SD202: Databases

Schema design with Entity-Relationship Diagrams

Antoine Amarilli

Télécom Paris

Introduction

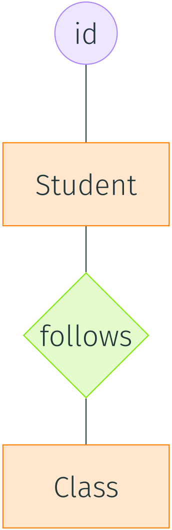
- We have seen the **general picture** of relational databases
- We have seen the **SQL language**
  - Installing a relational database
  - Creating tables
  - Filling the tables with data
  - Querying the tables (in the lab last week)
- Today's goal: **Find out which tables we should create**
  - Depending on the application

Overview

To decide which tables you should create for an application, you should:

- Be very clear about what the goal of the application is!
  - Do not overlook this step!
  - Often, schema design asks many tricky questions: which data to manipulate, which assumptions are made, what should be possible or not...
  - On large projects, the database schema is often the central reference on which data the application manages
- Formalize the logical schema, describing abstractly which data is managed
- Possibly, think about the operations that will be supported on this data (e.g., business processes)
- Implement the logical schema as a physical schema, i.e., concrete table definitions in a database
- Check the resulting schema for problems (normalization)

Entity-Relationship model



- Entity-Relationship diagrams are a general model to present the logical schema of your application
- This is not pure science! necessarily a bit handwavy, and many variants/notations
  - Relates to object-oriented programming
  - Specifically, to the Unified Modeling Language (UML)
- Basic notions:
  - Entities (and entity-types), describing the "objects"
  - Relationships (and relationship-types), describing the "relationships" between them

Goals

What are the goals of a good schema design?

- Being **complete**, i.e., can represent everything that is needed
- Being **clear** to developers and as **simple** as possible
- Being **precise**: clear how to map actual business needs to data
- Not being too **broad**, i.e., correctly reflect constraints that are assumed
- Avoiding **redundancy**: make sure every data item is in one place
- Ensuring good **performance** (often linked to simplicity)

Not being complete vs being too broad

Say you have customers, identified with an ID, having a name and a phone number. Here are three options, which is the best?

One table:	Two tables:	Two tables:
• Customer(id, name, phone)	• Customer(id, name) • Phone(id, phone)	• Customer(id, name) • Phone(id, phone)

Not being complete vs being too broad

Say you have **customers**, identified with an ID, having a name and a phone number. Here are three options, which is the best?

One table: • Customer(id, name, phone)	Two tables: • Customer(id, name) • Phone(id, phone)	Two tables: • Customer(id, name) • Phone(id, phone)
Fine if **every customer has exactly one phone number** (or none, if NULL is OK)	Fine if **customers can have zero, one, or many phone numbers**	Fine if **customers can have one phone number or none**

Sometimes, there are **multiple possible choices**!

Problems with redundancy

Student

id	name	email	email_type
41	John Student	john.student@telecom-paris.fr	pro
41	John Student	johndu91@hotmail.fr	perso
42	Jane Student	jane.student@telecom-paris.fr	pro

Problems with redundancy

Student

id	name	email	email_type
41	John Student	john.student@telecom-paris.fr	pro
41	John Student	johndu91@hotmail.fr	perso
42	Jane Student	jane.student@telecom-paris.fr	pro

- Say we want to **rename** “John Student” to “Jean Student”
- We must do it in **all tuples**!

Problems with redundancy

Student

id	name	email	email_type
41	Jean Student	john.student@telecom-paris.fr	pro
41	John Student	johndu91@hotmail.fr	perso
42	Jane Student	jane.student@telecom-paris.fr	pro

- Say we want to **rename** “John Student” to “Jean Student”
- We must do it in **all tuples**!
- Otherwise, **inconsistent!** (update anomaly)

Problems with redundancy

Student

id	name	email	email_type
41	Jean Student	john.student@telecom-paris.fr	pro
41	John Student	johndu91@hotmail.fr	perso
42	Jane Student	jane.student@telecom-paris.fr	pro

- Say we want to **rename** "John Student" to "Jean Student"
- We must do it in **all tuples**!
- Otherwise, **inconsistent!** (update anomaly)

Other problems

Student

id	name	email	email_type
41	John Student	john.student@telecom-paris.fr	pro
41	John Student	johndu91@hotmail.fr	perso
42	Jane Student	jane.student@telecom-paris.fr	pro

- We cannot insert a student who does not have an email address (insert anomaly)
- If we remove all email addresses of a student then we lose all information about the student (delete anomaly)

A better schema design

Solution: use two tables!

Student

id	name
41	John Student
42	Jane Student

Email

id	email	type
41	john.student@telecom-paris.fr	pro
41	johndu91@hotmail.fr	perso
42	jane.student@telecom-paris.fr	pro

Basic Entity Relationship notions

Translating an ER diagram to a schema

Basic Entity Relationship notions

Entities and entity-types

- An entity is a concrete object that we will have to manage. Examples:
  - A person, a company, e.g., a customer, a supplier
  - An actual object
  - A location, a house, a building, a room...
  - A file, a dataset, a data item
  - An event
  - An order, a request...
- Entities have attributes, e.g., name, size, date of birth, color, geographic coordinates, path, date, etc.
- An entity-type is a type of entity, e.g., a “class” in software engineering
  - Customer, Supplier, Location, File, Order, etc.
- All entities in the same entity-type have the same attributes

Composite attributes

The attributes of an entity-type can be sometimes subdivided, e.g., “address” becomes (in France) something like:

- number
- street
- extra_info
  - building
  - floor
  - apartment_number
- city
- post_code

These are called composite attributes

Attribute types

When we have an attribute we must think about its type:

- String (which language? which text encoding?)
- Integer
- Decimal
- Date/Time
- Geographical coordinates, etc.

We must also think about:

- The domain of the attribute (which values are allowed?)
- Whether the attribute is mandatory (can we have no value?)
- Which attribute(s) are the key that uniquely identifies the entity
    - There cannot be two different entities with the same values on all attributes
    - Either add the missing attributes, or add a surrogate key attribute

Two “special” kinds of attributes

- Derived attributes: can be deduced from other attributes
    - e.g., an “age” attribute can be deduced from a “date_of_birth” attribute

→ We will often not store the derived attribute, but compute it on the fly

Two “special” kinds of attributes

- Derived attributes: can be deduced from other attributes
  - e.g., an “age” attribute can be deduced from a “date_of_birth” attribute

  → We will often not store the derived attribute, but compute it on the fly

- Multi-valued attributes: there can be more than one value
  - e.g., email address, phone number...

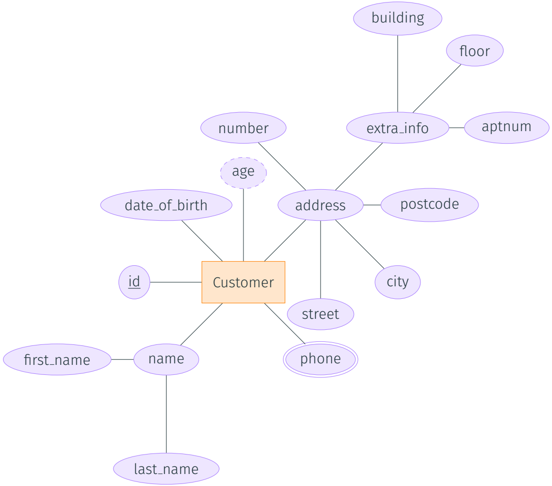
  → We will often store these attributes in a separate table

Drawing entities and attributes

- Entities (formally entity-types) are often drawn in a **rectangular box**
- Attributes of the entity-type can be **oval nodes**, or **lines in the box**

Customer

id
name
  first_name
  last_name
date_of_birth
age()
address
  number
  street
  extra_info
    building
    floor
    apartment_number
  city
  post_code
{ phone }

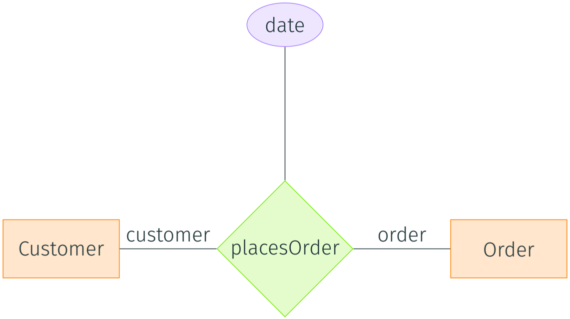


Relationships

- A **relationship** connects two or more concrete entities
  - → e.g., “Customer 42 placed order 45”
  - → e.g., “Professor Patricia supervised student John on topic 44”
- A **relationship-type** is a set of relationships with the same attributes and connecting the same entity-types
  - → e.g., placesOrder, advises
- The possible participating entities are called **roles**
  - → customer (Customer), order (Order)
  - → advisor (Professor), advisee (Student), topic (Topic)
  - → Can have the same entity-type twice, e.g., “isMentoring” with mentor (Employee) and mentee (Employee)
- A relationship (and relationship-type) can also have **attributes**
  - → e.g., date

Drawing relationships

- Relationships (formally relationship-types) are often drawn in a diamond box
- Relationships are connected to the entities that are involved in them
- Attributes are connected to the relationship
- Roles are written on the edges connecting the relationship and entity



Cardinality constraints

For a given entity-type in a relationship-type, there can be **cardinality constraints** to describe if an entity can be:

- In **no** relationship
- In **one** relationship
- In **multiple** relationships

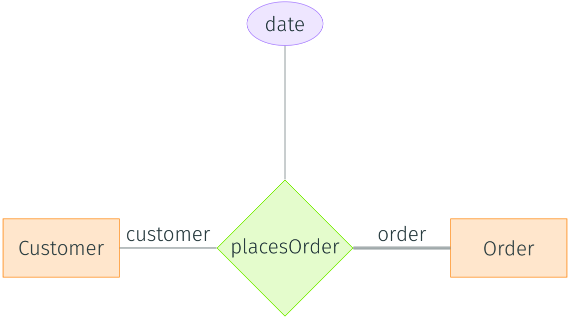
Beware of **confusion**:

- A given relationship always has **one entity** of each role!
- This is about the **number of relationships** to which a given entity participates
- Cardinality constraints apply **per relationship (type)**, not across all relationships

Drawing cardinality constraints: partial/total

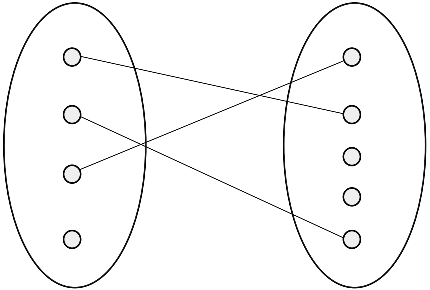
Indicate whether o is acceptable or not:

- Total participation: o is not acceptable, every entity must be in a relationship
  - → Represented by a **double line** in an ER diagram
- Partial participation (default): o is acceptable, some entities are not in a relationship



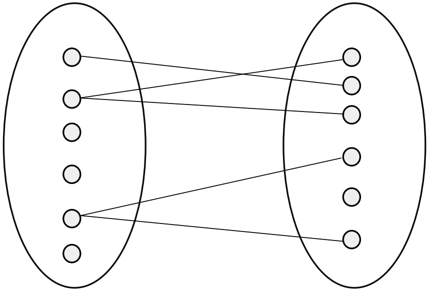
Drawing cardinality constraints: one/many to one/many (1)

One-to-one



Relation is functional and injective

One-to-many

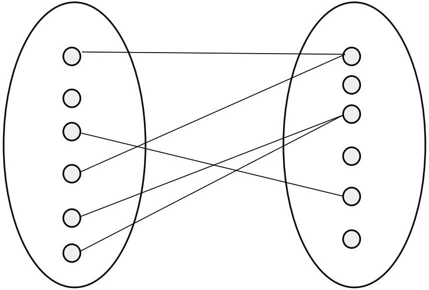


Relation is injective but not functional

Can you give examples?

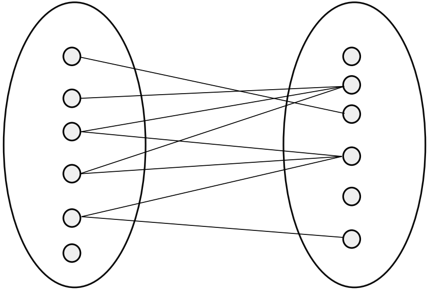
Drawing cardinality constraints: one/many to one/many (2)

Many-to-one



Relation is functional but not injective

Many-to-many

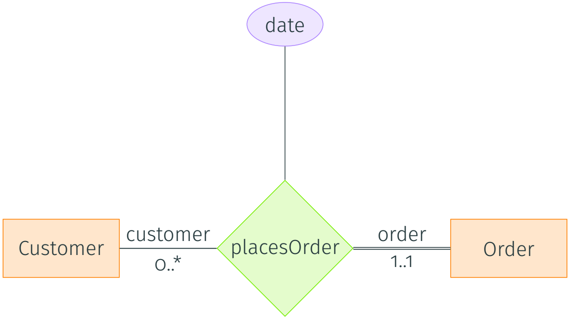


Relation is arbitrary and not functional

- These relation types are everywhere
- There are arrow notations for these cardinality constraints, but not universal

General cardinality constraints

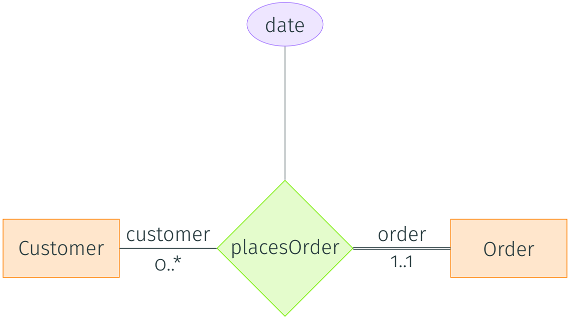
For each role, write below the role the minimal and maximal number of relationships to which an entity can participate, with "*" meaning "no limit"



- This indicates both total/partial and one/many to one/many
- Exercise: which kind of relation is this?

General cardinality constraints

For each role, write below the role the minimal and maximal number of relationships to which an entity can participate, with "*" meaning "no limit"



- This indicates both total/partial and one/many to one/many
- Exercise: which kind of relation is this? Beware, it is one-to-many

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  - → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1
- Room reservations are associated to a room

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  - → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1
- Room reservations are associated to a room
  - → The constraint on RoomReservation is 1..1 or 1..* and on Room it is 0..1 (at some point in time) or 0..* (in general)

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  - → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1
- Room reservations are associated to a room
  - → The constraint on RoomReservation is 1..1 or 1..* and on Room it is 0..1 (at some point in time) or 0..* (in general)
- Students are associated in student groups

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1
- Room reservations are associated to a room
  → The constraint on RoomReservation is 1..1 or 1..* and on Room it is 0..1 (at some point in time) or 0..* (in general)
- Students are associated in student groups
  → The constraint on Student depends on the semantics, the constraint on Groups is 1..*

Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  - → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1
- Room reservations are associated to a room
  - → The constraint on RoomReservation is 1..1 or 1..* and on Room it is 0..1 (at some point in time) or 0..* (in general)
- Students are associated in student groups
  - → The constraint on Student depends on the semantics, the constraint on Groups is 1..*
- Each employee is managed by an employee

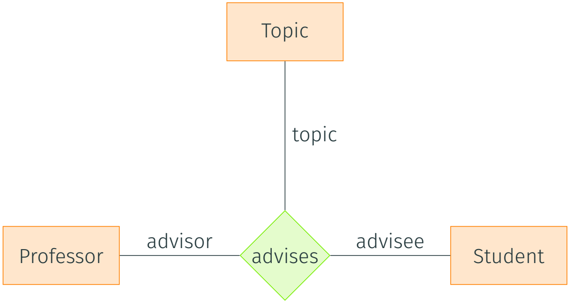
Cardinality exercises

What are the cardinality constraints on the following relations:

- Classes are composed of lectures
  - → The constraint on Class is 0..* or 1..* and on Lecture it is 1..1 or 1..*
- Class occurrences must have a room reservation
  - → The constraint on ClassOccurrence is 1..1 or 1..* and on RoomReservation it is 1..1 or 0..1
- Room reservations are associated to a room
  - → The constraint on RoomReservation is 1..1 or 1..* and on Room it is 0..1 (at some point in time) or 0..* (in general)
- Students are associated in student groups
  - → The constraint on Student depends on the semantics, the constraint on Groups is 1..*
- Each employee is managed by an employee
  - → The constraint on the “manager” role is 0..*, the constraint on the “managee” role is 1..1 (or is it?)

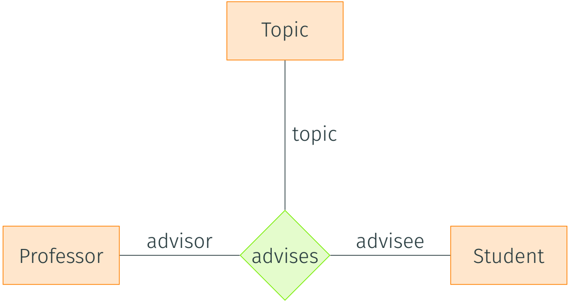
Cardinality constraints and non-binary relations

In a ternary relationship between professor, student, and topic...



Cardinality constraints and non-binary relations

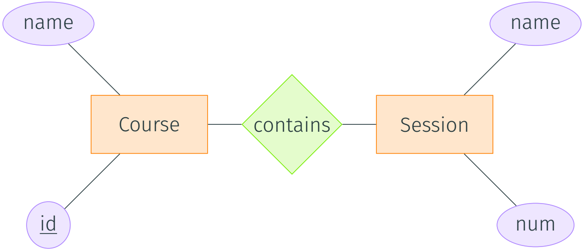
In a ternary relationship between professor, student, and topic...



- Cardinality constraints cannot express that every student on every topic is supervised by at most one professor

Weak entities (motivation)

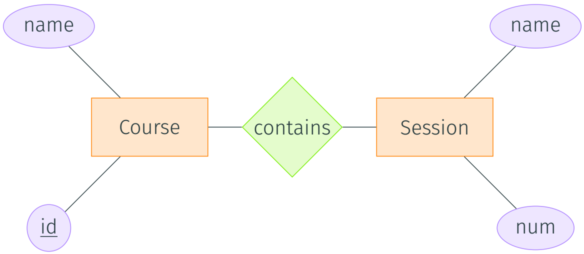
- Some entities only make sense **in the context of other entities**. For instance:
  - “A course has a id, a name, and has several numbered sessions having a title”



Do you see the problem?

Weak entities (motivation)

- Some entities only make sense in the context of other entities. For instance:
  - “A course has a id, a name, and has several numbered sessions having a title”

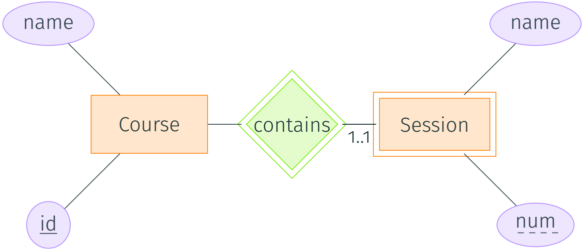


Do you see the problem?

- The Session entities do not have a key! We can have sessions with the same name and number in different courses
- Each Session is unique in the context of a course

Weak entities (solution)

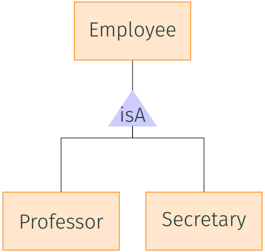
- The Session entity is a **weak entity**
- The "contains" relationship is the **identifying relationship** of Session
- These are materialized using **double lines**



The **key** of the weak entity will be the **key** of the **other entity in the identifying relationship** plus a set of attributes called **discriminator** which is dash-underlined

Specialization and Generalization

- A special kind of relationship: is-A
  - Every professor is an employee
  - Every employee is a person
  - Lab sessions and lectures are classes



- We could represent, e.g., each professor with two entities (e.g., a professor entity and an employee entity), and have an is-A relationship between the two
- Sometimes more legible to write them with an "isA triangle"
- The subclass inherits attributes from the superclass
- Related to inheritance in object-oriented programming

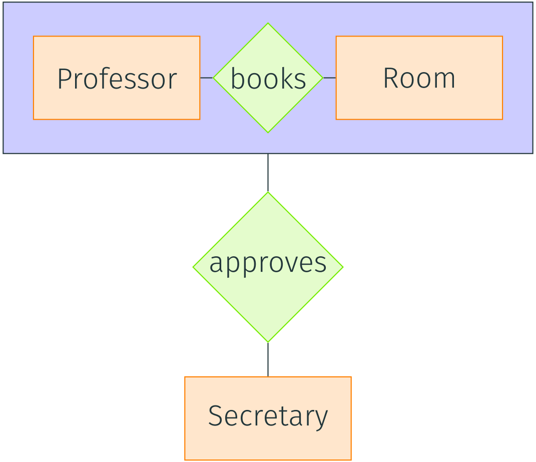
- Specialization: top-down design process subdividing entities in subclasses
- Generalization: bottom-up design process regrouping entities sharing common attributes

Constraints on specialization/generalization

- Completeness:
  - Complete: “each employee is either a professor or a secretary”
  - Not complete: “there can also be other kinds of employees”
- Disjointness:
  - Disjoint: “an employee cannot be both a professor and a secretary”
  - Not disjoint: “an employee can be both”

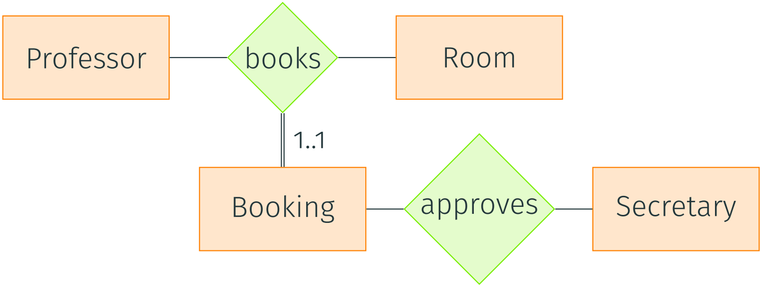
Aggregation

In complex cases, we may want to handle relationships as **entities** in another relationship



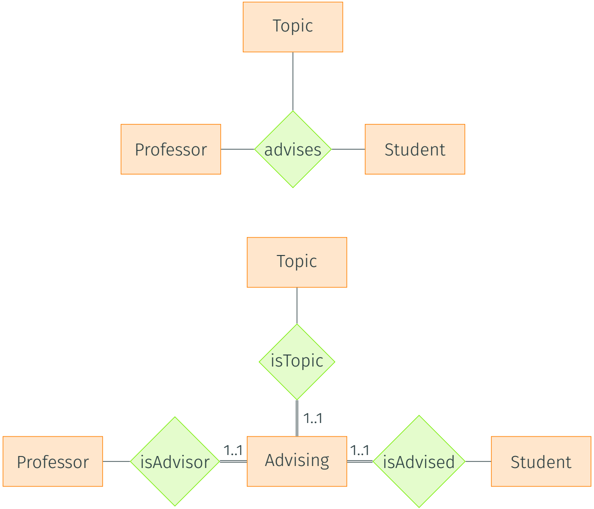
Eliminating aggregation with reification

- Introduce a **new entity-type** for the relationship with a **surrogate key**
- Introduce it as a **member** of the relationship
- Use the **entity** in the other relationship



Reification

Reification can also transform **non-binary** relationships into **binary relationships**



Basic Entity Relationship notions

Translating an ER diagram to a schema

Translating an ER diagram to a schema

Translating entities

Create one table per entity-type with all of the attributes

Customer

- <u>id</u>
- name
  - first_name
  - last_name
- date_of_birth
- age()

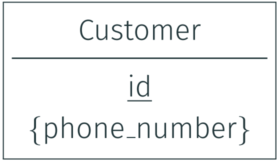
becomes

```sql
CREATE TABLE Customer(
  id SERIAL PRIMARY KEY,
  name_first_name VARCHAR,
  name_last_name VARCHAR,
  date_of_birth DATE);
```

- We drop the attribute hierarchy (we can remove prefixes if unambiguous)
- Derived attributes are not stored but computed on the fly
- See next slide for multi-valued attributes

Translating multi-valued attributes

- Add an **extra table** with a **foreign key** for multi-valued attributes:
- Can also handle **extra information**

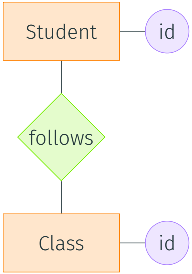


becomes

```
CREATE TABLE Customer(id SERIAL PRIMARY KEY);
CREATE TABLE Customer_phone_number(
  customer INT REFERENCES Customer,
  phone_number VARCHAR
  -- can add, e.g., phone_number_type VARCHAR
  );
```

Basic translation of relationships

Most naive idea: create one table per relationship



becomes

```
CREATE TABLE Follows(
  student INT REFERENCES Student,
  class INT REFERENCES Class);
```

This is the proper solution, e.g., for many-to-many relationships

Key choices when translating relationships

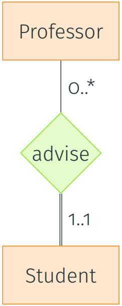
When we create a new table for a (binary) relationship, what is the key?

- In the general case of many-to-many relationships, the pair of identifiers
- For one-to-one, one-to-many, many-to-one relationships, an identifier on the "many" side is enough
- Note: always possible to create a surrogate key

Simpler translation of relationships (1)

In some cases we can **avoid** creating an additional table:

- **One-to-many or many-to-one relationship**: store the other objects and relationship attributes in attributes of the "many" side



becomes

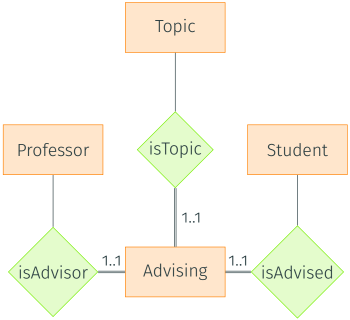
```sql
CREATE TABLE Professor (...);
CREATE TABLE Student(
  id SERIAL PRIMARY KEY,
  advisor INT REFERENCES Professor,
  -- add attributes of the advise relationship
);
```

Note: if the relation is not **total** on the "many" side, then the corresponding attributes may be **null**

Simpler translation of relationships (2)

In some cases we can **avoid** creating an additional table:

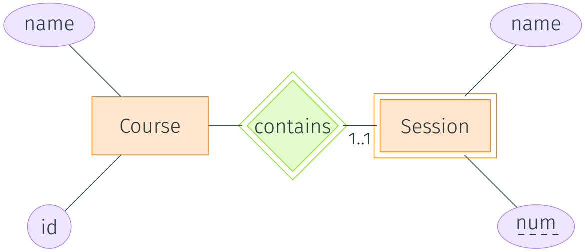
- **Total one-to-one relationships**: merge in one table




```sql
CREATE TABLE Advising(
  advisor INT REFERENCES Professor,
  advisee INT REFERENCES Student,
  topic INT REFERENCES Topic);
```

Translating weak entities

When translating **weak entities**, add a column or columns for the key of the other entity in the **identifying relationship**




```
CREATE TABLE Course(
  id SERIAL PRIMARY KEY,
  name VARCHAR);
CREATE TABLE Session(
  course INT REFERENCES Course,
  num INT,
  name VARCHAR,
  PRIMARY KEY (course, num));
```

The **key** is constituted of this foreign key plus the **discriminator**

Handling specialization/generalization

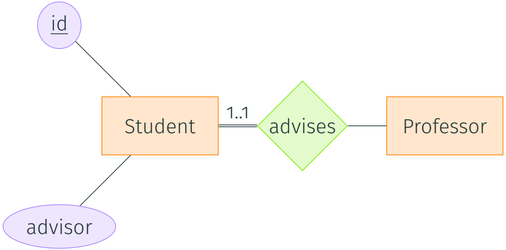
Say Employee has two subclasses, Professor and Secretary. Several options:

- Forget about Employee, and create tables Professor and Secretary, each containing the common attributes
  - → Good for disjoint and complete inheritance (every Employee is a Professor or Secretary)
- Create tables Employee, Professor, and Secretary, with the common attributes in Employee
  - → Each Professor, and each Secretary, also has a record in Employee with the common attributes
- Create table Employee, put the Professor and Secretary attributes in this table
  - → Some of these attributes will be NULL

Eliminating redundancy

What should be removed at the end of the process?

- **Useless relations**, e.g., created for relationships that can be represented with a foreign key instead
- **Redundant attributes**, e.g., that are also present in a relationship
  - For instance, if students are advised by professors, and this is represented both as a relationship and an attribute



Sources

- Database System Concepts, Seventh Edition
  https://www.db-book.com/db7/slides-dir/index.html
- Wikipedia
  https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
- https://www.tutorialspoint.com/dbms/er_diagram_representation.htm
- https://www.cs.uct.ac.za/mit_notes/database/htmls/chp07.html#mapping-specializationgeneralization-to-relational-tables