ER Design Choices

Table of contents

Equivalent vs inequivalent choices 1
The four common choice points 2
Worked example: attribute vs weak entity type 3
Worked example: binary vs n-ary relationships 4
Worked example: subclass vs superclass relationship 6
Why the choice matters after the diagram 8

When modelling complex organisations, a database designer faces design choices: two people modelling the same Universe of Discourse will not draw the same diagram, and that isn't necessarily a sign either of them is wrong. Introduced in 2026-03-02-weak-entities-the-eer-model-and-design-choices[1].

Equivalent vs inequivalent choices

- **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. Genuinely a matter of taste; pick either.
- **Inequivalent choices** — the UoD is *slightly ambiguous*, and an aspect of it can be mapped two different ways which both meet the specification of the UoD but lead to **different limitations/constraints being enforced by the database**. Here the choice has consequences, and the right move is to pick one and **note the assumption under the diagram**.

That second case is the reason a diagram is rarely a complete answer on its own. The medical-clinic model in week3-tutorial-applied-class-2-entity-relationship-diagrams[2] carries the explicit assumptions "every patient in the DB has made an appointment" and "every clinic has at least one doctor" — neither is stated in the brief, both change the participation constraints, so both get written down rather than smuggled into the drawing.

[1] courses/infs1200/2026-03-02-weak-entities-the-eer-model-and-design-choices.pdf
[2] courses/infs1200/week3-tutorial-applied-class-2-entity-relationship-diagrams.pdf

The four common choice points

Choice	The question	Worked example
Attribute vs (weak) entity type	Is this a fact *about* something, or a thing in its own right?	An interview's `Details` (`Department`, `Date`) as a key attribute of the `APPLIES` relationship, vs. a separate weak `INTERVIEW` entity
Attribute vs subclass	Is this a value, or a kind?	`EMPLOYEE.Type` as a plain attribute, vs. splitting into `SECRETARY` / `ENGINEER` subclasses
Subclass relationship vs superclass relationship	Does this relationship apply to everyone, or only to one kind?	Giving `WORKSON` (to `PROJECT`) to the `EMPLOYEE` superclass, vs. only to `ENGINEER`
Binary relationships vs n-ary relationships	Does the association hold between pairs, or genuinely among all three at once?	`SUPPLIER` / `PROJECT` / `PART` as three binaries (`SUPPLIES`, `CANSUPPLY`, `USES`) vs. one ternary `SUPPLIES` carrying a `quantity`

Each row is a real fork, not a stylistic one:

- **Attribute vs entity type.** Pulling something out into its own entity type gives it identity — it can then be referred to, counted, and given relationships of its own. The cinema model in the same applied class makes exactly this call about a `VIEWING`: it can be read as a weak entity owned by `CUSTOMER` + `MOVIE` with `Timestamp` as its partial key, or as an M:N relationship carrying `Timestamp` as a key attribute. Both fit the brief; they differ in whether a viewing is a *thing* or a *link*. (interpreting-er-diagrams[3] shows the reading side of the same call: without a distinguishing attribute, a plain M:N pair can only be recorded once.)
- **Attribute vs subclass.** A subclass earns its place when the subtypes have *different attributes or relationships* — `SECRETARY` recording a `TypingSpeed` that means nothing for an engineer. If the only difference is the label itself, a plain attribute says the same thing with less diagram. Applied class 2 gives the efficiency half of the argument explicitly: subclasses let a data attribute be stored for *some* entity instances rather than all of them, and so **reduce the number of null/empty values** — not every `SHOW` needs a `Rating`, only `ENTERTAINMENT` shows do. See specialisation-and-generalisation[4].

[3] courses/infs1200/interpreting-er-diagrams.pdf
[4] courses/infs1200/specialisation-and-generalisation.pdf

- **Binary vs n-ary.** Three binary relationships and one ternary relationship are not interchangeable: the binaries record that a supplier *can* supply a part and that a project *uses* a part, while the ternary records the specific fact that this supplier supplied this part to this project. Splitting a genuinely three-way fact into pairs loses information; forcing a ternary where only pairwise facts exist invents it. See relationship-types[5].
- **Subclass vs superclass relationship.** Attaching a relationship higher up the hierarchy widens what the schema will accept. That looks harmless on a diagram and is not harmless afterwards — see below.

Worked example: attribute vs weak entity type

A candidate applies to a company, and each application involves an interview with a department on a date. Both diagrams below satisfy that description. The world does not choose between them.

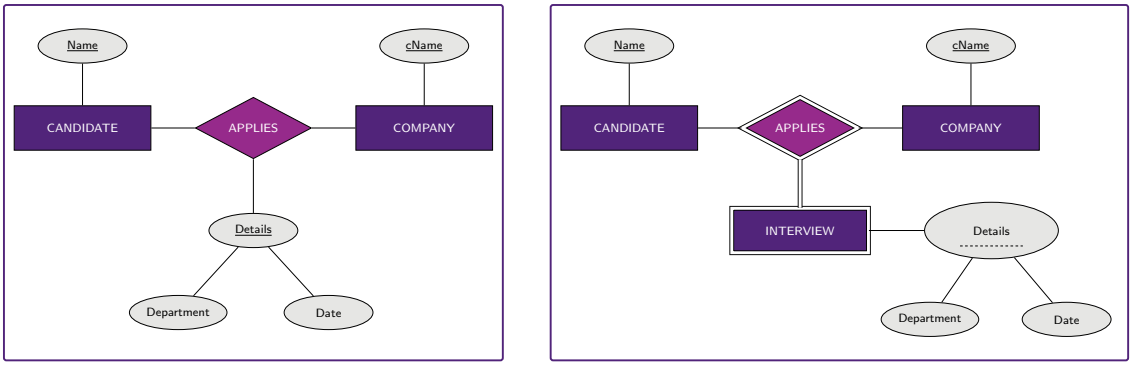


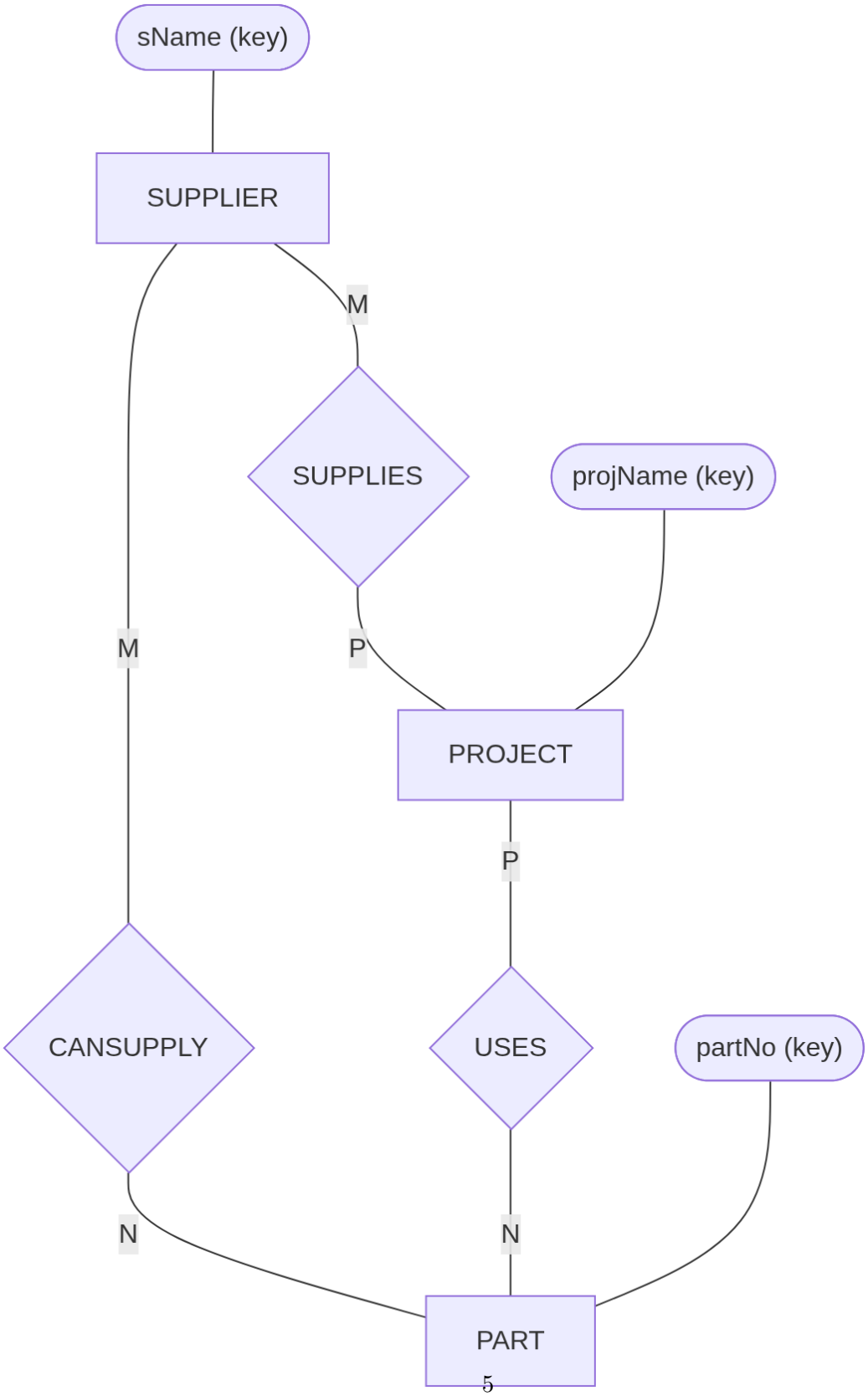
Figure 1: Attribute vs. (weak) entity type: two models of the same description. Left: the interview details hang off the `APPLIES` relationship as a composite **key** attribute `Details` (solid underline) made of `Department` and `Date`. Right: the interview becomes a thing — a weak entity owned by *both* `CANDIDATE` and `COMPANY`, so `APPLIES` becomes an identifying relationship (double diamond, double line into the weak entity) and `Details` demotes from key attribute to composite **partial** key (dotted underline). Nothing else on the diagram moves.

The two are alternatives, and the choice between them looks cosmetic. It stops looking cosmetic the moment the UoD says an interview *results in* a job offer: an attribute cannot be the endpoint of a relationship, so only the weak entity can carry a `RESULTSIN` link to a `JOBOFFER` entity. That extended diagram is drawn in 2026-03-02-weak-entities-the-eer-model-and-design-choices[6].

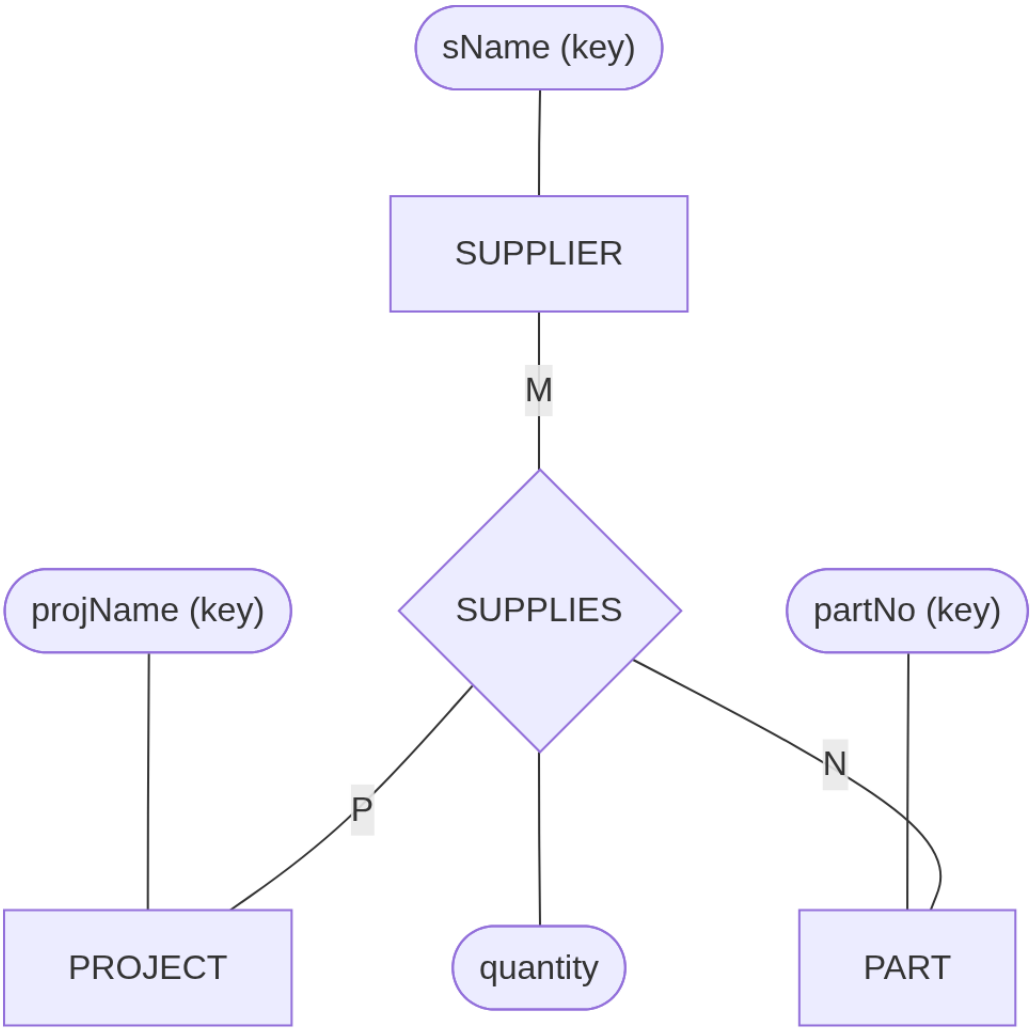
[5] courses/infs1200/relationship-types.pdf
[6] courses/infs1200/2026-03-02-weak-entities-the-eer-model-and-design-choices.pdf

Worked example: binary vs n-ary relationships

Suppliers, projects and parts. Modelled as **three binary relationships** — a supplier supplies a project, a supplier *can* supply a part, a project uses a part:


sName (key)
SUPPLIER
M
SUPPLIES
projName (key)
M
P
PROJECT
P
CANSUPPLY
USES
partNo (key)
N
N
PART

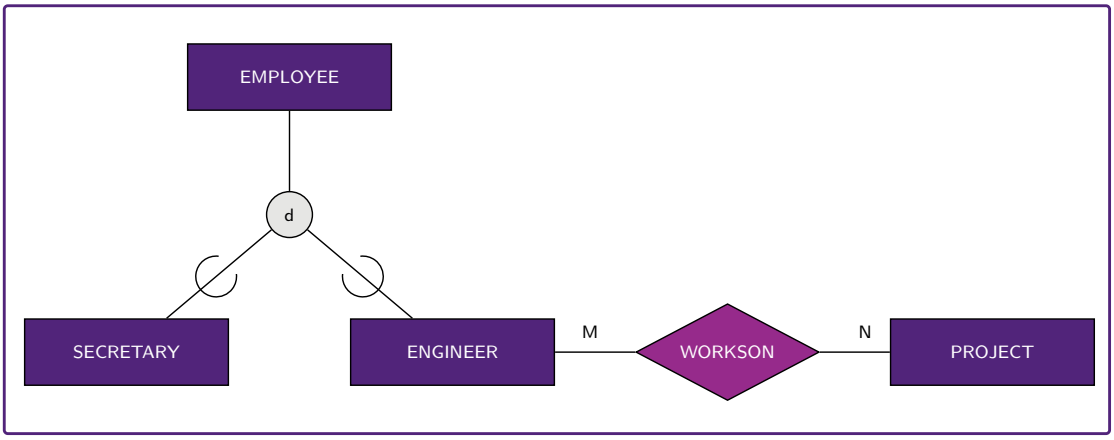
Modelled as **one ternary relationship**, with all three lines running into a single diamond that can then carry a `quantity`:



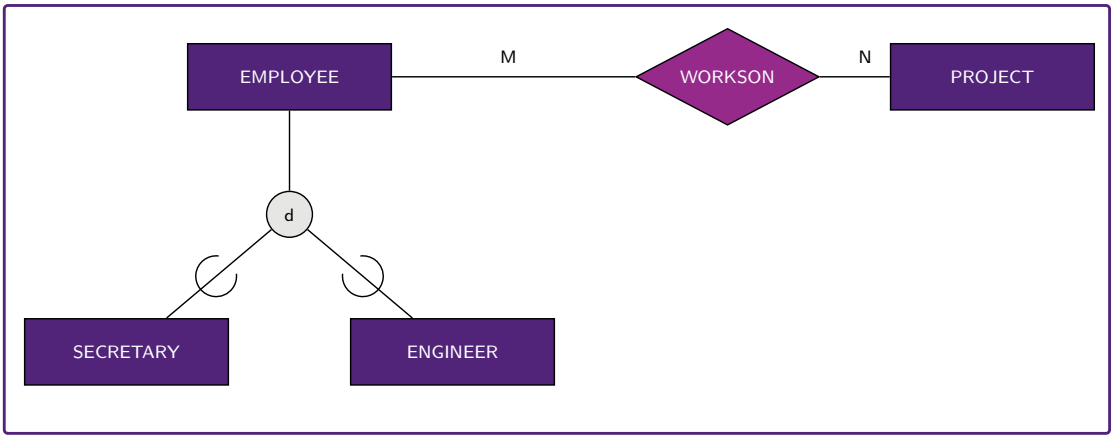
The `quantity` is the tell. It is a fact about a supplier-project-part *triple*, so there is nowhere to put it in the binary model — hanging it on `SUPPLIES` there would record how much a supplier sent a project without saying of what. This is the inequivalent case: the two diagrams do not produce the same database.

Worked example: subclass vs superclass relationship

`EMPLOYEE` specialises disjointly into `SECRETARY` and `ENGINEER`, and somebody works on projects. Which box does `WORKSON` attach to?



Option 1 — WORKSON attached to the subclass ENGINEER



Option 2 — WORKSON attached to the superclass EMPLOYEE

Figure 2: Subclass relationship vs. superclass relationship, reduced to the fragment where the two options differ. Option 1 attaches `WORKSON` to the subclass `ENGINEER`; Option 2 attaches it to the superclass `EMPLOYEE`. The rest of the company diagram (`MANAGES`, `WORKSFOR`, `CONTROLS` and `DEPARTMENT`) is the same under both options and is left out so the one difference is visible.

This is the clearest *inequivalent* choice of the four. Option 1 says only engineers work on projects — secretaries cannot be recorded on one at all. Option 2 says anyone may. Both satisfy a brief that only ever mentions engineers working on projects, but they enforce different things: Option 2 will happily accept a secretary on a project, and Option 1 will refuse an engineer's promotion out of the subclass without also dropping their project assignments. Attaching a relationship higher up the hierarchy always widens what the schema will accept.

Why the choice matters after the diagram

A conceptual choice becomes a *constraint the DBMS can or cannot enforce* once the diagram is mapped. The clearest example is the third mistake in week4-tutorial-case-study-3-dirt-road-driving[7]: mapping a trip's driver as a reference to `Staff` rather than to `Driver` lets an admin-only staff member be recorded as a trip's driver, because the schema no longer knows that a driver is a driver. The diagram-level decision about *which class a relationship attaches to* is the same decision as the schema-level decision about *which relation the foreign key points at*.

This is also why an assumption noted under a diagram is worth marks and worth memory: it's the record of a constraint that the notation alone couldn't distinguish. Once the schema exists, refining it further is a different exercise with different tools — see functional-dependencies[8] and normal-forms[9] for redundancy-driven refinement, and conceptual-database-design[10] for where in the pipeline each one sits.

[7] courses/infs1200/week4-tutorial-case-study-3-dirt-road-driving.pdf
[8] courses/infs1200/functional-dependencies.pdf
[9] courses/infs1200/normal-forms.pdf
[10] courses/infs1200/conceptual-database-design.pdf