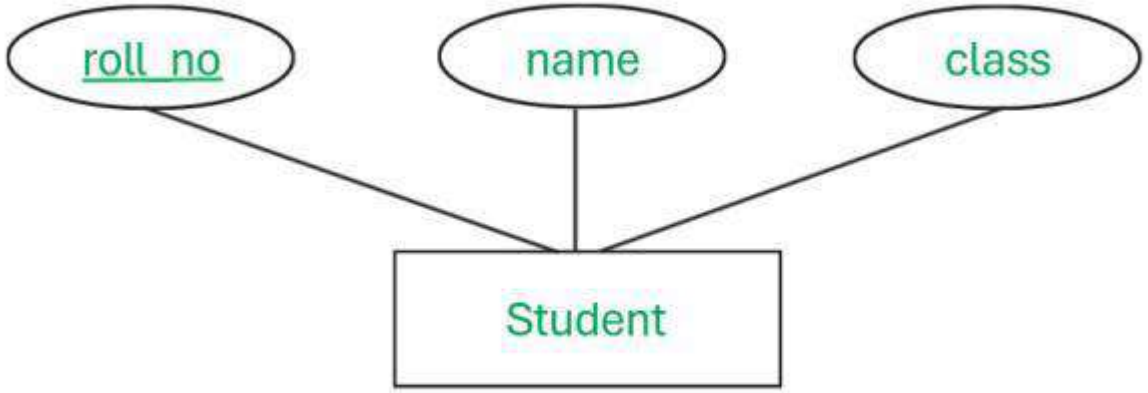
  - Customer database, the attributes might be name, address, and phone number.

  - Product database, the attributes might be name, price, and date of manufacture.

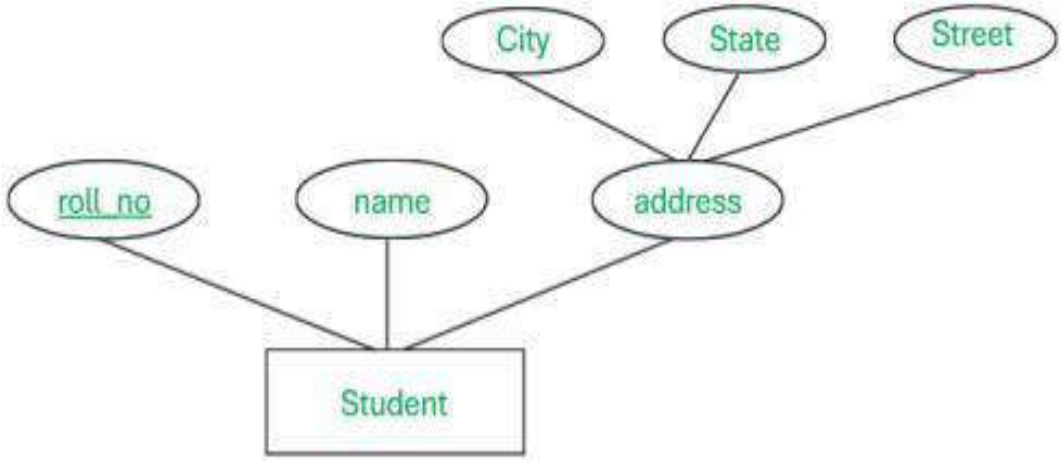
Types of Attributes

- Simple attributes
- Composite Attributes
- Single valued Attributes
- Multivalued Attributes
- Key Attributes
- Derived Attributes
- Stored Attributes
- Complex attributes

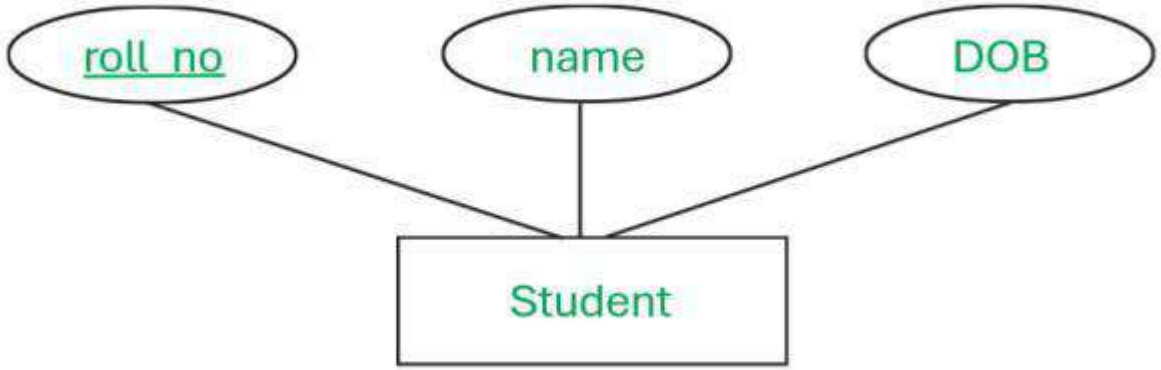
Simple



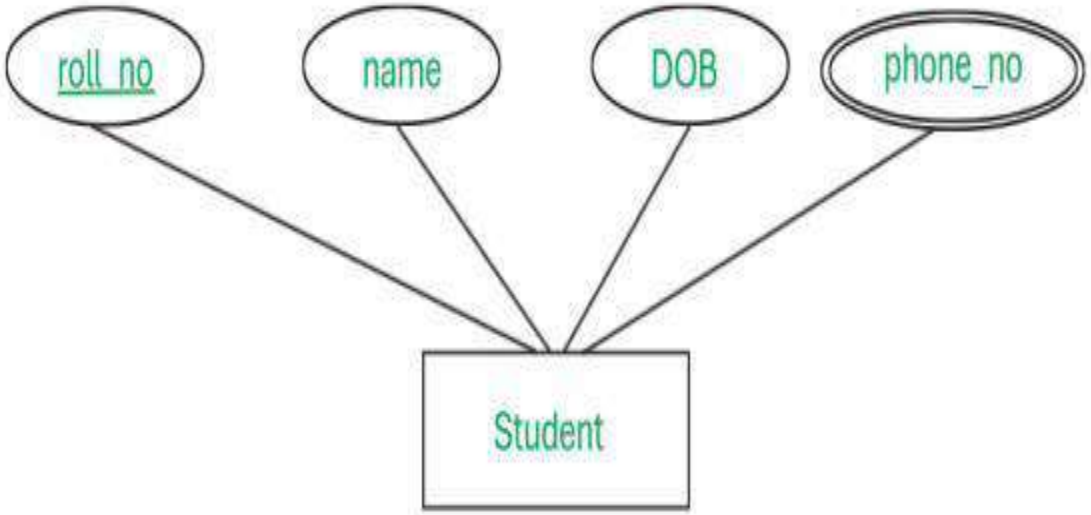
Composite



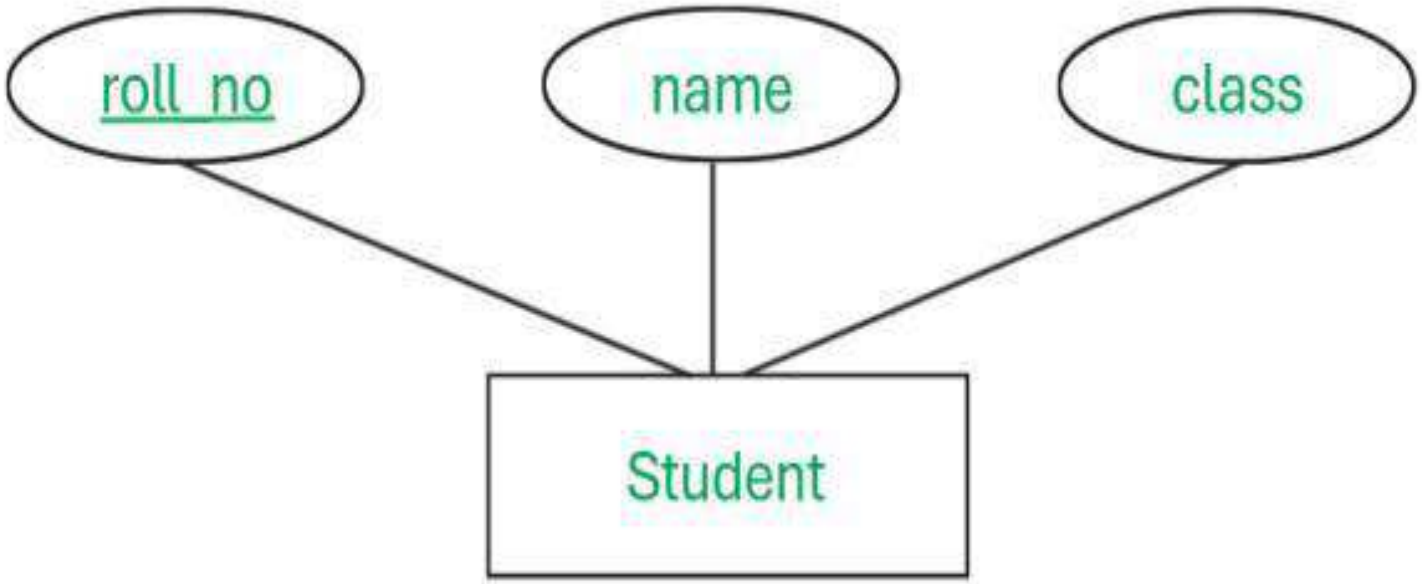
Single Valued



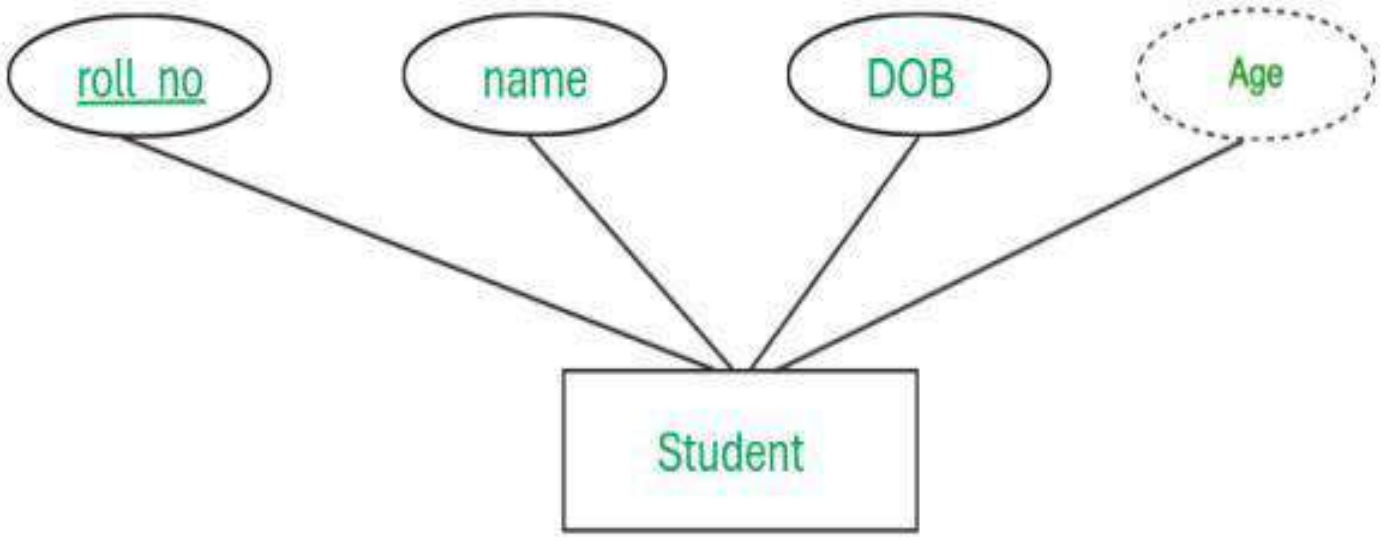
Multivalued



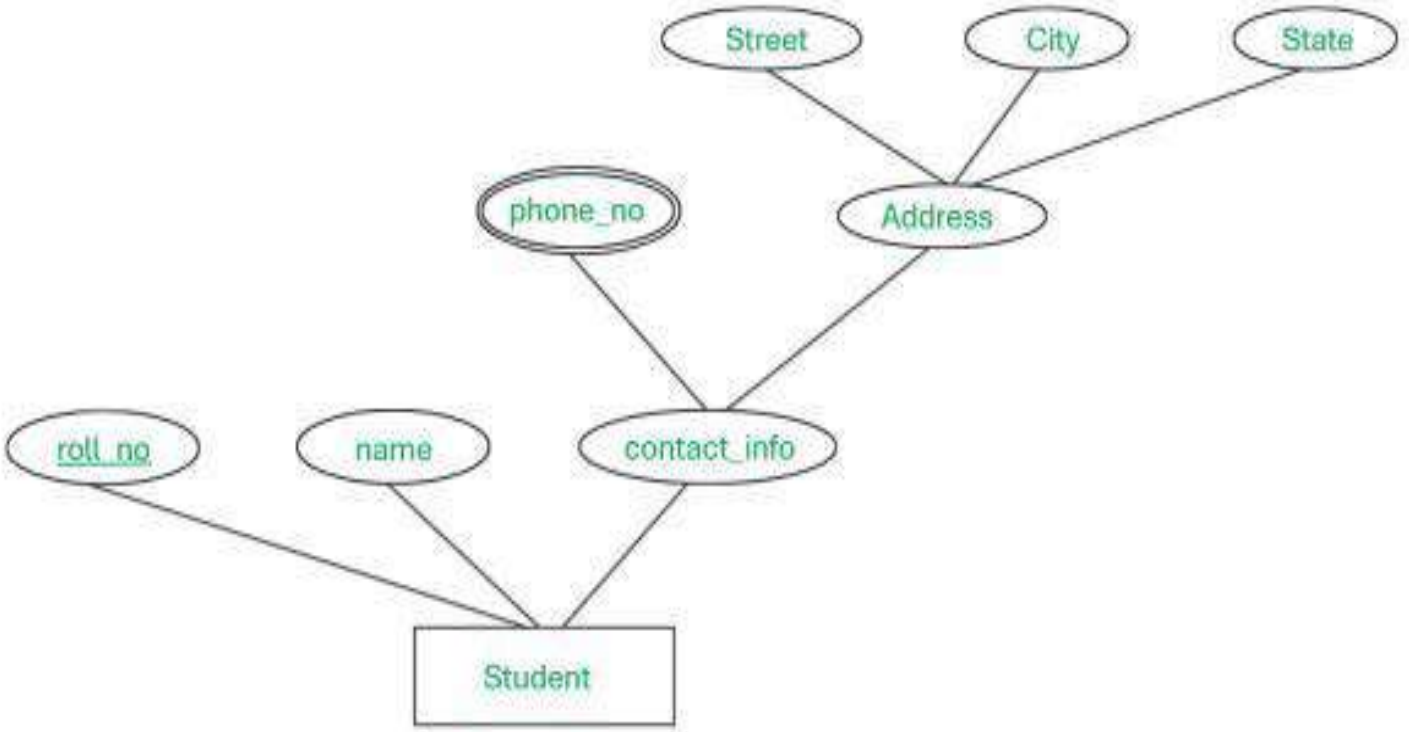
Key Attributes



Derived Atrributes

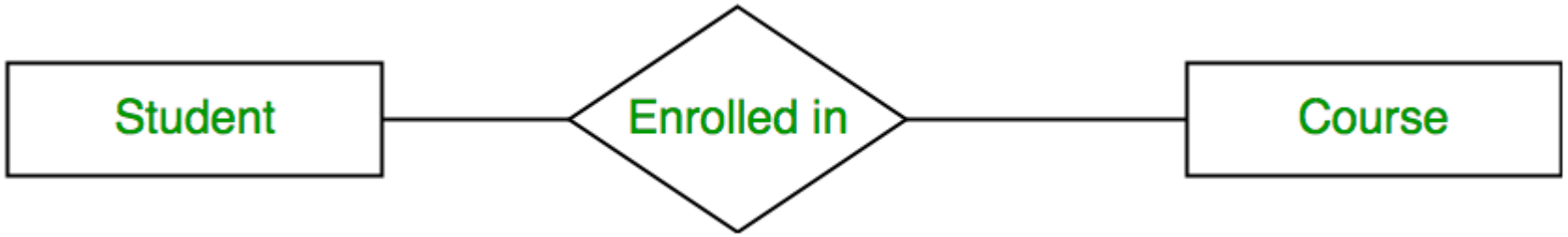


Complex Atrributes



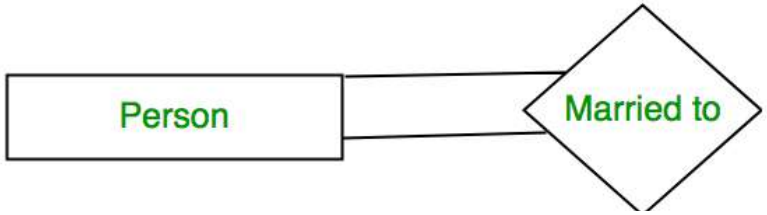
Relationship type and Relationship set

- A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.

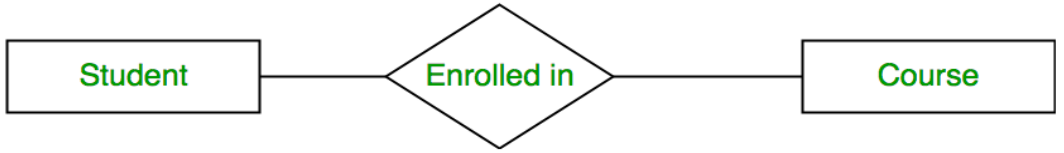


Types of Relationship

- **Unary Relationship:**
  - When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



- **Binary Relationship:**
  - When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



- **Ternary Relationship:**
  - When there are three entity sets participating in a relationship, the relationship is called a ternary relationship.
- **N-ary Relationship:**
  - When there are n entities set participating in a relationship, the relationship is called an n-ary relationship.

Purpose and Use

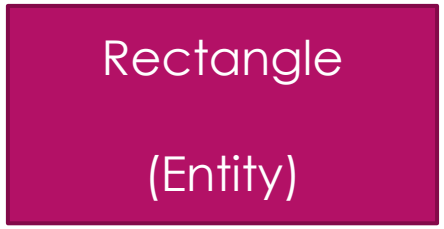
- **Database Design:**
  The ER model is used to design the logical structure of a database, providing a roadmap for its implementation.

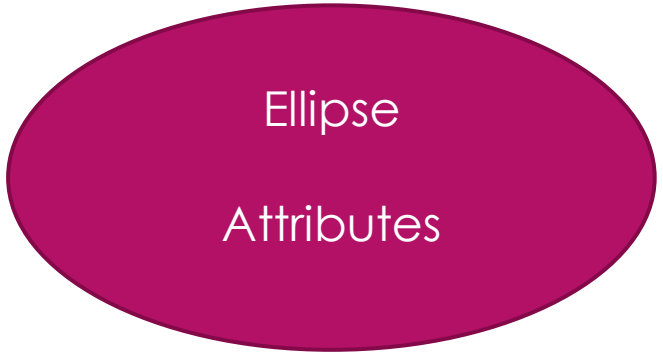
- **Data Modeling:**
  It helps in modeling data and defining the rules that govern it within a system or problem domain.

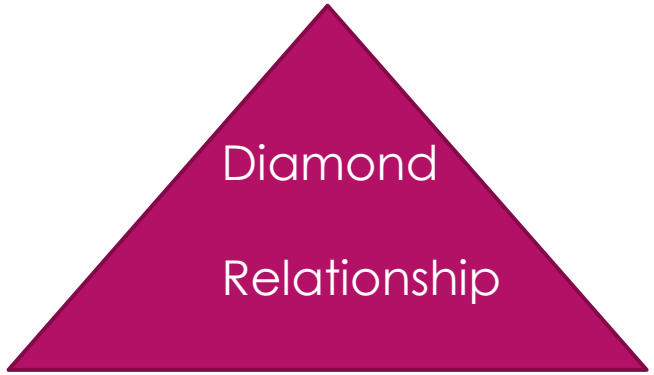
- **Communication Tool:**
  The visual nature of the ER diagram makes it an excellent tool for users and developers to understand the database structure before it is built.

- **Conceptual Blueprint:**
  It creates a high-level overview of the data, showing how different pieces of information relate to one another.

Symbols in ER Diagram







Connector

Data Flow Diagram

- A data flow diagram (DFD) is a visual map that illustrates how data moves through a system or process, showing its sources, destinations, processes, and storage points using standardized symbols.

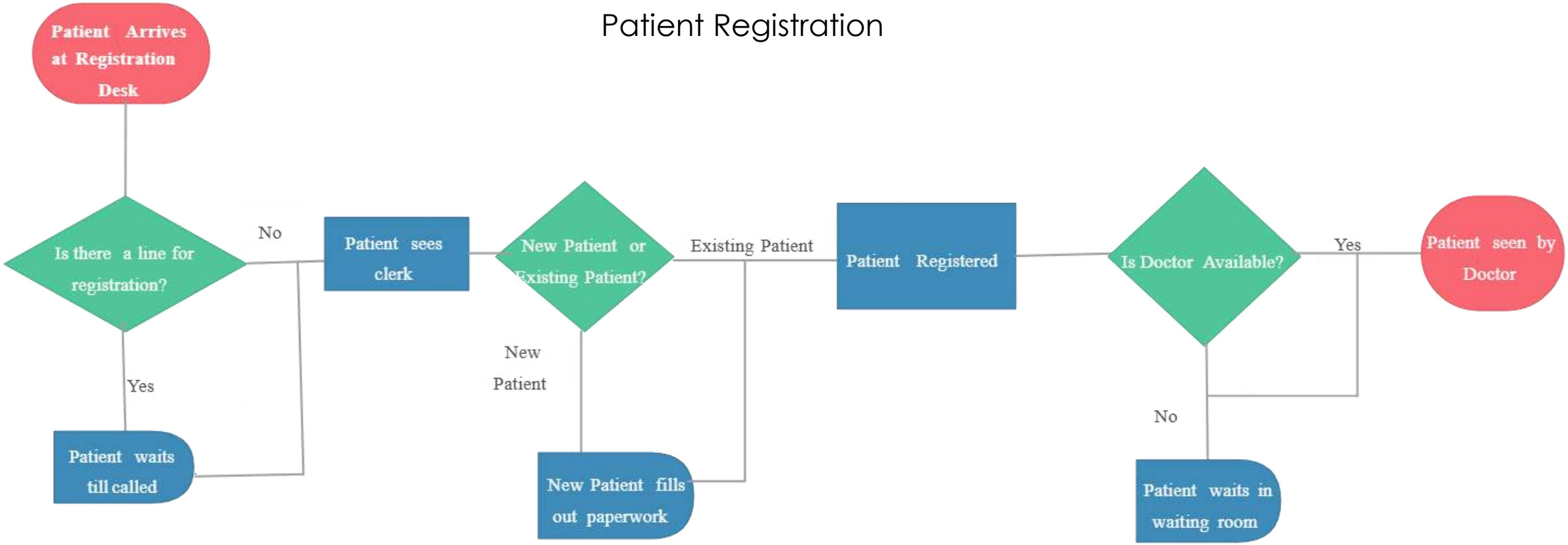
- A data flow diagram (DFD) is a visual representation of the flow of data through an information system or business process. DFDs make complex systems easier to understand.

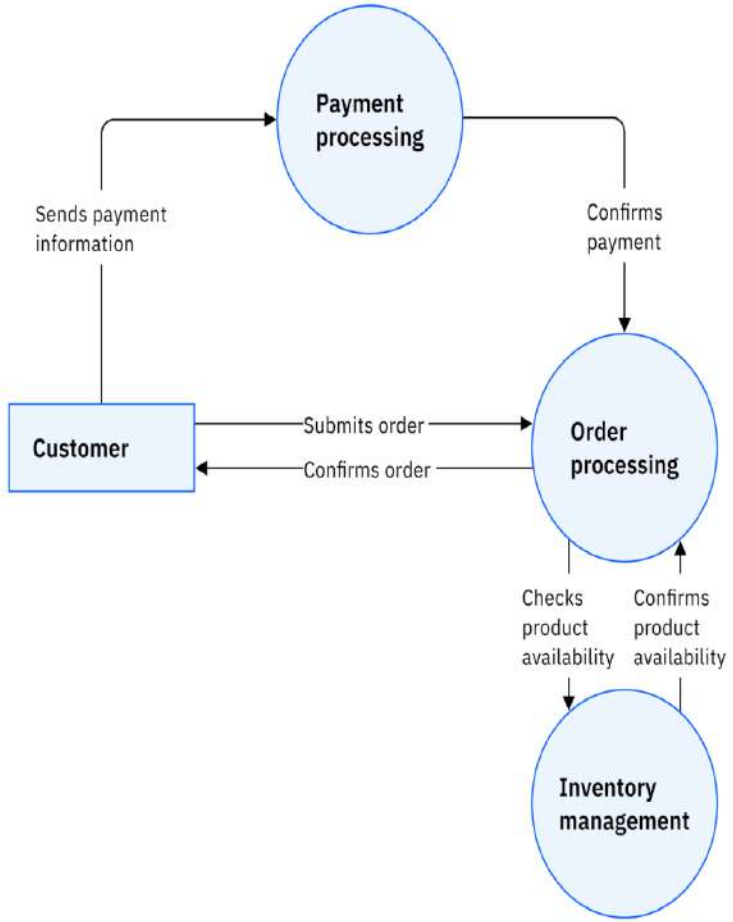
- DFDs help to understand, analyse, and improve system efficiency by making the flow of information clear and identifying potential issues in existing processes or in the design of new ones.

Symbols used in DFD

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

ER model (Data flow Diagram)




Payment processing
Sends payment information
Confirms payment
Customer
Submits order
Confirms order
Order processing
Checks product availability
Confirms product availability
Inventory management

Case study for DFD (students enrolment Process

- **Enquiry for course**
- **Enrolment in course**
- **Availability of information related to course**
- **Registration fee submission**
- **Course fee submission**
- **Collection of hall ticket**
- **Publication of Result**
- **Admit card issued for exam**
- **Entrance exam for enrolment**
- **Submission of registration application for entrance exam**
- **Collection of ID card**
- **Classes start**
- **Acceptance of time table and class allotment**
- **Status of result**

QUESTIONS:

1) Arrange in sequence.
2) Design DFD for students enrolment process

2nd case for DFD (Online order processing)

- **Collecting requirement**
- **Payment to merchant**
- **Selecting e-commerce website and visiting**
- **Reviewing information about product**
- **Delivery of goods**
- **Confirmation for dispatch of goods**
- **Order confirmation from merchant**
- **Checking Status of stock**
- **Confirming about purchasing by checkout process**
- **Generating tax invoice**
- **Receiving receipt of goods**
- **Consumption of product by consumer**

Data Relation Model

- The **relational data model** is a way to structure and organize data using **tables (called relations)**. Each table consists of **rows (tuples)** and **columns (attributes)**.
- **Key Concepts:**
  - **Table (Relation):** Represents an entity (e.g., Customers, Orders).
  - **Row (Tuple):** A single record in a table.
  - **Column (Attribute):** A field or property of the entity.
- **Primary Key:** Uniquely identifies each row in a table.
- **Foreign Key:** Links one table to another, creating relationships.

Example:
A Customers table might have columns like CustomerID, Name, and Email.
An Orders table could reference CustomerID as a foreign key to show which customer made the order.

Example of Relational Data Model

Customers Table-1

CustomerID	Name	Email
1	Alice Smith	alice@example.com
2	Bob Johnson	bob@example.com
3	Charlie Lee	charlie@example.com

Primary Key

Orders table-2

OrderID	OrderDate	CustomerID	Amount
101	2025-09-01	1	250.00

Foreign Key
Refers to customer.customerID

Join Command:

```
SELECT Customers.Name, Orders.OrderDate, Orders.Amount
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Types of Key

- Primary key;
  - Uniquely identifies each record in table, Cannot be NULL of Duplicate.
  - One per Table. Example: **CustomerID** in customer Table.
  - **Declaration: StudentID INT PRIMARY KEY**

- Foreign Key
  - A field in one table that refers to the primary key in another table.
  - Creates a relationship between tables.
  - **CustomerID** in order table (Reference customers)

  - **FOREIGN KEY (StudentID) REFERENCES Students(StudentID)**

- **Candidate Key**
  - A field (or combination of fields) that could serve as primary key.
  - Must be **unique** and not **null.**
  - A table can have multiple candidate keys, but only one is chosen as the **Primary key**.

- **Composite Key**
  - A key made up of two or more column to uniquely identify a row.
  - Used when no single column is unique on its own.
  - Example: (studentsID, CourseID) in a same table.

- **SUPER KEY**
  - Any set of columns that uniquely identifies a row.
  - May contain extra attributes beyond whats necessary.
  - All candidate keys are super key, but not all super keys are candidate keys.

in Students, these are all **super keys**:

StudentID and Email

(StudentID, Email)

(StudentID, Phone)

Aalternate Key

It may be potential or second primary key in table but not declared. As primary key.

Student_ID	Name	Email	Phone
101	Alice smith	abc@ghy.com	623-486-7854
102	charlie	Charlie@ghy.com	458-654-2586

Enrollment table

Student_ID	Course_ID	Grade
101	CS01	A
102	MA01	A+

Key	Table	Column	Notes
Primary Key	Students	StudentID	Uniquely identifies each students
Composite Key	Enrollement	StudentsID, CourseID	
Foreign Key	Enrollment	StudentsID	Referes to Students(studentID)
Candidate key	Students	StudentsID, Email	Both are unique and eligible for primary key
Alternate key	students	Email	Candidate key not selected as PK.
Composite Key	enrollment	StudentsID	
Super Key	Students	StudentsID, Email	Any unique compbination including PK

Codd Rule

Rule 0: The Foundation Rule

- A DBMS can only be called relational if it manages data entirely through its relational capabilities.

Rule 1: The Information Rule

- All information in the database must be represented in a logical, table-based format (rows and columns).

Rule 2: The Guaranteed Access Rule

- Every piece of data (atomic value) must be accessible by combining its table name, primary key, and column name.

Rule 3: Systematic Treatment of NULL Values

- NULL values must be handled systematically, representing missing or inapplicable information, not empty strings or zero.

Rule 4: Dynamic Online Catalog Rule

- The database structure (metadata) must be maintained in an online catalog, accessible using the same relational language as the data itself.

Cont...

- **Rule 5: The Comprehensive Data Sub-Language Rule**
  - The system must support a comprehensive language (like SQL) for data definition, manipulation, view definition, and integrity constraints.

- **Rule 6: The View Updation Rule**
  - Any view that can be theoretically updated must be updateable by the system.

- **Rule 7: High-level Insert, Update, and Delete Rule**
  - Operations to insert, update, and delete data must be supported at a high, set-based level, treating rows as sets.

- **Rule 8: Physical Data Independence**
  - Application programs and user activities should remain unaffected by changes to the physical storage structure or access methods.

- **Rule 9: Logical Data Independence**
  - Application programs and users should remain independent of changes in the logical structure of the database, such as adding new tables.

Cont...

- **Rule 10: Integrity Independence**
  - All integrity constraints must be definable and stored within the database's system catalog.

- **Rule 11: Distribution Independence**
  - The database must allow for manipulation of data that is distributed across different computer systems without affecting the application.

- **Rule 12: Non-Subversion Rule**
  - If a system provides low-level access, this access must not be able to bypass or subvert the database's integrity and security rules

Cross Product

- The **cross product** (also called **Cartesian product**) is a **binary operation** in relational algebra that combines **every row** of one relation with **every row** of another.
- Its denoted as a R × S.
- Where R and S are two relation.

The result includes **all combinations** of tuples from R and S

Table: R (Students)

StudentID	Name
1	Alice
2	Bob

Table: S (Courses)

CourseID	CourseName
C1	Math
C2	Physics

R × S (Cross Product):

StudentID	Name	CourseID	CourseName
1	Alice	C1	Math
1	Alice	C2	Physics
2	Bob	C1	Math
2	Bob	C2	Physics

Addition

- **Addition (+)**
  - The addition operator is used to sum values. Performs addition between:
  - Column and constant
  - Two columns

```
SELECT employee_id, employee_name, salary,
salary + employee_id AS "salary + employee_id"
FROM addition;
```

employee_id	employee_name	salary	salary+employee_id
1	Alex	25000	25001
2	RR	55000	55002
3	JPM	52000	52003
4	GGSHMR	12312	12316

```
SELECT employee_id, employee_name, salary,
salary + 100 AS "salary + 100"
FROM addition;
```

employee_id	employee_name	salary	salary+100
1	Alex	25000	25100
2	RR	55000	55100
3	JPM	52000	52100
4	GGSHMR	12312	12412

Substraction

The subtraction operator deducts one value from another.
Performs subtraction between:

- Column and constant
- Two columns

```
SELECT employee_id, employee_name, salary,
salary - 100 AS "salary - 100"
FROM subtraction;
```

employee_id	employee_name	salary	salary-100
12	Finch	15000	14900
22	Peter	25000	24900
32	Warner	5600	5500
42	Watson	90000	89900

```
SELECT employee_id, employee_name, salary,
salary - employee_id AS "salary - employee_id"
FROM subtraction;
```

employee_id	employee_name	salary	salary-employee_id
12	Finch	15000	14988
22	Peter	25000	24978
32	Warner	5600	5568
42	Watson	90000	89958

Multiplication

```
SELECT employee_id, employee_name, salary,
salary * 100 AS "salary * 100"
FROM addition;
```

employee_id	employee_name	salary	salary*100
1	Finch	25000	2500000
2	Peter	55000	5500000
3	Warner	52000	5200000
4	Watson	12312	1231200

```
SELECT employee_id, employee_name, salary,
salary * employee_id AS "salary * employee_id"
FROM addition;
```

employee_id	employee_name	salary	salary*employee_id
1	Finch	25000	25000
2	Peter	55000	110000
3	Warner	52000	156000
4	Watson	12312	49248

Division

```
SELECT employee_id, employee_name, salary,
salary / 100 AS "salary / 100"
FROM addition;
```

employee_id	employee_name	salary	salary/100
1	Finch	25000	250
2	Peter	55000	550
3	Warner	52000	520
4	Watson	12312	123.12

Modulus

When any arithmetic operation is performed on a NULL value, the result is always NULL.

Example:

```
SELECT employee_id, employee_name, salary, type,
type + 100 AS "type + 100"
FROM addition;
```

employee_id	employee_name	salary	type	type+100
1	Finch	25000	NULL	NULL
2	Peter	55000	NULL	NULL
3	Warner	52000	NULL	NULL
4	Watson	12312	NULL	NULL

- NULL means unknown/unavailable
- Any operation with NULL results in NULL

Join Operator

- A **join** is used to combine rows from two or more tables based on a related column between them. It allows you to query data that is spread across multiple tables.

- **Common Types of Join Operators**
  - **INNER JOIN**

CustomerID	Name
1	x1
2	x2
3	x3

Cus_Id	Name
1	Y1
2	Y2
3	y4

Cust_id	Name
1	X1,y1
2	X2, y2

Name	Product
Alice	Laptop
Charlie	Phone

```
SELECT column_name from tableA
INNER JOIN TableB  ON TableA.col_name=TableB.col_name;
```

Left Join

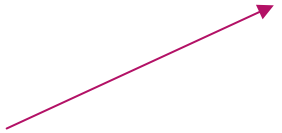
Returns **all rows from the left table** (Employees) and the **matched rows** from the right table (Departments).

If there is **no match**, the result will contain **NULLs** for the right table columns.

```
SELECT column_name from tableA
LEFT JOIN TableB  ON TableA.col_name=TableB.col_name;
```

Output

id	Name	ID	Name
1	X1	1	Y1
2	X2	2	Y2
3	x3	4	y4



ID	Name	
1	X1	y1
2	x2	y2
3	x3	Null

Right Join

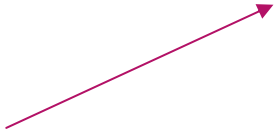
Returns **all rows from the right table** (Departments) and the **matched rows** from the left table (Employees).

If there is **no match**, the result will contain **NULLs** for the left table columns.

```
SELECT column_name from tableA
Right JOIN TableB  ON TableA.col_name=TableB.col_name;
```

id	Name
1	X1
2	X2
3	x3

ID	Name
1	Y1
2	Y2
4	y4



ID	Name	
1	X1	y1
2	x2	y2
4	Null	y4

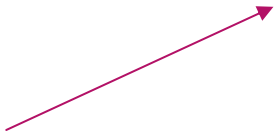
Full Join

Returns **all rows from both tables**.
•If there's a match between the tables, the matched data is shown.
•If there's **no match**, the result will include NULLs for the missing side.
Think of it as **LEFT JOIN + RIGHT JOIN combined**.

```
SELECT column_name from tableA
Full Outer Join TableB  ON TableA.col_name=TableB.col_name;
;
```

id	Name
1	X1
2	X2
3	x3

ID	Name
1	Y1
2	Y2
4	y4



ID	Name	
1	X1	y1
2	x2	y2
3	x3	Null
4	Null	y4

Relational Algebra Operator

- **Selection (σ)**
  - **Purpose:** Selects **rows (tuples)** from a relation that satisfy a given condition.
  - **Symbol:** σ\sigmaσ
  - **Works on:** Entire rows, filtering based on predicates (conditions).
- **Syntax:**
  - σcondition(Relation)
- **Example:**
- Get employees with salary greater than 50,000:
  - σsalary>50000(Employees)

  - Select * from employee where salary>50,000;

- **Purpose:** Selects **columns (attributes)** from a relation, discarding others.
- **Symbol:** π\piπ
- **Works on:** Columns, returning a relation with only those attributes.
- πattribute1, attribute2, ...(Relation)

- πname, department_id(Employees)

Database Normalization

From Functional Dependencies to Third Normal Form

What is Database Normalization?

Definition

Process of organizing data to reduce redundancy and improve data integrity

Purpose

Eliminate data anomalies and ensure efficient storage structure structure

Benefits

Reduces storage space, prevents inconsistencies, and improves data quality quality

Understanding Functional Dependency

Key Concept

A functional dependency exists when one attribute uniquely determines another another attribute in a relation.

Notation: X → Y (X determines Y)

- Student ID → Student Name
- Course Code → Course Title
- Employee ID → Department

First Normal Form (1NF)

01

Atomic Values

Each cell contains only single, indivisible values - no repeating groups or arrays

02

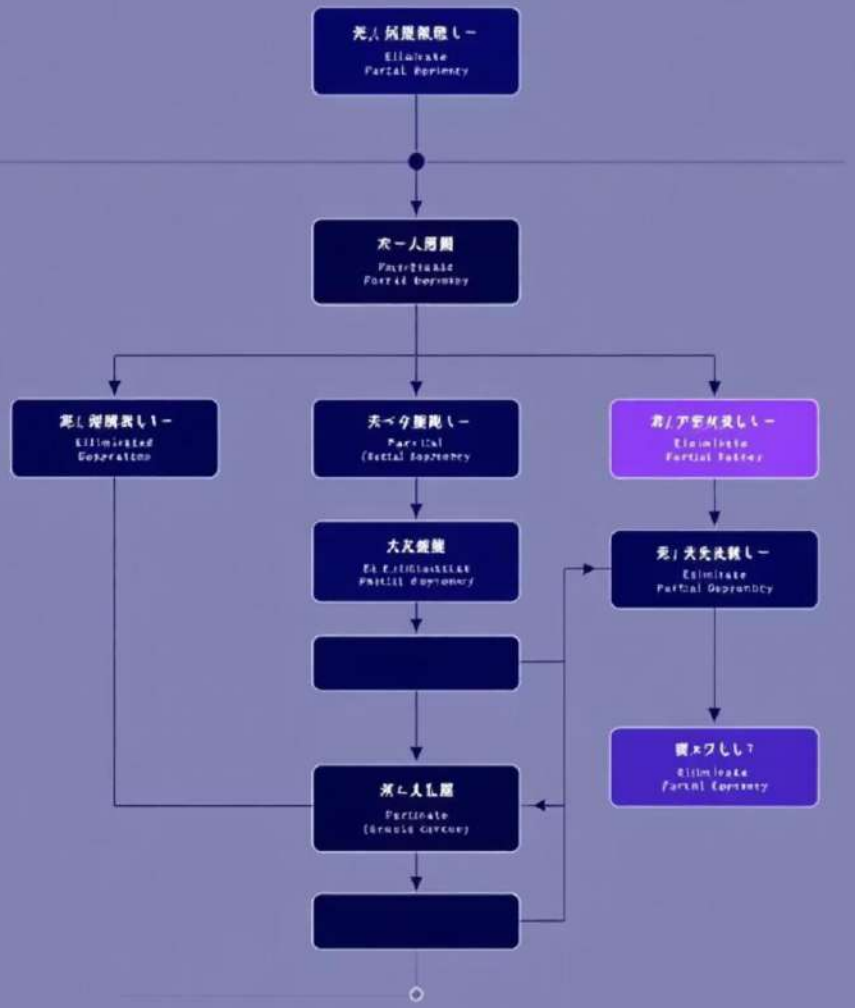
Unique Rows

Each row must be unique with no duplicate records in the table

03

Primary Key

Table must have a primary key to uniquely identify each record



Database

Second Normal Form (2NF)

1

Must be in 1NF

Table already satisfies First Normal Form requirements

2

No Partial Dependencies

Non-key attributes must depend on the entire primary key, not just part of it

Applies only to tables with **composite primary keys**. Remove partial dependencies dependencies by creating separate tables.

Third Normal Form (3NF)

Must be in 2NF

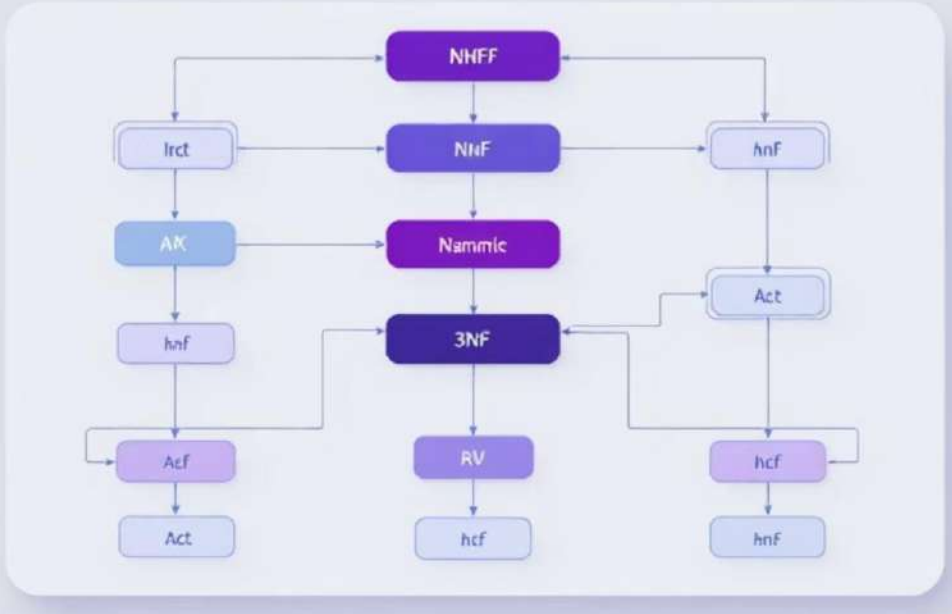
Table satisfies both 1NF and 2NF 2NF requirements

No Transitive Dependencies

Non-key attributes cannot depend on other non-key attributes

Direct Dependency Only

All non-key attributes must depend directly on the primary key



Normalization Process Overview

Unnormalized Form

Raw data with repeating groups and redundancy

First Normal Form

Eliminate repeating groups, ensure atomic values

Second Normal Form

Remove partial dependencies on composite keys

Third Normal Form

Eliminate transitive dependencies between non-key attributes attributes

Case Study: Student Enrollment System

Initial Unnormalized Table

Student_ID	Student_Name	Course_Code	Instructor
101	John Smith	CS101, CS102	Dr. Brown, Prof. Lee
102	Jane Doe	CS101	Dr. Brown

Problems: Repeating groups, non-atomic values, potential data inconsistency

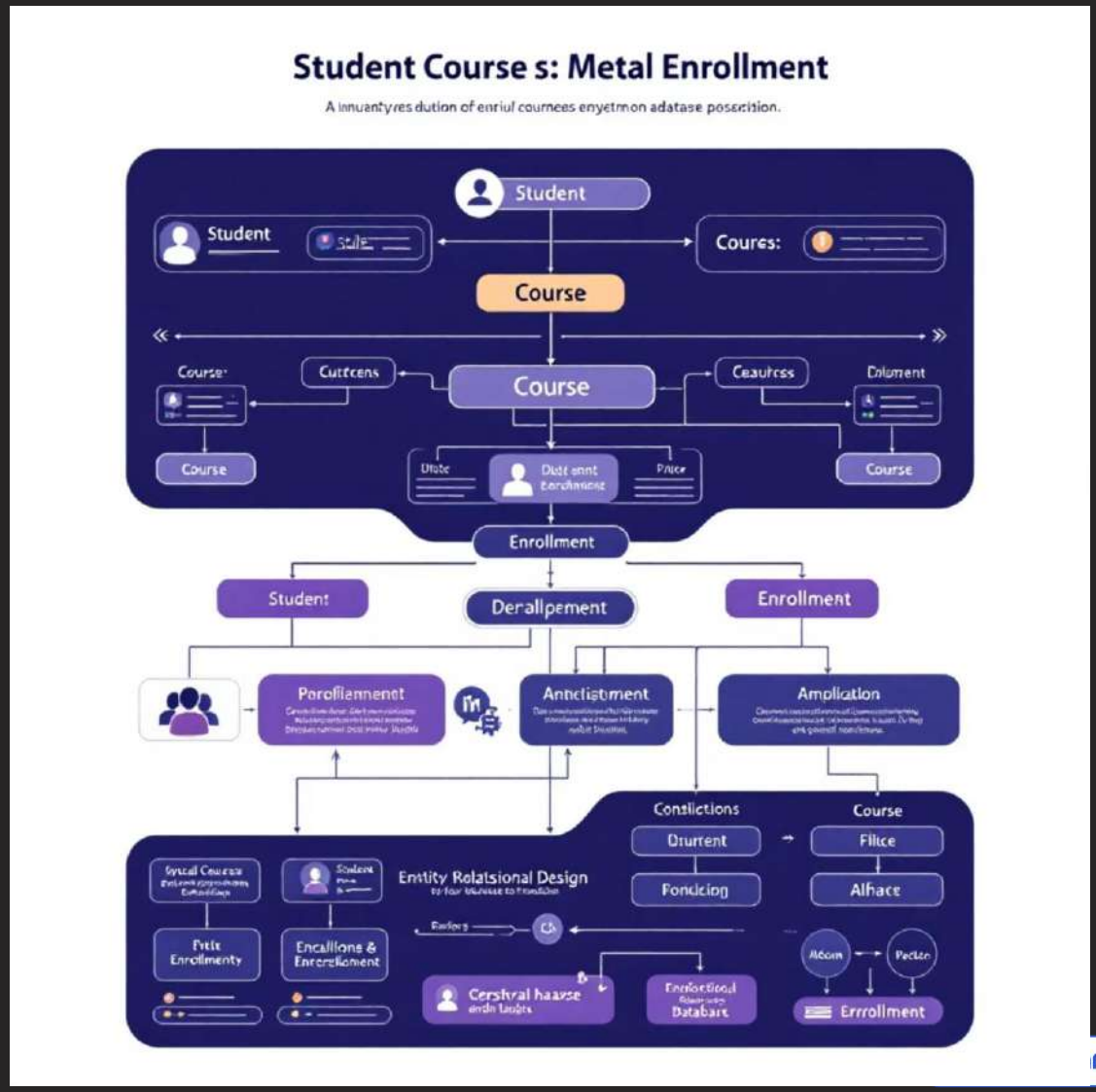
Case Study: ERD and Schema Design

Entity Relationship Diagram

Three main entities identified:

- **Student** (Student_ID, Name, Email)
- **Course** (Course_Code, Title, Credits)
- **Enrollment** (Student_ID, Course_Code, Grade)

Many-to-many relationship between Students and Courses



Final Normalized Schema (3NF) (3NF)

Students Table

Student_ID (PK), Student_Name, Email, Email, Phone

Courses Table

Course_Code (PK), Course_Title, Credits, Department

Enrollments Table

Student_ID (FK), Course_Code (FK), Semester, Grade

Result: Eliminated redundancy, improved data integrity, and achieved 3NF compliance

Unit-5 (Data Recovery)

Crash Recovery and backup

Introduction and Importance

- **Database Recovery** is the process of restoring the database to a correct state after a failure occurs.
- Failures could be due to system crashes, hardware failures, software bugs, or power outages.
- The goal is to **ensure durability and consistency** of the database despite failures.

- **Why is Recovery Important?**
  - Transactions that were **committed** before failure must be **preserved**.
  - Transactions that were **incomplete** or **in-progress** must be **rolled back**.
  - Prevent **loss of data** and **corruption** caused by partial transaction execution.

Types of Failure

- **Transaction Failure**
  - A transaction cannot complete due to logical errors or constraint violations.
  - Solution: Rollback the transaction.

•Logical errors (e.g., invalid data operations)
•Deadlocks (two or more transactions waiting indefinitely)
•User-initiated abort

- **System Crash**
  - The entire system fails (e.g., power loss).
  - Solution: Recover the database to a consistent state using logs.
- **Media Failure**
  - Physical damage to storage devices (disk crashes). A failure of the storage medium (e.g., hard disk crash, bad sectors).
  - Solution: Restore data from backups and apply logs.

Recovery Techniques

- **Application Failure**
  - The application interacting with the database fails or behaves unexpectedly.
- **Examples:**
  - Incorrect input from a user, Bugs in application logic, API misuse
- **Handled By:**
  - Transaction controls (commit/rollback), Input validation and error handling
- **Concurrency Failure**
  - Multiple transactions interfere with each other in an uncontrolled manner.
- **Examples:**
  - Lost updates
  - Dirty reads
  - Uncommitted data access
- **Handled By:**
  - Concurrency control protocols (e.g., locking, timestamp ordering)

Database Recovery Techniques

- **Deferred Update**
- **Concept:** Changes made by a transaction are **not immediately applied** to the database. Instead, all updates are **deferred until the transaction commits**. Only after commit, the changes are written to the database.
- **How it works:**
  - During the transaction, all changes are stored in a **buffer** or **log**.
  - If the transaction aborts, no changes are applied — no undo needed.
  - If the transaction commits, all changes are applied to the database in one go.
- **Advantages:**
  - Easy recovery (no undo needed, since nothing is applied before commit).
  - Atomicity is straightforward.
- **Disadvantages:**
  - May delay updates and reduce concurrency.
  - Requires enough memory for buffering

- **Immediate Update**
  - **Concept:** Changes made by a transaction are **applied immediately** to the database, even before the transaction commits.
  - **How it works:**
    - Updates are written directly to the database as they happen.
    - A **log** is maintained to record old and new values for recovery.
    - If a transaction aborts, changes are **undone** using the log.
- **Advantages:**
  - More concurrency, as updates are visible earlier.
  - No delay in applying changes.
- **Disadvantages:**
  - Requires complex recovery (both undo and redo).
  - Must follow **write-ahead logging (WAL)** to ensure logs are saved before changes.

- **Shadow Paging**
  - **Concept:** Instead of overwriting database pages directly, a **shadow copy** of the database is maintained.
- **How it works:**
  - When a transaction starts, the system keeps a **shadow page table** referencing the current pages.
  - Updates are made on copies (shadow pages), not on the original pages.
  - Upon commit, the shadow page table replaces the current page table, making changes visible atomically.
  - If abort occurs, discard the shadow pages, so original database remains unchanged.
- **Advantages:**
  - No need for undo logging since the original pages are untouched until commit.
  - Atomic commit via page table swap.
- **Disadvantages:**
  - Can consume more disk space (due to shadow copies).
  - Managing page tables can be complex.

- **Checkpoint**
  - **Concept:** A **checkpoint** is a snapshot of the database state at a certain point in time, which simplifies recovery.
- **How it works:**
  - The system periodically writes a checkpoint record in the log.
  - At checkpoint time, all modified data pages and log entries up to that point are flushed to disk.
  - On recovery, the system starts from the last checkpoint, reducing the amount of log that needs to be processed.
- **Advantages:**
  - Speeds up recovery by limiting the search space in logs.
  - Provides a consistent state from which to start redo/undo.
- **Disadvantages:**
  - Checkpointing can cause a temporary performance hit.

Recovery Process

- **Analysis Phase**
  - Identify which transactions were active or committed at the time of failure.
- **Redo Phase**
  - Reapply all operations of committed transactions to ensure durability.
- **Undo Phase**
  - Roll back all incomplete or aborted transactions to maintain atomicity.

- **Backup** refers to creating a duplicate of the database (or parts of it) at a specific point in time, so it can be restored if needed.
- Purpose of backup:
  - •Recover from **hardware failures**, **software bugs**, **user errors**, or **cyberattacks**.
  - •Restore the database to a **consistent state**.
  - •Ensure **data availability** and **business continuity**.

Types of Backup

1) **Full Backup**
   - Copies the entire database. Takes more time and storage but simplest for recovery.

2) **Incremental Backup**
   - Backs up only the data changed since the last backup (full or incremental).
   - Faster and uses less storage but recovery is slower.

3) **Differential Backup**
   - Backs up data changed since the last full backup. Faster than full backup, slower than incremental.

- **Mirror Backup (Real-time Backup)**
  - A live, exact copy of the database is maintained at another location.
  - Very fast to switch to in case of failure. High cost and complex setup.
- **Physical Backup**
  - Copies the actual database files, such as tablespaces, datafiles, control files.
  - Faster to restore and more space-efficient, Usually done by the DBMS itself (e.g., Oracle RMAN, MySQL physical copy).

Aspect	Description
Backup	Copying data for protection
Types of Backup	Full, Incremental, Differential
Recovery	Restoring data after failure
Failure Types	Transaction, System crash, Media failure
Recovery Techniques	Rollback, Rollforward, Log-based recovery
Key Tools	Checkpoints, Transaction logs, Archiving

Data Definition Language

- **DDL** is Data Definition Language, which deals with database structure and descriptions, of how the data should reside in the database.

  - **CREATE:** to create a database and its objects like (table, index, views, store procedure, function, and triggers)
  - **ALTER:** alters the structure of the existing database
  - **DROP:** delete objects from the database
  - **Deletion:** remove all records from a table, including all spaces allocated for the records are removed
  - **COMMENT:** add comments to the data dictionary
  - **RENAME:** rename an object

Data Manipulation Language

- **DML** is Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

- D**ata query language(DQL)** is the subset of "Data Manipulation Language". The most common command of DQL is **SELECT** statement. SELECT statement help on retrieving the data from the table without changing anything in the table.
  - **SELECT:** retrieve data from a database
  - **INSERT:** insert data into a table
  - **UPDATE:** updates existing data within a table
  - **DELETE:** Delete all records from a database table
  - **MERGE:** UPSERT operation (insert or update)
  - **LOCK TABLE:** concurrency Control

Data Control Language

- **DCL** is short for Data Control Language which acts as an access specifier to the database.(basically to grant and revoke permissions to users in the database

  - **GRANT:** grant permissions to the user for running DML(SELECT, INSERT, DELETE,…) commands on the table

  - **REVOKE:** revoke permissions to the user for running DML(SELECT, INSERT, DELETE,…) command on the specified table

Transactional Control Language

- **TCL** is short for Transactional Control Language which acts as an manager for all types of transactional data and all transactions. Some of the command of TCL are

  - **Roll Back:** Used to cancel or Undo changes made in the database
  - **Commit:** It is used to apply or save changes in the database
  - **Save Point:** It is used to save the data on the temporary basis in the database

Thank You