CSCI 403: Databases
10 - ERD to Relational Schema Mapping

The Algorithm

In order for our ERD to be useful, we need to turn it into a relational schema for our database. This can be accomplished in a fairly straightforward fashion by following a few simple steps. The book provides the following seven-step algorithm, here presented in brief with examples drawn from the ERD created in the last lecture (see figure 1).

1 - Regular entities

Regular entities in the ERD become relations in the relational schema with all simple attributes of the entity becoming attributes in the relation. "Simple attributes" means that we take only the components of composite attributes, and we do not include multivalued or derived attributes. Multivalued attributes will be addressed in step 6 below. Note that additional attributes may be added on to this relation in later steps.

Choose one key of the entity and make its corresponding attribute(s) the primary key for the relation. Other keys can simply be created with unique constraints.

Looking at figure 1, we have three regular entities: course, instructor, and department. These three have only simple attributes, and single keys, thus the production of the schema is straightforward, and we obtain three initial relations:

course:
(course_id, title, hours)

instructor:
(name, office, email)

department:
(name, chair, ...)

Here we have underlined the primary keys. Note that one thing not captured in the ERD is the data type, so that is one design decision to make in the transition from ERD to schema. Also note that we have complete discretion to change entity and attribute names as necessary to make them valid for use in the database or to meet our own style requirements.

2 - Weak entities

Weak entities in the ERD become relations in the relational schema, again with all simple attributes becoming attributes in the relation. In addition to the simple attributes, the primary key attribute(s) of the relation arising from the owning entity are added to the new relation as a foreign key referencing the owning entity relation. The primary key of the new relation is the combination of the primary key borrowed from the owning entity and the partial key of the weak entity.

Looking at our example, we have one weak entity, section. It is owned by course and has a partial key attribute section id. It also has a multivalued attribute meetings, but since it is multivalued, we ignore it for now. We create a table named section:

section
(course_id, section_id).

The course_id and section_id attributes together form the primary key of the new relation. The course_id attribute is a foreign key referencing course_id in the course relation.

3 - 1:1 relationships

Let R be a 1:1 relationship between entities S and T. There are three basic choices (depending on the participation constraints). (Note: from now on, I will speak of the entities and the relations generated from them interchangeably. So S stands both for the entity S and for the relation created from it in steps 1 and 2.)

(a) Suppose S has total participation in R (the case when T has total participation is symmetrical). Then the first option we have is

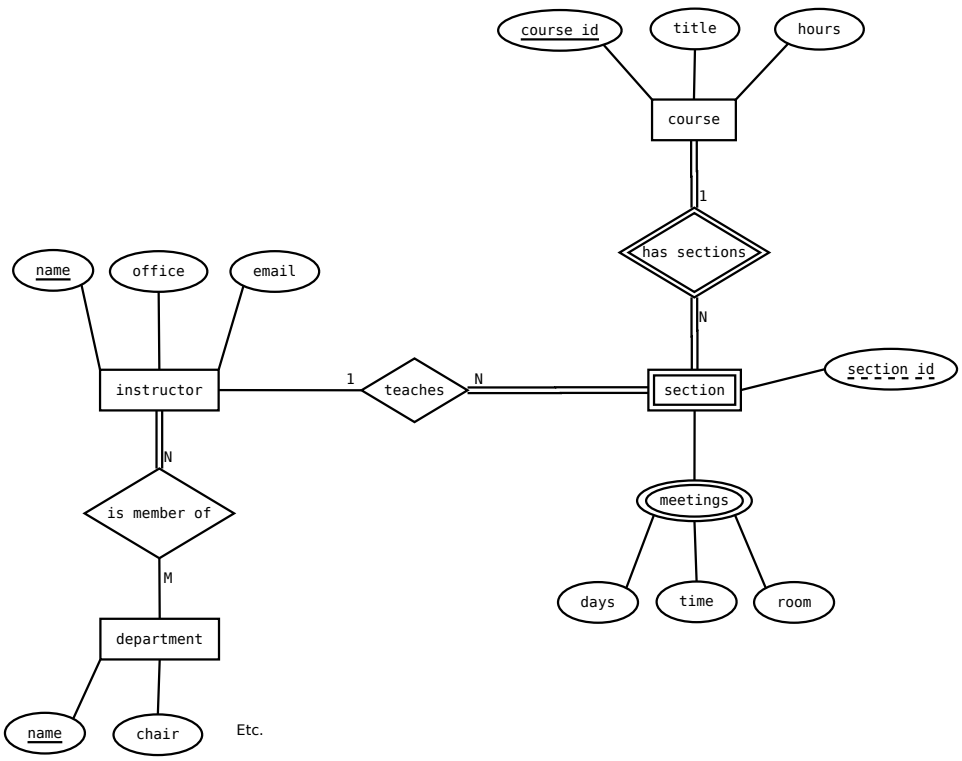

Figure 1: A model for a Mines courses database

to take the primary key of T and add it as an attribute in S. The borrowed attribute in S is made a foreign key referencing the primary key of T.

(b) If both S and T have total participation in R, then every tuple in S will correspond with a tuple in T and vice-versa. Thus it is safe, if desired, to merge S and T into one relation with all the attributes from both S and T. However, option (a) is preferred, particularly if the two entities model different concepts (simply choose one entity to take the primary key of the other).

(c) A third option, not recommended unless there is no total participation on either side, is to create a third relation as a cross-reference (or lookup) table. This option will be discussed further below when we discuss N:M relationships.

In our example, there are no 1:1 relationships. However, option (a) is essentially the same as for option (a) in the 1:N relationship mapping, so we will move on.

4 - 1:N relationships

Let R be a 1:N relationship between entities S and T, with T being on the 1 side and S on the N side.

(a) If S has total participation in R (which is more often the case than not), then once again we can take the primary key from T and add it to the relation S as a foreign key referencing back to T. Note that this is not a symmetrical option: since S can have many tuples for each tuple of T, there can be more than one primary key value in S which would have to be associated with a single tuple of T.

(b) The second option is the same as option (c) above: create a cross-reference table.

Our example has two 1:N relationships. The "has sections" relationship is already taken care of; the primary key from course is already included in the section relation, so nothing more needs to be done. The teaches relationship is also 1:N, with instructor being on the 1 side. Thus we take the primary key from instructor (name) and add it to the section relation. Noting that name is not particularly descriptive, we'll take the liberty of renaming the attribute in the section relation to instructor_name. The modified section relation now looks like:

section
(course_id, section_id, instructor_name).

We make instructor_name a foreign key referencing name in instructor. The previously created primary and foreign keys are unchanged.

5 - N:M relationships

In the case of N:M (many-to-many) relationships, there is no way to add a key from one relation to the other to represent the relationship. The only possibility in this case is to create a cross-reference, or lookup table. The table contains primary key attributes from both relations, and each entry in the table is a pair representing the existence of a relationship between a tuple on one side with a tuple on the other side.

More formally, suppose S and T are the two relations participating in the relationship, with S having primary key P_S and T having primary key P_T. Then we create a third relation, Q, with attributes P_S and P_T. The primary key for the new relation is composed of both attributes. The attribute P_S is set as a foreign key referencing P_S in S, and likewise P_T is a foreign key referencing P_T in T.

In our example, "is member of" is an N:M relationship between instructor and department. We must create a new relation, the name of which is up to us; in general follow the style that your organization uses. I will take the approach of calling the new relation S_T_xref (or in our example, instructor_department_xref). Note that the primary keys of both instructor and department are named "name", so at least one must be renamed in the new table; since both are non-descriptive, I will rename them both.

Here's the new table:

instructor_department_xref:
(instructor_name, department_name),
with instructor_name being a foreign key referencing name in instructor, and department_name being a foreign key referencing name in department.

Now we can easily retrieve all departments of

an instructor or all instructors of a department by joining the three tables and applying the necessary where condition, e.g.

```
SELECT i.name
FROM instructor AS i,
department AS d,
instructor_department_xref AS x
WHERE d.name = x.department_name
AND i.name = x.instructor_name
AND d.name = 'Computer Science';
```

6 - Multivalued Attributes

Mapping a multivalued attribute onto a relational schema is essentially the same as mapping a weak entity, treating the multivalued attribute itself as the partial key and only attribute of the weak entity. That is, we create a new relation which contains as attributes the (simple components of) the multivalued attribute plus the primary key attribute from the owning relation.

The primary key of the resulting relation is composed of all attributes. The primary key attribute from the owning relation, as usual, is made into a foreign key referencing the owning table primary key.

In our example, meetings is a multivalued attribute of section. The meetings attribute is also composite, so in our new table we will have attributes for each component of meetings. We also borrow the primary key from the owning relation; in this case, the owning relation is section. Looking back through our work so far, we see that the primary key of the owning relation is also composite, made from course_id and section_id. Accordingly, we create a table (again name is up to us):

section_meetings
(course_id, section_id, days, time, room),
with (course_id, section_id) as a foreign key back to (course_id, section_id) in section.

7 - n-ary relationships

In general n-ary relationships (relationships between 3 or more entities) must be modeled using a cross-reference table. There are some subtleties that can arise (see the book for details), but generally you can get away with creating a cross-reference table from the n primary keys of the participating relations.

Notes

In the above, I didn't address what happens when a relationship itself has attributes. What you do depends on the nature of the relationship and the choices you made in mapping the relationship onto the schema. If you chose a cross-reference table option, then adding the attributes into the cross-reference table is the simplest and most effective option. Otherwise, you will need to add the attributes into one of the participating relations, or create an additional table just for the relationship attributes to live in (with appropriate foreign key references).