McCormick

Northwestern Engineering

# EECS 495 – Introduction to Database Systems

*Lecture – 2*

*Prepared by* **Mas-ud Hussain**

Part of the materials are supplied by Prof. Peter Scheuermann & Prof. Goce Trajcevski

# TA Office Hours

- Will start from **9/25 (Friday)**

**Tuesday:** 5:00 – 6:00 pm

**Wednesday:** 4:00 – 5:00 pm

**Friday:** 2:30 – 3:30 pm

(and/or, *appointment via email*)

**Location:** Tech L580

**Email:** mmas-ud.hussain@u.northwestern.edu
mmasud.hussain@gmail.com

- There will be additional grader(s) to help in grading.

# Last Week...

- Introduction
- File Systems vs. DBMS
- Core Database Functionalities
  - Data Independence
  - Declarative Querying
  - Transactions (contd..)
- Metadata (contd..)

# Transactions: *An Execution of a DB Program*

- Key concept is *transaction*, which is an *atomic* sequence of database actions.
  - Insert
  - Delete
  - Update
  - Reads/Writes
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.

# Transactions: *Example of Inconsistency*

**Instructors**

DELETE ID 1

| ID | Name | Designation |
|---|---|---|
| ~~1~~ | ~~Peter Scheuermann~~ | ~~Professor~~ |
| 2 | Douglas Downey | Associate Professor |
| ….. | ……. | ……. |

**Courses**

| ID | Name | Instructor ID |
|---|---|---|
| EECS 495 | Intro to DB | 1 |
| EECS 317 | Data Management | 2 |
| EECS 230 | Intro to Programming | 3 |
| ….. | …… | …… |

Inconsistency

# Transactions: *Example of Inconsistency*

**Account**

| ID | Type | Balance | Credit Line | Total Cashback |
|---|---|---|---|---|
| 1 | Gold | **600** | 2000 | **21** |
| 2 | Premium | 220 | 4000 | 12 |
| ….. | ……. | ……. | ……. | …….. |

**Transaction**

| ID | Account ID | Amount | Cashback (5%) |
|---|---|---|---|
| 1 | 2 | 50 | 2.5 |
| 2 | 1 | 40 | 2 |
| ….. | …… | …… | …… |

INSERT new transaction

| | | | |
|---|---|---|---|
| **3** | **1** | **100** | **6** |

# Transactions: *Maintaining Consistency*

- Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints. (1st example)
- Beyond this, the DBMS does not really understand the *semantics* of the data. (e.g., it does not understand how the interest on a bank account is computed). (2nd example)
- Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

# Queries in a DBMS

- DBMS provides a specialized language for accessing data:
  - **Query Language**
- Query language can be further classified into:
  - DML - Data Manipulation Language (Insert/Delete/Update)
  - DDL - Data Definition Language (Create Tables)
  - DCL - Data Control Language (Procedural Tasks)

# Queries in a DBMS

- Standard for relational DBMS: **SQL**
  - Based on formal query languages: Relational Algebra and Relational Calculus
- Queries are evaluated as efficient as possible
  - Huge influence of physical design

# Queries: “What”, not “How”

- It is convenient to indicate declaratively *what* information is needed, and leave it to the system to work out *how* to process through the data to extract what you.
  - Just like programming, e.g., if..else, for, while, printf, etc. keywords to declare what to do.
- A query language is based on declarative logic.
  - SELECT name FROM customer WHERE id=1 (SQL)

# DB System Requirements

= 


= Petabytes of data

- **High Availability**: on-line => must be operational while enterprise is functioning (Page not loading!)
- **High Reliability**: correctly tracks state, does not lose data, controlled concurrency (Lose a Facebook photo)
- **High Throughput**: many users => many transactions/sec (1.5+ billion total users)
- **Low Response Time**: on-line => users are waiting (Respond within ms)
- **Security**: sensitive information must be carefully protected since system is accessible to many users (passwords, security question, privacy controlled images)

# Roles in DB Management

- System Analysts
  - Specifies system using input from customer; provides complete description of functionality from customer's and user's point of view
  - Conceptual database design
- Database Designer
  - Specifies structure of data that will be stored in database ( logical & physical database schemas)
- DB Application Programmer
  - Implements application programs (transactions) that access data and support enterprise rules
- DB Admin
  - Maintains database once system is operational
- End-Users
  - Often unaware that they are dealing with data in a DBMS

# Summary of Introduction

- DBMS used to maintain & query large datasets that are shared by many application programs/users
- Some powerful ideas:
  - Program-Data Independence
  - Controlled Data Redundancy
  - Declarative Queries
  - Transactions
- Every 'knowledge worker' or scientists needs database know-how, as do all IT experts (application developers, software engineers, system analysts, …) - not just DBAs
- Databases are one of the broadest and most useful areas in CS and IS

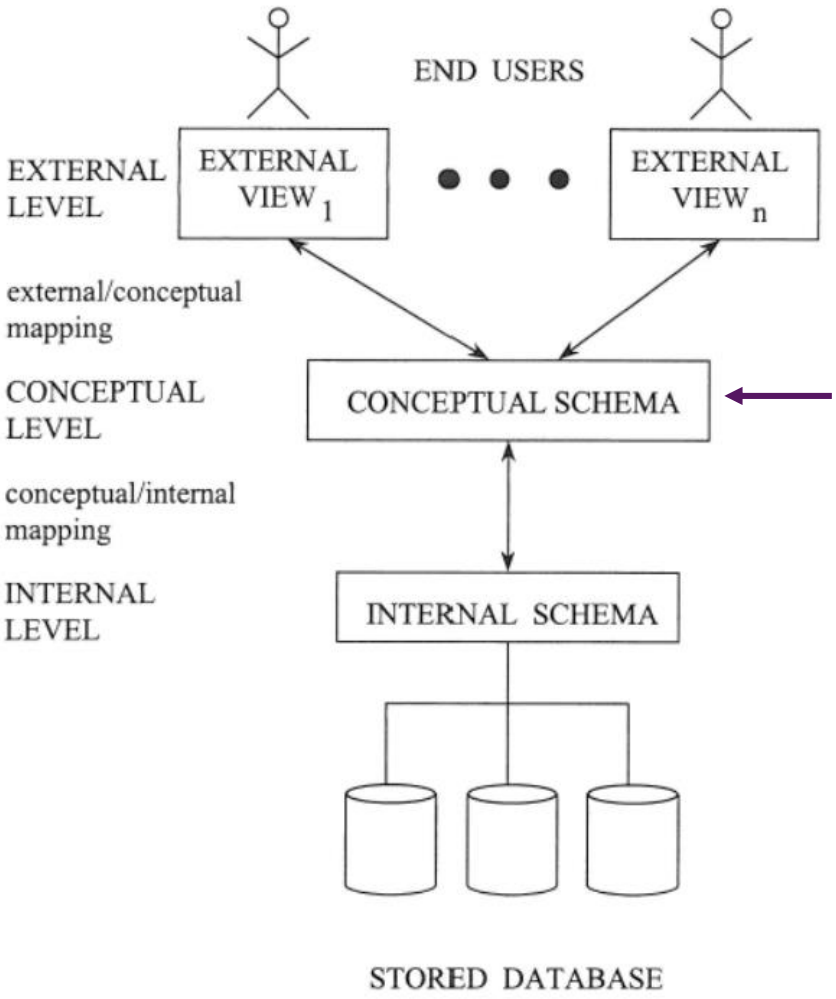
# ENTITY RELATIONSHIP DIAGRAM (ERD)

Data Modeling

# Outline

- Introduction --- Database Architecture Framework
- **Data Modeling with Entity Relationship Diagram**
- Introduction to the Relational Model
- Transformation of ER-Schema to Relational-Schema
- Normalization in the Relational Model
- Overview of SQL

# ANSI/SPARC Architecture



# Model

- A formal representation of user specification/requirements for a project.
- Description of the real world!
- An important tool in the communication among users, system analysts, database administrators.

# Entity Relationship Model

- In this class, we will confine to the **ER** = Entity-Relationship model, where the basic "players" are:
  - Entities (Entity-Set)
  - Their Attributes
  - Relationship among the entities
  - The attributes of the relationships
- There are other modeling tools on the market, with a similar nature/objectives, but used in different context(s):
  - UML (Unified Modeling Language) used for different software purposes

# Entity - Definition

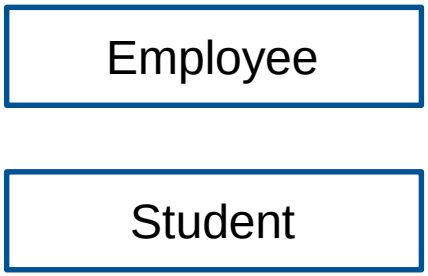
- **Entity**: Anything that can be distinctly identified, i.e., has existence in the real world. E.g., a person (Barack Obama).
- **Entity-Set:** a group of entities with identical properties. E.g., Politician.
- Barack Obama (the entity) is a member of the entity-set Politician.
- Sometimes the word Entity is used to refer to Entity-Set.

# Entity - Example

- **Database of a University (e.g., Northwestern):**
  - Student
  - Faculty
  - Non-Academic Stuff
  - Department
  - Courses, etc.
- **Database of a Bank (e.g., Chase):**
  - Customers
  - Employee
  - Manager
  - Branches
  - Account
  - Deposit/Transaction, etc.

# Entity - Representation

- An entity is represented in the diagram as below:



# Attributes - Description of Entities

- NOTE: entity classes need not be “fully disjoint” (in the sense of not having anything in common whatsoever…):
  - A very same person can be both employee and customer of the bank.
- Essentially, one can think that every **entity** is represented by a set of **Attributes** (table columns):
  - (descriptive) set of **properties** that every instance of that entity class ***should*** have

# Attributes - Example

- An employee can have:
  - Name
  - Age
  - Salary
  - Join Date
  - Address
  - Phone No., etc.
- A student can have:
  - Name
  - Net-id
  - Major
  - CGPA
  - Degree Level, etc.

# Attributes - Example

- We can represent an entity-set as follows:

*Name_Of_Entity* (*Attribute1*, *Attribute2*, ….*AttributeN*)

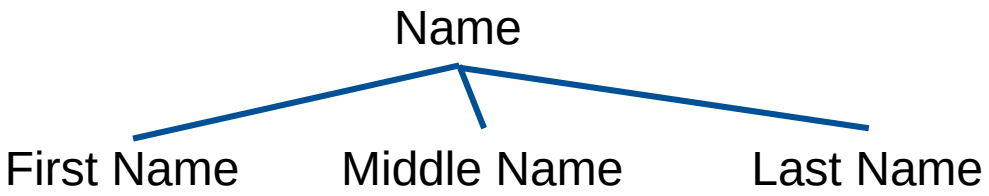
- **Example 1:** Employee (Name, Age, Salary, Join Date, Address)
- **Example 2:** Student (Net-id, Name, Major, CGPA, Degree-Level)

# Attributes – Types (Single-valued vs Multi-valued)

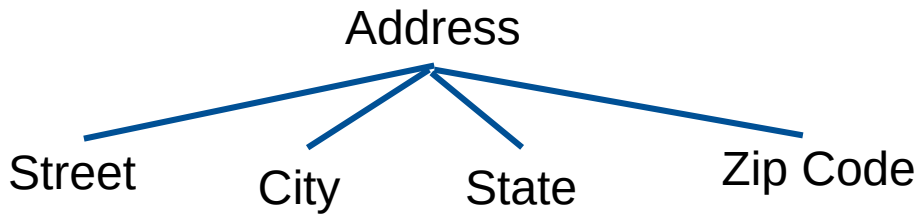
- Most attributes are Single-valued, e.g., Net-id, Salary, etc.
- The nature of certain attributes is such that they have > 1 from the domain (multi-valued).
- **Example1:** Phone Number --- A person may have multiple phone numbers.
- **Example2:** Degrees --- A person can have multiple degrees (B.Sc., M.Sc., Ph.D., etc.)

# Attributes – Types (Simple vs Composite)

- **Composite attributes** can always be split into a collection of attributes that are “atomic” (i.e., their domain is “simpler”)
- **Example1:**




- **Example2:**



# Attributes - Domain

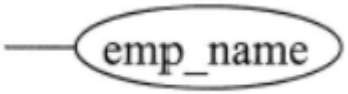
- For each individual attribute, there has to be a specification of the (set of the) “permissible” values, called domain. In addition, for each domain, we need:
  - **Type** of the domain
  - **Restrictions** on the **values** from the respective domain
- **Example1:** Attribute :- **Name**, Domain :- **String**, Restriction:- **Maximum 30 characters**.
- **Example2:** Attribute :- **Age**, Domain :- **Number**, Restriction:- **[16, 70]**.

# Attributes – NULL Values

- **NULL** is a special **symbol** that denotes the situation in which a particular instance does not have a value for the given attribute. It can indicate:
  - Non-Applicable values
    - Single family home has no apt no
  - Missing Data
  - Not Known
    - We don’t know the state where Mary smith lives.

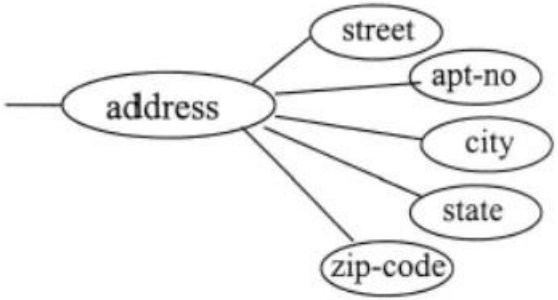
# Attributes – Representation

- Attributes are represented as below (oval-shaped) 
- Example:




- Multi-valued:


- Composite:

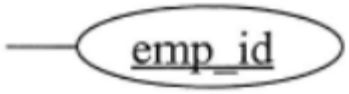


# Attributes – Primary Key

- How to retrieve a particular entity from an entity-set? (i.e., retrieve a tuple from a table)
- **Primary Key:** An attribute (or, a set of attributes) that uniquely identifies an entity within an entity-set.
- **Example 1:** In case of the student entity, net-id must be unique for all the students. Thus, net-id can be a primary key of the student entity.

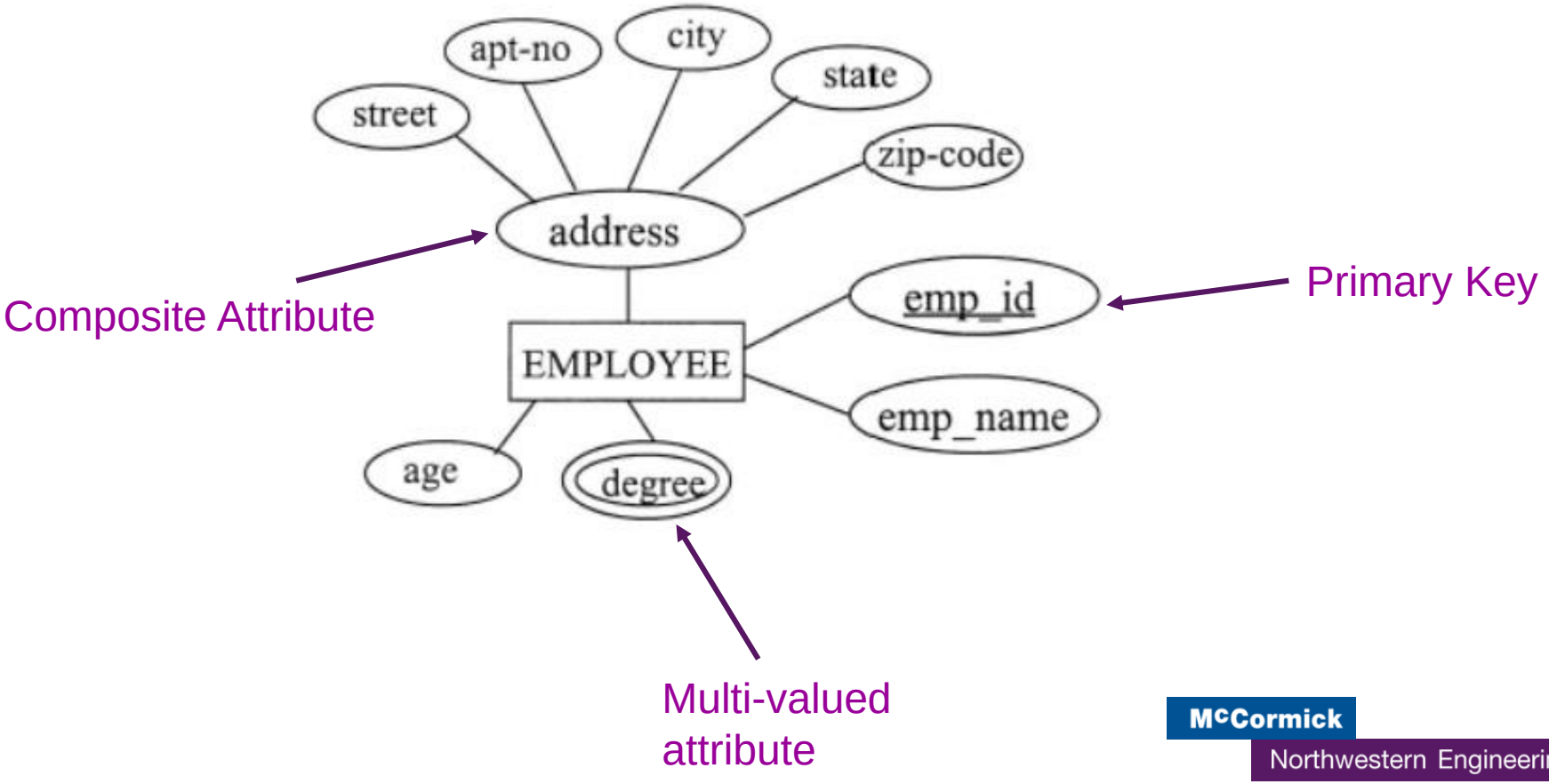
# Attributes – Primary Key

- **Example 2:** In case of the course entity, the combination of two fields: department and course number must be unique for all the courses. E.g., “EECS 339”. Thus, (department, course number) can be a primary key for the course entity.
- Each entity(table) must have a primary key.
- Primary key is represented in the model as below:



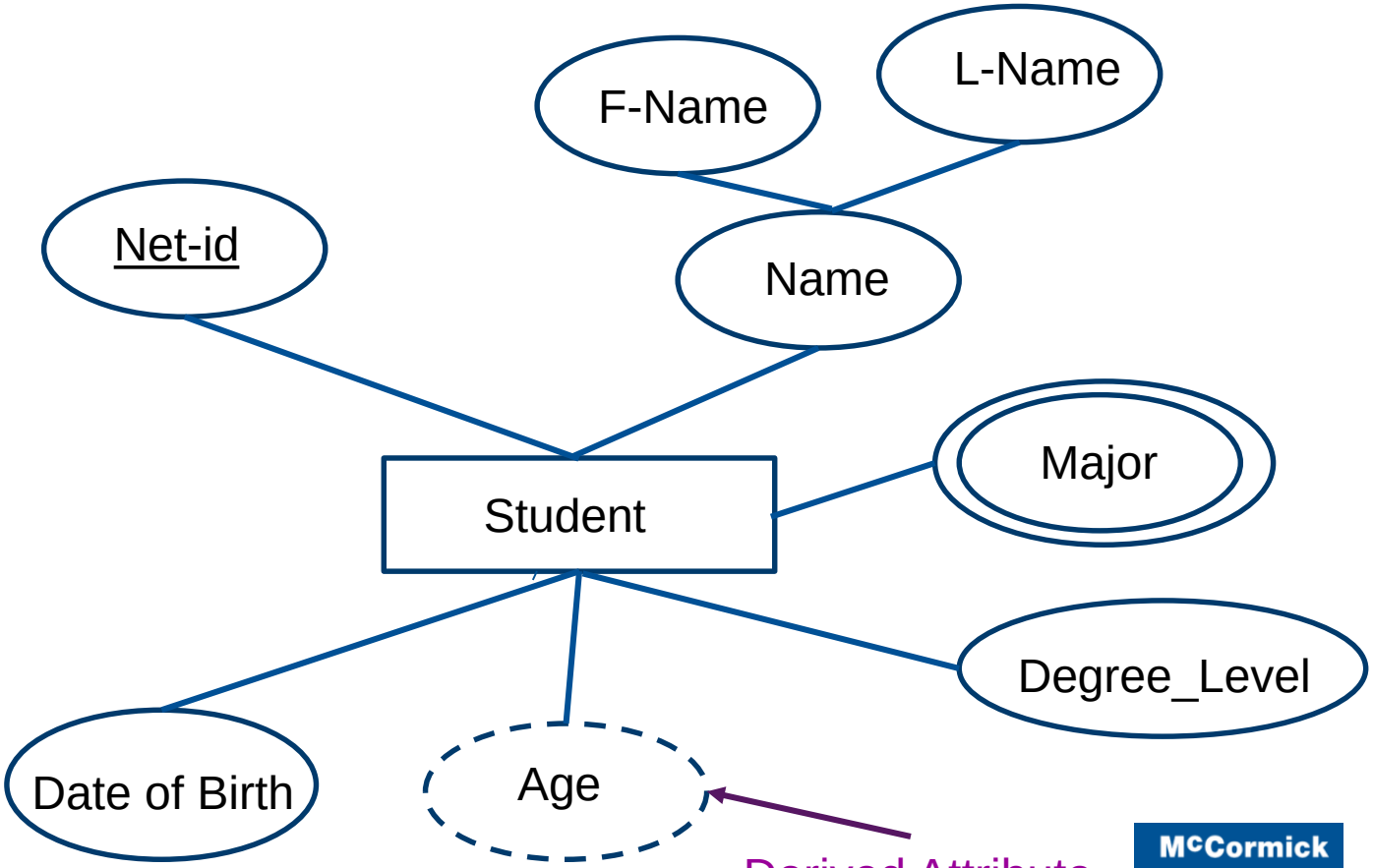
# Example Representation

- Employee (emp-id, emp-name, address, degree, age)



# Example Representation

- Student (net-id, first_name, last_name, major, degree_level, date_of_birth, age)

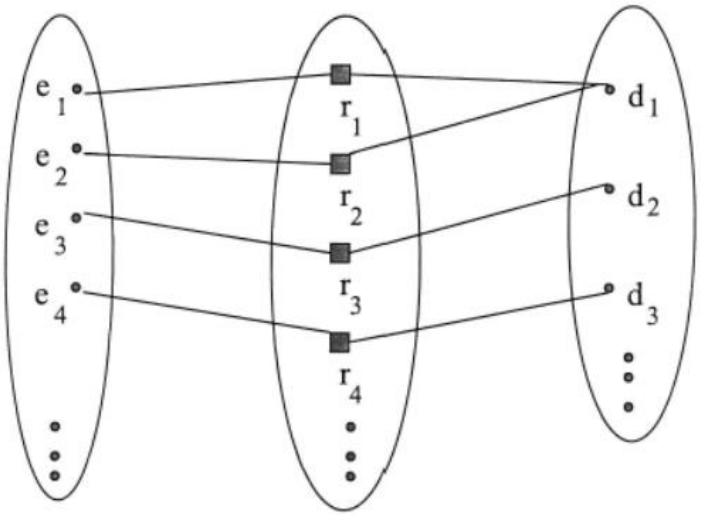


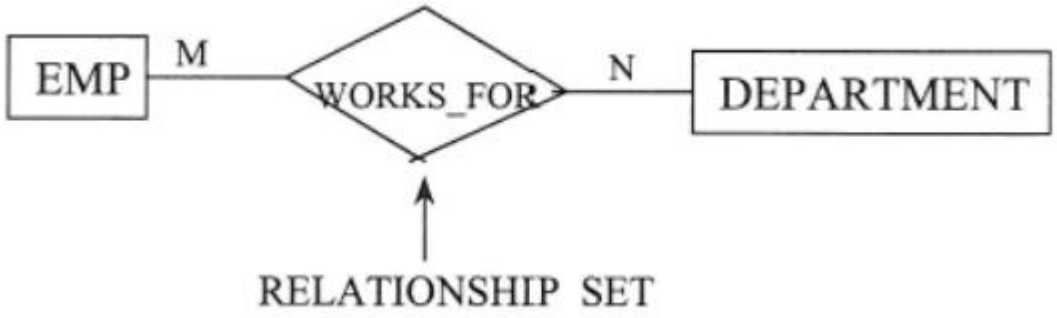
# Modeling Relationship

- **Relationship:** A (possible) association among the instances of two (or more) entities.
- **Example:**

| Hayes | *depositor* | A-102 |
|---|---|---|
| *customer* entity | relationship set | *account* entity |

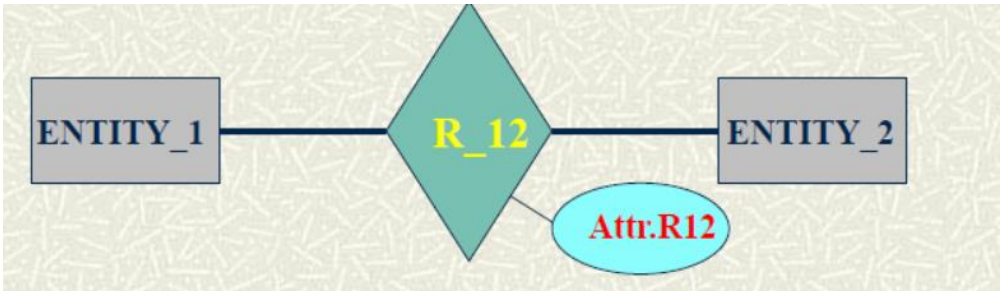
# Modeling Relationship - Examples



# Modeling Relationship

- **Relationship (Type):** A description of the possibility of two (binary) or more (ternary, quaternary,…) entity classes having some association among their instances. **NOTE: may have attributes of its own.**
- **Relationship Instance:** An association between two (or more) entities. Note that in an actual database, these will be rows/tuples of the tables.
- Symbol:

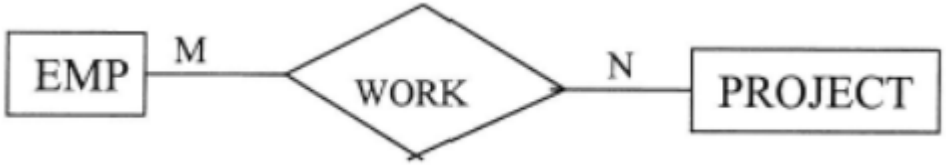


# Modeling Relationship

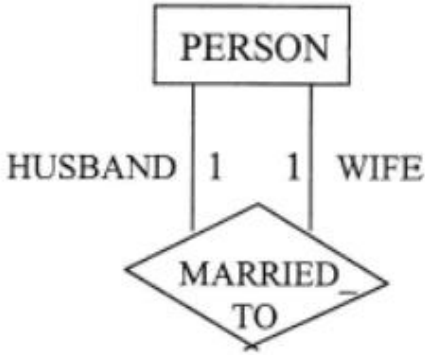
- Relationships are characterized in terms of:
  - *Degree*
  - *Cardinality*
  - *Constraints*
  - *Attributes*

# Degree of Relationship

- **Degree**: No. of participating entity-sets.
- Binary Relation:

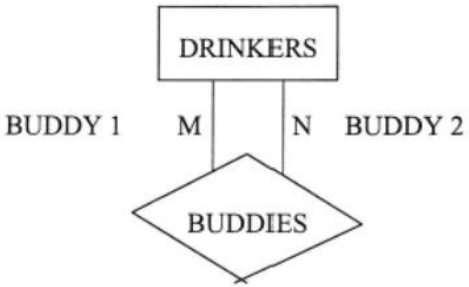



- Recursive Binary (Relation between same entity-sets):



# Degree of Relationship

- Recursive Binary (Relation between same entity-sets):

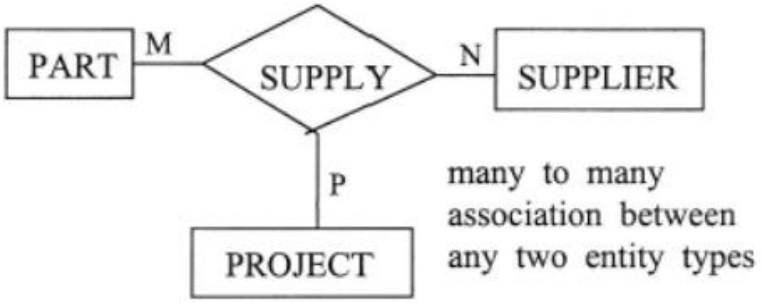



| Buddy 1 | Buddy 2 |
|---|---|
| $d_1$ | $d_2$ |
| $d_1$ | $d_3$ |
| $d_2$ | $d_1$ |
| $d_2$ | $d_4$ |

- Buddies is *symmetric*, married is not.
- No way to model *symmetry*.
- **Design Question:** Should husband and wife be replaced by spouse?

# Degree of Relationship

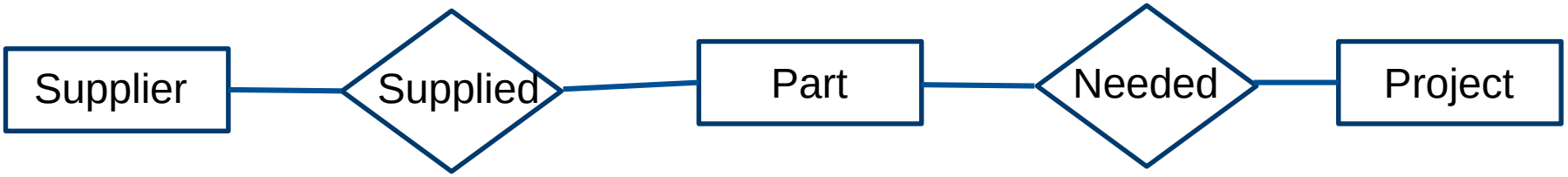
- Ternary Relationship:




| PART# | SUPPLIER# | PROJ# |
|---|---|---|
| 25 | 4 | 1 |
| 25 | 5 | 2 |
| 10 | 4 | 2 |
| 17 | 4 | 3 |
| 17 | 2 | 1 |
| 17 | 5 | 1 |

# Degree of Relationship

- Ternary (and higher) Relationship should be avoided if possible (into binary relations):



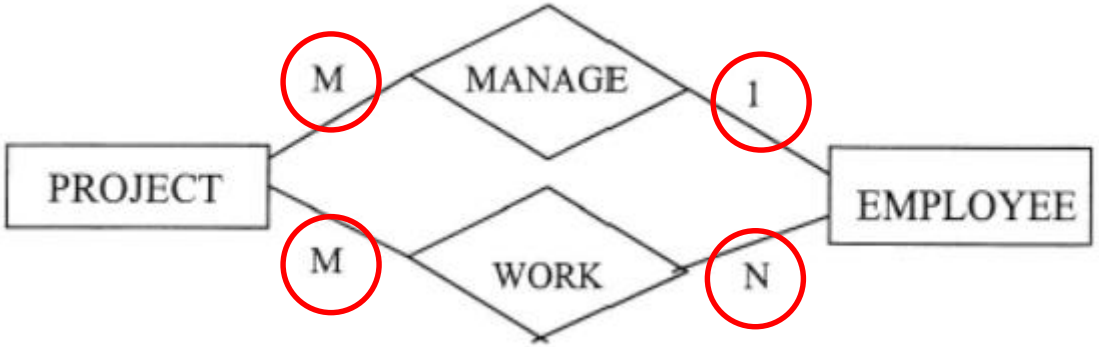

- But, it is unavoidable when two binary relationships are not equivalent
  - E.g., Supplier 4 supplied Part 25 for Project 1, but Supplier 5 supplied Part 25 for Project 2.

| PART# | SUPPLIER# | PROJ# |
|---|---|---|
| 25 | 4 | 1 |
| 25 | 5 | 2 |

# Cardinality of Relationship

- **Cardinality Ratio**: Specifies the number of relationship instances that an entity can participate in.

Each project is managed by ***one*** employee (Manager),
But an employee can manage ***many*** projects

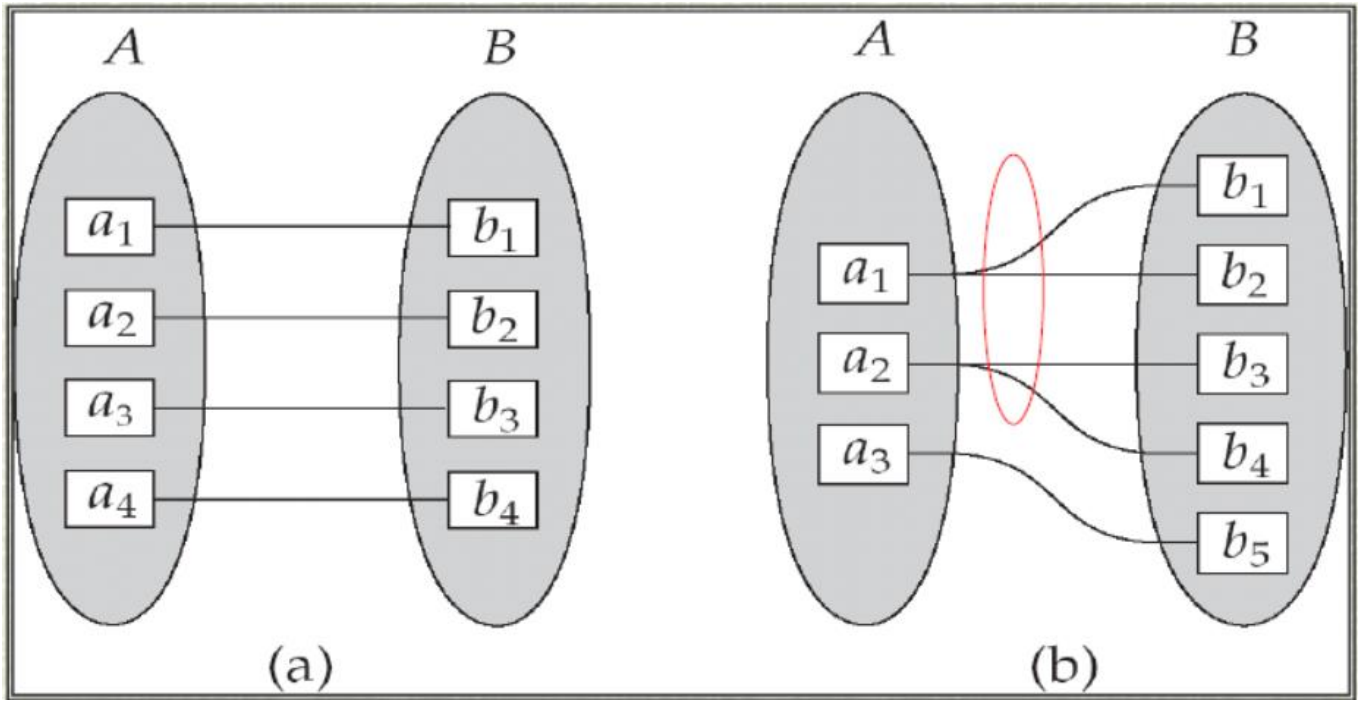



Each project has ***many*** employees,
And an employee can work on ***many*** projects

- Possible mappings are ---**1:1, 1:M, M:1, M:M (or, M:N)**

# Cardinality of Relationship

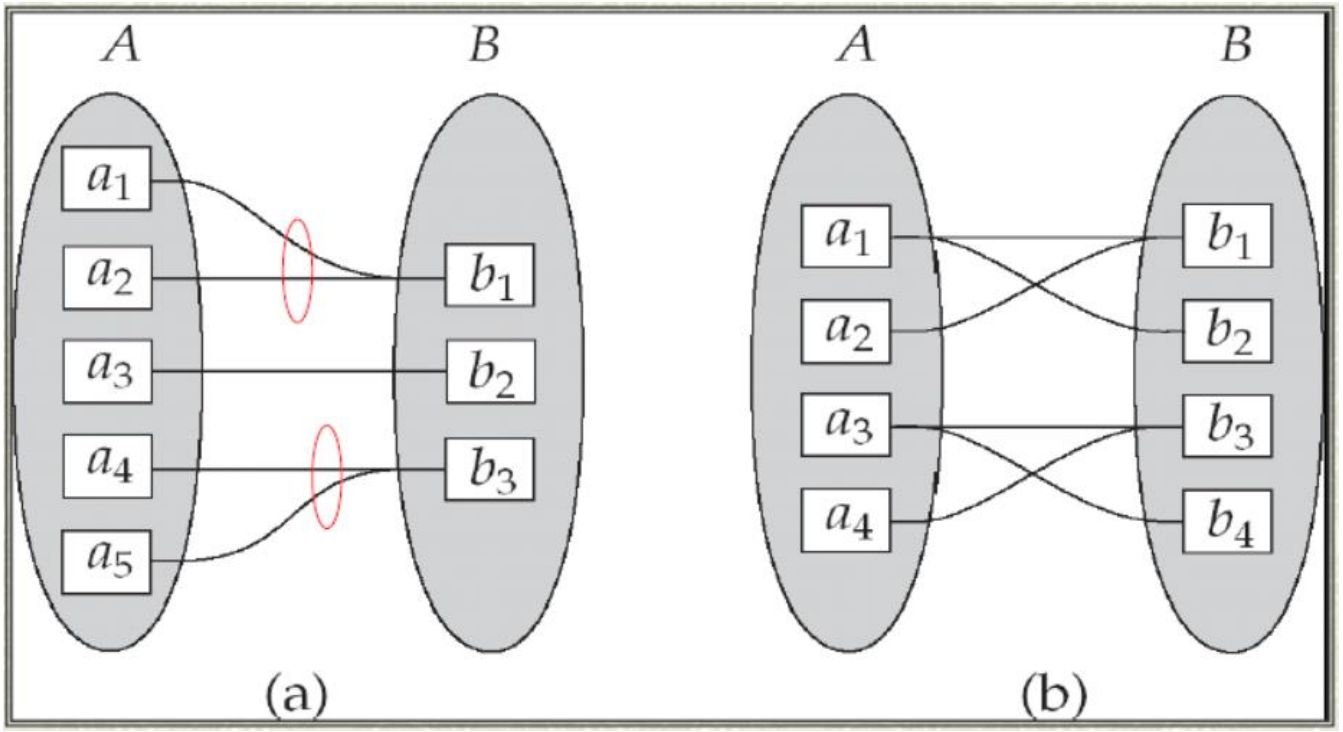
- **Important**: *one* and *many* refers to the maximum cardinality.




**1-1** **1-M**

# Cardinality of Relationship

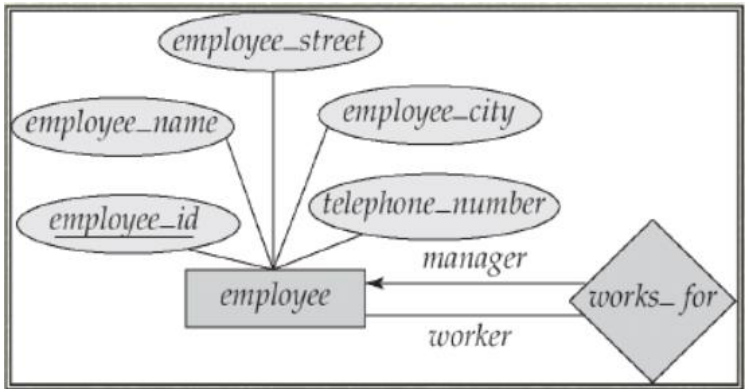
- **Important**: It could quite be the case that some elements in *A* and/or *B* ***are not mapped*** to any element(s) in the other set

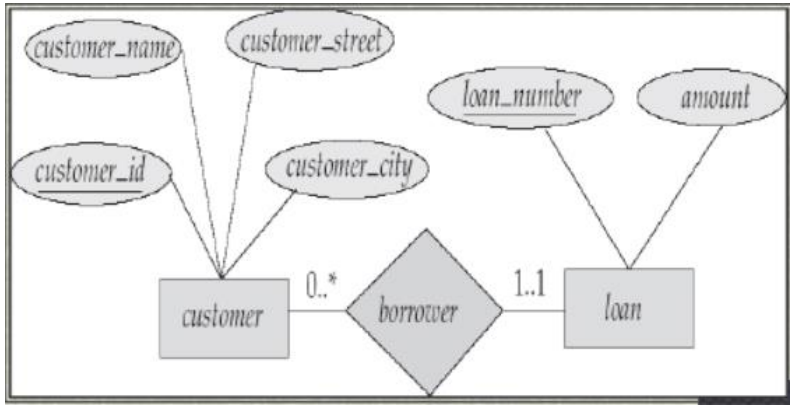



**M-1** **M-M**

# Cardinality of Relationship

- **Important**: The notation is not fully standardized, e.g., other notations are also used.



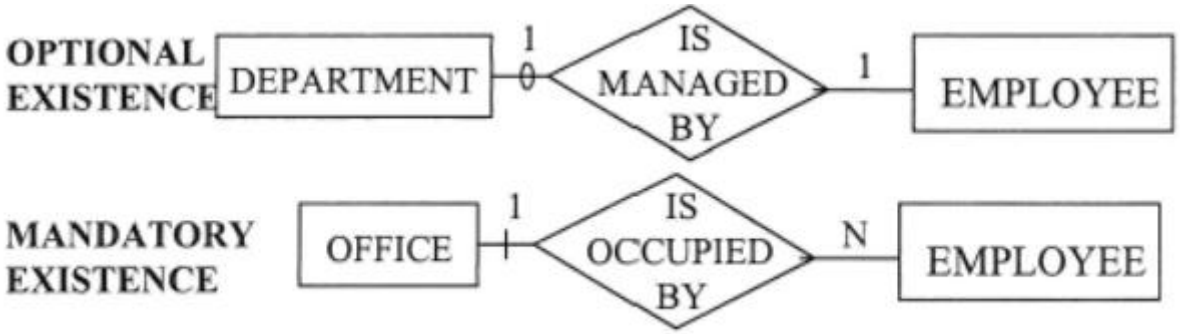




One

Many

# Constraints of Relationship

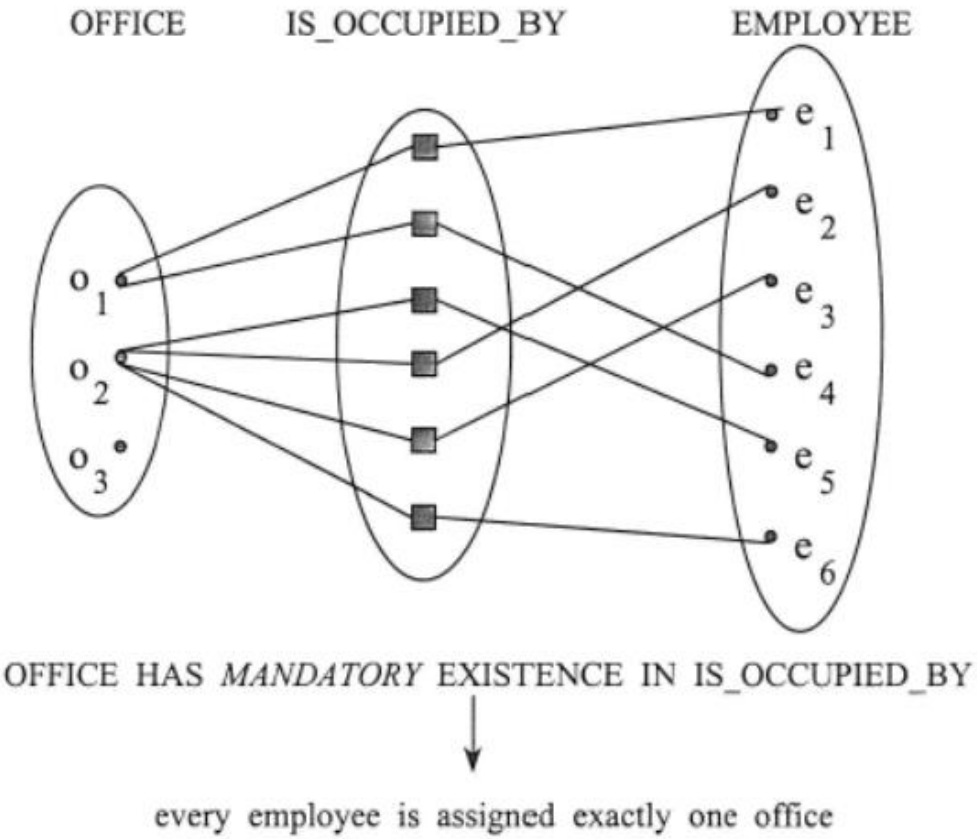
- MAY vs. MUST (AKA “partial” vs. “total” participation constraint)




**OPTIONAL EXISTENCE:** defines a minimum cardinality of zero **(Partial)**

**MANDATORY EXISTENCE:** defines a minimum cardinality of one **(Total)**

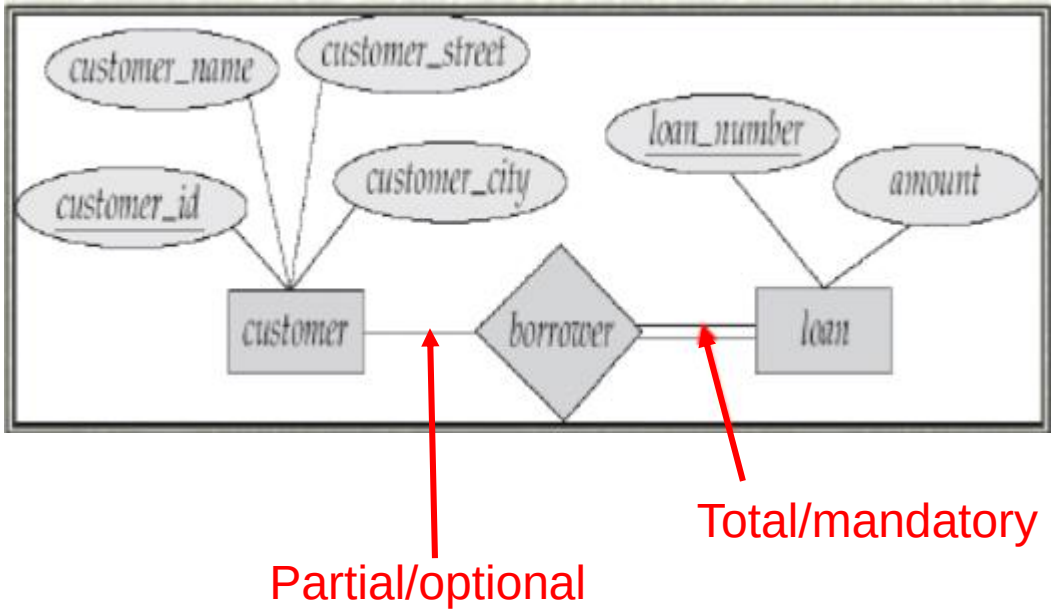
# Constraints of Relationship




**Same is not true for the Department and Employee in the previous example**

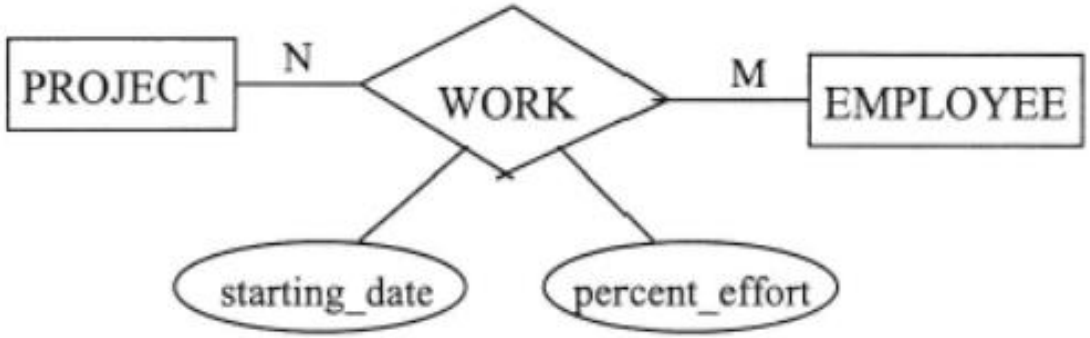
# Constraints of Relationship

- Participation of ***Loan*** is **total**, in the sense that every instance of ***Loan*** **must** by in a relationship with **some instance** from ***Customer*** (not vice versa, though)

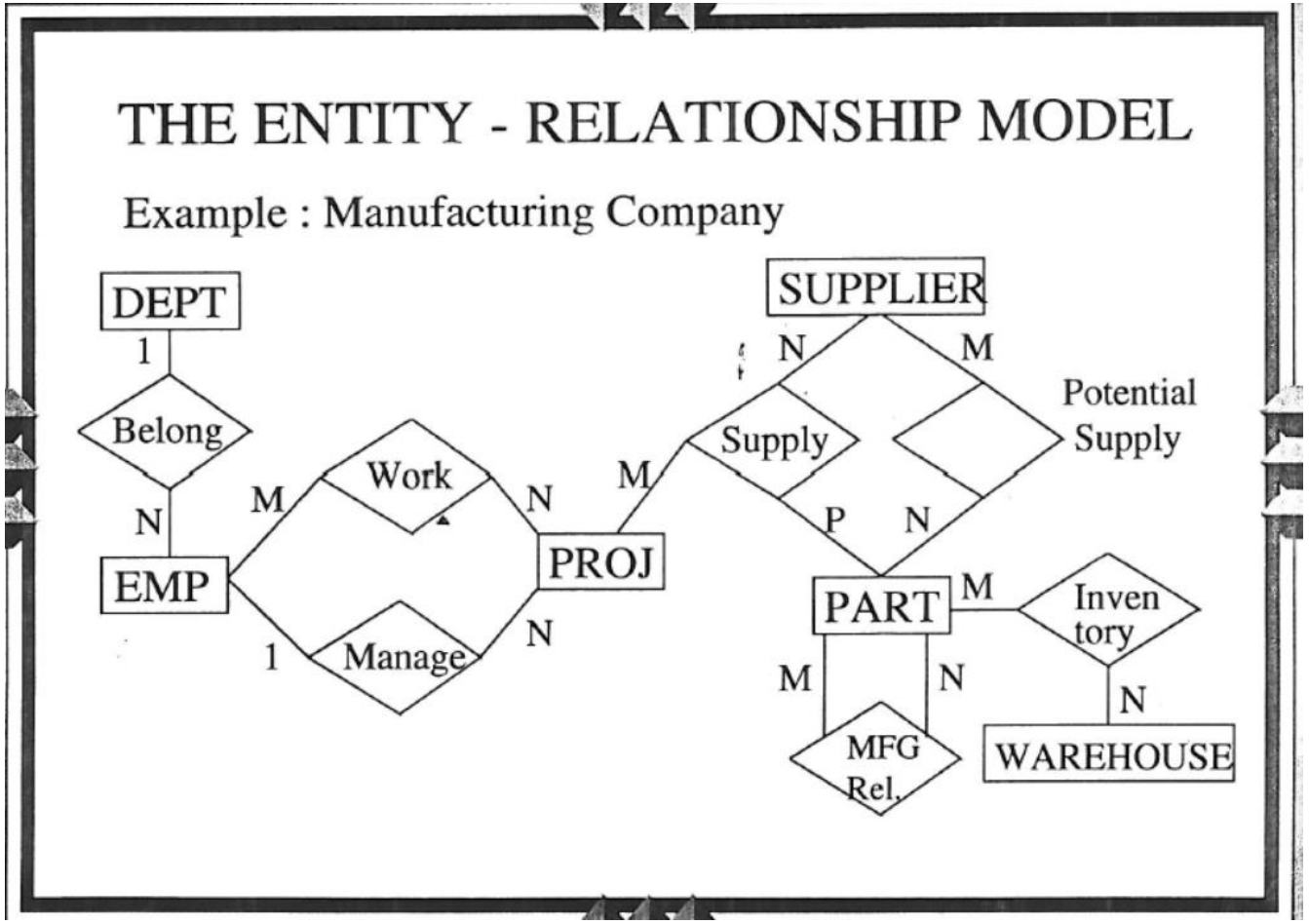


# Attributes of Relationship

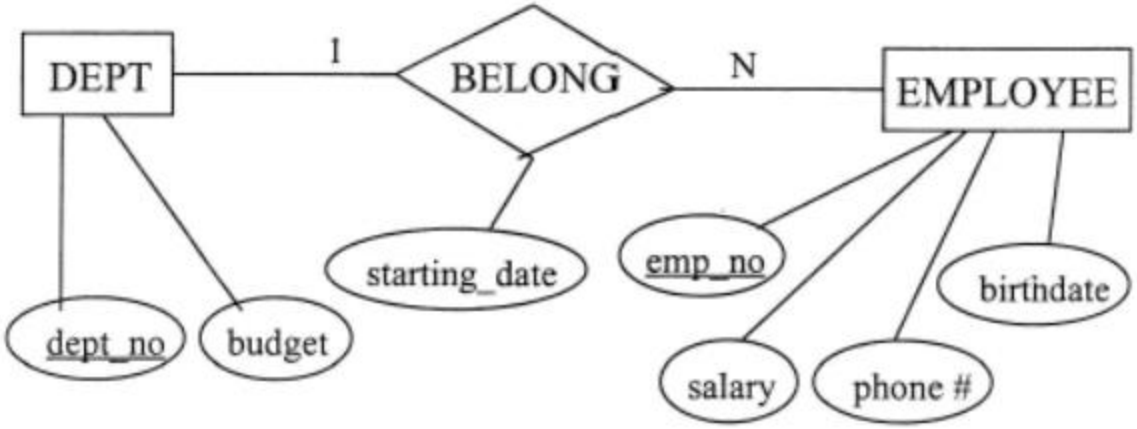
- Relationships can have **attributes** too!



# ERD - Examples



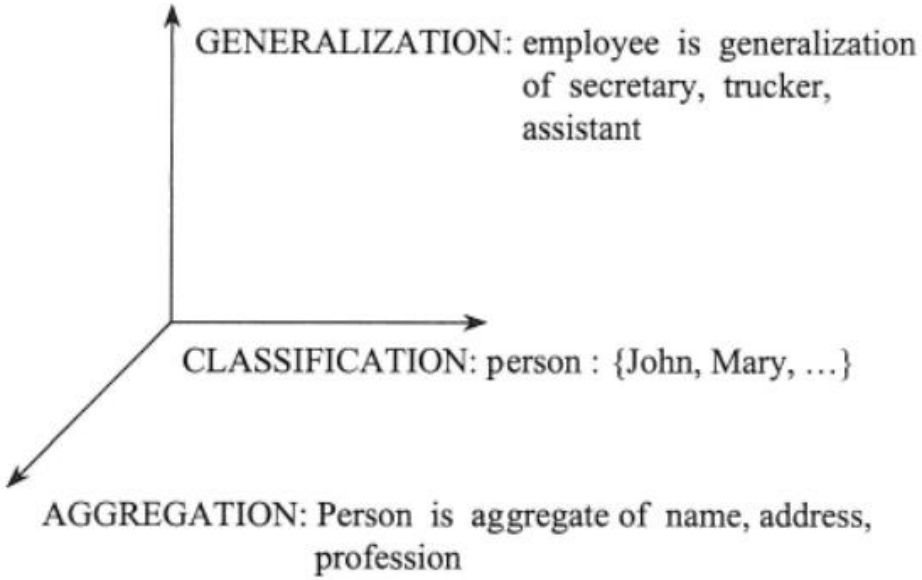
# ERD - Examples




**NOTE**: attributes *birthdate* and *starting_date* are defined on the same value-set

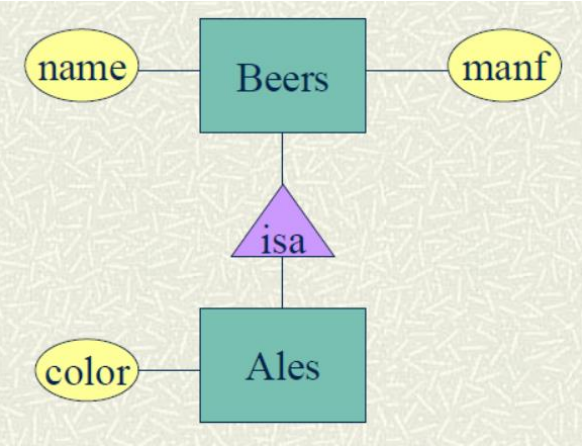
# ERD – Advanced Topics

- **Classification:** Collection of objects with similar properties viewed as a class.
- **Aggregation:** Abstraction by which an object is constructed from its constituent objects.
- **Generalization:** Set of dissimilar objects are considered as a single generic object.



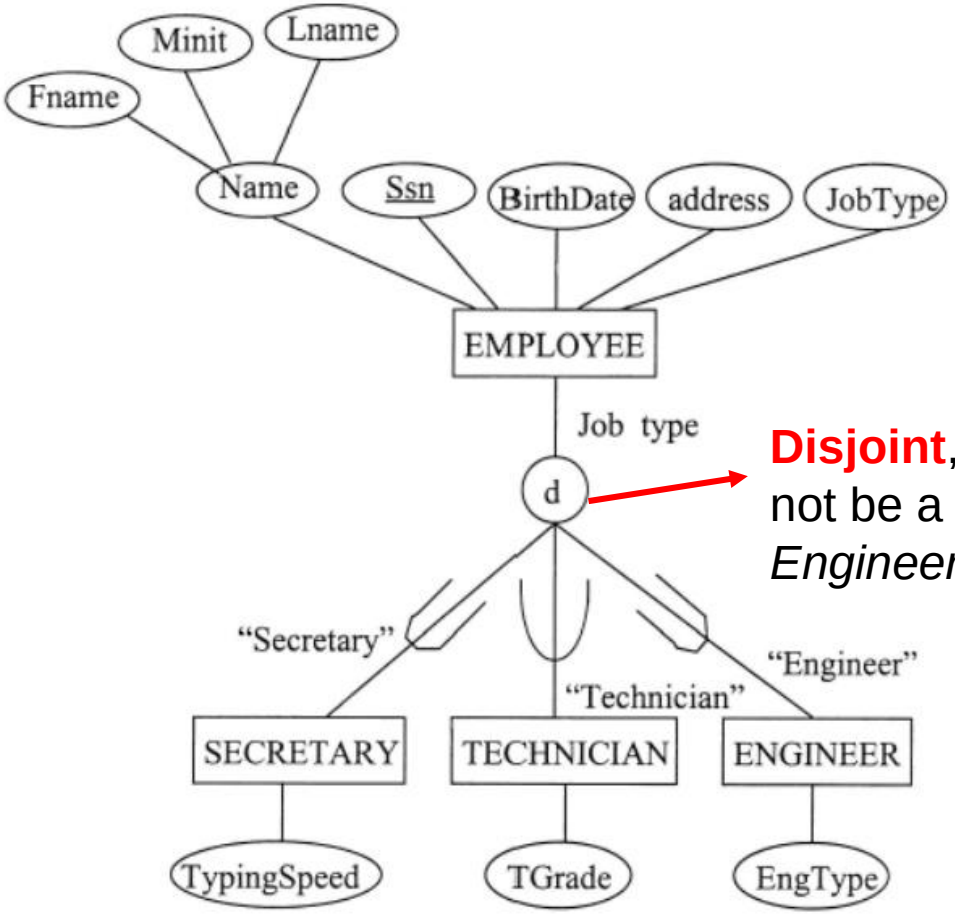
# Generalization Hierarchy

- **Class/Subclass Hierarchy:** Similar to object-oriented programming concept *inheritance*, e.g., in Java, C++, etc.
- **Example:** Ales are a kind of beer.
  - Not every beer is an ale, but some are.
  - Let us suppose that in addition to all the *properties* (attributes and relationships) of beers, ales also have the attribute color.




**NOTE:** there may be other "conventions" to indicates subclasses…

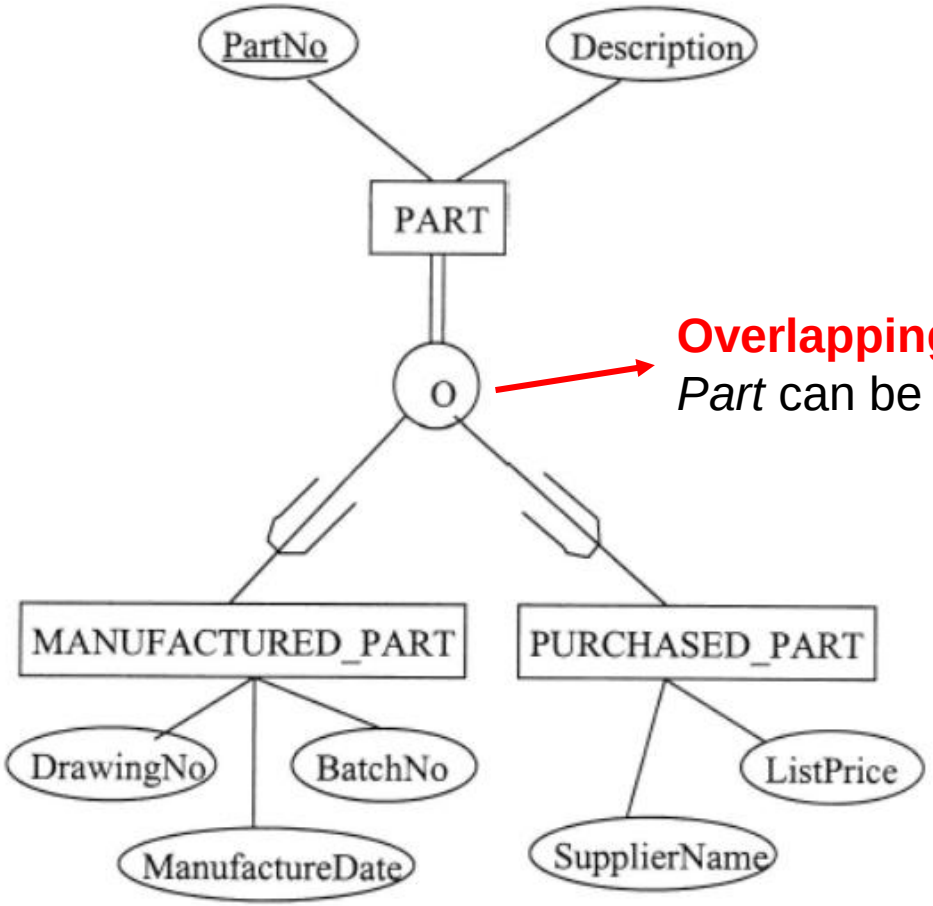
# Generalization Hierarchy




**Disjoint**, a *Secretary* can not be a *Technician* or *Engineer* as well

Note: subset symbol on a line indicates direction of class/subclass relationship.

# Generalization Hierarchy




**Overlapping**, a *Manufactured Part* can be *Purchased* as well

# Generalization Hierarchy

employee - generalized class contains only attributes in common

secretary, engineer, trucker - specializations of employee

•**ATTRIBUTE INHERITANCE**

an entity that is a member of a subclass inherits all the attributes of the entity as a member of the superclass. the entity will also inherit all *relationship* instances for relationship types in which the superclass participates.

•**IMPLIED CONSTRAINTS**

an entity cannot exist in database by being only a member of a subclass, it must also be member of superclass (reverse is not true).

# Thank You.