Session 3/4 – notes

Preparation

During the two lectures this week, we'll practice *ER modeling*, which is one way to decide what tables we want to use in our database (in a few weeks time we'll see a completely different way of doing the same thing).

We had some preparatory notes here – and will solve them during the lectures.

UML notation for ER modeling

We started the lecture with a very short recapitulation of what we talked about last week (SQL queries). When we wrote our queries, the database design was given, this week we'll see the first of two ways of coming up with proper database designs ourselves.

An ER model is a conceptual model of some domain, it describes the objects (*entity sets*) in the model, and their relationships. There are (unfortunately) *many, many* standards for drawing diagrams of our models, some of the most popular are:

- UML class diagrams – that's what we're going to use
- Crow's feet notation
- Chen's notation
- ...

UML is becoming increasingly popular for this purpose, and it's easy to learn the small subset of UML's class diagrams we'll need – it's described in the slides we used. Since we're using class diagrams to draw our models, we're sometimes going to use the word *class* instead of *entity set*, but it's essentially the same (to make things even more confusing, the entity sets can be translated into *relations*, which in turn will become *tables* in our database) – we'll talk more about this on Thursday.

A first example

One of the most important aspects of modeling is to decide the granularity of our model. For the student application problem, we have some obvious entity sets:

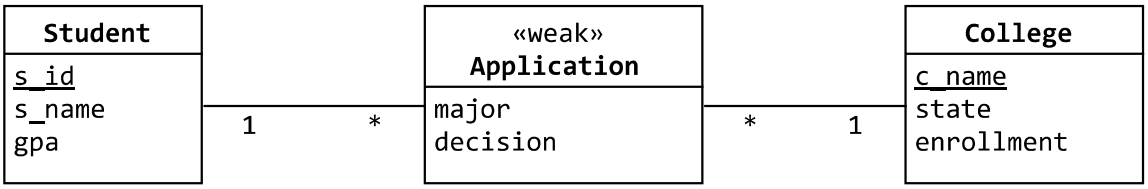
- `Student`

- `College`
- `Application`

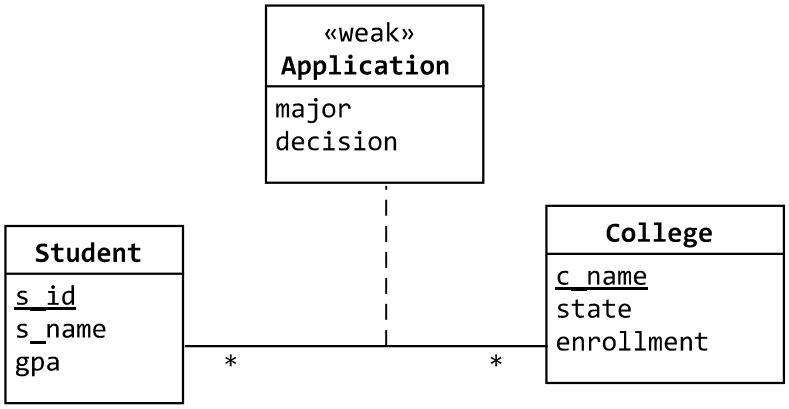
These correspond to the tables we saw during lecture 2. But we could have had more – some candidates are:

- `State`: here we put the state code ("CA" for California), the population, and the name of the capital.
- `Major`: to keep information about the various majors, and also to make it more obvious which majors there are (now they're only represented as strings, and thus prone to misspellings).
- `Program`: here we combine a specific college and a major (during the lecture we saved the major as an attribute in this entity set, but we could have associated a `Program` with both a `College` (as we did), and a `Major`).

We used UMLet to draw some ER-diagrams, beginning with:



As we'd seen on a slide, we could also draw `Application` as an *association class*:

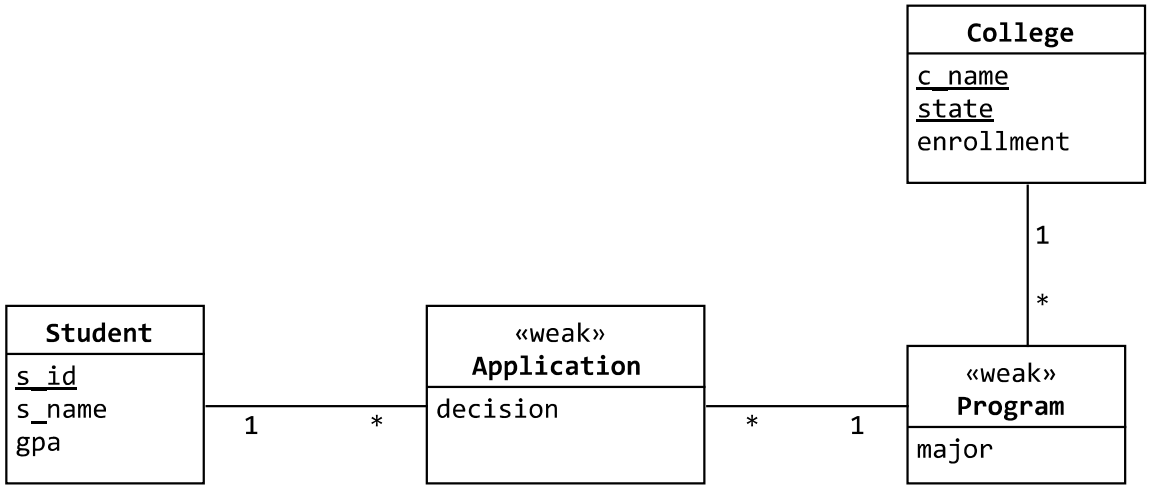


Initially we had the attributes `s_id` and `c_name` in our `Application` class, but removed them as they are already present in the diagram, as the associations from `Application` to `Student`

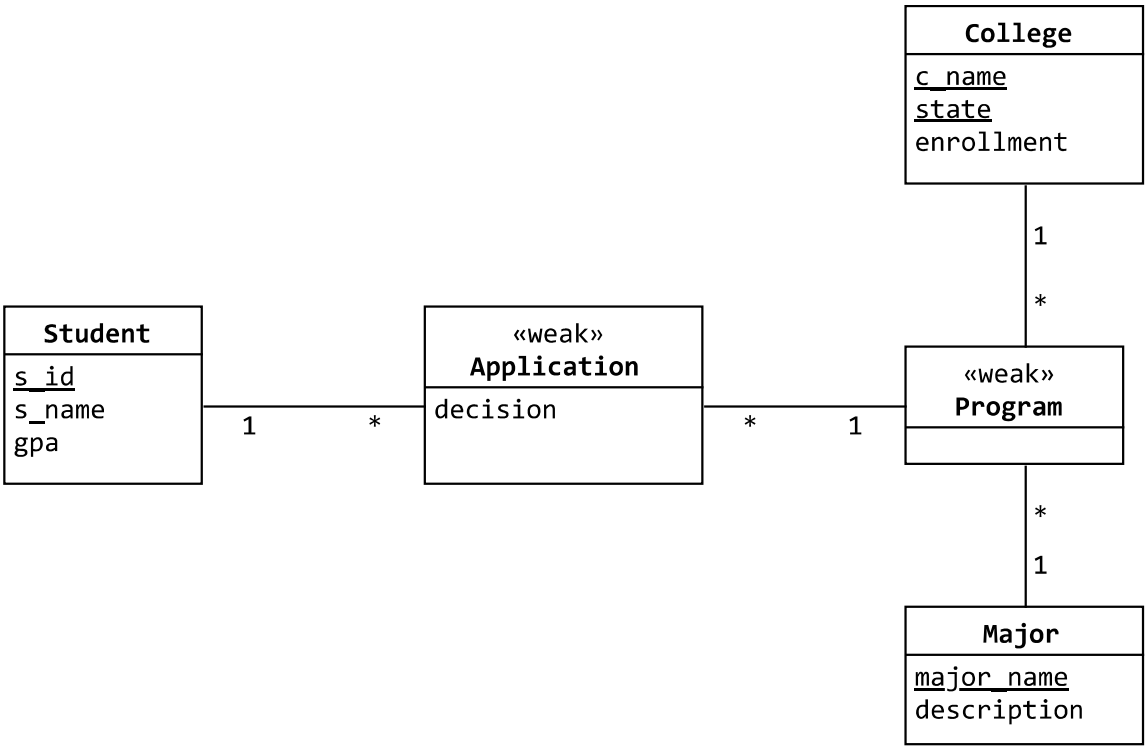
and from `Application` to `College`, respectively.

One problem with the tables we saw during lecture 2 is that we can't necessarily see all 'programs' offered, i.e., we can see the combinations of `College` and `Major` for which there are applications, but there might be others which we're now unable to save in our tables (this is called a deletion anomaly, and we'll address it in week 4).

So we tinkered with adding 'programs' which combines a major and a college, we ended up with:



Here we could also have used a separate entity set for our majors, and have something like:



Observe that we've removed the attribute `major` from `Application` in the two final diagrams, it will be in the `Program` which the application links to (the `Application` table will have what's called a *foreign key* to the `Program` table, but we'll talk more about that on Thursday).

Keys

We then had our first talk about keys – we'll have some more in the coming weeks.

I said that a *key* is a *minimal* set of columns/attributes which we can use to uniquely identify a specific row in a table. The word 'minimal' here means that nothing can be removed from the key for it to remain a key – *it does **not** mean there can't be other, shorter keys with other attributes*.

At its core, this is a very simple definition, but it sometimes creates confusion.

- If one set of attributes uniquely identifies the rows of a table, then adding some attributes to this set 'doesn't hurt', we're still uniquely identifying the rows. But now the set is no longer minimal, it's called a *superset* (it may sound more impressive than just a 'key', but is in fact less interesting). The term superset is defined to also include 'proper

keys', i.e., minimal keys – this can be a bit confusing, but it simplifies some definitions which we'll see later.
- In a table there can potentially be more than one set of attributes which could act as a (minimal) key, we use the term *candidate key* to denote each one of these keys.
- If we have more than one candidate key, we usually pick one of them to be our *primary key* (and in case there is ony one key, that's our primary key). If we define a primary key in a table (and normally we should), the database will not allow duplicates – if we try to insert a new row whose key attributes are the same as the ones in an existing row, we'll get an error.

Some of the slides we used describes keys.

We then looked at the following example: Let's say we have a table with our childrens classmates – they have:

- a first name: `first_name`
- a last name: `last_name`
- an address: `address`
- a phone number: `phone`

How can we uniquely identify a classmate using these attributes? There could be several children with the same first name, and several children with the same last name, there might even be children sharing both first and last names. So neither `first_name`, `last_name` or the tuple `(first_name, last_name)` are keys. If there are siblings in the class, the `address` isn't unique either (and adding `last_name` doesn't help in this case). But the combination of `address` and `first_name` should be unique, no (sane) parent would give two of their children the same first name. Assuming the phone number could be a family phone, it could be shared by siblings, so it doesn't work as a key by itself, but the tuple `(phone, first_name)` should be unique.

So, here we have two possible keys, either `(address, first_name)` or `(phone, first_name)` works – in our table we can choose any of them as our primary key, and we can still use the other key when searching for someone. Adding the last name doesn't narrow down our search in any way, so `(address, first_name, last_name)` and `(phone, first_name, last_name)` are superkeys (as is `(address, phone, first_name, last_name)`).

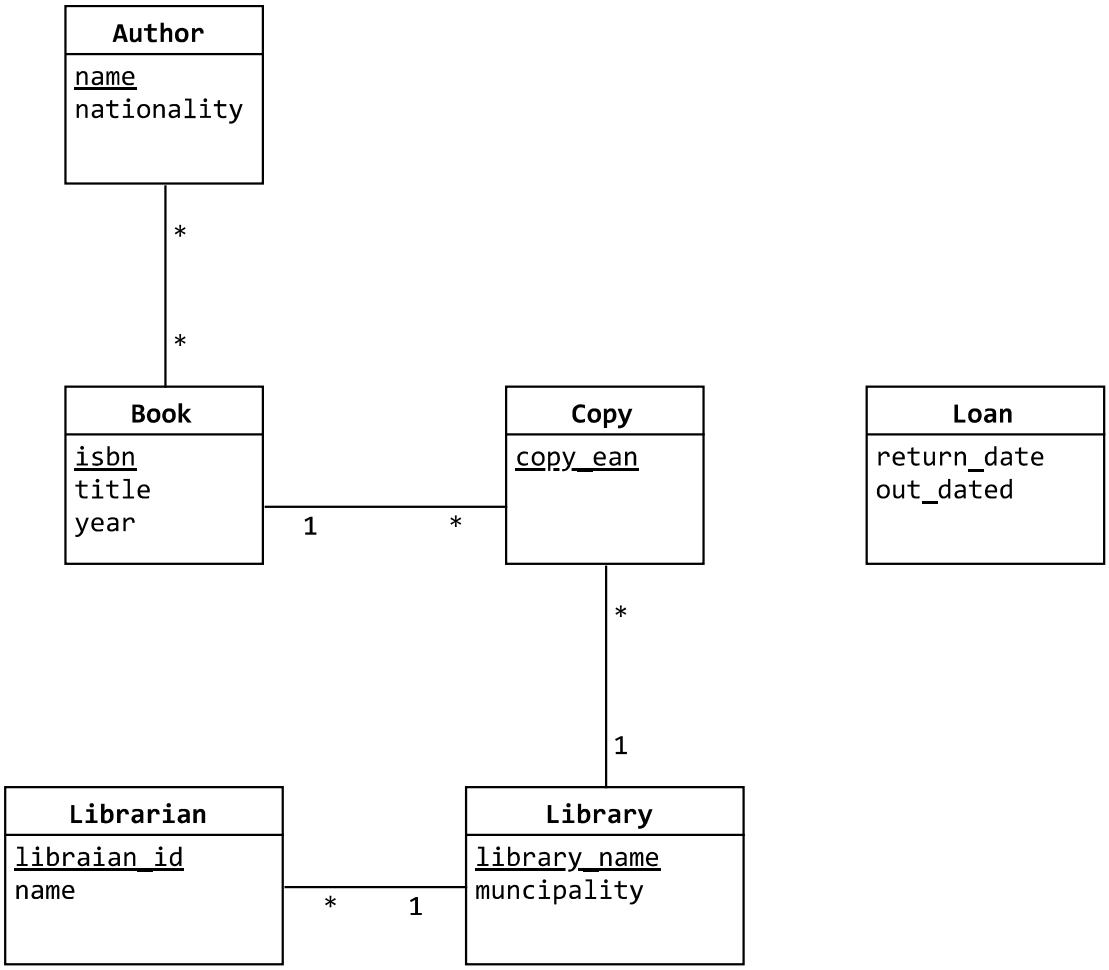
If we had the childrens social security numbers, it would have been a key by itself. But wouldn't have taken the 'key-ness' away from our two previous keys – they are both 'longer' than `ssn`, but they're still minimal in the sense that nothing can be taken away from them if they're supposed to uniquely identify the children. So they're both still candidate keys.

A bigger example

Towards the end of the Tuesday lecture, we started working on this problem (from the preparation notes, in was given in the exam in April 2017):

> I den här uppgiften ska vi utveckla en databas för bibliotek för utlåning av böcker. Databasen är nationell och ska användas vid Sveriges alla bibliotek. Ett bibliotek identifieras av ett unikt namn och hör till en kommun. Bibliotekarier arbetar vid bibliotek och identifieras av ett unikt anställningsnummer, men vi behöver också spara deras namn. En bibliotekarie kan bara arbeta vid ett bibliotek i taget, men anställningsnumret är unikt för hela landet.
>
> Databasen behöver hålla reda på böcker, där titeln, publiceringsår, förlag och ett unikt ISBN (International Standard Book Number) krävs. Varje bok är skriven av en eller flera författare, och för varje författare behöver vi hålla reda på vilket land författaren kommer från. I den här uppgiften kan vi anta att författarnas namn är unika. Av varje bok finns det ett antal exemplar, och varje exemplar tillhör ett givet bibliotek. Ett bibliotek kan ha flera bokexemplar av samma bok. Varje bokexemplar har en unik kod, information om den är utlånad eller inte och numret på hyllan boken står i.
>
> Sedan har vi låntagare som lånar böcker. De identifieras av deras personnummer, men vi behöver också spara deras namn och epostadress, så att vi kan skicka email vid förseningar. Varje låntagare har ett lånekort som har ett kortnummer och en pinkod. Även barn kan vara låntagare, men eftersom barn inte är myndiga behöver vi spara en vårdnadshavare för barnet.
>
> Låntagare kan låna böcker, eller mer precist, bokexemplar. När en bok lånas ut registreras lånedatumet, när boken senast ska lämnas tillbaka och vilken bibliotekarie som hanterade utlåningen. När en bok sedan lämnas tillbaka registreras återlämningsdatumet och vilken bibliotekarie som hanterade återlämningen. Systemet ska stödja möjligheten för låntagare att dela upp sin återlämning på flera dagar. Till exempel kan en låntagaren låna fem böcker på fredagen, och lämna tillbaka två böcker på måndagen och de resterande tre böckerna på tisdagen.

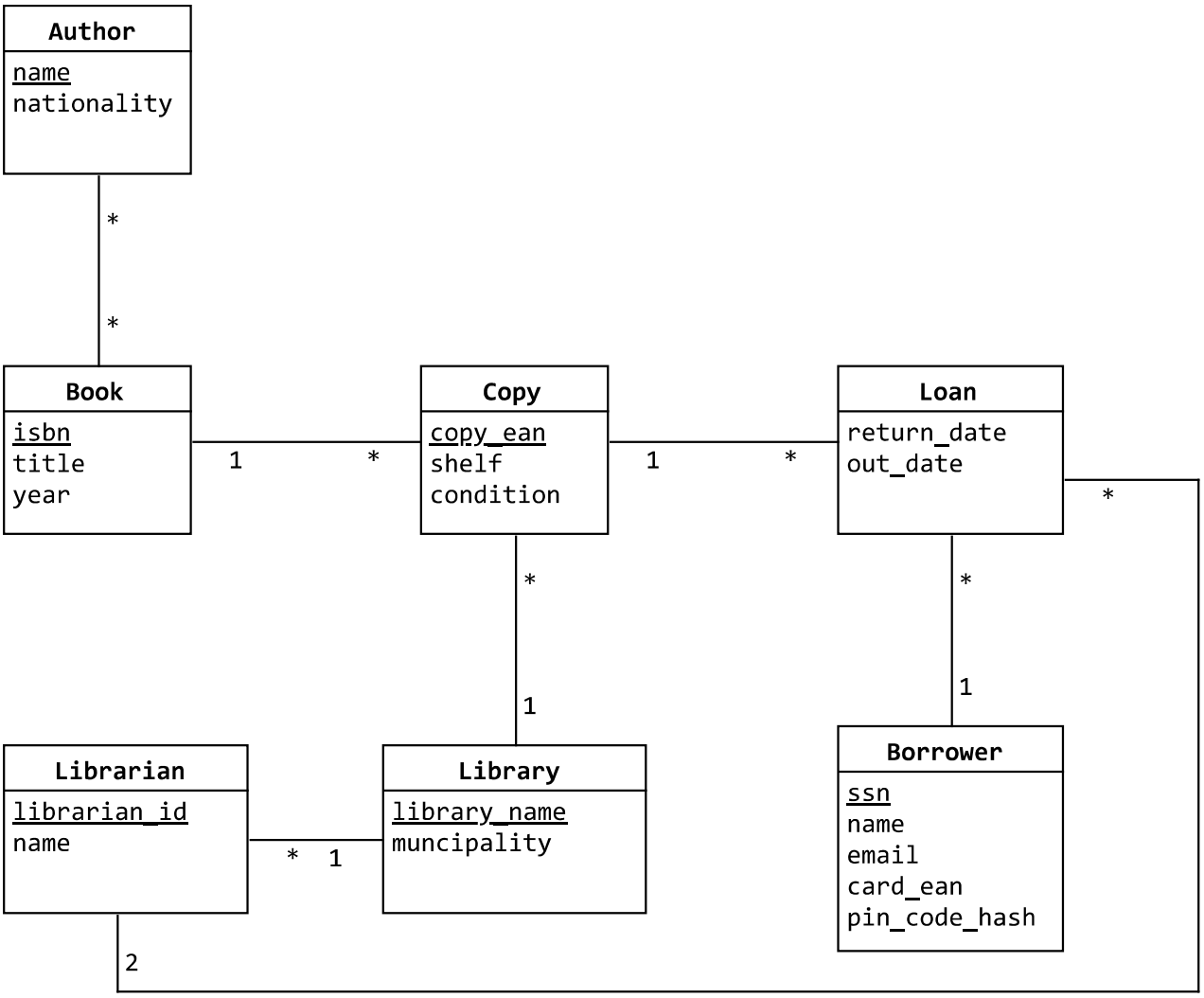
We didn't get any further than to:



Try to finish this on your own before the Thursday lecture – we'll do it together first thing on Thursday, before we talk about how to create databases out of our ER-models. To prepare for the Thursday letter, you could also try to decide what tables we're going to need for the database (after Thursday, you'll know!).

Thursday

We worked some more on the ER model – unfortunately the projector was very flimsy, so I had to draw on the whiteboard. We didn't look too closely at the problem text (since the projector wasn't working...), so we ended up with a model which doesn't describe the problem text exactly (we didn't handle children, etc.), but was good enough for the purpose of the exercise. Here is what we came up with (with some minor adjustments):



Translating our model into a database

In the slides I had some 'recipes' for translating an ER model into tables in a relational database, we started out by creating tables for some of our entity sets:

```
CREATE TABLE authors (
  name         TEXT,
  nationality  TEXT,
  PRIMARY KEY  (name)
);

CREATE TABLE books (
  isbn         TEXT,
  title        TEXT,
  year         INT,
```

```
  PRIMARY KEY  (isbn)
);

CREATE TABLE copies (
  copy_ean     TEXT,
  shelf        TEXT,
  condition    INT,   -- 10 = perfect shape, 0 = completely wrecked
  PRIMARY KEY  (copy_ean)
);
```

We then stopped and looked at how to translate different kinds of associations, and we started by trying to figure out how to implement the `*-1` from `Copy` to `Book`. You came up with the idea of letting the `copies` table have an attribute which keeps track of the proper row in the `books` table – in the slides this kind of attribute is called a *foreign key*:

```
CREATE TABLE copies (
  copy_ean     TEXT,
  shelf        TEXT,
  condition    INT,
  isbn         TEXT,
  PRIMARY KEY  (copy_ean),
  FOREIGN KEY  (isbn) REFERENCES books(isbn)
);
```

We then pondered how to implement a `*-*` association, and I hope it was clear to you why we couldn't just let each table have a foreign key to the other (we can only do that if we're referencing just one element in the other table, here we're potentially referencing many).

So we came up with *join tables* (see the slides), and we connected the `authors` table and `books` table with:

```
CREATE TABLE authorships (
  author_name  TEXT,
  book_isbn    TEXT,
  PRIMARY KEY  (author_name, book_isbn),
  FOREIGN KEY  (author_name) REFERENCES authors(name),
  FOREIGN KEY  (book_isbn) REFERENCES books(isbn)
);
```

Those three 'rules' will take us quite far when we translate an ER model into a relational database:

- an entity set will become a table
- a `*-1` ('many-to-one') association will become a foreign key on the `*` side of the association
- a `*-*` ('many-to-many) association will become a new (join) table

For those `*-*` associations with an association class, the attributes of the association class will become attributes in the join table.

Exercise: I'd recommend that you try to implement the whole ER model above, and then write a query to find the names and nationalities of the three most borrowed authors.

We then looked at some kinds of association which requires careful thought (see the slides), and three methods to implement inheritance (see the slides).

We concluded by looking at how to insert, update and delete data in our database – see the links above, and the slides.

If you have any questions about what we did, please go to Moodle and sign up for the QA session next week.