Relationship Constraints

Table of contents

Cardinality ratio 1
Participation constraint (existence dependency) 2
Weak entities 3

Constraints on a relationship type limit the possible combinations of entities that may participate in its relationship set — determined by the Universe of Discourse (UoD), not chosen arbitrarily. There are exactly two kinds, **cardinality constraints** and **participation constraints**. Introduced in 2026-02-23-conceptual-database-design-and-the-er-model[1]; see er-diagram-notation[2] for the full symbol legend.

Cardinality ratio

A **cardinality ratio** for a relationship set specifies the number of relationships in the set that an entity can participate in. The intuition is a customer buying items: one particular customer can purchase many different items, *however* an individual item — this particular cake — cannot be purchased by several different customers.

Ratio	Name	Meaning
1:1	One-to-one	Both entities can participate in only one relationship instance
N:1	Many-to-one	One entity can participate in many relationship instances, the other in only one
1:N	One-to-many	The same constraint, viewed from the other side

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

Ratio	Name	Meaning
M:N	Many-to-many	Both entities can participate in many different relationship instances

Three worked readings, all on the same two entity types, are worth memorising as a set because the English is nearly identical each time and only the ratio changes:

Statement of the UoD	Ratio
"Each employee can work on any number of projects, and each project can have any number of employees working on it"	`EMPLOYEE` —N— `WORKSON` —M— `PROJECT` (M:N)
"Each department can have any number of employees, but an employee can work for at most one department"	`EMPLOYEE` —N— `WORKSFOR` —1— `DEPARTMENT` (N:1)
"Each department can have at most one manager and each employee can manage at most one department"	`EMPLOYEE` —1— `MANAGES` —1— `DEPARTMENT` (1:1)

Note where the `N` sits in the second row: on the `EMPLOYEE` side, because it is employees that are many. Reading the label on the wrong end is the standard error — see interpreting-er-diagrams[3].

Participation constraint (existence dependency)

The course's own name for the participation constraint is **existence dependency**: it indicates whether the existence of an entity *depends* on its relationship to another entity — i.e. whether every instance of the entity type must participate in at least one relationship instance.

- **Total participation** (double line in the diagram) — every entity instance *must* participate. "Every employee must work for a department" makes `EMPLOYEE` totally participate in `WORKSFOR`, and the phrasing to use is that **`EMPLOYEE` is existentially dependent on `DEPARTMENT` via the `WORKSFOR` relationship type**.
- **Partial participation** (single line) — an entity instance is *not* required to participate. "Every employee can manage 0 or more departments" makes `EMPLOYEE`'s participation in `MANAGES` partial.

[3] courses/infs1200/interpreting-er-diagrams.pdf

The two ends are constrained independently, and `MANAGES` is the example that shows it: `EMPLOYEE` participates partially (most employees manage nothing) while `DEPARTMENT` participates totally (every department must have a manager), so `DEPARTMENT` is existentially dependent on `EMPLOYEE` via `MANAGES` and not the other way round.

Why the participation constraint is worth getting right

It is not just documentation: it decides where a foreign key ends up. The Module 2 mapping rule for a binary **1:1** relationship says to pick one of the two participating entity types and give it the foreign key, and that *preference is given to entity types with total participation.* `MANAGES` is exactly that case — because `DEPARTMENT` totally participates, the mapping extends `Department` with `mgrSSN` (plus the relationship's own `startDate`, renamed `mgrStartDate`) rather than putting a department reference in `Employee`, which would be null for almost every employee. The mirror-image answer fails for precisely this reason: given `S 1 R 1 T`, putting the foreign key in `T` is wrong, because `S` has full participation and is preferred over `T`. See er-to-relational-mapping[4] for the rule in full.

Weak entities

An entity type with no key attribute of its own is a **weak entity** — it can be identified uniquely only by considering the primary key of another **owner entity** together with its own **partial key** (underlined with a dotted line). The relationship type that relates a weak entity to its owner is the **identifying relationship**, and the weak entity always has **total participation** in it: it cannot exist without its owner. Both the weak entity box and the identifying relationship diamond are drawn with double lines.

An example is an `EMPLOYEE` (key `SSN`) who `INSURES` a `DEPENDANT` (partial key `Pname`, plus `Age`), 1:N, with `Cost` as a descriptive attribute on the identifying relationship. Two different employees can each insure a dependant named John, which is precisely why `Pname` cannot be a key on its own — the identity of a dependant is `(SSN, Pname)`.

Provinces and cities have the same shape, and it is the compact example to memorise because all three notations — cardinality, participation and partial key — are on it at once:

The two `name`s do different jobs. `PROVINCE.name` identifies a province outright; `CITY.name` identifies a city only *within* a province, so the mapped `Cities` relation needs a `pName` column as part of its primary key and as a foreign key to `Provinces` — the mapping is `Cities [cName, pName, mayor]` with `Cities.pName references Provinces.pName`, and the wrong mappings are the ones that drop `pName` from the key, make `cName` a foreign key, or turn the identifying

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

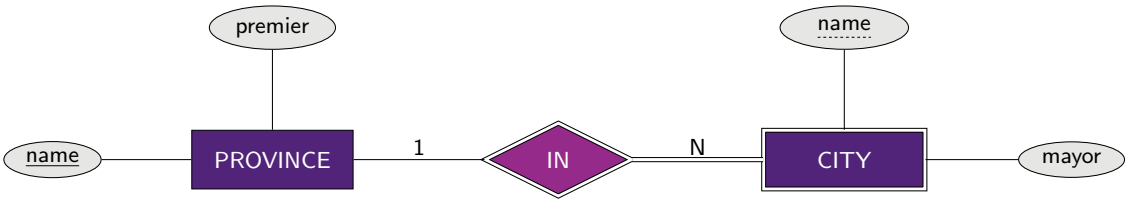

Figure 1: A weak entity in full notation. `CITY` is weak (double box) with the partial key `name` (dash-underlined); `IN` is the identifying relationship (double diamond); the `CITY`–`IN` edge is doubled for `CITY`'s total participation, and the ratio is 1:N, so one province holds many cities. `PROVINCE`'s own `name` is a full key (solid underline). Entity boxes are purple, relationship diamonds magenta and attribute ovals grey.

relationship `IN` into a relation of its own. See er-to-relational-mapping[5] for the mapping rule that produces this.

A weak entity can have two (or more) owner entity types, in which case the identifying relationship is n-ary rather than binary. The generic version shows how the constraints are placed in that case — many weak entities per combination of owners, one owner of each kind per weak entity:

`INTERVIEW` is the worked case: it is owned by both `CANDIDATE` and `COMPANY`, so the identifying relationship `APPLIES` is **ternary**, and `INTERVIEW`'s composite partial key `Details` (`Department`, `Date`) only distinguishes interviews *within* one candidate/company pair. `INTERVIEW` is a full entity type despite being weak, so it can carry relationships of its own — it `RESULTSIN` a `JOB OFFER`. See er-design-choices[6] for the decision that produced it and interpreting-er-diagrams[7] for the claims it makes false.

[5] courses/infs1200/er-to-relational-mapping.pdf
[6] courses/infs1200/er-design-choices.pdf
[7] courses/infs1200/interpreting-er-diagrams.pdf

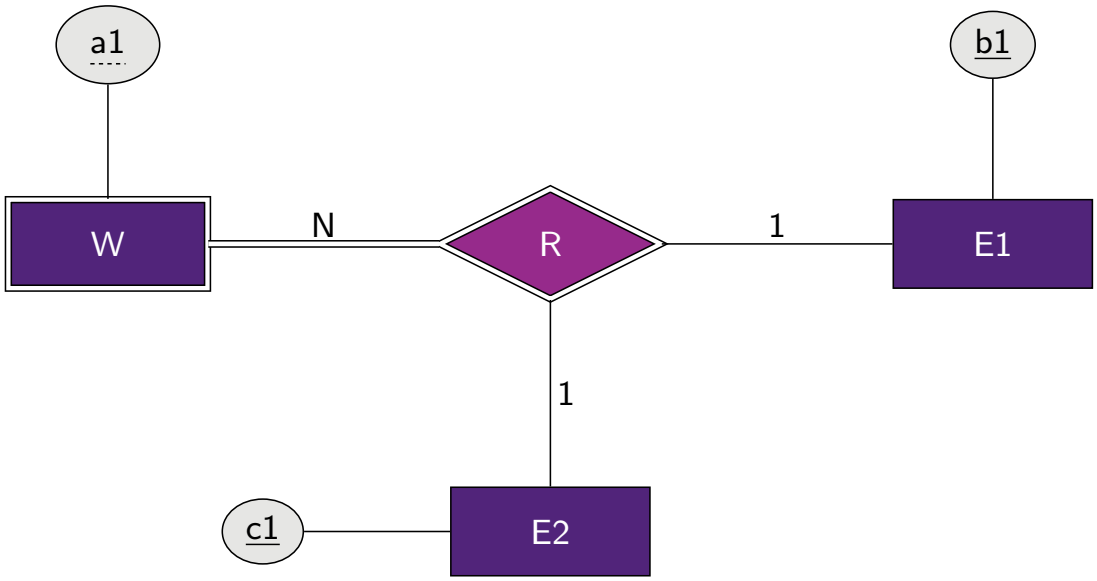

Figure 2: A weak entity with two owner entity types. The identifying relationship `R` is ternary: `N` on the weak side, `1` towards each owner, with the double edge marking `W`'s total participation. `W`'s identity is its partial key `a1` plus *both* owners' keys, which is why the mapped relation is `W [a1, b1, c1]` with `b1` and `c1` as foreign keys.