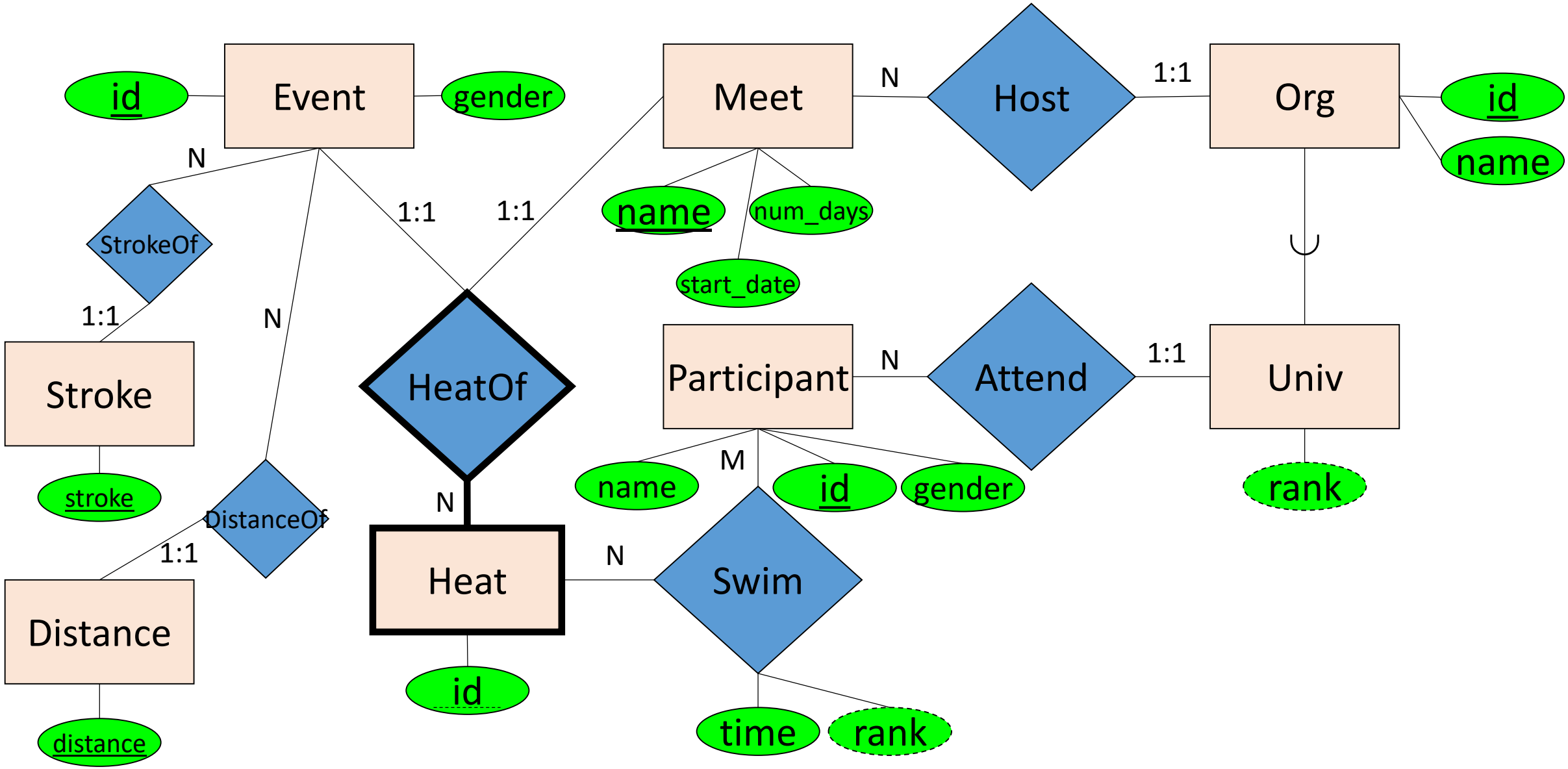
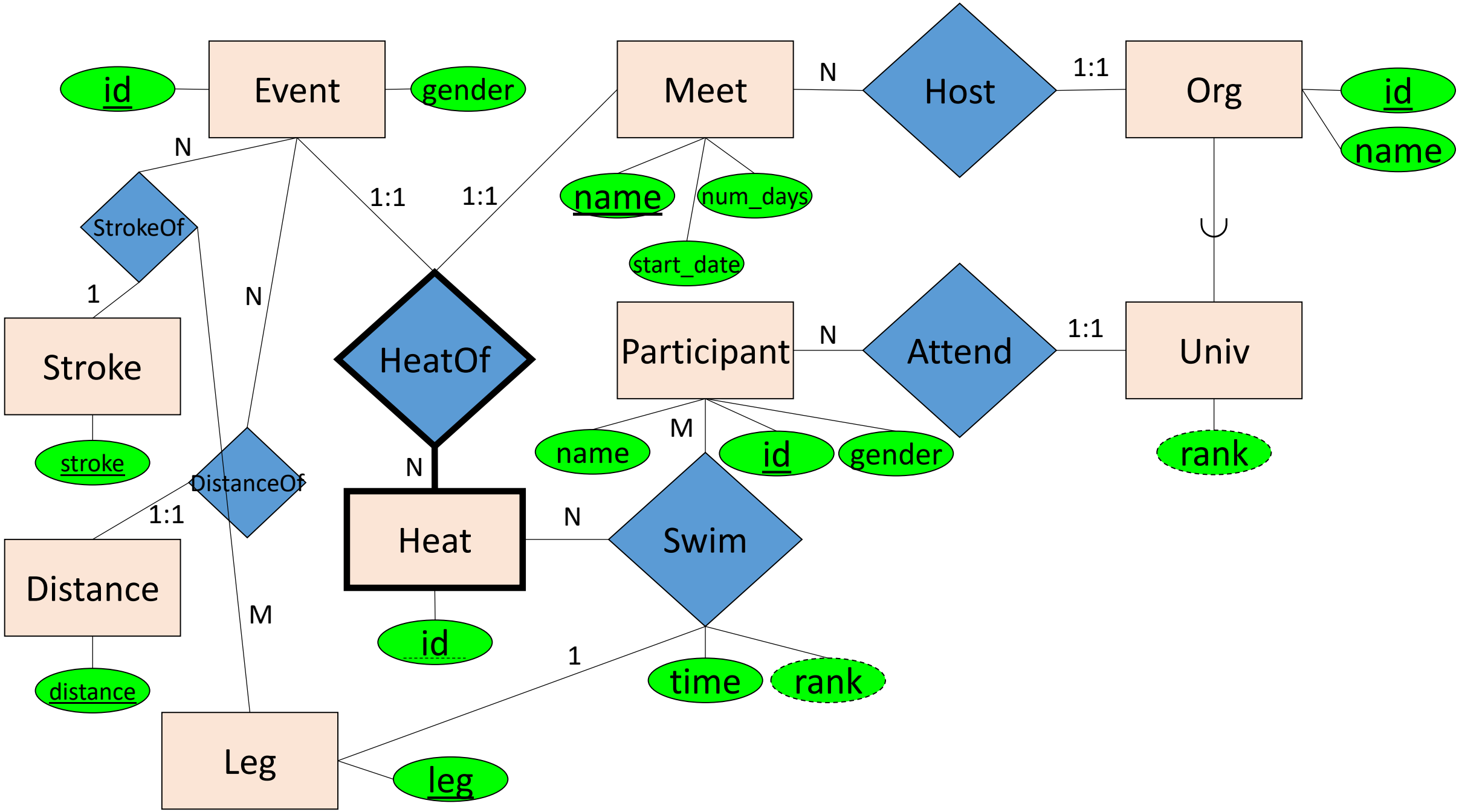

id
Event
gender
Meet
N
Host
1:1
Org
id
name
N
StrokeOf
1:1
1:1
1:1
name
num_days
start_date
1:1
N
HeatOf
Participant
N
Attend
1:1
Univ
Stroke
stroke
N
DistanceOf
1:1
Distance
distance
N
Heat
id
N
Swim
M
name
id
gender
rank
time
rank

Comments on Swimming ERD

- Attributes – Appropriate attributes for Event, Heat, Participant, Org, and Univ were not specified. At a minimum, each needs a key attribute, so a simple ID attribute is used here. Providing additional attributes that are obvious from context (e.g., swimmer name) is acceptable.
- Event – As shown here, Event represents a particular stroke & distance combination independent of any Meet. Alternatively, Event could be related to Meet, and Heat then be related to and dependent on only Event.
- Heat – Heat is considered a weak entity because heats are normally only referred to as, e.g., "Heat 7 of the 100m Butterfly at the Rice Invitational", i.e., dependent on the Event and Meet. (Acceptable to not be weak, if you argue such.)
- Org – We could have a second disjoint subclass of non-university organizations. However, it needlessly adds another entity set.
- Meet attributes – A meet could have an end date instead of num_days. However, then you'd want a constraint that the end date was after the start date. It's slightly simpler to store the duration, instead, and constraint it to be positive.
- Stroke, Distance – These are just lookup tables. An alternative would be to have one lookup table of appropriate stroke-distance combinations, which would then be related to the Stroke and Distance lookup tables.
- Team/Univ – Since there is only one team per university, there's no need to distinguish a separate Team entity set.
- Host – While it would be realistic to allow multiple Orgs to host a single Meet, this was not allowed by the specification.
- A swimmer "specialization" is a nebulous concept that isn't ever defined or used, so trying to explicitly represent it is a bad idea. With the given information, it would be at most a calculated attributed, with its value calculated based upon what a swimmer has swum in.
- Attend – The possibility of a swimmer transferring schools isn't mentioned, and so isn't allowed here.
- Gender – Since each Event has a gender, then we also need to know the gender of each Swimmer. With this representation, the consistency of gender in an event could be checked by a query. Adding a gender attribute to Heat provides no benefit, but provides a chance for a Heat and Event to be inconsistent in gender. Subclassing Event, Heat, and Participant into separate genders (similar to the addition of relays) is also acceptable, but whether there is significant conceptual benefit for this complexity is debatable.


id
Event
gender
Meet
N
Host
1:1
Org
id
name
N
StrokeOf
1:1
1:1
name
num_days
start_date
1
N
Stroke
HeatOf
Participant
N
Attend
1:1
Univ
stroke
N
M
name
id
gender
rank
DistanceOf
1:1
N
Heat
N
Swim
Distance
M
id
1
time
rank
distance
Leg
leg

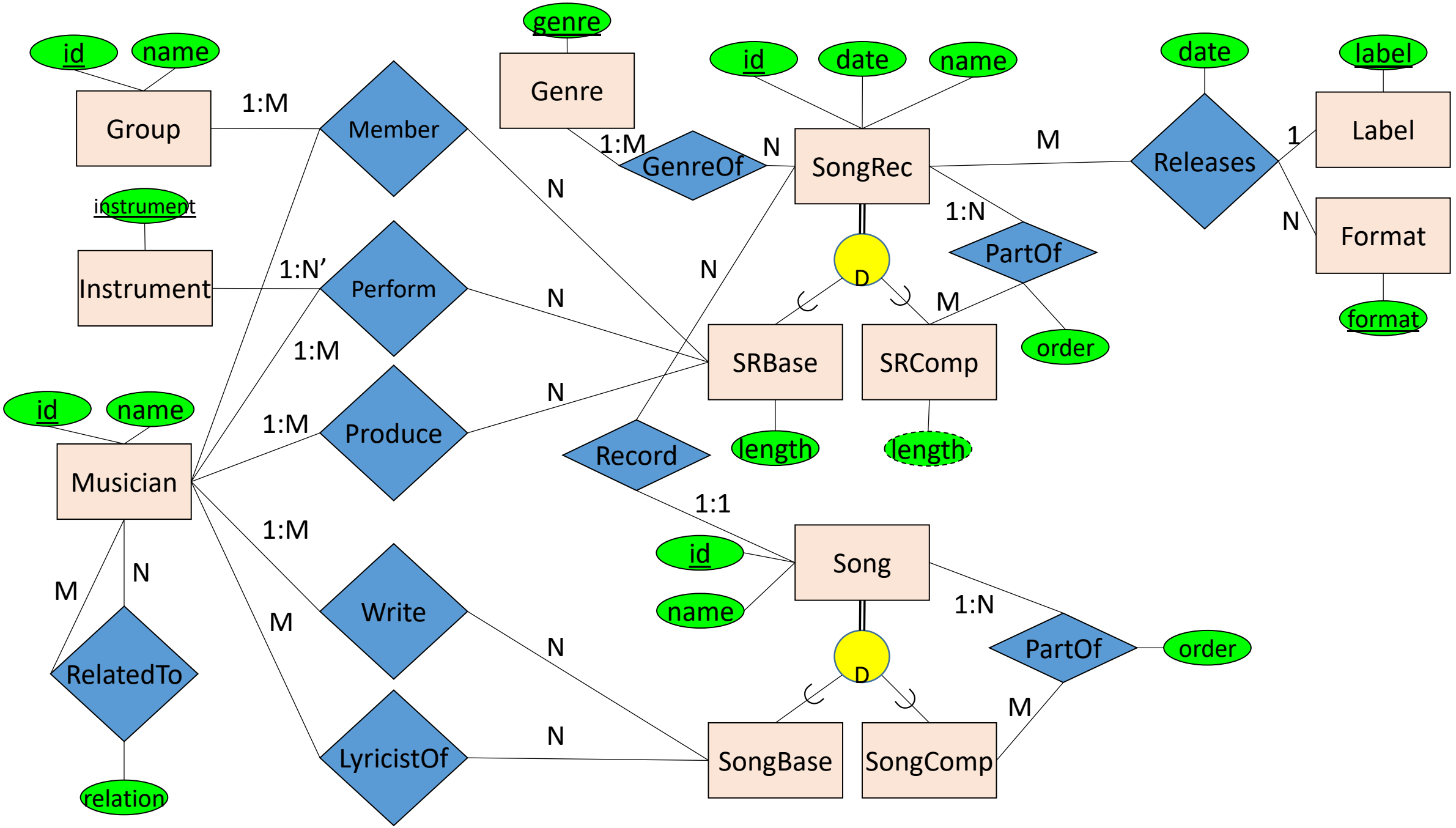
Comments on Swimming ERD – COMP 533

I thought of three basic approaches:

- Barely modifying the original diagram – Keep the diagram simple, as we did for gender. Adapting to relay races, however, will generally abuse some of the diagram semantics.
- Add individual- and relay-based subclasses to Participant, and possibly Heat and Event. Most accurate, but most complex. (Remember that the complexity can disappear during implementation if we only implement the superclasses.)
- Change the semantics of Event, Heat, and Participant so that every race is a relay, and that an individual race is just a relay with one leg. This allows future flexibility if relays of, say, 3 legs are used. Also simple. But allows inconsistencies such as 2 people in a 4-person event.

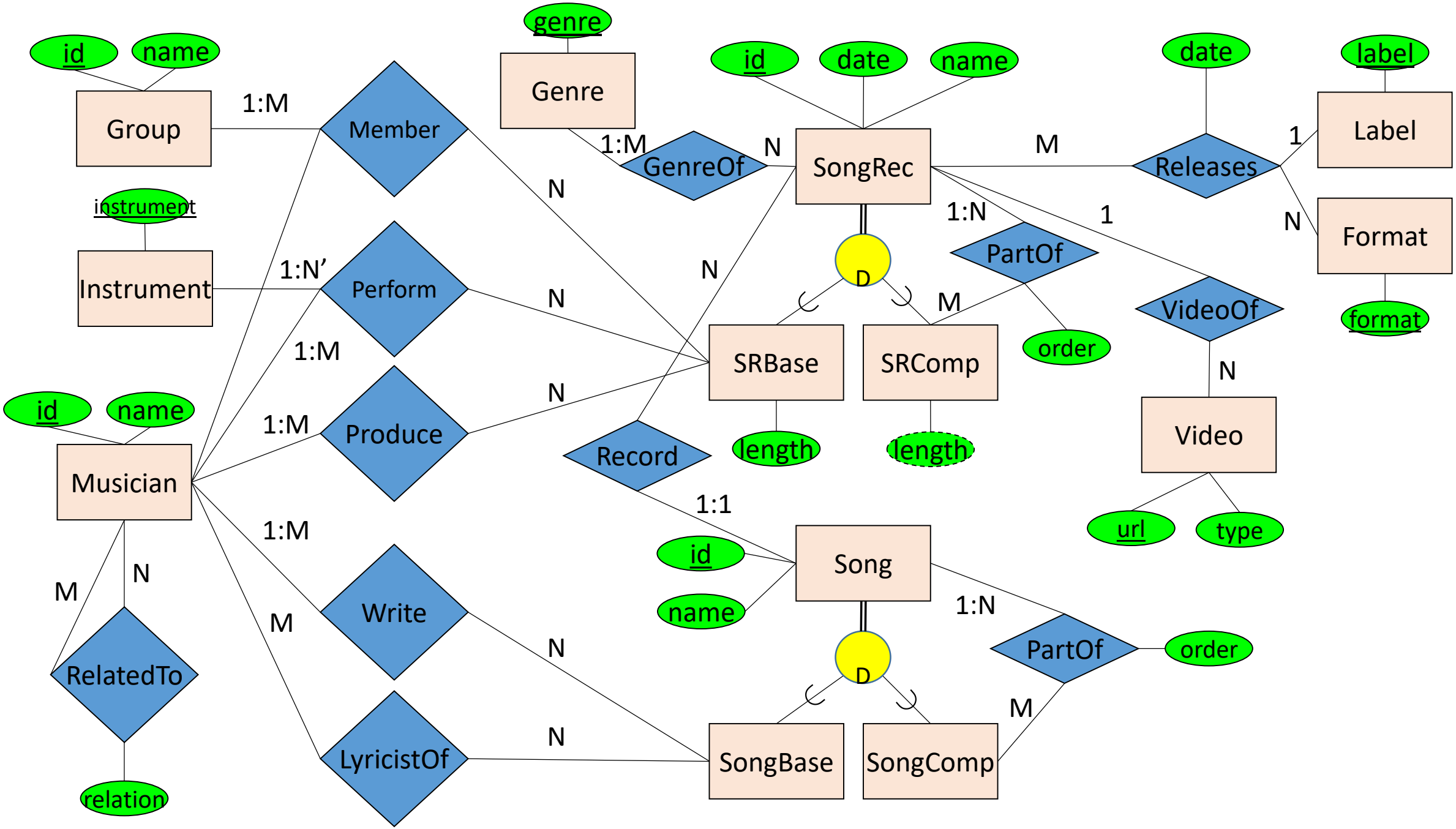
I chose the last version due to a preference for extensibility and simplicity.

- More extensible means that the system design can handle some kinds of changes in system requirements. It is generally a good principle of software engineering.
- Added Leg to Swim relation to show the order of a swimmer in the relay.
- Added Leg to StrokeOf relation to show allow each leg of a relay to have a different stroke.


id
name
Group
1:M
Member
genre
Genre
1:M
GenreOf
N
id
date
name
SongRec
M
date
Releases
1
label
Label
N
Format
format
instrument
Instrument
1:N'
Perform
N
N
1:N
PartOf
D
M
order
SRBase
SRComp
length
length
1:M
id
name
Musician
1:M
Produce
N
Record
1:1
1:M
M
N
RelatedTo
relation
Write
M
N
id
name
Song
1:N
PartOf
order
D
M
LyricistOf
N
SongBase
SongComp

Comments on Music ERD

- Musician and Group
  - IDs and names weren't explicitly mentioned in the specification, but seem obvious additions. At a minimum, these entity sets each need a primary key.
  - For modeling simplicity, only Groups can record, but a Musician can be part of a one-person Group. A more obvious alternative is to have Musician and Group each be a subclass of RecordingArtist.
- Perform
  - In this design, the Member relation may appear to be somewhat redundant with Perform. However, the Perform relation is really all about who is playing what instruments, whereas Member is really more about creative ownership. Note that a member of the relevant group need not actually perform on a song (e.g., "Yesterday" by the Beatles is performed only by Paul McCartney plus a string quartet).
  - It isn't clear whether a "role" attribute is needed (e.g., "session" vs. "guest"). It is omitted here since even Group members are in this relation, and their "role" information would be redundant with being in the relevant Group.
- Instrument – This is a table instead of an attribute so that each Musician can play multiple instruments in a SongRec.
- Song, SongRec
  - It is questionable as to whether Write and LyricistOf should be related to SongBase or to Song. As an alternative, connect these relations to Song, and the subclassing of Song is no longer useful. However, the alternative version allows redundancy and inconsistency, e.g., two songs are written by person A, but their combination is written by person B.
  - The same applies to whether Member, Perform, and Produce should be related to SRBase or to SongRec.
- SongComp – It isn't clear whether a "type" attribute is needed (e.g., specifying whether it is an "album", "film score", or whatever).
- Record – While it may seem odd to only relate the superclasses, this is for flexibility, so that a SongRec isn't necessarily decomposed the same as the Song. Arguably, the subclasses should be related by a pair of such relations, instead.
- GenreOf – Whether Genre should be related to SongRec or SRBase is unclear. Relating to SRBase is more meaningful. Relating to SongRec might make more sense in the user interface also also seems more implied by the specification.
- Releases – The specification didn't allow for multiple releases or multiple labels for the same SongRec. One release, one date, multiple formats.
- RelatedTo – Remember, the specification says a full family tree is not desired.
- Minimum cardinalities – The minimum cardinality of 1 on many relations (Member, Perform, Produce, Write, and Genre) all assume perfect information. E.g., every song must have a writer, and we know who it is. Arguably, however, it is more realistic in these cases to allow incomplete knowledge and a minimum cardinality of 0.


id
name
Group
1:M
Member
genre
Genre
1:M
GenreOf
N
id
date
name
SongRec
M
Releases
date
1
label
Label
N
Format
format
instrument
Instrument
1:N'
Perform
N
N
1:N
PartOf
1
D
M
order
VideoOf
N
Video
url
type
SRBase
SRComp
length
length
1:M
Produce
N
id
name
Musician
Record
1:1
Song
id
name
1:N
PartOf
order
D
M
M
N
RelatedTo
relation
1:M
Write
N
M
1:M
LyricistOf
N
SongBase
SongComp

Comments on Music ERD – COMP 533 only

Repeating the SongRec hierarchy and its relations would be very redundant. So, to me the most natural is to say that we can have videos of each recording.

Grading Guidelines

There will be great variation – more than anticipated by my comments. The following are ***rough*** guidelines.

- Main entity sets & relations: 4 points each
- Lookup tables & relations: 1 point each
- Attributes: 0.5 point each
- Cardinalities: 0.5 point each
- Poor design decisions:
  - Well motivated: 2/3 credit
  - Poorly motivated: 1/3 credit
  - More if the design decision is better than “poor”