Lecture 8 — Modelling Diagrams

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

August 11, 2023

Good Design

We have talked at some length about the tools for defining the database, but we really should spend significant time on good design.

Knowing how hammers work is not enough; we also need to understand what good architectural plans look like.

The first thing we would like to watch out for is redundancy in the database and eliminate it wherever possible.

In short, if there is redundant data then there is the potential for data to get out of date or be in some other inconsistent state.

Design with Diagrams

When we have some data that we would like to represent we need to turn it into some database tables.

Going directly from some ideas about what data we need to the tables is sometimes difficult and may produce undesirable results.

Instead, we should devise a plan for how it should be implemented and for that we will create diagrams.

Diagrams are done on paper or a whiteboard or similar and it is very easy to visualize, understand, and change the design at that stage.

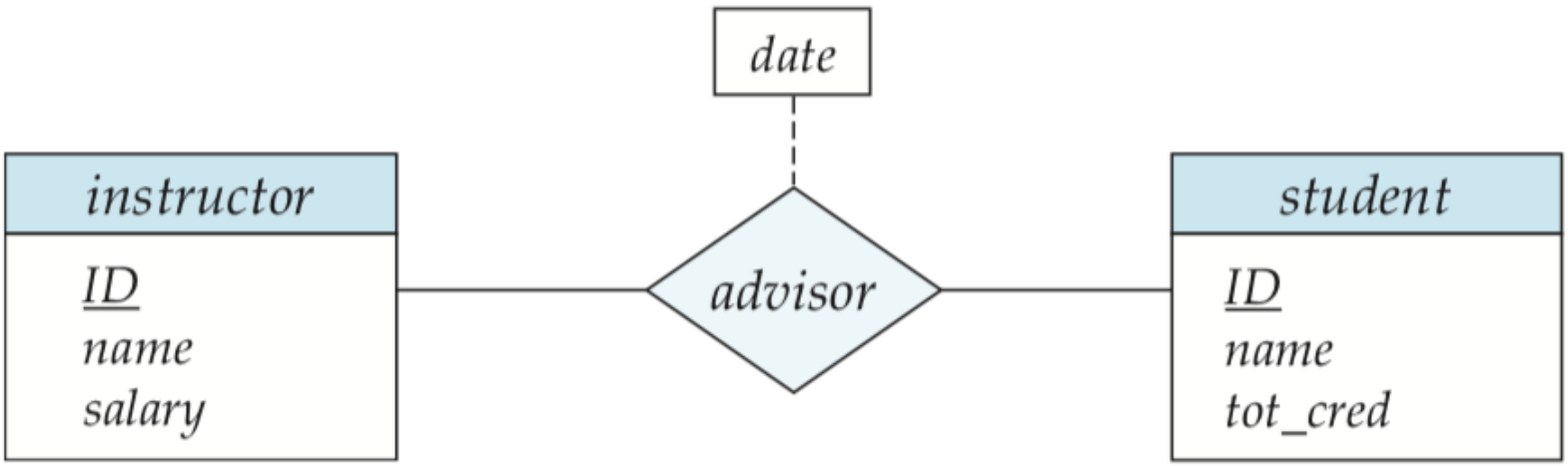
Entity-Relationship Diagrams

Entity-relationship diagrams have the following symbols and their meanings:

- **Divided Rectangles** represent an entity; an entity has a title (the top part) and a list of the attributes of that entity (the bottom part).
- **Diamonds** represent a relationship set.
- **Undivided Rectangles** represent the attributes of a relationship set.
- **Lines** link entity sets to relationship sets.
- **Dashed Lines** link the attributes of a relationship set to its relationship set.
- **Double lines** indicate total participation of an entity in a relationship set.
- **Double Diamonds** represent identifying relationship sets linked to weak entity sets.

Note also that in rectangles, attributes that are or form part of the primary key are underlined.

E-R Diagram Example



Diagrams with Different Cardinalities

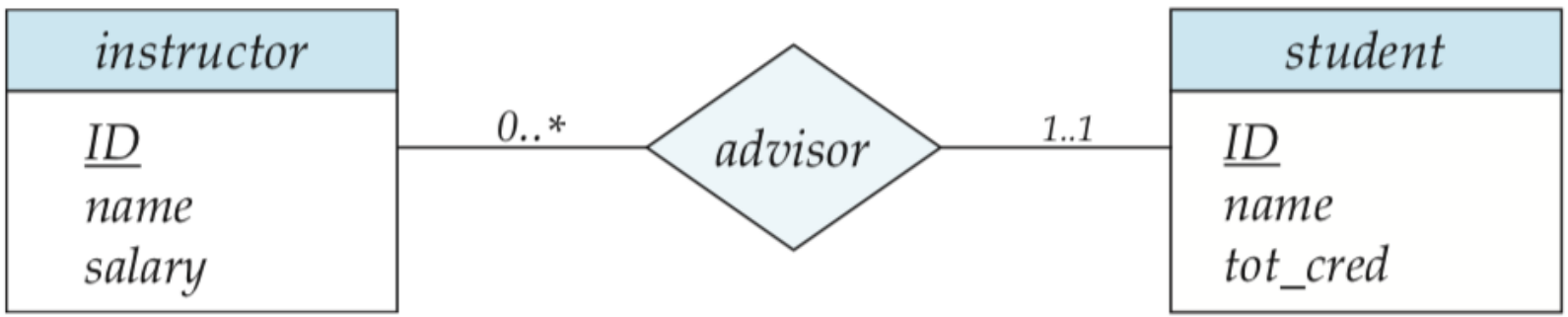
(a)

(b)

(c)

Specific Numbers

In the diagram below, an instructor can have zero or more students and a student can have exactly one advisor.

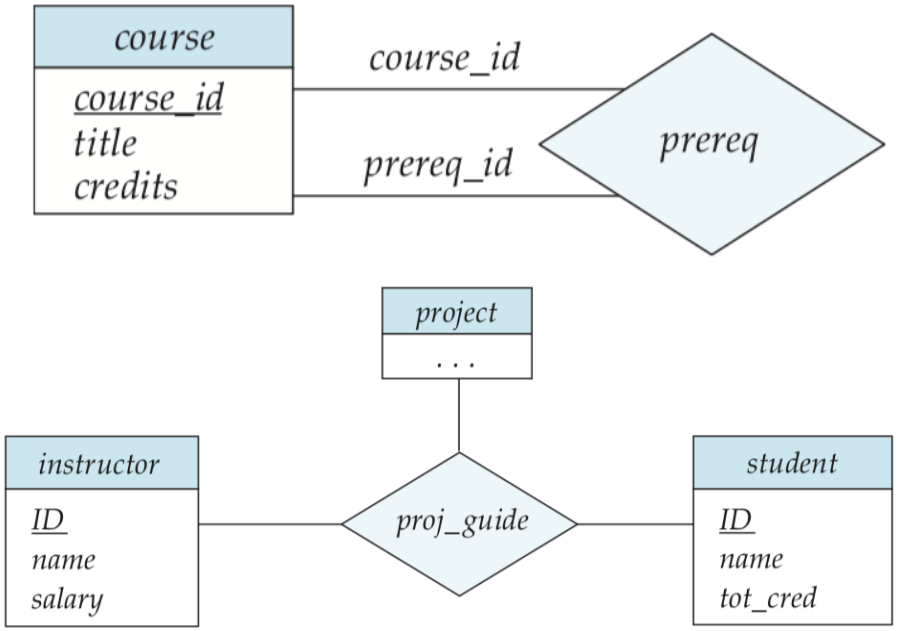


Non-Binary Relationships

There is no reason why a relationship must be binary.

A table may reference itself, as in the case of courses that have prerequisites.

Or a relation may involve many relations as in a ternary relationship:



Weak Entity Sets

Weak entity sets are not entities that do not even lift.

Instead, they are entities that are dependent on another entity, namely, one that cannot be identified uniquely based on its own attributes.

The example that builds on this “university” model is a section: a section of a course, such as 001, is dependent on a course in a particular term.

Weak Entity Set: Section

Suppose that a course has an id, name, and code.

A section has a section number, year, and term.

A particular course, ECE 356, will have one or more sections, e.g. 001.

But the attributes of the section on their own (001, 2018, Winter) are not enough to uniquely identify a single course.

Weak Entity Set: Section

Because at least one another course, such as ECE 459, also has a section that matches (001, 2018, Winter).

To differentiate between two sections that match on those same criteria we need a course number, but the course number isn't a part of the section entity.

That is part of the course entity. And this is what makes the section entity a weak entity.

Weak Entity Set

More formally, an entity that on its own does not possess sufficient attributes to form a primary key is a weak entity.

One that is not weak is a strong entity.

A weak entity set is associated with another entity, called its identifying set or its owner entity set.

If Lost, Please Return To...

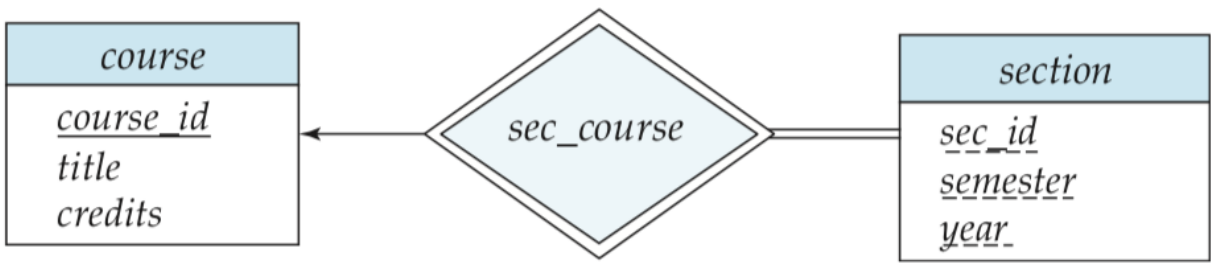
In a practical sense the weak entity will probably have some attribute added to it that identifies which strong entity the weak entity belongs to.

That is called the discriminator.

That would mean some attribute added to the section entity that references a particular course.

If Lost, Please Return To...

Still, that is an implementation detail that plays no role in the E-R diagram, which looks like the diagram below:



Design Alternative: Strong Entity

We could choose a different option, of course, and add some sort of unique identifier to the section that would promote the weak entity to a strong one.

But that doesn't necessarily make logical sense in the context of the application.

A section doesn't make sense on its own and giving it a unique identifier does not really correspond with reality.

A section is logically dependent on the course, isn't it?

Weak Entities

Weak entity sets can exist in other configurations than just the identifying relationship.

A weak entity can be the owner of another weak entity, or a weak entity can belong to more than one identifying set.

When to Use Weak Entities

When does it make sense to have a weak entity?

You might ask yourself if a particular entity can exist independently of any other entity.

In an e-commerce scenario, there are customers, who have orders composed of items.

Customers can exist if they don’t have any orders, and items can exist if they don’t appear in any orders.

In that case, we would expect that both customer and order both to be strong entities.

Customer Reviews

Continuing this analogy, what about product reviews?

A review belongs to an item, and if it is not anonymous then it also belongs to a user.

But a review does not make sense independent of an item, otherwise, what would it be a review of?

In this case, review can be modelled as a weak entity and it can have one identifying set (item) or two (item and customer).

We could assign it an ID and promote it to a strong entity...

There must be at least SOME strong entities in the system.

Specialization and Generalization

Specialization is the same as a subclass in object-oriented-programming.

We have some parent class (e.g., user) that has some number of attributes like ID, name, email, et cetera.

And then there are some particular classifications: a staff member is a user, but has certain staff specific attributes like office and phone number.

A customer is a user but has different attributes like frequent flyer number, airline status, et cetera.

Specialization and Generalization

The mirror image operation of this is generalization, which is analogous to extracting a superclass in object-oriented programming.

In that case we would identify some common attributes between certain entities.

Consolidate the common attributes in a single entity from which the other ones are derived.

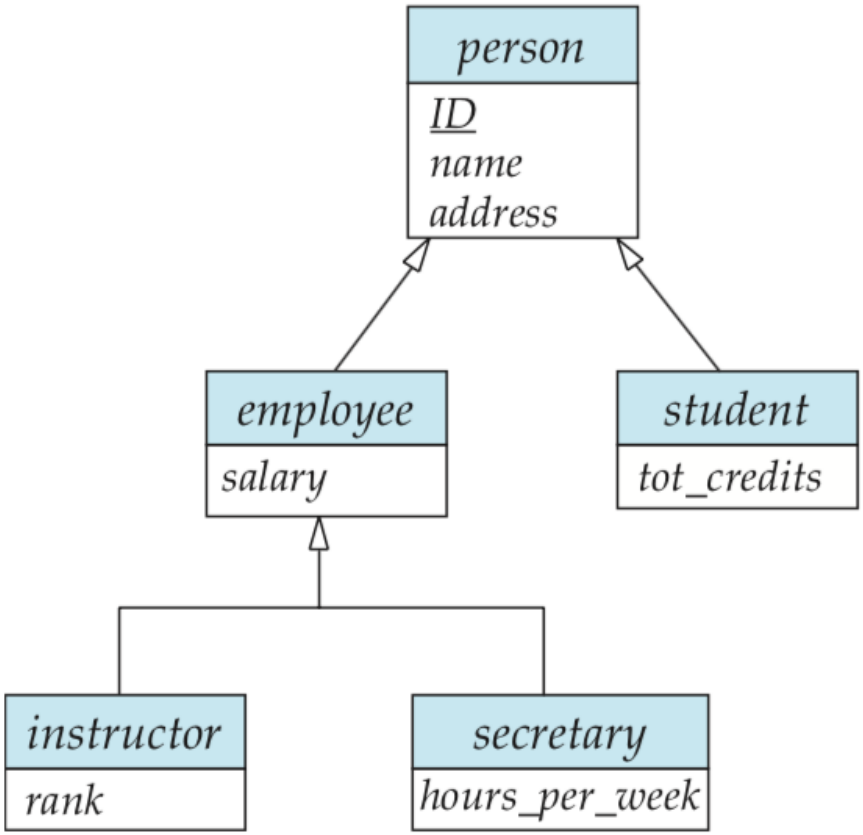
Specialization and Generalization

Obviously, this only works if we have attribute inheritance: sub-entities receive the attributes of their super-entities.

As you know about OOP we don’t have to go into this topic.

Note: Information hiding (encapsulation) principles aren’t present in the database design.

Specialization and Generalization



Definition of Generalization

We have some choices about how the generalization is defined.

The first option is to have it be condition defined, such as having some rules that say that a person who fulfills a certain condition is included in that group.

So a customer who has status of “Frequent Flyer” in the system is automatically included in that specialization.

In other cases, an operation must be taken to specifically include a person in that subset, such as assigning an employee to a specific department

Disjoint or Not

We could also have rules that say sets must be disjoint (a member of some group cannot be a member of another group), or if they can be overlapping.

In our system we could say that a staff person can also be a customer, but that is specific to the domain we are discussing.

We could write in some rules that say that no instance of “user” can exist, and everyone must be a staff member or customer (or both).

This is called total specialization.

Aggregation

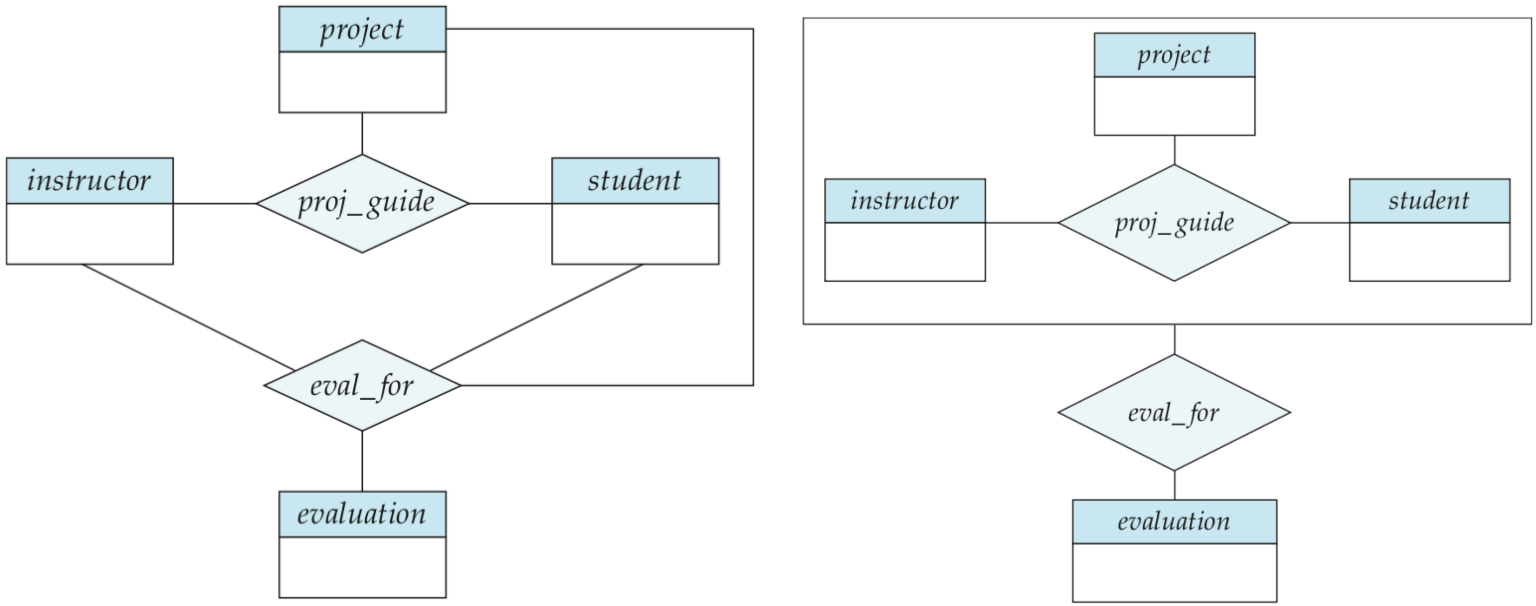
Aggregation allows us to treat a grouping of entities that are related as a single block.

This can simplify the diagram and show us a relationship between relationships.

Aggregation may not seem very useful in the context of a database schema that can be handily represented on a piece of paper.

For a large database it can be a good way to sequester parts of the design in a different area so the diagram is comprehensible.

Aggregation Example



Aggregation Example

It looks like the relationship *eval_for* and *proj_guide* can be combined because they seem to connect the same three things.

But we may not want to do that if some sets of (instructor, student, project) do not have an associated evaluation.

Now possibly the data is redundant!

If the evaluation is a simple enough element (e.g., it is an enumeration or an integer letter grade) then it could be an attribute of the *proj_guide* relationship.

But this option is not suitable if the evaluation relation is used in some other relationship or in some other context.

We also might not be able to combine it if there are multiple evaluations on a project...

Aggregation Example

We can consider *proj_guide* a higher level entity, leaving a binary relationship between that aggregated entity and the evaluation relation.

The textbook contains a couple of alternative diagram notations.

You may encounter them in the real world, but we will not invest any time discussing them as they could be confusing.

Given a good understanding of how diagrams are formed, we will next think about how to turn models into tables, and what makes up a good design.