Weak Entities, the EER Model and Design Choices

Table of contents

Outline 1
Weak entities 1
Question — Weak Entity (hotel/room) 4
The EER model — superclasses and subclasses 4
Exercise — University database (UofU) 6
Design choices for ER conceptual design 9

Module 1, part 2 — continues on from 2026-02-23-conceptual-database-design-and-the-er-model[1]. See er-diagram-notation[2] and relationship-constraints[3] for the reference symbol legend and weak-entity definition.

Outline

- Weak entities (closing out the "Entities and Relationships" section)
- The Enhanced ER (EER) diagram — superclasses/subclasses
- Design choices for conceptual modelling

Weak entities

Entity types that do not have key attributes of their own are called weak entities. Two rows of data show why: an employer insures dependants, and two different employees (`SSN` 12345 and `SSN` 22336) each have a dependant named John. `Pname` on its own cannot identify a dependant; `Pname` *together with* the insuring employee's `SSN` can.

[1] courses/infs1200/2026-02-23-conceptual-database-design-and-the-er-model.pdf
[2] courses/infs1200/er-diagram-notation.pdf
[3] courses/infs1200/relationship-constraints.pdf

That gives the four terms:

- A weak entity **can be identified uniquely only by considering the primary key of another owner entity and its own partial key**, which is underlined with a dotted line.
- The relationship type that relates a weak entity to its owner is the **identifying relationship**.
- **A weak entity and its identifying relationship are represented with double lines.**
- The weak entity always has **total participation** in the identifying relationship — it cannot exist without an owner.

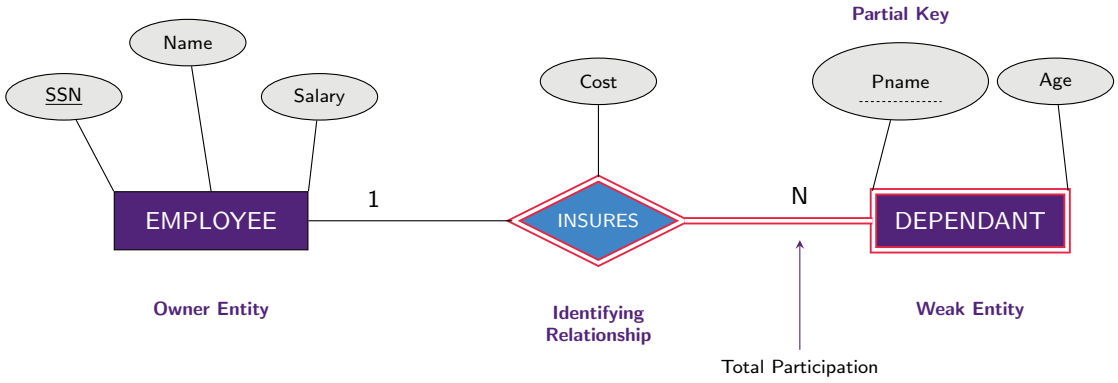


Figure 1: A weak entity and its owner. `DEPENDANT` has no key of its own: `Pname` is only a *partial* key (dotted underline), so its real key is `(EMPLOYEE.SSN, DEPENDANT.Pname)`. `INSURES` is the identifying relationship and carries its own descriptive attribute `Cost`. Here the identifying relationship is filled blue and the weak-entity/identifying double lines are crimson; the course's notation guide draws the same symbols as a magenta diamond and black double lines.

A weak entity can have more than one owner

The identifying relationship does not have to be binary. In this example it is ternary: a `CANDIDATE` applies to a `COMPANY`, and the `INTERVIEW` that results is identified only by *both* of them together plus its own partial key `Details` (itself composite — `Department` + `Date`).

This matters downstream: a weak entity with two owners takes a foreign key to *each* of them, and its primary key is the union of both owner keys and its partial key — see er-to-relational-mapping[4].

[4] courses/infs1200/er-to-relational-mapping.pdf

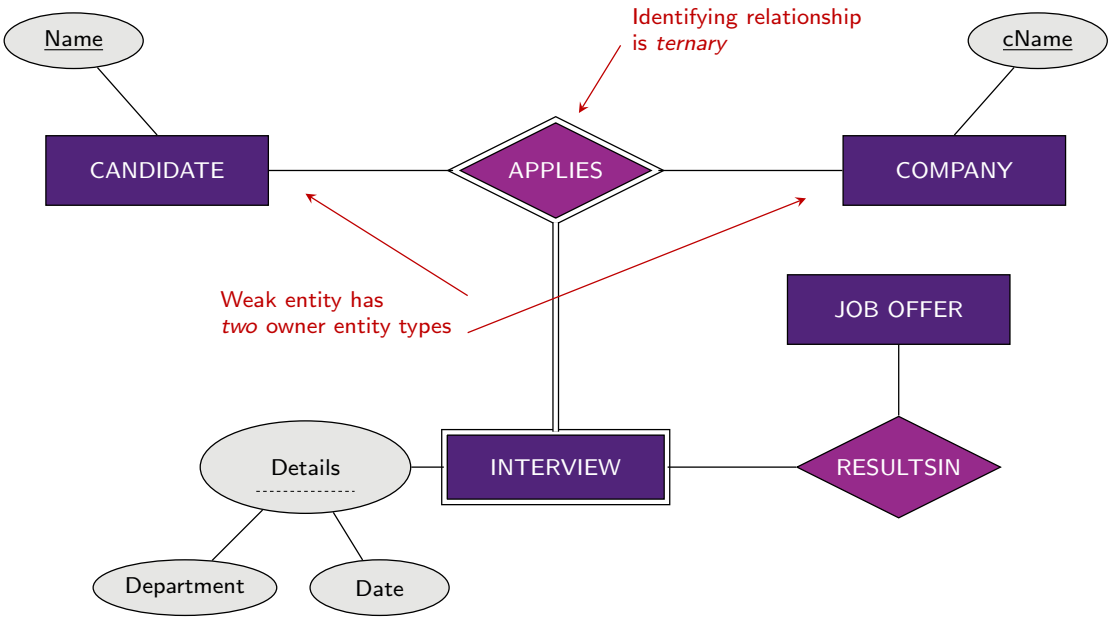


Figure 2: A weak entity with two owners. `INTERVIEW` is a weak entity with **two** owner entity types, so its identifying relationship `APPLIES` is ternary and its full key is `(CANDIDATE.Name, COMPANY.cName, Details)`. Because `INTERVIEW` is an entity rather than an attribute, it can be the endpoint of a further relationship — `RESULTSIN` to `JOB OFFER`.

Question — Weak Entity (hotel/room)

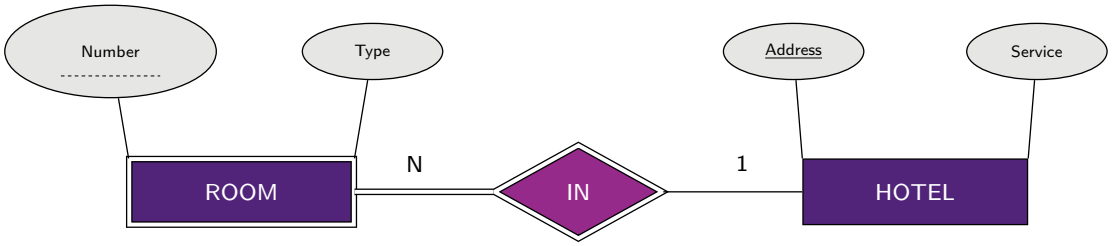


Figure 3: `ROOM` and `HOTEL` joined by the identifying relationship `IN`. `ROOM` is weak: `Number` carries a dotted underline (partial key), while `Address` carries a solid one (key). Everything the four options claim turns on telling those two underline styles apart.

Which of the following is true? **(A)** Two hotels can have the same address. **(B)** No two rooms can have the same number. **(C)** No two hotels can have rooms with the same number. **(D)** No two rooms with the same number can have the same type. **(E)** None of the above.

> **i** Solution
>
> **(E) None of the above.** `ROOM`'s real key is the composite `(HOTEL.Address, ROOM.Number)` — a room number is only guaranteed unique *within* its owning hotel, so (B) and (C) are both false; (A) is false since `Address` is `HOTEL`'s key attribute and must therefore be unique per hotel; (D) doesn't follow from any stated constraint, since nothing at all is asserted about `Type`.

The EER model — superclasses and subclasses

An entity type is called a **class** in the EER model. Entities in the same class have the same attributes; a class can be a **superclass** or a **subclass**:

- Attributes of a superclass are **inherited** by the subclasses.
- A subclass can have its own specific attributes.
- A subclass can have its own specific **relationships**.
- Every entity in a subclass is a member of its superclass(es).

The notation is a drawn as an open arc on the connecting line, and **the direction of the is what says which is which**: in `E1` `E2`, `E1` is the subclass of `E2`.

Motivating example: a supermarket `ITEM` has a `ProductName` and a `Price` in common with everything else on the shelves; a `FOOD` item additionally has an `ExpiryDate`. `ITEM` is the

superclass — it captures the data common to a variety of different objects — and `FOOD`, an extension of a regular item, is the subclass.

Two ways of arriving at the same picture:

- **Specialisation** (top-down): define a number of subclasses of an entity type. Each subclass contains a *subset* of the entities of the superclass, and is defined by a more specific distinguishing characteristic of those entities.
- **Generalisation** (bottom-up): abstraction — ignore the differences amongst some subclasses and generalise them into a superclass.

In summary: subclasses are a specialisation of the superclass, and the superclass is a generalisation of the subclasses. See specialisation-and-generalisation[5].

Constraints on specialisation

Two independent choices, and one diagram puts all four combinations together by giving `EMPLOYEE` *two* specialisations at once:

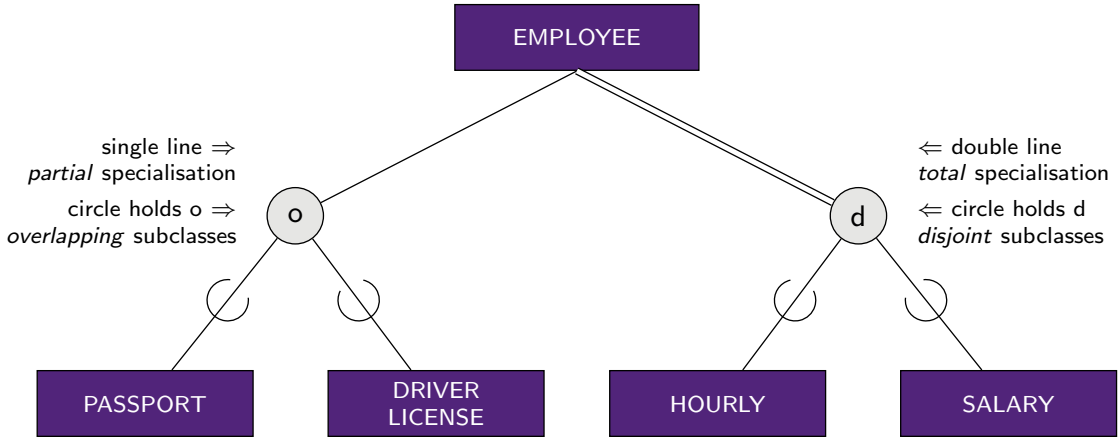


Figure 4: Constraints on specialisation. `EMPLOYEE` carries two independent specialisations: a **partial, overlapping** one (single line into an `o` circle — an employee may hold a passport, a driver licence, both, or neither) and a **total, disjoint** one (double line into a `d` circle — every employee is either hourly-paid or salaried, and never both). Line style fixes total vs partial; the letter in the circle fixes disjoint vs overlapping. They are orthogonal: any of the four combinations is legal.

[5] courses/infs1200/specialisation-and-generalisation.pdf

Choice	Drawn as	Asserts
Total specialisation	Double line from superclass to the circle	Every superclass entity belongs to at least one subclass
Partial specialisation	Single line	A superclass entity need not belong to any subclass
Disjoint subclass set	`d` in the circle	An entity belongs to at most one of the subclasses
Overlapping subclass set	`o` in the circle	An entity may belong to more than one subclass at once

The `EMPLOYEE` → `d` → `SECRETARY`/`ENGINEER` split on the module's overview diagram (in 2026-02-23-conceptual-database-design-and-the-er-model[6]) is the same apparatus, with the subclasses carrying their own attributes — `TypingSpeed` on `SECRETARY`, `EngineerDetails` on `ENGINEER`.

Why bother with a subclass at all?

Subclasses let an attribute be stored for *some* entity instances rather than all of them, which reduces the number of null/empty values. In the event system of Applied class 2, not every `SHOW` needs a `Rating` — only `ENTERTAINMENT` shows do — so putting `Rating` on the subclass rather than the superclass is a storage-efficiency argument as well as a modelling one. Compare the three reasons a null appears in entity-types-and-attributes[7].

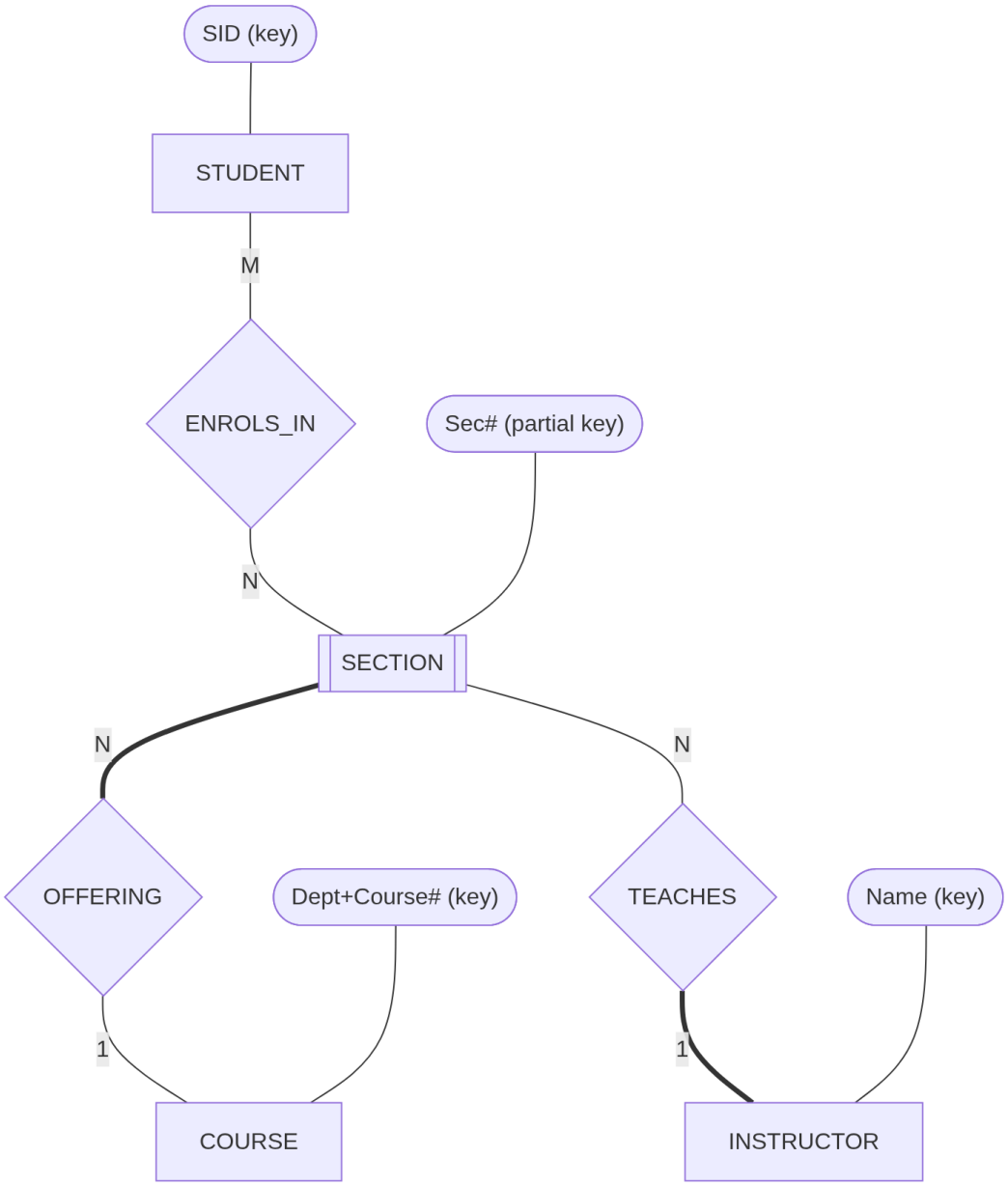
Exercise — University database (UofU)

Students: unique student id, name, address, phone, registered in a major; visiting students stay for a year. Courses: identified by department + course#, with title and credits. A course may be offered many times, even in the same semester; each offering gets a section# unique for a given course and semester, and is taught by a single instructor. Every instructor is responsible for some section (no idle instructors); instructors have unique names and a higher degree recorded. A student enrols in a course section and gets a mark. A course may have any number of other courses as prerequisites.

[6] courses/infs1200/2026-02-23-conceptual-database-design-and-the-er-model.pdf
[7] courses/infs1200/entity-types-and-attributes.pdf

Solution

STUDENT (key SID, Name, Address, Phone, Major) —M:N ENROLS_IN→ SECTION (attribute Mark); STUDENT has a subclass VISITING_STUDENT (HomeInst, StartDate). SECTION is a **weak entity** (partial key Sec#, owner COURSE, composite key Sec# + Semester), —N:1 OFFERING→ COURSE (key Dept + Course#, Title, Credits); SECTION —N:1 TEACHES← INSTRUCTOR (key Name, Degree), with total participation on the INSTRUCTOR side ("no idle instructors"). COURSE has a recursive M:N PREREQUISITE relationship with itself (roles *higher*/*requires*).



Two clauses do real work. "Section# is unique for a given course and semester" makes `SECTION` weak with `COURSE` as its owner — a section number means nothing without the course it belongs to. "There are no idle instructors" is a *participation* constraint, not a cardinality one: it makes `INSTRUCTOR` total in `TEACHES` without changing the N:1 ratio. "Visiting students stay for a year" is the distinguishing characteristic that earns `VISITING_STUDENT` its subclass.

Design choices for ER conceptual design

When modelling complex organisations a database designer faces design choices, which fall into two categories:

- **Equivalent choices** — an aspect of the UoD can be represented in an ER diagram by two different methods which are both equivalent, i.e. the resulting database will be the same.
- **Inequivalent choices** — when a UoD is *slightly ambiguous*, an aspect of it can be mapped two different ways which both meet the specification but lead to **different limitations/constraints being enforced by the database**. Note the assumption you made under the diagram.

The four common choice points, each with a worked before/after pair of diagrams in er-design-choices[8]:

- **Attribute vs. (weak) entity type** — an interview's `Details` (`Department`, `Date`) as a key attribute of the `APPLIES` relationship, vs. pulled out into the weak `INTERVIEW` entity of

    1.

- **Attribute vs. subclass** — `EMPLOYEE.Type` as a plain attribute, vs. splitting into `SECRETARY`/`ENGINEER` subclasses with their own extra attributes.
- **Subclass relationship vs. superclass relationship** — giving `WORKSON` (to `PROJECT`) to the `EMPLOYEE` superclass vs. only to `ENGINEER`.
- **Binary relationships vs. N-ary relationships** — `SUPPLIER`/`PROJECT`/`PART` as three binaries (`SUPPLIES`, `CANSUPPLY`, `USES`) vs. one ternary `SUPPLIES` carrying a `quantity`.

[8] courses/infs1200/er-design-choices.pdf