>>

ER Model

^ >>

❖ Entity-Relationship Data Modelling

The world is viewed as a collection of inter-related entities.

ER has three major modelling constructs:

- attribute: data item describing a property of interest
- entity: collection of attributes describing object of interest
- relationship: association between entities (objects)

The ER model is not a standard, so notational variations exist

Lecture notes use notation from SKS and GUW books (simple)

<< ^ >>

❖ 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

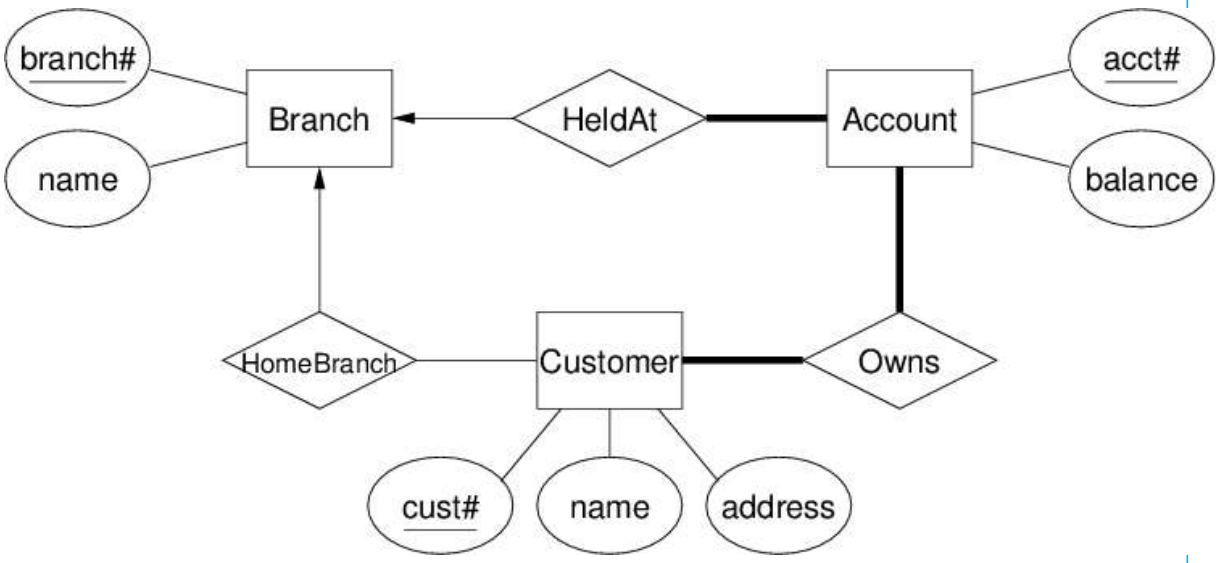
Terminology abuse:

- we say "entity" when we mean "entity set"
- we say "relationship" when we mean "relationship sets"
- we say "entity instance" ro refer to a particular entity

<< ^ >>

❖ Entity-Relationship (ER) Diagrams (cont)

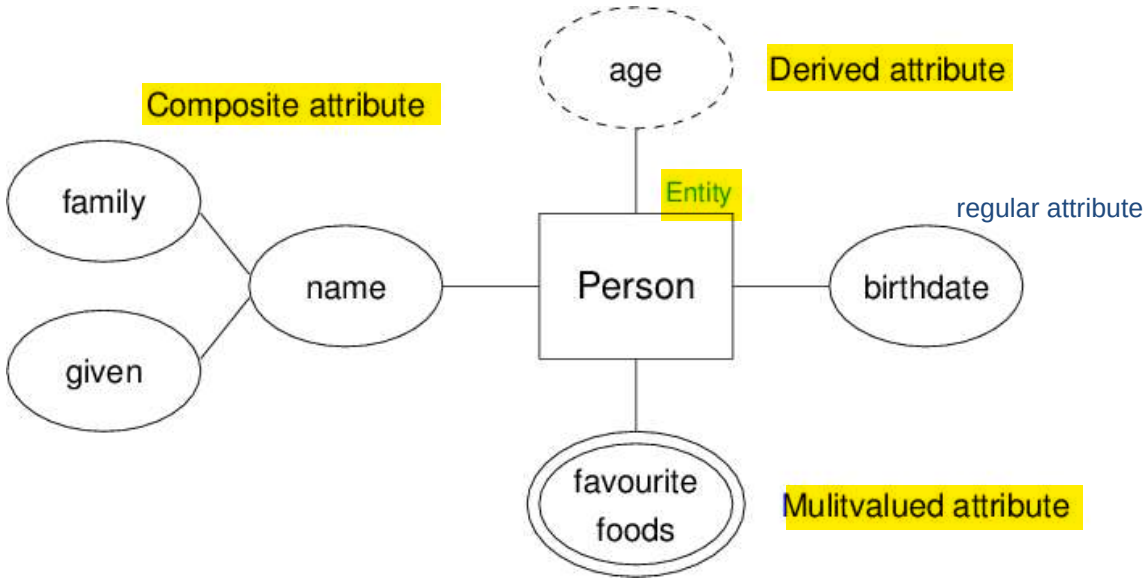
Example ER diagram:



<< ^ >>

❖ Entity-Relationship (ER) Diagrams (cont)

Example of attribute notations:



<< ^ >>

❖ Entity Sets

An entity set can be viewed as either:

- a set of entities with the same set of attributes (extensional)
- an abstract description of a class of entities (intensional)

Key (superkey): any set of attributes

- whose set of values are distinct over entity set
- natural (e.g., name+address+birthday) or artificial (e.g., SSN)

everyone has these by default

these are generated constructs

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

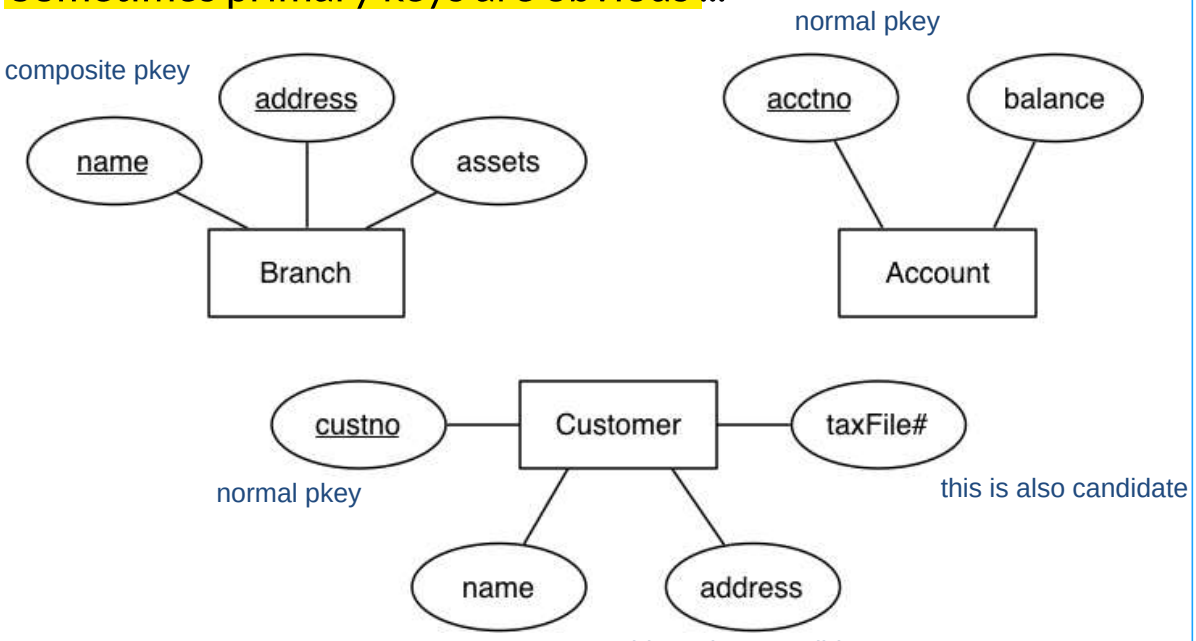
smallest set of attributes to uniquely identify the entity

Primary key = candidate key chosen by DB designer

Keys are indicated in ER diagrams by underlining

❖ Keys

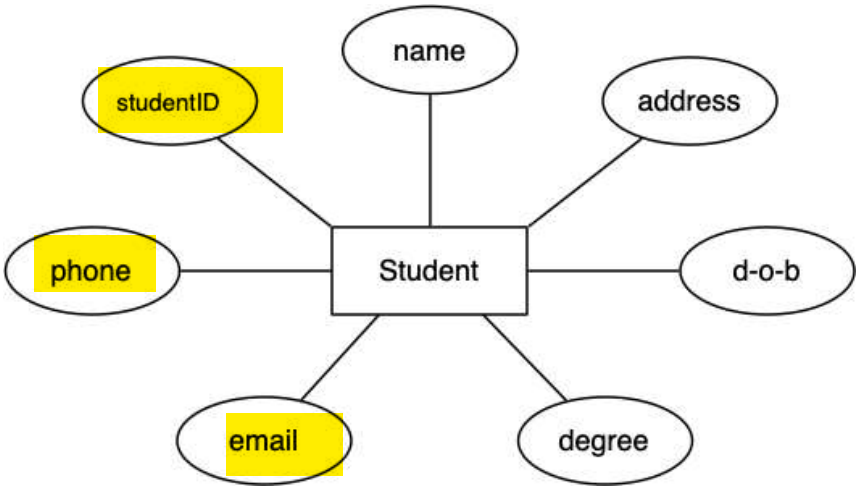
Sometimes primary keys are obvious ...



<< ^ >>

❖ Example: Identifying Keys

Candidate keys in the following ER diagram ...



Possibilities: {studentID}, {phone}, {email}, {name,address,d-o-b}?

realistically you would choose the studentID, that is the whole purpose of it after all, to be a unique identifier.

<< ^ >>

❖ Relationship Sets

Relationship: an association among several entities

- e.g., Customer(9876) is the owner of Account(12345)

Relationship set: collection of relationships of the same type

Degree = # entities involved in reln (in ER model, ≥ *2*)

number of entity TYPES

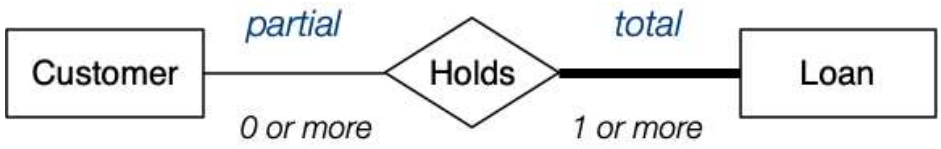
Cardinality = # associated entities on each side of reln

0, 1, many number of entity INSTANCES

Participation = must every entity be in the relationship

impossible for it to exist without being part of this relationship e.g. child must have parents.

Example: relationship participation

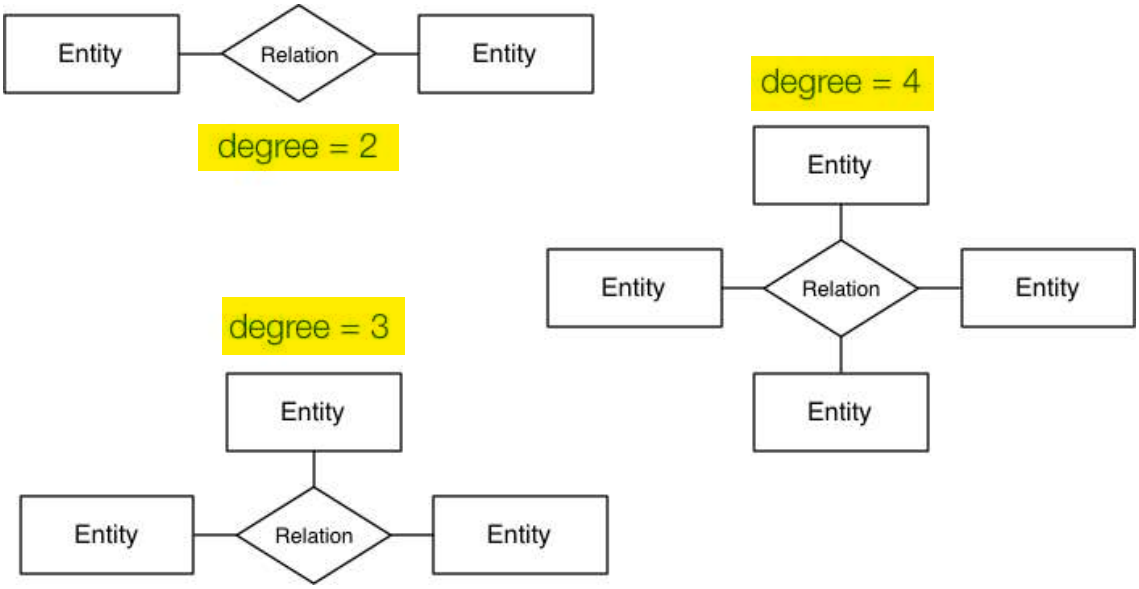


<< ^ >>

❖ Relationship Sets (cont)

Examples: relationship degree

degree: number of entity TYPES involved in the relationship.



<< ^ >>

❖ Relationship Sets (cont)

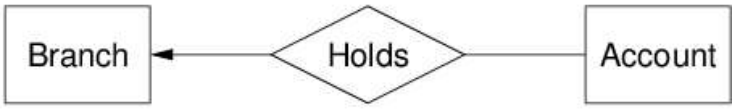
Examples: relationship cardinality

Cardinality: the number of instances of the entity type.
Note that the degree is 2 in all of these examples, but the cardinality differs.3

one–to–one

a single manager can manage at most 1 branch,
a single branch can be managed by at most 1 manager.

one–to–many



A single branch can hold many accounts,
a single account can only be held at 1 branch.

many–to–many

a single customer can own many accounts,
a single account can be owned by many customers.

Notes;
- a plain line means an unlimited maximum, an arrow means a maximum of 1.
- when reading you ignore the first arrow i.e. reading left-to-right you look at the right side arrow, right-to-left you look at the left side arrow.

<< ^ >>

❖ Example: Relationship Semantics

Semantics of the following relationships ...

A lecturer may teach one course
Every course is taught by 1 or more lecturers

Every lecturer teaches one or more courses
Every course is taught by 1 or more lecturers

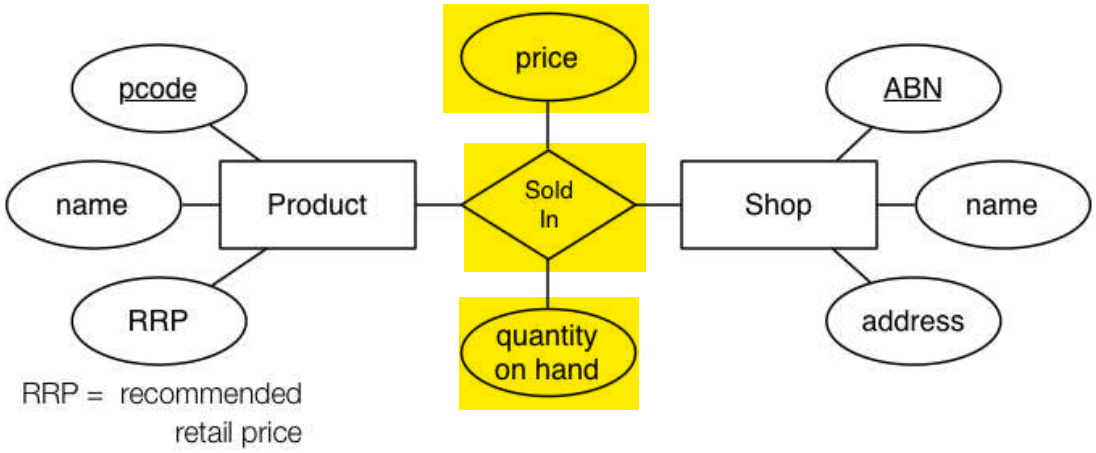
Every lecturer may teach one course
Every course is taught by one lecturer

Notes;
- a bold line enforces a minimum of 1 i.e. an instance of the entity the bold line is connected to must participate in that relationship.
- a regular line doesn't enforce a minimum i.e. an instance of the entity connected by a regular line doesn't necessarily participate in that relationship.
- if you assert one side exists, then by the nature of relationships, the other side has to exist. For this reason, you can ignore the lower bound for the other side and just look at the upper bound in this context. For example, in scenario 1 above, not all lecturers have to teach a course, but if we assume they do, then we are guaranteed that a course exists in this case otherwise the relationship wouldn't be possible.
- when drawing a conclusion for an entity in a relationship, you look at its own line to determine whether it itself must participate (its minimum), then you check whether the other entities line is an arrow or plain line to see if it is 1 or many. This follows the pattern: "A foo [MAY/MUST] bar [ONE/ONE OR MORE] foobar".
- another way of wording the above is: boldness only matters to the owning entity, arrowness only matters to the other entitiy.

<< ^ >>

❖ Example: Relationship Semantics (cont)

In some cases, a relationship needs associated attributes.



(Price and quantity are related to products in a particular shop)

That is, they are not the price and quantity of that item in general, but specific to a certain shop.

If you put price on the product, then every shop would have to sell it for the same price. The only price that makes sense to be on the product is the RRP and not the actual price because shops want to sell for different prices.

Likewise, if you put the quantity of a product on the product itself, then yes you would still know the total amount of that product left in the world, but that is not very useful, it is better to know how much is in each specific shop (then you can say out of stock for specific shops) - furthermore, total stock could be derived from the on-relationship version, but stock per shop cannot be derived from the on-entity version.

Note that you also wouldn't put quantity-on-hand on the shop, unless they only sell 1 thing (even then, this is very rigid and not extensible/flexible as it doesn't account for the possibility of ever selling anything else).

SO WHAT IS A GOOD GENERAL RULE?
An attribute needs to go on a relationship rather than one of the involved entities if the attribute is contextual to at least one of the entities it wouldn't otherwise go on. For example, the reason price and q.o.h need to go on the relationship is because price is contextual to shop (because different shops want to sell it for different prices) and it would have otherwise went on product. If it only pertains to one of the involved entities (e.g. the pcode of the product only matters to the product), then it can just stay on that entity.
So, place it where you think it should go, i.e. you would initially place price on shop, but then determine whether it is contextual to any of the other entities involved (in this case the only other entity involved is the shop and it is contextual to shop because they would want to sell at differing prices, hence move price to relationship).

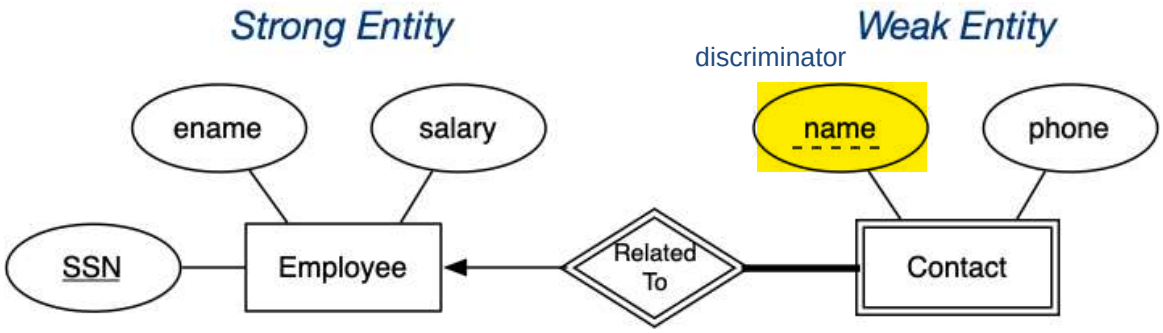
<< ^ >>

❖ Weak Entity Sets

Weak entities

- exist only because of association with strong entities.
- have no key of their own; have a discriminator

Example:



Won't see these much.

<< ^ >>

❖ Subclasses and Inheritance

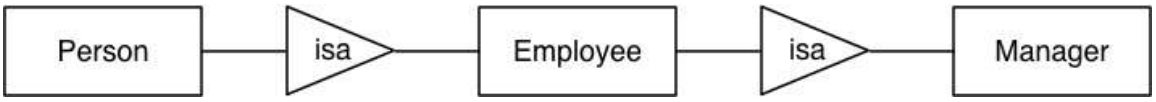
A subclass of an entity set *A* is a set of entities:

- with all attributes of *A*, plus (usually) it own attributes
- that is involved in all of *A*'s relationships, plus its own

Properties of subclasses:

- overlapping or disjoint (can an entity be in multiple subclasses?)
  i.e. can a superclass instance have multiple related subclass instances?
- total or partial (does every entity have to also be in a subclass?)
  i.e. is there at least one instance of one of the subclasses related to each of the superclass instances?

Special case: entity has one subclass ("B is-a A" specialisation)

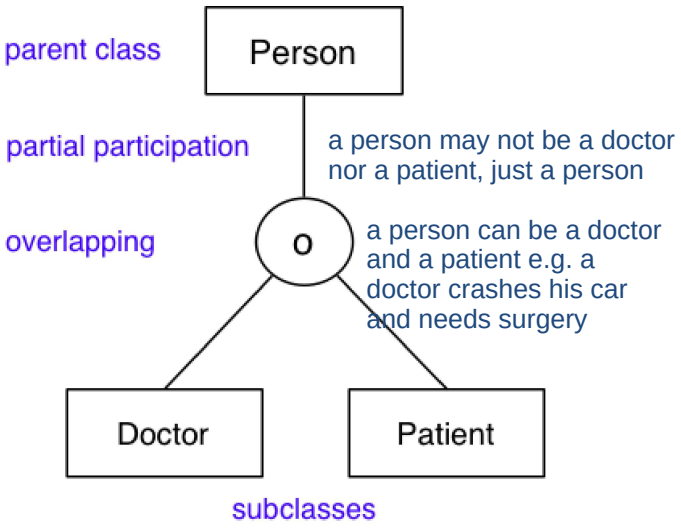


<< ^ >>

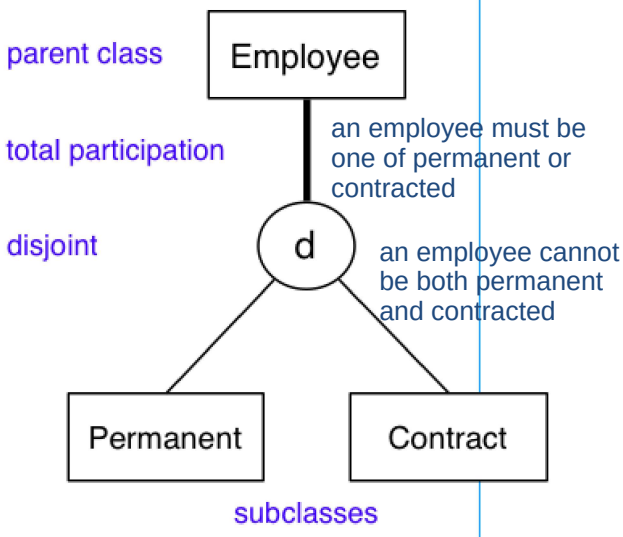
❖ Subclasses and Inheritance (cont)

Example:

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



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



<< ^ >>

❖ Design Using the ER Model

ER model: simple, powerful set of data modelling tools

Some considerations in designing ER models:

- should an "object" be represented by an attribute or entity?
  does it need to be shared? If yes, make it an entity.
  does it need its own attributes? if yes, most likely make it an entity.
- is a "concept" best expressed as an entity or relationship?
  is it a verb (rel) or a noun (entity)? is it modelling the result/side-effect of an operation (rel) or not (entity)?
- should we use *n*-way relationship or several 2-way relationships?
- is an "object" a strong or weak entity? (usually strong)
- are there subclasses/superclasses within the entities?

Answers to above are worked out by *thinking* about the application domain.

<< ^ >>

❖ Large ER Diagrams

ER diagrams are typically too large to fit on a single screen
(or a single sheet of paper, if printing)

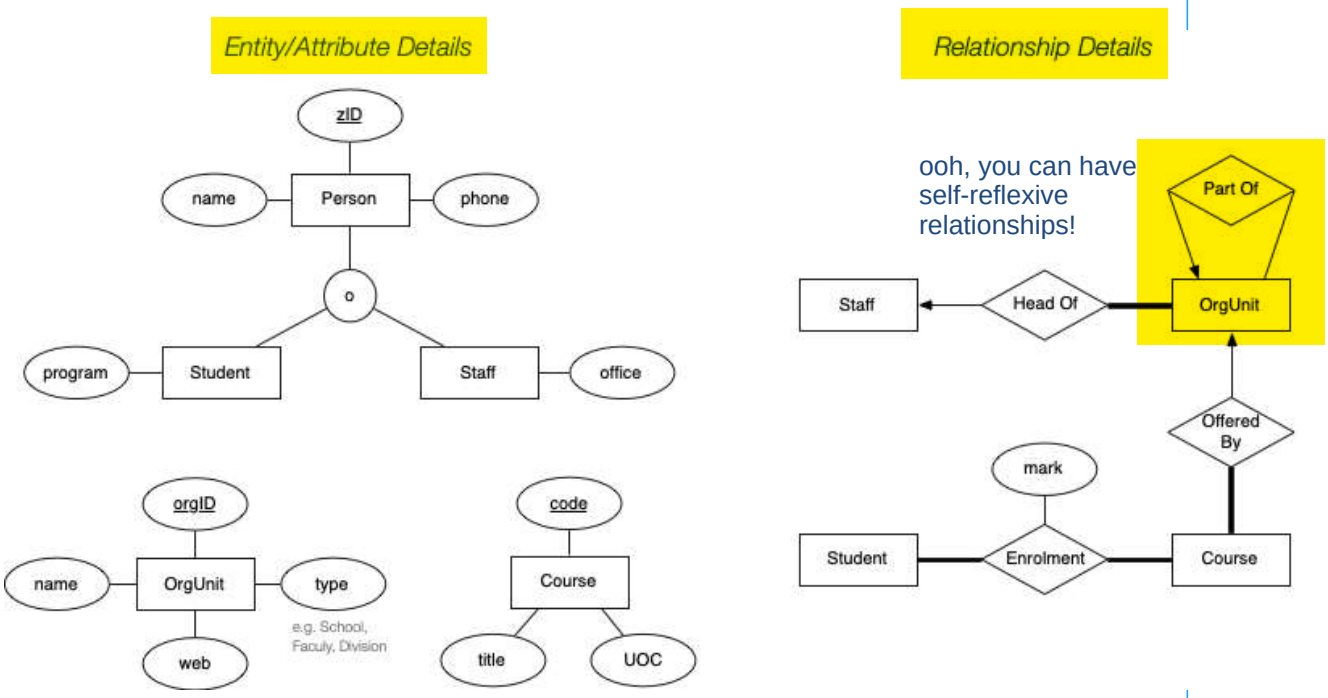
One commonly used strategy:

- define entity sets separately, showing attributes
- combine entitities and relationships on a single diagram
  (but without showing entity attributes)
- if very large design, may use several linked diagrams

<< ^ >>

❖ Large ER Diagrams (cont)

Example of drawing large ER diagram:



<< ^

❖ Summary of ER

ER model is popular for doing conceptual design

- high-level, models relatively easy to understand
- good expressive power, can capture many details

Basic constructs: entities, relationships, attributes

Relationship constraints: total / partial, n:m / 1:n / 1:1

Other constructs: inheritance hierarchies, weak entities

Many notational variants of ER exist
(especially in the expression of constraints on relationships)

Produced: 13 Sep 2020