ER Mapping Pitfalls

Table of contents

Keys 1
Relations that shouldn't exist 2
Choosing sides 3
Attributes 3
EER 3
Style 4

A checklist for reviewing a finished relational schema — every item here is a mistake actually made and corrected in week5-tutorial-applied-class-4-entity-relationship-mapping[1], week4-tutorial-case-study-3-dirt-road-driving[2] or the worked questions in 2026-03-16-er-to-relational-mapping[3]. The mapping rules themselves are in er-to-relational-mapping[4]; this is what to check *after* applying them.

Keys

Most of this section is one question asked five ways: *what exactly is this relation's primary key?* (relational-keys[5]).

A weak entity's partial key is not a foreign key. For a weak `CITY` owned by `PROVINCE`, the correct mapping is `Cities [cName, pName, mayor]` with only `Cities.pName references Provinces.pName`. `cName` is the city's own partial key — it references nothing. Declaring both columns as foreign keys is wrong.

A weak entity's key includes its owner's key. Mapping `Cities [cName, mayor]`, keyed on `cName` alone, silently asserts that city names are globally unique — which is precisely the assumption that made `CITY` weak in the first place.

[1] courses/infs1200/week5-tutorial-applied-class-4-entity-relationship-mapping.pdf
[2] courses/infs1200/week4-tutorial-case-study-3-dirt-road-driving.pdf
[3] courses/infs1200/2026-03-16-er-to-relational-mapping.pdf
[4] courses/infs1200/er-to-relational-mapping.pdf
[5] courses/infs1200/relational-keys.pdf

An ordinary 1:N foreign key is not part of the primary key. For `A -N R 1→ B`, the answer is `A [a, b, d]` keyed on `a` alone, with `A.b references B.b` as a plain column. Promoting `b` into `A`'s key is weak-entity treatment applied to a strong entity. (The genuine exception is an entity with no key of its own, identified through its owners — see foreign-keys[6].)

A composite foreign key is one constraint, not several. The cinema schema must say

```
ViewingSnack.{name, id, date, time} references Viewing.{name, id, date, time}
```

and not four separate single-column constraints. Splitting it breaks the link to *the specific* `Viewing` row: nothing then forces all four columns to match one real viewing rather than four unrelated ones. In the same botched attempt, `name` was pointed back at `Movie` rather than `Viewing` — a foreign key meant to identify *which viewing* a snack belongs to must target `Viewing`.

When the owner's key is composite, the inherited foreign key is composite. `Branch`'s key is `(bankCode, branchNo)`, so accounts and loans at a branch carry both columns: `Account.{bankCode, branchNo} references Branch.{bankCode, branchNo}`.

Relations that shouldn't exist

An identifying relationship never becomes a relation. A weak entity is connected to its owner purely through the inherited key. Inventing an `In [cName, pName]` relation alongside the weak entity's own is a distinctive wrong answer.

1:1 and 1:N relationships don't get their own relation by default. They are foreign keys on an existing relation. Creating an `S [b, c]` relation for a 1:N relationship `S` treats a relationship type as an entity type. (Doing this *deliberately* for a sparse relationship, to avoid nulls, is a different matter — a considered choice, not the default rule.)

Two entity types don't merge into one relation. Collapsing `S` and `T` into `ST [a1, b1]` because they happen to be in a 1:1 relationship discards a conceptual modelling decision that was made on purpose.

Nor does a relationship simply vanish. Mapping `S -1:1 R- T` to `S [a1]`, `T [b1]` with no foreign key anywhere loses all information about `R`.

[6] courses/infs1200/foreign-keys.pdf

Choosing sides

For 1:1, put the foreign key on the total-participation side. If every department has a manager but only some employees manage, extend `Department`. Choosing the other side doesn't lose information, but it fills the column with nulls, and the course's rule is to prefer the total-participation side. Between a schema putting `b1` into `S` and one putting `a1` into `T`, the first is correct for exactly one reason: `S` is preferred over `T` because it has full participation.

For 1:N, the foreign key goes on the N side — no judgement involved. In a chain `A -N R 1→ B -N S 1→ C`, that means `A` references `B` and `B` references `C`; each arrow is decided independently by where the N is.

For N-ary, only the many-sides make the key. All participants get foreign keys, but a `1`-side participant's foreign key is a plain attribute — and if the relationship carries a key attribute of its own, that attribute joins the key alongside the many-side foreign keys.

A ternary relationship with no many side gets no relation. `1:1:1` and `N:1:1` are explicitly excluded from step 7: creating an `R` relation for one of them is a wrong answer, because the relationship is absorbed into a participating relation as two foreign keys — into the `N` side if there is one, otherwise into whichever side is chosen. See step 7 of er-to-relational-mapping[7].

Attributes

Derived attributes are never columns. `NumberOfEmployees` is computed from the data, so storing it just creates something that can disagree with the data.

Composite attributes contribute their leaves, not themselves. `Name` becomes `fName`, `mIt`, `lName` — there is no `name` column.

Multivalued attributes always become their own relation, keyed on the owner's key plus the value. A column can only hold one atomic value (see relational-model[8]).

EER

Relationships drawn to a subclass reference the subclass relation. If only a `Specialist` can run a `SpecialistClinic`, then `Runs.licenceNo references Specialist.licenceNo` — not `Doctor`. Pointing at the superclass silently widens the rule to every doctor and every clinic, losing the constraint the specialisation was drawn to express (specialisation-and-generalisation[9]) — and the same widening is what er-design-choices[10] calls the subclass-vs-superclass relationship fork.

[7] courses/infs1200/er-to-relational-mapping.pdf
[8] courses/infs1200/relational-model.pdf
[9] courses/infs1200/specialisation-and-generalisation.pdf
[10] courses/infs1200/er-design-choices.pdf

A subclass with no attributes of its own still gets a relation. `SpecialistClinic [regNo]` with a foreign key back to `Clinic` records *membership*, which is information even when there are no extra columns.

Map subclasses before the relationships that use them. The `+1` step runs right after step 1 or 2 for a reason — a later step that needs to reference `Specialist` can't if `Specialist` doesn't exist yet.

Style

Schemas are marked against the course style guide as well as the mapping rules: UpperCamelCase table names, lowerCamelCase attribute names, each relation's `references` lines directly beneath it, and a single continuous underline for a composite primary key. See relational-mapping-notation[11].

[11] courses/infs1200/relational-mapping-notation.pdf