COMP9311:
DATABASE SYSTEMS

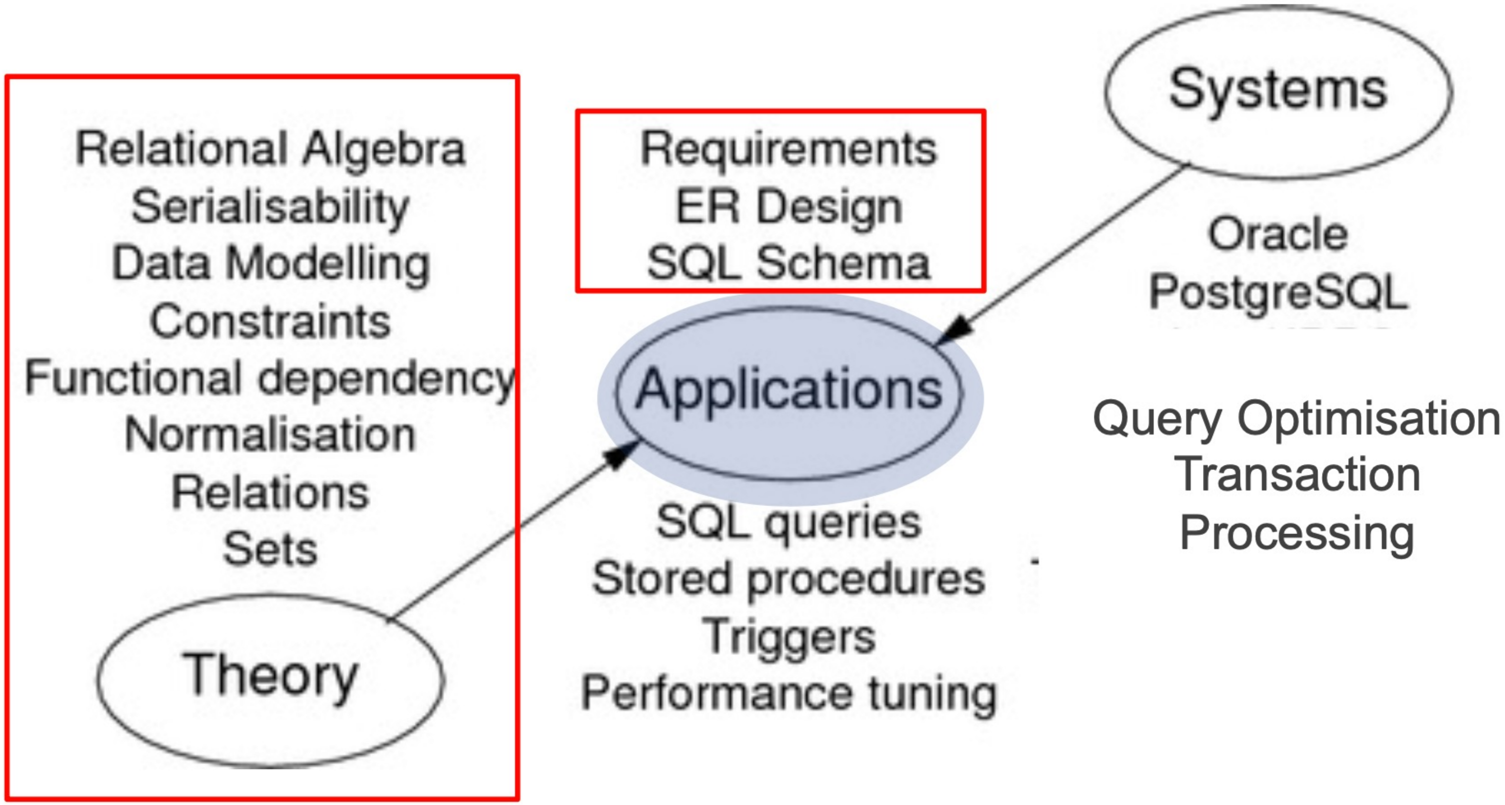
Term 1 2024

Week 1 – Data Modelling

By Xiaoyang Wang, CSE UNSW

Disclaimer: the course materials are sourced from previous offerings of COMP9311 and COMP3311

Overview of the Databases Field



Relational Algebra
Serialisability
Data Modelling
Constraints
Functional dependency
Normalisation
Relations
Sets

Theory

Requirements
ER Design
SQL Schema

Applications

SQL queries
Stored procedures
Triggers
Performance tuning

Systems

Oracle
PostgreSQL

Query Optimisation
Transaction
Processing

Database Application Development

A variation on standard software engineering process:

- analyse application requirements
- develop a data model to meet these requirements
- define operations (transactions) on this model
- implement the data model as relational schema
- implement operations via SQL and procedural PLs
- construct an interface to these operations
- At some point, populate the database (may be via interface)

Data Modelling

Aims of data modelling:

- describe what information is contained in the database (e.g., entities: students, courses, accounts, branches, patients, ...)
- describe relationships between data items (e.g., John is enrolled in COMP3311, Andrew's account is held at Coogee)
- describe constraints on data (e.g., 7-digit IDs, students can enrol in no more than four courses per term)

Data modelling is a design process

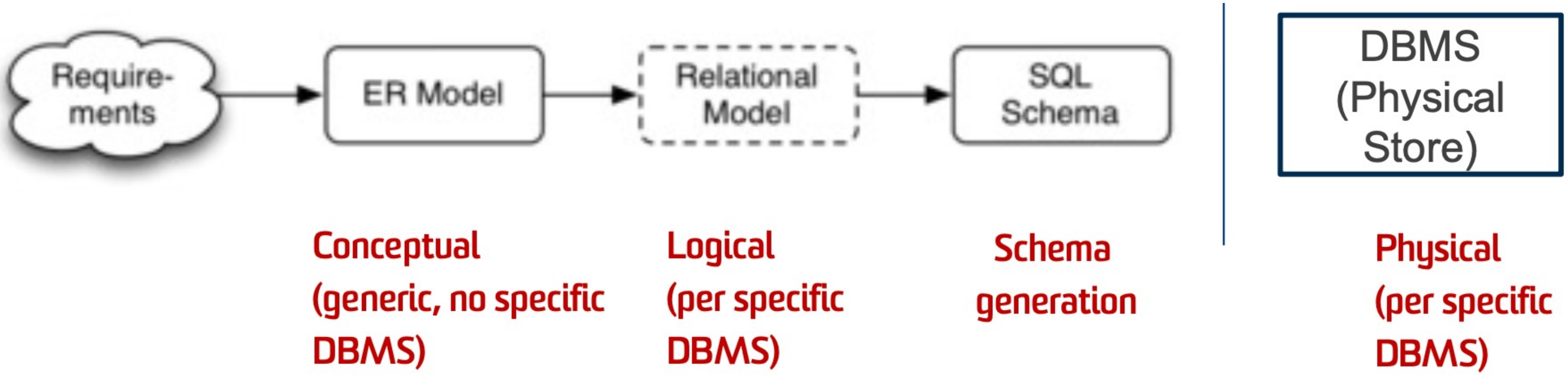
- converts requirements into a data model

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



Requirements

ER Model

Relational Model

SQL Schema

DBMS (Physical Store)

Conceptual (generic, no specific DBMS)

Logical (per specific DBMS)

Schema generation

Physical (per specific DBMS)

Some Design Ideas

Consider the following when you work through a design exercise:

- start simple ... evolve design as problem better understood
- identify objects (and their properties), then relationships
- most designs involve kinds (classes) of people
- keywords in requirements suggest data/relationships (rule-of-thumb: nouns → data, verbs → relationships)
- consider all possible data, not just what is available

Example - Gmail Data Model

Consider the Google Mail System:

Let’s develop an informal data model for it by identifying:

- the data items involved (objects and their attributes)
- relationships between these data items
- constraints on the data and relationships

Quality of Designs

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)

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

Entity-Relationship Data Modelling

In ER, The world is viewed as a collection of inter-related “entities”.

ER has **three** major modelling constructs:

- entity: objects (“things”) in your world that you are interested

  Person, Restaurants, Books, University Courses,...

- attribute: data item describing a property of interest

  Person (name, phone number, DOB, ...)

- relationship: association between entities (objects)

  Person dines-at Restaurant

Entity-Relationship (ER) Diagrams

ER diagrams are a graphical tool for data modelling.

An ER diagram consists of:

- a collection of entity set definitions
- a collection of relationship set definitions
- attributes associated with entity and relationship sets
- connections between entity and relationship sets

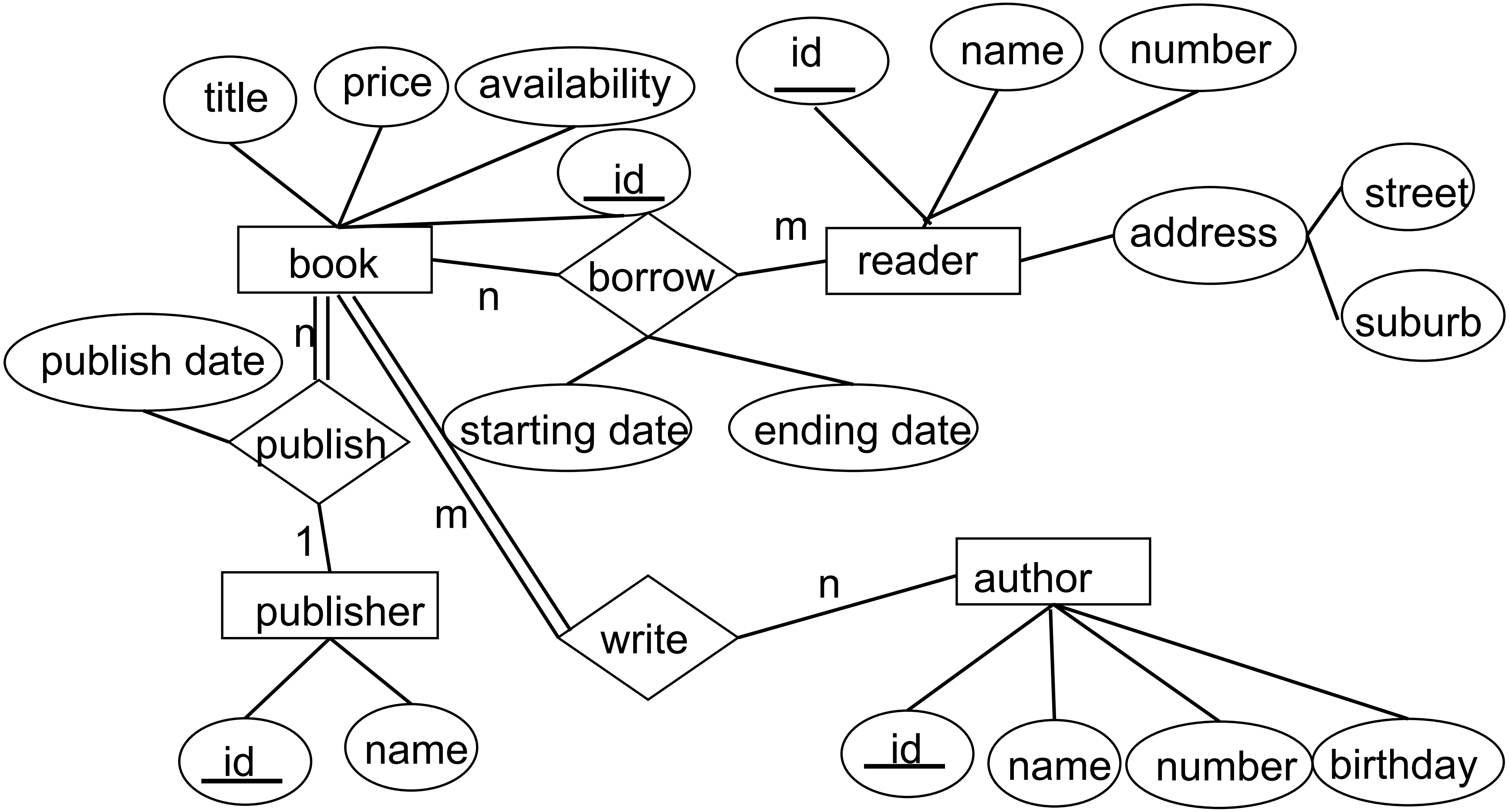
Terminology: when discussing "entity sets", we frequently say just "entity”

The ER model is not a standard, so many variations exist.

Lecture notes use simple notations -> as ‘COMP9311 standard’.

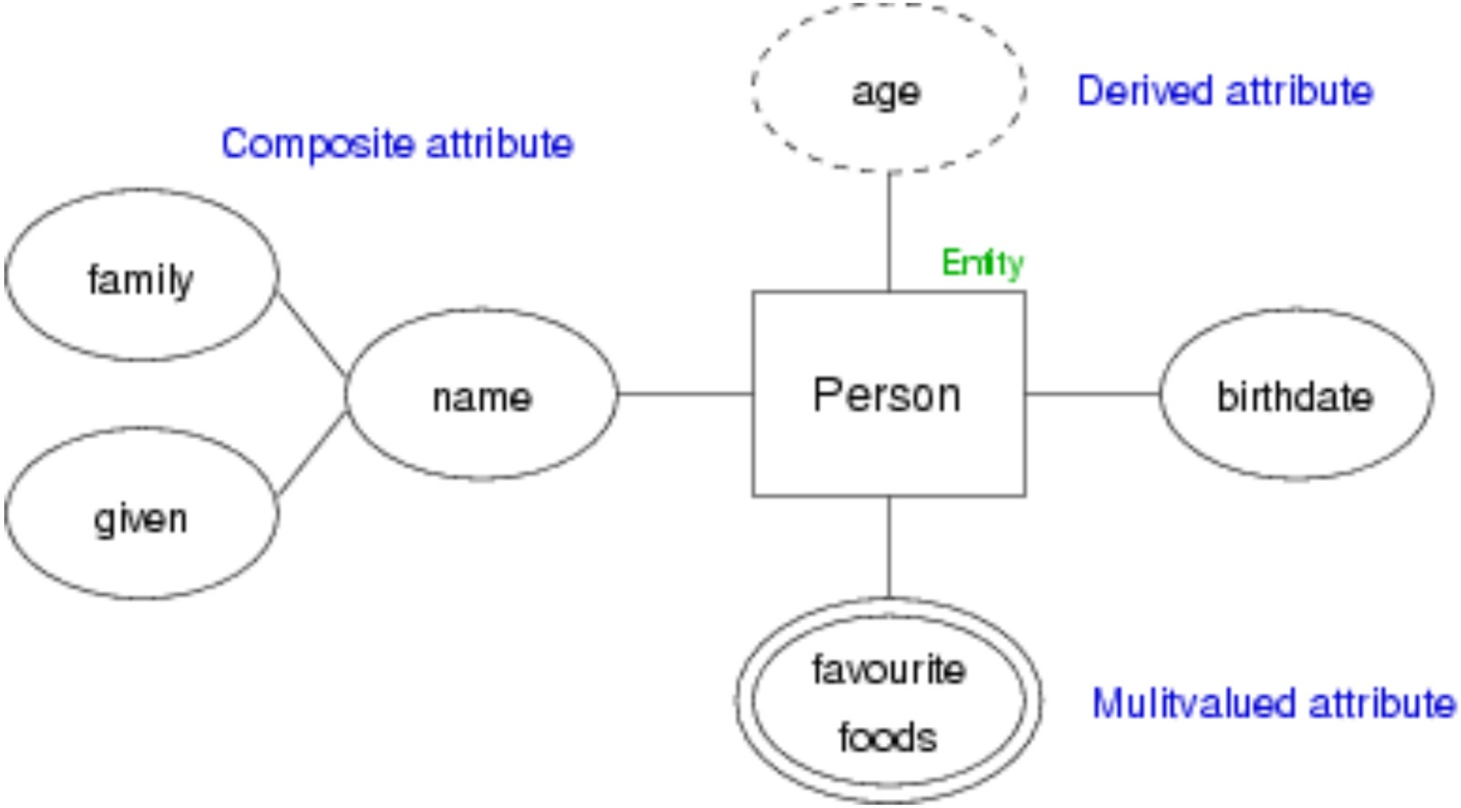
Entity-Relationships

Example ER Diagram: entities, attributes, relationships/connections



Attribute

Example of attribute notations



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.
- e.g., Entity = Student, Attributes = Student number, name…

Attributes can also be multi-valued

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

Question

Do we need multi-value 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 colour 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 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"*

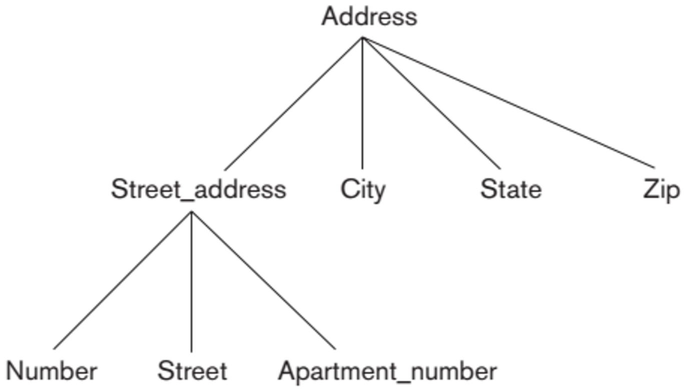


Figure 7.4
A hierarchy of composite attributes.

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?**

Entity

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

- An entity type describes the **schema** for a *set of entities* that share the same structure.
- 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.
- An entity type is represented in ER diagrams as a <u>rectangular box</u> enclosing the entity type name.

Entity Type Example

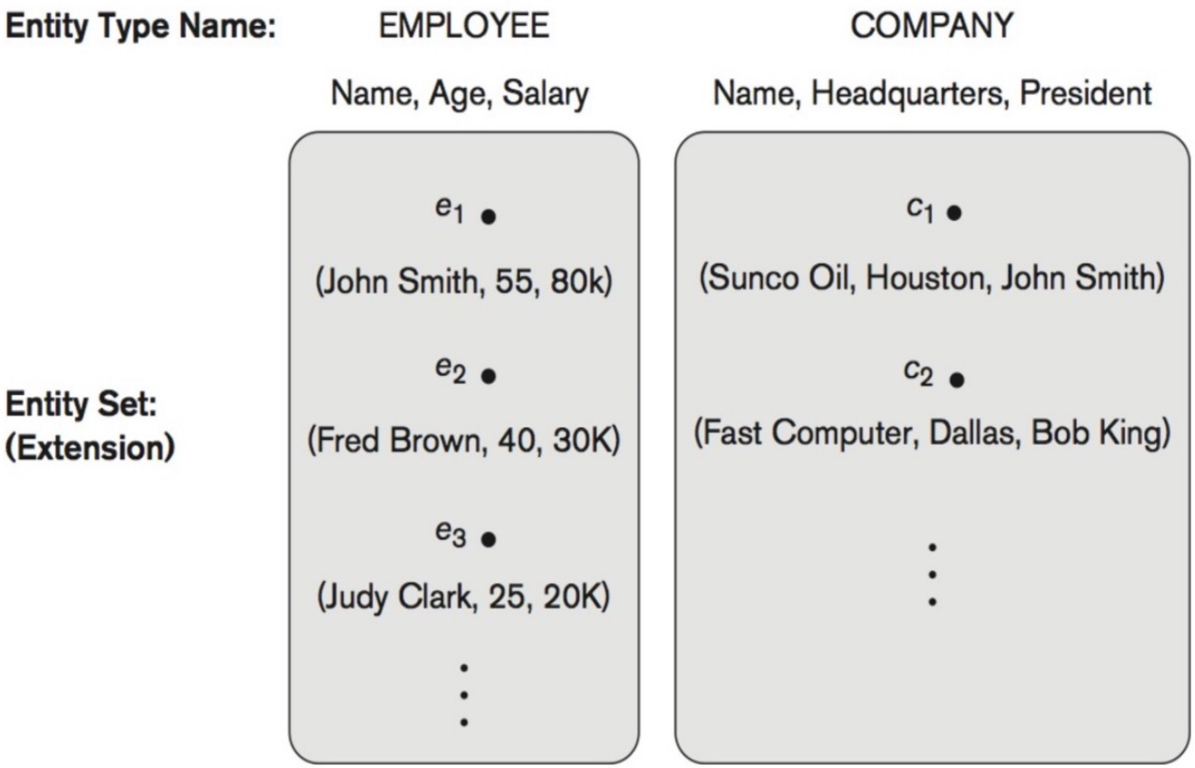


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

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, {Colour})

Car instance:

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

NULL Value

What if an instance doesn't have a value?

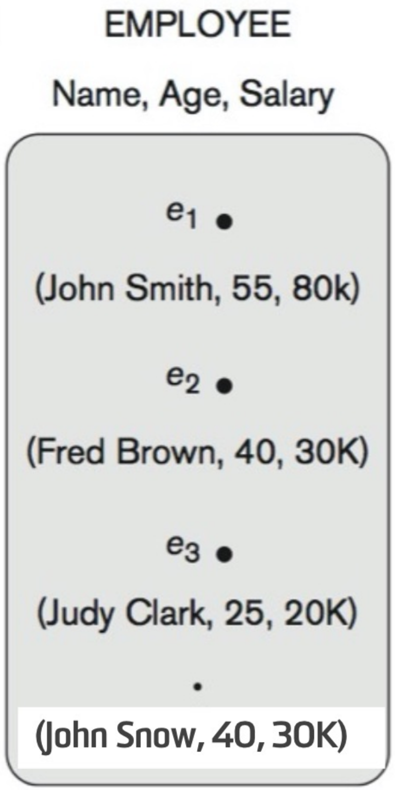
Examples:

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

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.

Entity Sets and Keys

Entities of an entity type, say EMPLOYEE needs a key to distinguish each other in a set.



Key (superkey): any set of attributes whose set of values are distinct over entity set

- natural (e.g., name+age+salary) or artificial (e.g., employee number)
- Keys containing multiple attributes as composite keys.

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

Primary key = candidate key chosen by DB designer later in the development stage

Keys are indicated in ER diagrams by underlining

Key

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

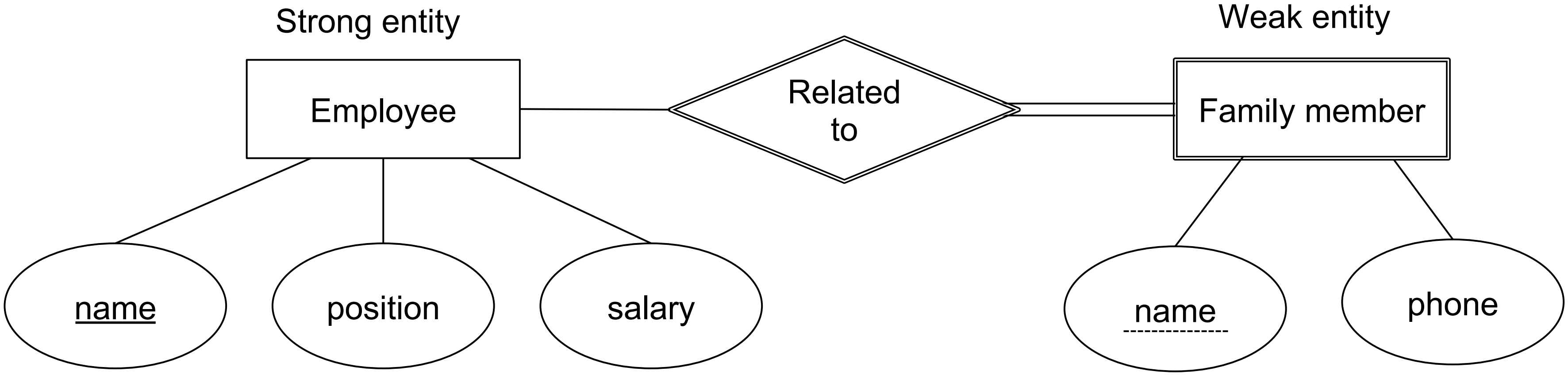
A: Since the entity is a **set**, in the worst case where there is no natural composite key, the set of all it’s attributes together should uniquely identify its values (strictly speaking).

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

Weak Entity Type

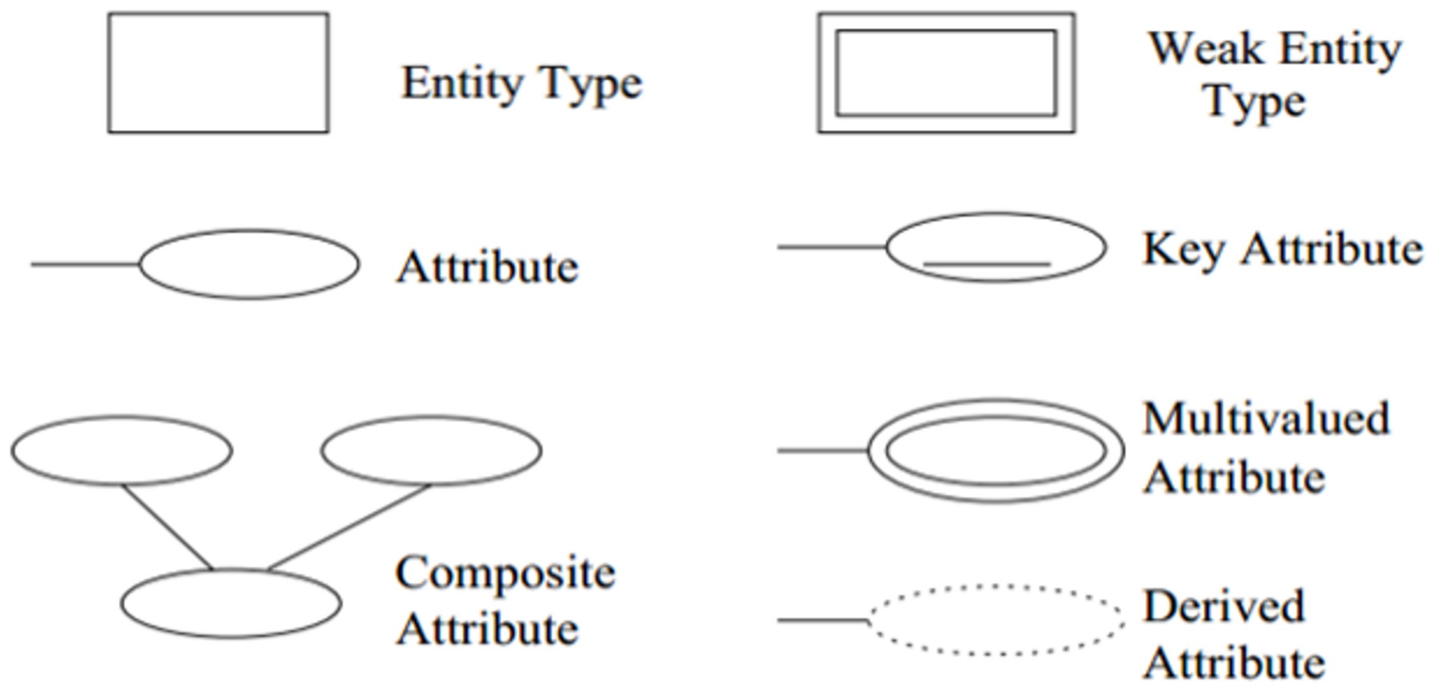
Weak entities

- exist only because of association with strong entities.
- typically, these entities do not have key of their own; can only be identified by considering the primary key of another (owner) entity
- must have total participation in the relationship with the owner entity
- could have a partial key/discriminator



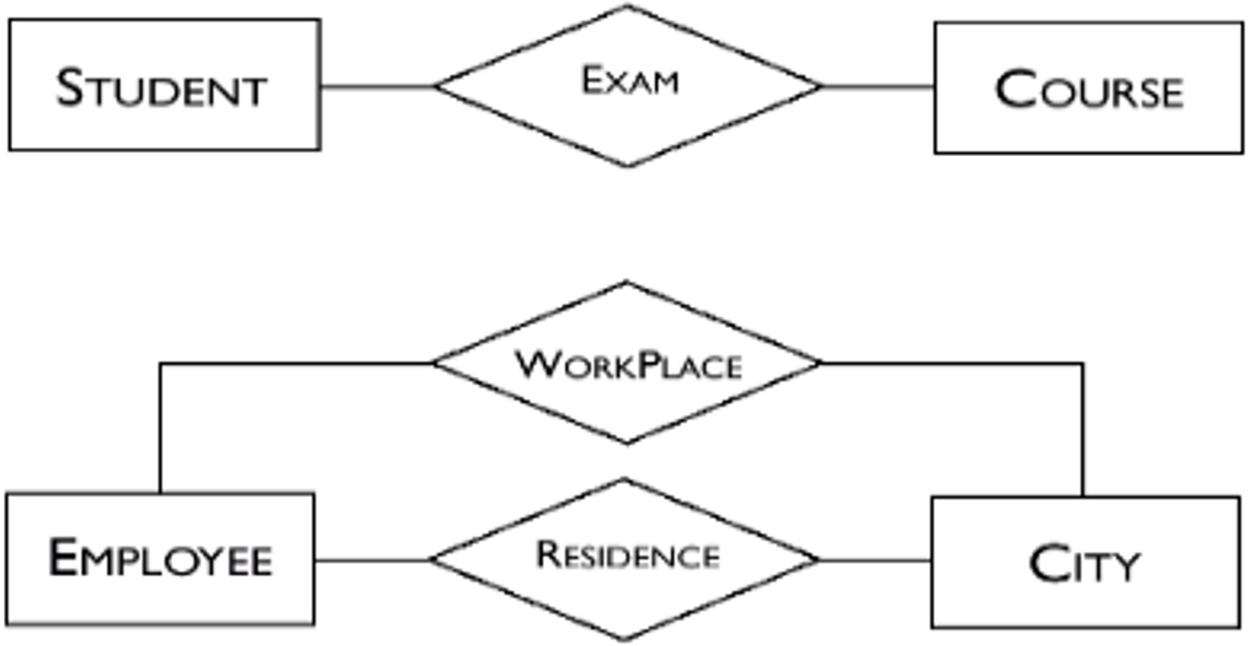
Visualizing an ER Data Model

Notations:



Relationship

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

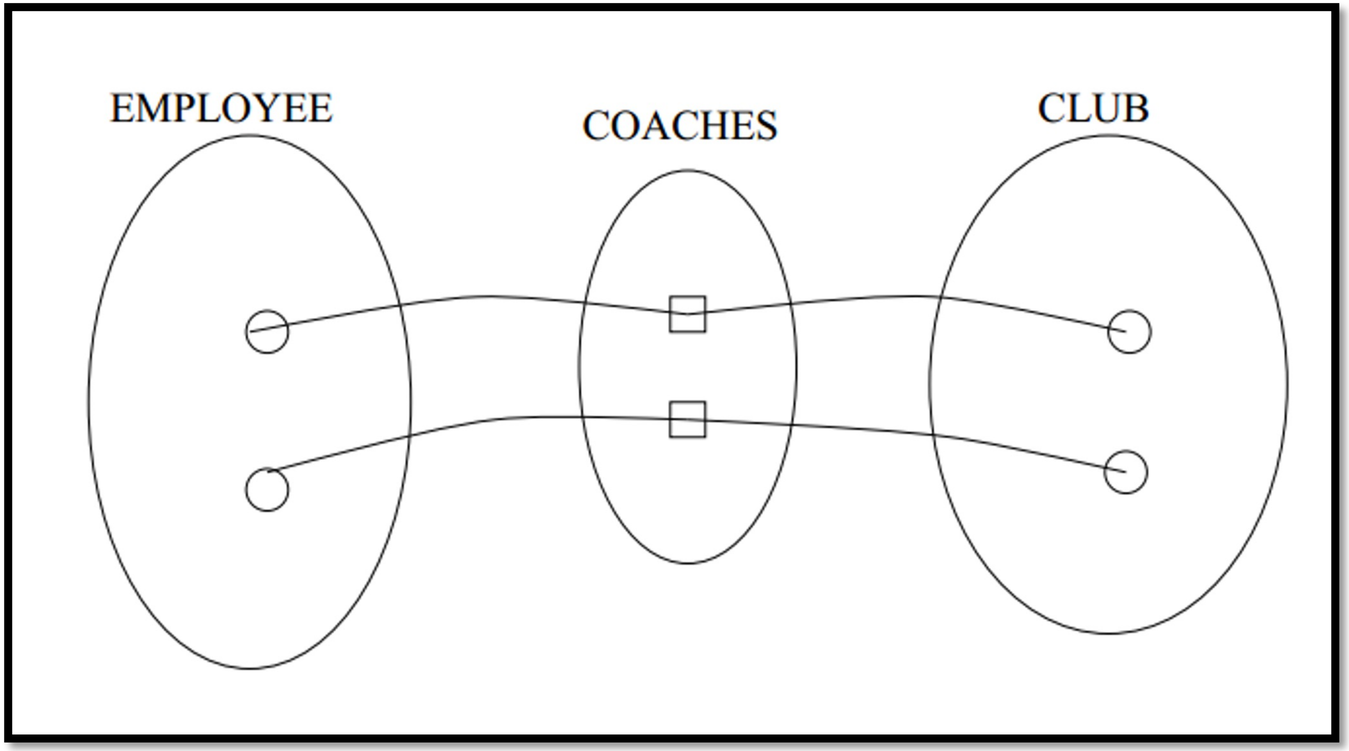


Relationship

- Relationships relates one entity type to another entity type.
- *Relationship set*: collection of relationships of the same type
- An entity can be related to more than one other entities and will have a different role to play for each relationship
- An entity can also have different relationships with another entity.
- Relationships can be illustrated using *occurrence diagrams*

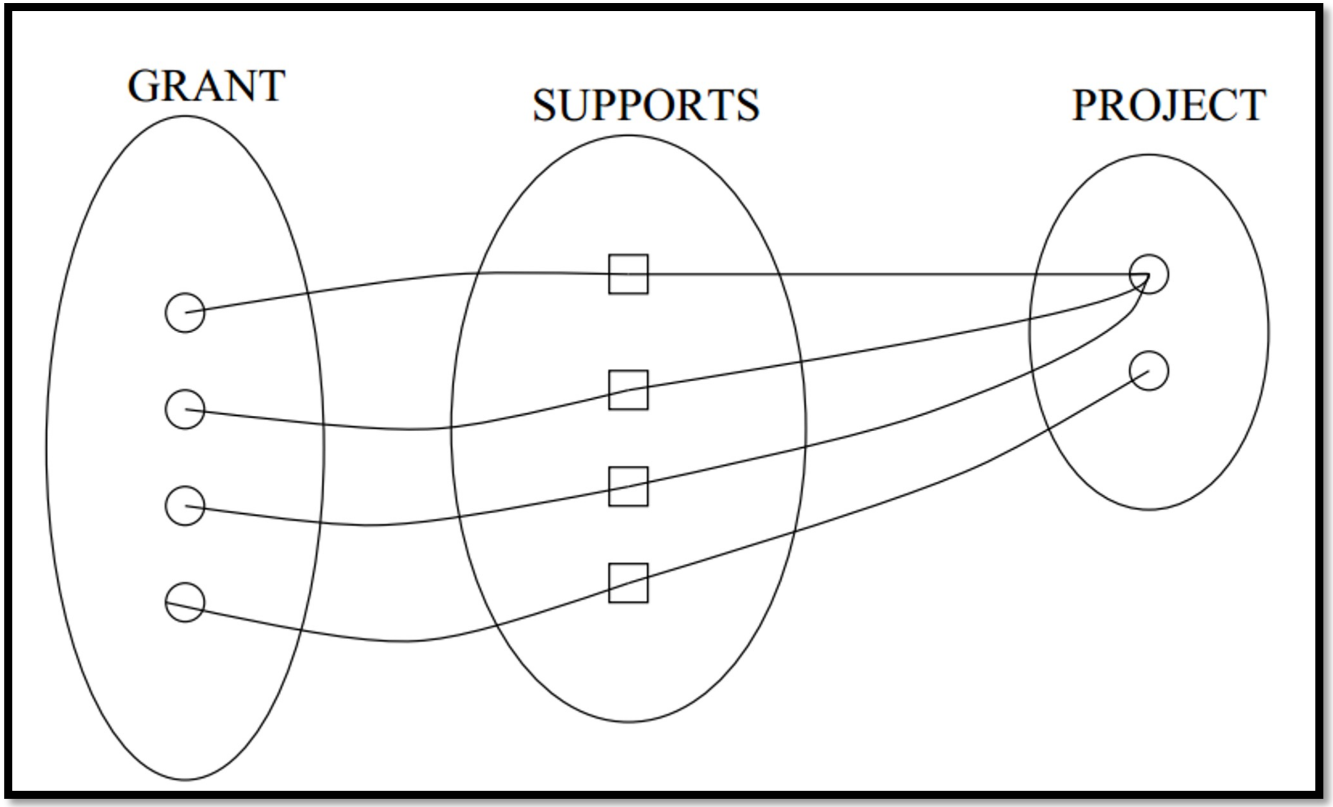
Practice

Coaches is a relationship. Its instances describes the relation between a coach and a club.



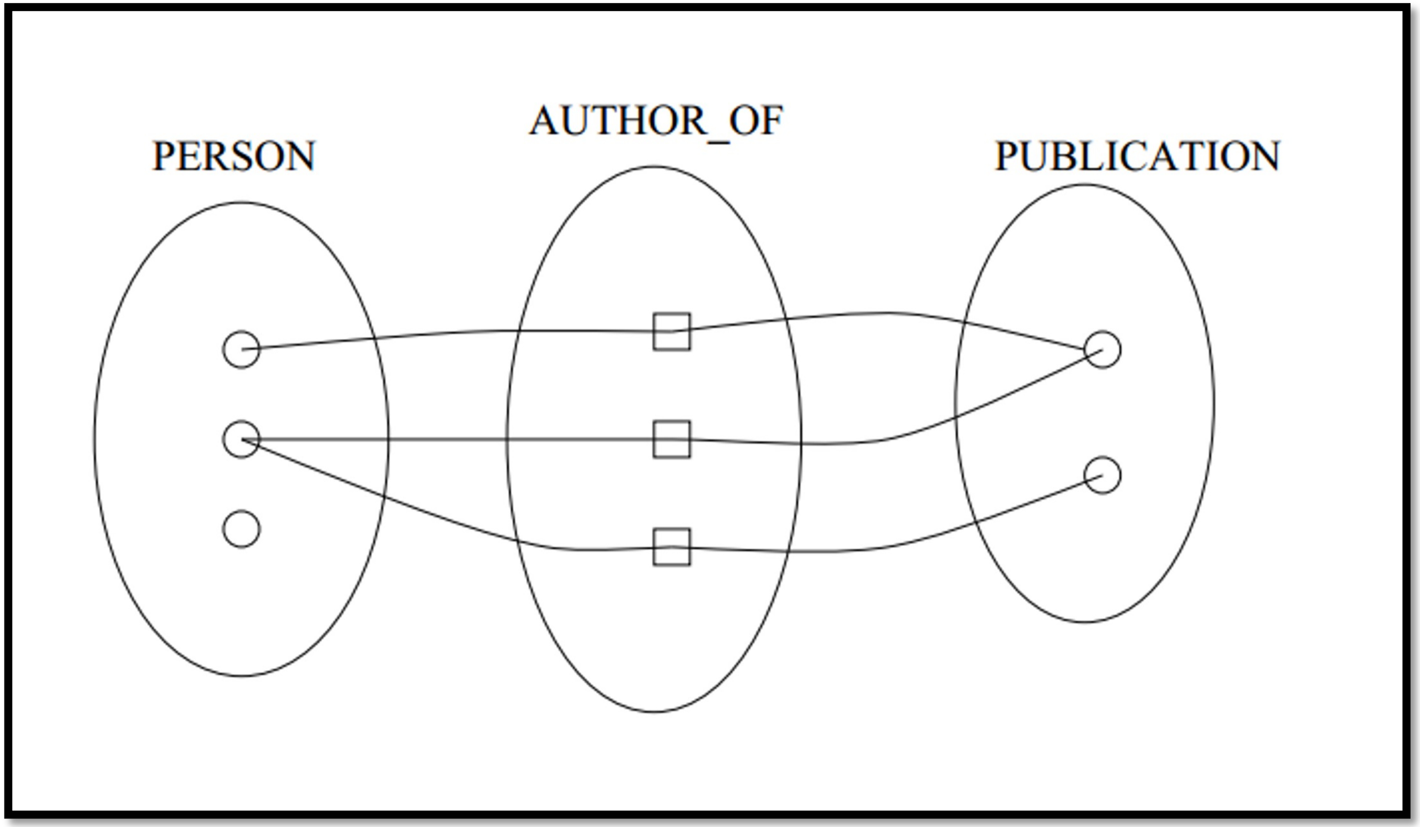
Practice

Supports is a relationship. Its instances describe the how grants are given to projects.



Practice

Author-of is a relationship. Its 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

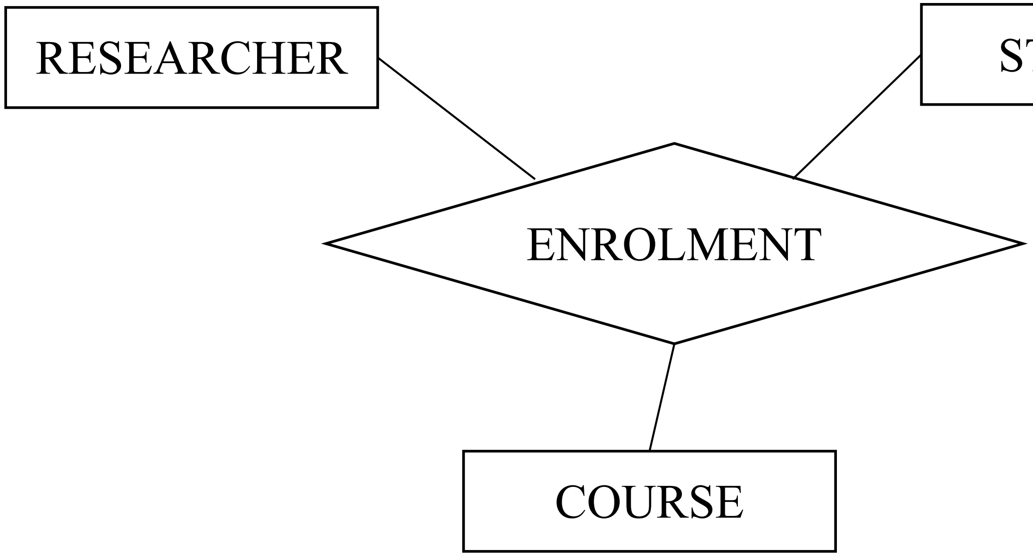
- Degree = the number of entities involved in reln (in ER model, ≥ 2)
- Cardinality = the number of relationship instances an entity can participate in
- Participation = must every entity be in the relationship

Degree of a Relationship

Binary relationship: relationship of degree 2

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

Lectured cover binary relationships.



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.

Placing ‘constraints’ on the relationships

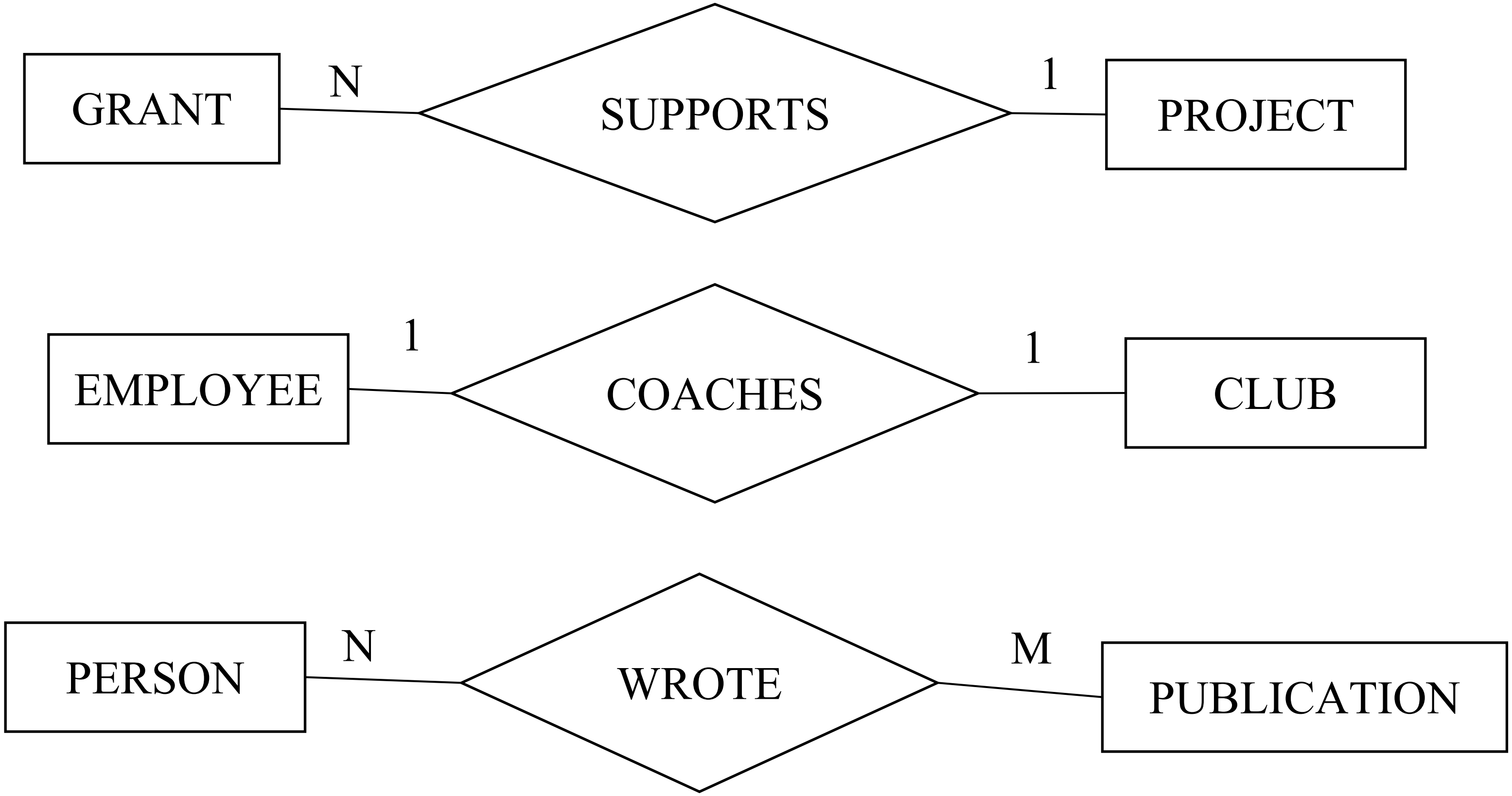
Cardinality: the number of relationship instances an entity can participate in.

There are three types of cardinality in relationship: 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.

Placing 'constraints' on the relationships

We can also show this in an ERD:



Placing ‘constraints’ on the relationships

Question: Will each instance participate in this relationship?

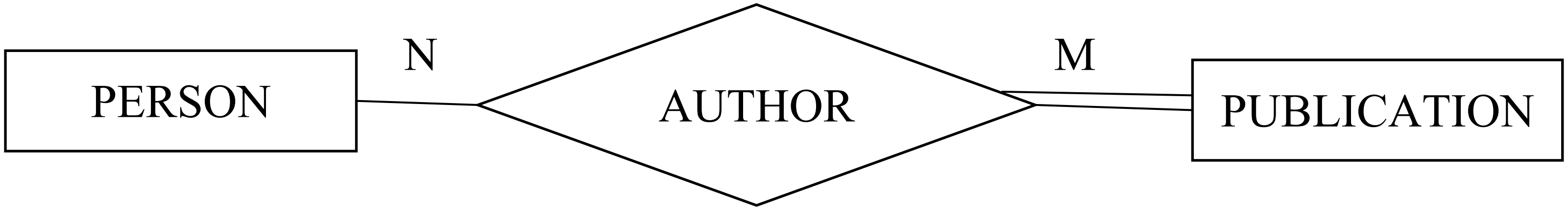
For example, all university students must be enrolled to university/universities.

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

Placing ‘constraints’ on the 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.

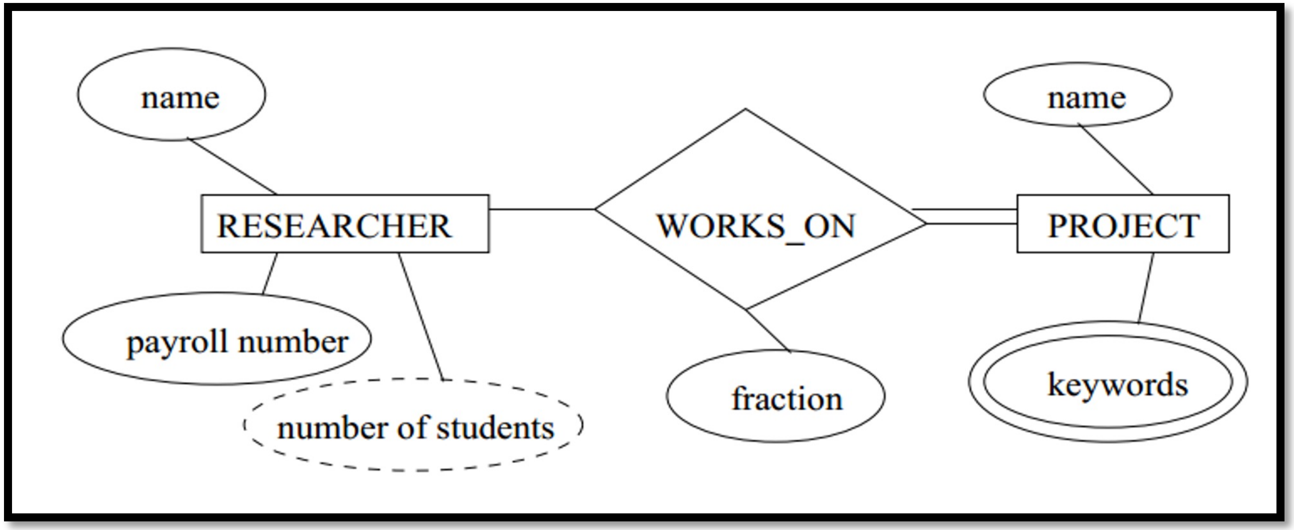
- Example: Not every person has publication

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.

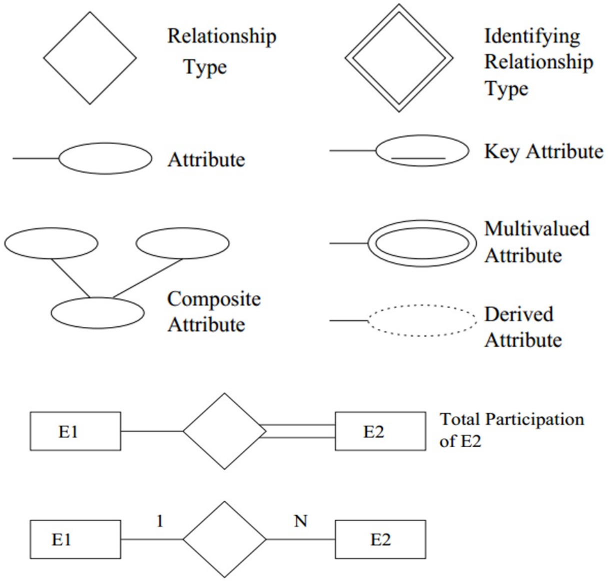
Q: Why not put the attribute on either researcher or project?



Conceptual Data Modelling

Pick the right kind of **entity**: does it have significant properties and describe objects with independent instances?

Pick the relevant **relationships**: does it provide a logical link between two (or more) entities?



Design Using the ER Model

ER model: simple, powerful set of data modelling tools

Allowed a lot of people to take a step from ‘requirements of applications’ to ‘implementing the database’

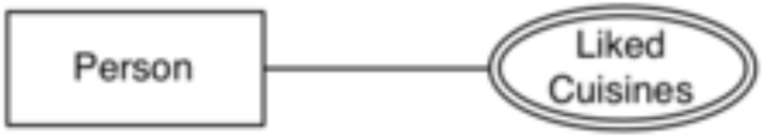
Some considerations in designing ER models:

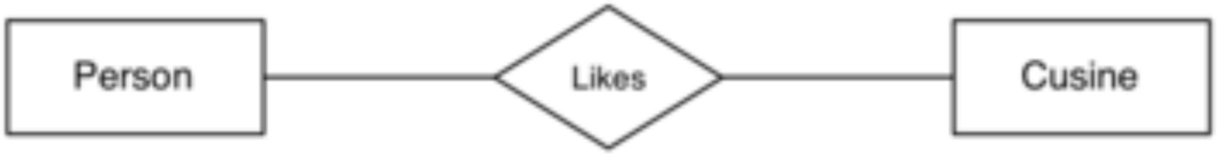
- should an "object" be represented by an attribute or entity?
- is a "concept" best expressed as an entity or relationship?
- should we use n-way relationship or several 2-way relationships?
- is an "object" a strong or weak entity? (usually strong)

Entities vs. Attributes

The following two diagrams both represent

a person has some types of food that they like





Why might we favour one over the other?

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: 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

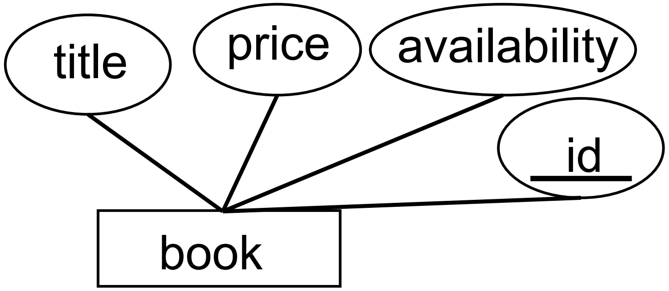
- A **book** is uniquely identified by its *book-id*. For each book, we also record its *title*, *price*, and *availability*.
- A **reader** is uniquely identified by his/her *reader-id* and we also record his/her *name*, *phone-number* and *address*. The address is composed of *street* and *suburb*.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

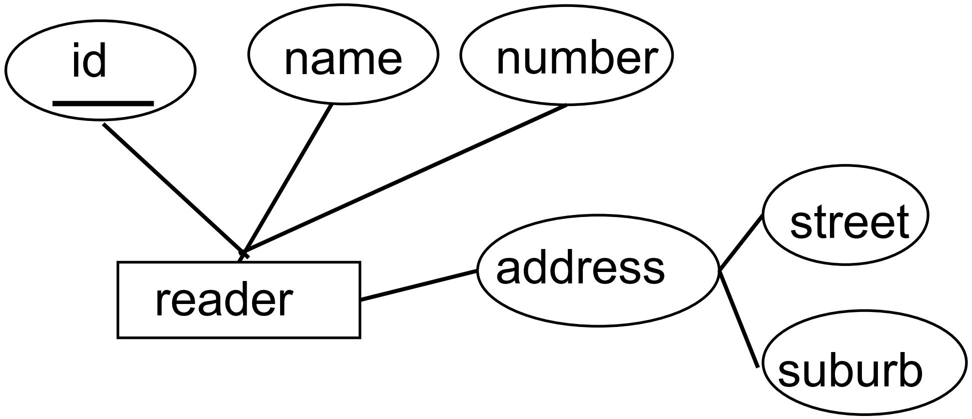
Exercise (1/7)

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



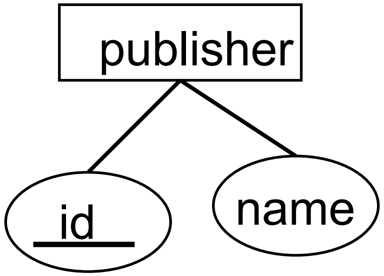
Exercise (2/7)

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.



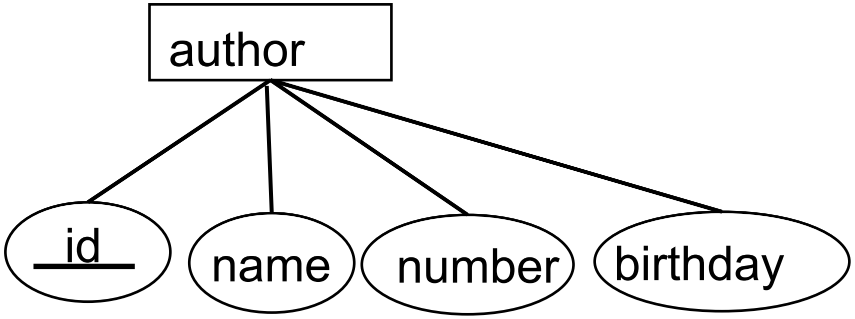
Exercise (3/7)

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



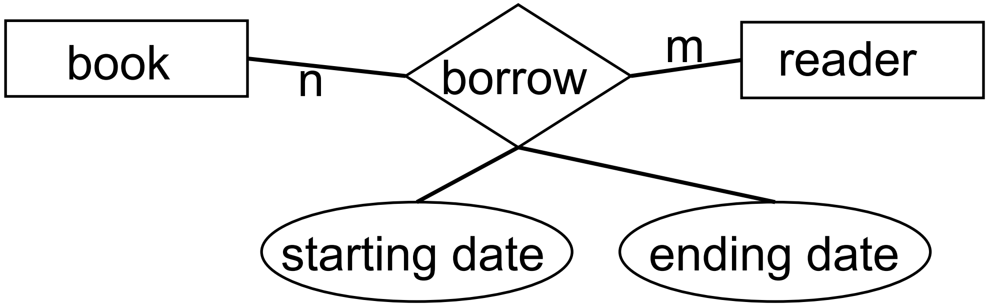
Exercise (4/7)

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



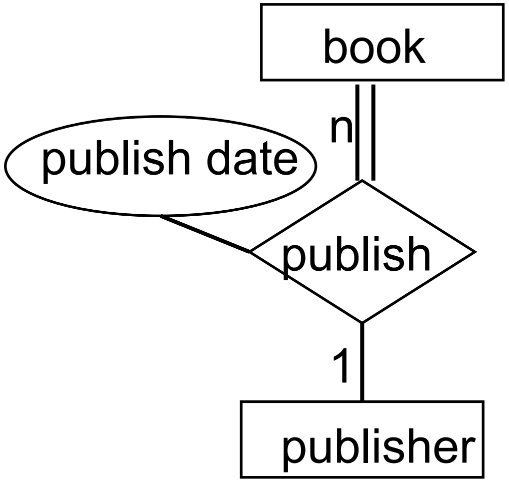
Exercise (5/7)

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.



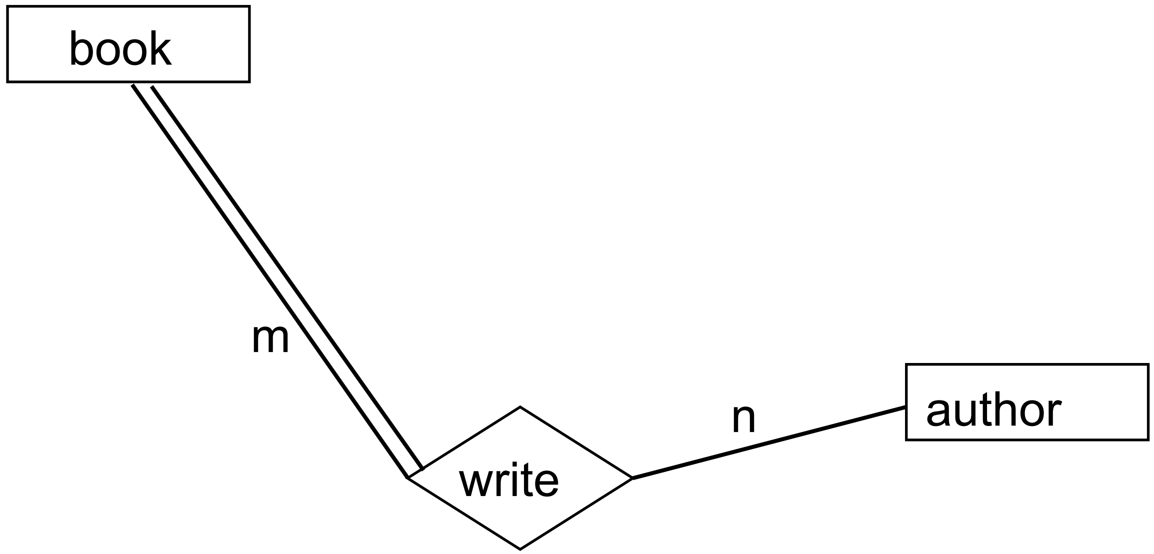
Exercise (6/7)

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.

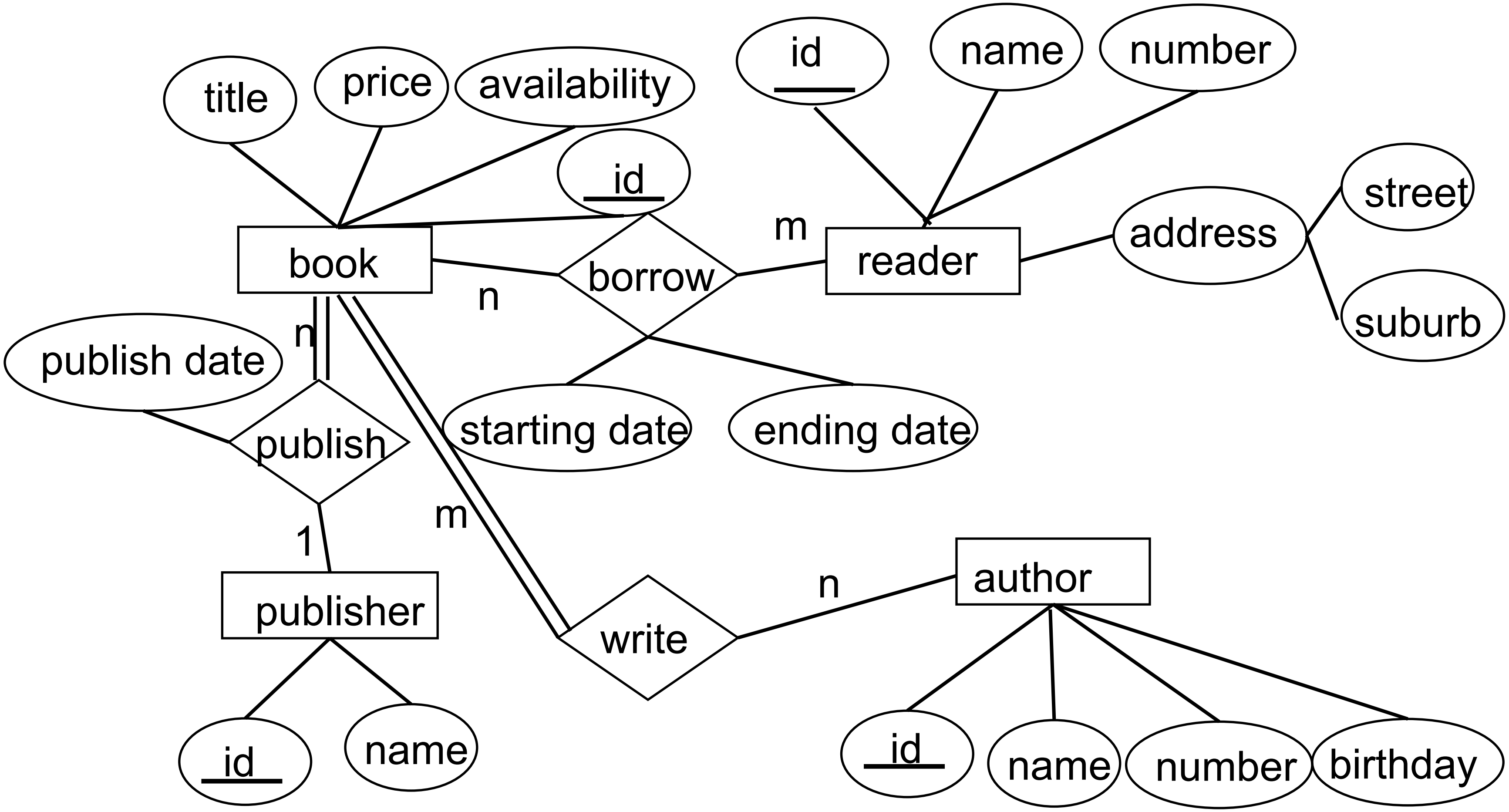


Exercise (7/7)

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



Full Sample Solution



Learning Outcome

- Given application requirements, how to use ER-diagram to accurately reflect the application (and all the specific requirements).
- Understand ER model as a solution for expressing any application requirement.

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