Database Schema Design

Using

Entity-Relationship Approach

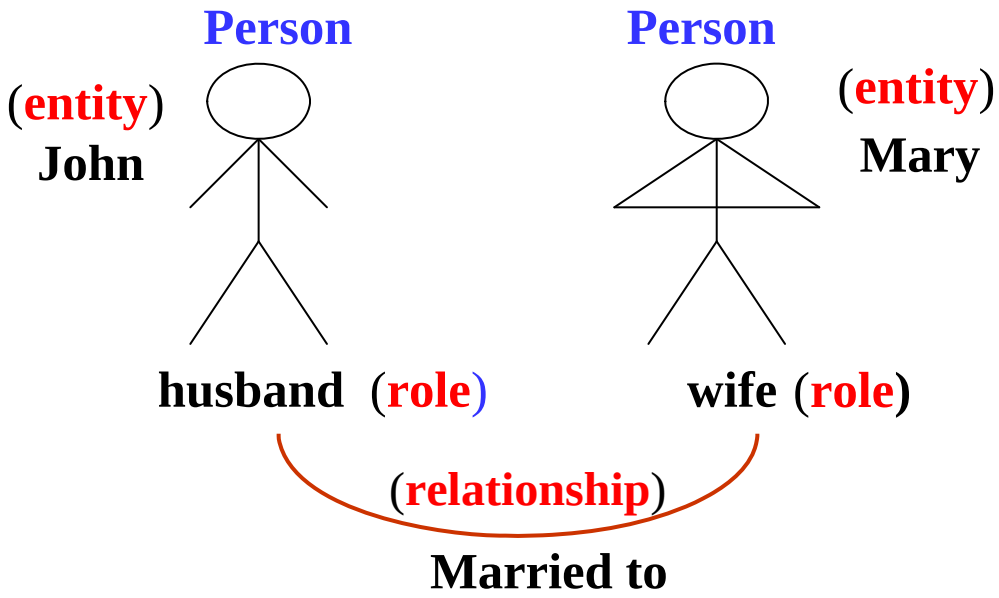
(ER Approach or ER Model)

Tok Wang Ling
National University of Singapore

Topics

- Concepts/Constructs in ER Approach and diagram
  - Cardinality vs. Participation Constraint
  - Weak Entity Type, EX/ID Relationship Types, generalization and specialization
  - Some extensions: Aggregation, Multiple FDs Representation
- English Sentence Structure and ER Diagram
  - self study
- ER Construct Notation Comparison
- Database Schema Design using ER Approach
- Translation of a (Normal Form) ER Diagram to a RDB
- A Normal Form for ER Diagram

- **ER approach** was proposed by Prof. Peter Chen in TODS 1, 1976.
- Main Concepts:
  - **entity** (i.e. **object**)
  - **relationship**
  - **attribute**



Brief ideas:

English correspondence

noun → entity

verb → relationship

Ref:

- Peter Chen paper TODS 1976
- Elmasri&Navathe's book
- Korth's book
- Hawryzkiewycz's book

Entity: An **entity** is an object which exists in our mind and can be distinctly identified.

Q: How to identify entities?

E.g. - Ng Hong Kim with NRIC S0578936I
- Account# 563978 of DBS Bank
- Car with car plate number SBG 3538P

Entity type

- Entities can be classified into different types.
- Each entity type contains a set of entities each satisfying a set of predefined common properties.

E.g. Employee, Student, Car, House, Bank Account

Q: What are the common properties of each of the above entity types?

List of common entity types:

- People: humans who carry out some function
  Employees, students, customers
- Places: sites or locations
  Cities, offices, routes, countries
- Things: tangible physical objects
  Equipments, products, buildings
- Organizations
  Teams, suppliers, departments
- Events: things that happen to some other entity at a given date and time or as steps in an ordered sequence.
  Employee promotions, project phases, account payments
- Concepts: intangible ideas used to keep track of business or other activities.
  Projects, accounts, complaints

Relationship. A relationship is an association among several entities.

E.g. A relationship which associates customer Ng Hong Kim identified by NRIC S0578936I and DBS bank account 5075610.

Relationship type (or **Relationship set**)

Each relationship type contains a set of relationships of the same type each satisfying a set of predefined common properties.

If $E_1, E_2, \ldots, E_n$ are entity types, then an **n-ary relationship type** R is a subset of the Cartesian Product $E_1 \times E_2 \times \ldots \times E_n$,

i.e. $R \subseteq E_1 \times E_2 \times \ldots \times E_n$

or $R \subseteq \{(e_1, e_2, \ldots, e_n) \mid e_i \in E_i, i = 1,2,\ldots,n\}$

where $(e_1, e_2, \ldots, e_n)$ is a relationship.

When n = 2 (or 3), we call R a binary (or ternary) relationship type.

E.g. We define a binary relationship type Work to denote the association between two entity types Department and the Employee

Work $\subseteq$ Department × Employee

Attribute

- An entity type E (or a relationship type R) has attributes representing the structural (static) properties of E (or R resp.).
- An **attribute** A is a mapping from E (or R) into a Cartesian Product of n values sets, $V_1 \times V_2 \times \ldots \times V_n$.

❖ • If $n \geq 2$, then we call attribute A a **composite attribute**, otherwise (i.e. when n=1) call it a **simple attribute**.

E.g. DATE is a composite attribute with values sets
DAY, MONTH, YEAR

❖ • The mapping can be one-to-one (1:1), many-to-one (m:1), one-to-many (1:m), many-to-many (m:m).

- If an attribute A is a 1:1 or m:1 mapping from E (or R) into the associated value sets, then A is called a **single valued attribute**, otherwise it is called a **multivalued attribute**.

Note the difference between type and instance. We use

- entity type vs. entity
- relationship type vs. relationship
- attribute vs. attribute value

Some books and papers use slightly different terms:

- entity and entity instance
- relationship and relationship instance
- attribute and attribute value

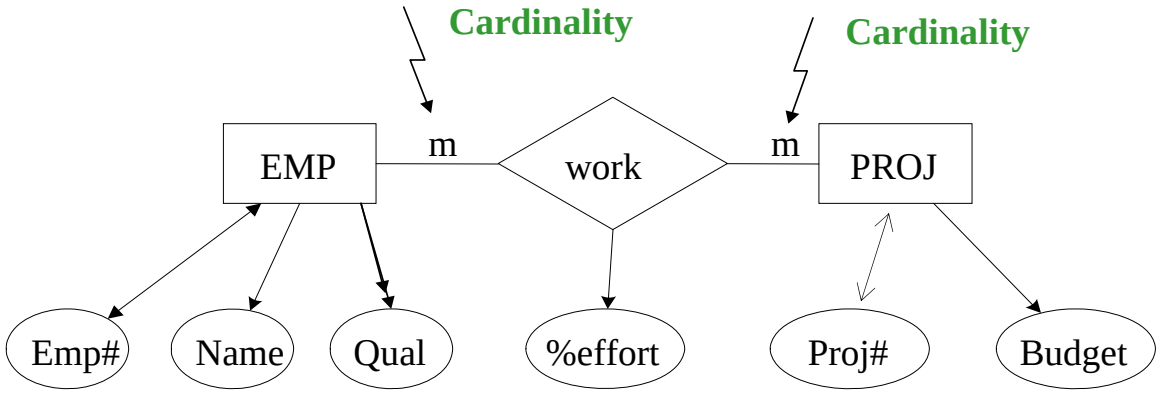
Some books and papers just don't differentiate them, simply use entity and relationship for both type and instance, may have interpretation problem.

Entity-Relationship Diagram (ER Diagram or ERD)

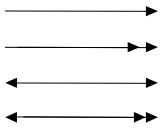
- The structure (i.e. schema) of a database organized according to the ER approach can be represented by a diagrammatic technique called an **Entity-Relationship diagram**.

Notation:

- entity type → rectangle
- relationship type → diamond
- attribute → ellipse



❖ **Notations**



m : 1 (single valued attribute)
m : m (multi-valued attribute)
1 : 1 (one to one attribute)
1 : m (one to many attribute)

❖ **Note:** There are some other different representations (notations). Many books don't use arrows and have problem to interpret ER diagrams precisely.

Different cardinalities of binary relationship types

FD: Emp → Dept

work is a **many to one** relationship type

FDs: Mgr → Dept
Dept → Mgr

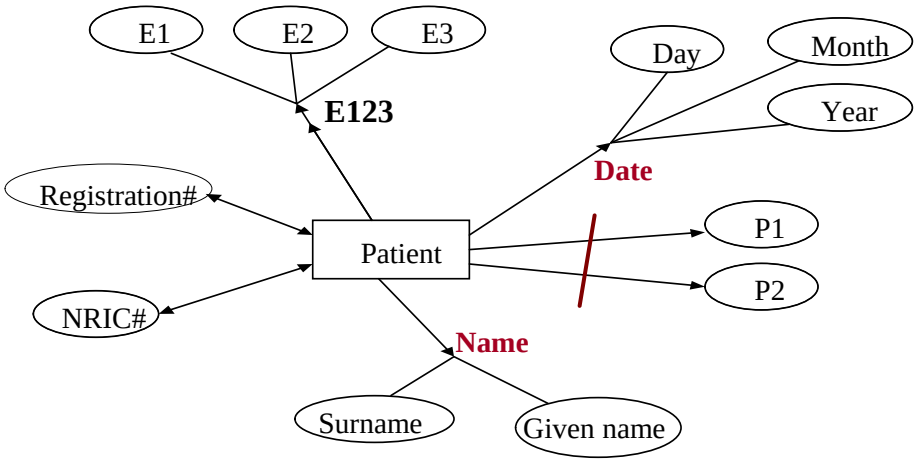
manage is a **one to one** relationship type

No FD between Emp and Project,

work is a **many to many** relationship type.

Q: What are the intuitive meanings of the above relationship types?

Composite Attribute



E.g.
Name, Date, and E123 are composite attributes.

Note:
A line joining the two attributes' arrows indicates the two attributes form a key of the entity type Patient.

E.g. P1 and P2 form a key of Patient.

Identifier of entity type

- A **minimal set** of attributes K of an entity type E which defines a one-to-one mapping from E into the Cartesian Product of the associated value sets of K is called a **key** of E.
- One of the keys of an entity type is designed as the **identifier**.

❖ **Q:** How to choose the identifier of an entity type?

E.g.. Registration#, NRIC#, and {P1, P2} are 3 keys of PATIENT entity type, we choose Registration# as the identifier. **Why?**

❖ **Q:** Are the concepts of **identifier** of entity type and **primary key** of relation of relational model the same? If not, what are the main differences between them?

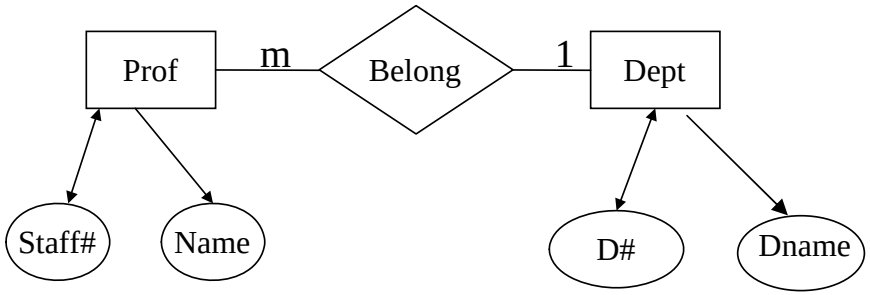
Identifier of relationship type

- Let K be a set of identifiers of some entity types participating in a relationship type R. K is called a **key** of the relationship type R if there is an 1:1mapping from R into the Cartesian Product of the associated value sets of K and no proper subset of K has such property.
- One of the keys of R is designated as the **identifier** of R.

❖ **Q:** Why do we need to define identifiers for relationship types? How?

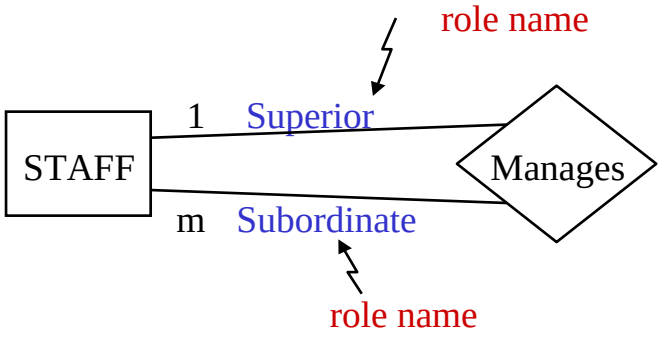
E.g. {Emp#, Proj#} is the only key of the relationship type " work" in the previous EMP-PROJ ER diagram.

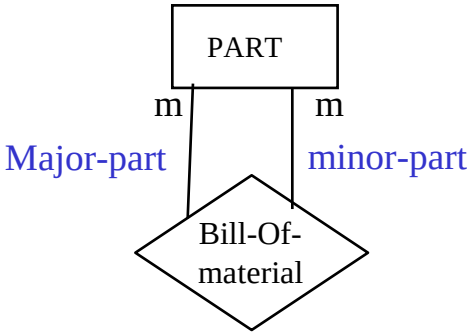
E.g.



Staff# is the identifier of the binary relationship type Belong.

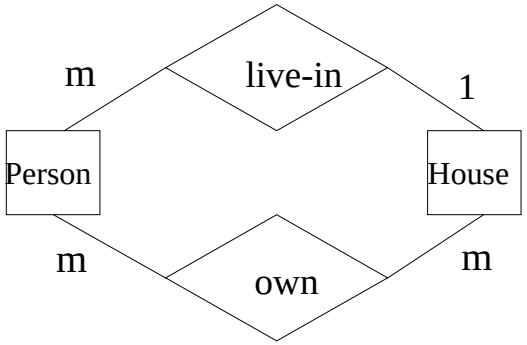
Recursive relationship type





- **Q:** How to represent recursive relationship type and in a relational database?

Two relationship types between the same set of entity types



- **Q:** How to represent these 2 relationship types in a relational database?

Participation Constraint - another way to specify constraints:

E.g. Students take courses

Here **2:8** means each student must take at least (minimum) **2** courses and at most (maximum) **8** courses.

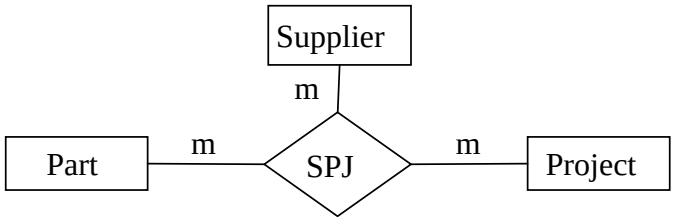
4:m means each course must have minimum 4 students and no maximum limit. (m means many, no limit).

If every entity of an entity type must participate in some relationship(s) of the relationship type then that entity type has total (or mandatory) participation in the relationship type.

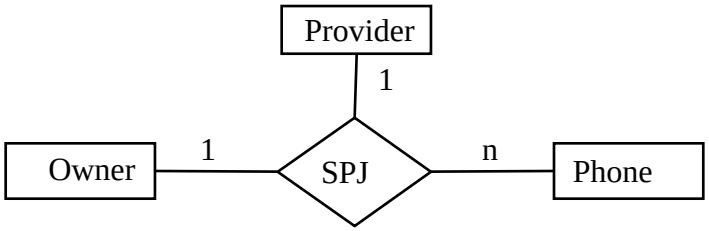
If some entities of an entity type need not participate in any relationship of the relationship type then the participation of that entity type in the relationship type is partial (or optional).

- **Q: What are the differences between cardinality and participation constraint? Which one is better (i.e. more powerful)?**

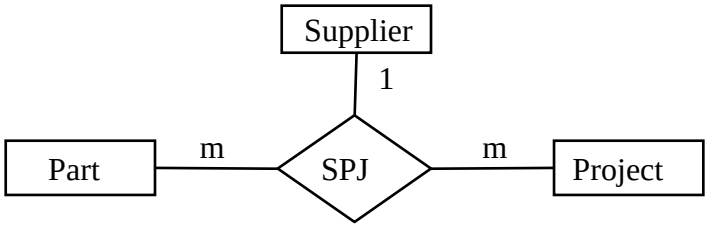
Ternary relationship type



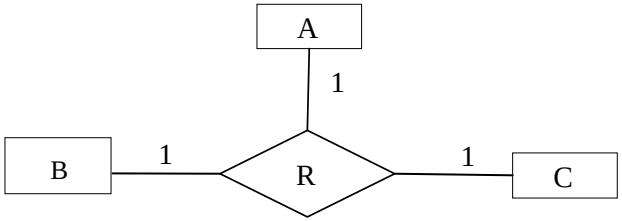
No FDs among Part, Supplier, and Project.
It is a m:m:m ternary relationship type.



FD: Phone →Owner, Provider

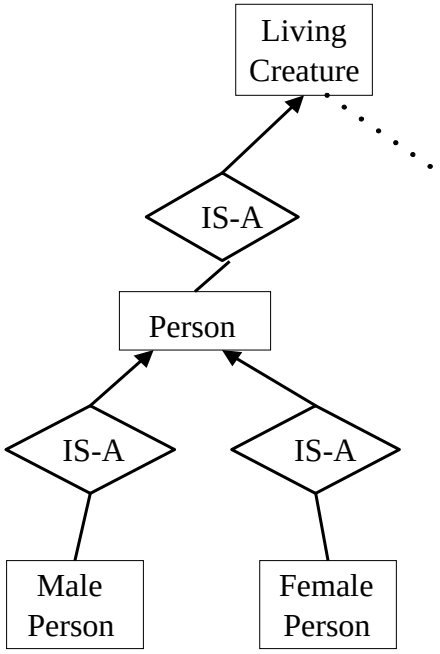


FD: Part, Project → Supplier

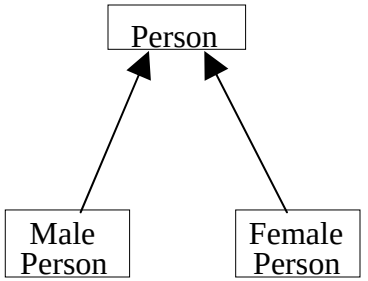


FDs: A → B C
B → A C
C → A B

Subtype relationship (IS-A hierarchy)

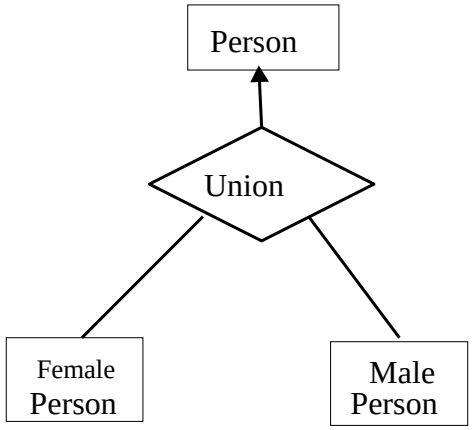


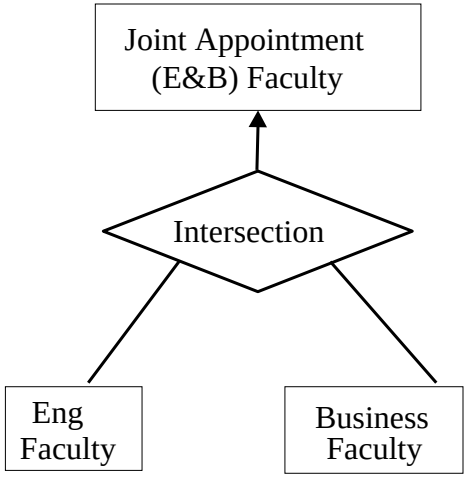
IS-A relationship is the same as the sub-class relationship in OO



Another notation

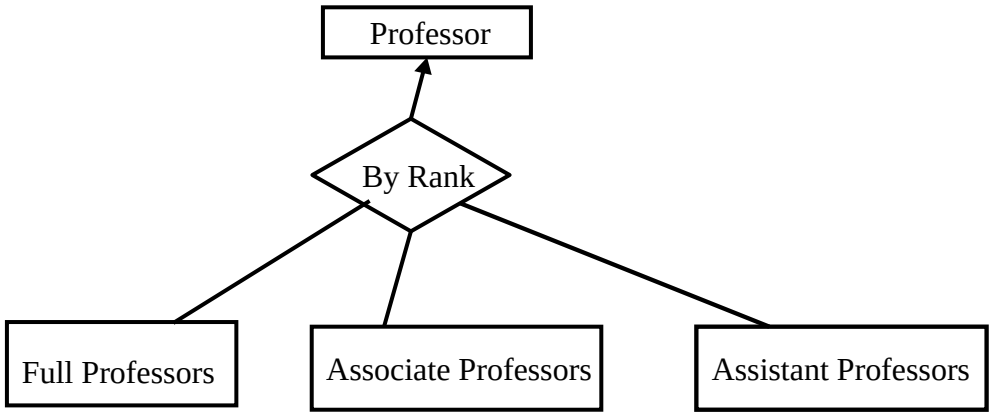
Set-operation relationships





Note the directions of the arrows.

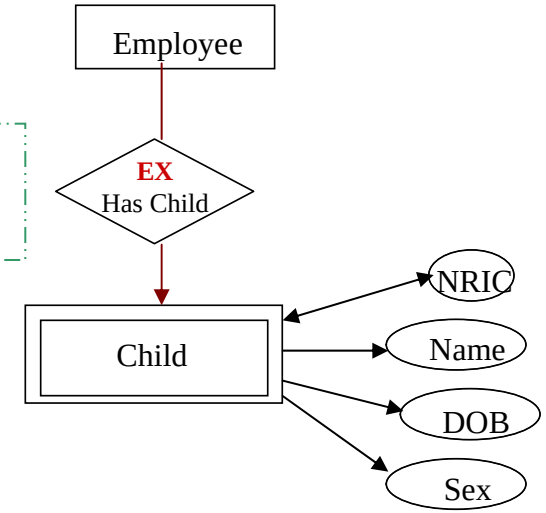
Decomposition relationship type



Note the directions of the arrows.

Existence-dependency (EX) relationship type and weak entity type

E.g.



Note the directions of the arrows.

FD: NRIC → Name, DOB, Sex

Note: NRIC is the identifier of the entity type Child.

The existence of a Child entity depends on the existence of an associated EMPLOYEE entity. Thus, if an Employee entity is deleted, its associated Child entities are also deleted.

The dependent entity type Child is a **weak entity type** (represented by a double rectangle), and Employee is a regular entity type.

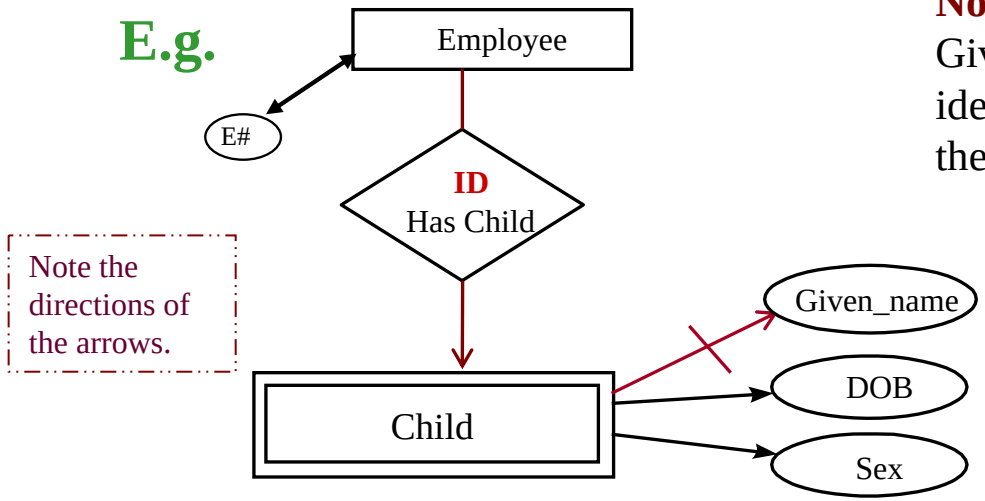
Relationship type which involves a weak entity type is called **existence-dependency relationship type** (denoted with "**EX**" together with a relationship type name)**.**

Note: An EX relationship type is a **1:m (one to many)** relationship type.

❖ Identifier-dependency (ID) relationship type

- An entity cannot be identified by the value of its own attributes, but has to be identified by its relationship with other entity. Such a relationship is called **identifier-dependency relationship**.

E.g.



Note: The line on the arrow to the attribute Given_name indicates this attribute together with the identifier of Employee (i.e. E#) form the identifier of the weak entity type Child. So, we have:

FD: **E#,** Given_name ⟶ DOB, Sex

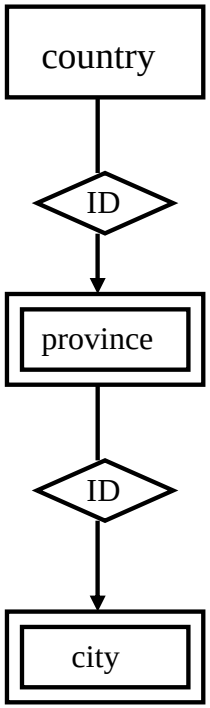
Note: The Child has no NRIC attribute.

Note: By the original definitions, an identifier-dependency relationship type (denoted by **ID**) is also an existence-dependency relationship type. However, we should not just indicate an **ID** as **EX**.

Q: What are the differences between this ER diagram and the previous page's ER diagram.

❖ **Note:** ID dependency relationships occur in XML data quite often.

Another example of Identifier-dependency relationship type



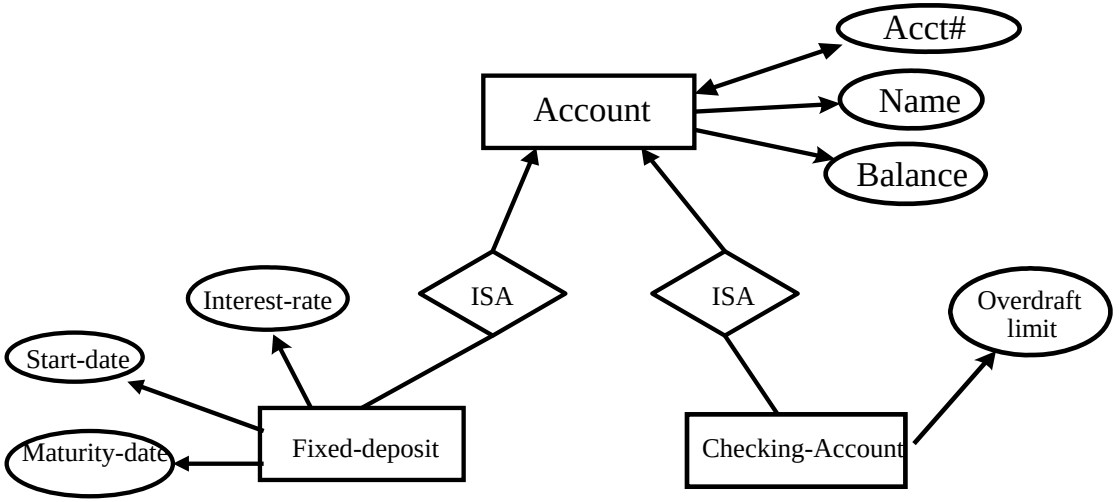
Note: Different provinces of a country may have cities with the same name. So, city name cannot be used to identify a city.

E.g. City Waterloo, Ontario, Canada
City Waterloo, Iowa, US
City Waterloo, Illinois, US

Q: What are the identifiers of the entity types province and city?

Generalization and Specialization

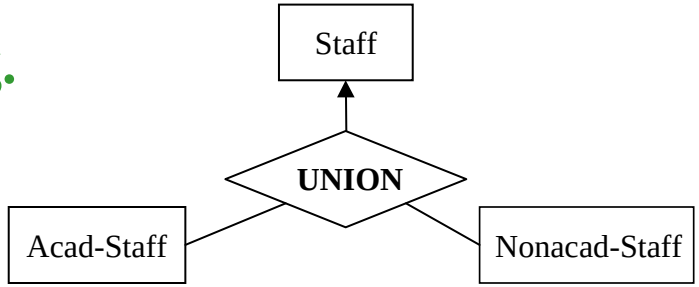
- **Generalization** is the result of taking the union of two or more (lower level) entity types to produce a higher level entity type.
- **Specialization** is the result of taking a subset of a higher-level entity type to form a lower-level entity type.
- Generalization is the same as UNION.
  Specialization is the same as ISA.
- In generalization, every higher-level entity must also be a lower-level entity.
  Specialization does not have this constraint.



(Specialization)

Note: There are other types of accounts, e.g., AUTOSAVE account and Fixed Deposit account.

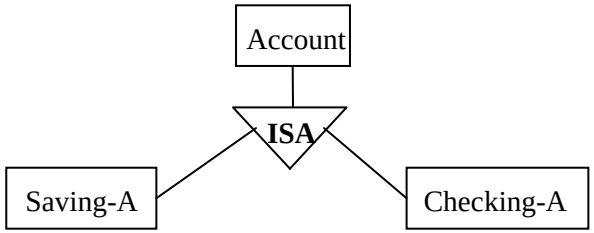
E.g.



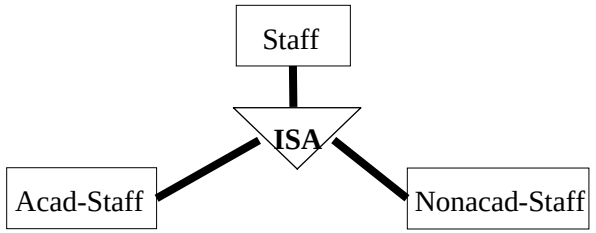
(Generalization)

- **Generalization** is used to emphasize the similarity among lower-level entity types and to hide their difference. **Specialization** is the inverse, i.e. to highlight the special properties of the lower level entity types.
- The attributes of the higher-level entity types are to be inherited by lower-level entity types.

Another notation (**Bad! Why?**)



(Specialization)

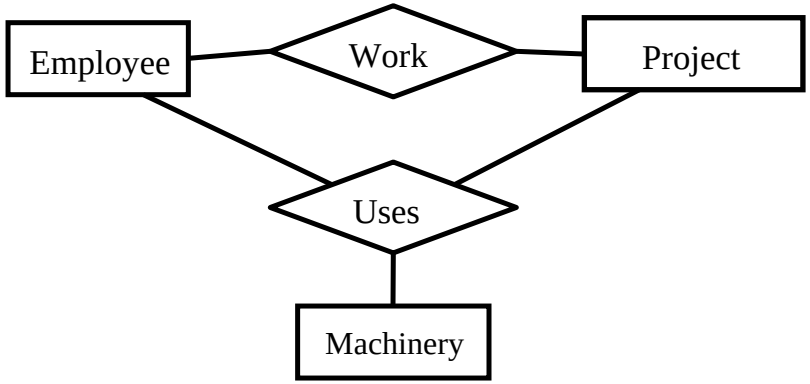


(Generalization)
(use thick lines)

Some **extensions** to the ER approach

(1) The ER approach does not allow **relationships among relationships**.

E.g. In an EMPLOYEE database, we want to describe information about employee who work on a particular project and use a number of different machines doing that work.
Using the basic ER approach, we may have the ER diagram.

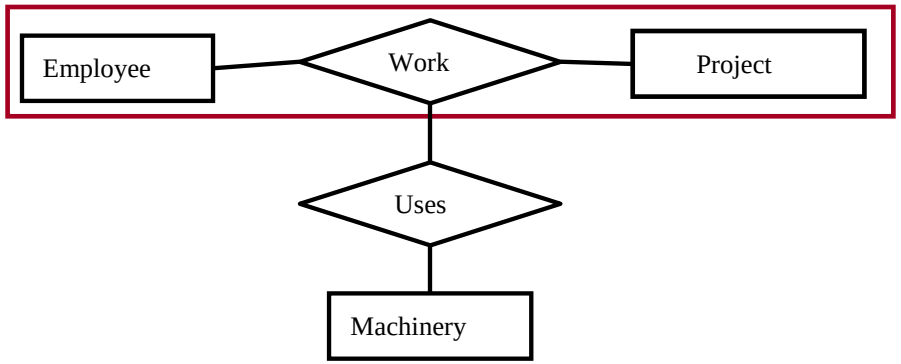


Note: The constraint between the two relationship types Work and Uses is not captured explicitly in the above ER diagram.

Q: How do we capture this constraint in an ER diagram?

Q: Can we combine the 2 relationship types into one?

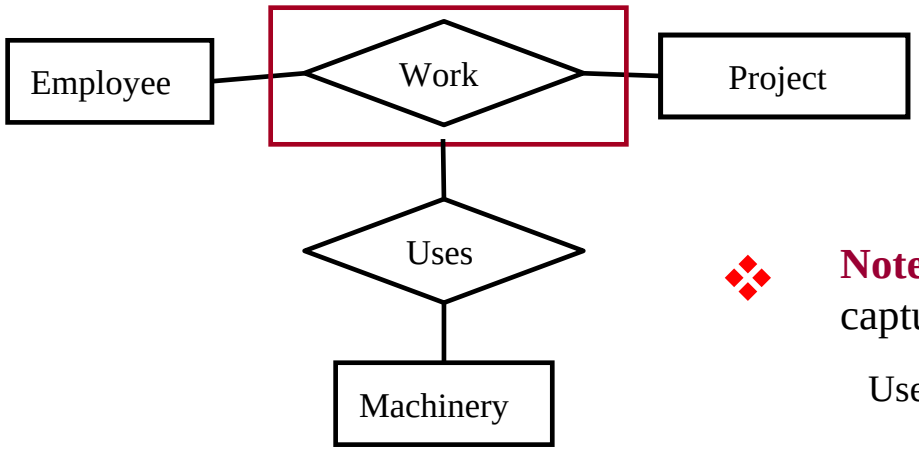
- One solution is use aggregation.
  Aggregation is an abstraction through which relationships are treated as higher-level entities.



(not a good notation)

In the ER diagram, we treat the relationship type Work and the entity types Employee and Project as a higher-level entity type called Work.

❖ My better notation:



Note: The line joining the diamond of the "Uses" relationship type and the aggregation does not touch the diamond of the "Work" relationship type.

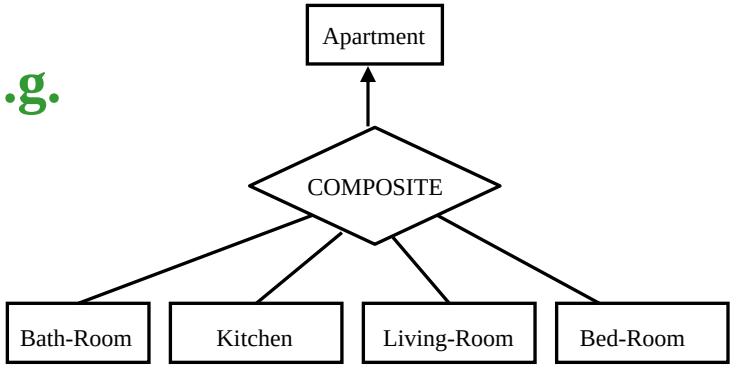
❖ **Note:** The constraint is explicitly captured by the 2 ER diagrams, i.e.

Uses [Employee, Project] ⊆ Work

(2) We introduce a new construct called **composite** for constructing complex objects (complex entities) from some other simple and/or complex objects.

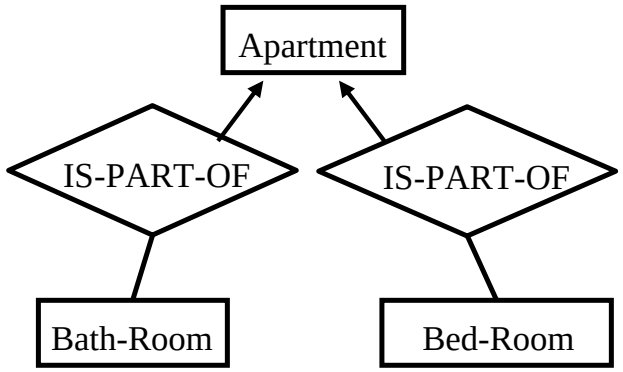
The complex object and its component objects are not necessarily type (or object) compatible.

E.g.



Note: COMPOSITE is similar to **IS-PART-OF** relationship but not exactly the same.

E.g..

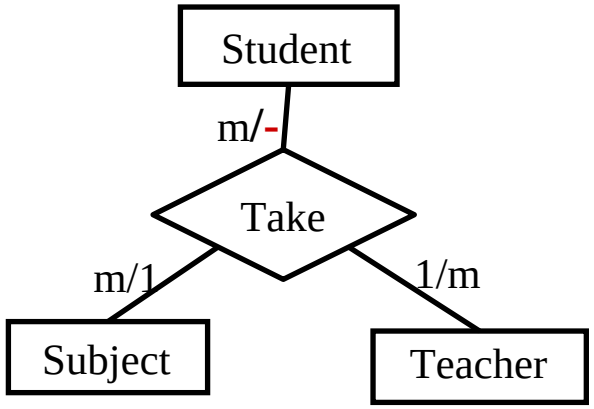


Note: An apartment may have other components, e.g. kitchen.

- (3) Represent more than one FD in a relationship type

  (using more than one set of cardinalities).

E.g.



There are 2 FDs in the relationship type Take:

Student, Subject $\rightarrow$ Teacher
Teacher $\rightarrow$ Subject

- **Note:** "-" means the entity type is not involved the respective set of cardinalities.

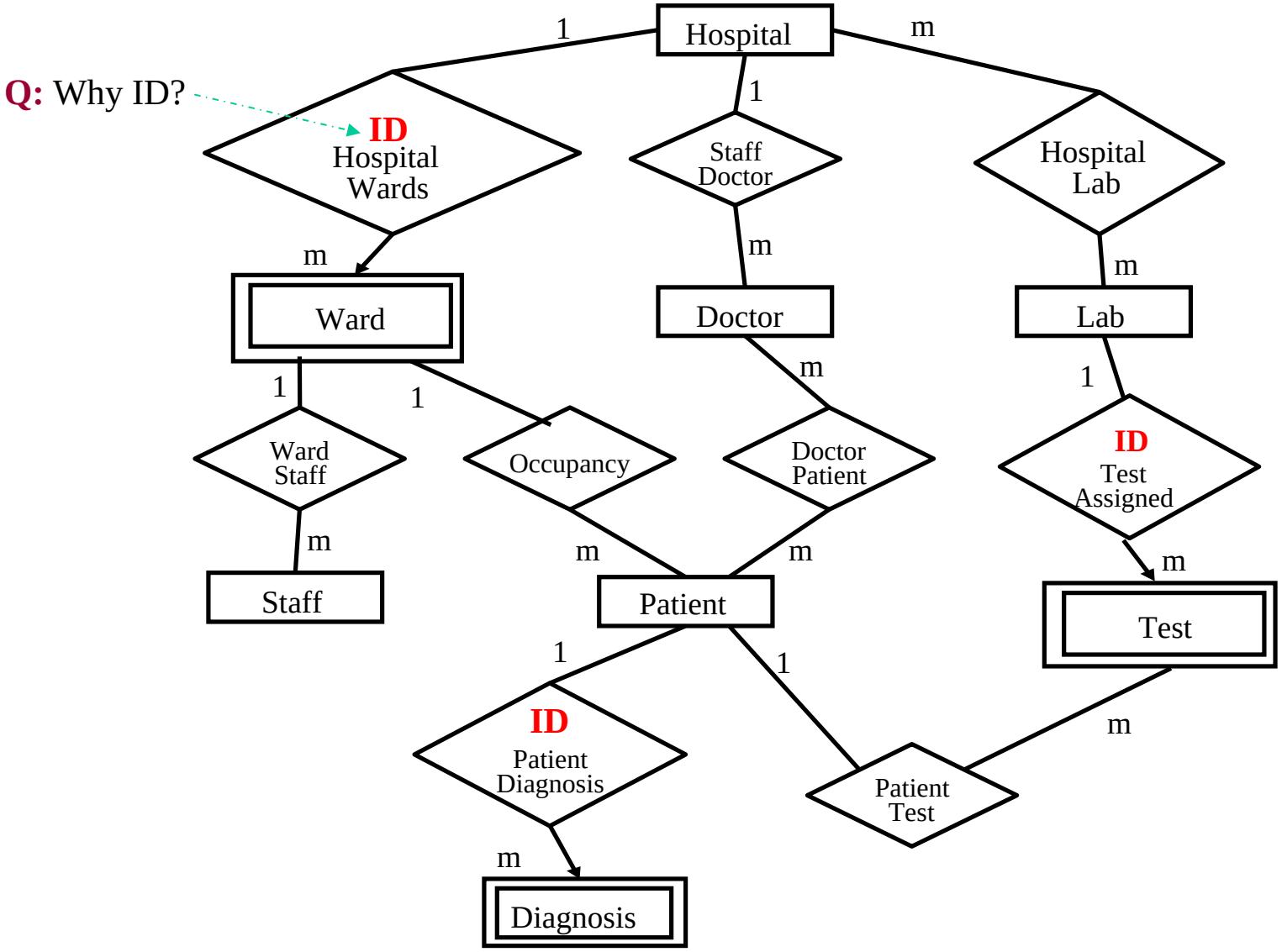


Figure: Entity-relationship diagram for medical database.

(from Tsichritzis's book)

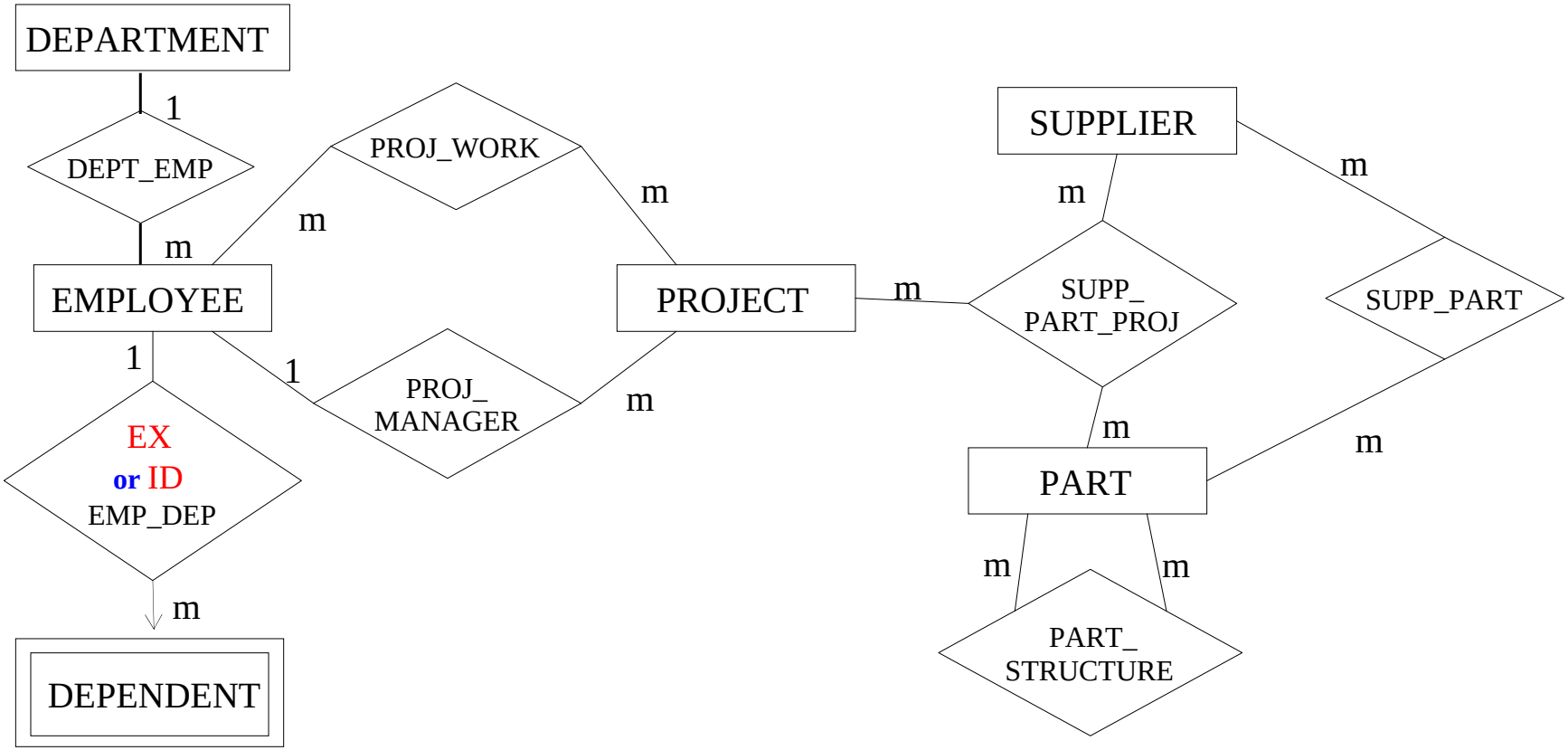


Figure: An Entity-relationship diagram.

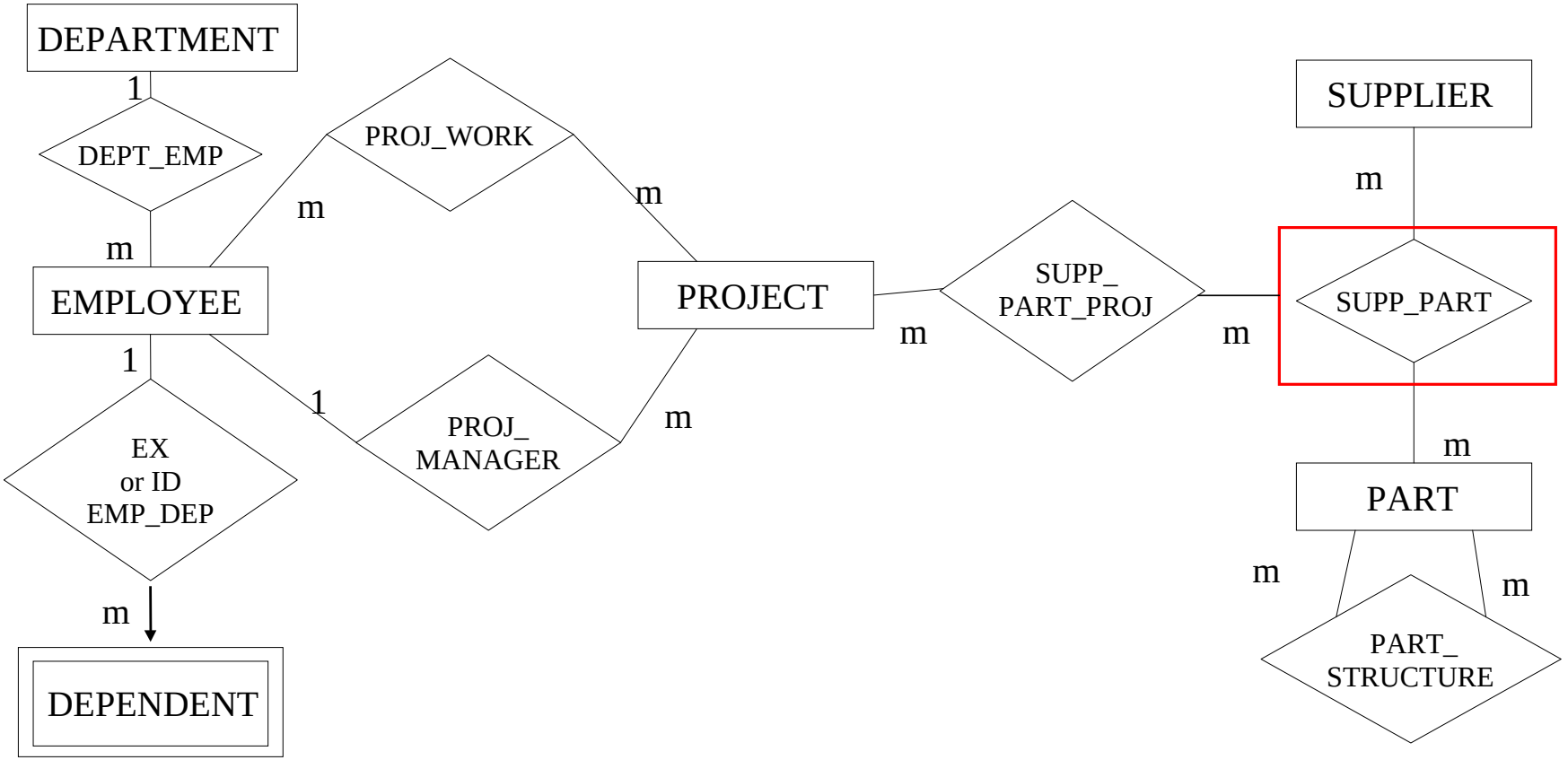
Note: The relationship type EMP_DEP can be EX or ID depending on whether DEPENDEMT has identifier or not.

Q: Is there any **constraint** between SUPP-PART and SUPP_PART_PROJ ?

Ans: Yes.

Q: What is the constraint?

(A better solution)



Constraint shown by the aggregation is:

SUPP_PART_PROJ [SUPPLIER, PART] ⊆ SUPP_PART [SUPPLIER, PART]

English Sentence Structure and ER Diagram

(**self study**)

Ref: Peter P Chen. English Sentence Structure and Entity-Relationship Diagrams. 1983 Inf Sci 29(2-3) :127-149.

In order to construct a database using an ER diagram, the database designer not only has to interview users but also must study the system specification documents which are written in some natural language, such as English.

Some guidelines/rules for translating English sentences into ER diagrams are presented below:

Guideline 1: A **common noun** (such as student and employee) in English corresponds to an entity type in an ER diagram:

common noun ⟶ entity type

Note: Proper nouns are entities not entity types, e.g. John, Singapore, New York City.

Guideline 2: A **transitive verb** in English corresponds to a relationship type in an ER diagram:

transitive verb ⟶ relationship type

Note: A transitive verb must have an object.

E.g. A person may own one or more cars and a car is owned by only one person.

Note: The cardinalities of person and car in the owns relationship type are 1 and m respectively, and the participation of person in the owns relationship type is **optional** (**or partial**) participation as some people may not own any car. Alternatively, we can have the participation constraints of person and car as **0:m** and **1:1** respectively. Since a car must have an owner, the participation of car in the owns relationship type is **mandatory** (or **total**) participation.

Guideline 3: An **adjective** in English corresponds to an attribute of an entity type in an ER diagram:

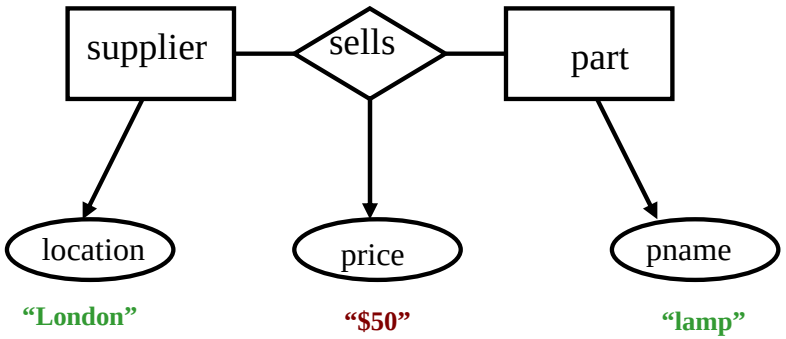
adjective ⟶ attribute of entity type

E.g. A **London** supplier, a **red** part, a **male** person.

Guideline 4: An **adverb** in English corresponds to an attribute of a relationship type in an ER diagram:

adverb ⟶ attribute of relationship type

E.g. A London supplier sells a part with part name lamp for $50.



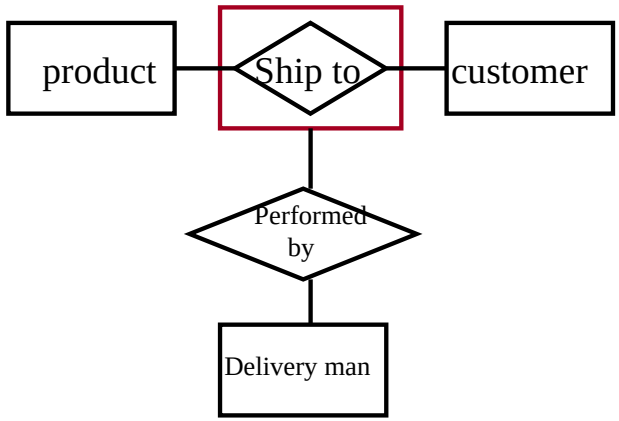
London and Lamp are adjectives (and attribute values) of supplier and part resp, and $50 is an adverb which is an attribute of the relationship type sells.

Guideline 5: A **gerund** in English corresponds to a high level entity type (or aggregation) converted from a relationship type in an ER diagram:

gerund ⟶ aggregation (or high level entity type)

Note: A gerund is a noun in the form of the present participle of a verb. For example 'shopping' in the sentence 'I like shopping‘.

E.g. Products are shipped to customers, and the shipping is performed by delivery men.

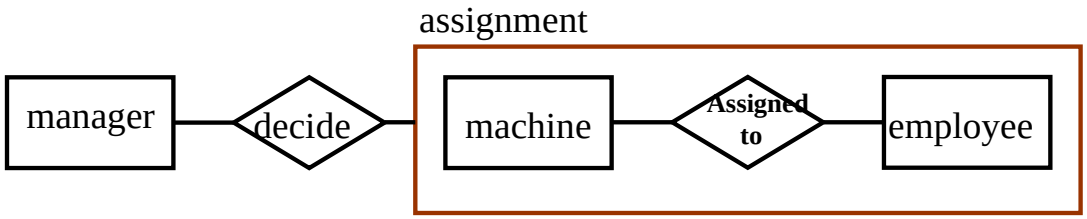


Guideline 6: A **clause** in English is a high level entity type abstracted from a group of interconnected low level entity and/or relationship types in an ER diagram:

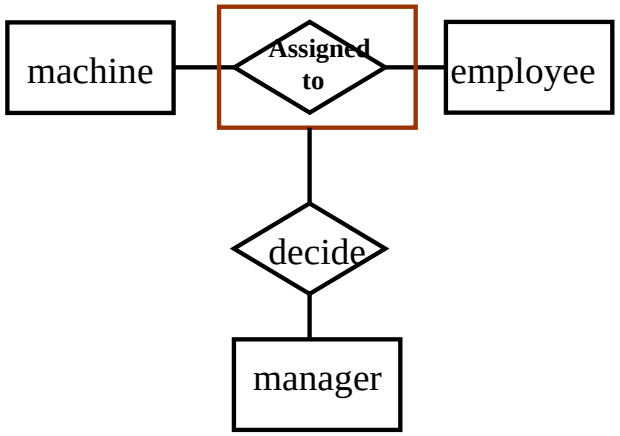
clause ⟶ aggregation (or high level entity type)

Note: A clause is a group of words that contains a subject and a verb, but it is usually only part of a sentence.

E.g. Managers decide which machine is assigned to which employee.



(One representation, not so good)



(Another representation, better one)

Q: Which representation is better?

ER Construct notation Comparison

ER Model Construct

used in Teorey's book
for **cardinality**

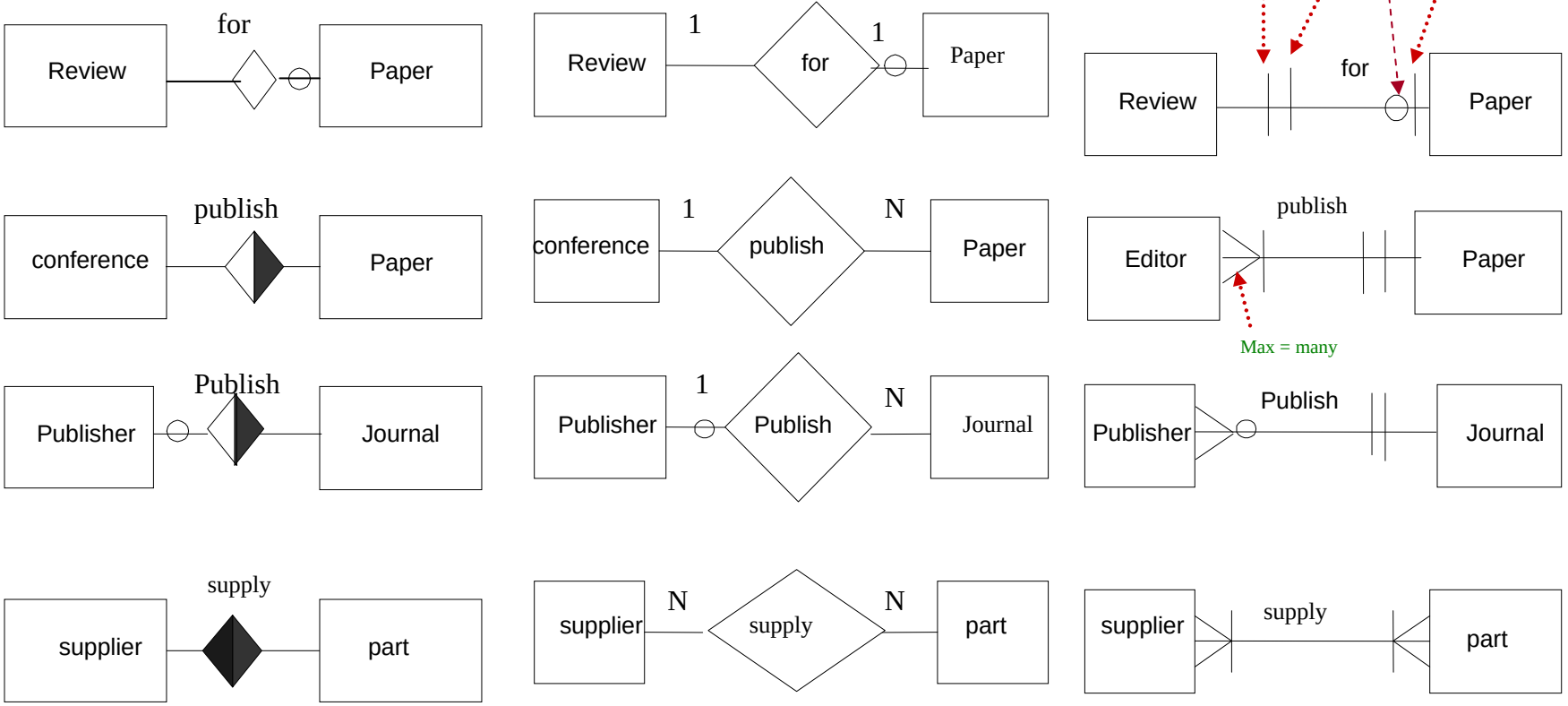
[Rein85]

ER Model Construct using

the **Chen approach**
for **cardinality**

ER Model construct using the

"**crow's foot**"- notation
for **participation constraint**
[Ever85, Knowledgeware]



Notation: O means optional participation.

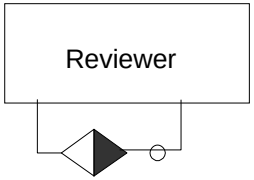
ER Construct Notation Comparison (cont.)

ER Model Construct	ER Model Construct using	ER Model construct using the
used in Teorey's book for **cardinality** [Rein85]	the **Chen approach** for **cardinality**	"**crow's foot**"- notation for **participation constraint** [Ever85, Knowledgeware]

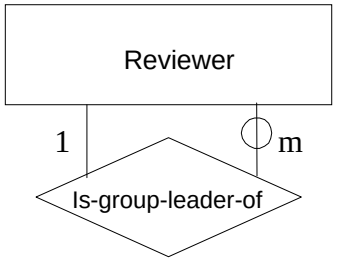
Weak Entity
or
Intersection Enity

Weak Entity

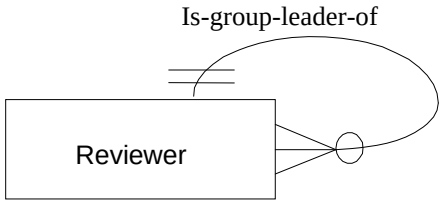
Intersection Entity



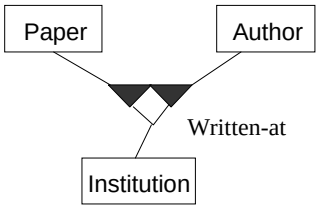
Recursive relationship



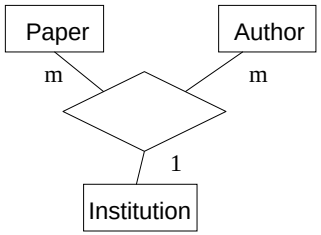
Recursive relationship



Recursive entity



N-ary relationship



ER Model

?
Can't represnt
n-ary relationships

ER Construct Notation Comparison (cont.)

ER Model Construct	ER Model Construct using	ER Model construct using the
used in Teorey's book for **cardinality** [Rein85]	the **Chen approach** for **cardinality**	"**crow's foot**"- notation for **participation constraint** [Ever85, Knowledgeware]

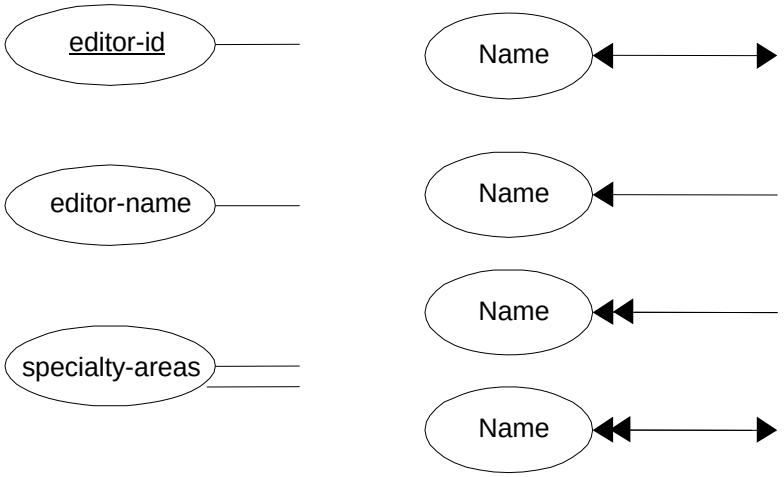
---

identifier (key)

descriptor (nonkey)

multivalued descriptor

m:1 Attribute



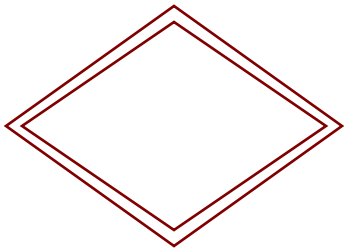
No attributes are allowed for relationships

No attributes for relationships

Q: What are the strong and weak points of each notation?

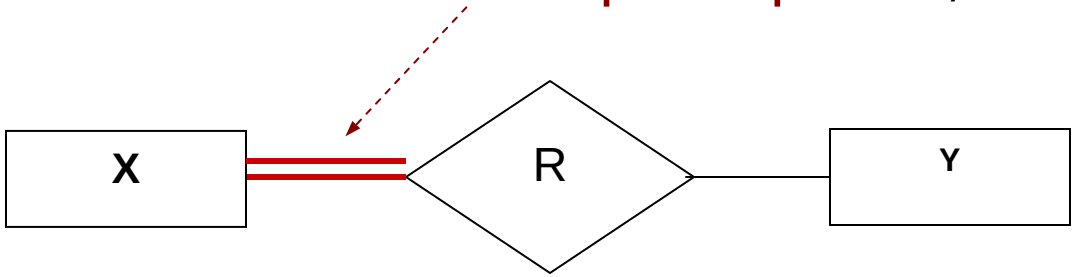
Some other ERD notations

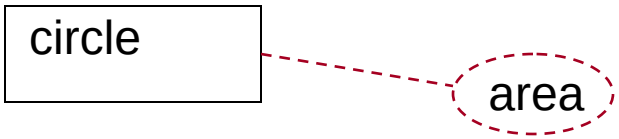
Multivalued attribute



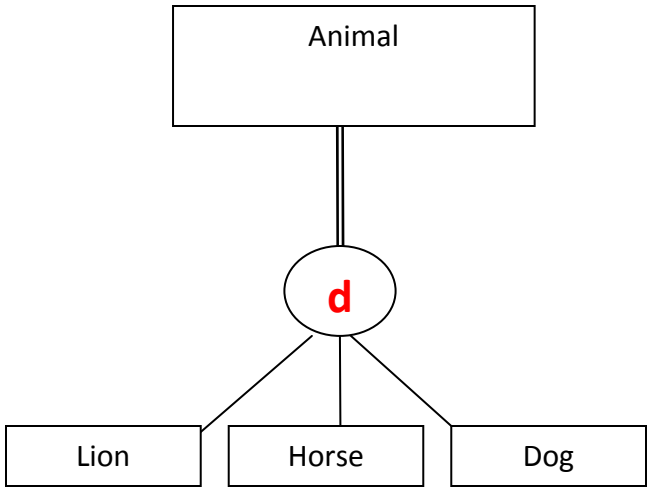
ID dependency relationship

Total participation, i.e. min occur is 1

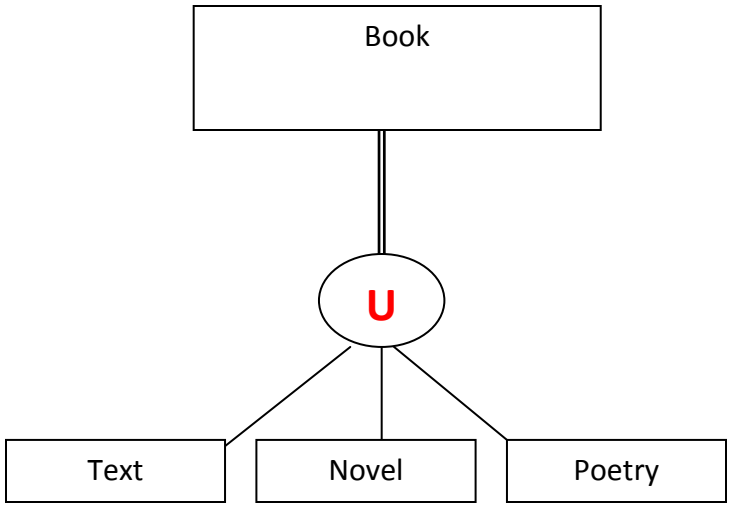




Dotted oval and line indicate derived attribute



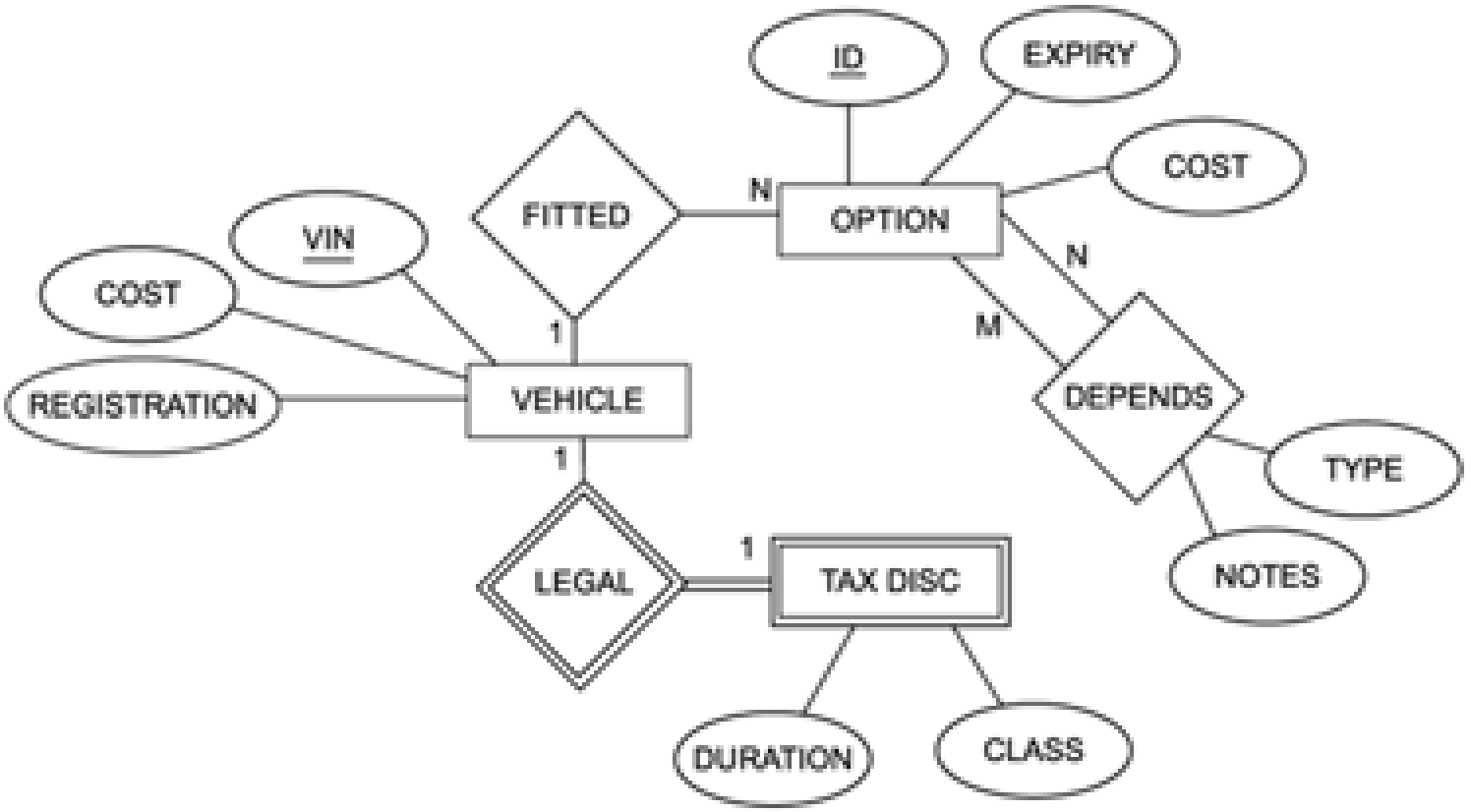
Disjoint – every member of the super-class can belong to at most one of the subclasses. For example, an Animal cannot be a lion and a horse, it must be either a lion, a horse, or a dog.



Union – every member of the super-class can belong to more than one of the subclasses. For example, a book can be a text book, but also a poetry book at the same time.

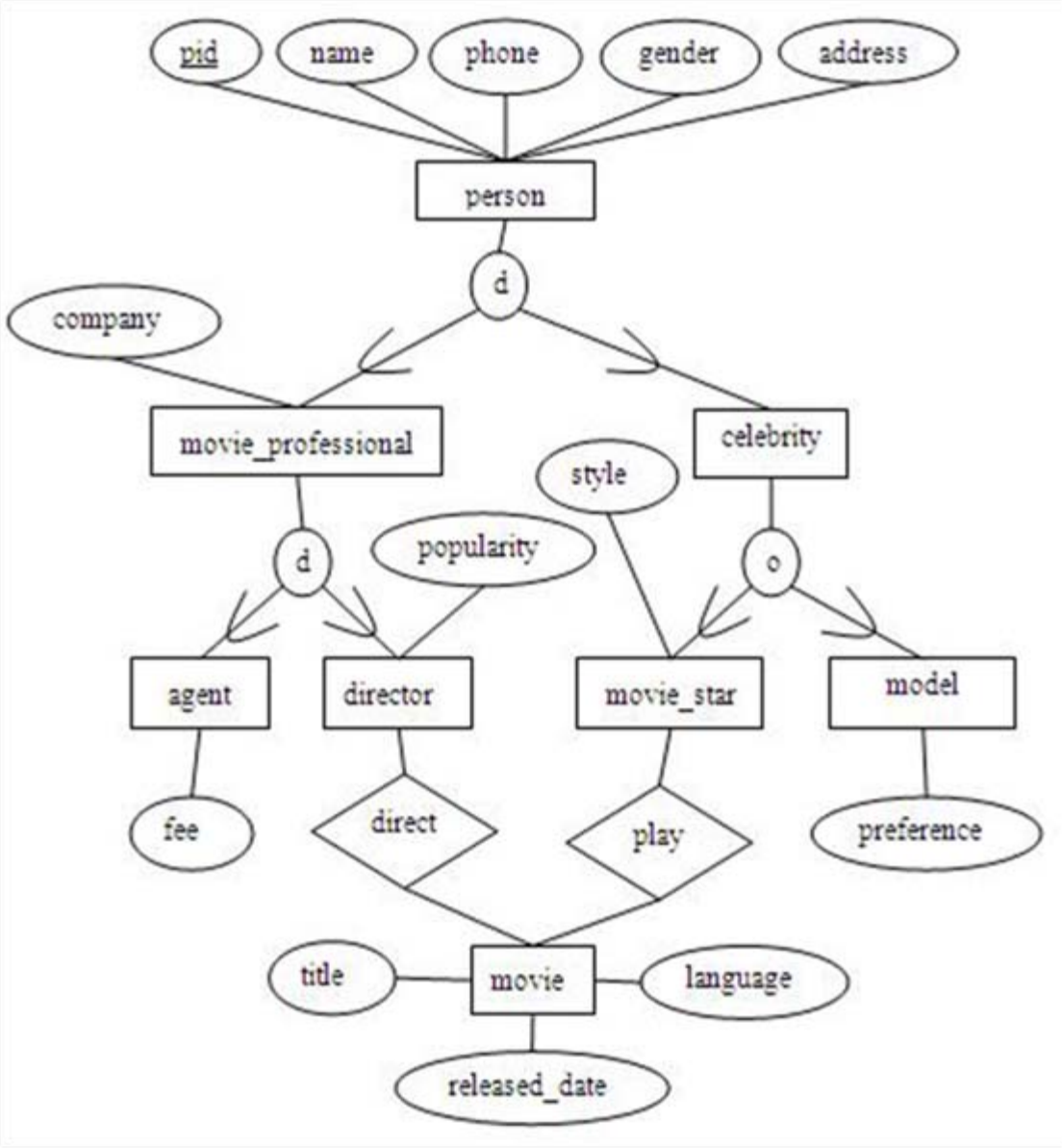
Example ERD notation.

Underlined attributes are identifiers (key attributes) of entity types.



Example ERD notation.

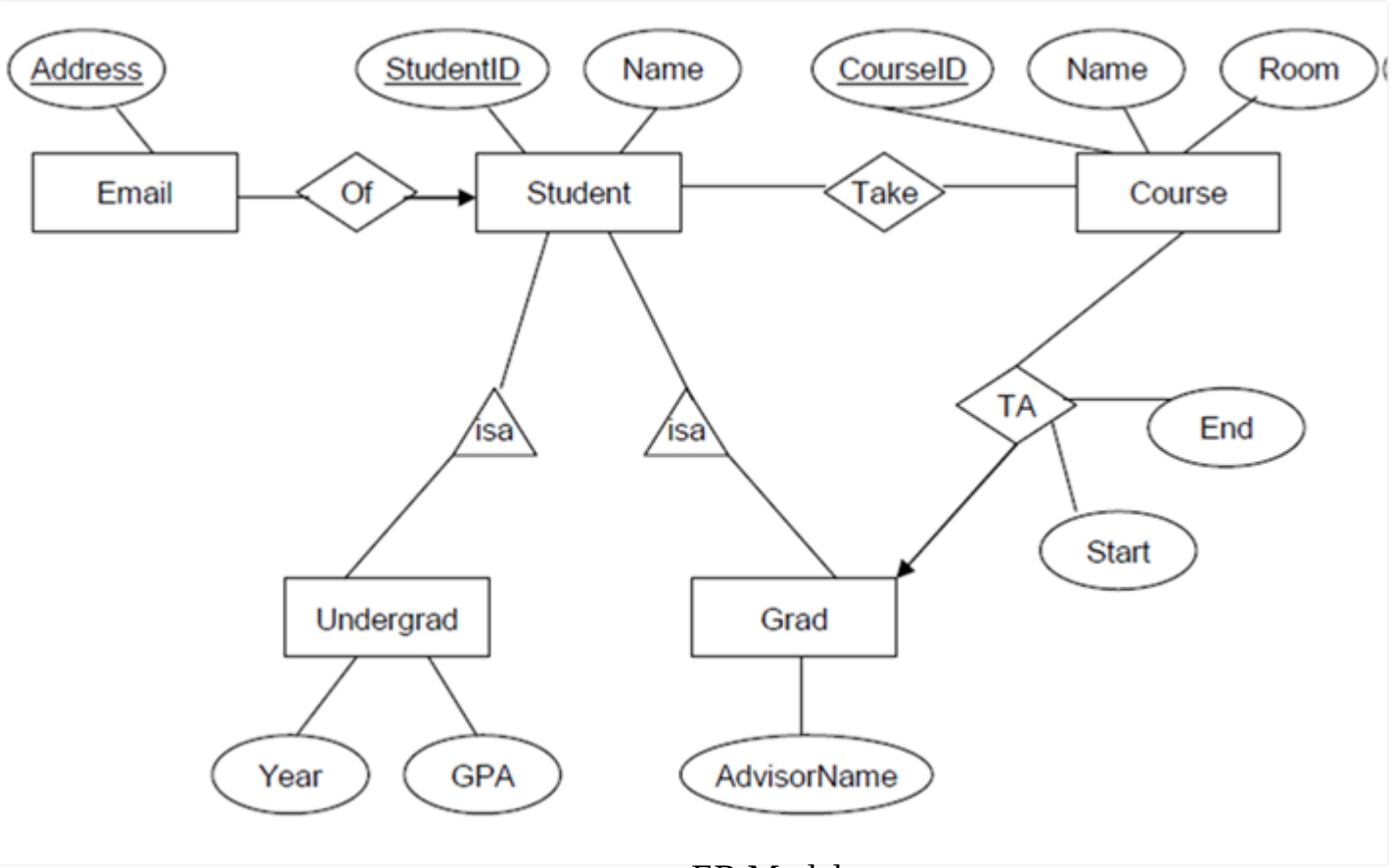
Notations for **disjoint** and **overlap** subclasses.



Another ERD notation.

Arrow of a relationship type show the functional dependency.

E.g. Course → Grad in TA relationship type;
Email → Student in Of relationship type.



Steps of database design using ER Approach

Step 1. Understand the problem domain. Analyze database requirements.

- Write a summary specification if not created yet.
- What do we need to store into the database?
- What queries and reports do we need to generate?
- What are *important* people, places, physical things, organizations, events and abstract concepts in the organization?

Step 2. Design a conceptual database schema by creating an ER diagram. Detail will be discussed.

Step 3. Design a logical database schema.

- Convert the ERD to a normal form ERD.
- Translate the ERD into a relational schema.

Note: Another approach. First translate the ERD to a relational database schema, then normalize the relations in the schema to at least 3NF.

Step 4. Design a Physical database schema.

- **Denormalization**. May need to de-normalize some normal form relations for better performance.
  - * adding redundant attributes in some relations
  - * adding derived attributes in some relations
  - * merging/splitting relations.
- choosing primary keys and foreign keys
- defining indexes
- Do database prototyping & modify the design if necessary.
- Summarize the design assertion (integrity, security).

Detail will be discussed in the Physical Database Design chapter.

Step 5. Verify the design with users. Iterate the steps if necessary.

Design a conceptual schema by creating an ER diagram

Step 1. Identify the entity types.

Step 2. Identify the relationship types and their participating entity types.

Step 3. Identify the attributes, keys, and identifier of each entity type and relationship type and obtain an ER diagram

Step 4. Convert the ER diagram to a normal form ER diagram.

Step 5. Translate the NF-ER diagram to a relational database schema (or other data models)

Question: Is the resulting relational schema in normal form?

Advantages:

- DBMS independent
- Concentrate on "information requirements" not on "storage or access efficiency", i.e. conceptual design.
- Easy to understand by users and database designers

Some decision rules/guidelines

Rule 1:

Every entity type should be important in its own right within the problem domain.

Rule 2:

IF an entity type (noun) has only one property to store and relates to only one other entity type

THEN it is an attribute of another entity type

ELSE it is an entity type.

Rule 3:

IF an entity type has only one data instance

THEN do not model as an entity type.

Rule 4:

IF a relationship type needs to have a unique identifier

THEN model it as an entity type.

The first three rules are used to evaluate entity/object types or nouns, and the fourth rule is used to evaluate relationship types or verbs.

Example 1. (from C J Date's book Question 14.2)

A database used in an order-entry system is to contain information about customers, items, and orders. The following information is to be included.

- For each **customer**:
  - Customer number (unique)
  - Valid "ship to" addresses (several per customer)
  - Balance
  - Credit limit
  - Discount

- For each **order**:
  - **Heading information**: customer number, "ship-to" address, date of order.
  - **Detail lines** (several per order), each giving item number, quantity ordered

- For each **item**:
  - Item number (unique)
  - Warehouses
  - Quantity on hand at each warehouse
  - Item description

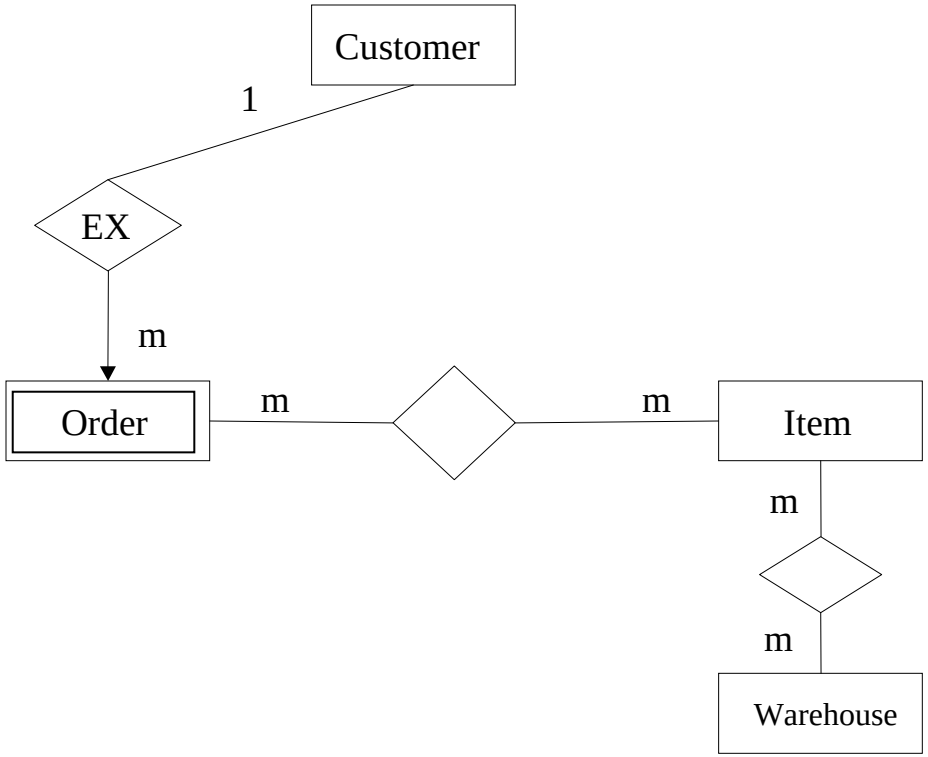
For internal processing reasons, a "**quantity outstanding**" value is associated with each detail line of each order. [This value is initially set equal to the quantity of the item ordered and is progressively reduced to zero as partial shipments are made.]

Design a database for this data. As in the previous question, make any semantic assumptions that seem necessary.

Example 1. (cont.) An Order-Entry System

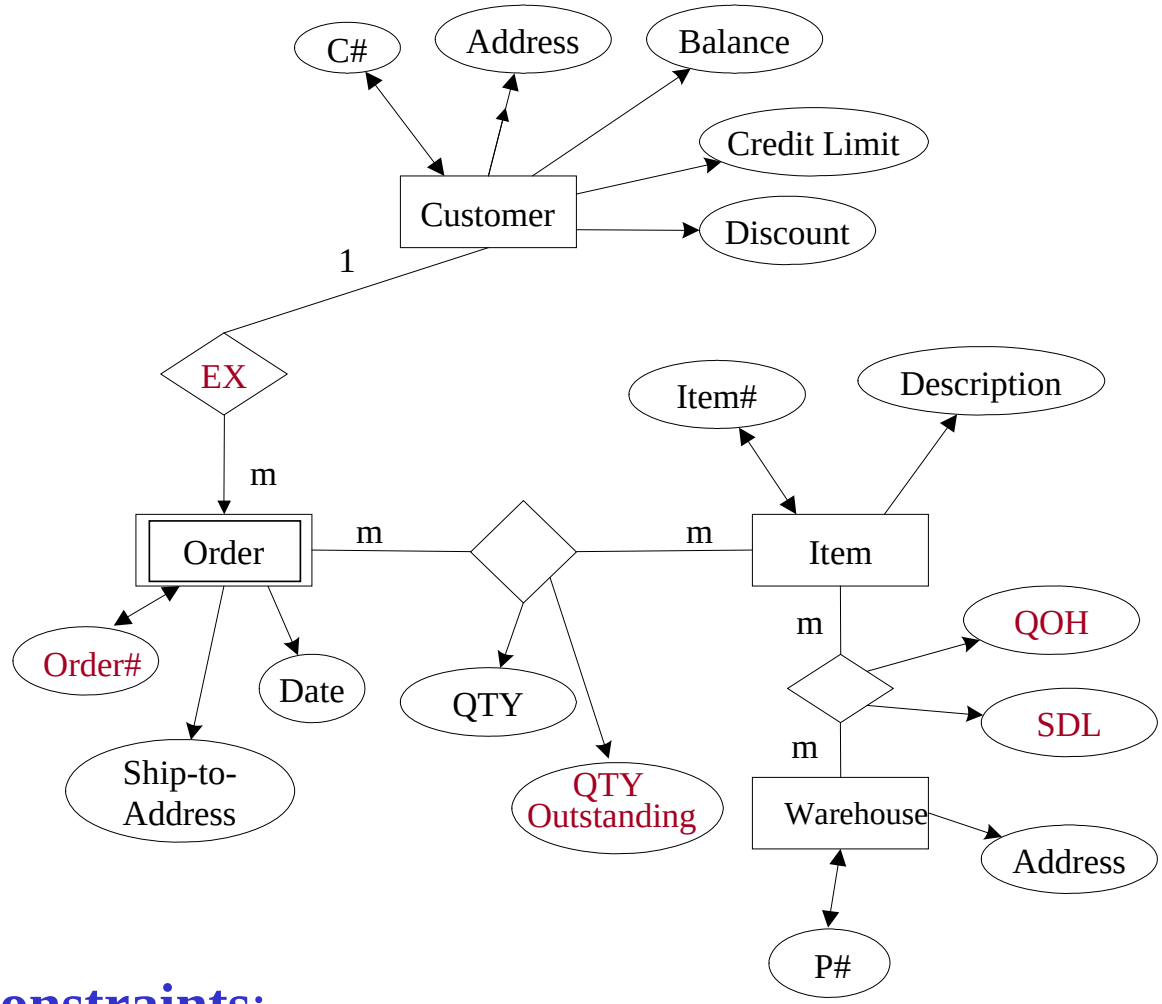
(Question 14.2 in CJ. Date's book)

- First, only decide entity types and relationship types.



Example 1. (cont.) An Order-Entry System

- then add attributes of entity types and relationship types.

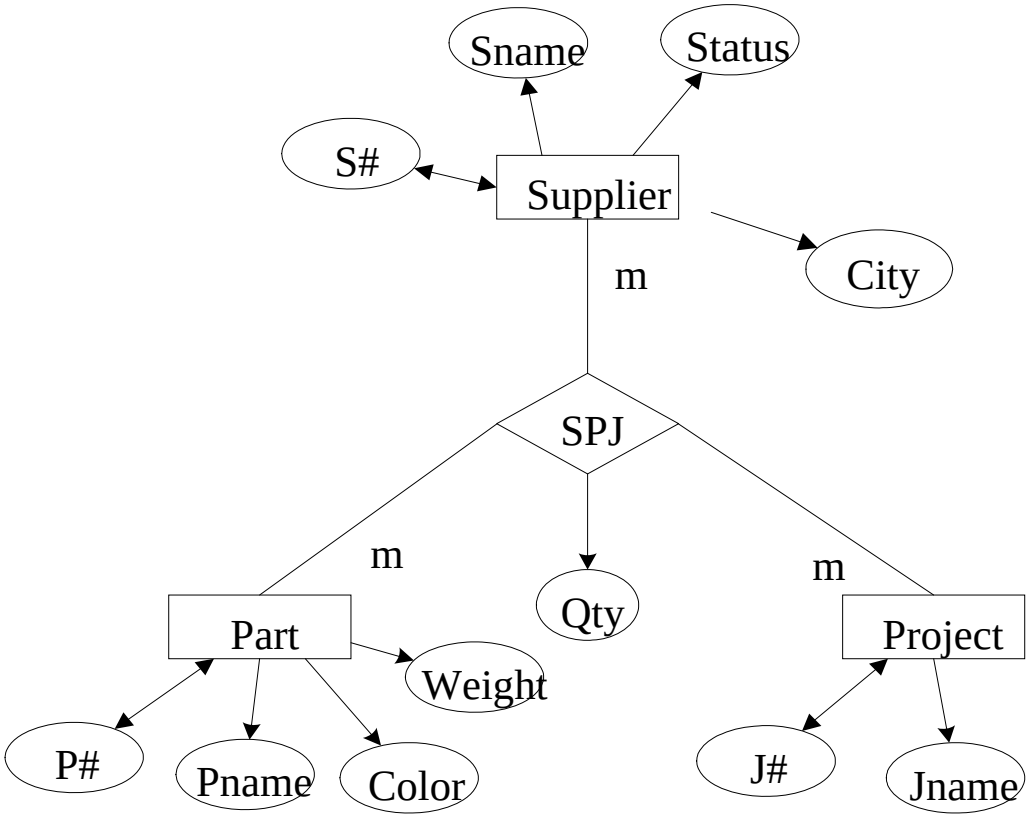


SDL – Stock Dangerous Level

Constraints:

Between Address & Ship-to-Address

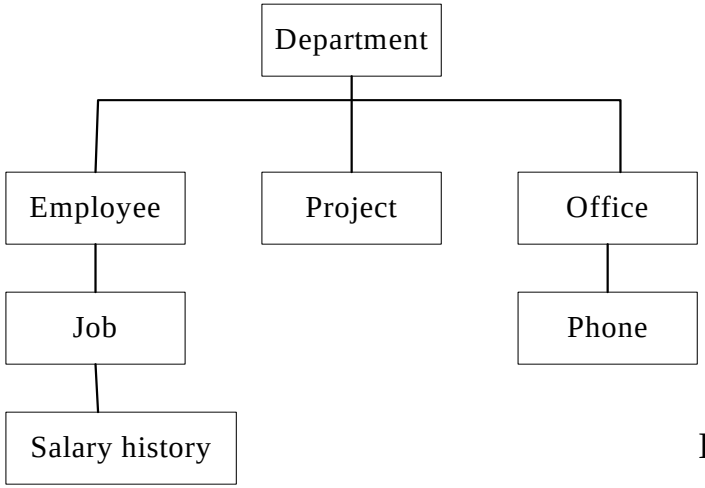
Example 2. Recall the supplier-part-project database - SPJ



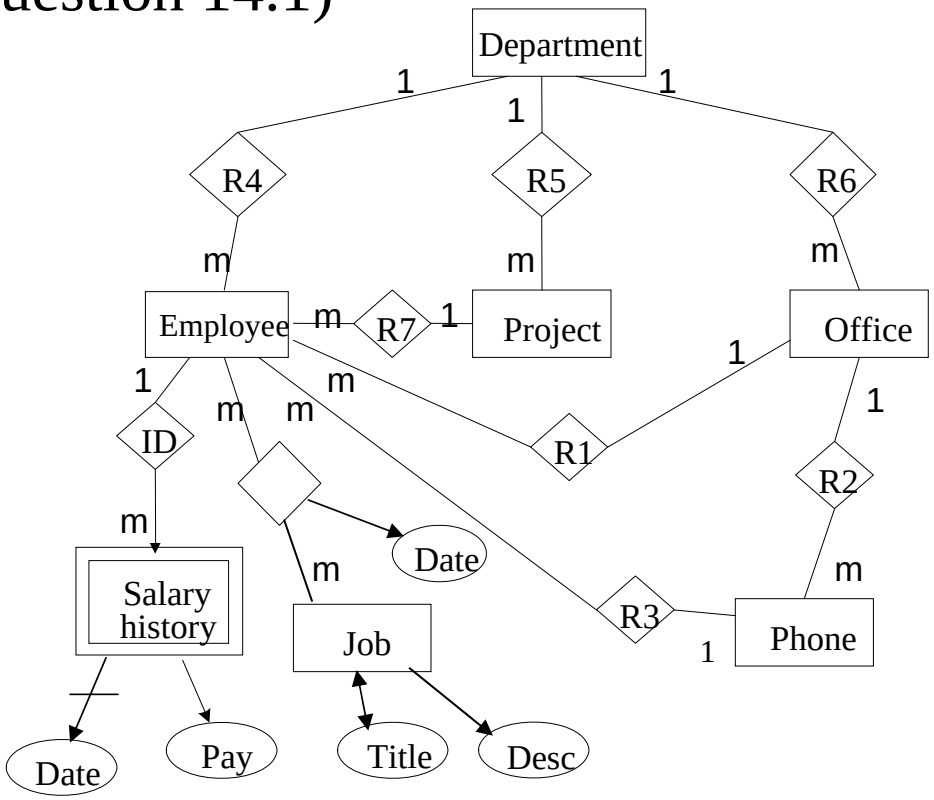
Example 3. (From C J. Date's book Question 14.1)

For each **department** the database contains a department number (unique), a budget value, and the department manager's employee number (unique). For each department the database also contains information about all **employees** working in the department, all **projects** assigned to the department, and all **offices** occupied by the department. The employee information consists of employee number (unique), the number of the project on which he or she is working, and his or her office number and **phone** number; the project information consists of project number (unique) and a budget value; and the office information consists of an office number (unique) and the area of the office in square feet. Also, for each employee the database contains the title of each job the employee has held, together with date and **salary** for distinct salary received in that job; and for each office it contains the numbers (unique) of all phones in that office.

Convert this **hierarchical structure** to an appropriate collection of normalized relations. Make any assumptions you deem reasonable about the dependencies involved.



Example 3. (cont.) (Question 14.1)



There are cycles in the ERD, each cycle may represent a constraint.

E.g. $R3 \subseteq R1 * R2$ [Employee, Phone]

$R1 = R3 * R2$ [Employee, Office]

$R4 = R7 * R5$ [Employee, Department]

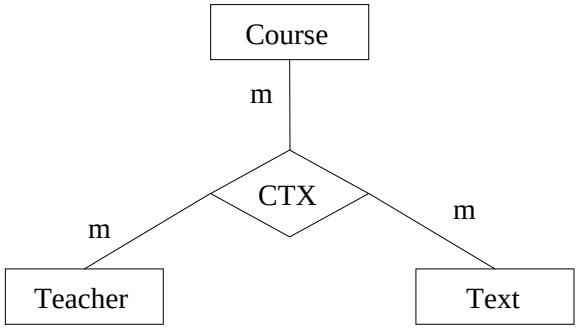
$R4 = R1 * R6$ [Employee, Department]

Q: What are the meanings of the above constraints?

Q: How to know whether these are really constraints or not?
Automatically or manually?

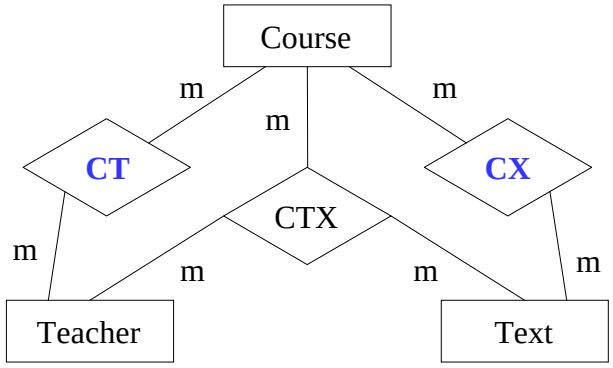
Example 4. Recall the relation CTX (Course, Teacher, Text) which has a **Multivalued dependency (MVD):**

Course $\longrightarrow\!\!\!\!\!\rightarrow$ Teacher | Text



Note: CTX has a MVD

Should be:



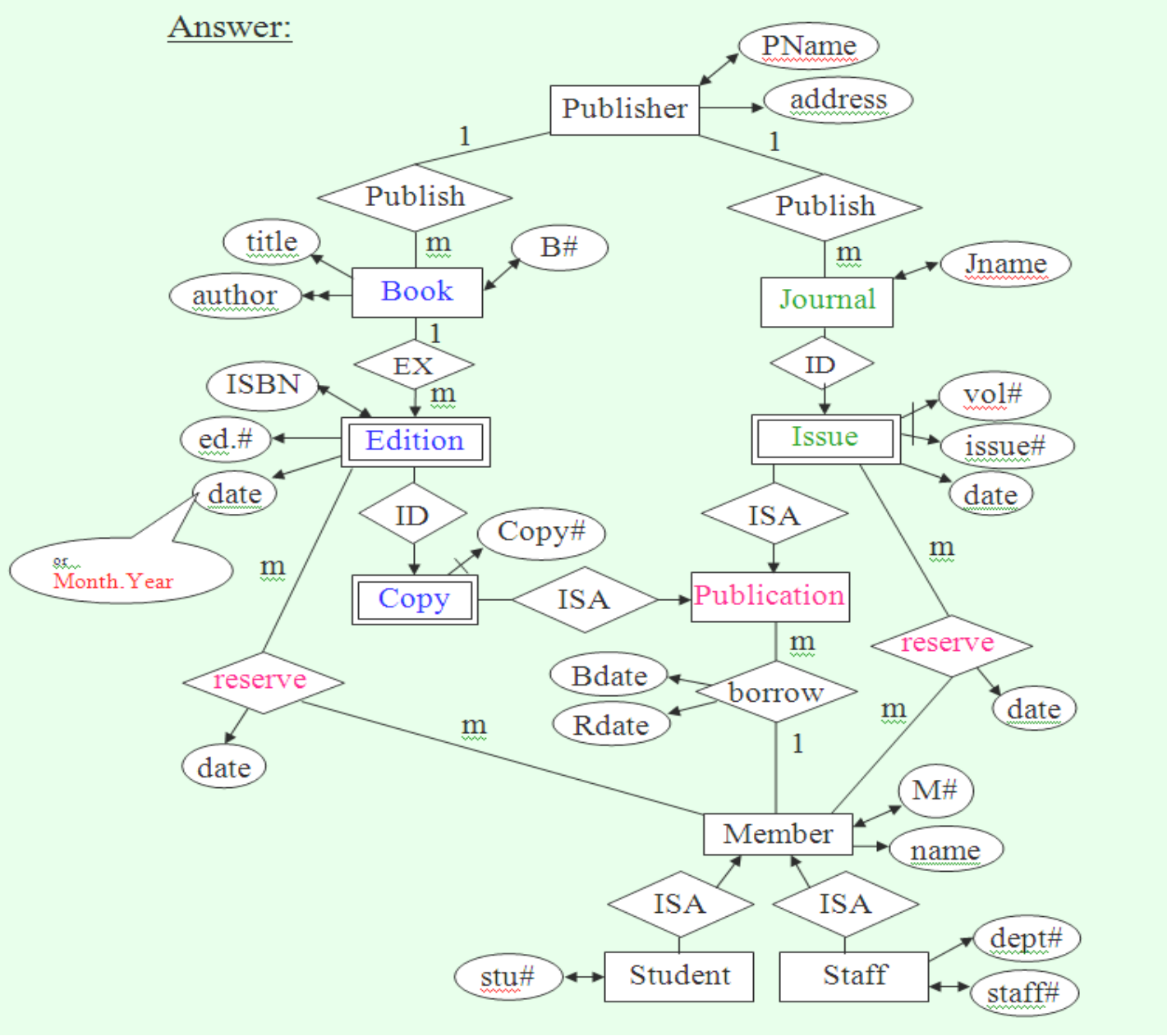
Note: CTX can be derived:

$CTX = CT \bowtie CX$

Example 5. Library Information System.

Draw a normal form ER diagram for a university library information system which stores information about books, journals, publishers, students, staff, borrowing of books, and reservation of books. Note that the library may have more than one copy for some of the books.

Example 5. (solution)



Example 6. Online Auction

Consider an ONLINE AUCTION database system in which members (buyers and sellers) participate in the sale of items. The data requirements for this system are summarized as follows:

- The online site has **members**, each of whom is identified by a unique member number and is described by an e-mail address, name, password, home address, and phone number.
- A member may be a **buyer** or a **seller**. A buyer has a shipping address recorded in the database. A seller has a bank account number and routing number recorded in the database.
- **Items** are placed by a seller for sale and are identified by a unique item number assigned by the system. Items are also described by an item title, a description, starting bid price, bidding increment, the start date of the auction, and the end date of the auction.
- Items are also categorized based on a fixed **classification hierarchy.**

Example 6. Online Auction (cont.)

- Buyers make **bids** for items they are interested in. Bid price and time of bid is recorded. The bidder at the end of the auction with the highest bid price is declared the winner and a transaction between buyer and seller may then proceed.

- The buyer and seller may record feedback regarding their completed transactions. Feedback contains a rating of the other party participating in the transaction (1-10) and a comment.

Design an Entity-Relationship diagram for the ONLINE AUCTION database. (Do it yourself).

Translation of a (normal form) ER diagram to a relational database

Step 1. Assign role names to certain arcs in a **cycle** in the ER diagram (in order to conform to the universal relation assumption).

Note. Here, a cycle in an ER diagram is defined as a cycle in the corresponding graph of the ER diagram in which all entity types and relationship types are nodes in the graph and arcs which connect entity types and relationship types are edges in the graph, except for cycles formed only by ISA, UNION, INTERSECT, and DECOMPOSE special relationships.

- Note: We can instead use the **relaxed universal relation assumption** which only requires the identifiers of entity types be unique.

Step 2. Assign identifiers for entity types involved in special relationships such as: ISA, UNION, INTERSECT, DECOMPOSE, and so far with no identifiers yet.

Step 3. Generate relations for each entity type.

(1) **All** the m:1 and 1:1 attributes of an entity type E form a relation. The keys and identifier of E are the keys and primary key of the generated relation.

- (2) **Each** m:m attribute and the identifier of E form an **all key relation**.
- (3) **Each** 1:m attribute and the identifier of E form a relation with the 1:m attribute as its key.

Note. All composite attributes are replaced by their components in the generated relations.

Q: How about weak entity type?

Step 4. Generate relations for each regular relationship type R.

(1) **All** the identifiers (or role names) of the entity types participating in R and all m:1 and 1:1 attributes of R form a relation. Keys and 1:1 attributes of R are keys of the generated relation, and the identifier of R is the primary key of the generated relation.

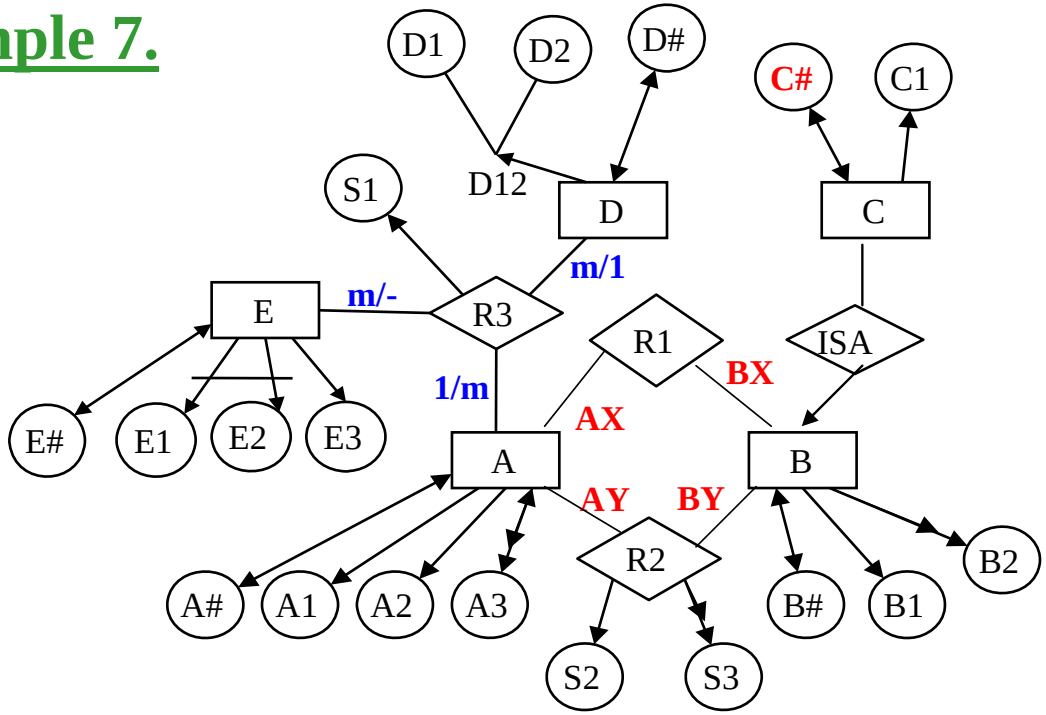
Further more, if $A \rightarrow B$ is a full FD in the generated relation and A is not a key of R, then we record $A \rightarrow B$ as a constraint of the relation generated.

❖ **(2)** **Each** m:m attribute of R and the identifier of R form an all key relation (i.e. all attributes of the relation form the only key of the relation).

(3) **Each** 1:m attribute of R and the identifier of R form a relation with the 1:m attribute as the key of the relation.

❖ **Q:** How about special relationships such as ISA, EX, ID, etc., and aggregations?

Example 7.



- The role names AX, AY, BX, BY are assigned by step 1.
- Note that the relationship type R3 has two FDs:

  E#, D# → A#

  A# → D#

❖ {E#, D#} and {A#, E#} are keys of R3, and we designate {E#, D#} as the identifier of R3 .

Q: How to know there is another key (i.e. {A#, E#}) and how to find it?

Relations generated are:

(1) For entity type A

AE1(A#, A1, A2)

AE3(A3, A#)

(2) For entity type B

BE1(B#, B1)

BE2(B#, B2)

(3) For entity type C

CE1(C#, C1) (C# is generated)

Constraint: C# ISA B# of BE1

i.e. CE1[C#] $\subseteq$ BE1[B#]

(4) For entity type D

DE1(D#, D1, D2) (D12 is replaced by D1 and D2)

(5) For entity type E

EE1(E#, E1, E2, E3) E# is primary key.

(6) For relationship type R1

R1R1(AX, BX) (AX, BX are generated)

Constraints: AX ISA A#, BX ISA B#.

(7) For relationship type R2

R2R1(AY, BY, S2) (AY, BY are generated)

R2R2(AY, BY, S3)

Constraints: AY ISA A#, BY ISA B#

(8) For relationship type R3

R3R1(A#, D#, E#, S1)

❖ {D#, E#} is the primary key and {A#, E#} is another key

Constraint: A# → D#

Note that (1) R3R1 is in 3NF but not in BCNF.

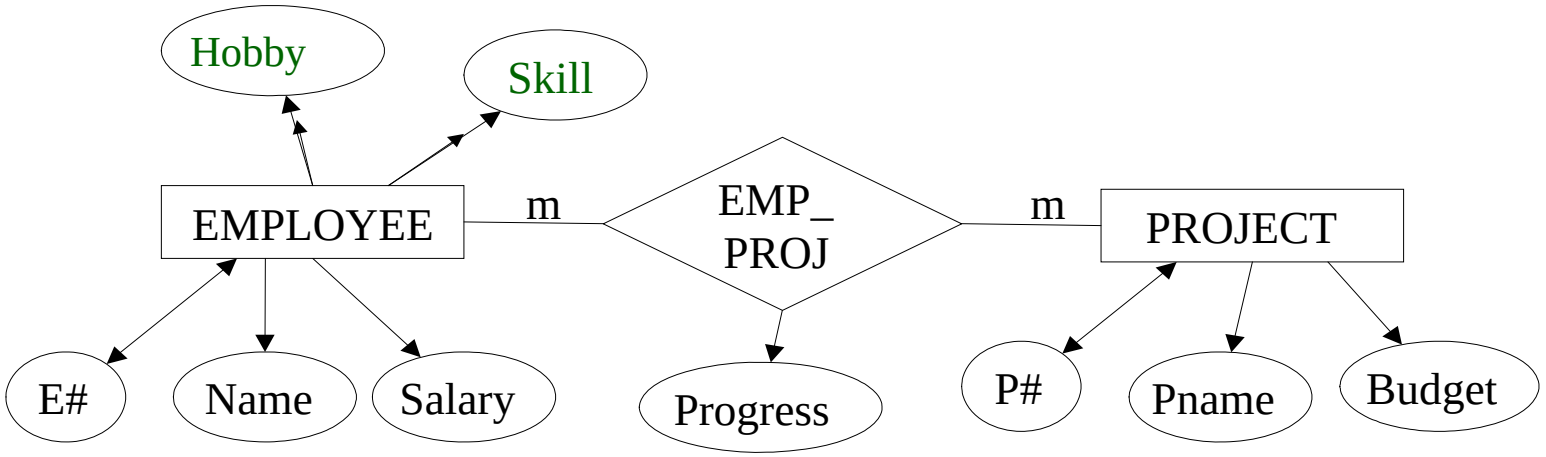
(2) **No** relation is generated for ISA relationship, it is translated to: C# ISA B#.

Questions:

(1) Are all relations generated for each of the entity types in 3NF? BCNF? 4NF?

(2) Are all the relations generated for each of the relationship types in 3NF? BCNF? 4NF?

Example 8.



Entity Relations:

Employee (*E#*, Name, Salary)

- Employee_Hobby (E#, Hobby)
- Employee_Skill (E#, Skill)

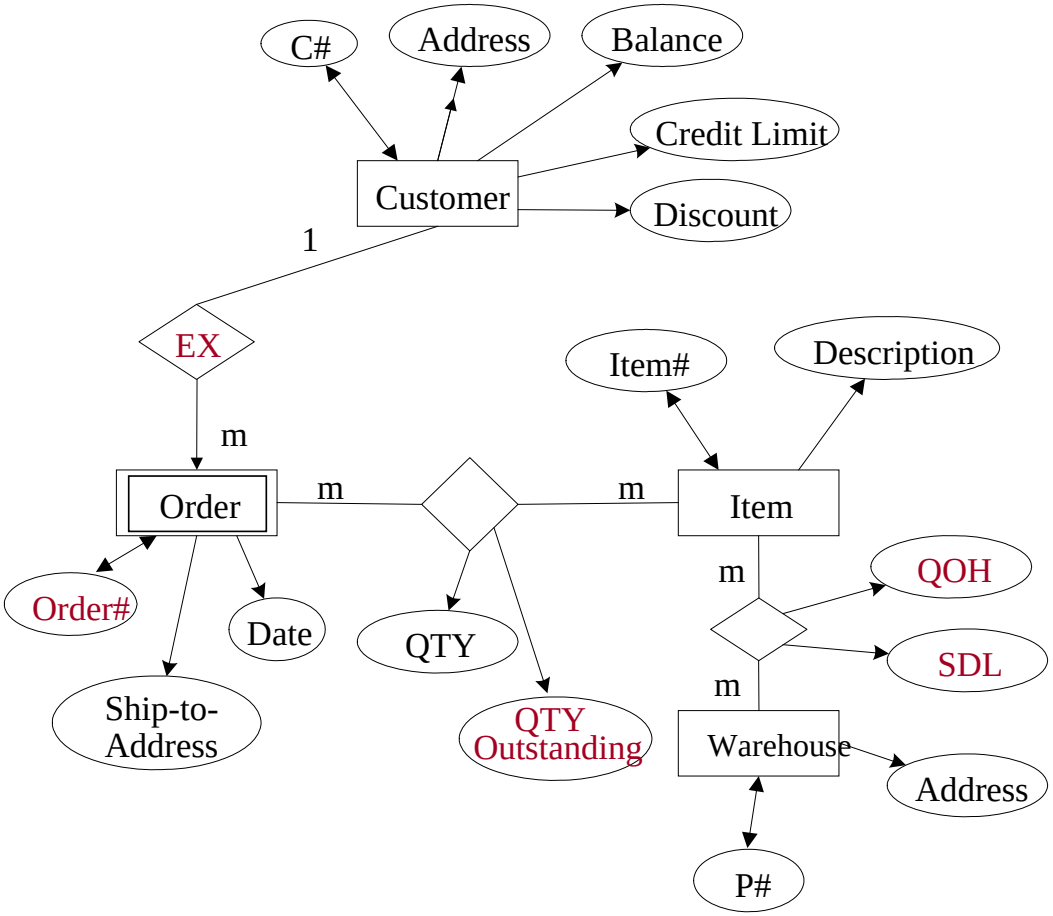
Project (P#, Pname, Budget)

Relationship relation:

Emp_Proj(E#, P#, Progress)

Recall **Example 1 -** An Order-Entry System.

Q: What are the relations translated from the entity types and the relationship types of the ER diagram?



A Normal Form for Entity Relationship Diagrams

Ref: Tok Wang Ling (4th ER conference, 1985)

Objectives for defining a normal form for ERDs.

1. to capture and preserve all the semantics of the real world of a database which can be expressed in term of functional, multivalued, and join dependencies, by representing them **explicitly** in the ERD,
2. to ensure that all the relationship types represented in the ERD are **non-redundant,**
3. to ensure that all the relations translated from the ERD are in good normal form: either in **3NF** or **5NF**.

A **normal form ERD** may consist of

- composite attributes
- multivalued attributes
- weak entity types
- special relationships such as:
  existence dependent (EX), identifier dependent (ID),
  ISA, UNION, INTERSECT, DECOMPOSE relationships.

Outline

1. Define the set of basic dependencies of an entity type
2. Define the entity normal form (E-NF)
3. Define the set of basic dependencies of a relationship type
4. Define the relationship normal form (R-NF)
5. Define the set of basic dependencies of an ERD
6. Define the normal form ERD (ER-NF)
7. Convert an ERD to a normal form ERD

Defn 1 The **set of basic dependencies** **of an** **entity type E** with identifier **K**, denoted by **BD(E)**, is defined as:

(1) For each m : 1 attribute A of E
$K \rightarrow A \in BD(E)$

(2) For each 1 : m multivalued attribute A of E
$A \rightarrow K \in BD(E)$

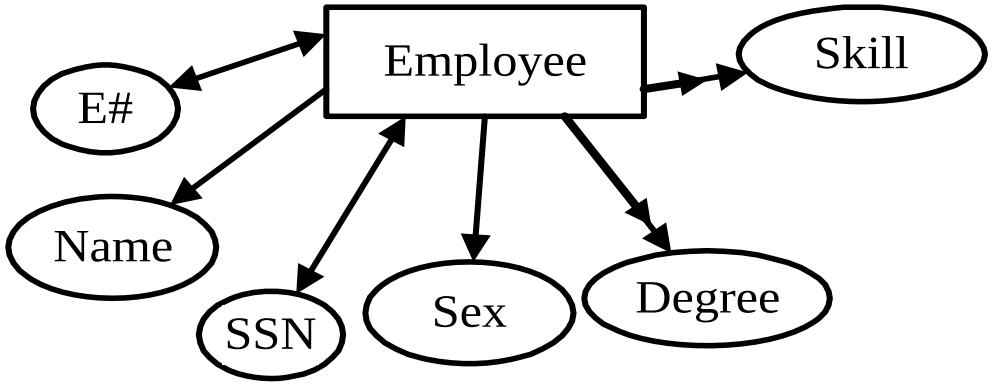
(3) For each 1 : m and m : m multivalued attribute A of E
$K \longrightarrow\!\!\!\!\!\rightarrow A \in BD(E)$

(4) For each key K_1 of E, $K_1 \neq K$
$K \rightarrow K_1 \in BD(E)$
$K_1 \rightarrow K \in BD(E)$

(5) No other FDs or MVDs are in BD(E).

Defn 2 An entity type E is in **entity normal form** (**E-NF**) if and only if all given FDs and MVDs which only involve attributes of E, can be derived from BD(E).

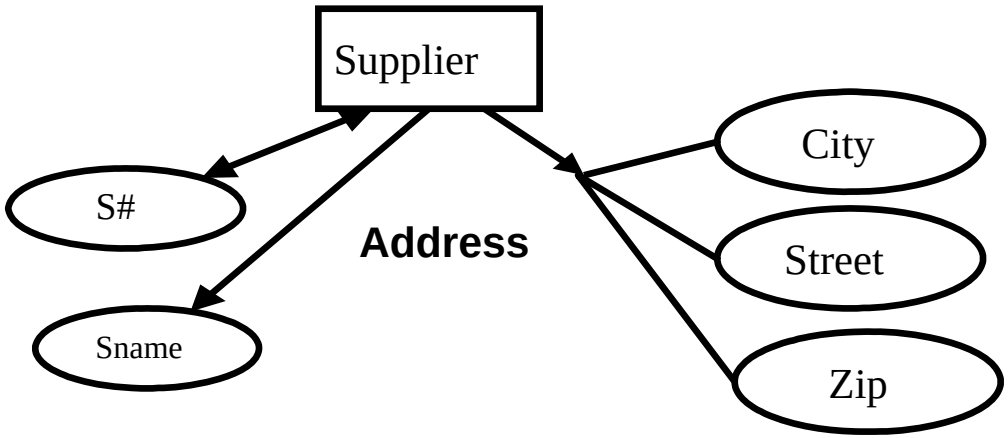
E.g. 1. (E# is the identifier)



BD(Employee) = { E# $\rightarrow$ SSN, Name, Sex
SSN $\rightarrow$ E#
E# $\twoheadrightarrow$ Skill
E# $\twoheadrightarrow$ Degree }

Employee is in E-NF.

E.g. 2. (Address is a m : 1 composite attribute)



BD(Supplier) = { S# $\rightarrow$ Sname, S# $\rightarrow$ {City, Street, Zip}}

- Two well-known FDs are not in BD(Supplier), i.e.,
  City, Street $\rightarrow$ Zip $\notin$ BD(Supplier)$^+$,
  Zip $\rightarrow$ City $\notin$ BD(Supplier)$^+$
- Supplier is not in E-NF

Defn 3 Let R be a relationship type with identifier **K**, and $\mathcal{F}$ be the associated set of FDs which only involve the identifiers of the set of entity types participating in R. The **set of basic dependencies of R, BD(R)** is defined as:

(1) For each **1 : 1** attribute A of R

$K \rightarrow A \in BD(R)$

$A \rightarrow K \in BD(R)$

(2) For each **m : 1** single valued attribute A of R

$K \rightarrow A \in BD(R)$

(3) For each **1 : m** multivalued attribute A of R

$A \rightarrow K \in BD(R)$

(4) For each **1 : m** or **m : m** multivalued attribute A of R

$K \twoheadrightarrow A \in BD(R)$

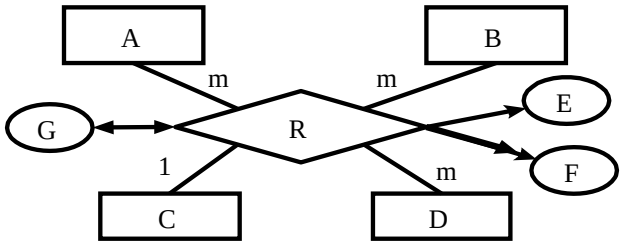
(5) Let $A \rightarrow B$ be a full dependency in $\mathcal{F}$ such that A is a set of identifiers of some entity types participating in R, and B is the identifier of some entity type participating in R. If A is a key of R or B is part of key of R, then

$A \rightarrow B \in BD(R)$

(6) No other FDs or MVDs are in BD(R).

Defn 4 A relationship type R of an ER diagram is said to be in **relationship normal form** (**R-NF**) iff all given FDs and MVDs which only involve attributes of R and identifiers of entity types participating in R are implied by BD(R).

E.g. 3



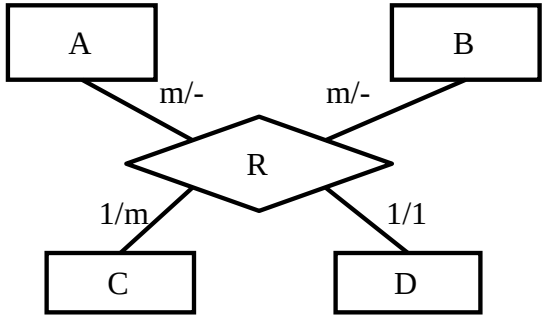
ABD is the identifier of R.

$BD(R) = \{ABD \rightarrow EG,\ G \rightarrow ABD,\ ABD \twoheadrightarrow F,\ ABD \rightarrow C\}$

R is in R-NF.

Note: Suppose we have $C \rightarrow D \in \mathcal{F}$, then $C \rightarrow D$ is also in BD(R) by definition of BD(R) and so R is still in R-NF.

E.g. 4



$BD(R) = \{AB \rightarrow CD\}$

$C \rightarrow D \notin BD(R)^+$

So, R is not in R-NF.

Defn 5 The **set of basic dependencies of an ERD D**, denoted by **BD(D)**, is defined as the **union** of the sets of basic dependencies of all the entity types of D and the sets of basic dependencies of all relationship types of D.

Defn 6 An ERD D is in **normal form (ER-NF)** if it satisfies the following conditions:

(1) All attribute names are of different semantics, and all identifiers of entity types are unique.

(This is to conform to the Relaxed Universal Relation Assumption which only requires the identifiers of all entity types be unique. We don't really need to conform to the Universal Relation Assumption)

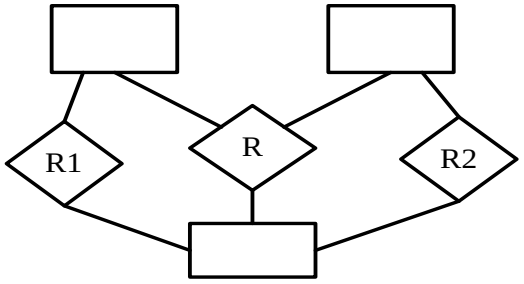
(2) Every entity type in the ERD is in E-NF.

(3) Every relationship type in the ERD is in R-NF.

(4) All given relationships and dependencies are implied by BD(D).

(5) **No** relationship type (including its attributes) can be derived from other relationship types by using join and projection.

E.g.



Note: If $R = R1 \bowtie R2$

then R can be derived from R1 and R2, and we don't need to store its contents physically. The ERD is not in ER-NF.

Informally speaking,

(1) Cond'n (1) is required in order to conform to the Relaxed Universal Relation Assumption.

(2) Cond'n (2) ensures that all relations generated for all entity types are in 5NF.

(3) Cond'n (3) ensures that all relations generated for all regular relationship types are either in 3NF or in 5NF and there is no relation in BCNF but not in 4NF or 5NF.

(4) Cond'n (4) ensures that ERD has captured all relationship types and dependencies of the given database.

(5) Cond'n (5) ensures that no relationship type (together with its attributes) can be derived from other relationship types using join and projection operation.

Theorem:

The relations generated for each of entity type of a NF-ER diagram are in 5NF.

The relations generated for each of the relationship type of a NF-ER diagram are either in 3NF or 5NF.

Convert an ERD to a normal form ERD

Step 1: Ensure that all identifiers of entity types are unique and all attributes are of different semantics (to conform to the **Relaxed Universal Relation Assumption**).

Step 2: Convert any non E-NF entity type to E-NF. (remove all undesirable FDs and/or MVDs by introducing new entity types and relationship types)

Step 3: Convert any non R-NF relationship type to R-NF. (remove all undesirable FDs, MVDs and/or JDs by introducing new entity types and relationship types or by splitting the relationship type into smaller ones)

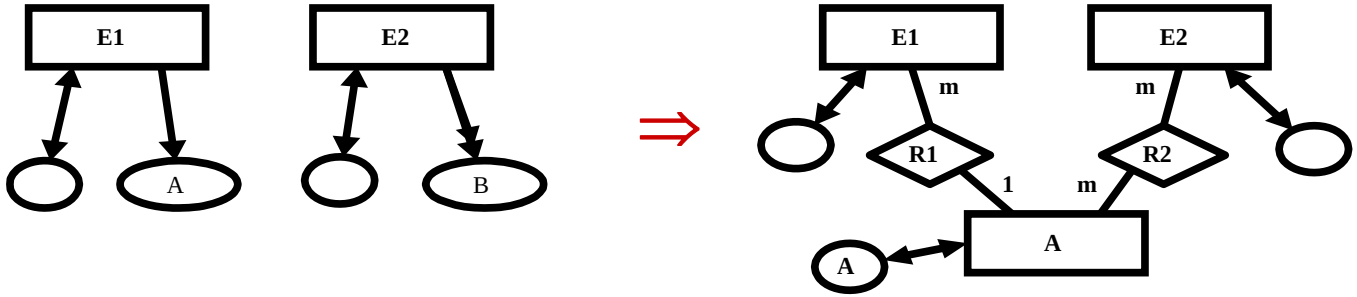
Step 4: Remove relationship types which have no associated attributes and is equal to the join and/or projection of other relationship types (as they can be derived).

Details of Step 2

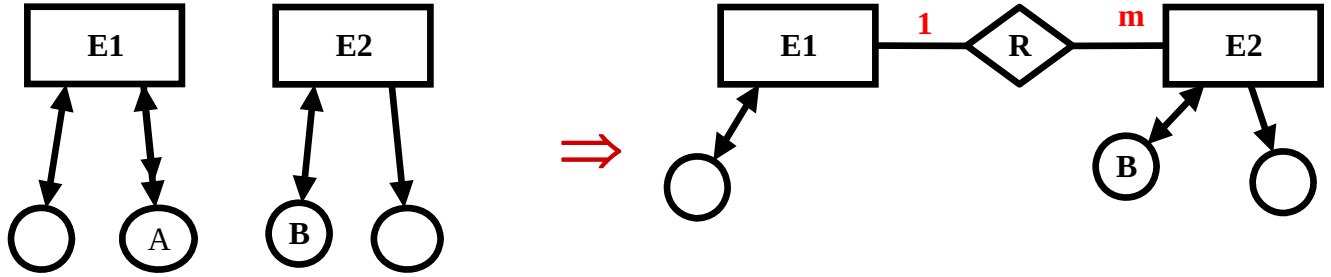
(1) Each single valued attribute A is fully dependent on each key of E which does not contain A.
(2) All components of any composite single valued attribute A are fully dependent on each key of E which does not contain A.
(3) There is no non-trivial FDs defined among components of any composite attribute of E.
(4) No multivalued attribute of E is multi-dependent on a part of a key of E.
(5) No component of a composite many-to-many attribute of E is multi-dependent on the identifier of E.
(6) No component of a composite many-to-one attribute determines the identifier of E.
(7) Condition 1 of Lemma 4.2 of the NF-ER paper.

Step 1

E.g. A and B are of the same semantics



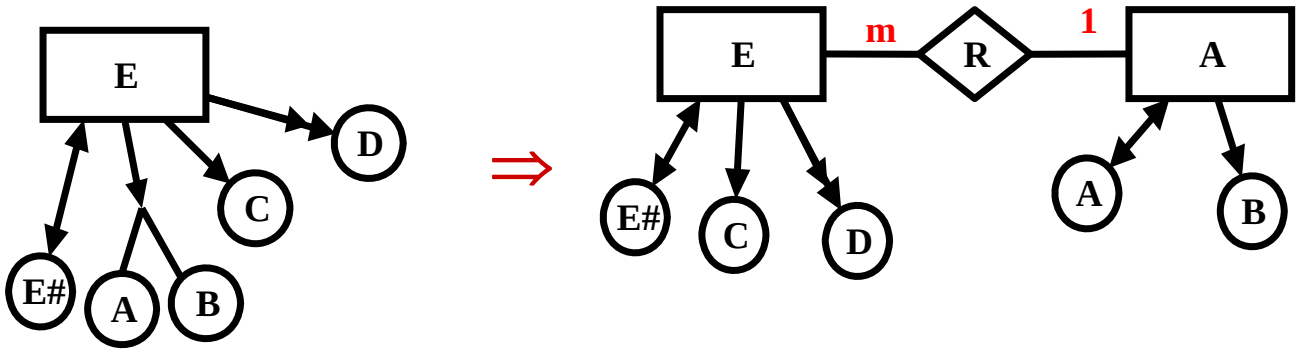
E.g. A and B are of the same semantics



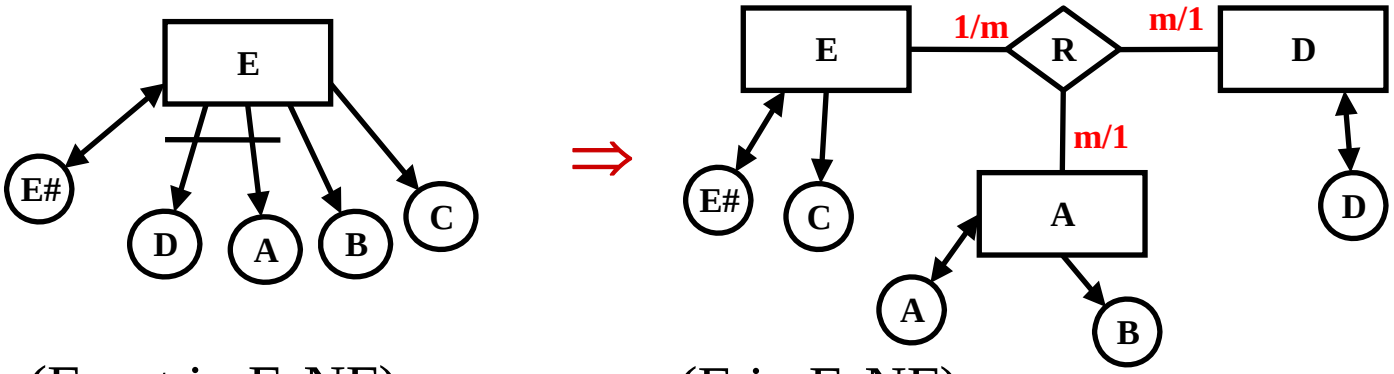
Step 2

E.g. A → B in a composite attribute of E

(E not in E-NF) (E in E-NF)

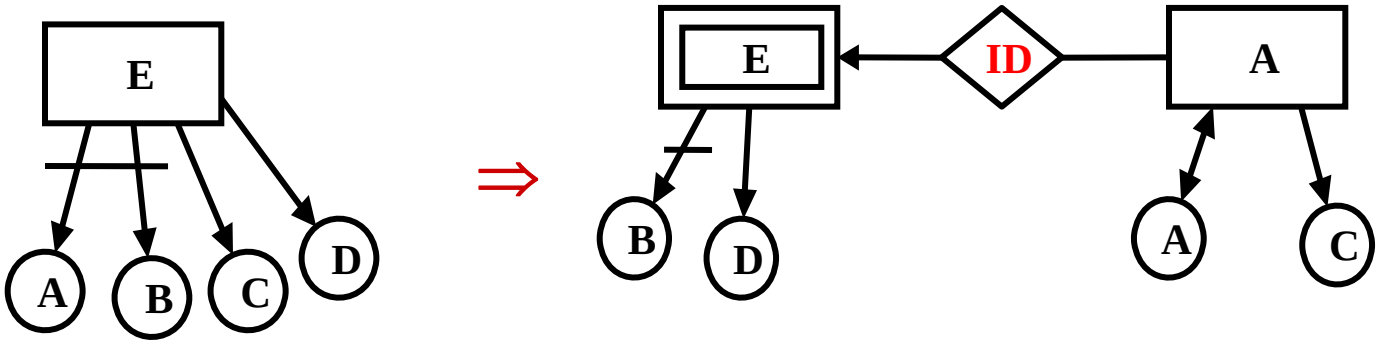


E.g. A → B in E and A is part of a key of E



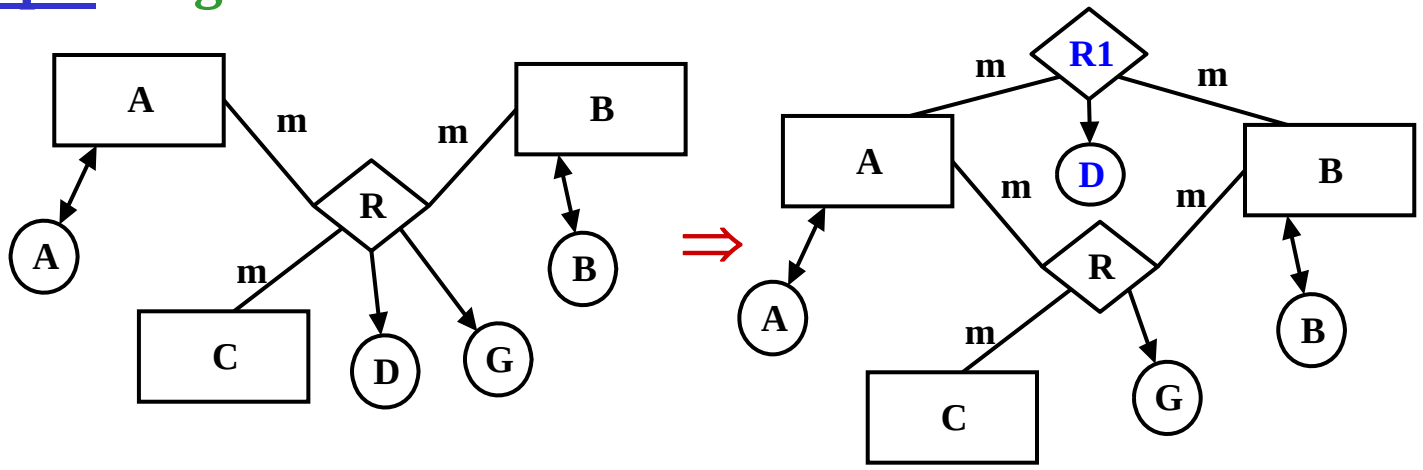
(E not in E-NF) (E in E-NF)

E.g. A → C and A is part of the identifier of E, E becomes a weak entity type and the identifier is still AB.



(E not in E-NF) (E in E-NF)

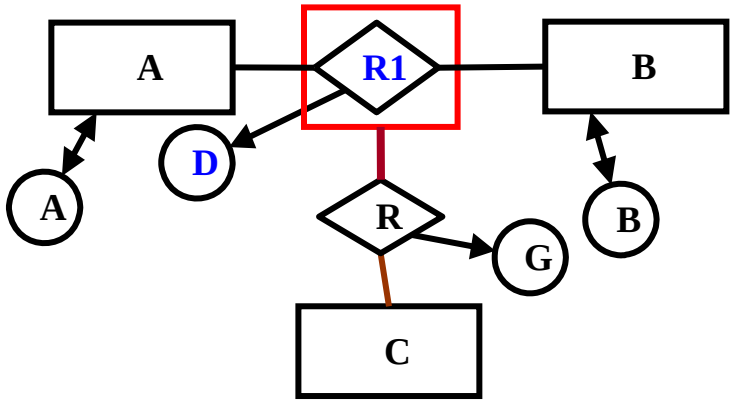
Step 3 **E.g.** AB → D



(R not in R-NF)

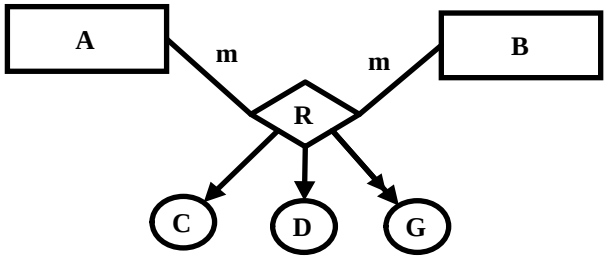
(R1 and R in R-NF)

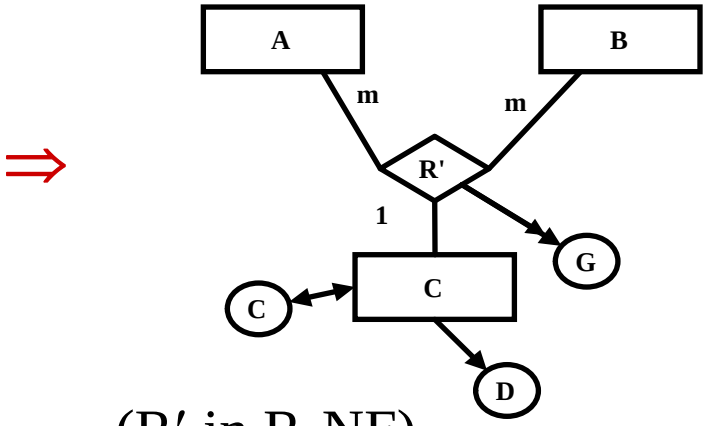
or ⇒ (Use aggregation, better solution)



(R1 and R in R-NF)

E.g. C → D

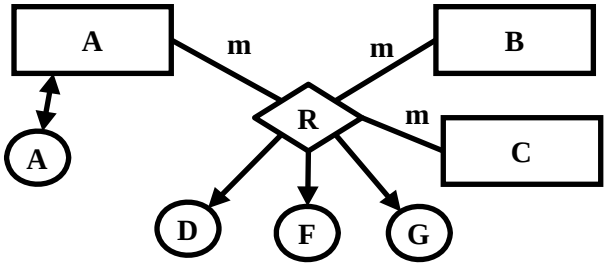


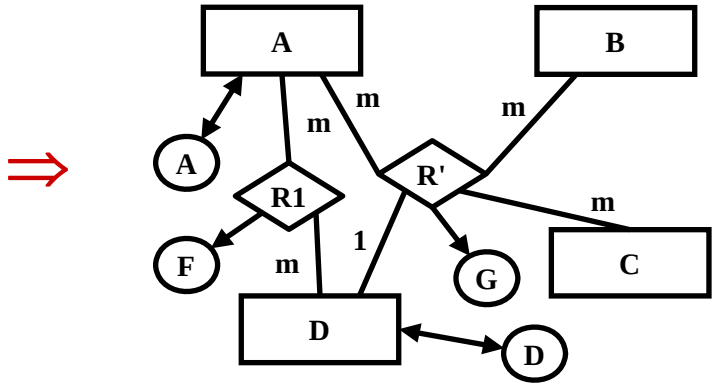


(R not R-NF)

(R′ in R-NF)

E.g. AD → F in R





(R not in R-NF)

(R1 and R′ in R-NF)

Some Database Design Tools

Some free and commerce Database Modeling tools can be found at

http://www.databaseanswers.org/modelling_tools.htm

Summary

- How to choose the identifier for an entity type?
- Are the concepts of identifier of entity type and primary key of relation of the relational model the same? If not, what are the main differences between them?
- When and why do we need to define identifiers for relationship types?
- What are the differences between cardinality and participation constraint? Which one is better (more powerful)?
- What is the main difference between EX and ID dependency relationships?
- Notation for aggregation in ERD

- Represent more than one FD in a relationship type
- Generalization vs. specialization
- English Sentence Structure and ER Diagram
- In general when should a concept/thing be designed as an attribute or as an entity type? E.g. phone?
- Translation of a (normal form) ER diagram to a relational database
- Universal relation assumption and relaxed universal relation assumption
- What are the advantages of using ER Approach for database design as compared to Decomposition and Synthesizing methods?