Conceptual Database Design and the ER Model

Table of contents

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

Today's outline

- Conceptual database design
- Entities and relationships
- Relationship constraints

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.

[1] courses/infs1200/er-diagram-notation.pdf
[2] courses/infs1200/relationship-constraints.pdf

- 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").

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).

Attributes

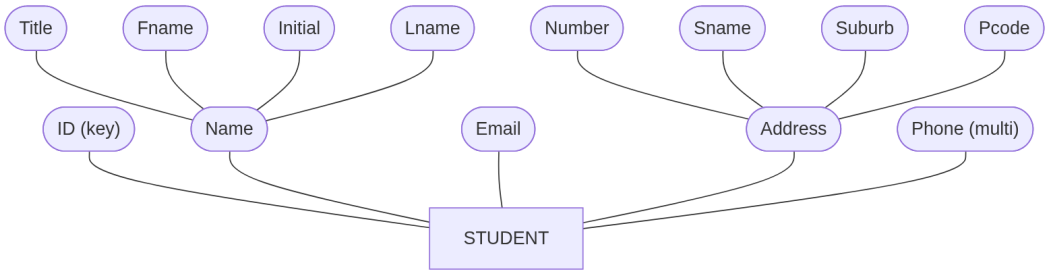
- **Key attribute**: every entity type has at least one — its value is unique for each entity in the entity set (name underlined). Multiple keys are possible, and a key must hold for every possible extension of the entity type.
- **Composite vs simple**: a composite attribute (e.g. `Name`) can be split into simple attributes with independent meaning (`FirstName`, `MiddleName`, `LastName`).
- **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**: 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 (dashed oval) is computed from another stored attribute (e.g. `Age` derived from `BirthDate`).
- **Value sets**: the set of legal values for an attribute (e.g. `employeeAge`: integers 21-65) — never shown on the diagram itself. A `null` value represents an inapplicable, unknown, or missing value.

In-class 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).



Relationships

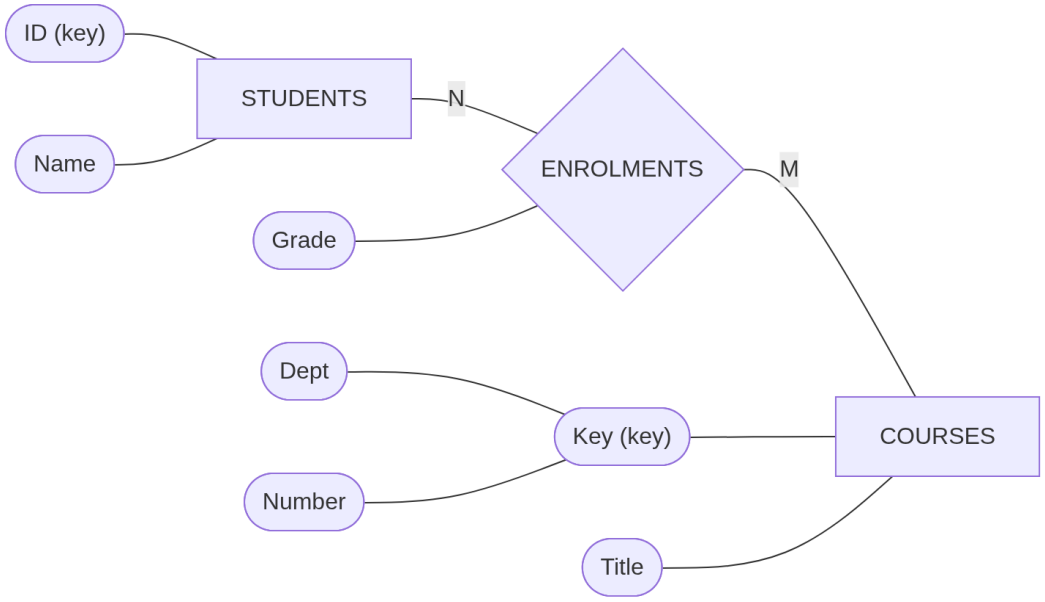
- A **relationship** is an association among two or more entities (e.g. "Paris Lane works on the project FileZilla"). A **relationship type** defines it — drawn as a diamond connected to its entity types, and may have its own descriptive/key attributes.
- **Relationship degree** = number of participating entity types: **binary** (2, e.g. `EMPLOYEE WORKSFOR DEPARTMENT`), **ternary** (3, e.g. `SUPPLIER SUPPLIES PART` via a `PROJECT`), or **n-ary** (3+) in general.
- **Roles**: each participating entity type plays a named role in the relationship, explaining what the relationship means (e.g. `DEPARTMENT` is the *Employer*, `EMPLOYEE` is the *Worker* in `WORKSON`).
- **Recursive relationships**: the same entity type can participate more than once in one relationship type, under different roles (e.g. `EMPLOYEE MANAGES EMPLOYEE`, with *Manager*/*Subordinate* roles).
- The **relationship set** is the collection of all relationship instances of one relationship type at a point in time.

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.



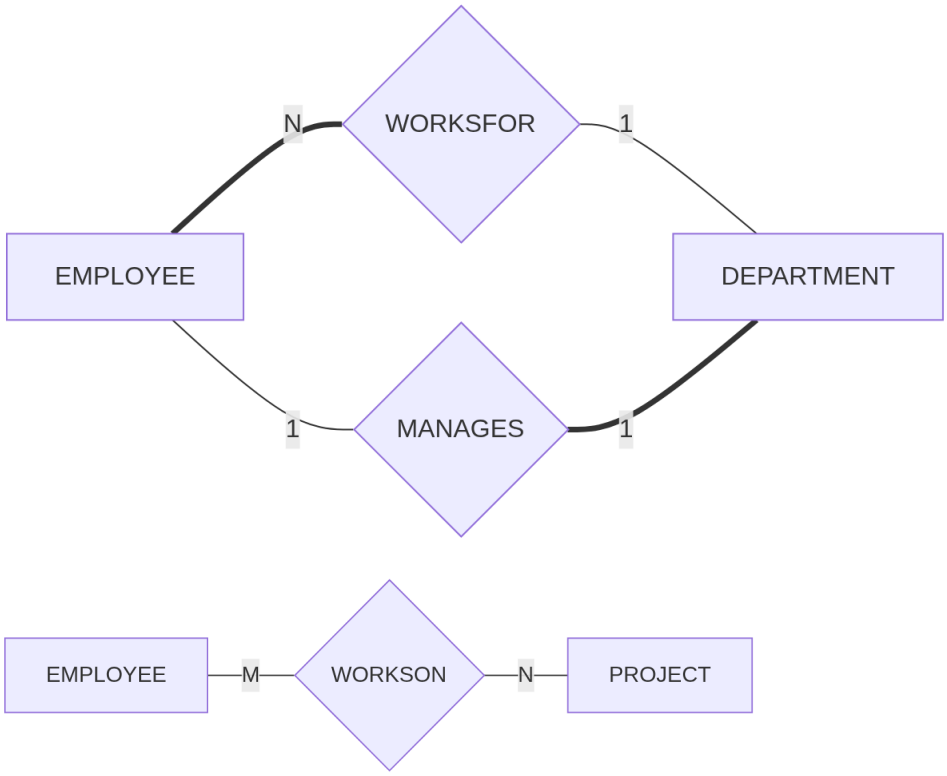
Relationship constraints

See relationship-constraints[3] for the cardinality ratio (1:1/1:N/M:N) and participation constraint (total/partial) definitions.

Worked example — `EMPLOYEE/DEPARTMENT`:

- `WORKSFOR`: N:1 — "each department can have any number of employees, but an employee can work for at most one department", and EMPLOYEE has total participation ("every employee must work for a department").
- `MANAGES`: 1:1 — "each department can have at most one manager and each employee can manage at most one department"; DEPARTMENT has total participation ("every department must have a manager") while EMPLOYEE's participation is partial ("every employee can manage 0 or more departments").
- `WORKSON` (`EMPLOYEE/PROJECT`): M:N — "each employee can work on any number of projects, and each project can have any number of employees working on it".

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

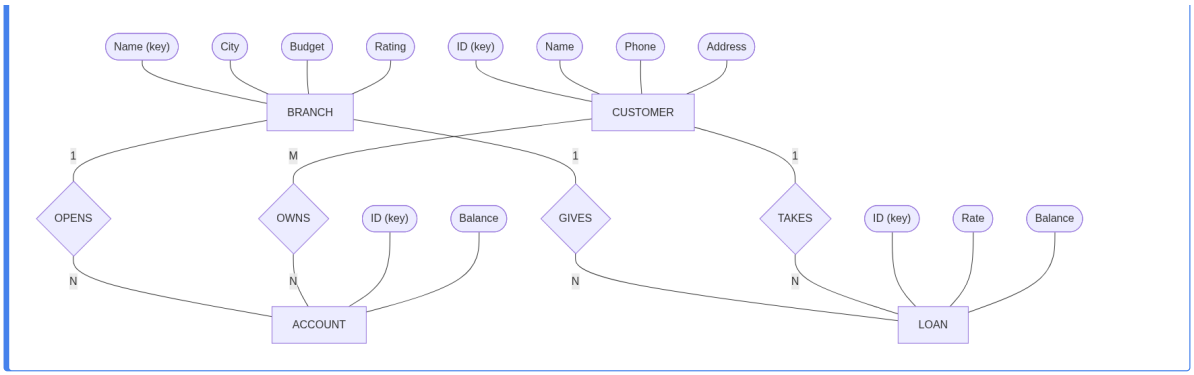


Exercise — ABC Banks

Model a bank with branches (unique name, city, budget, rating), customers (name + phone, address), accounts and loans (unique number, created/ maintained by a single branch), where an account is assigned to 1 customers and a loan is assigned to a single customer.

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`.


Name (key)
City
Budget
Rating
ID (key)
Name
Phone
Address
BRANCH
CUSTOMER
1
M
1
1
OPENS
OWNS
ID (key)
Balance
GIVES
TAKES
ID (key)
Rate
Balance
N
N
N
N
ACCOUNT
LOAN