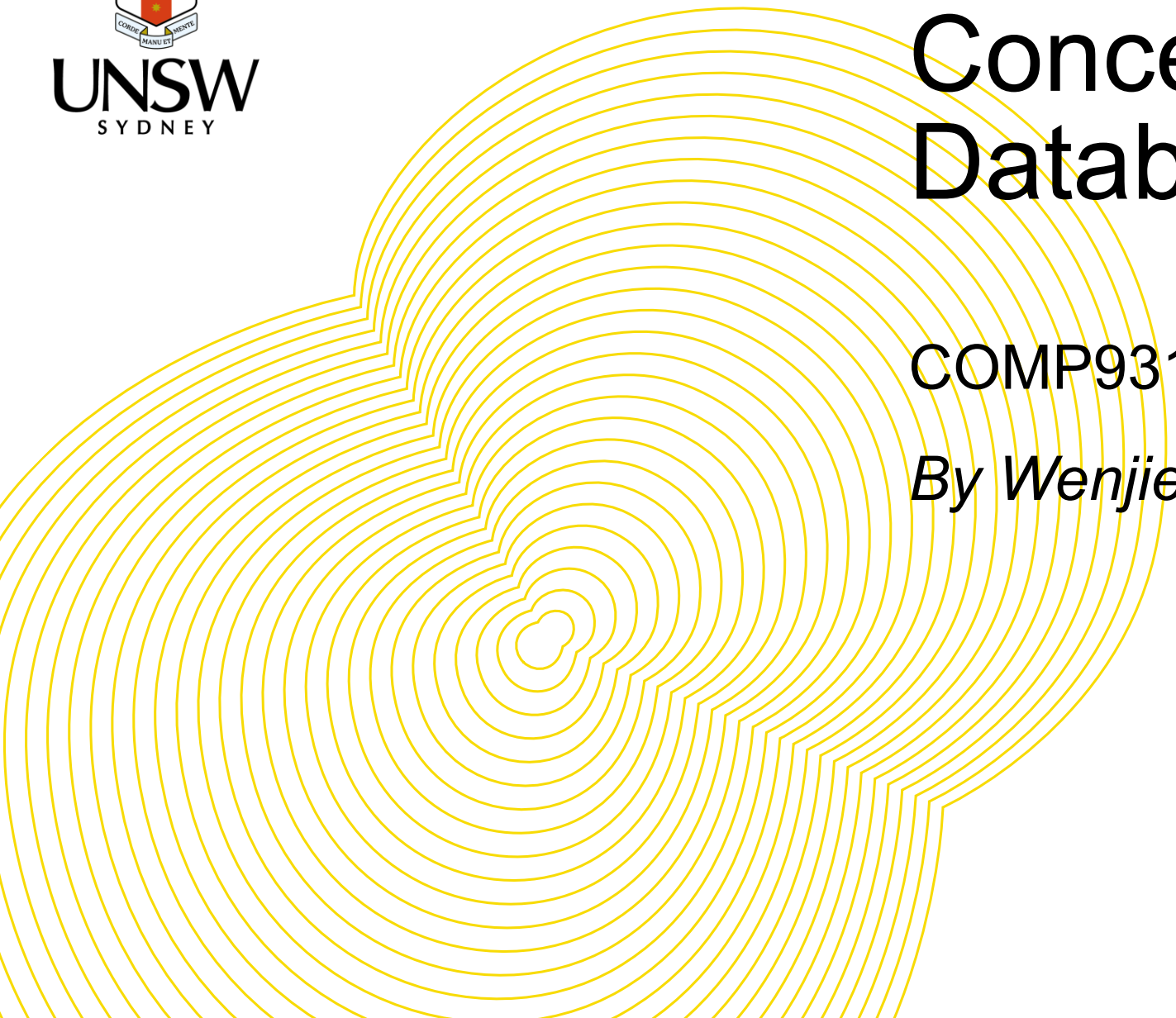


Conceptual Database Design

COMP9311 24T3; Week 1.2

By Wenjie Zhang, UNSW

Notice

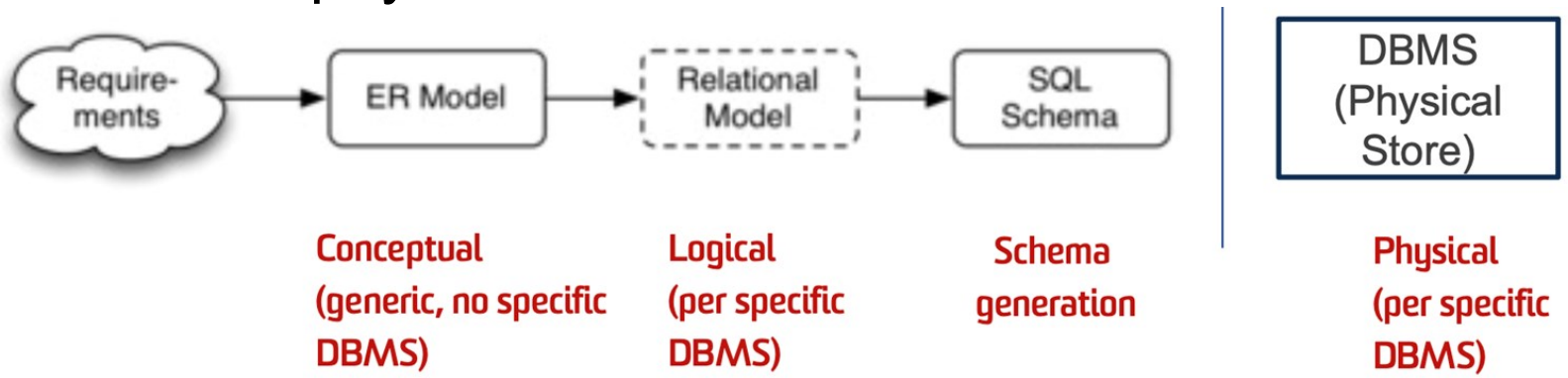
- No Lab in Week 1, first lab will be in Week 2
- Assignment 1 is released on WebCMS
- Submit your answer via Moodle
- Deadline: 5pm Monday, Week 4 (30 September)

Data Modelling

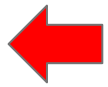
Kinds of data models:

- conceptual: abstract, high-level data model, e.g., ER, ODL (object data language) – user friendly
- logical: concrete, for implementation in specific DBMS, e.g., relational
- physical: internal file storage (inside a specific DBMS)

Strategy: design using abstract model; map to logical model, DBMS takes care of the physical model



Recall the Following Stages

- **Conceptual Design** 
- **Logical Design**
- **Physical Design**

Scenario

A: “Welcome to the job, can you build us a database to organize all the information on a publisher and present it at the meeting?”

B: “okay, what’ll be in the database?”

A: “Well… authors, books, editors, printers etc. An author can write zero or more books and a book is written by one or more authors. Where a book is uniquely identified by its book-id. For each book, we also record its title, price, and availability. An editor is uniquely identified by his/her reader-id and we also record his/her name, phone number, address. The address is composed of street and suburb…”

The Application

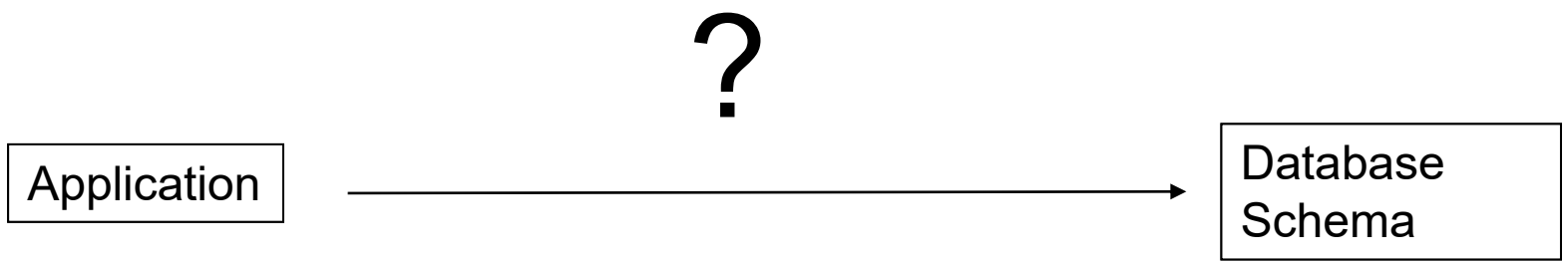
“An author can write zero or more books and a book is written by one or more authors. Where a book is uniquely identified by its book-id. For each book, we also record its title, price, and availability. An editor is uniquely identified by his/her reader-id and we also record his/her name, phone number, address. The address is composed of street and suburb…”

This is the application we need to design the design for; the application has a set of requirements.

Requirements Collection & Analysis

Simplified Database Workflow

What should we do?

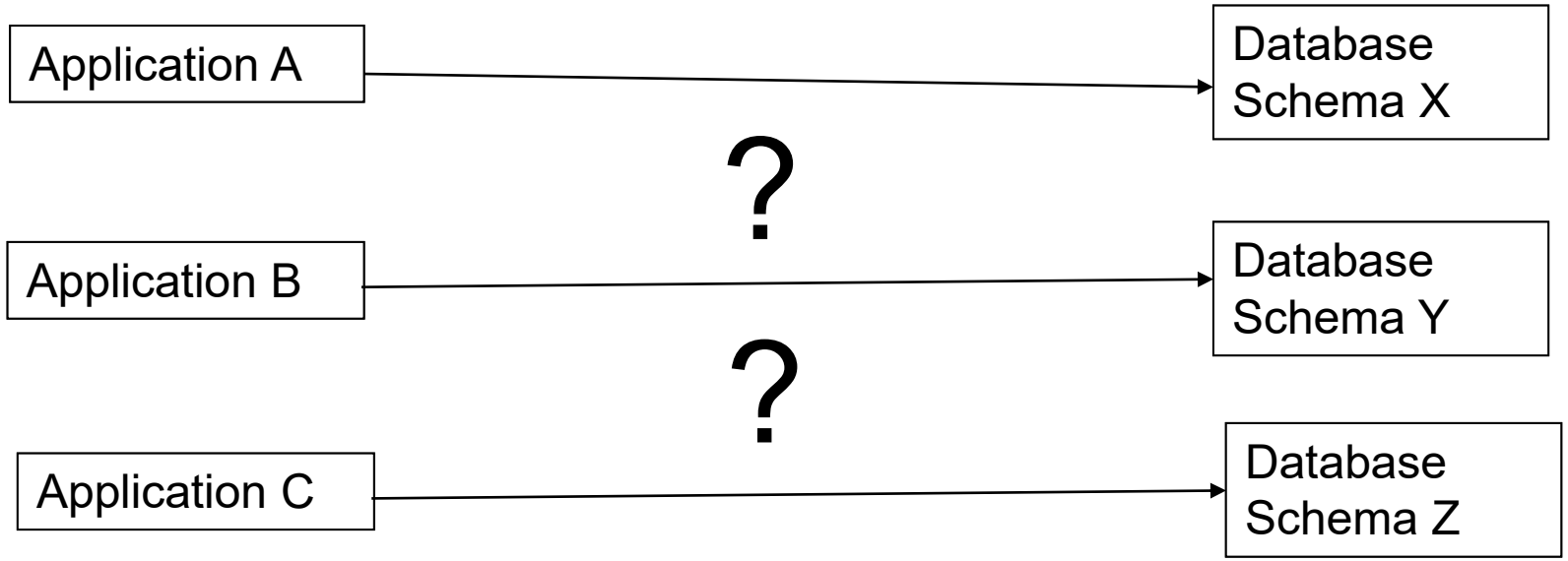


Application: has many requirements

Database Schema: provides a logical view of data model

Simplified Database Workflow

More Generally, how should everyone build his/her databases for an application for its given requirements?



Database Schema: provides a logical view of data model

Solution

Dr. Peter Chen

https://www.csc.lsu.edu/~chen/

His work (in 1976) started a new field of research and practice:

Conceptual Modeling

See paper on the Entity-Relationship Model in link

https://bit.csc.lsu.edu/~chen/pdf/english.pdf

English Sentence Structure and Entity-Relationship Diagrams

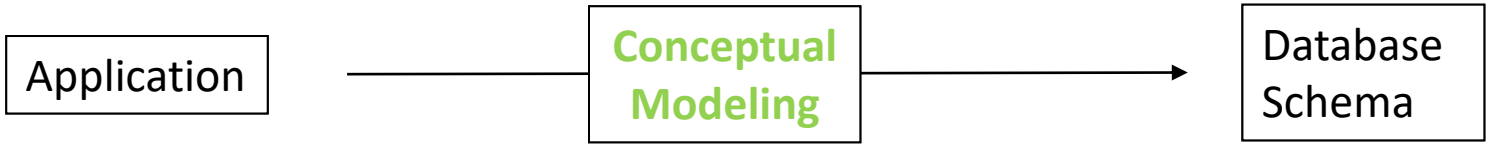
PETER PIN-SHAN CHEN*

Graduate School of Management, University of California, Los Angeles, California 90024

Solution

(Chen. 1976)

- Entities
- Relationships
- Attributes



Entity-Relationship Model

Chen's ER model has two major components:

- ***Entity***: collection of attributes describing object of interest
- ***Relationship***: association between entities (objects)

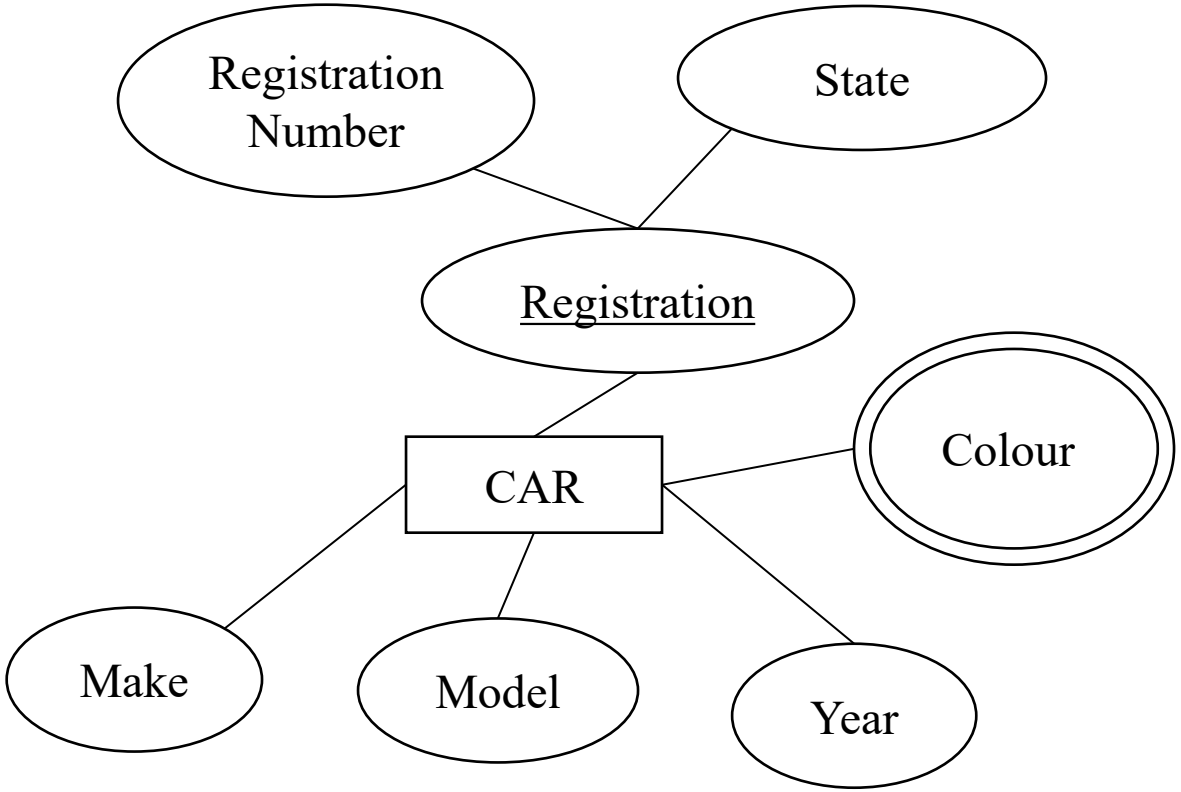
There's also the third components unofficially

- ***Attribute***: data item describing a property of interest

Note: The ER model is not a standard, so many variations exist.
Lecture notes use simple notations.

ER Diagrams

Entities and their attributes can also be described with Entity-Relationship Diagrams (ERDs). e.g.



Entity

“**Entities** represent things in the real world, and attributes describe properties of entities.”

Entities

Some examples of entities (in a company)

Simple Attributes

Simple Attributes (or **Atomic** Attributes) are attributes that are not divisible.

Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

For example:

- Entity = Student
- Attributes = Student number, name…

Student A:

- Student number = z12345678
- Name = Bob

Multi-Valued Attributes

Attributes can also be **multi-valued**

- Multivalued: has more than one value
- No longer a simple attribute with only one value.

Questions:

- Each simple attribute is meant to hold one value, sure it can change… but one value at a time.
  - What if… I'm asking you to model shirts with two colors?
  - How can we model this without multivalued attributes?

Multi-Valued Attributes

Attributes can also be **multi-valued**

- Multivalued: has more than one value
- No longer a simple attribute with only one value.

Questions:

- Each simple attribute is meant to hold one value, sure it can change… but one value at a time.
  - What if… I'm asking you to model shirts with two colors?
  - How can we model this without multivalued attributes?

Answer: To describe the entity faithfully, we can use multi-valued attributes to model the colors. Multi-valued attributes are necessary, otherwise, it is very hard to express such values.

Multi-Valued Attributes

The attributes you design should be able to describe, and the combinations of all its instances should be able to express the entity **faithfully**

Example: more expressively model the attribute color for entity shirt.

Composite Attributes

Recall what is a simple attribute? Attributes that are not divisible are called *simple* or *atomic attributes*.

Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

Some semantics cannot be captured using atomic attributes

Question:

Question: is *Address* a simple attribute/value?

- *Address = 'Computer Science Building (K17), Engineering Rd, UNSW Sydney, Kensington NSW 2052"*

How can should we model Addresses ?

Question:

Question: is *Address* a simple attribute or a composite attribute?

- *Address = 'Computer Science Building (K17), Engineering Rd, UNSW Sydney, Kensington NSW 2052"*

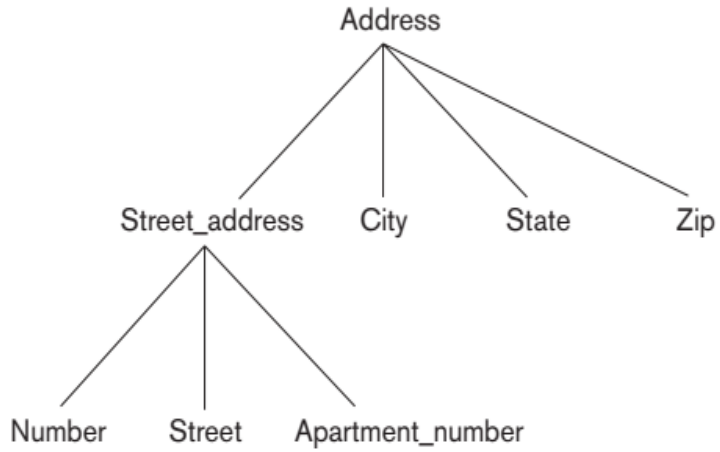


Figure 7.4
A hierarchy of composite attributes.

Question:

Composite attributes are useful for situations when

- The end-user sometimes refers to the composite attribute as a unit,
- But at other times refers specifically to its components.

Question: Can’t I just let my composite attributes, be simple attributes instead?

Question:

Composite attributes are useful for situations when

- The end-user sometimes refers to the composite attribute as a unit,
- But at other times refers specifically to its components.

Question: Can't I just let my composite attributes, be simple attributes instead?

Answer: If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes.

Derived Attributes

Attributes that are problematic if modeled with a simple value.

Scenario: modelling a person's age:

Why? Your values can change and are dynamic.

The age attribute is called a **derived attribute**, because age is said to be **derivable from** the Date-of-birth attribute, which is called a **stored attribute**.

Derived Attributes

Where there is a derived attribute, there must also be an attribute where it's values can be derived from.

Derived attribute values are not stored, the stored attribute that derives the value is stored.

Question: Why include it at all if it's derived? Why not leave it out completely in the data model?

Derived Attributes

Where there is a derived attribute, there must also be an attribute where it's values can be derived from.

Derived attribute values are not stored, the stored attribute that derives the value is stored.

Question: Why include it at all if it's derived? Why not leave it out completely in the data model?

Answer: 1) The derived attribute could have a **business meaning**, including it in the model helps to present a complete picture of the entity. 2) If many queries request the derived attribute, including it helps improve the **query performance**. (Will be discussed in later lectures)

Entity Type

An **entity type** defines a collection of entities that have the same attributes.

1. An entity type describes the **schema** or **intension** for a *set of entities* that share the same structure.
2. An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name.
3. The collection of entities of a particular entity type is grouped into an **entity set**, which is also called the **extension** of the entity type.

Entity Type Example

CAR

Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1

((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2

((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3

((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

Take away

- Given an entity, the attributes you design should be able to describe, and the combinations of all its instances should be able to express the entity faithfully.
- This is why attribute types such as multi-valued attributes, composite attributes exist.
- Can I have an entity to describe something abstract? Of course, entities don’t have to be concrete objects. You will still need to find the right attributes to describe it.
- Relations can also have attributes

Entity Instances/Facts

A good entity schema should have good attributes that can be filled with good values.

Car Entity Schema:

- ((Reg. Num, State), Make, Model, Year, {Color})

Car instance:

- (CS9311, NSW), Toyota, Toyota Corolla, 1999, {White, Silver})

NULL Value

What if an instance doesn't have a value?

Examples:

- Height of a person is not known (True value is not yet known)
- A person may not have a college degree (No suitable value)

Exercise: List more cases where use of a NULL value would be appropriate.

NULL values represent attributes whose values are **unknown** or **do not exist** for some entity instance. In general, it is a special value to indicate a lack of value.

Keys

Key constraint in ER Modelling: in any extension of the entity type, no two entities have the same values for their key attribute or key attributes.

For example:

- {payroll number} is a key of EMPLOYEE,
- {car registration number} is a key of CAR.

All elements of a set are distinct; an entity can be defined as the set of its entity instances.

Therefore, the entity instances in a relation must also be distinct. How do we identify them? There must be a key for all entities.

Key

A ***super key*** is a set of one or more attributes that uniquely identify any entity instance of an entity type.

- natural (e.g., name + address + birthday)

Q: What if none of my attributes can uniquely identify my entity?

A: you can make an attribute *artificially* (e.g., SSN, Passport Number, etc.)

Keys containing multiple attributes as **composite keys.**

Key

Q: Technically, can’t I make an entity key to be all its attributes?

A: Since the entity is a **set**, in the worst case where there is no natural composite key, the set of all it’s attributes together should uniquely identify its values (strictly speaking).

Every entity must have a final primary key: a minimal set of attributes that can uniquely identity its entity instances.

Key Summary

Key/Super key: a set of one or more attributes that can uniquely identify an entity instance of an entity type.

Candidate key: minimal superkey (no subset is a key)

Primary key : a candidate key chosen by DB designer

Keys are indicated in ER diagrams by *underlining*.

Weak Entity

Some entity types do not have a key of their own.

Such entity types are called **weak entity types.**

Entities of a weak entity type can be identified by a partial key and by being related to another entity type - *owner.*

The relationship type between a weak entity type to its owner is the *identifying relationship* of the weak entity type.

Weak Entity

A weak entity doesn’t have any primary key but does have a partial key (discriminator).

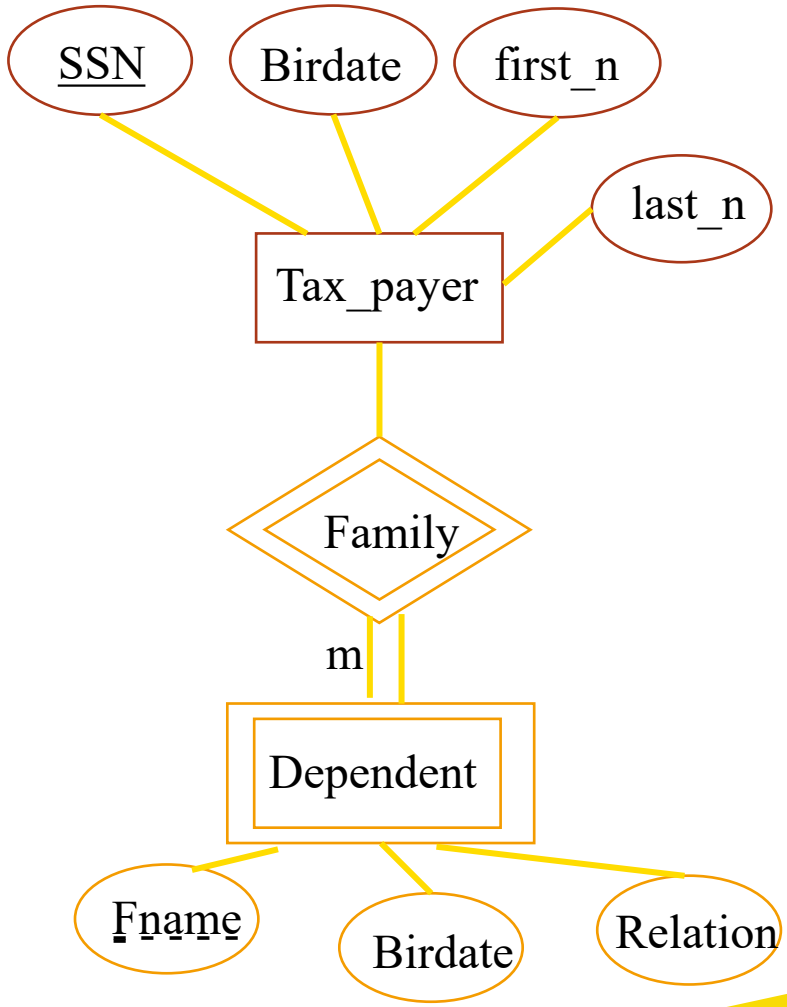
Partial key/discriminator: a key that is *partially unique*. i.e., only a subset of the attributes can be identified using it

A ***strong entity*** is an entity a regular entity where there is a primary key that uniquely identifies all instances.

Identifying relationship: the relationship type between a weak entity type to the owner of the weak entity type.

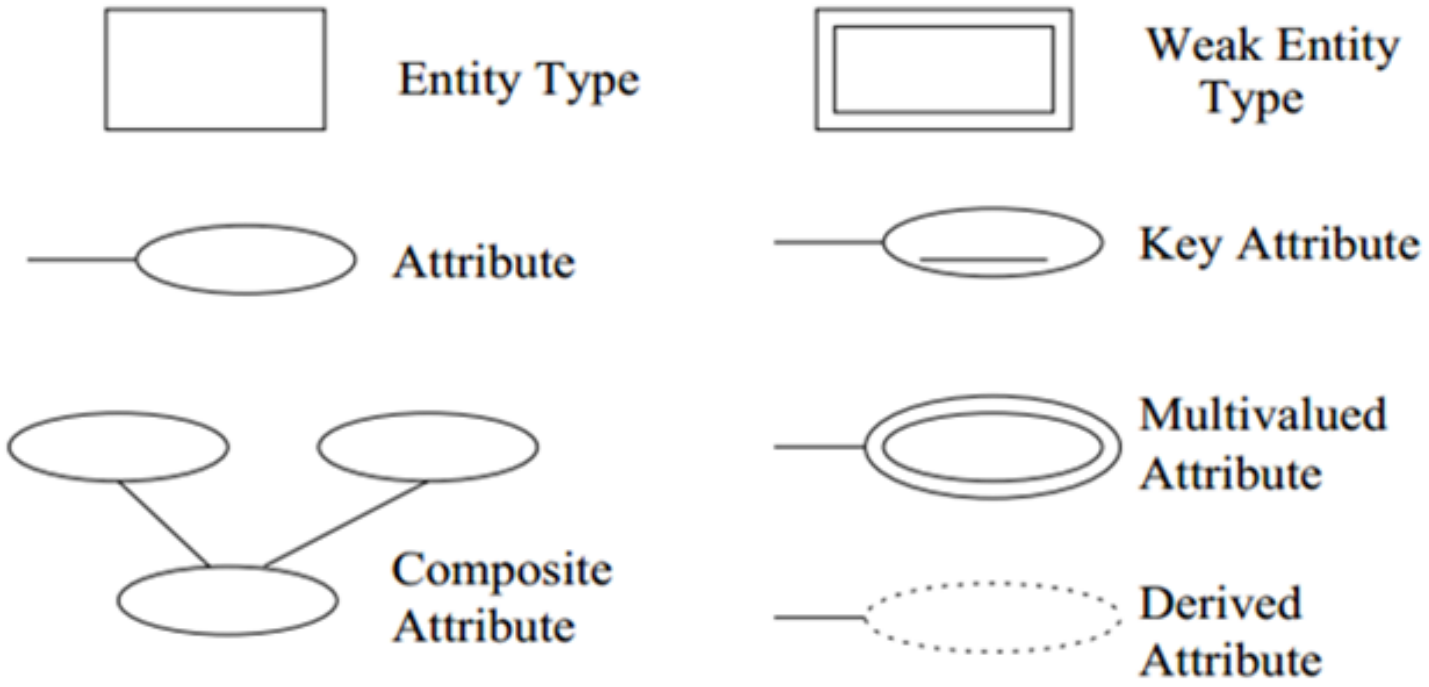
Example

1. A TAX PAYER entity may be related to several DEPENDENTS, identified by their names.
2. DEPENDENT is called a weak entity, {Name} is a partial key for it.
3. The identifying relationship between DEPENDENT and TAX PAYER is IS-DEPENDENT-OF.
4. TAX PAYER is said to own DEPENDENT



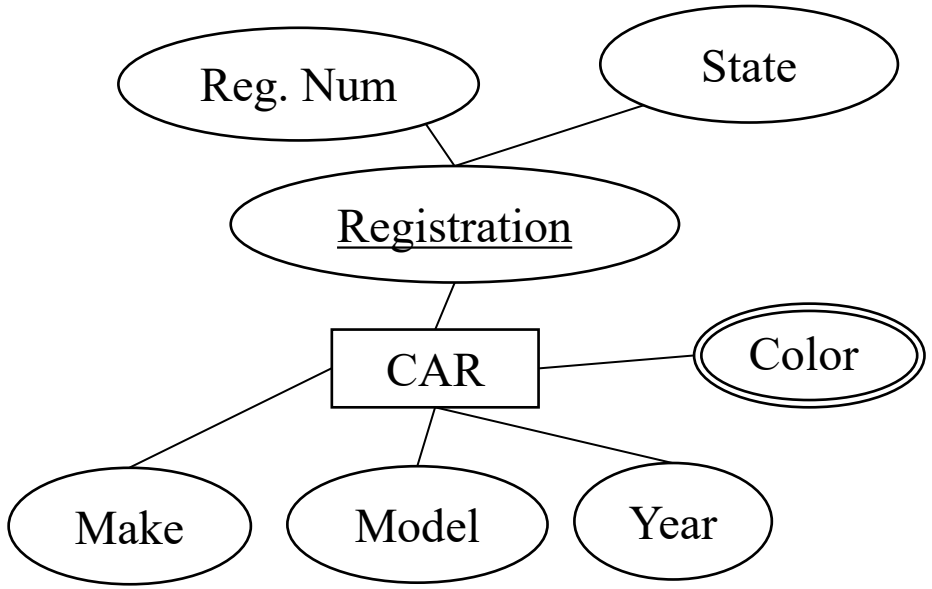
Visualizing an ER Data Model

Notations:



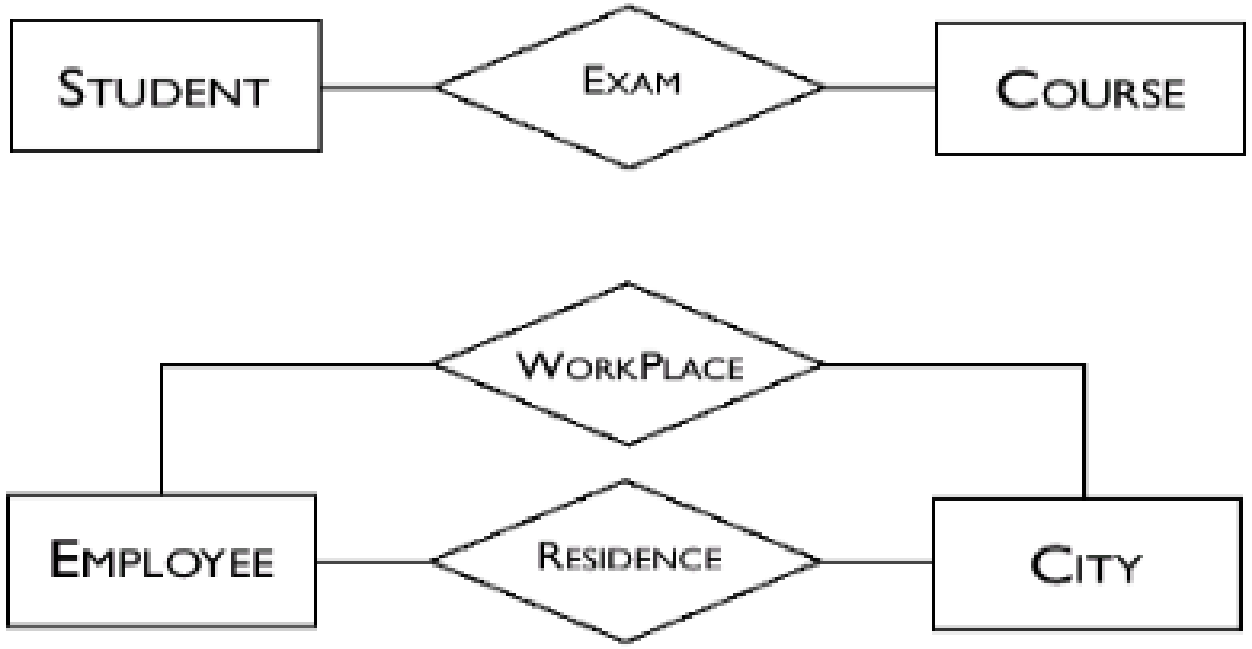
Visualizing an ER Data Model

The data model can be effectively described using the Entity-Relationship Diagrams (ERDs).



Relationship

Second Big Component of ER: They represent logical links between two or more entities.

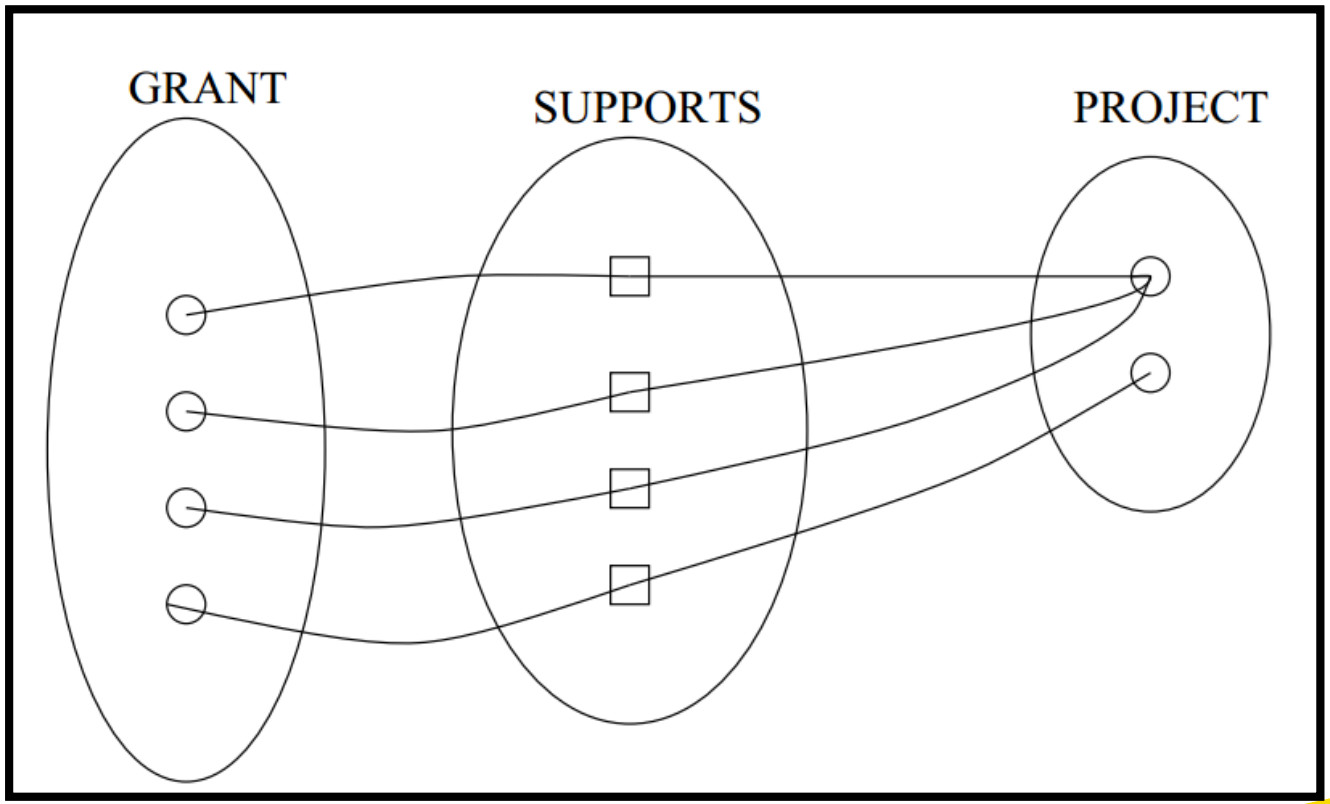


Relationships

- Relationships relates one entity type to another entity type.
- An entity type can be related to more than one other entities, and will have a different role to play for each relationship (name each relationship to distinguish them).
- An entity can also have different relationships with another entity.
- Relationships can be illustrated using *occurrence diagrams*

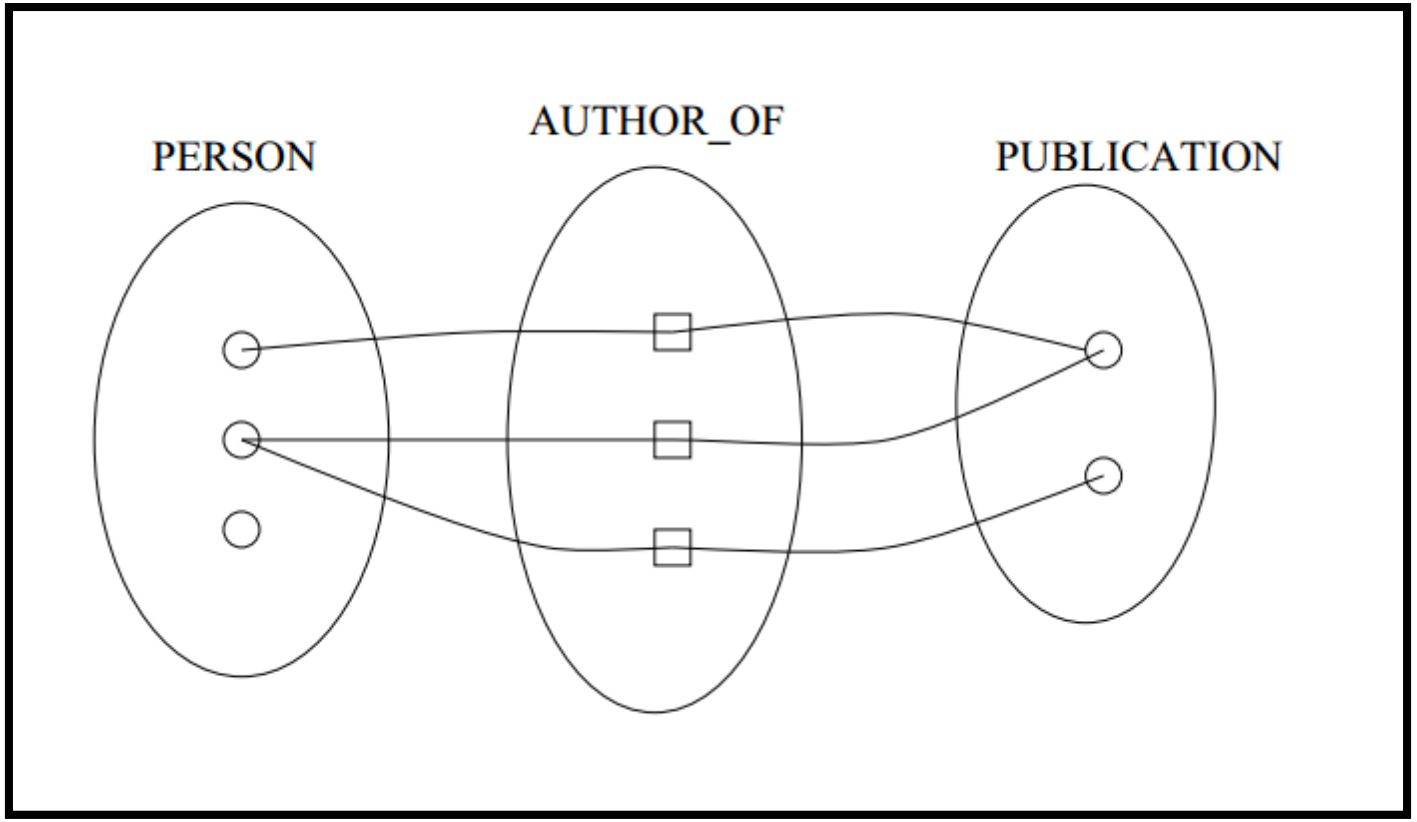
Practice

Supports is a relationship. It's instances describe the how grants are given to projects.



Practice

Author-of is a relationship. It's instances describe who wrote what publication.



Modelling relationships

Relationship types usually have certain properties that limit the possible combinations of entities participating in relationship instances.

By default, entities can be related to other entities freely.

Are all relationships like that?

1. Relationship between employee and club? (An employee can be a member of many clubs, and a club can have many employee members)
2. Relationship between manager and company? (A company can have several managers, but a manager must belong to one company)

Modelling relationships

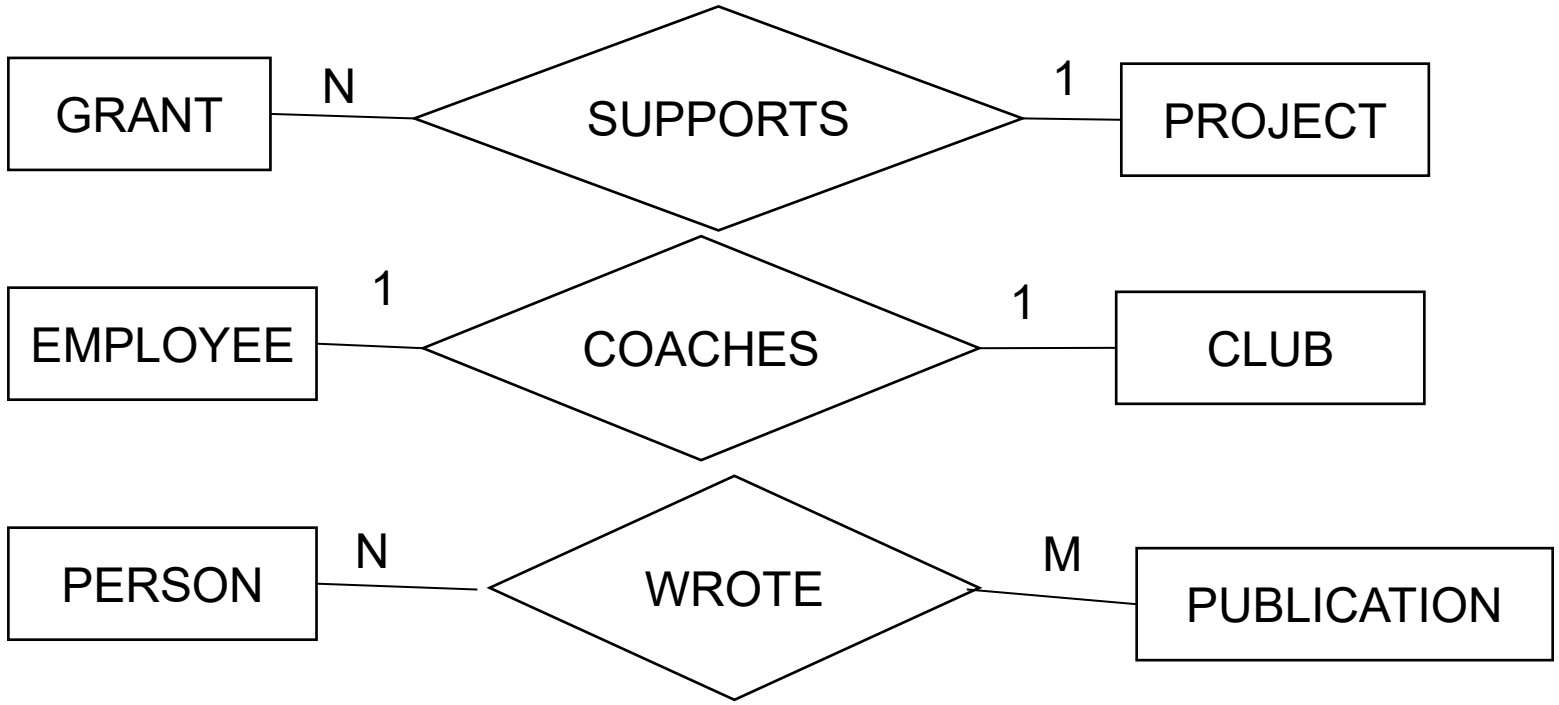
Cardinality ratio: the number of relationship instances an entity can participate in.

There are three types of cardinality in relationship: is **M:N (many to many), 1:N (one to many)** is stricter, and **1:1 (one to one)** is the strictest.

Example: A research grant supports only one research project, but a research project may be supported by many grants. PROJECT:GRANT is a 1 : N relationship.

Constraints on relationship types(cont)

We can also show this in an ERD:



Participation Constraint on Relationships

Question: Will each instance participate in this relationship?

Answer: It depends on the entity type. For example, each patent must be written by a person, but a person may not have a patent.

For example, all university students must be enrolled to university/universities.

So far, entities don’t have to participate in the relationships they are in.

Participation Constraint on Relationships

- **Total Participation:** each entity instance must patriciate in at least one relationship instance.
  - Example: We want this relationship to express all publications must be written by a person.

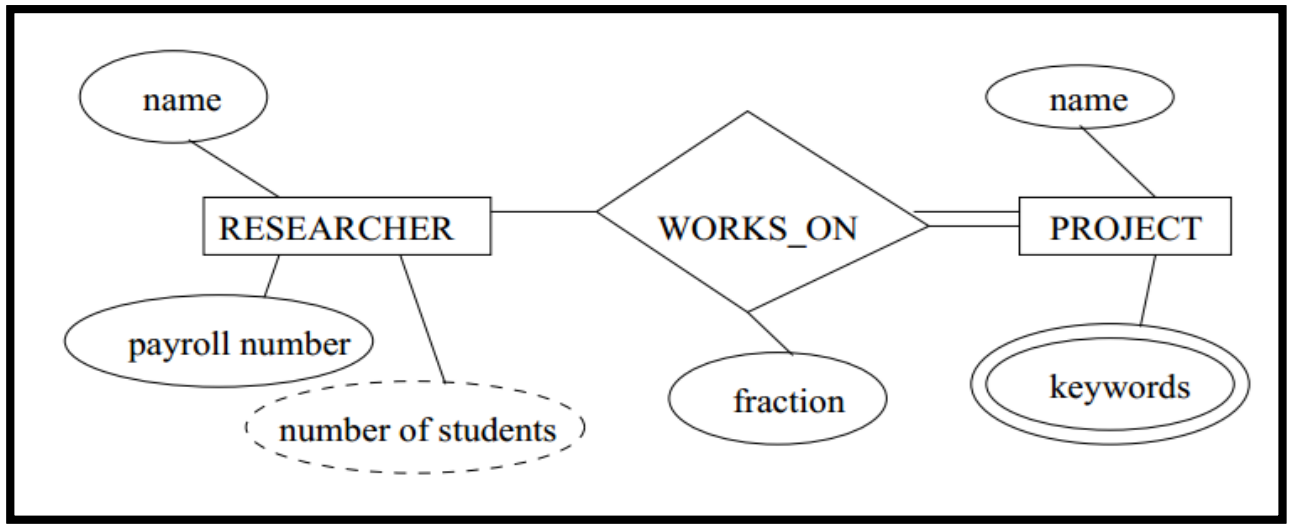
- **Partial Participation:** not necessarily total.
  - Example: Not every person has publication

Attributes (Relationship)

A researcher may work on several projects.

The fraction of her time devoted to a particular project could be an attribute of the WORKS ON relationship type.

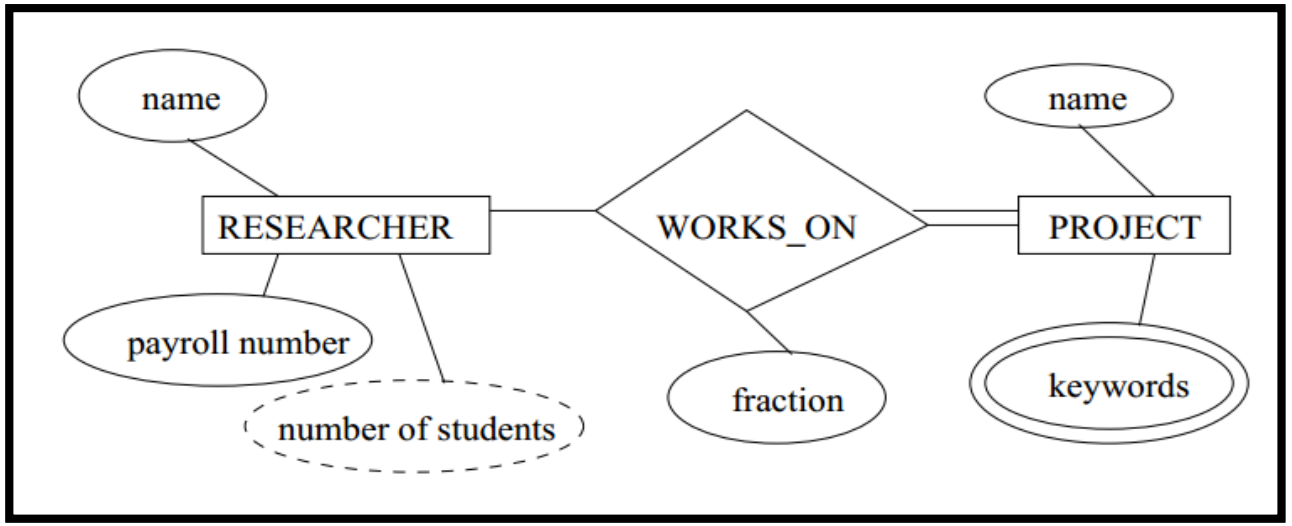
Question: Why not put the attribute on either researcher or project?



Attributes (Relationship)

Question: Why not put the attribute on either researcher or project?

Answer: If we put the attribute on the researcher, then we cannot know which project the researcher has spent time on. If we put the attribute on the project, then we cannot know who is spending time on that project.



More on Relationships

Binary relationship: relationship of degree 2

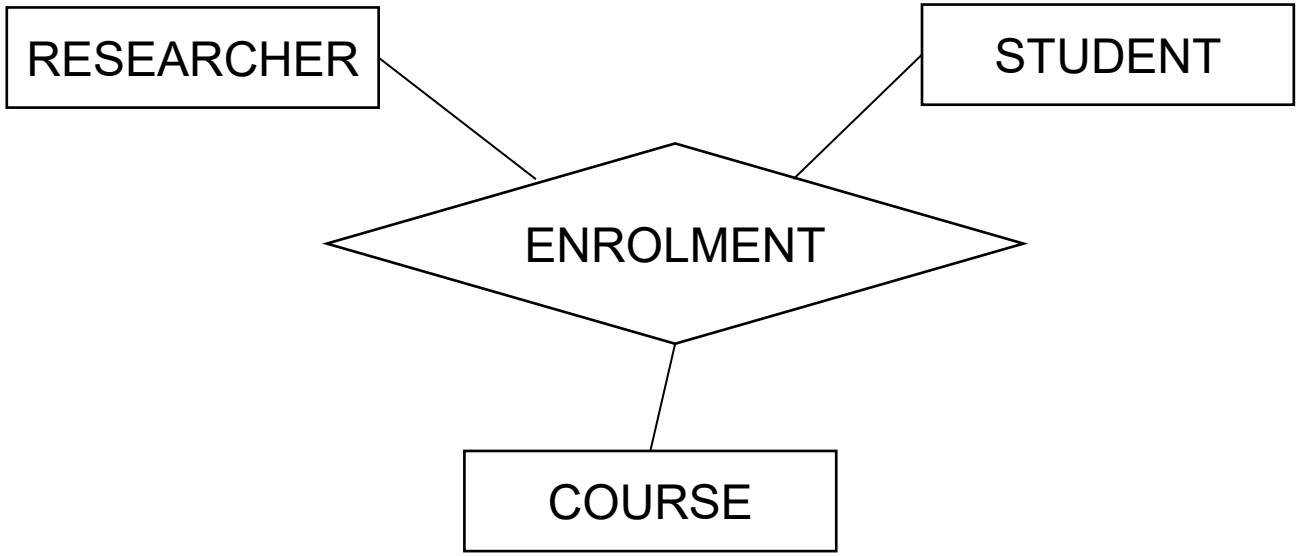
Degree (of a relationship): number of participating entity types.

Lectured cover binary relationships.

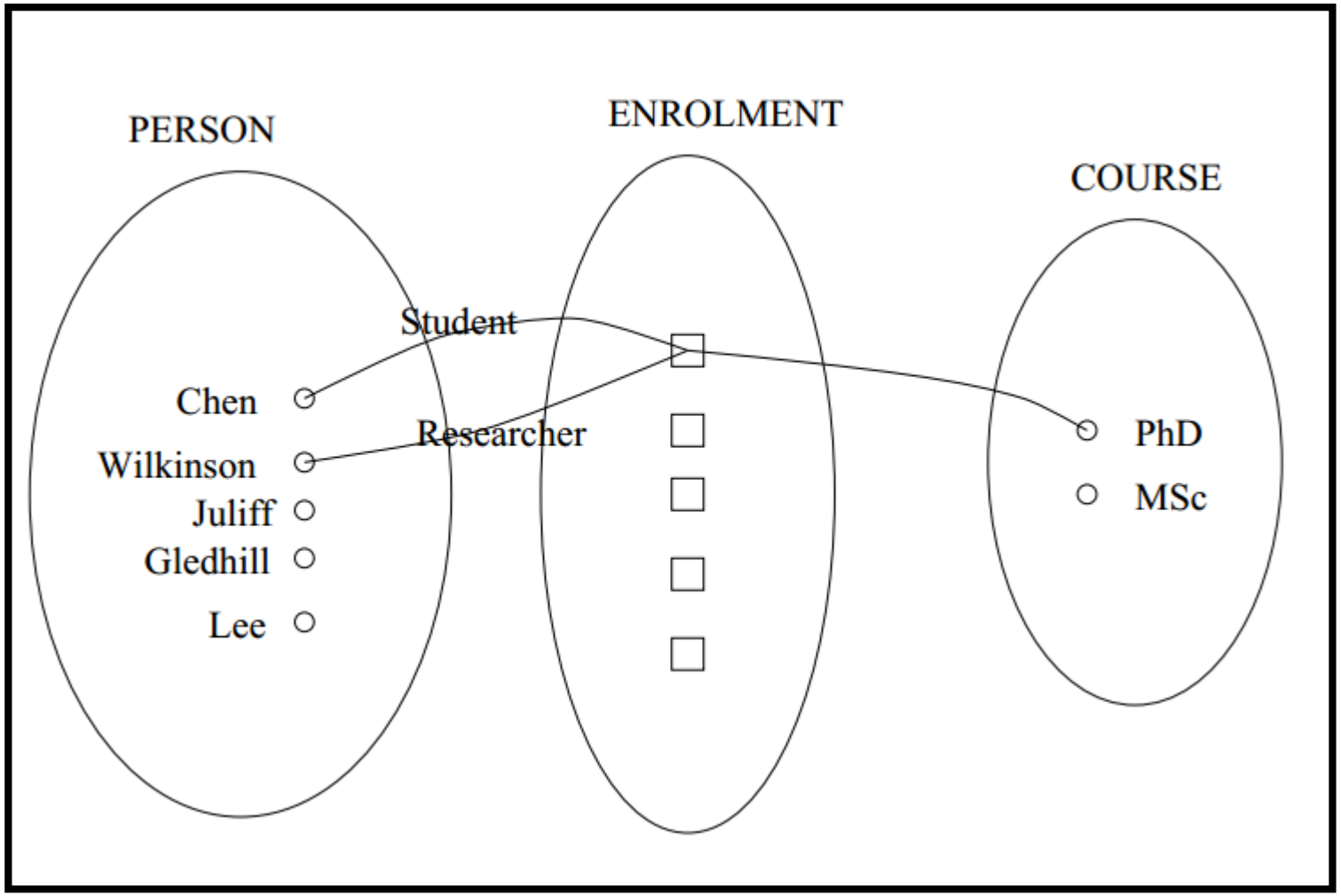
More on Relationships

Consider the setting: relationship between RESEARCHER, STUDENT and COURSE.

ENROLMENT could be a ternary relationship (degree 3)



More on Relationships



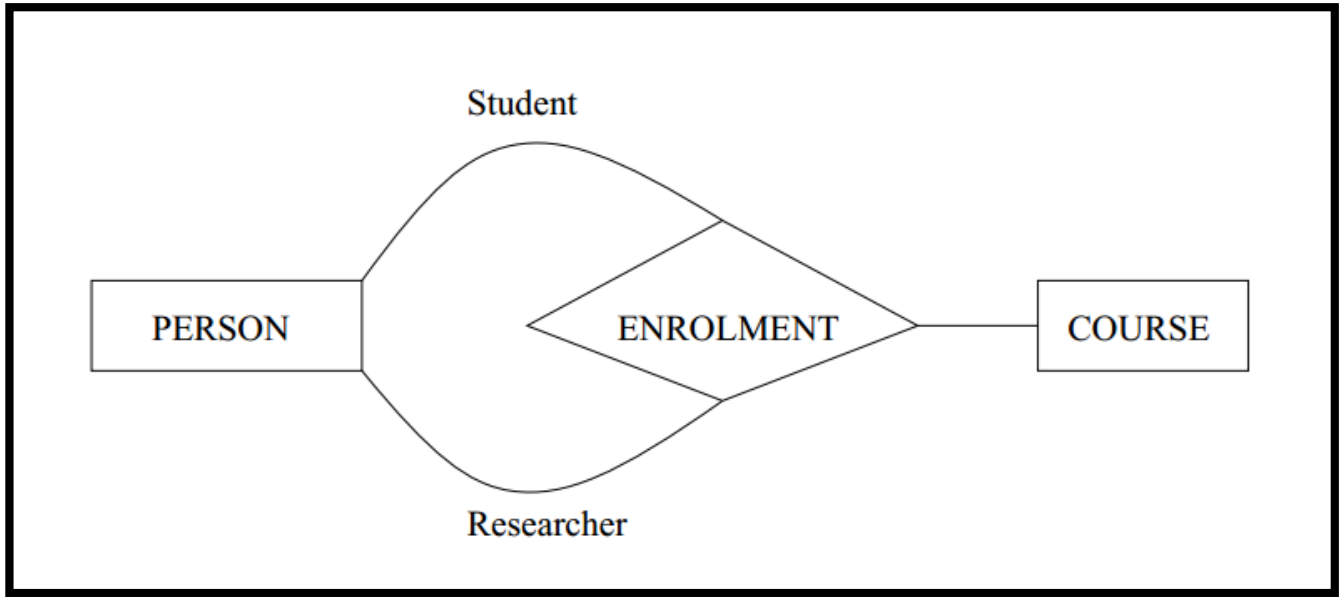
On the Degree of a Relationship:

The more entities in a relationship, the more careful you must be when describing it. What you think your expressing may not be what the diagram means exactly.

Rule of Thumb as you are starting out: If you think you have a ternary (or higher) relationship, try decompose it to binary relationships.

Backtrack: ENROLMENT

To make things interesting, this is another way of modelled the former.



Conceptual Data Modelling

1. Picking the right kind of **entity**: does it have significant properties and describe objects with independent instances? (Strong Entity or Weak Entity)

2. Picking the relevant **relationships**: does it provide a logical link between two (or more) entities? (Cardinality, Participation Constraint and Degree of Relationship, etc.)

The Best Data Model?

There is no single "best" design for a given application.

Most important aspects of a design (data model):

- correctness (satisfies requirements accurately)
- completeness (all reqs covered, all assumptions explicit)
- consistency (no contradictory statements)
- readability (easy to understand)

Potential inadequacies in a design:

- omits information that needs to be included
- contains redundant information (⇒ inconsistency)
- leads to an inefficient implementation
- violates syntactic or semantic rules of data model

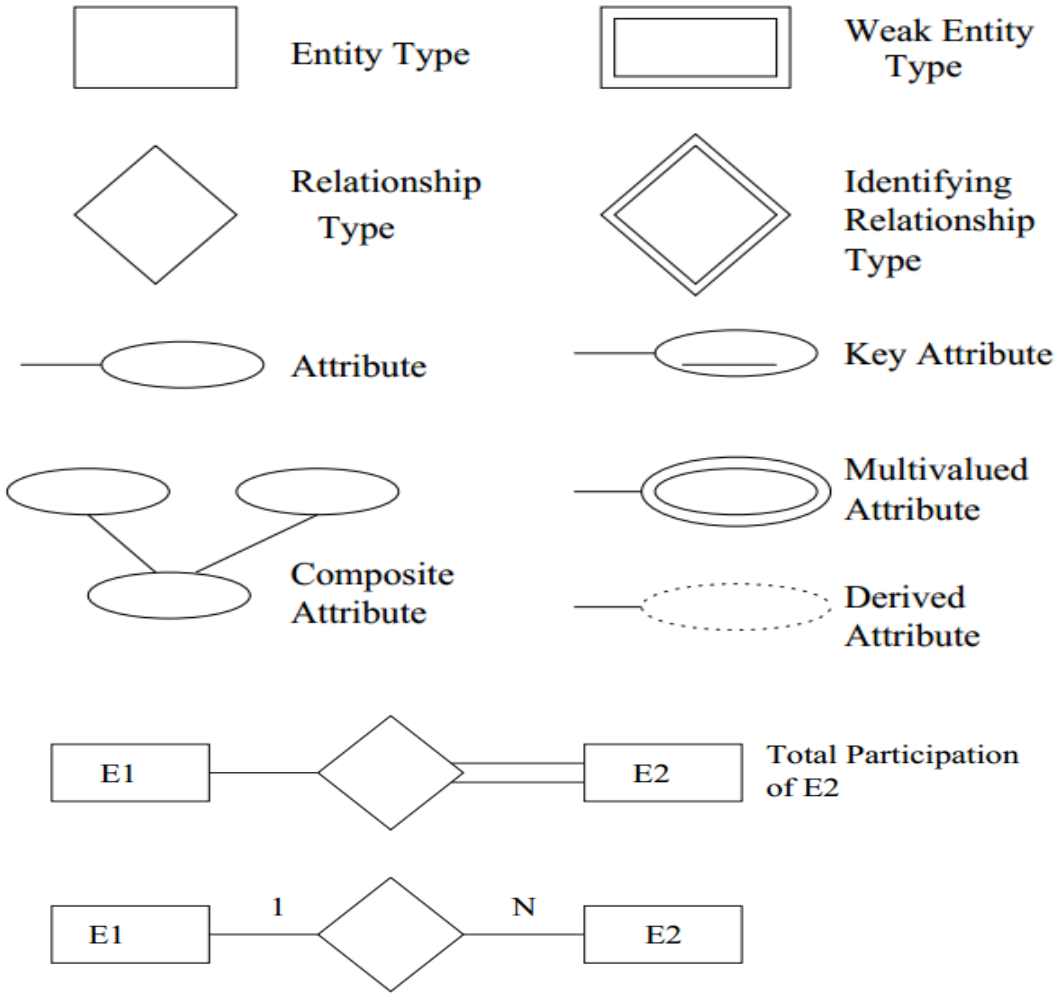
That's the Entity Relation Model

1. ER models gave us the first rules to capture the semantics

2. Systematically change it to Relational logical models

3. Allowed a lot of people to take a step from 'requirements from applications' to 'implementing the database'

ERD Notations



Final Exercise: A Library Database

- A **book** is uniquely identified by its *book id*. For each book, we also record its *title*, *price*, and *availability*.
- A **reader** is uniquely identified by his/her *reader id* and we also record his/her *name*, *phone number* and *address*. The address is composed of *street* and *suburb*.
- A **publisher** is uniquely identified by its *publisher id*. For each publisher, the *name* is also recorded.
- An **author** is uniquely identified by his/her *author id*. For each author, the *name*, *phone number* and *birth date* are also recorded.
- A **reader** can borrow zero or more books and a book can be borrowed by zero or more readers. Thus, we need to record the *starting date* and *ending date* for the borrowing relationship.
- A **publisher** can publish zero or more books and a book is published by exactly one publisher. We also need to record the *date of publication*.
- An **author** can write zero or more books and a book is written by one or more authors.

Exercise

Entity focused

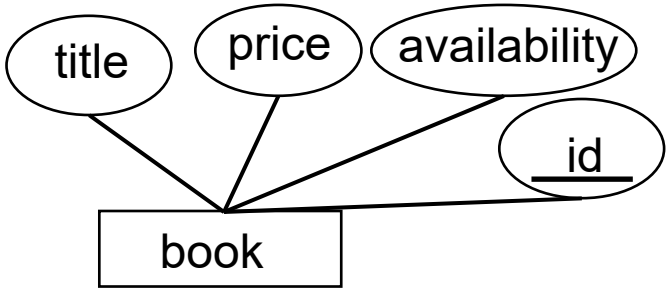
- A **book** is uniquely identified by its *book id*. For each book, we also record its *title*, *price*, and *availability*.
- A **reader** is uniquely identified by his/her *reader id* and we also record his/her *name*, *phone number* and *address*. The address is composed of *street* and *suburb*.
- A **publisher** is uniquely identified by its *publisher id*. For each publisher, the *name* is also recorded.
- An **author** is uniquely identified by his/her *author id*. For each author, the *name*, *phone number* and *birth date* are also recorded.

Relation focused

- A **reader** can borrow zero or more books and a book can be borrowed by zero or more readers. Thus, we need to record the *starting date* and *ending date* for the borrowing relationship.
- A **publisher** can publish zero or more books and a book is published by exactly one publisher. We also need to record the *date of publication*.
- An **author** can write zero or more books and a book is written by one or more authors.

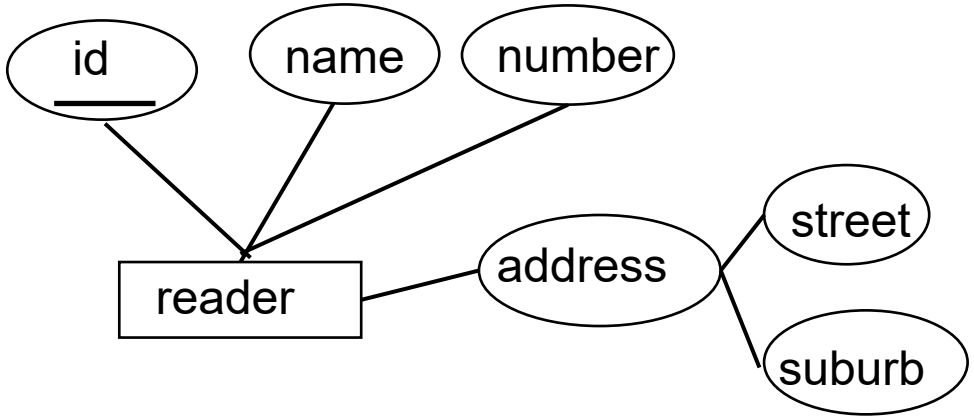
Exercise (1/7)

A book is uniquely identified by its book id. For each book, we also record its title, price, and availability.



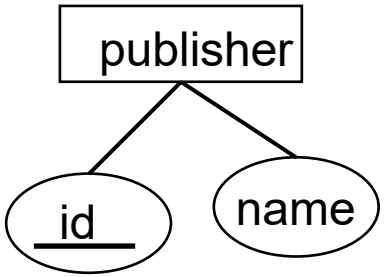
Exercise (2/7)

A reader is uniquely identified by his/her reader id and we also record his/her name, phone number, dob and address. The address is composed of street and suburb.



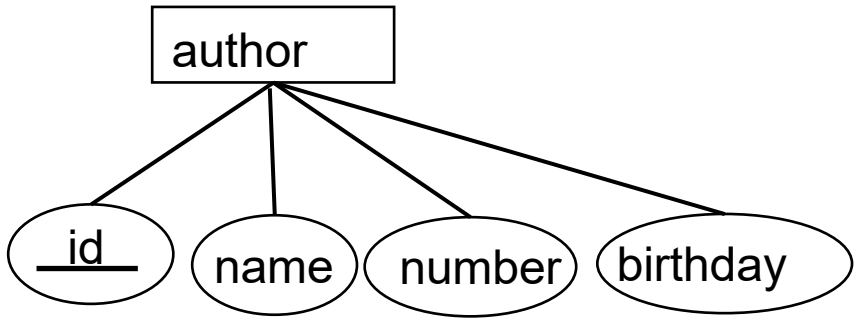
Exercise (3/7)

A publisher is uniquely identified by its publisher id. For each publisher, the name is also recorded.



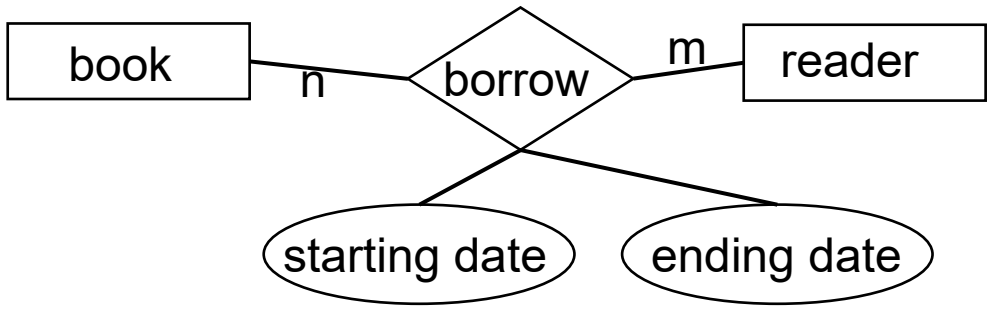
Exercise (4/7)

An author is uniquely identified by his/her author id. For each author, the name, phone number and birth date are also recorded.



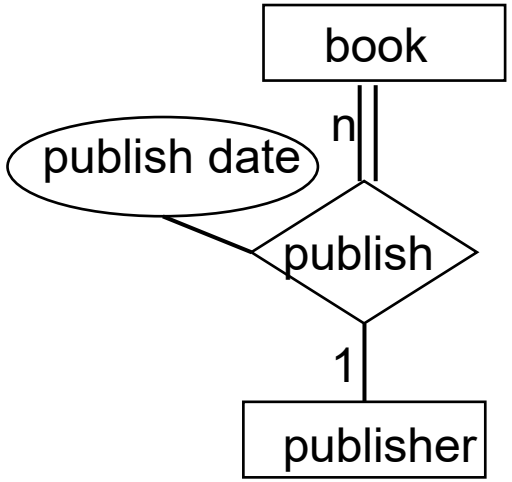
Exercise (5/7)

A reader can borrow zero or more books and a book can be borrowed by zero or more readers. Thus, we need to record the starting date and ending date for the borrowing relationship.



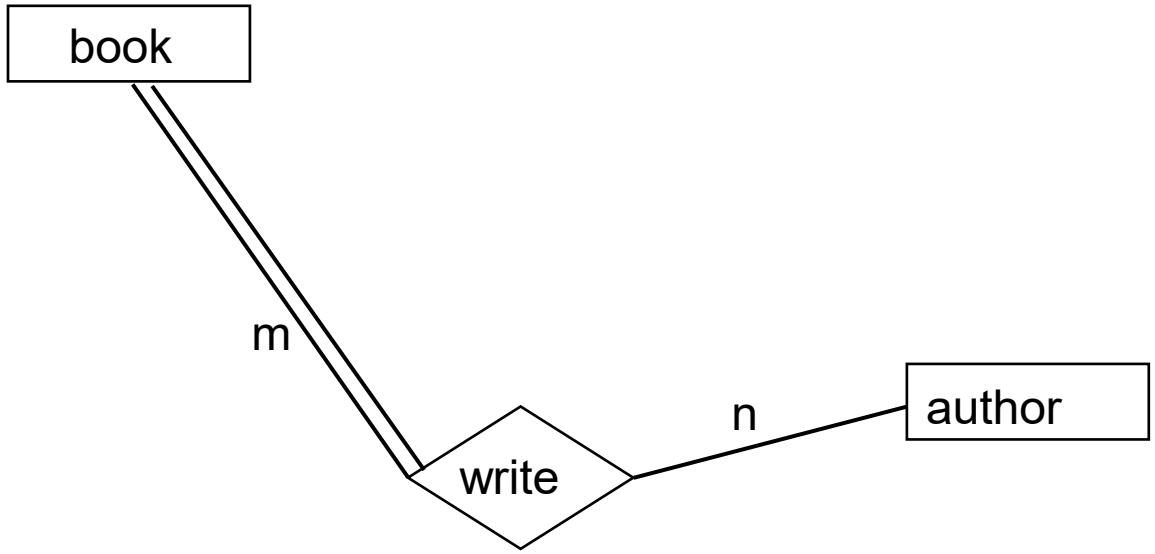
Exercise (6/7)

A publisher can publish zero or more books and a book is published by exactly one publisher. We also need to record the date of publication.

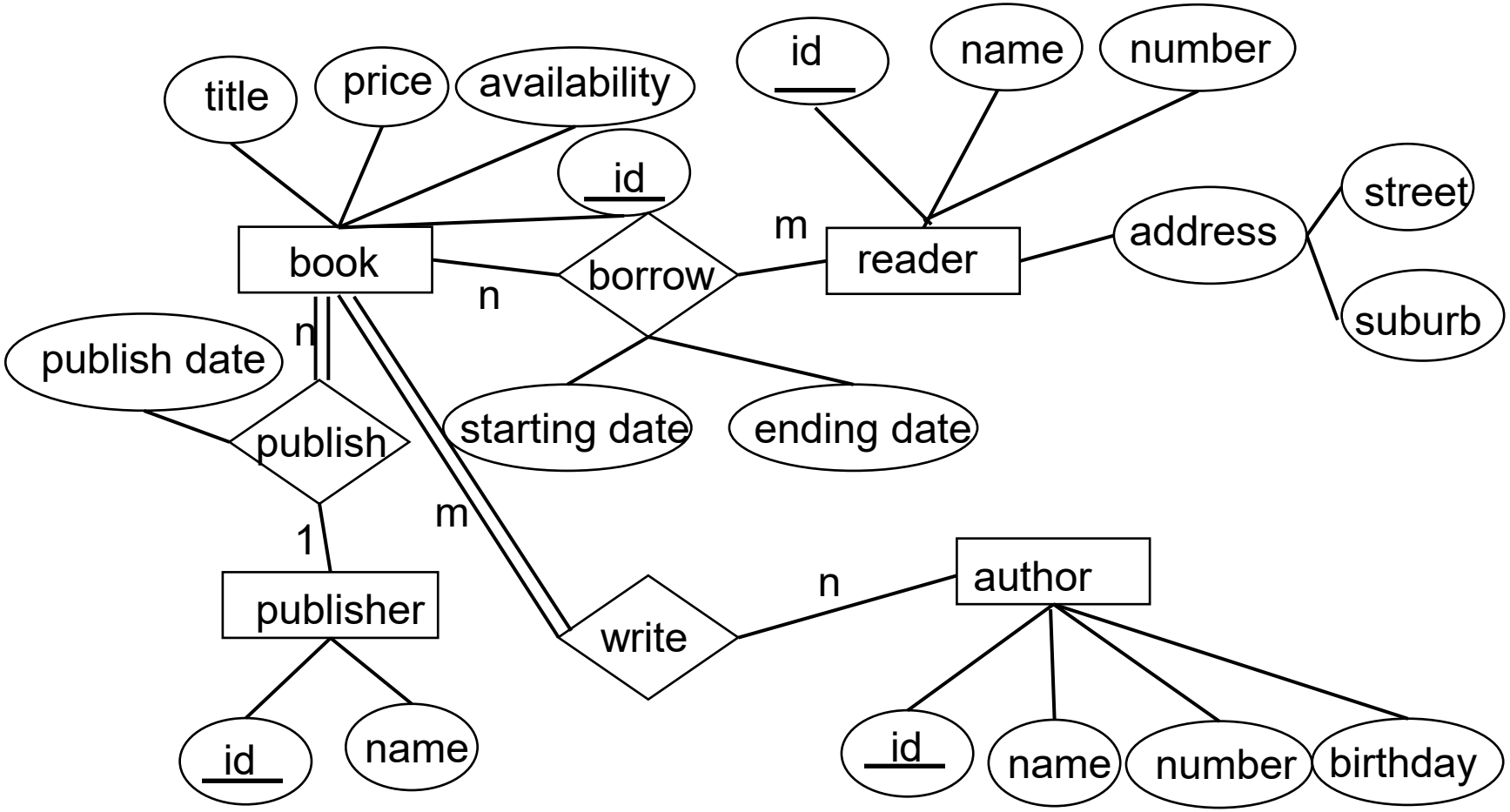


Exercise (7/7)

An author can write zero or more books and a book is written by one or more authors.



Full Sample Solution



Summary

A high-level process that converts applications into a data model

1. describe the information in a way that is suitable for database implementation (e.g., entities: students, courses, accounts, branches, patients, ...)

2. describe how information is related (e.g., John is enrolled in COMP3311, Andrew's account is held at Comm Bank)

Learning Outcome:

1. Given application requirements, how to use ER-diagram to accurately reflect the application (and all the specific requirements).

2. Understand ER model as a solution for expressing any application requirement.

NOTE: You must use the notation in the lecture for this course.