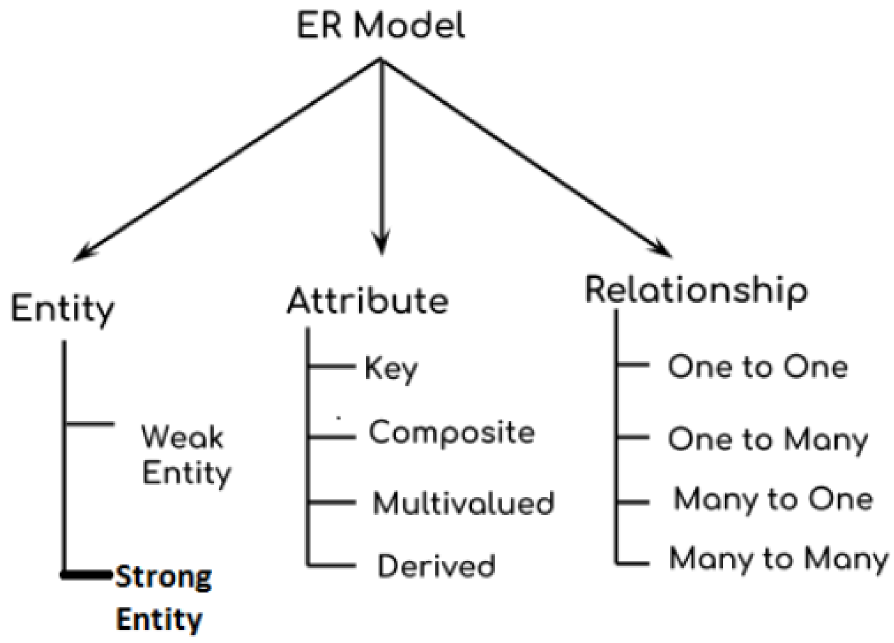
CHAPTER 2

1. What is ER- model ? Explain the Er model with an example

The **ER (Entity-Relationship) Model** is a high-level data model used in **database design**. It provides a **graphical way** to represent the **structure of a database**, showing entities (real-world objects), their attributes (properties), and the relationships among those entities.

Component of Er model:

The E-R diagram has three main components. 1) Entity 2) Attribute 3) Relationship



Components of ER Diagram

Example: ER Model

a. Entities:

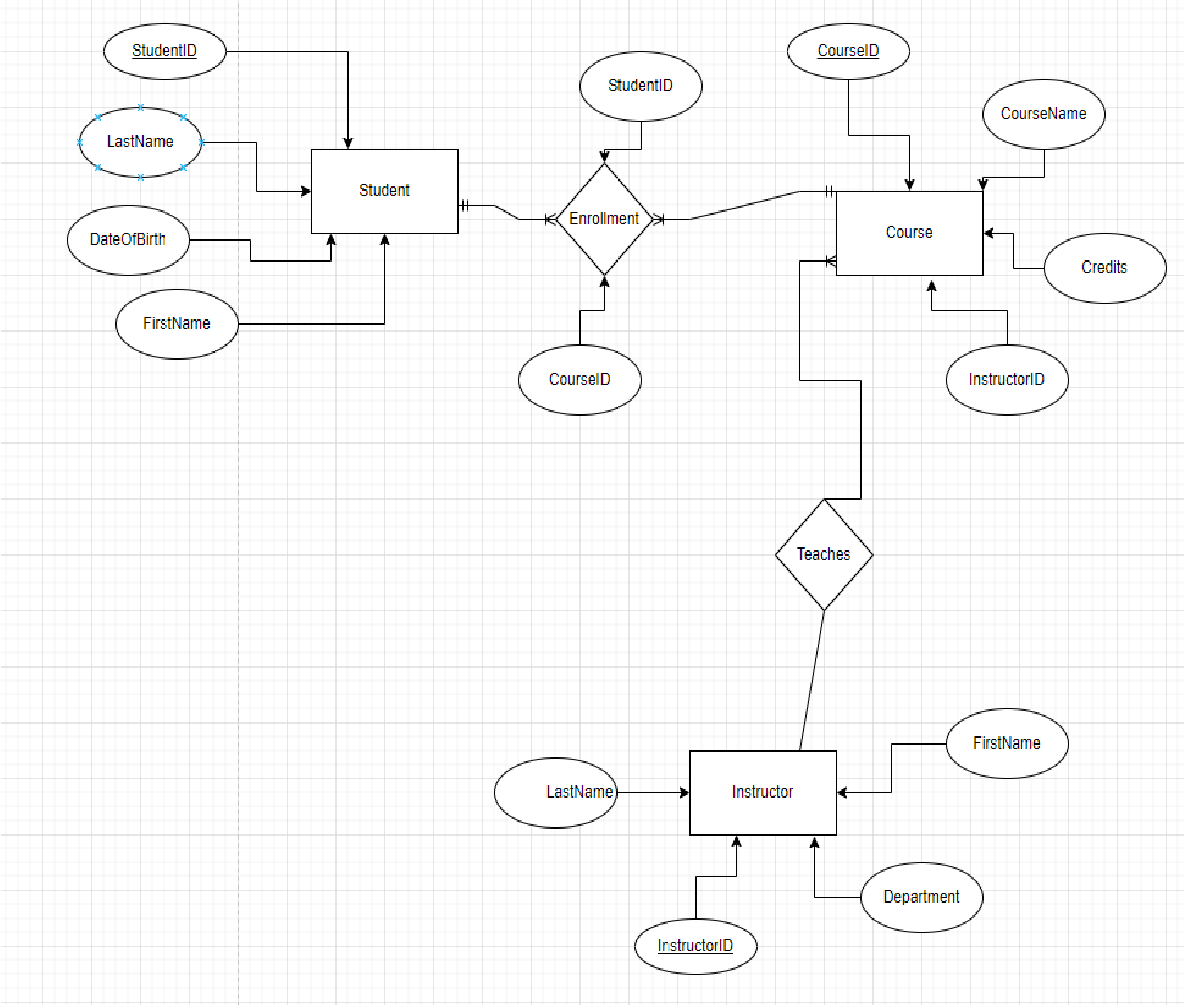
    i. Student (StudentID, FirstName, LastName, DOB)

    ii. Course (CourseID, CourseName, Credits)

b. Relationships:

i. **Enrollment**: between Student and Course

ii. **Teaches**: between Instructor and Course



Conclusion:
The ER Model is a **conceptual tool** for database design, helping to map out the structure clearly before actual implementation. It simplifies complex data by organizing it into entities, attributes, and relationships.

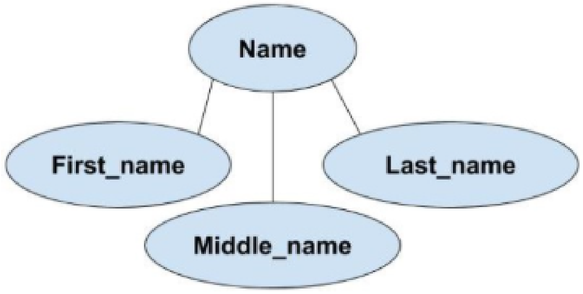
2. What are attributes? Explain the types of attributes with example

An **attribute** defines the **information** about an entity that needs to be stored in a database.

1. Simple and Composite Attributes

- **Simple Attribute**:
  - ➢ Cannot be divided into smaller parts.
  - ➢ Example: `Age`, `Gender`, `Roll No`

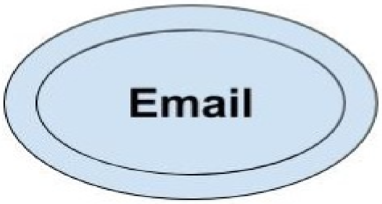
- **Composite Attribute**:
  - ➢ Can be divided into smaller parts.
  - ➢ Example: `Name` → First Name, Middle Name, Last Name
    `Address` → Street, City, District



2. Single-valued and Multi-valued Attributes

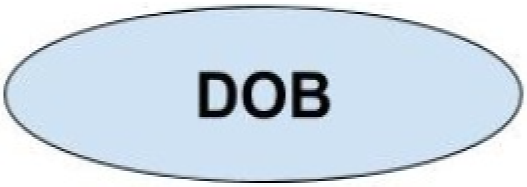
- **Single-valued Attribute**:
  - ➢ Can store **only one value**.
  - ➢ Example: `Date of Birth`, `Nationality,Section`

- **Multi-valued Attribute**:
  - ➢ Can store **more than one value**.
  - ➢ Example: `Phone Numbers`, `Email IDs`

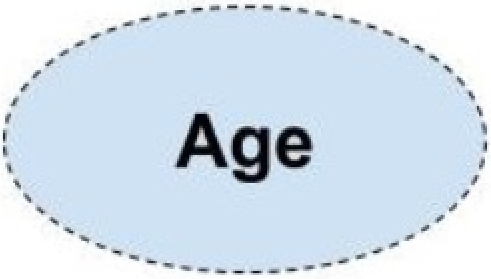


3. Stored and Derived Attributes

- **Stored Attribute**:
  ➢ Directly stored in the database.
  ➢ Example: `Date of Birth`

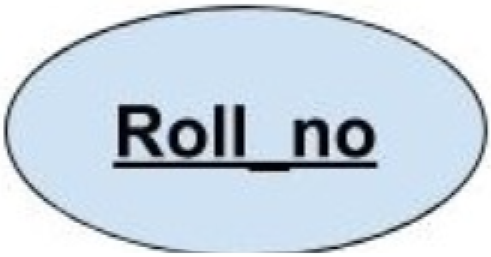


- **Derived Attribute**:
  ➢ Not stored directly, but **calculated** from stored attributes.
  ➢ Example: `Age` (calculated from Date of Birth)

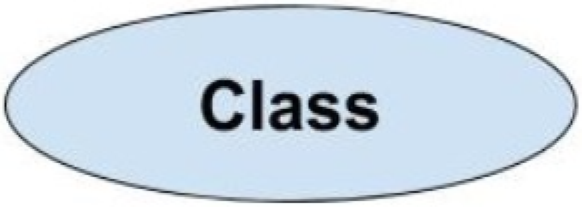


4. Key and Non-Key Attributes

- **Key Attribute**:
  ➢ Uniquely identifies each entity.
  ➢ Example: `Student ID`, `Roll No`



- **Non-Key Attribute**:
  - ➢ All other attributes except the key attribute.
  - ➢ Example: `Name`, `Age`, `Class`



Conclusion

Attributes help define and organize the **data** in a database. Understanding different types of attributes helps in designing a proper **ER Model** for any system.

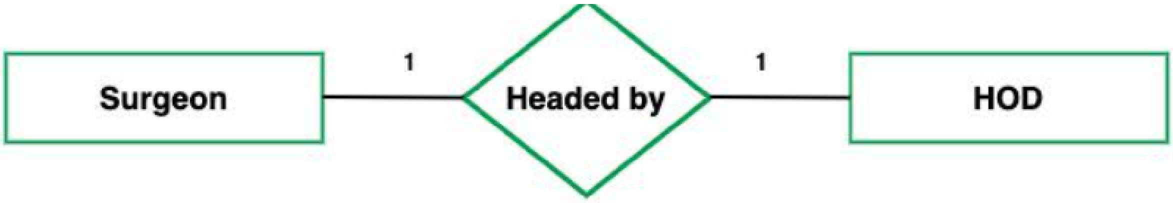
3. What is a relationship? Explain its types?

In an **ER (Entity-Relationship) Model**, a **relationship** is an **association** or **connection** between two or more **entities**.

Types of Relationships:

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

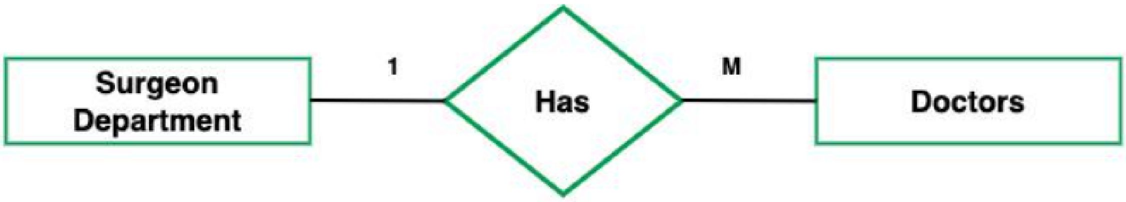
- One entity is related to only one entity of another type.
- **Example:**
  A **Surgeon** has **one HOD**



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

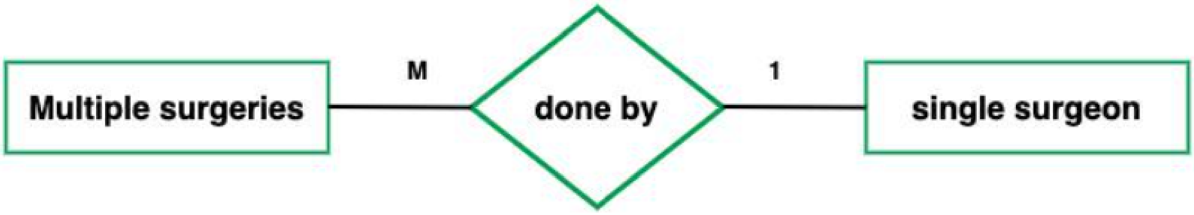
- One entity is related to **many** entities of another type.

For example:
One **surgeon department** can have many **doctors**.



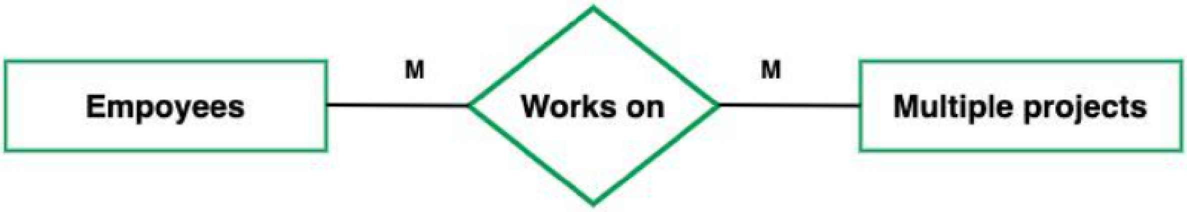
3. Many-to-One (M:1) Relationship

- Many entities are related to **one** entity.
- **Example:**
  Many surgeries can be done by a single surgeon.



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

- Many entities of one type are related to many entities of another type.
- **Example:**
  Many employees works on multiple projects



Relationships help to define how different **entities** are connected in a database. Understanding the types of relationships is important to design **correct and efficient database structures**

4. What is mapping cardinalities? Discuss with an example

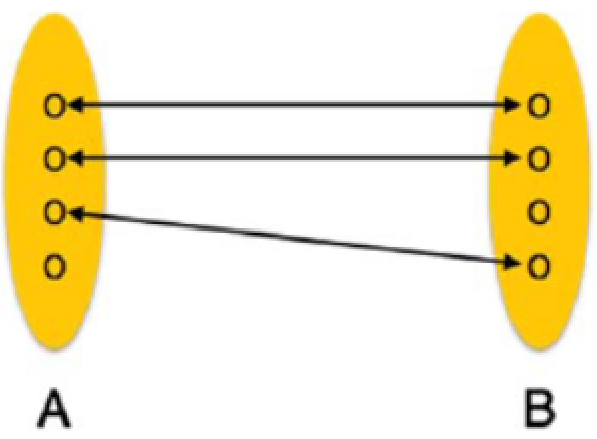
Mapping cardinality defines how many entities in one entity set are related to entities in another set through a relationship.

It tells **how many instances** of entity A are related to how many instances of entity B.

Types of Mapping Cardinalities:

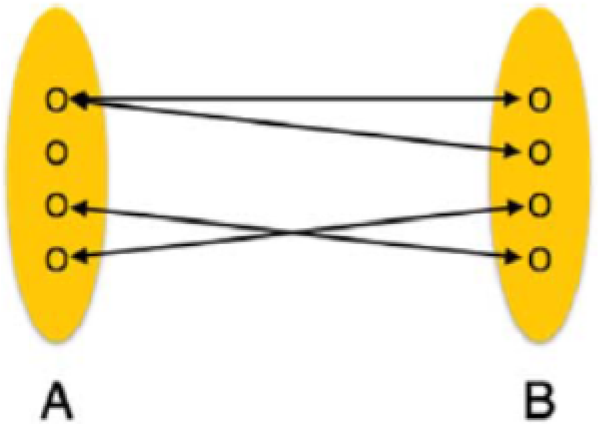
1️⃣One-to-One (1:1)

One entity from set A is related to **only one** entity from set B, and vice versa.



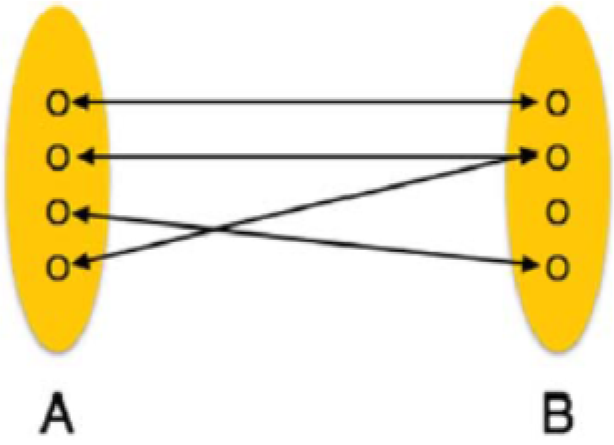
2️⃣One-to-Many (1:N)

One entity from set A is related to **multiple entities** in set B. But an entity from B is related to **only one** in A.



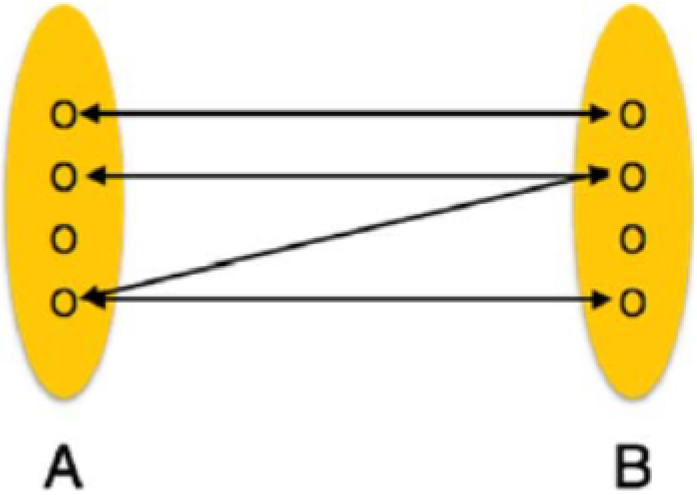
3️⃣Many-to-One (N:1)

Many entities from set A are related to **one** entity in B. But one entity in B is related to **many** in A.



4️⃣ Many-to-Many (M:N)

Entities from A can be related to **many** in B, and entities in B can be related to **many** in A.



5. **What are keys? Write different types of keys**

Keys are used in a database to uniquely identify rows (records) in a table and to build relationships between different tables.

1. Primary Key

- A **Primary Key** is a column (or set of columns) in a table that **uniquely identifies each row**.

- It **cannot be null** and **cannot be duplicated**.

Example:
In a student table, `roll_no` is a primary key.

2. Super Key

- A **Super Key** is any set of one or more attributes (columns) that can **uniquely identify a record**.
- It can include **extra unnecessary attributes**.

Example:

- `roll_no`
- `roll_no + st_name + st_address`
  (Both are super keys)

3. Candidate Key

- A **Candidate Key** is a **minimal super key** – it has **no extra attributes**.
- There can be **multiple candidate keys** in a table.

Example:

- `roll_no`
- `phone + email`
  (Both are candidate keys)

4. Foreign Key

- A **Foreign Key** is a column in one table that **refers to the Primary Key** of another table.
- It is used to **create relationships between tables**.

Example:
`student_id` in the `Marks` table refers to `student_id` in the `Student` table.

5. Composite Key

- A **Composite Key** is a key made of **two or more columns** that **together uniquely identify** a row.
- Each column individually **may not be unique**, but together they are.

Example:
`student_id + subject_code` in a `Marks` table.

6. Secondary (Alternative) Key

- A **Secondary Key** is a **candidate key** that is **not chosen as the primary key**.
- It can still uniquely identify records.

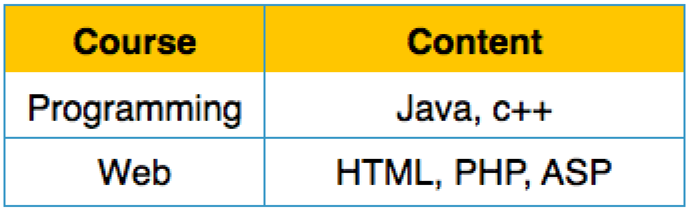
Example:
If `roll_no` is the primary key, then `phone + email` can be a secondary key.

CHAPTER 4

1. Explain Normalization process with examples.

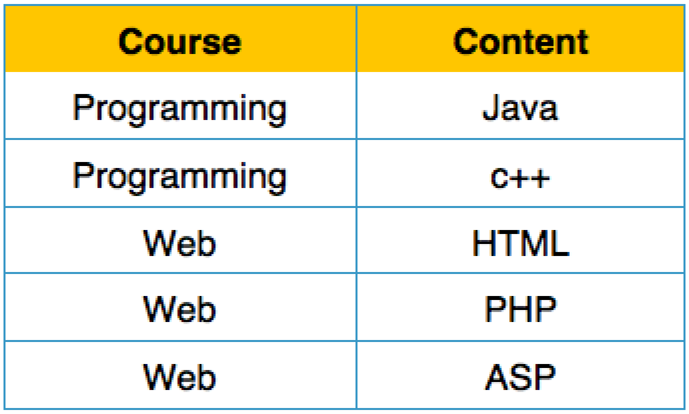
Normalization is a step by step process of removing different kinds of redundancy and normally at each step.

1. First normal form
   This rule defines that all the attributes in a relation must have atomic domains.
   Suppose a table which does not contain atomic values



Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

   To convert the table in first normal form



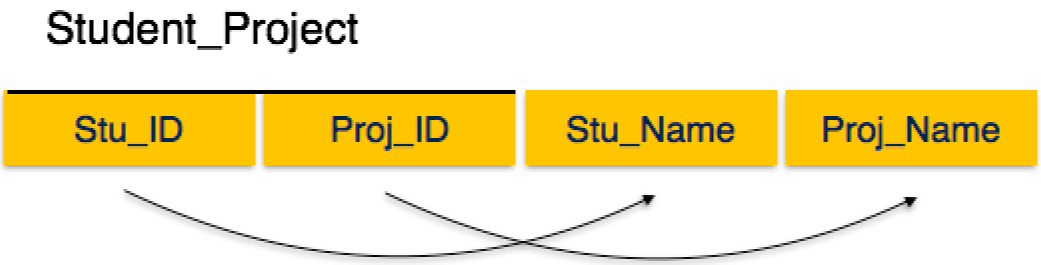
Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

2. Second normal form

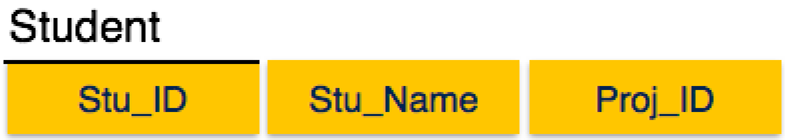
Every non-prime attribute should be fully functionally dependent on prime key attribute as well as it is in first normal form is called second normal form.

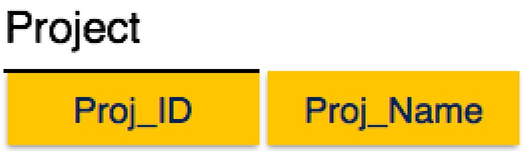
A relation is in 2NF , if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.

Suppose a table which contains partial dependency



To convert above table in second normal form





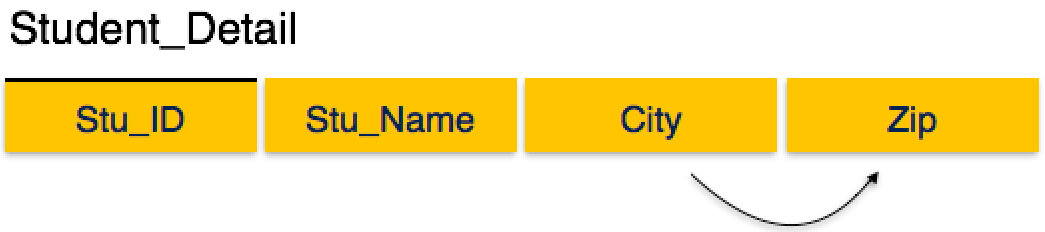
3. Third normal form

No non-prime attribute is transitively dependent on prime key attribute as well as it is in second normal form is called third normalization.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency A –> B

a. A is a super key.
b. B is a prime attribute (each element of B is part of some candidate key).

Suppose a table which transitive dependency



Student_Detail

Stu_ID	Stu_Name	City	Zip

To convert above relation in third normal form

Student_Detail

Stu_ID	Stu_Name	Zip

ZipCodes

Zip	City

CHAPTER 6

1. Describe ACID properties

A **transaction** is a set of operations performed on a database. To ensure **data accuracy, reliability, and integrity**, every transaction must follow the **ACID properties**:

1️⃣Atomicity

- A transaction is treated as a **single unit**.
- **All operations must complete**, or **none should happen**.
- If one part fails, the entire transaction is **rolled back**.

Example: Money transfer — either both debit and credit happen, or neither.

2️⃣Consistency

- A transaction must bring the database from one **valid state to another**.
- The database remains **logically correct** before and after the transaction.

- If a transaction violates consistency, it will be **rejected or rolled back**.

3️⃣Isolation

- Multiple transactions can run at the same time but must not **interfere** with each other.

- The final result must be the **same as if they were run one by one** (serially).

Example: Two people booking the same train seat — isolation prevents double booking.

4️⃣Durability

- Once a transaction is **committed**, the changes are **permanent**.

- Even if there is a **system crash or power failure**, the data remains safe.

CHAPTER 4

1. **`Explain the different types of languages used in sql`**

The different types of languages are:

1. DDL – Data Definition Language

Purpose: Used to define or modify the structure of database objects such as tables, schemas, indexes, etc.

Command	Description
`CREATE`	Creates a new table, database, view, or index.
`ALTER`	Modifies an existing database object (e.g., adding a column to a table).
`DROP`	Deletes an existing table, view, or other database object.
`TRUNCATE`	Deletes all records from a table but not the structure.
`RENAME`	Renames a database object.

2. DML – Data Manipulation Language

Purpose: Used for manipulating (inserting, updating, deleting) the data in tables.

Command	Description
`INSERT`	Adds new records into a table.
`UPDATE`	Modifies existing records.
`DELETE`	Removes existing records.
`SELECT`	Retrieves data from one or more tables (sometimes grouped under DQL).

3. DCL – Data Control Language

Purpose: Used to control access to data in the database (permissions and security).

Command	Description
`GRANT`	Gives user access privileges to the database.

REVOKE	Removes user access privileges.

SQL JOIN

SQL **JOIN** is used to combine rows from two or more tables based on a related column.

Example Tables

Students Table:

student_id	name	class_id
1	Sita	101
2	Ram	102
3	Gita	101

Classes Table:

class_id	class_name
101	Science
102	Management

---

1. INNER JOIN

Returns only the matching rows from both tables.

```
SELECT Students.name, Classes.class_name
FROM Students
INNER JOIN Classes ON Students.class_id = Classes.class_id;
```

Result:

name	class_name
Sita	Science
Ram	Management
Gita	Science

2. LEFT JOIN

Returns all rows from the left table (`Students`), and matched rows from the right table (`Classes`).

```
SELECT Students.name, Classes.class_name
FROM Students
LEFT JOIN Classes ON Students.class_id = Classes.class_id;
```

If a student had no class assigned, their name would still appear with `NULL` in `class_name`.

3. RIGHT JOIN

Returns all rows from the right table (`Classes`), and matched rows from the left table (`Students`).

```
SELECT Students.name, Classes.class_name
FROM Students
RIGHT JOIN Classes ON Students.class_id = Classes.class_id;
```

Useful to show all classes, even those without students.

4. FULL OUTER JOIN *(Not supported in some DBMS like MySQL)*

Returns all rows when there is a match in one of the tables.

```
SELECT Students.name, Classes.class_name
FROM Students
FULL OUTER JOIN Classes ON Students.class_id = Classes.class_id;
```