Advanced ER Modeling

1 Weak Entities

We expect all entities to have an identifier, that is a set of attributes that is guaranteed to distinguish that entity from all other entities of the same type. If not, we couldn't distinguish among different entities ! However, in a few cases we have entities that cannot be fully identified in the database, but they can be identified when associated with other entities.

We call the entities that need of other entities for their identifier **weak entities** (in contrast, we can call normal entities **strong entities**) and we say they are **identification dependent** on some other entity.

For me, one of the best examples would be an employee's dependents (kids). Assume my company's HR decides it would be great to send my kids a postcard for their birthday, so they will keep info about kids on their DB. They could ask for their SSN, but many employees would not want to provide that information, so they may opt for letting the dependent be a weak entity. This means the dependents would not necessarily be distinguishable from each other solely by their information on the DB, but all dependents from the same employee need to be distinguishable.

So a first cut of the ER diagram would look like this:

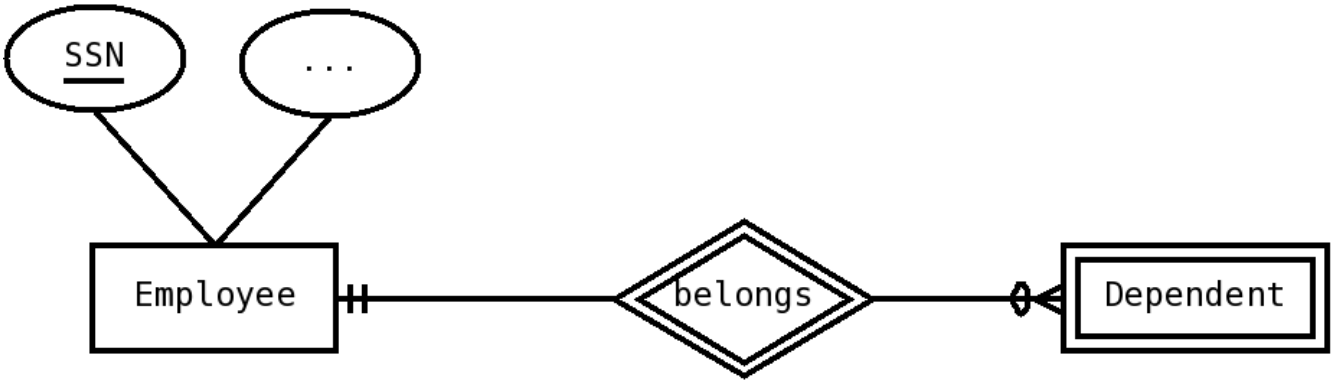

Figure 1: Employee and dependents, INCOMPLETE

Notice how dependent is marked as a week entity by using double lines on the rectangle, and that belongs is marked as the identifying relationship by the same double lines. Also, each dependent, by necessity, belongs to exactly one Employee.

Now we can worry about the dependents' attributes. Let's assume we want to keep their name and their date of birth (named DoB), and that we judge that dependents from the same employee will never have the same name and date of birth (although this can't actually be guaranteed in the real world, it seems like a safe assumption).

So we can say that the dependents name and date of birth function as **discriminators** or **weak identifiers**; that is, although they can't distinguish among all dependents, they can distinguish among all dependents for the same employee. For convenience, I use a slightly different notation from the book. I use dash-underlining to denote weak identifiers, whereas the book uses double underlining (if you're using dia for your diagrams, like me, it produces dash-underlining).

The ER diagram would look like this:

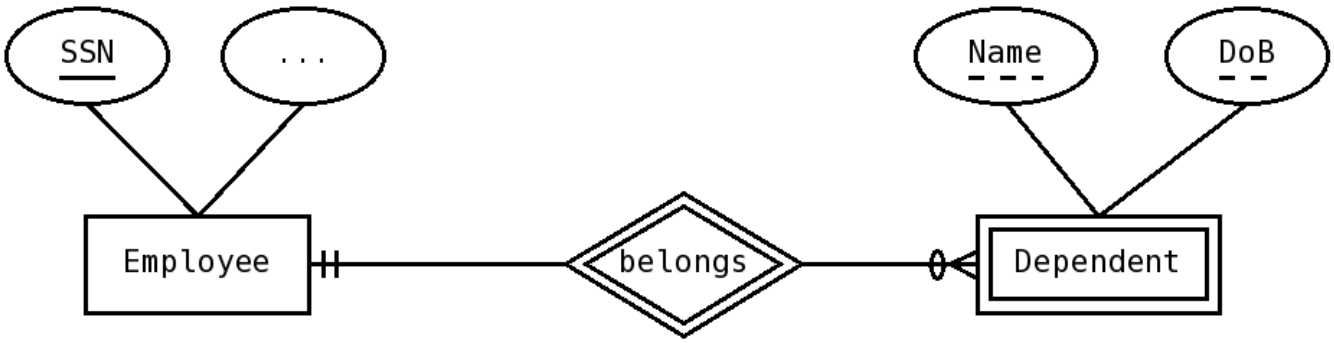

Figure 2: Dependent as a weak entity. Notice weak identifiers.

Notice that the fact that makes dependent a weak entity is that it doesn't have an identifier; so an easy solution is to give it one. If we agree to assign a number to each dependent as it enters the database, then Dependent becomes a normal (strong) entity type, and the diagram would look as in Figure 3.

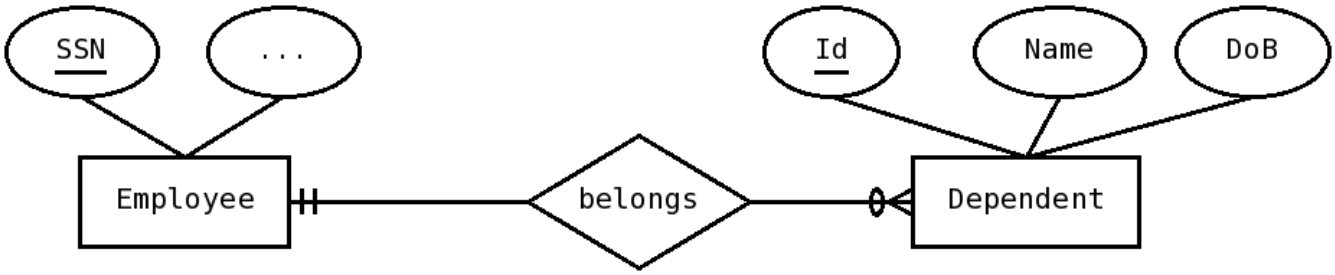

Figure 3: Dependents, now as a strong entity

In this particular case, it also allows us to not enforce the artificial constraint that name and dob be discriminators (if an employee wants to use the same name for each one of their triplets, let them do so :). The cost is introducing an artificial attribute, Id.

2 Identification vs Existence dependency

Notice that the book definition tends to highlight another aspect of weak entities. They cannot exist without the strong entity. Oftentimes, we can't even think about the weak entity unless it is associated with the strong entity.

When an instances of an entity type A cannot exist without those of another type B, we call A **existence dependent** on B. Identification dependence implies existence dependence, but not vice versa.

In Figure 2, dependents can't exist without employees, and Dependent also needs Employee for identification. In figure 3, dependents still can't exist without an employee, since belongs is a mandatory relationship, but Dependent is a strong entity now. The deciding factor for weak entities is that they can't be distinguished unless associated with the other entity, not that they need it for existing.

3 Section Example

Another common example for weak entities would the sections of a course. Many students don't realize it, but the school actually keeps information on which *sections* they enroll to, and

the information on their transcripts is derived from this. Of course, we need information about which course does each section belongs to, and each section belongs to exactly one course.

Here we have a clear case of existence dependence; we cannot even think of a section without a course; so we could model section as a weak entity; we can use a section number to distinguish among different sections of the same course. The diagram would look like this:

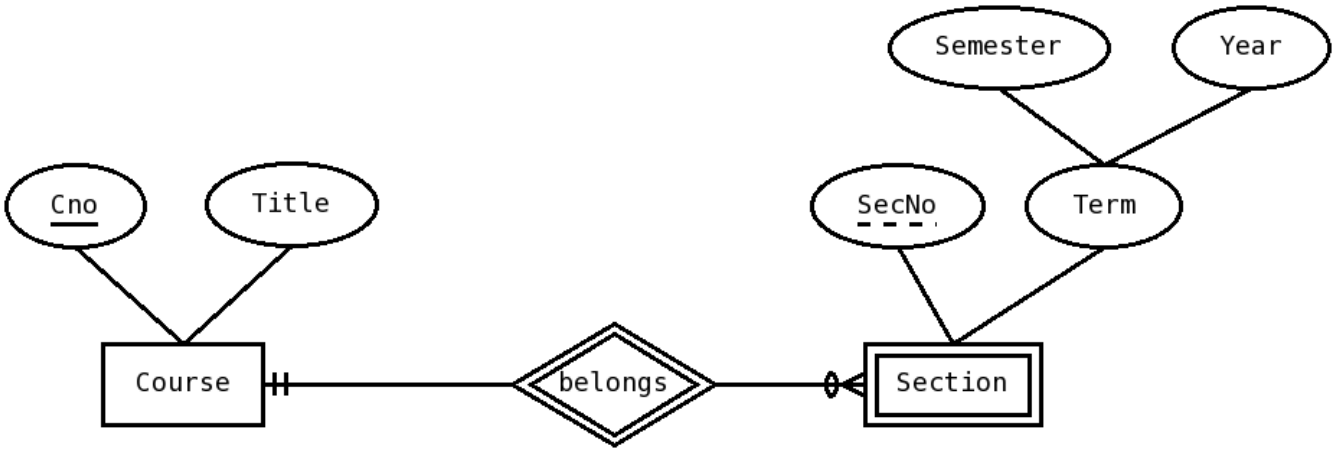

Figure 4: Section as weak entity, with SecNo as discriminator

Since Section is a weak entity, we cannot use SecNo to distinguish among all sections; there may be two sections with number 1, but they have to be from different courses. So in reality we are using the course and the number to identity (so we have things like CS3153-001 or CS3153-900, which are the sections for the undergraduates in this course)

Now this diagram would say that we use the SecNo (section number) attribute to distinguish among entities from the same course. At SPSU we allow for sections of the same course to have the same section number, as long as they are not from the same semester (so there may be a CS3153-001 for Fall 07, and another one for Summer 07). So, the weak identifier is actually the Section number plus the term, and the diagram would look like this:

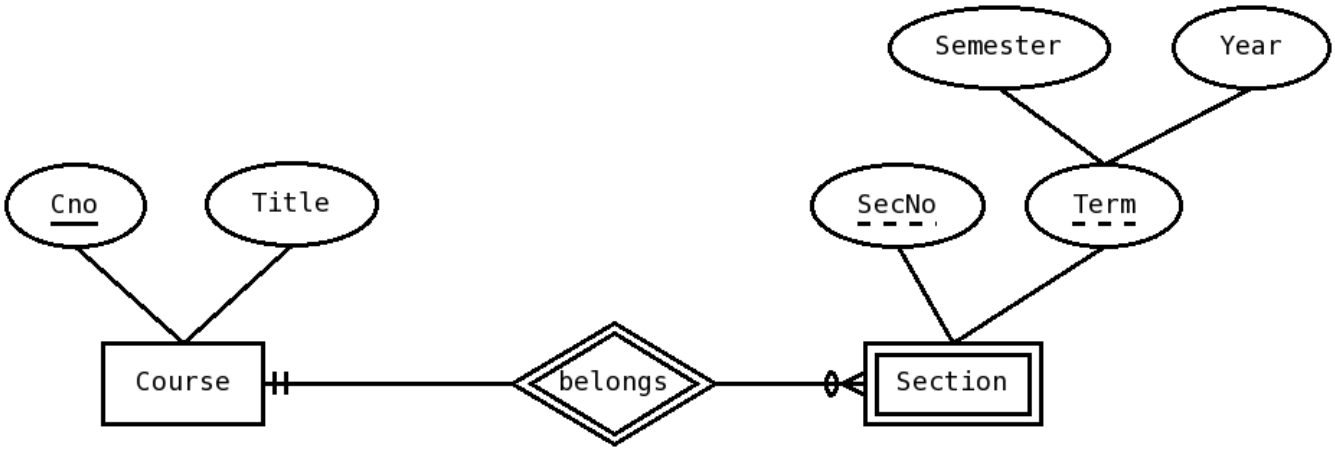

Figure 5: Section as weak entity, with SecNo plus term as discriminator

Notice that sine both SecNo and Term are underlined, they *together* act as the weak key.

Of course, we could add an identifier to each section to make section a strong entity type. At SPSU, we call that identifier the CRN. The diagram would look like Figure 6. Notice that we are now not expressing the constraint that the SecNo and Term could be used as weak identifiers, and so we would need to express that constraint as a comment somewhere else.

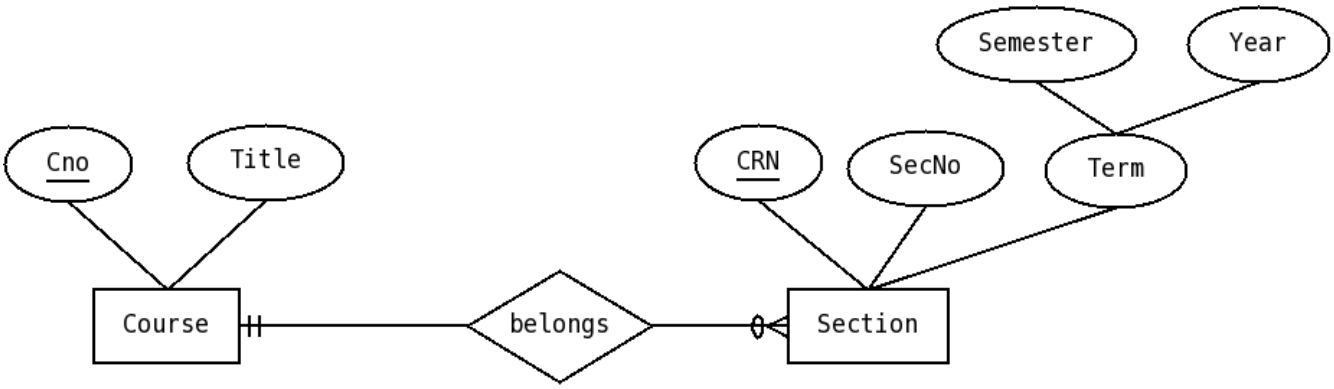

Figure 6: Section is now a strong entity, since CRN is an identifier

Now for truth in advertising, the situation at SPSU is slightly more complicated, since the CRN for a section could be repeated, but not on the same term (mental exercise, how would you change the diagram ?).

2 Associative Entities

Associative entities are one of the hardest concepts in ER modeling, but they are sometimes necessary. An associative entity is a relationship that is transformed into an entity. Each *instance* of an associative entity represents an *instance* of the relationship.

Associative entities could be used for many-to-many relationships with attributes, but I think that just complicates the model for no gain; it is needed to represent ternary relationships, and for cases when we need to convert a relationship into an entity, to relate it to other entities.

An example will (hopefully :) illustrate this. Assume that we want to have a database of skills for employees, where we record the fact that an employee has certain skills. For each employee we keep its SSN (identifier) and name, and for each skill we keep its id and a name. We will also keep the date we realized the employee has the skill. The ER diagram would look like this:

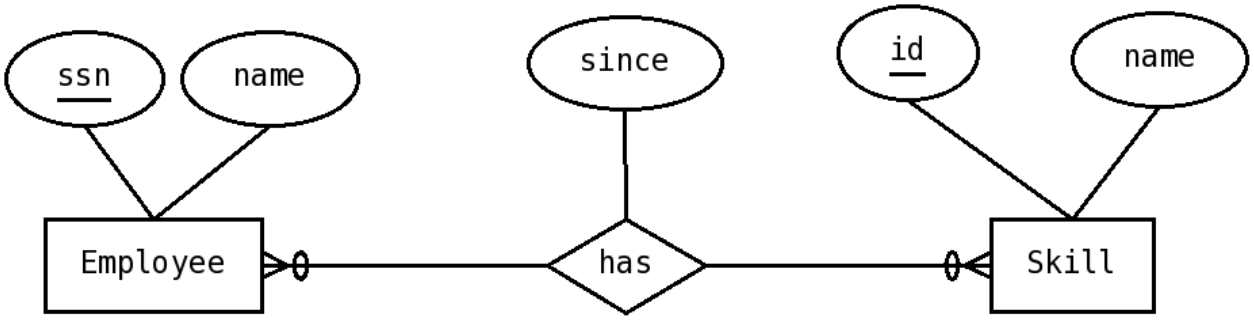

Figure 7: Employees and Skills, with a binary relationship

Now, some people feel that many-to-many relationships with attributes are weird, and so they'd create an associative entity. An associative entity is an entity whose instances represent associations. It is denoted by a rectangle with a diamond inside, and the special relationships with the other entities it associates are marked by NOT having a diamond, just

the lines; the diagram from figure one above would change as follows:

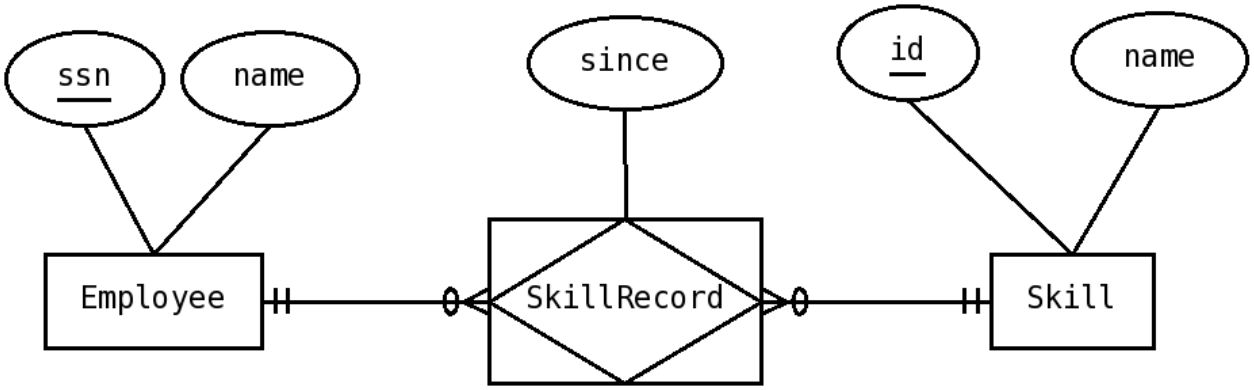

Figure 8: Associative entity relating Employees and Skills

We made the following changes:

- Instead of the relationship *has* we now have the associative entity *SkillRecord* (notice we need to change the name, since it is an entity and should have a noun as its name)
- SkillRecord has two special relationships (with no diamonds !); with Employee and Skill. A SkillRecord is associated with exactly one employee and exactly one Skill (since it represents the association between that employee and the skill).
- An employee can be associated with zero or more SkillRecords (since it could have zero or more skills), and a Skill can be associated with zero or more SkillRecords. Notice that we can go *through* the SkillRecord, from Employees to Skill and vice versa.
- Notice that the relationship had a *set* of instances, meaning we wouldn't record twice the fact that an employee has a particular skill (for the same employee and skill).

This is not a great application of associative entities, but it is a simple example. Their power is evident in more complicated applications. Imagine that we want to also record information about certifying organizations, and which organization certifies a given skill. For example, we could record that Red Hat certifies that we know Linux administration, or that Oracle certifies that we know SQL. It could be that the same skill can be certified by many organizations (for example, we can consider both IBM and Oracle to be able to certify my SQL skills).

In Figure 1, our original diagram, we would add an entity type representing the certifying organizations; but then we have a problem; we want to associate this entity with the *has* relationship, which we *can't* do, since a relationship can only associate *entities*. Using an associative entity solves this problem, since now we can associate the certifier with the SkillRecord.

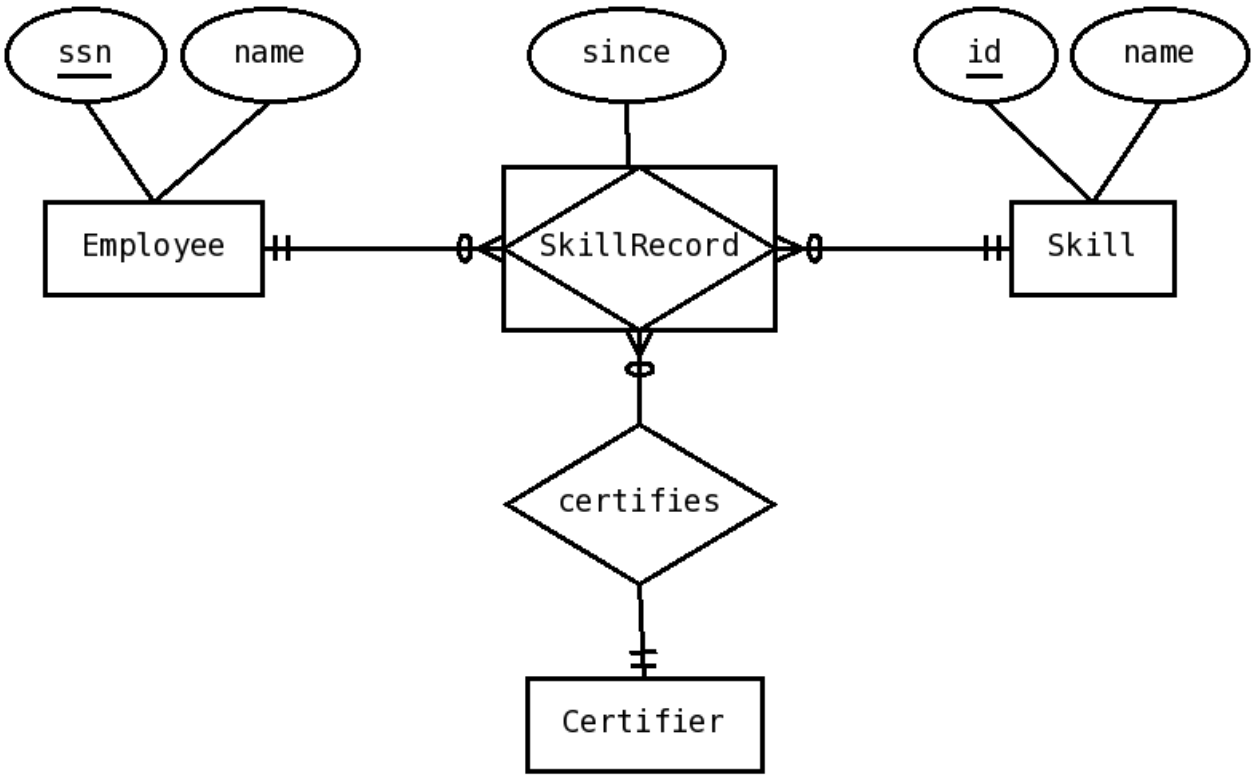

Figure 9: Associative entity with a normal relationship

Notice that SkillRecord still associates just the Employee and the Skill, and is this record which is now associated with the certifier; basically we would still record just once the fact that an employee has a certain skill. This is slightly different from a *ternary relationship*, which allows for free association between three different entities.

3 Ternary Relationships

A ternary relationship is one that associates three entities *at the same time*. It may be hard to see why do we need ternary relationships. Can't we just live with binary relationships ? Let's see.

We can think of a relationship instance as stating a fact relating the entity instances it associates. The relationship type refers to our description of all the possible facts of that type.

Say we wanted to keep track of which Person got which Degrees from Which University. We could represent this with a ternary relationship as follows:

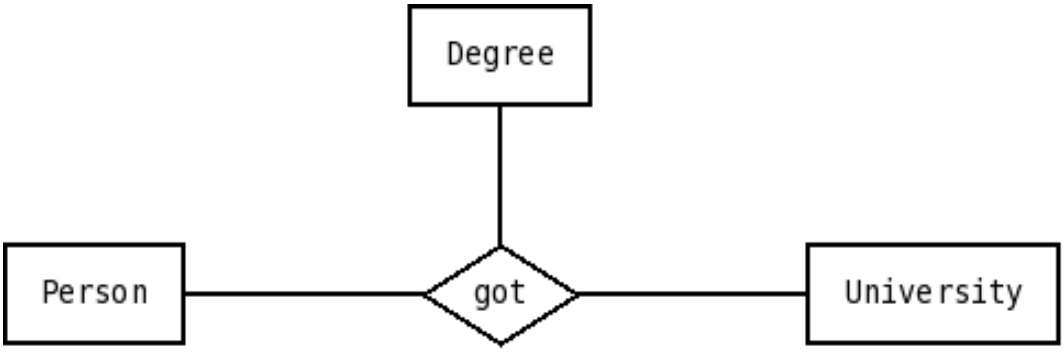

Figure 10: Got as ternary relationship. Notice we're not including cardinality constraints. INCOMPLETE

Notice this is an incomplete diagram, since we are not specifying the cardinalities.

Now, the question is, can we do this with a set of binary relationships ? It turns out we can't, unless we introduce an associative entity. Let's see why.

The diagram says that our facts for the got relationship relate three object. They would be of the form:

- Person P got Degree D from University U

For example, let's see the degrees of one person. For simplicity we will use the names of the things (person, degree and University) to refer to them.

So, let's see Orlando's degrees.

1. Orlando got a Bachelor's degree from University of Yucatan
2. Orlando got a PhD from Tulane University

Given that ternary relationships are complicated, we may be tempted to try to represent this relationship with a set of binary relationships among those same entities. The more general possibility is to have a relationship among each pair of entity types. The ER diagram would look like this:

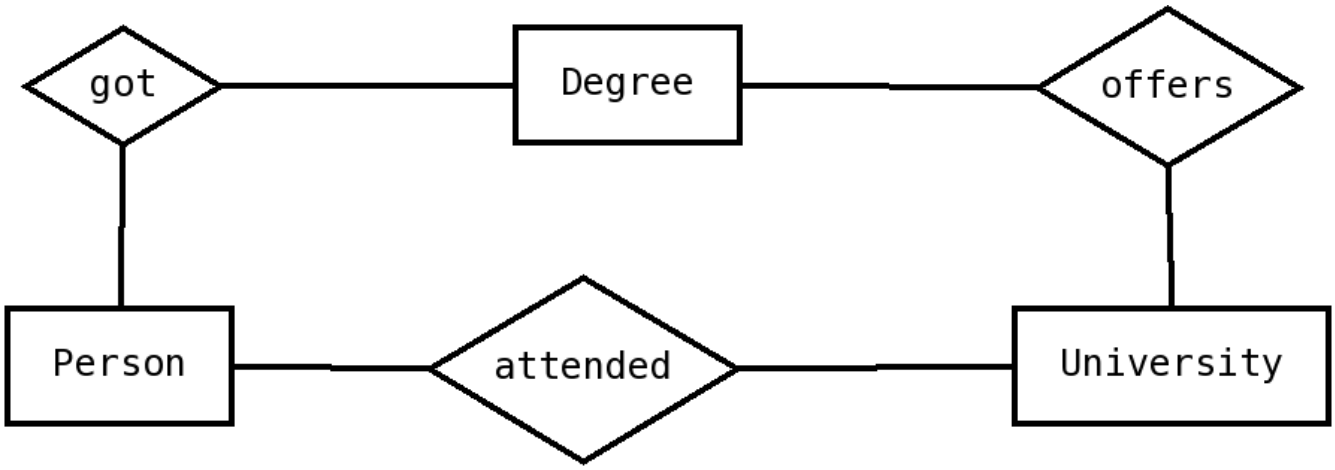

Figure 11: Trying to represent a ternary relationship with three binary relationships among the original entities. WRONG and INCOMPLETE

OK, so let's see what this means in terms of facts. We now have three kinds of facts:

1. Person P got Degree D
2. Person P attended University U
3. University U offers Degree D

Now, let's try to represent Orlando's info with this three kinds of facts. Well, we actually need a little bit more info since we don't know all the degrees a university offers. To see the problem clearly let's assume that both Tulane U and U of Yucatan offer Bachelor's and PhD degrees (which is actually true).

So we would have:

Offers

1. Tulane U offers Bachelor's

2. Tulane U offers PhD
3. U of Yucatan offers Bachelor's
4. U of Yucatan offers PhD

Got

1. Orlando got a Bachelor's
2. Orlando got a PhD

Attended

1. Orlando attended Tulane U
2. Orlando attended U of Yucatan

Now, from this last 8 facts only, can you say where did Orlando got each one of his degrees from ? Think it through.

Hopefully, you convinced yourself you can't. This binary relationships cannot represent the same information (and of course, having less kinds of facts would be even worse !).

So, we need ternary relationships. OK, so what's the problem with them. The problem is notational. In our notation, we put cardinality constraints for an entity on the opposite side; but we now have two opposite sides. Well, we could change our notation and make this a special case; but we would still have more complicated semantics.

There is an arguably simpler way to deal with ternary relationships. We can introduce an extra entity type, and use it to represent the original facts.

Putting it in English, we need to represent:

1. Orlando got a Bachelor's degree from University of Yucatan
2. Orlando got a PhD from Tulane University

but we can't use a sentence relating three things (if you actually know grammar, we can't use any indirect or circumstantial objects). So what we can do is to add a new kind of entity, say Degree Assignment, and 3 new binary relationships; has_person, has_degree and has_university. So, to represent the first statement we could say three new statements:

a) The Degree Assignment 1 has_person Orlando (meaning has Orlando as its Person)
b) The Degree Assignment 1 has_degree Bachelor's (has Bachelor's as its Degree)
c) The Degree Assignment 1 has_university U of Yucatan (has U of Yucatan as its University)

This way we CAN represent the same information using only binary relationships, at the expense of adding a new entity type.

Now this new entity type is of a special kind. We call it an **associative entity** since it is an entity that represents an association. Associative entities are represented with a diamond inside a rectangle. Since it is an entity, it has relationships with the other entities; since an associative entity may have relationships with other entities besides the ones it was originally associating, we distinguish those by NOT using a diamond for them.

So, here's how it would look, WITHOUT cardinality constraints.

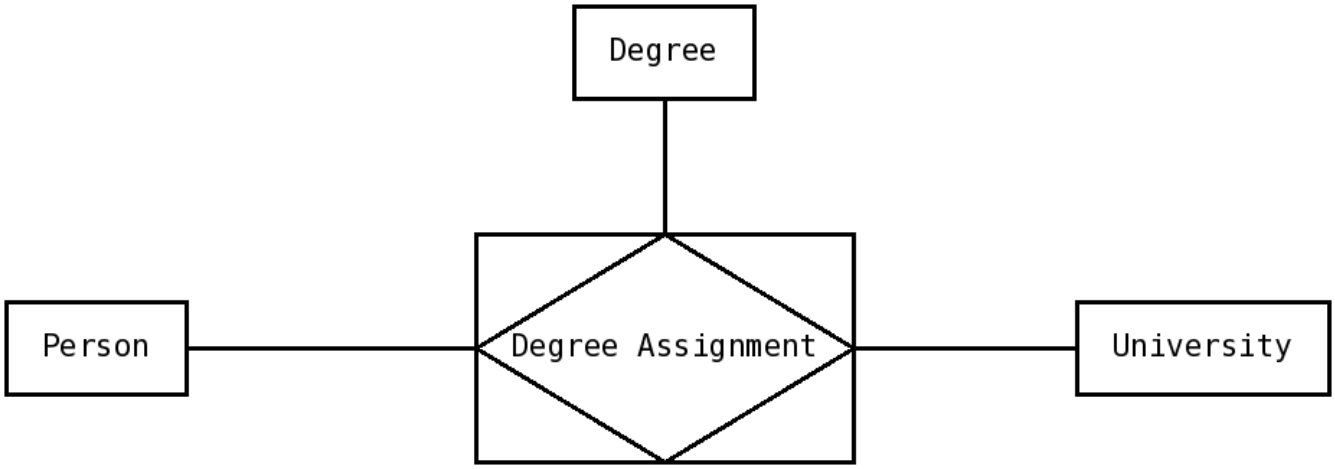

Figure 12: Ternary relationship Got is now an Associative Entity, degree assignment. INCOMPLETE

Now, how can we express those constraints ? simple. The lines joining Degree Assignment with the other entities now represent relationships (although we do NOT use the diamond), and so we can assign the constraints.

Notice that, since each entity of Degree Assignment actually represents an instance of the original ternary relationship 'got', it is, necessarily, associated with exactly one Degree, University and Person; so the constraints next to each of those three entities is exactly one. Moreover, if each University, for example, could just be associated to one Degree Assignment (that is, one degree-person pair), then the binary relationships would work, and would, arguably, be a simpler choice, so each University should be associated with many degree assignments. A similar reasoning applies to Person and Degree, and so each one of the normal entities should most probably have a maximum cardinality of one, leading to a (still incomplete) diagram as follows:

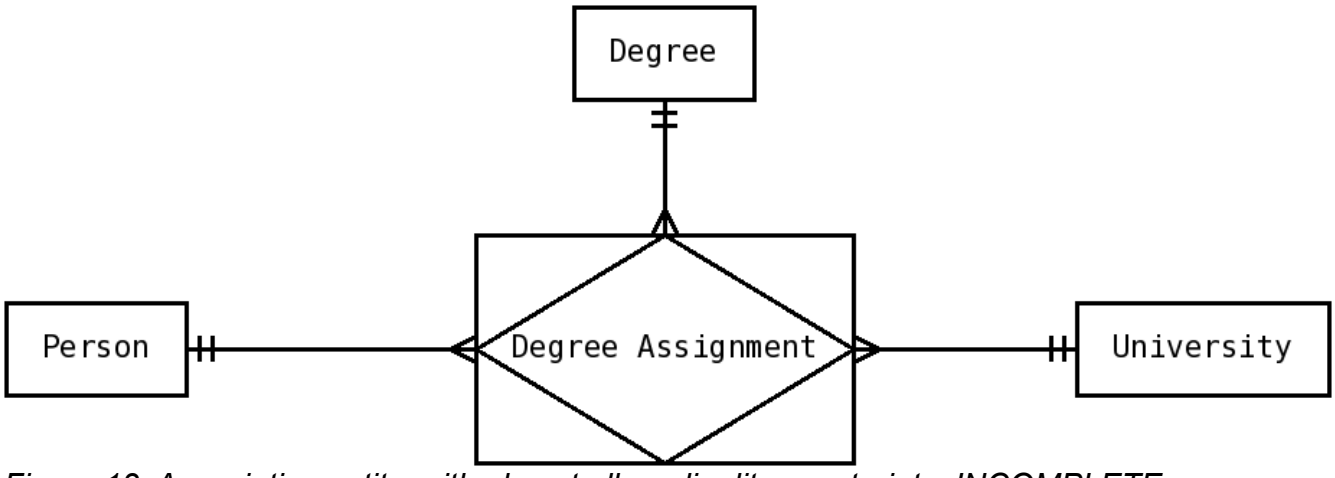

Figure 13: Associative entity, with almost all cardinality constraints. INCOMPLETE

So, once we have decided that the relationship is ternary and that we will model it as an associative entity, we have three out four choices (on each relationship) selected. The only possibly hard choice is the optional or mandatory cardinalities. This would depend on the particular situation, but, in this case, assuming it is optional on all sides we get the following diagram:

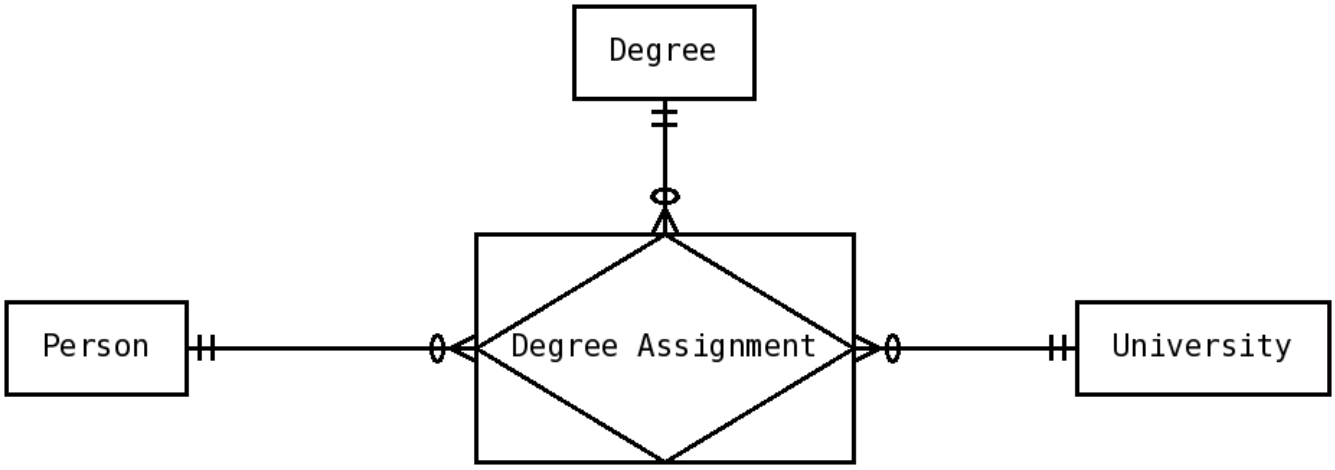

Figure 14: Ternary relationship as associative entities, with all cardinality constraints.

4 Exercises

1. We want to model information about TV Series and their episodes.
   - For each series, we keep its id (identifier and title).
   - We also keep track of the episodes of a series. For each episode we keep its number (which is unique among all episodes of the same series, but not among all episodes), title, and the date if first aired.
2. We want to model information about computer programs, their versions, and the features supported by each version.
   - For each program, we keep its id (identifier), name, and company. (Example: Firefox)
   - For each version, we keep its major and minor numbers (the major and minor numbers uniquely identify versions of the same program), its date of release, and the operating systems it runs on. A version of a program may run in more than one OS.
   - We also keep track of the features (of interest to us) that each version of a program supports. For each feature, we keep its id and a name. A version supports one or more features, and a feature is supported by zero or more programs.