Weak Entities, the EER Model and Design Choices

Table of contents

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 this lecture uses.

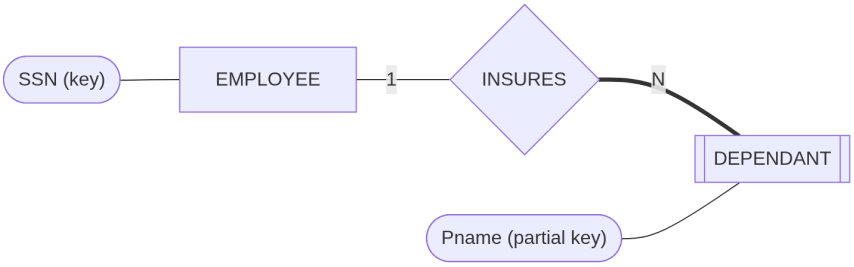
Today's outline

- Weak entities
- The Enhanced ER (EER) diagram — superclasses/subclasses
- Design choices for conceptual modelling

Weak entities

See relationship-constraints[4]#weak-entities for the definition (owner entity, partial key, identifying relationship, total participation). Example: `EMPLOYEE` —1:N `INSURES`→ `DEPENDANT`, where `DEPENDANT`'s partial key is `Pname` and its full key is `(EMPLOYEE.SSN, DEPENDANT.Pname)`.

[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
[4] courses/infs1200/relationship-constraints.pdf

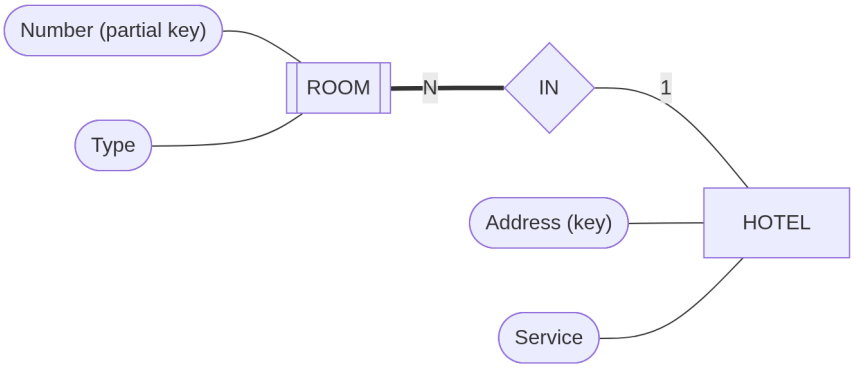


Question — Weak Entity (hotel/room)

`ROOM` (partial key `Number`, attribute `Type`) is a weak entity, N:1 `IN`→ `HOTEL` (key `Address`, attribute `Service`). Which is true: (A) two hotels can share an address, (B) no two rooms share a number, (C) no two hotels have rooms with the same number, (D) no two same-numbered rooms share a type, (E) none of the above?

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 (must be unique per hotel); (D) doesn't follow from any stated constraint.



The EER model — superclasses and subclasses

An entity type is called a **class** in the EER model. Entities in the same class share the same attributes; a class can be a **superclass** or **subclass** — a subclass inherits its superclass's attributes/ relationships, and can also have its own specific attributes/relationships. Every entity in a subclass is a member of its superclass(es).

Motivating example: a supermarket `ITEM` (superclass: `ProductName`, `Price`) vs a `FOOD` item (subclass: adds `ExpiryDate`) — `FOOD` is just an extension of a regular `ITEM`.

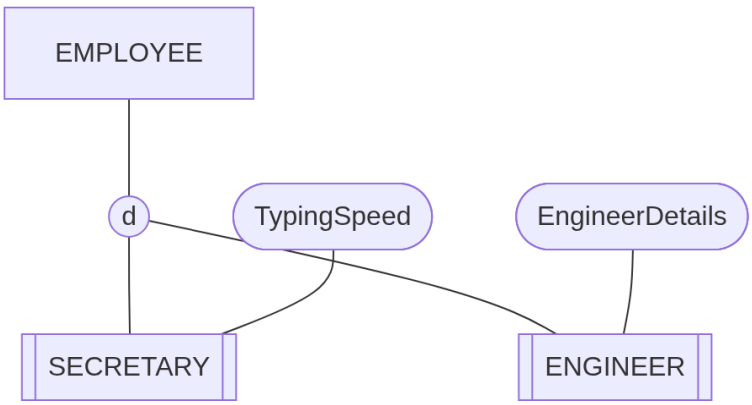
- **Specialisation**: define subclasses of an entity type based on a more specific distinguishing characteristic (top-down).
- **Generalisation**: abstract away differences between several existing entity types to identify a common superclass (bottom-up).
- Subclasses are a specialisation of the superclass; the superclass is a generalisation of the subclasses.

Constraints on specialisation

See er-diagram-notation[5]#subclasses-superclasses-eer for the notation.

- **Total vs partial**: whether every superclass instance must belong to some subclass, or not.
- **Disjoint (`d`) vs overlapping (`o`)**: whether an instance can belong to at most one subclass, or more than one at once.

Example: `EMPLOYEE` splits disjointly (`d`) into `SECRETARY`/`ENGINEER` (each with their own attribute — `TypingSpeed`/`EngineerDetails`), and separately (still under `EMPLOYEE`) an overlapping (`o`) split into `DRIVER`/`PASSPORT`-holder isn't required to be disjoint or total depending on the UoD.



Exercise — University database (UofU)

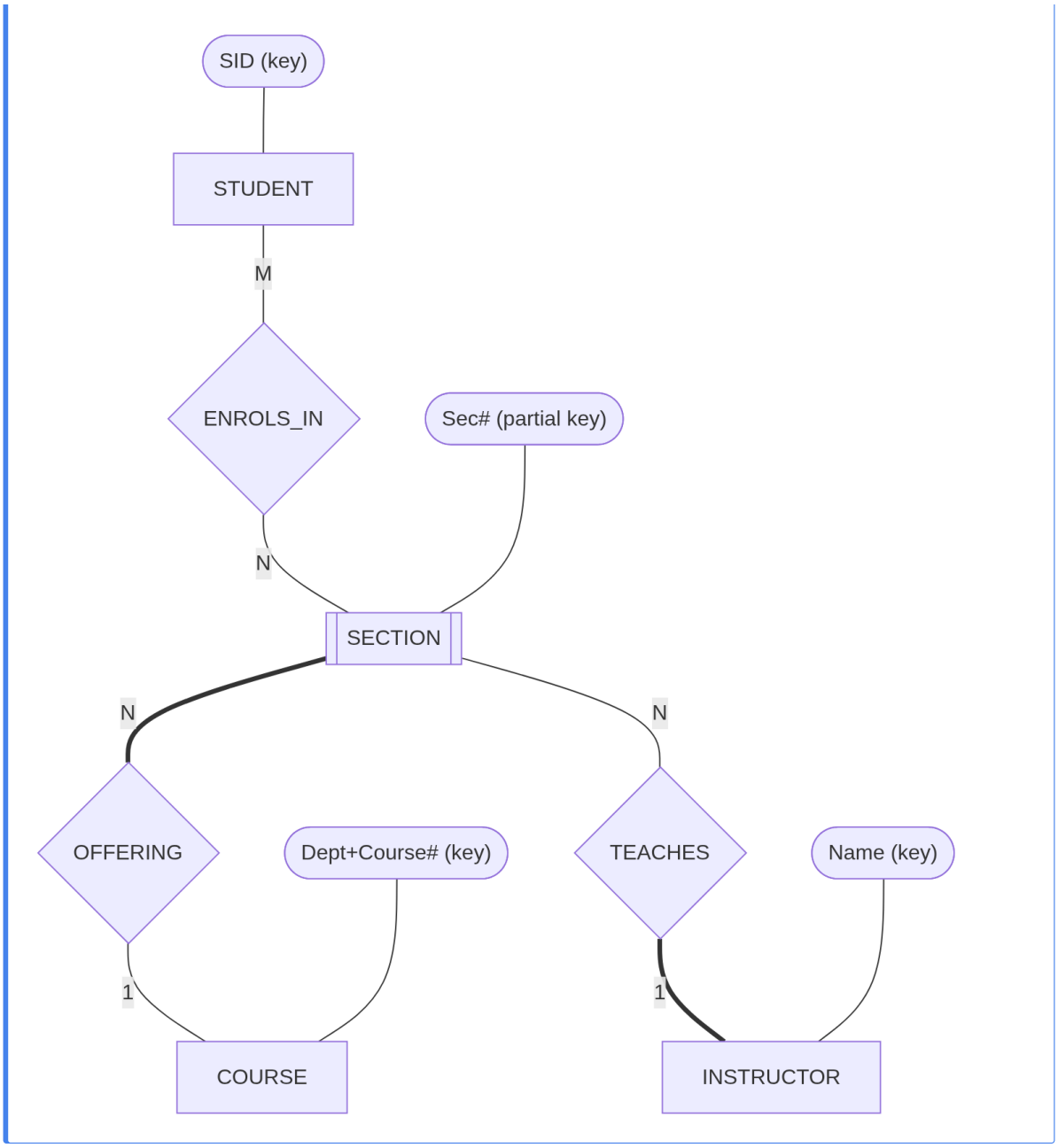
Students: unique student id, name, address, phone, registered major; visiting students stay for a year. Courses: identified by department + course#, with title and credits. Course sections:

[5] courses/infs1200/er-diagram-notation.pdf

unique section# per course/semester, taught by exactly one instructor (no idle instructors); instructors have a unique name and a higher degree recorded. Students enrol in a section and get a mark. A course may have other courses as prerequisites.

i 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*).



Design choices for ER conceptual design

Modelling the same UoD can involve genuine design choices:

- **Equivalent choices** — two ER representations that produce the same resulting database.

- **Inequivalent choices** — the UoD is ambiguous, and different mappings meet the spec but enforce different constraints; note your assumption under the diagram when this happens.

Common choice points:

- **Attribute vs. (weak) entity type** — e.g. an interview's `Details` (`Department`, `Date`) can be modelled as attributes of the `APPLIES` relationship, or pulled out into its own weak `INTERVIEW` entity.
- **Attribute vs. subclass** — e.g. `EMPLOYEE.Type` as a plain attribute, vs. splitting into `SECRETARY`/`ENGINEER` subclasses with their own extra attributes.
- **Binary vs. n-ary relationships** — e.g. `SUPPLIER`/`PROJECT`/`PART` as three binary relationships (`CANSUPPLY`, `USES`, `SUPPLIES`) vs. one ternary `SUPPLIES` relationship (with a `quantity` attribute) linking all three at once.
- **Subclass relationship vs. superclass relationship** — e.g. giving `WORKSON` (a `PROJECT` relationship) directly to the `EMPLOYEE` superclass vs. only to specific subclasses like `ENGINEER`.