Design of Relational Schemas

Databases

Pierre Senellart

11 March 2026

Outline

Why schema design matters

A poor schema leads to:

- redundancy: the same fact is stored repeatedly
- update anomalies: one change requires many updates
- insertion anomalies: some facts cannot be inserted independently
- deletion anomalies: deleting one fact accidentally deletes another

Schema design usually proceeds in two stages:

1. conceptual design: model the application domain
2. logical design: derive a good relational schema

Running example: a university information system

We consider:

- students
- teachers
- courses
- course sections in a semester (in large universities, the same course may be given several times, even within a semester, possibly by different teachers – these are course sections)
- enrollments
- grades

We will:

1. model this domain with an E–R diagram
2. translate it to a relational schema
3. study functional dependencies and normal forms

Outline

Entity–Relationship diagrams

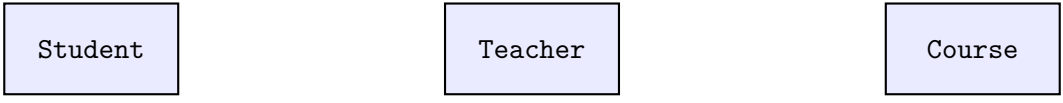
- E–R diagrams are a classical formalism for conceptual design
- They describe:
  - entity sets: classes of objects
  - relationship sets: associations between objects
  - attributes: pieces of information
  - constraints: keys, cardinalities, participation, etc.
- Goal: describe the application domain before deciding how to store it.
- Many different variants of E–R diagrams, we present specific notation, but you may encounter others

Entity sets

An entity set describes a family of objects.

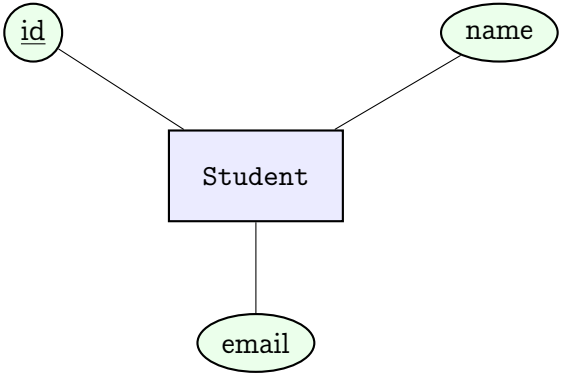
In our example:

- `Student`
- `Teacher`
- `Course`



Attributes

Entity sets have attributes.

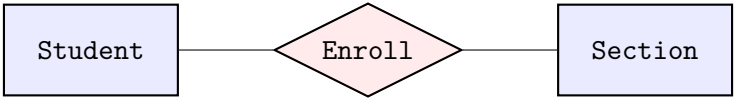


Here, id is underlined as a key attribute, i.e., an attribute whose value suffices to fully identify an entity (no two entities have the same value for this attribute).

Relationship sets

A relationship set associates several entity sets.

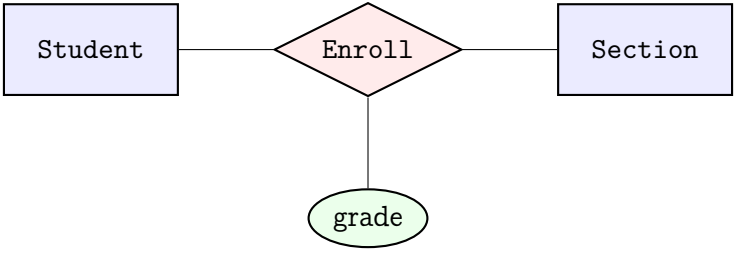
Example: a student enrolls in a section.



Attributes of relationships

Relationships can also have attributes.

For example, an enrollment can have a grade.



Cardinalities

Cardinalities describe how many times an entity can participate in a relationship.

Interpretation:

- a section is taught by at least one teacher
- a teacher may teach any number of sections, including zero

Interpreting cardinalities

A cardinality (m, n) on a relationship edge means:

- each entity of the entity set on that edge participates in at least m relationships (typically, 0 or 1 – partial or total participation, see next)
- and in at most n relationships (typically, 1 or ∞)

Examples:

- $(0, 1)$: optional, at most one
- $(1, 1)$: exactly one
- $(0, \infty)$: optional, arbitrarily many
- $(1, \infty)$: at least one, arbitrarily many

Other common notation:

- The lower bound is denoted by single/double lines, see next
- Only indicate the upper bound n, either 1 or ∞ (and ∞ is alternatively denoted N for an abstract number $\geqslant 1$)

Participation constraints

Participation can be:

- total: every entity participates ($m \geqslant 1$)
- partial: participation is optional ($m = 0$)

Examples:

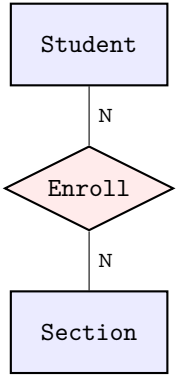
- every `Section` must correspond to some `Course`
- a `Student` may be enrolled in no `Section`

In classical notation, total participation is often shown by a double line. If this is done, no need to display m as a cardinality constraint.

Partial vs total participation

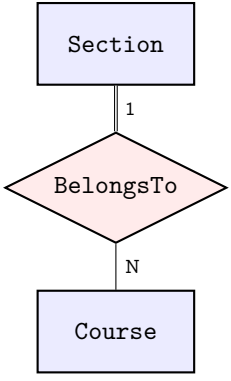
Participation constraints specify whether an entity must appear in a relationship.

Partial participation



A student may enroll in any number of sections, or even none (if on a break year) – a section may include any number of students.

Total participation



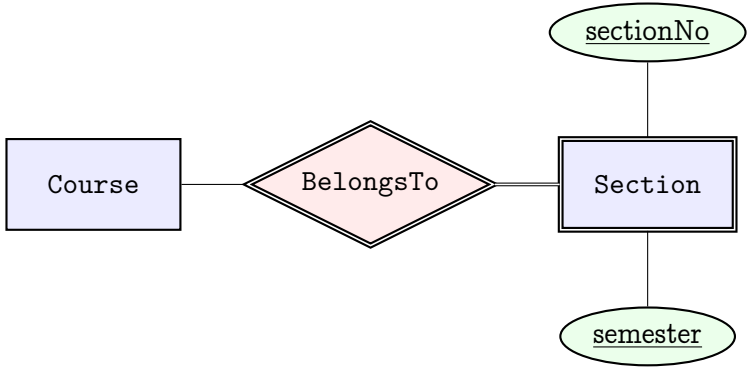
Every section must belong to a course, and can only belong to a single course. A course may have no section (if not offered).

Weak entity sets

A weak entity set has no key of its own and is identified relative to another entity set. In other words, its existence depends on another entity set.

Example: `Section` can be identified by:

(`course`, `sectionNo`, `semester`)

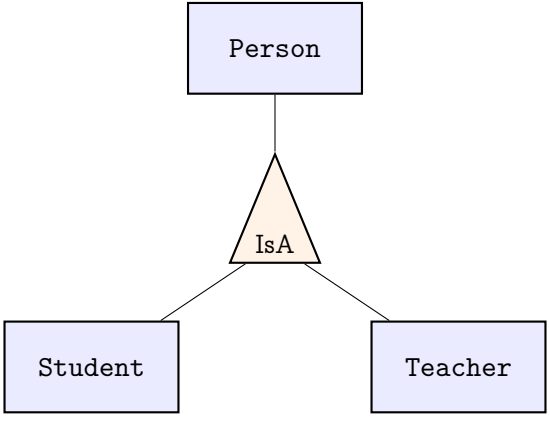


IsA hierarchies

Sometimes one entity set is a specialization of another.

Example:

- `Person`
- specialized into `Student` and `Teacher`



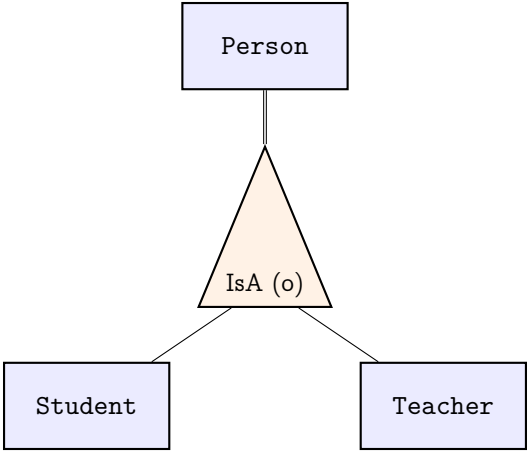
Specialization constraints

A specialization of a superclass into subclasses has two independent aspects:

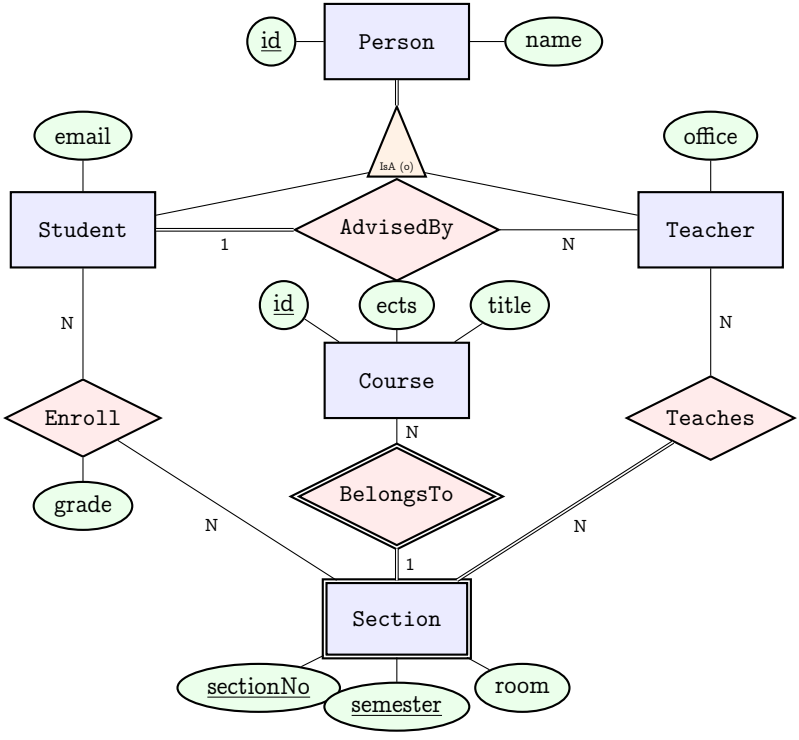
Partial vs total. Whether every entity of the superclass must belong to at least one subclass. Common notation: double line from the superclass to the IsA node for total participation.

Disjoint vs overlapping. Whether one entity may belong to several subclasses at the same time. Common notation: "(d)" or "(o)" annotations

Example of specialization constraints



Total, Overlapping: every person is at least one of the two


id
Person
name
email
IsA (o)
office
Student
AdvisedBy
Teacher
1
N
N
id
ects
title
N
N
Course
Enroll
Teaches
N
grade
BelongsTo
N
N
1
Section
sectionNo
semester
room

Discussion of the E–R model

Main modeling choices:

- `Section` is a weak entity attached to `Course`
- `Enroll` is a many-to-many relationship with attribute `grade`
- `AdvisedBy` is many-to-one: each student has at most one advisor
- `Teaches` is many-to-many, with total participation of `Section`: each section is taught by at least one teacher

This already suggests the keys and foreign keys of the relational schema.

Outline

Systematic translation principles

A standard translation from E–R to a relational schema is:

1. each strong entity set becomes a relation
2. each weak entity set becomes a relation containing the key of its owner as part of its primary key
3. each many-to-many relationship (i.e., N–N) becomes a relation
4. each many-to-one or one-to-one relationship (i.e., N–1 or 1–1) can often be translated using a foreign key
5. relationship attributes become attributes of the relation created for that relationship

Strong entity sets

Strong entity sets become relations with their attributes:

- `Person(id, name)`
- `Student(id, email, advisor)`
- `Teacher(id, office)`
- `Course(id, title, ects)`

Here:

- `id` in `Student` and `Teacher` are foreign keys referencing `Person(id)`
- `advisor` is a foreign key referencing `Teacher(id)` if `AdvisedBy` is translated as a many-to-one relationship.

Weak entity sets

If `Section` is weak and identified relative to `Course`, then:

`Section(`<u>`course`</u>`, `<u>`sectionNo`</u>`, `<u>`semester`</u>`, room)`

with:

- primary key `(course, sectionNo, semester)`
- foreign key `course` referencing `Course(id)`

Many-to-many relationships

The `Enroll` and `Teaches` relationships become their own relations:

`Enroll(`<u>`student`</u>`, `<u>`course`</u>`, `<u>`sectionNo`</u>`, `<u>`semester`</u>`, grade)`
`Teaches(`<u>`teacher`</u>`, `<u>`course`</u>`, `<u>`sectionNo`</u>`, `<u>`semester`</u>`)`

with foreign keys:

- `student` references `Student(id)`
- `(course, sectionNo, semester)` references `Section(course, sectionNo, semester)`
- `teacher` references `Teacher(id)`

Translation of IsA hierarchies

For `Person` specialized into `Student` and `Teacher`, common translations are:

1. one relation for the superclass and one for each subclass, with foreign key referencing the key of the superclass
2. one relation for the whole hierarchy, with nullable attributes
3. one relation per concrete subclass

There is no universally best choice: it depends on the intended use and on the constraints of the application.

Resulting relational schema

Our running example yields:

- `Person(`<u>`id`</u>`, name)`
- `Student(`<u>`id`</u>`, email, advisor)`
- `Teacher(`<u>`id`</u>`, office)`
- `Course(`<u>`id`</u>`, title, ects)`
- `Section(`<u>`course`</u>`, `<u>`sectionNo`</u>`, `<u>`semester`</u>`, room)`
- `Enroll(`<u>`student`</u>`, `<u>`course`</u>`, `<u>`sectionNo`</u>`, `<u>`semester`</u>`, grade)`
- `Teaches(`<u>`teacher`</u>`, `<u>`course`</u>`, `<u>`sectionNo`</u>`, `<u>`semester`</u>`)`

What the translation gives us

The E–R model mainly determines:

- the relations
- their attributes
- their keys
- their foreign keys

But it does not always guarantee that each relation is well normalized.

For that, we need a finer notion: functional dependencies.

A possible SQL realization (1/3)

SQL

```sql
CREATE TABLE Person(
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL
);

CREATE TABLE Teacher(
  id INTEGER PRIMARY KEY REFERENCES Person(id),
  office TEXT
);

CREATE TABLE Student(
  id INTEGER PRIMARY KEY REFERENCES Person(id),
  email VARCHAR(256) UNIQUE,
  advisor INTEGER NOT NULL REFERENCES Teacher(id)
);
```

A possible SQL realization (2/3)

SQL

```
CREATE TABLE Course(
  id VARCHAR(16) PRIMARY KEY,
  title TEXT NOT NULL,
  ects INTEGER NOT NULL CHECK (ects >= 0)
);

CREATE TABLE Section(
  course VARCHAR(16) NOT NULL REFERENCES Course(id),
  sectionNo INTEGER NOT NULL,
  semester VARCHAR(16) NOT NULL,
  room TEXT,
  PRIMARY KEY(course, sectionNo, semester)
);
```

A possible SQL realization (3/3)

SQL

```
CREATE TABLE Enroll(
  student INTEGER REFERENCES Student(id),
  course VARCHAR(16), sectionNo INTEGER, semester VARCHAR(16),
  grade NUMERIC,
  PRIMARY KEY(student, course, sectionNo, semester),
  FOREIGN KEY(course, sectionNo, semester)
    REFERENCES Section(course, sectionNo, semester)
);

CREATE TABLE Teaches(
  teacher INT REFERENCES Teacher(id),
  course VARCHAR(16), sectionNo INTEGER, semester VARCHAR(16),
  PRIMARY KEY(teacher, course, sectionNo, semester),
  FOREIGN KEY(course, sectionNo, semester)
    REFERENCES Section(course, sectionNo, semester)
);
```

Remark on cardinality constraints

Minimum cardinality constraints ($m \geqslant 1$):

- When part of a many-to-one or one-to-one relation: can often be expressed using `NOT NULL` (cf. `AdvisedBy`)
- When part of a weak entity set: implied by the encoding of weak entity sets (cf. `BelongsTo`)
- Otherwise: cannot generally be expressed using only keys and foreign keys (cf. `Teaches`)

Outline

Why functional dependencies?

E–R diagrams are excellent for conceptual modeling.

To reason about redundancy inside a relation, we need a more fine-grained notion:

functional dependencies

These are the basis of normal forms.

Functional dependency

Definition

Let R be a relation schema and let X, Y be sets of attributes of R.
We say that $X \to Y$ is a functional dependency if, in every valid instance of R, any two tuples that agree on X also agree on Y.

Intuition: the values of X determine the values of Y.

Examples of functional dependencies

In relation

`Student(id, name, email, advisor)`

we have:

- `id` $\rightarrow$ `name`, `email`, `advisor`
- `email` $\rightarrow$ `id`, `name`, `advisor` if email is unique

In relation

`Section(course, sectionNo, semester, room)`

we have:

- `course`, `sectionNo`, `semester` $\rightarrow$ `room`

Keys and superkeys

A set of attributes K is a superkey if:

$$K \rightarrow \text{all attributes of } R$$

A candidate key is a minimal superkey.

Examples:

- in `Student(id, name, email, advisor)`, `id` is a candidate key
- if email is unique, `email` is also a candidate key
- in `Section(course, sectionNo, semester, room)`, `(course, sectionNo, semester)` is a candidate key

A deliberately bad relation schema

Consider:

$$\texttt{EnrollmentInfo}(\texttt{student}, \texttt{studentName}, \texttt{email}, \texttt{course}, \texttt{title}, \texttt{ects}, \texttt{teacher}, \texttt{office}, \texttt{grade})$$

Plausible FDs:

$$\begin{aligned} \texttt{student} &\to \texttt{studentName}, \texttt{email} \\ \texttt{course} &\to \texttt{title}, \texttt{ects} \\ \texttt{teacher} &\to \texttt{office} \\ \texttt{student}, \texttt{course} &\to \texttt{grade}, \texttt{teacher} \end{aligned}$$

This relation is highly redundant.

Anomalies in the bad relation schema

With

$$\texttt{EnrollmentInfo}(\texttt{student}, \texttt{studentName}, \texttt{email}, \texttt{course}, \texttt{title}, \texttt{ects}, \texttt{teacher}, \texttt{office}, \texttt{grade})$$

we get:

- update anomaly: changing a course title requires many updates
- insertion anomaly: cannot add a course before some student enrolls
- deletion anomaly: deleting the last enrollment in a course may delete information about the course

Outline

Normalization

Normalization consists in decomposing relations to reduce redundancy, while preserving meaning.

We will study:

- 1NF
- 2NF
- 3NF
- BCNF
- 4NF

First Normal Form (1NF)

A relation schema is in 1NF if every attribute stores atomic values.

Not in 1NF:

`Student(id, name, emails)`

where `emails` stores a set or list.

A better design is:

- `Student(`<u>`id`</u>`, name)`
- `StudentEmail(`<u>`student`</u>`, `<u>`email`</u>`)`

Second Normal Form (2NF)

A relation is in 2NF if:

- it is in 1NF
- every attribute that is not part of a candidate key depends on the whole of every candidate key

In other words: no partial dependency on part of a composite key.

Example: a relation not in 2NF

Consider:

`EnrollBad(student, course, studentName, courseTitle, grade)`

with key `(student, course)` and FDs:

$$\texttt{student} \to \texttt{studentName}$$
$$\texttt{course} \to \texttt{courseTitle}$$
$$\texttt{student}, \texttt{course} \to \texttt{grade}$$

This relation is not in 2NF.

Decomposition to reach 2NF

Decompose:

`EnrollBad(student, course, studentName, courseTitle, grade)`

into:

- `Student(`<u>`student`</u>`, studentName)`
- `Course(`<u>`course`</u>`, courseTitle)`
- `Enroll(`<u>`student`</u>`, `<u>`course`</u>`, grade)`

Now every non-key attribute depends on the full key.

Third Normal Form (3NF)

A relation is in 3NF if:

- it is in 2NF
- non-key attributes do not depend transitively on a key

Equivalent formulation: for every nontrivial FD $X \to A$, either:

- X is a superkey, or
- A is an attribute that is part of a candidate key

Example: a relation not in 3NF

Consider:

$$\texttt{TeacherInfo(teacher, name, office, building)}$$

with:

$$\texttt{teacher} \rightarrow \texttt{name}, \texttt{office}$$

$$\texttt{office} \rightarrow \texttt{building}$$

Then:

$$\texttt{teacher} \rightarrow \texttt{office} \rightarrow \texttt{building}$$

So `building` depends transitively on the key `teacher`.

Decomposition to reach 3NF

Decompose:

`TeacherInfo(teacher, name, office, building)`

into:

- `Teacher(`<u>`teacher`</u>`, name, office)`
- `Office(`<u>`office`</u>`, building)`

Boyce–Codd Normal Form (BCNF)

Definition

A relation schema is in BCNF if for every nontrivial FD $X \to Y$, X is a superkey.

Thus every determinant must itself be a key.

Example: 3NF but not BCNF

Consider

$$R(\texttt{student}, \texttt{course}, \texttt{teacher})$$

with

$$(\texttt{student}, \texttt{course}) \to \texttt{teacher}$$

$$\texttt{teacher} \to \texttt{course}$$

Candidate keys:

$$(\texttt{student}, \texttt{course}),\ (\texttt{student}, \texttt{teacher})$$

The dependency

$$\texttt{teacher} \to \texttt{course}$$

violates BCNF because `teacher` is not a superkey, but the relation is still in 3NF since `course` is part of a candidate key.

BCNF decomposition algorithm

Goal: decompose a relation into BCNF.

Starting from a relation schema R:

1. If R is already in BCNF, stop.
2. Otherwise, find a nontrivial FD $X \to Y$ that violates BCNF (so X is not a superkey).
3. Decompose R into:

$$R_1 = X \cup Y \qquad R_2 = R \setminus (Y \setminus X)$$

4. Apply the procedure recursively to R_1 and R_2.

This decomposition is lossless, but may fail to preserve all dependencies.

Example of BCNF decomposition

Consider

$$R(\texttt{student}, \texttt{course}, \texttt{teacher})$$

with

$$(\texttt{student}, \texttt{course}) \to \texttt{teacher} \qquad \texttt{teacher} \to \texttt{course}$$

The FD

$$\texttt{teacher} \to \texttt{course}$$

violates BCNF, since teacher is not a superkey.

So decompose into:

$$R_1(\texttt{teacher}, \texttt{course}) \qquad R_2(\texttt{student}, \texttt{teacher})$$

Now both relations are in BCNF.

BCNF may lose dependency preservation

Consider again

$$R(\texttt{student}, \texttt{course}, \texttt{teacher})$$

with

$$(\texttt{student}, \texttt{course}) \to \texttt{teacher} \qquad \texttt{teacher} \to \texttt{course}$$

After BCNF decomposition, we obtain:

$$R_1(\texttt{teacher}, \texttt{course}) \qquad R_2(\texttt{student}, \texttt{teacher})$$

Now the dependency

$$(\texttt{student}, \texttt{course}) \to \texttt{teacher}$$

is not present in either relation alone.

So the decomposition is lossless, but not dependency-preserving.

Mnemonic: "So help me Codd"

A classical mnemonic for normalization is:

"The key, the whole key, and nothing but the key, so help me Codd."

Interpretation:

- the key: attributes depend on a key
- the whole key: no dependency on only part of a composite key (idea behind 2NF)
- nothing but the key: no transitive dependency through non-key attributes (idea behind 3NF)
- so help me Codd: a more complete formalization is Boyce–Codd Normal Form (BCNF)

Fourth Normal Form (4NF)

4NF deals with multivalued dependencies.

Intuition: sometimes two independent multi-valued facts are stored together, creating a cartesian-product redundancy.

Example:

```
StudentFact(student, language, hobby)
```

Example: a relation not in 4NF

Suppose a student may speak several languages and have several hobbies, independently.

Then storing

`StudentFact(student, language, hobby)`

creates one tuple for every combination of a language and a hobby.

Decompose into:

- `StudentLanguage(`<u>`student`</u>`, `<u>`language`</u>`)`
- `StudentHobby(`<u>`student`</u>`, `<u>`hobby`</u>`)`

Starting point

Consider again:

$$\texttt{EnrollmentInfo}(\texttt{student}, \texttt{studentName}, \texttt{email}, \texttt{course}, \texttt{title}, \texttt{ects}, \texttt{teacher}, \texttt{office}, \texttt{grade})$$

Assume:

$$\begin{aligned} \texttt{student} &\to \texttt{studentName}, \texttt{email} \\ \texttt{course} &\to \texttt{title}, \texttt{ects} \\ \texttt{teacher} &\to \texttt{office} \\ \texttt{student}, \texttt{course} &\to \texttt{grade}, \texttt{teacher} \end{aligned}$$

Why it is not in 2NF

If the key is (`student`, `course`), then:

- `student` $\rightarrow$ `studentName`, `email`
- `course` $\rightarrow$ `title`, `ects`

So some non-key attributes depend on only part of the key.

Therefore the relation is not in 2NF.

After 2NF decomposition

Decompose into:

- `Student(`<u>`student`</u>`, studentName, email)`
- `Course(`<u>`course`</u>`, title, ects)`
- `EnrollmentTmp(`<u>`student`</u>`, `<u>`course`</u>`, teacher, office, grade)`

Better, but still:

$$\texttt{teacher} \rightarrow \texttt{office}$$

inside `EnrollmentTmp`.

Why the intermediate result is not in 3NF

In

`EnrollmentTmp(student, course, teacher, office, grade)`

with key `(student, course)`, we have:

$$\texttt{student}, \texttt{course} \rightarrow \texttt{teacher} \rightarrow \texttt{office}$$

So `office` depends transitively on the key.

Hence the relation is not in 3NF.

Final decomposition

We obtain:

- `Student(`<u>`student`</u>`, studentName, email)`
- `Course(`<u>`course`</u>`, title, ects)`
- `Teacher(`<u>`teacher`</u>`, office)`
- `Enrollment(`<u>`student`</u>`,` <u>`course`</u>`, teacher, grade)`

This is much closer to what we would naturally design from the E–R model.

Outline

What makes a good decomposition?

A decomposition should ideally satisfy:

- lossless join: joining the decomposed relations gives back exactly the original information
- dependency preservation: the important FDs can still be enforced locally

3NF or BCNF?

- BCNF removes more redundancy
- but a BCNF decomposition may fail to preserve all dependencies
- 3NF is a common compromise:
  - still avoids most redundancy
  - often preserves dependencies better

In practice, many schemas aim at something close to BCNF, but 3NF is often already very good.

Outline

Summary

- E–R diagrams capture the conceptual structure of the domain
- They can be translated systematically into a relational schema
- Functional dependencies help reason about redundancy
- Normal forms provide increasingly strong criteria for good schema design

Normal forms, to analyze an existing schema:

- 1NF: atomic values
- 2NF: no partial dependency on a composite key
- 3NF: no transitive dependency of non-key attributes
- BCNF: every determinant is a superkey
- 4NF: no unwanted multivalued dependencies

Design advice

In practice:

- start from the application semantics
- model entities and relationships clearly
- identify keys and constraints early
- normalize enough to avoid anomalies
- but also keep an eye on usability and future queries

Good schema design is both a theoretical and a practical skill.

Licensing