# Conceptual Database Design and the ER Model

## Table of contents


Module outline . . . . . 1
Conceptual database design . . . . . 2
The ER model . . . . . 2
Attributes . . . . . 2
Exercise — Student entity . . . . . 4
Relationships . . . . . 5
Question — Entities and Relationships (courses/students) . . . . . 6
Relationship constraints . . . . . 6
Question — Relationship constraints . . . . . 9
Exercise — ABC Banks . . . . . 10


Module 1, part 1. See er-diagram-notation[1] and relationship-constraints[2] for the reference symbol legend and the cardinality/participation definitions introduced here.

### Module outline

Module 1 has five sections. This note covers the first three, and 2026-03-02-weak-entities-the-eer-model-and-design-choices[3] covers the last two:

1. Database design overview
2. Entities and relationships
3. Relationship constraints
4. *The EER diagram* — next lecture
5. *Design choices* — next lecture

[1] courses/infs1200/er-diagram-notation.pdf
[2] courses/infs1200/relationship-constraints.pdf
[3] courses/infs1200/2026-03-02-weak-entities-the-eer-model-and-design-choices.pdf

The module answers three questions: why conceptual modelling matters for a successful database application; how the Entity-Relationship model is used to do it; and how to evaluate trade-offs between design choices by comparing alternative ER diagrams for the same system.

## Conceptual database design

- **Step 1**: identify the "Universe of Discourse" (UoD) — the database models some "mini-world"/UoD, not everything in the world.
- **Step 2**: convert the UoD into a data model that a database can capture.
- A model is never perfect — three categories of phenomena around any conceptual schema: **common phenomena** (captured in the model, e.g. "every employee works for a department"), **phenomena not captured** (e.g. "some employees go out for dinner on Fridays" — irrelevant, left out), and **phenomena not true in the world** (e.g. an over-general assumption like "every secretary can type").

See conceptual-database-design[4], which draws the two-step pipeline and the three-phenomena picture.

## The ER model

The Entity-Relationship (ER) model provides a **graphical representation of data entities** — it helps define a project's scope/requirements for clients and businesses. An **entity** is a physical or conceptual object with data (attributes) associated with it; the same real-world entity can have different attributes recorded depending on the system's requirements (e.g. an R&D organisation records an employee's `Degree`/`Field of Study`, a retail organisation records `Sales Experience` instead).

An **entity type** provides the format (name + attributes) for recording a particular kind of entity — drawn as a rectangle, with attribute ovals attached. The **entity set** is the collection of all entities of one entity type in the database at a point in time (maps to a table).

The whole notation fits on one diagram, which is the target to work towards. Every symbol on it is defined later in this note or the next:

## Attributes

- **Key attribute**: every entity type has at least one **key (or uniqueness) constraint** — its value is distinct for each individual entity in the entity set (name underlined). Multiple keys are possible, and a key must hold for **every** possible extension of the entity type.

[4] courses/infs1200/conceptual-database-design.pdf

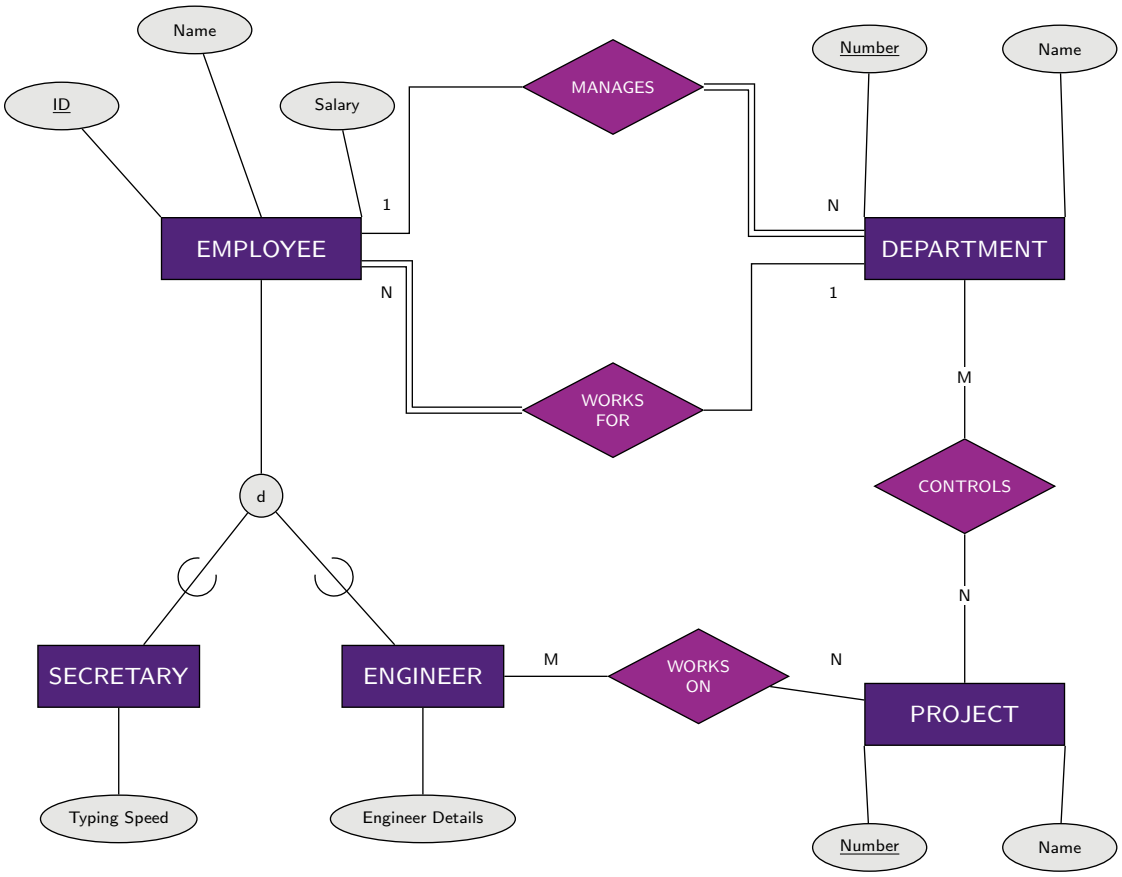



Figure 1: An example company ER diagram. Everything the module teaches is on it: entity types (purple rectangles), relationship types (magenta diamonds), key attributes (underlined ovals), cardinality ratios (1/N/M on the edges), total participation (double edges — every employee must work for a department, every department must have a manager) and an EER disjoint specialisation (the `d` circle). The diagram exists to gauge how the system works and how data within it is connected.

- **Composite vs simple**: a composite attribute (e.g. `Name`) can be split into simple attributes with independent meaning (`Fname`, `Mname`, `Lname`).
- **Composite key**: a combination of simple attributes that together must be unique (e.g. a car's `Registration` = `State` + `Number`, alongside a separate simple key `VehicleID`) — an entity type can have several candidate keys like this.
- **Single-valued vs multivalued**: the split is stated over *simple* attributes. A multivalued attribute (double-lined oval) can hold more than one value per entity (e.g. `Degree` — one person can hold multiple degrees).
- **Stored vs derived**: a derived attribute (dotted oval) is computed from a related attribute value (e.g. `Age` from `BirthDate`).
- **Value sets**: the set of values that may be assigned to an attribute (e.g. `employeeAge`: integers between 21 and 65; `vehicleRegistrationNum`: a string of three alphabetic characters followed by three integers) — never shown on the diagram itself. A `null` value represents an inapplicable (`tertiaryDegree`), unknown (`homePhone`) or missing (`height`) value.

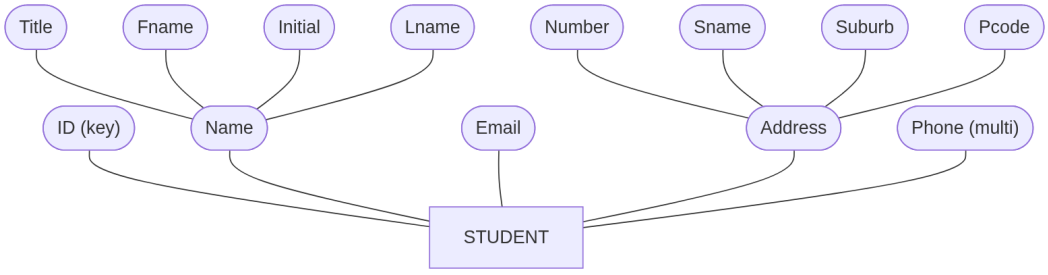
entity-types-and-attributes[5] draws the `EMPLOYEE` figure with every kind of attribute on it, and the `CAR` candidate-key figure.

## Exercise — Student entity

*Every student has a unique id, name (title/first/middle initial/last), email, address (number/street/suburb/postcode), and one or more phone numbers. Draw an ER diagram.*

**Solution**

`STUDENT` entity type with key attribute `ID`; composite attribute `Name` (→ `Title`, `Fname`, `Initial`, `Lname`); simple attribute `Email`; composite attribute `Address` (→ `Number`, `Sname`, `Suburb`, `Pcode`); and multivalued attribute `Phone` (double-lined oval, since a student can have more than one phone number).




[5] courses/infs1200/entity-types-and-attributes.pdf

## Relationships

A **relationship** is an association among **two or more** entities (e.g. "Paris Lane works on the project FileZilla"). A **relationship type** defines it — drawn as a diamond connected to the associated entity types. A relationship type may have **descriptive attributes**, and it may have **key attributes** as well:

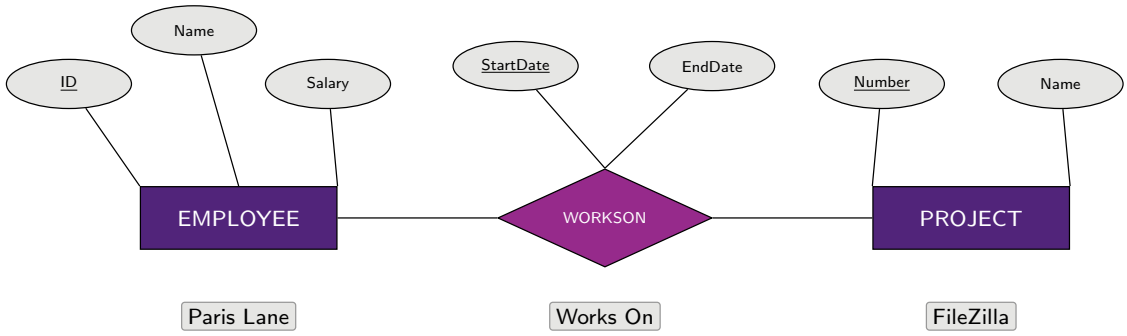



Figure 2: Relationship types carry attributes of their own. `WORKSON` records a `StartDate` and an `EndDate` for each employee/project pairing; `StartDate` is underlined, so it is a *key* attribute of the relationship, not merely a descriptive one — a fact the mapping stage has to preserve. The row underneath is the instance-level reading: Paris Lane *works on* FileZilla.

- **Relationship degree** = the number of *participating entity types*: **binary** (2 entities, e.g. `DEPARTMENT WORKSFOR EMPLOYEE`), **ternary** (3 entities, e.g. a single `SUPPLIES` diamond joining `SUPPLIER`, `PROJECT` and `PART`), or **n-ary** (3+) in general.
- **Roles**: each entity type participating in a relationship type plays a particular **role**, and the role name explains *what the relationship means*. In `DEPARTMENT WORKSON EMPLOYEE`, `DEPARTMENT` is the *Employer* and `EMPLOYEE` is the *Worker*.
- **Recursive relationships**: the same entity type can participate more than once in one relationship type, under different roles — `EMPLOYEE MANAGES EMPLOYEE`, with the two lines labelled *Manager* and *Subordinate*.
- The **relationship set** is the collection of all relationships of one relationship type at a point in time. The same relationship set has three representations: the ER fragment (`EMPLOYEE` N — `WORKSON` — M `PROJECT`), a ball-and-line instance diagram, and a three-column table (`WorksOnID`, `EmployeeID`, `ProjectID`), which is the shape the relationship eventually takes once mapped (er-to-relational-mapping[6]).

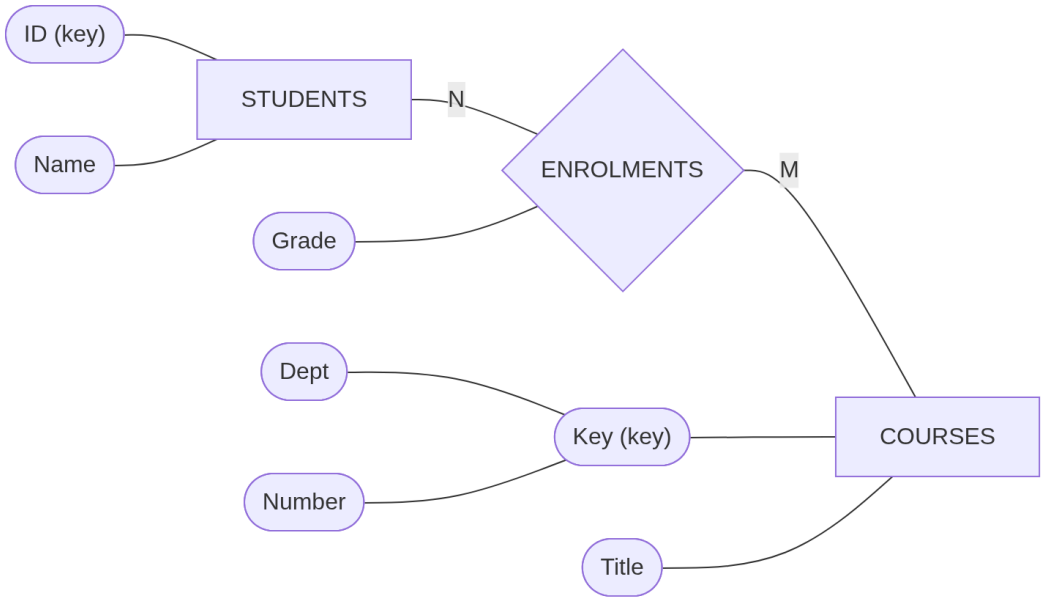
[6] courses/infs1200/er-to-relational-mapping.pdf

## Question — Entities and Relationships (courses/students)

*Store info about students, courses, courses taken, and grades. Courses have a number/department/title (numbers assigned per-department, so different departments may reuse a number); students have a unique student ID and name; students enrol in courses and receive a grade.*

**Solution**

`COURSES` (composite key `Key` = `Dept` + `Number`, plus `Title`) `ENROLMENTS` (attribute `Grade`) `STUDENTS` (key `ID`, `Name`) — an M:N relationship, since a student can enrol in many courses and a course has many enrolled students.




The composite key comes from the clause that numbers are assigned by departments and different departments may use the same number: `Number` alone is not unique, so `Dept` + `Number` together are the key.

## Relationship constraints

Constraints on a relationship type **limit the possible combinations of entities that may participate in the corresponding relationship set**, and both kinds are **determined by the UoD** — they are read off the description, not chosen for convenience. There are two:

| Constraint | What it fixes |
|---|---|
| **Cardinality constraint** | How many relationships of the set one entity can participate in |

| Constraint | What it fixes |
|---|---|
| **Participation constraint** (also called **existence dependency**) | Whether an entity *has* to participate at all |

See relationship-constraints[7] for what each ratio and each participation constraint asserts.

### Cardinality ratio

A **cardinality ratio** for a relationship set specifies the number of relationships in the set that an entity can participate in. Take a customer buying items: one customer can purchase many different items, but an individual item (a particular cake) cannot be purchased by several different customers.

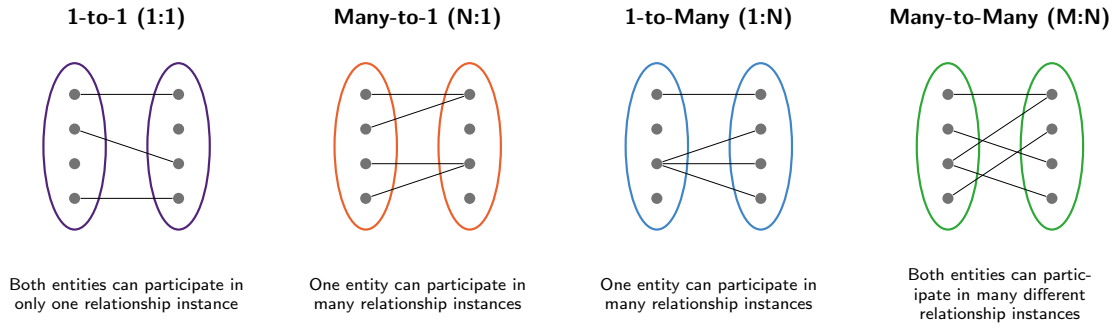



Figure 3: The four cardinality ratios as instance diagrams, colour coded: purple 1:1, orange N:1, blue 1:N, green M:N. Each dot is one entity, each line one relationship instance. Which dots are joined is illustrative; the constraint is the shape — a matching (1:1), a fan-in (N:1), a fan-out (1:N), or an unrestricted graph (M:N).

**N:1 and 1:N are the same constraint read from opposite ends.** The one relationship `EMPLOYEE` N — `WORKSFOR` — 1 `DEPARTMENT` fits *both* the orange N:1 picture and the blue 1:N picture. Which name you use depends on which entity type you name first, and the course's notation guide fixes the reading: `E1 -1- R -N- E2` is "cardinality ratio 1:N for E1:E2 in R". The practical rule for reading a diagram is that the number written next to an entity says how many of *that* entity may take part with one of the other: `N` beside `EMPLOYEE` = many employees per department; `1` beside `DEPARTMENT` = one department per employee.

Three worked examples:

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

| UoD sentence | Ratio |
|---|---|
| "Each employee can work on any number of projects, and each project can have any number of employees working on it" (`WORKSON`) | M:N |
| "Each department can have any number of employees, but an employee can work for at most one department" (`WORKSFOR`) | N:1 (equivalently 1:N the other way) |
| "Each department can have at most one manager and each employee can manage at most one department" (`MANAGES`) | 1:1 |

### Participation constraints (existence dependency)

**Existence dependency indicates whether the existence of an entity depends on its relationship to another entity.** A **double line** between an entity type and a diamond means **total** participation; a **single line** means **partial**.

- "Every employee must work for a department" — `EMPLOYEE` has total participation in `WORKSFOR`, so *`EMPLOYEE` is existentially dependent on `DEPARTMENT` via the `WORKSFOR` relationship type.*
- "Every department must have a manager" — `DEPARTMENT` has total participation in `MANAGES`, so *`DEPARTMENT` is existentially dependent on `EMPLOYEE` via `MANAGES`*, while `EMPLOYEE`'s participation stays partial (some employees manage nothing, and legally so).

### Worked example — `EMPLOYEE` / `DEPARTMENT`

Figure 1 above is the diagram. It encodes four sentences:

1. Every department must have a manager — *total participation of `DEPARTMENT` in `MANAGES`* (double line on that edge).
2. Every employee can manage 0 or more departments — *partial participation of `EMPLOYEE` in `MANAGES`*, and the `1`/`N` labels make `MANAGES` 1:N for `EMPLOYEE`:`DEPARTMENT` (a department has at most one manager; an employee may manage several).
3. Every employee must work for a department — *total participation of `EMPLOYEE` in `WORKSFOR`.*
4. Each department can have any number of employees — with `N` beside `EMPLOYEE` and `1` beside `DEPARTMENT`, `WORKSFOR` is N:1.

The 1:1 example in the table above ("each department can have at most one manager **and each employee can manage at most one department**") is a *different* UoD sentence from statement 2: it is drawn `1 - MANAGES - 1`, whereas the company diagram is drawn `1 - MANAGES - N`. Two sentences, two ratios, same relationship name, so the ratio has to be read off the UoD and not off the verb.

`WORKSON` (`EMPLOYEE` M:N `PROJECT`) completes the picture: each employee can work on any number of projects, and each project can have any number of employees on it.

## Question — Relationship constraints

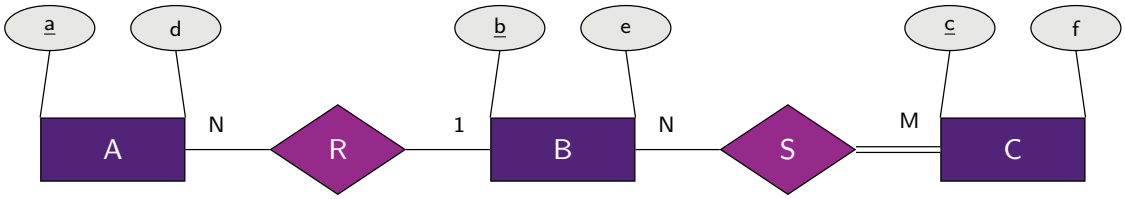



Figure 4: Entity types `A`, `B` and `C` joined by relationship types `R` and `S`. `A` contains `a1`, `a2`; `B` contains `b1`, `b2`; `C` contains `c1`, `c2`. Which relationship sets for `R` and `S` are possible? The double line into `C` is the whole difficulty — it constrains the *instances*, not just the schema.

**A.** `R = {}, S = {}`
**B.** `R = {(a1,b1)}, S = {(b2,c2)}`
**C.** `R = {(a1,b1), (a1,b2)}, S = {(b1,c1), (b2,c2)}`
**D.** `R = {(a1,b2)}, S = {(b1,c2), (b2,c1), (b1,c1)}`

**i** Solution

**(D).** Two constraints have to hold at once:

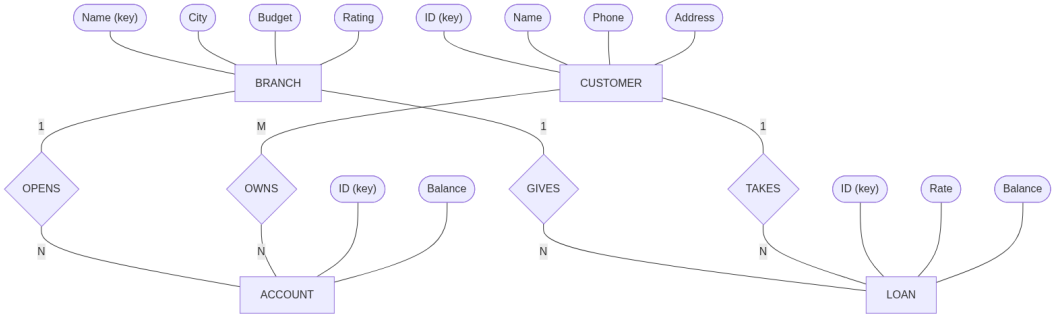
- `R` is N:1 for `A`:`B`, so each `a` may appear at most once in `R`. That kills **C**, where `a1` is paired with both `b1` and `b2`.
- `C` has *total participation* in `S` (the double line), so every `c` must appear in `S`. That kills **A** (nothing appears) and **B** (`c1` is missing). `S` being M:N places no further limit, so **D** — which lists both `c1` and `c2` and pairs each `a` at most once in `R` — is legal.

## Exercise — ABC Banks

*Model a bank with branches (unique name, city, budget, rating 1–10), customers (identified by name + phone, address recorded), accounts and loans (unique number, created/maintained by a single branch), where an account is assigned to one or more customers (balance never negative) and a loan is always assigned to a single customer (fixed interest rate, balance never negative).*

**Solution**

`BRANCH` (`Name`, `City`, `Budget`, `Rating`) —1:N `OPENS`→ `ACCOUNT` (`ID`, `Balance`); `BRANCH` —1:N `GIVES`→ `LOAN` (`ID`, `Rate`, `Balance`); `CUSTOMER` (`ID`, `Name`, `Phone`, `Address`) —M:N `OWNS`→ `ACCOUNT`; `CUSTOMER` —1:N `TAKES`→ `LOAN`.




Two clauses in the brief are *constraints the notation cannot carry*: "its balance can never be negative" is a value-set restriction, and value sets are never shown on an ER diagram — it resurfaces later as a domain constraint (integrity-constraints[8]). "Each account and loan is created and maintained by a single branch" is the participation constraint that makes `ACCOUNT` and `LOAN` total in `OPENS`/`GIVES`.

[8] courses/infs1200/integrity-constraints.pdf