Lecture 4 – EERD

CPE241 – Database Systems

3 Febuary 2026

Dr. Jaturon Harnsomburana
Dr. Piyanit Ua-areemitr

Department of Computer Engineering
King Mongkut's University of Technology Thonburi

Recap

ERD

- Basic
- UML class diagram

CRUD

- create, select, alter, delete, drop, truncate

Constraints

- PK,FK
- Not Null, default

Milestones

More ERD + Enhanced ERD

- Weak Entity/Relationship
- Generalization/Specialization
- Aggregation/Composition

ERD to Relational Database Tables

Examples

ERD Again!

Basic of ERD

- Entity
- Relationship
- Attribute
- Cardinality
- Keys: Primary and Foreign

ERD to Relational Database Tables

Advanced ERD & Enhanced ERD

- Weak Entity/Relationship
- Generalization/Specialization
- Aggregation/Composition
- Disjoint/Overlapping
- Total/Partial

ERD to Relational Database

1. Convert Entities to Tables

    PK and attributes

2. Convert Relationships to Tables or Foreign Keys

    M2M relationship turns into table. 121 can be combined. 12M will store pk of 1 side to M table

3. Convert Attributes

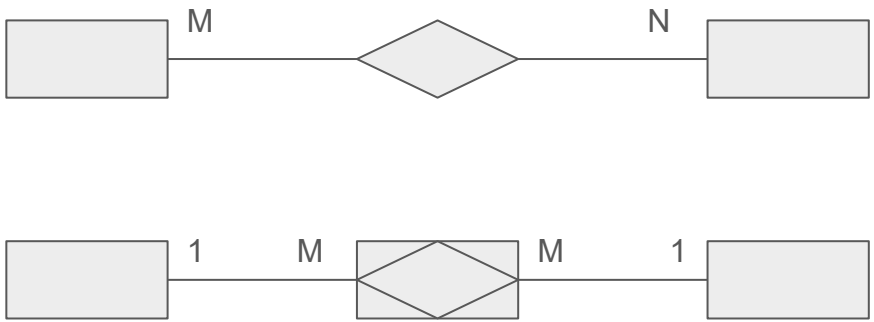
    Multivalue will get a table while derived can be omitted.

4. Convert Weak Entities

5. Enforce Constraints

6. Optimize with Indexing and Normalization

M:N to 1:M-M:1



N-ary Relationship

An **n-ary** relationship is a relationship that involves three or more entities. Instead of simple binary relationships (between two entities), n-ary relationships capture complex interactions among multiple entities.

A unary relationship involves itself.

A ternary relationship (3-ary) involves three entities.

A quaternary relationship (4-ary) involves four entities.

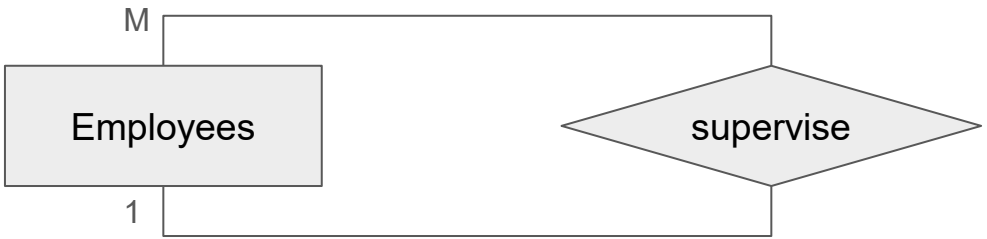
Unary Example

The relationship connects itself: Employees, Customers, Persons

Supervisors(Employees) supervise Employees

New customers refer Customers

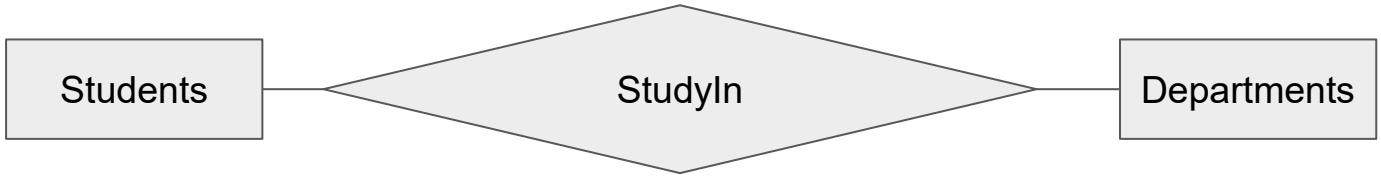
Parents and Child can be Persons



Employees(EmpID, SupervisID)

Binary Example

The relationship connects two entities: Students and Department



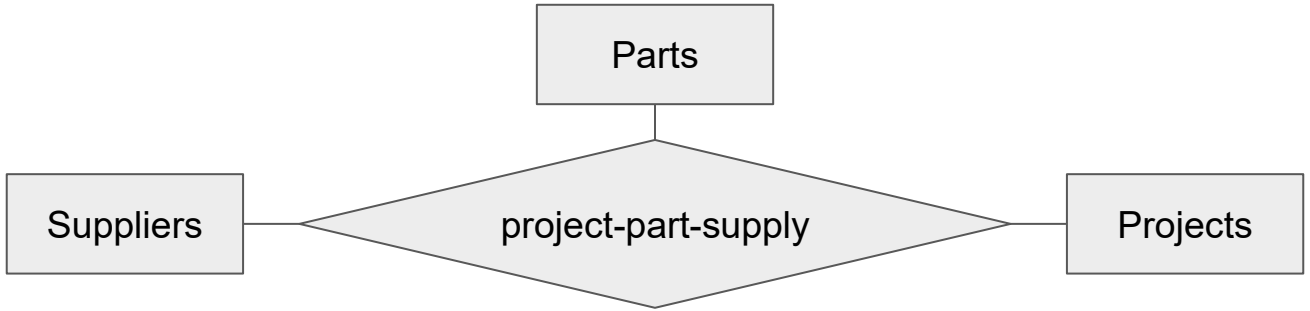
Departments(DepID)
Students(StuID,DepID)

Ternary Example

The relationship connects three entities: Supplier, Part, and Project.

Supplies (SupplierID INT, PartID INT, ProjectID INT, Quantity INT)

If we break this into three binary relationships (Supplier-Part, Part-Project, Supplier-Project), we lose key information (e.g., which supplier provided which part for a specific project).



Quaternary Example

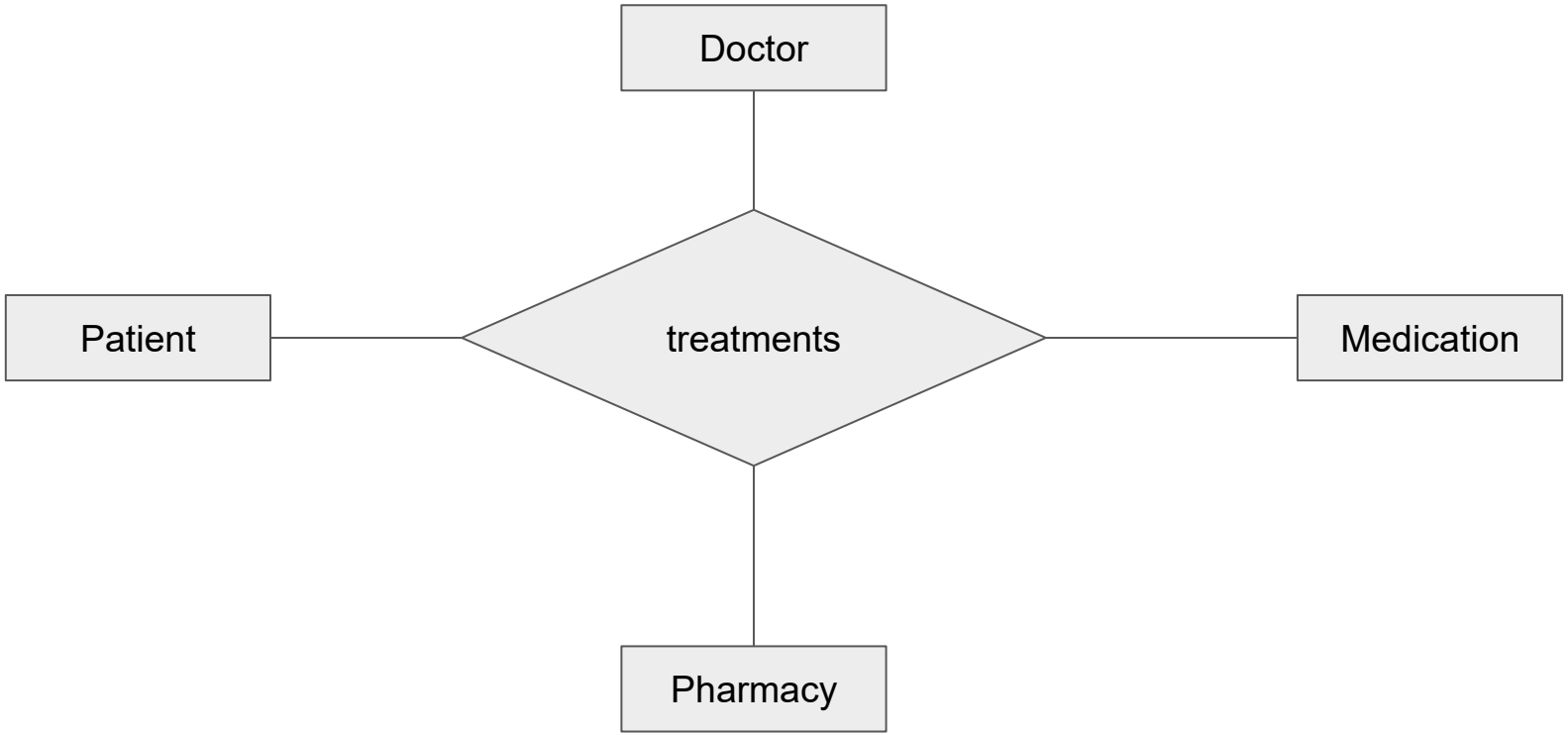
A quaternary relationship (4-ary) involves four entities.

A single prescription links Doctor, Patient, Medication, and Pharmacy.

Prescribes (DoctorID INT, PatientID INT, MedicationID INT, PharmacyID INT,

Dosage VARCHAR(50), PrescriptionDate DATE)

If we break this into smaller binary or ternary relationships, we lose the full prescription context (e.g., which doctor prescribed which medication for which patient from which pharmacy).


Doctor
Patient
treatments
Medication
Pharmacy

Dependency and Weak Entity

Dependency refers to the reliance of one entity on another for its identification or existence. It is commonly seen in weak entities, which cannot exist without a strong (or owner) entity.

Existence Dependency - has primary key (order-order items-product)

Identification Dependency - uses parent primary key (employee-dependent)

Weak Entity

A weak entity is an entity that cannot be uniquely identified by itself and depends on a strong entity for identification.

Characteristics of a Weak Entity:

No Primary Key: It does not have a sufficient primary key.

Uses Foreign Key as Part of Primary Key: The primary key of the strong entity is included in the weak entity.

Requires an Identifying Relationship: Connected to a strong entity through a double-lined relationship (Chen notation).

Has Total Participation: Every weak entity must be associated with a strong entity.

Examples

an employee that can have dependents (insurance and other type too)

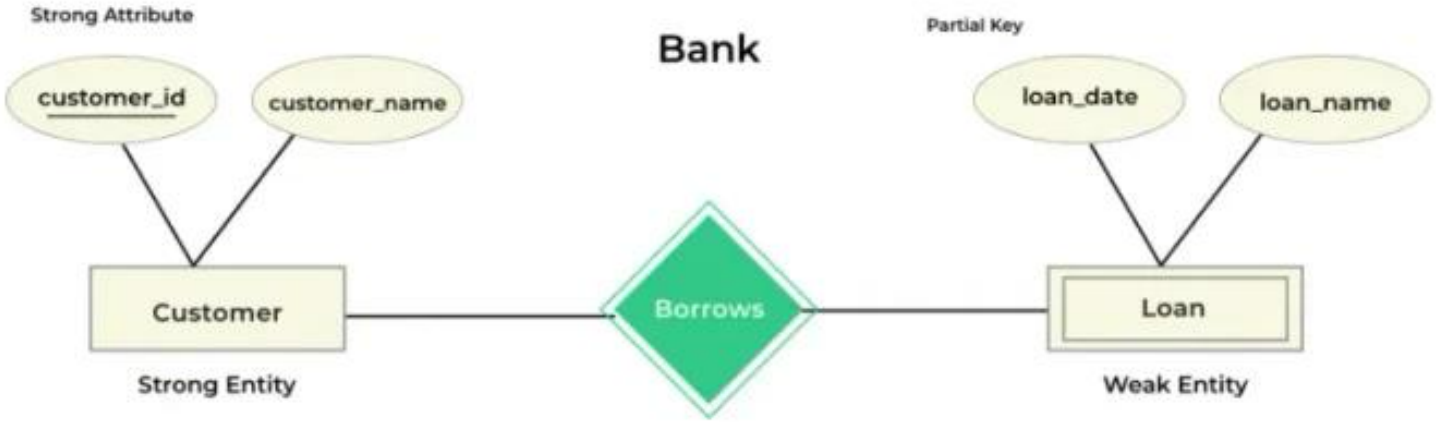
a mother that can have children

Line items in invoices, orders, menus

Classes or courses in the current semester depend on course catalog

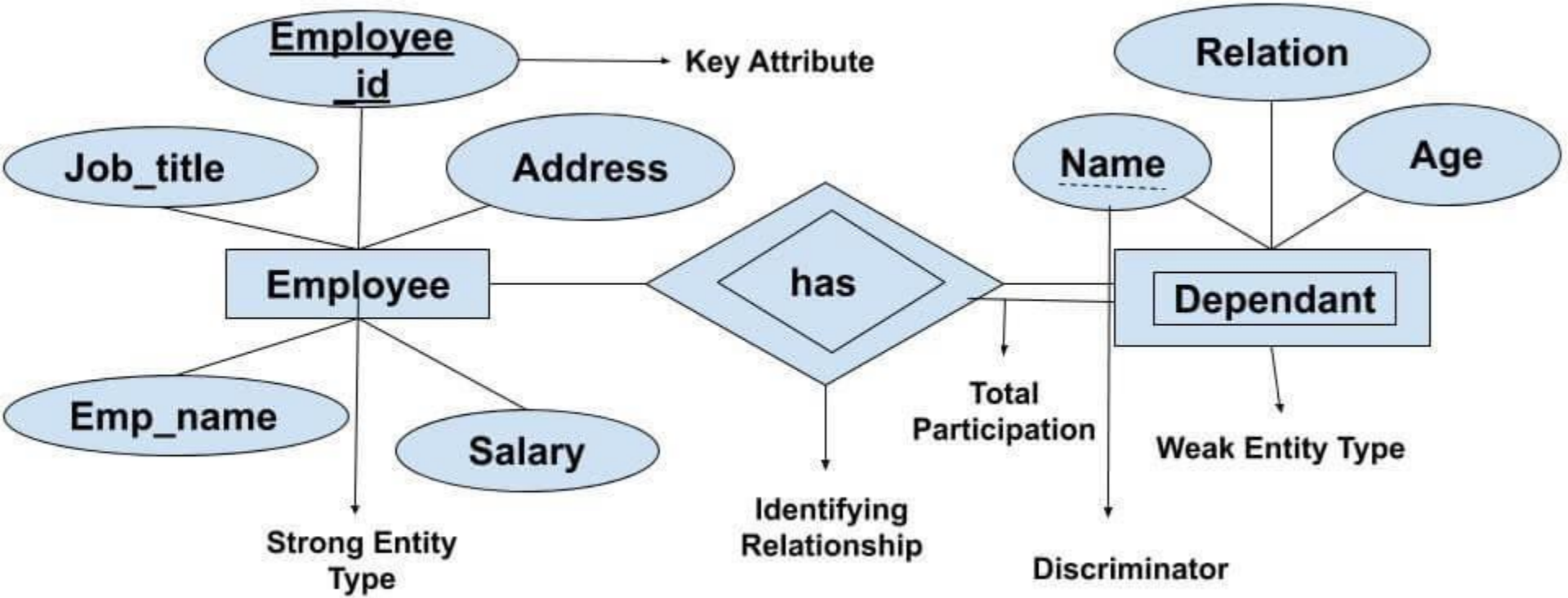
Accounts of a bank only exists if the bank exist

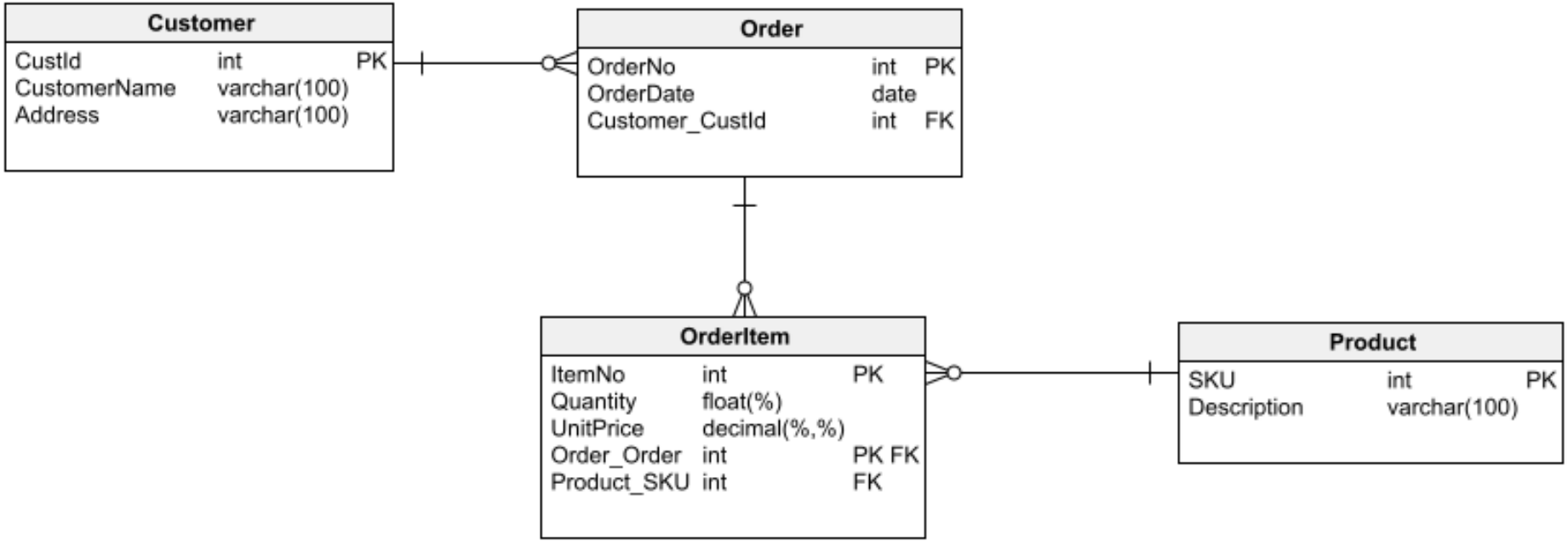
Weak Entity



Initialisation

Strong Entities	Customer
Weak Entities	Loan
Strong Attributes	customer_id
Partial Key	loan_name


Employee id
Key Attribute
Relation
Job_title
Address
Name
Age
Employee
has
Dependant
Emp_name
Salary
Total Participation
Weak Entity Type
Strong Entity Type
Identifying Relationship
Discriminator


Customer
CustId int PK
CustomerName varchar(100)
Address varchar(100)
Order
OrderNo int PK
OrderDate date
Customer_CustId int FK
OrderItem
ItemNo int PK
Quantity float(%)
UnitPrice decimal(%,%)
Order_Order int PK FK
Product_SKU int FK
Product
SKU int PK
Description varchar(100)

Vertabelo

superclass/subclass

A superclass is a higher-level entity that contains common attributes shared by multiple lower-level entities (subclasses).

- Represents a general entity (parent).
- Has attributes common to all its subclasses.
- Can participate in relationships shared by all subclasses.

A subclass is a lower-level entity that inherits attributes from the superclass while also having its own unique attributes.

- Represents a specialized entity (child).
- Inherits attributes from the superclass but can have additional attributes.
- Can participate in relationships unique to itself.

This represents inheritance concept

Generalization/Specialization

Generalization: Combining multiple entities into a higher-level superclass
Specialization: Breaking down a superclass into subclasses based on distinguishing features

(Person → Student, Professor).

(Employees → Secretary, Engineer, Technician).

(Shape → Circle, Triangle, Rectangle).

Categorization

Categorization is a type of specialization where a single subclass can belong to multiple superclasses.

A Person can be categorized into different roles:

- Employee (works in a company)
- Student (studies at a university)
- Athlete (plays for a sports team)

If an entity belongs to more than one superclass, it falls under categorization.

- Superclasses (Employee, Student, Athlete) connect to a subclass (Person) through a triangle.
- A union constraint is used to indicate that the subclass is derived from multiple superclasses.

Examples

Accounts can be saving, current, and fix

Employee can be secretary, administrator, technician, engineer

Vehicle can be motorbike, car, truck, boat

Disjoint Constraint

Disjoint Constraint (Exclusive)

Definition: An entity can belong to only one subclass at a time.

Notation: “d” inside the specialization/generalization triangle.

Example:

A Vehicle can be a Car or a Truck, but not both at the same time.

Vehicle → {Car, Truck} (Disjoint rule applies)

EERD Representation:

A single entity cannot be classified into multiple subclasses.

If an entity belongs to one subclass, it cannot belong to another.

Overlapping Constraint (Inclusive)

Definition: An entity can belong to multiple subclasses.

Notation: “o” inside the specialization/generalization triangle.

Example:

A Person can be both a Student and an Employee at the same time.

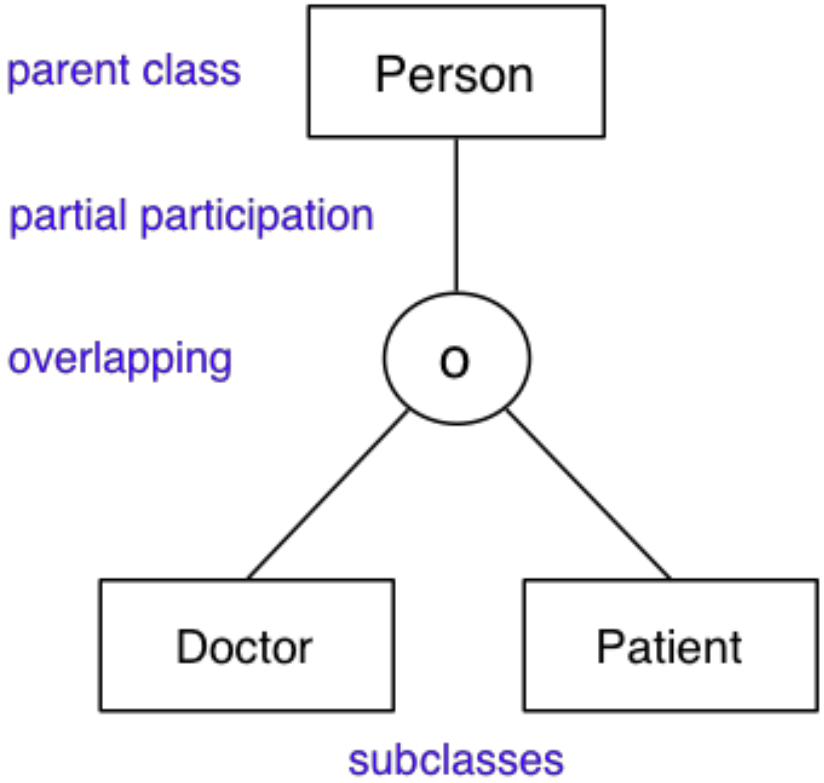
Person → {Student, Employee} (Overlapping rule applies)

EERD Representation:

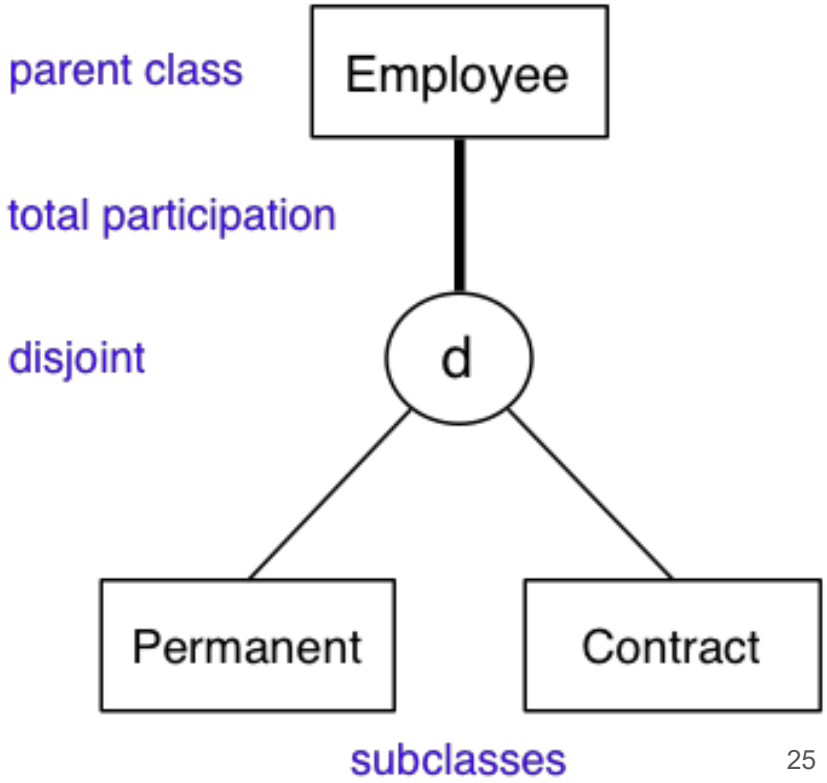
A single entity can belong to multiple subclasses simultaneously.

If an entity belongs to one subclass, it can also belong to others.

A person may be a doctor and/or may be a patient or may be neither



Every employee is either a permanent employee or works under a contract



Total vs Partial Specialization

Total: Every entity in the superclass must belong to at least one subclass.

Notation: Double line between the superclass and subclasses.

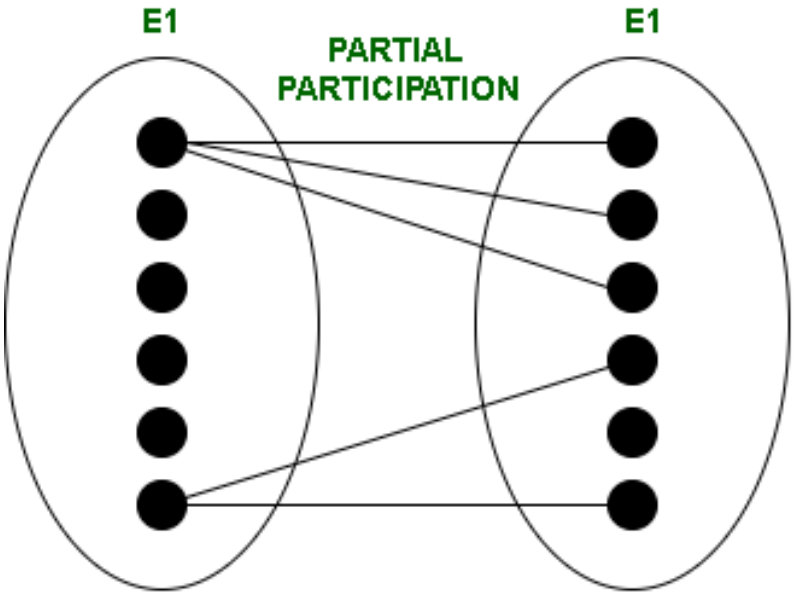
Every Employee must be either a Manager or a Technician.

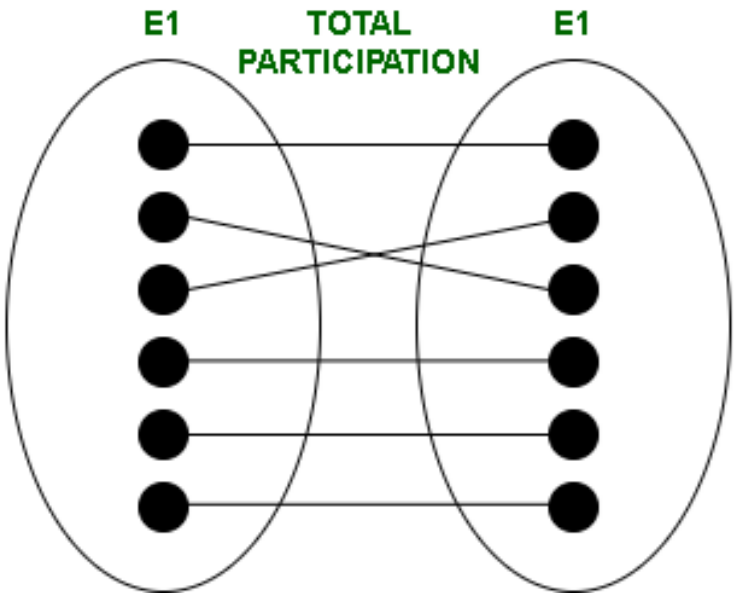
There is no Employee without a subclass.

Partial: Some entities in the superclass may not belong to any subclass.

Notation: Single line between the superclass and subclasses.

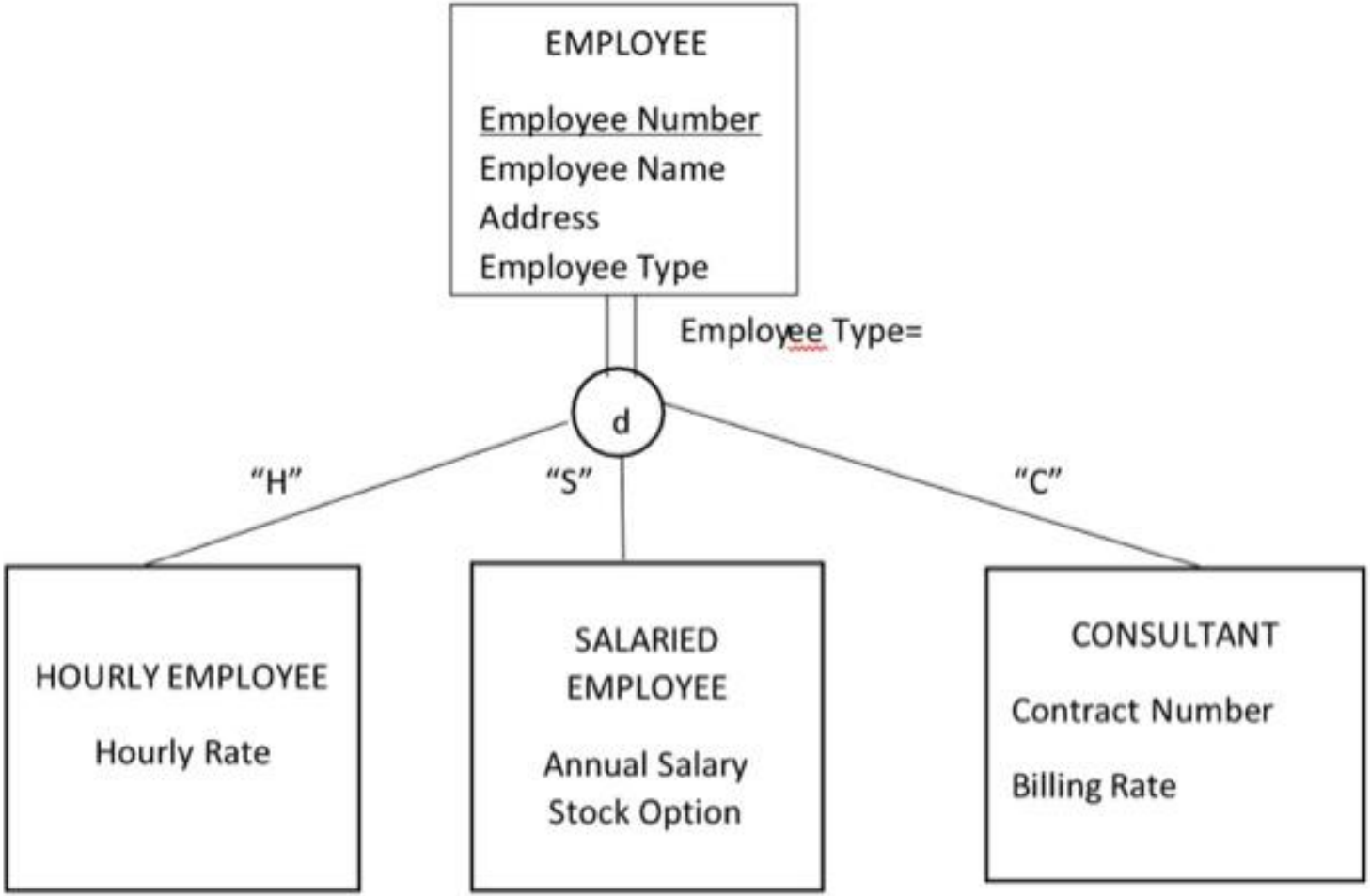
A Person may be a Student or an Employee, but some People belong to neither subclass.


E1
PARTIAL
PARTICIPATION
E1


E1
TOTAL
PARTICIPATION
E1

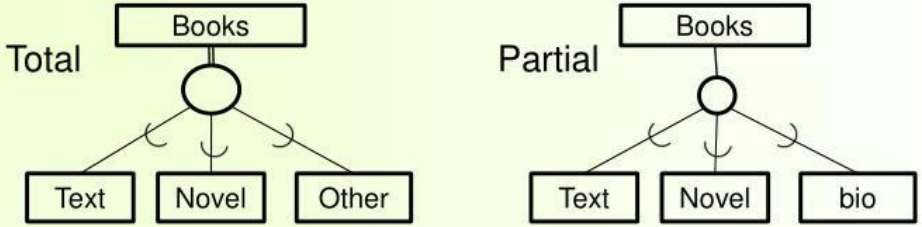
Constraint	Type	Meaning	Example
Categorization	Superclass-subclass	A subclass belongs to multiple superclasses	A **Person** can be an **Employee, Student, or Athlete**
Disjoint (d)	Specialization	An entity belongs to **only one** subclass	A **Vehicle** is either a **Car** or a **Truck**, but not both
Overlapping (o)	Specialization	An entity can belong to **multiple** subclasses	A **Person** can be **both a Student and an Employee**
Total (Double Line)	Specialization	Every superclass entity **must** belong to a subclass	Every **Employee** is a **Manager or Technician**
Partial (Single Line)	Specialization	Some superclass entities **do not need** to belong to a subclass	A **Person** may be a **Student or Employee**, but some are neither

(a)

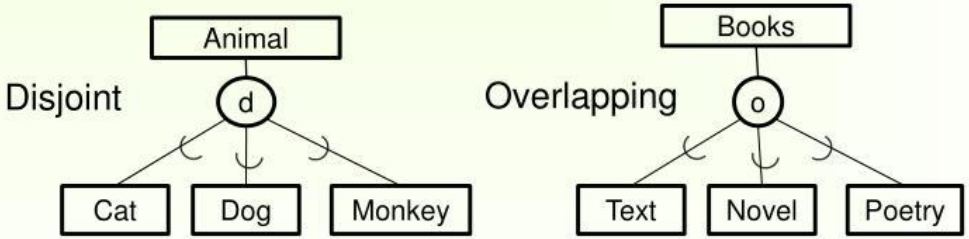

EMPLOYEE
Employee Number
Employee Name
Address
Employee Type
Employee Type=
d
"H"
"S"
"C"
HOURLY EMPLOYEE
Hourly Rate
SALARIED EMPLOYEE
Annual Salary
Stock Option
CONSULTANT
Contract Number
Billing Rate

ER Diagram Notation (cont.)

- Total or partial specialization:



- Disjoint or overlapping specialization:



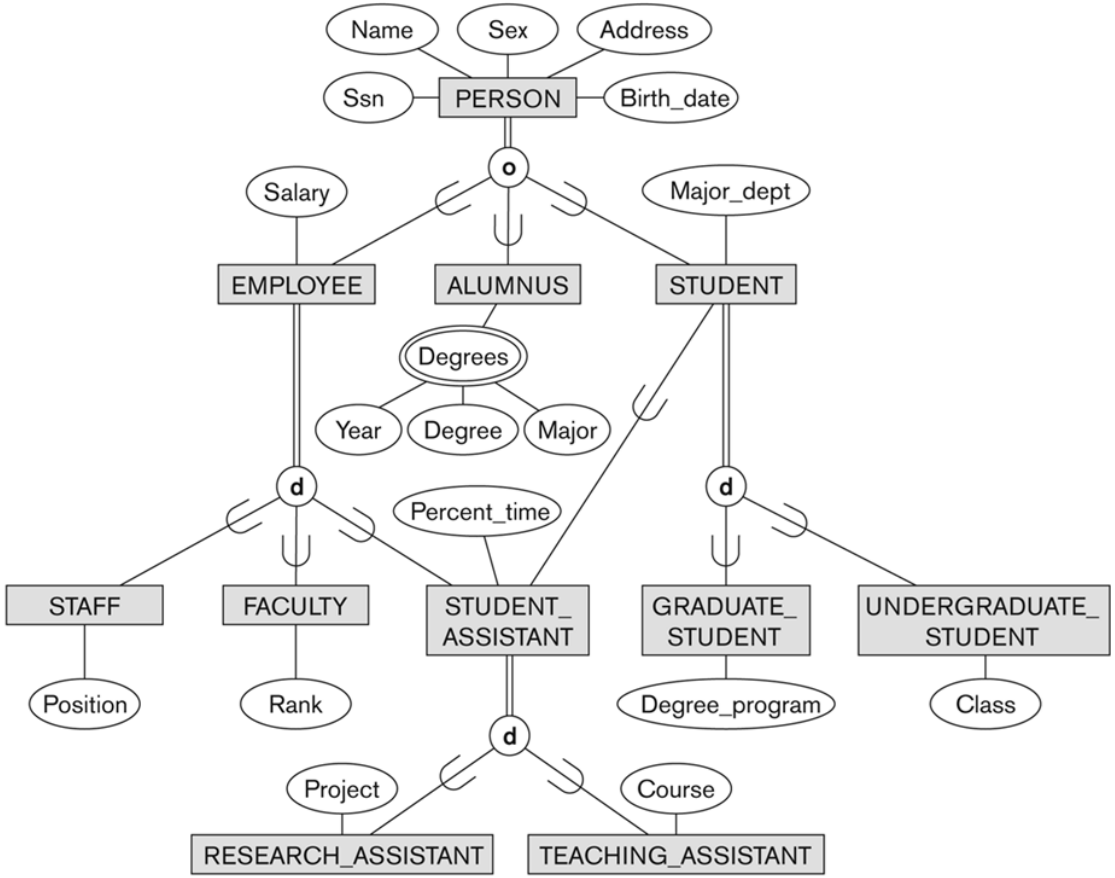


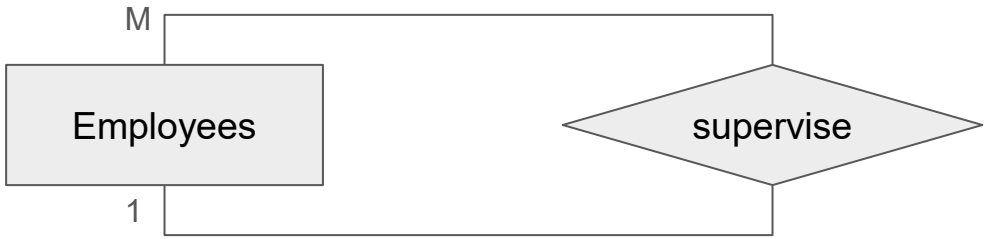
Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

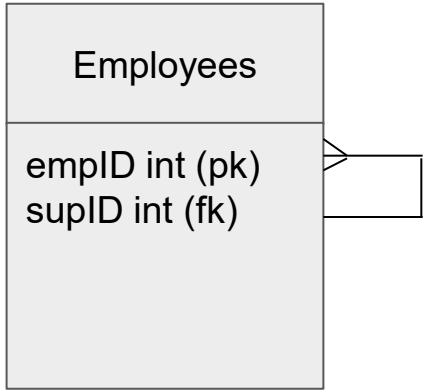
Break Time!

ERD to Relational Database Tables

Now we want to see this clear example how.

Unary



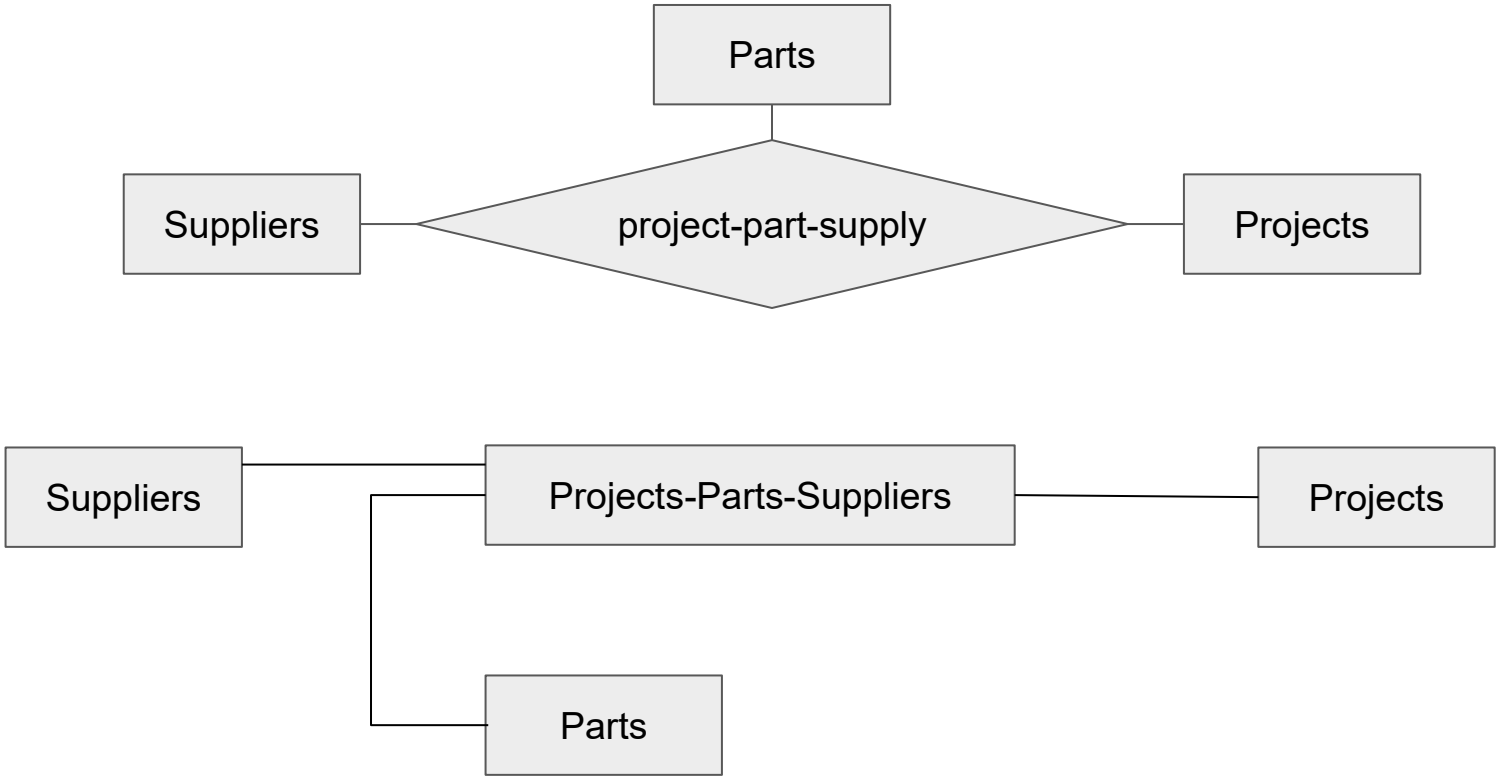



```
create table Employees {
        empID    integer primary key,
        supID    integer references Employees(empID),
}
```

Binary


```
create table Departments { depID   int primary key }
create table Students {
        stuID     int primary key,
        depID     int references Departments(depID)
}
```

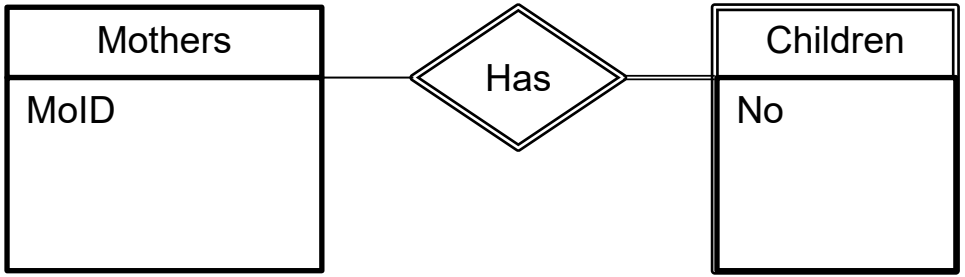
Ternary


Parts
Suppliers
project-part-supply
Projects
Suppliers
Projects-Parts-Suppliers
Projects
Parts

Ternary (cont'd)

```
create table Projects(prjID int primary key)
create table Parts(prtID int primary key)
create table Suppliers(supID int primary key)
create table Projects-Parts-Suppliers(
        prjID int references Projects(prjID),
        prtID int references Parts(prtID),
        supID int references Suppliers(supID),
        quanti int,
        primary key(prjID,prtID,supID)
)
```

Weak Entity and Relation




```
create table Mothers { MoID        int primary key }
create table Children {
            MoID        int references Mothers(MoID),
            No          int,
            primary key (MoID,No)
}
```

Requirements

Definition: A requirement is a clear statement of a need or condition that a system must satisfy to solve a problem or achieve a goal.

Why Requirements Matter

- Align stakeholders on expectations
- Guide system design and development
- Provide a basis for testing and validation

1.Business Requirements

What the organization wants to achieve

- Describe **business goals or value**
- High-level, technology-independent
- Define **why** the system is needed

Example

"Reduce customer service response time to improve customer satisfaction."

2. User Requirements

What users need to do to accomplish their work

- Describe **user goals and interactions**
- Focus on user tasks and expectations
- Often written from the user's perspective

Example

"Customer service agents need to view customer history in one screen."

3. System Requirements

What the system must do to support users and business

- Detailed, precise, and testable
- Define **system behavior, functions, and constraints**
- Used by designers and developers

Example

"The system shall display customer profile, past orders, and support tickets on a single dashboard within 2 seconds."

Good Requirements

Clear – easy to understand
Specific – no ambiguity
Testable – can be verified
Necessary – adds real value

Examples of Bad requirements

❌ **Too Vague**

- *"The system should be fast."*

❌ **Ambiguous**

- *"The system must support many users."*

❌ **Solution-Oriented**

- *"Use a mobile app to manage customer orders."*

❌ **Unverifiable**

- *"The system should be user-friendly."*

❌ **Incomplete**

- *"Users can generate reports."*

❌ **Combined / Overloaded**

- *"The system shall save data securely and generate reports and send emails."*

Example of changes to Good

✅ **Clear and Measurable**
- *"The system shall respond to user requests within 2 seconds for 95% of transactions."*

✅ **Specific**
- *"The system shall support at least 1,000 concurrent users."*

✅ **Problem-Focused**
- *"Customers need a way to place and track orders using their mobile devices."*

✅ **Testable**
- *"New users shall be able to complete registration without training in under 5 minutes."*

✅ **Complete**
- *"The system shall allow users to generate monthly sales reports in PDF format."*

✅ **Atomic (One Requirement per Statement)**
- *"The system shall store customer data using encryption."*
- *"The system shall generate sales reports."*
- *"The system shall send notification emails."*

Let’s Rock ‘n’ Roll

Now we got to implement this in our lab!