Databases: Entity-Relationship Modelling

Gordon Royle

School of Mathematics & Statistics
University of Western Australia

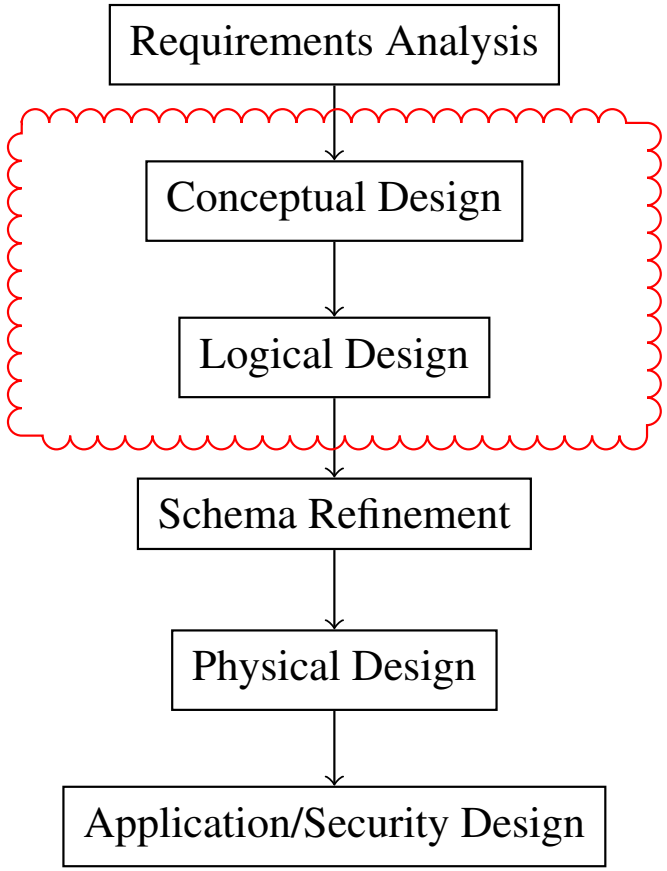
Database Design Process

Ramakrishnan & Gehrke (the authors of *Database Management Systems*), identify six main steps in designing a database:

- Requirements Analysis
- Conceptual Design
- Logical Design
- Schema Refinement
- Physical Design
- Application & Security Design

Database Design

ER-Modelling



Requirements Analysis

Requirements Analysis is the process of determining *what* the database is to be used for.

It involves interviews with user groups and other stakeholders to identify what functionality they require from the database, what kinds of data they wish to process, and the most frequently performed operations.

This discussion is at a non-technical level and enables the database designers to understand the *business logic* behind the desired database.

Conceptual & Logical Design

Using the ER data model, the conceptual design stage involves identifying the relevant *entities* and the *relationships* between them and producing an *entity-relationship* diagram.

The logical design stage involves translating the ER diagram into actual relational database schema.

Once a suitable ER model has been constructed, then this can be translated into a relational database fairly easily.

However data modelling is a *craft* more than a science, and often there are trade-offs to be made, meaning that there is not usually a single perfect solution.

After the modelling

The remaining steps involve the *effective use* of the database after its fundamental structure has been determined.

This involves:

- Mathematical analysis and refinement of the schema
- Performance-based decisions on indexing, machine capacities, required performance etc.
- Interfacing with client applications and security

Iterative Process

As with all software engineering processes, information uncovered during later phases of the project may alter some design decisions so the nice neat diagram of the 6 phases is really an *iterative process* where each stage feeds back to the previous stages.

One of the major causes of design alterations is *incomplete requirements analysis* — this is frequently attributed to users not being aware of the possibilities until they start using the system, or at least a prototype.

Running Example

We use a modified version of an example from the *Database Management Systems* text, which is to produce a data model that will capture the following information:

- A lecturer has a *staff number*, a *name* and a *rank*
- Research projects have a *project id*, a *sponsoring organization* and a *budget*
- Each project has one lecturer as the *principal investigator* (PI)
- Each project may have other lecturers as *co-investigators*
- Each lecturer can be a principal investigator or co-investigator on multiple projects

Entity Sets

An *entity set* is just a set of "things", or "objects", that are involved in the process that is being modelled.

We've seen examples where the entities are *sailors*, *boats*, *students*, *units*, *cities*, *countries* and so on.

Each of the *relevant entities* will determine one of the tables in the database.

In the example, we'll need `Lecturer` and `Project` at least.

Keys

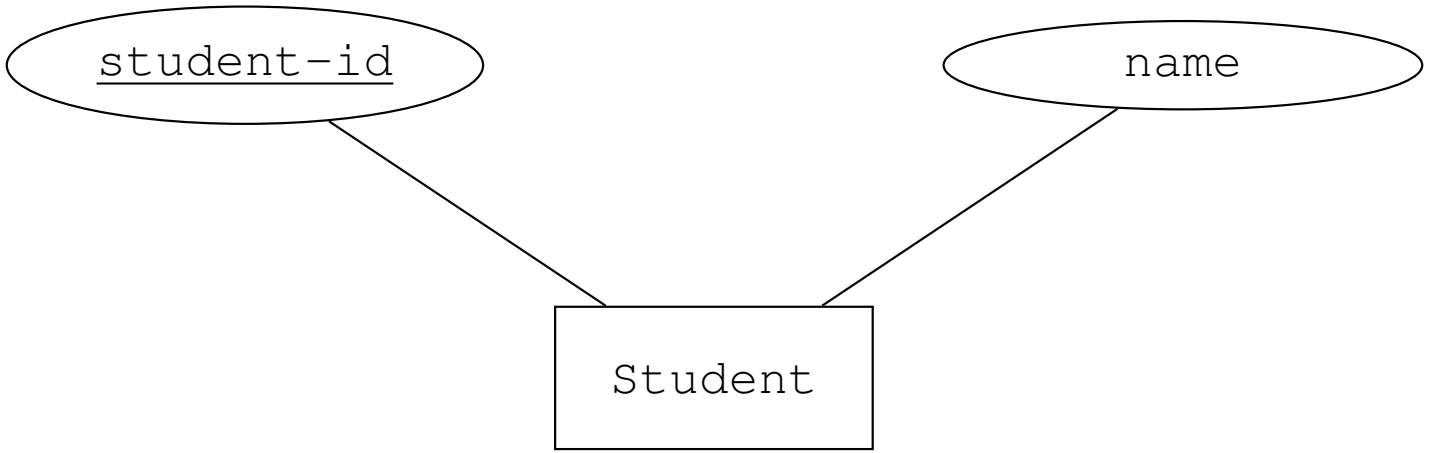
One of the mathematical properties of a *set* is that there should be *no duplicate elements*, and in most cases, a table representing an entity should have *no repeated rows*.

A *key* for an entity set (i.e. table) is *an attribute* or a *combination of attributes* that uniquely identifies an entity (i.e. row).

Often an artificial key is used — numbers such as *student number*, *staff number*, *invoice number* and so on — but sometimes there are natural keys.

Example key

A `Student` is always uniquely identified by their `student-id` so this attribute is a suitable *key* for this relation.



In an ER diagram, an entity set's key is designated by *underlining* the attribute or attributes that form the key.

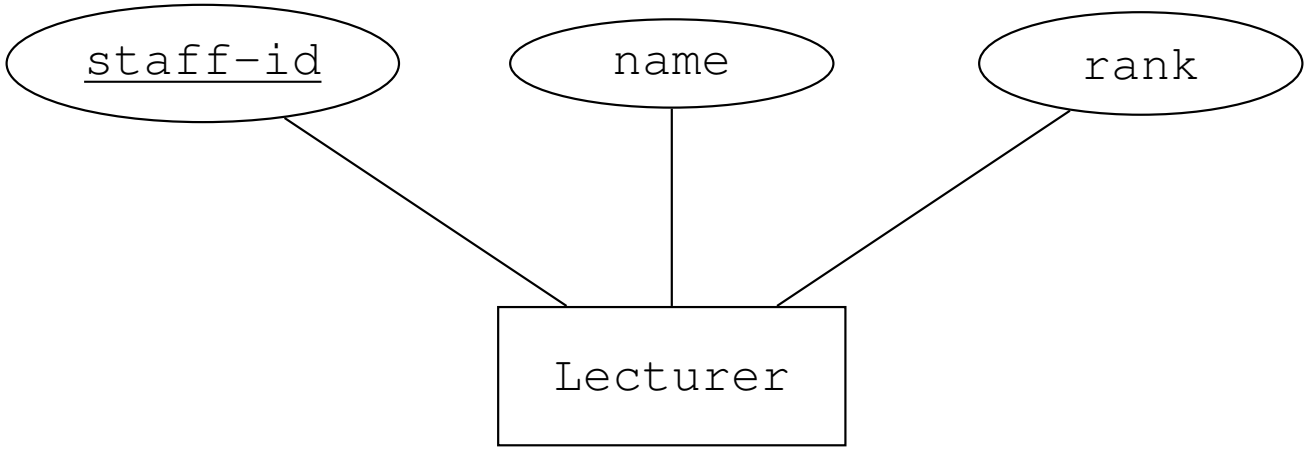
Why are keys important?

It is important to identify the key for an entity set (in fact for *any table* in a database) for several reasons:

- Explicitly identifying a key helps to ensure that the data model is logically consistent.
- When implemented, the DBMS can enforce *key constraints* that ensure that the nominated key does indeed uniquely identify each row.
- The DBMS can *index* the table using the key values and manipulate that relation very efficiently.
- The DBMS can enforce *referential integrity* by ensuring that tables that refer to *other tables* remain in a consistent state — this comes later!

Example – Lecturer Entity

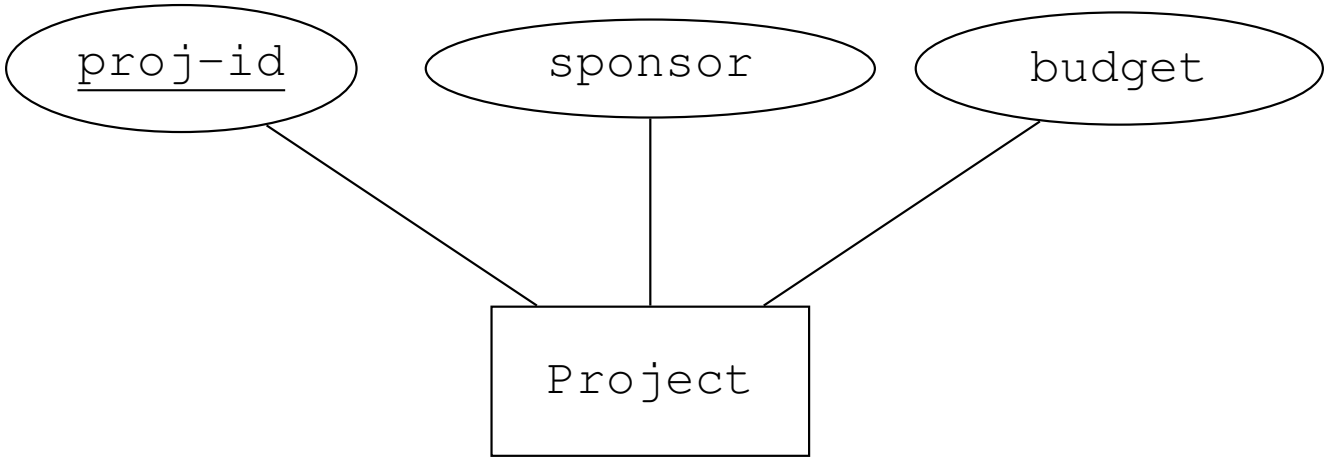
Modelling of the *entities* in this example is fairly easy:



So the actual entities would be things like

staff-id	name	rank
00144238	"J Smith"	"Associate Professor"

Example – Project Entity

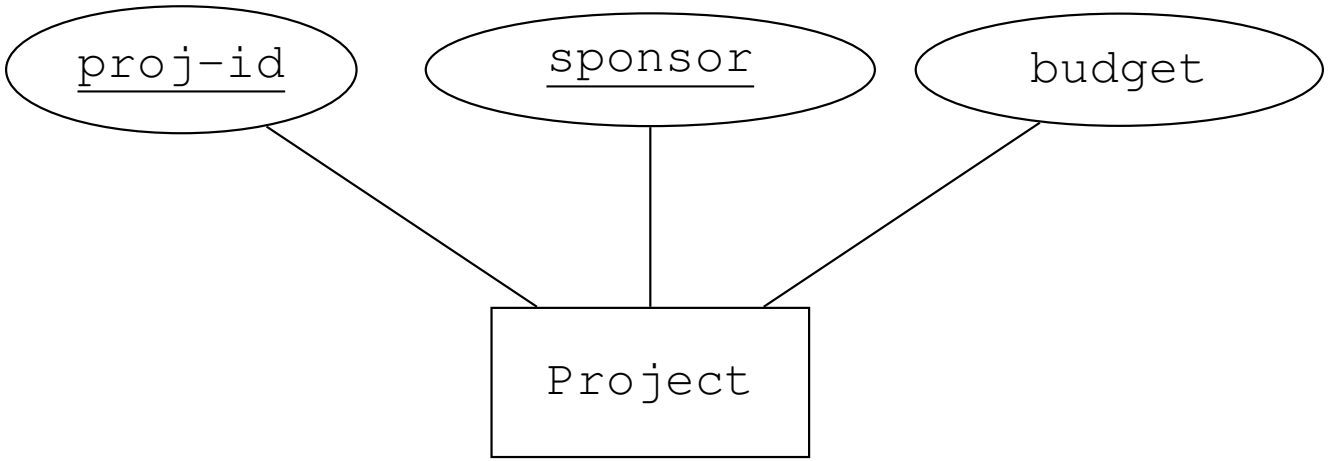


So the actual entities would be things like

prof-id	sponsor	budget
DH10304	ARC	$120000

Hmmm, what about the key

Although it is unlikely that two sponsoring organizations use the same format for project id numbers, it is *possible*. So maybe the key should consist of the *combination* of two attributes `(proj-id, sponsor)`?



In this case, the attributes that form the key are *all* underlined.

Relationship Sets

The power of relational databases comes from the ability to model and query *relationships between entity sets*.

If $E_1, E_2, \ldots, E_n$ are entity sets, then a *relationship set* is a subset

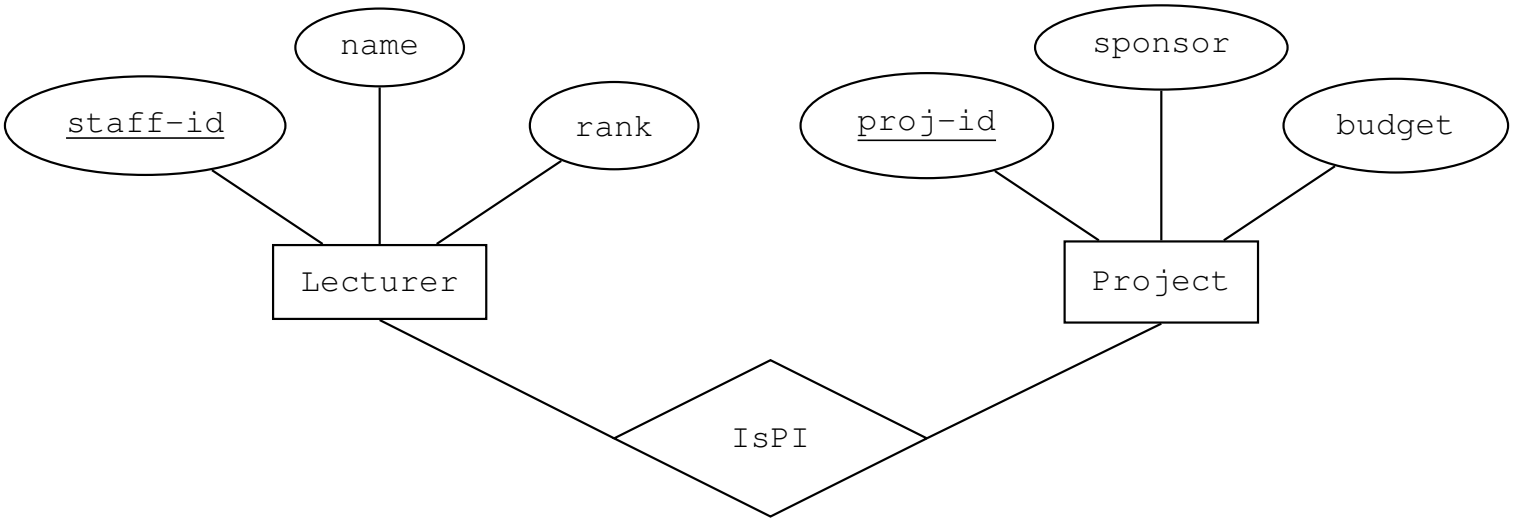
$$R \subseteq E_1 \times E_2 \times \cdots \times E_n$$

In other words R is an n-ary relation whose elements are *entities*.

As entity sets are *relations*, this means that we are using relations/tables to model both entity sets *and* relationship sets!

Diagramming Relationships

A *relationship* is diagrammed by a named *diamond shape* that is connected by lines to the related entity sets.



Here we are modelling the relationship `IsPI` which relates projects to their principal investigators.

Elements of the relationship set

The actual *elements* of the relationship set are the *pairs* that specify *which lecturer* manages *which project*.

Thus, if Associate Professor Smith manages the project DH10304 then that *pair of entities* would be an element of the `IsPI` relationship set.

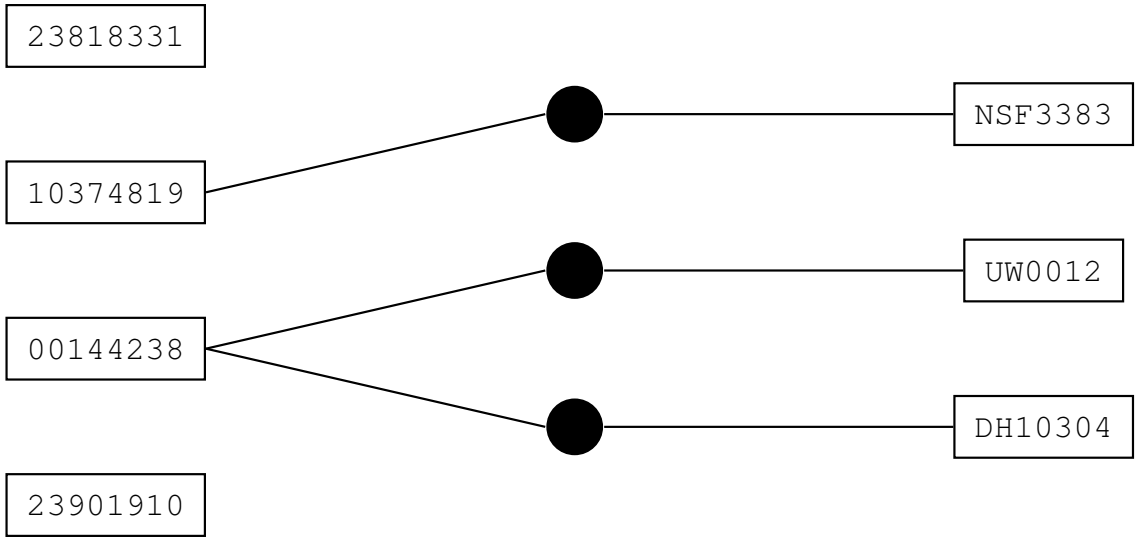
Notice that we can *unambiguously specify* this pair just by storing the *keys* for this pair:

$$(\texttt{00144238}, \texttt{DH10304}) \in \texttt{IsPI}$$

This of course is exactly how an RDBMS stores a relationship set.

Participation Constraints

Consider the following instance of the relationship set `IsPI`.



Participation Constraints

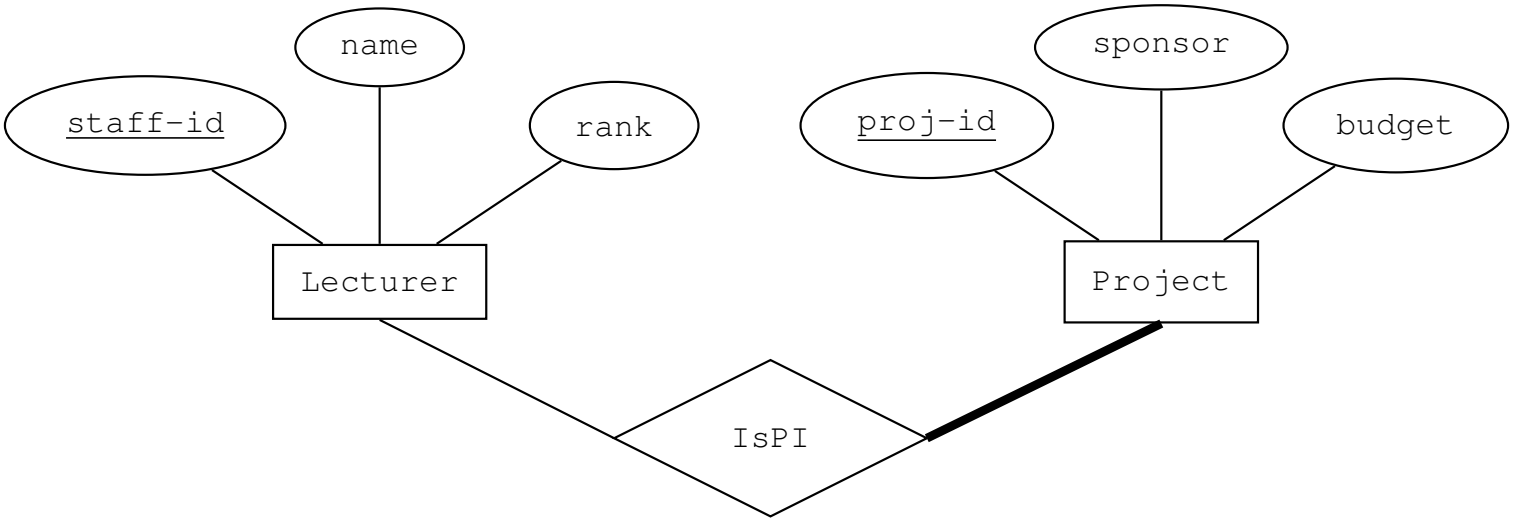
Sometimes the participation is a *constraint* based on the business logic, rather than just an accident of the data at this particular time.

For example,

- *Every* project must have a principal investigator, and so there is *total participation* of the entity set `Project` in the relationship set `IsPI`.

This is indicated in the ER diagram by a *thick black line* connecting the entity set with the relationship set.

ER diagram with participation constraints



Key constraints

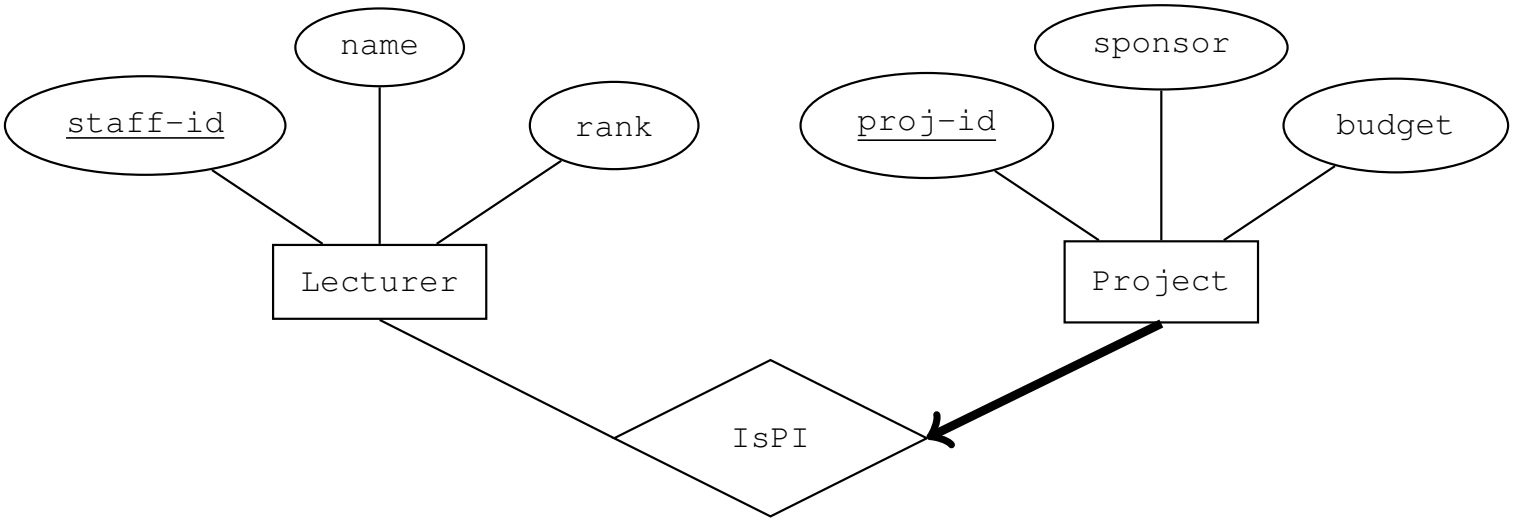
The relationship `IsPI` is *one-to-many* because each project has just *one* principal investigator.

This is called a *key constraint* because it means that in any allowable instance of `IsPI` each entity from `Project` appears at most once.

In particular, this means that the key for `Project` can be used as a key for `IsPI`.

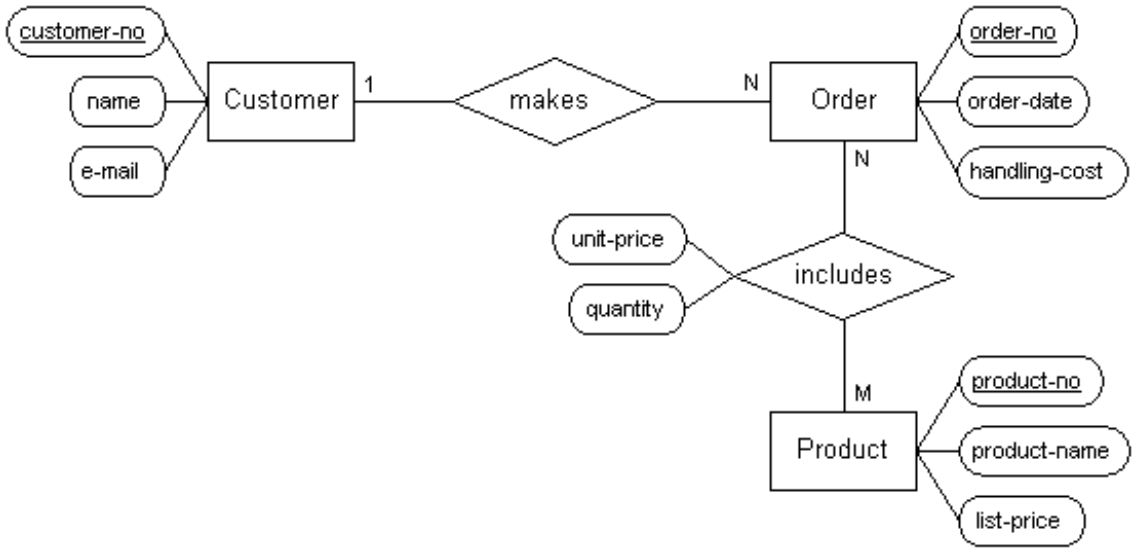
This is indicated in the ER diagram by an arrow-head on the line connecting the entity set with the relationship set.

ER diagram with participation and key constraints



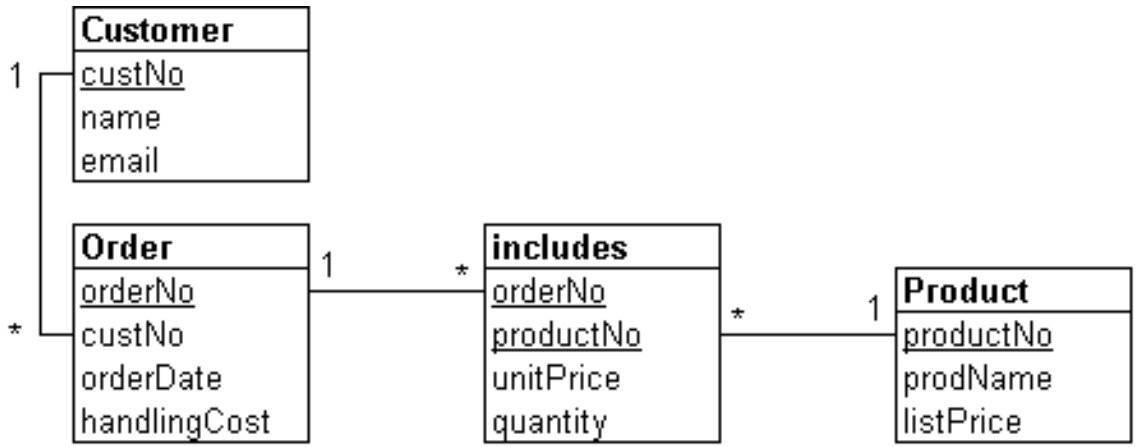
Warning

There are a number of different *data modelling techniques*, and even when using a single technique, there are different *diagramming conventions* used for that technique.



Structure Diagram

A DB structure diagram contains similar information to an entity-relationship diagram (ERD), but with slightly different symbols — however diagrams of this type are also often called ERDs.



A DB structure diagram is closer to the actual database implementation.

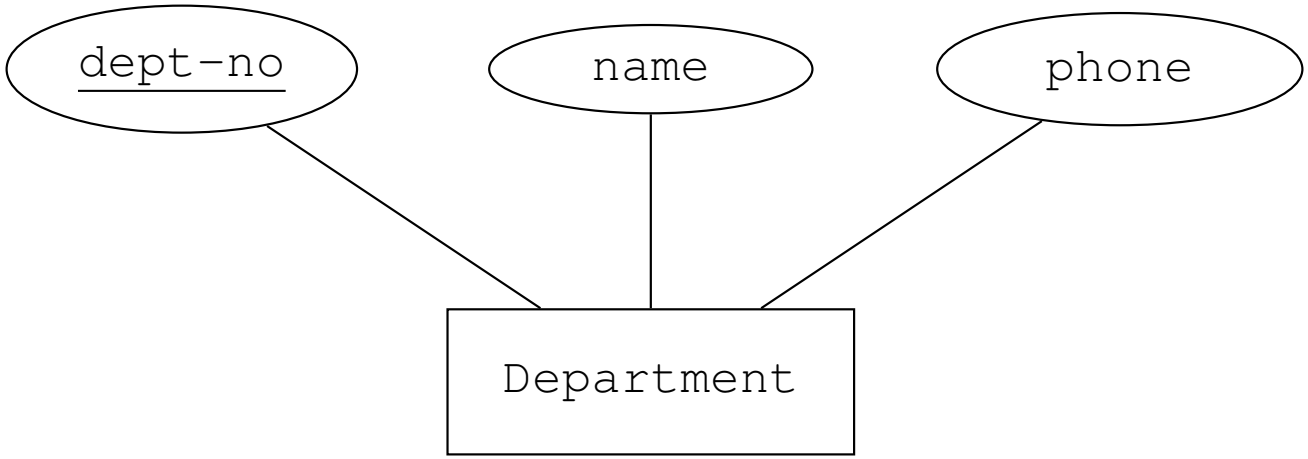
Additional Requirements

So far we have only considered a few of the requirements for the university database — here are some more.

- Departments have a department number, a department name and a contact phone number
- Departments have a lecturer who is the head of department
- Lecturers may work in more than one department, in which case they have a percentage of their time associated with each department

Another entity set

These requirements introduce another straightforward *entity set*.



By now the procedure for identifying entity sets and their attributes and key should seem straightforward — at least for these very well specified small examples!

Headship – participation constraints

The `Head` relationship is between lecturers and departments.

First we cover the *participation constraints*.

Every department has a head of department, and so there is *total participation* of the entity set `Department` in this relation, to be indicated by a thick black line.

However not every lecturer is a head, and so there is only *partial participation* of the entity set `Lecturer`, hence just a normal line.

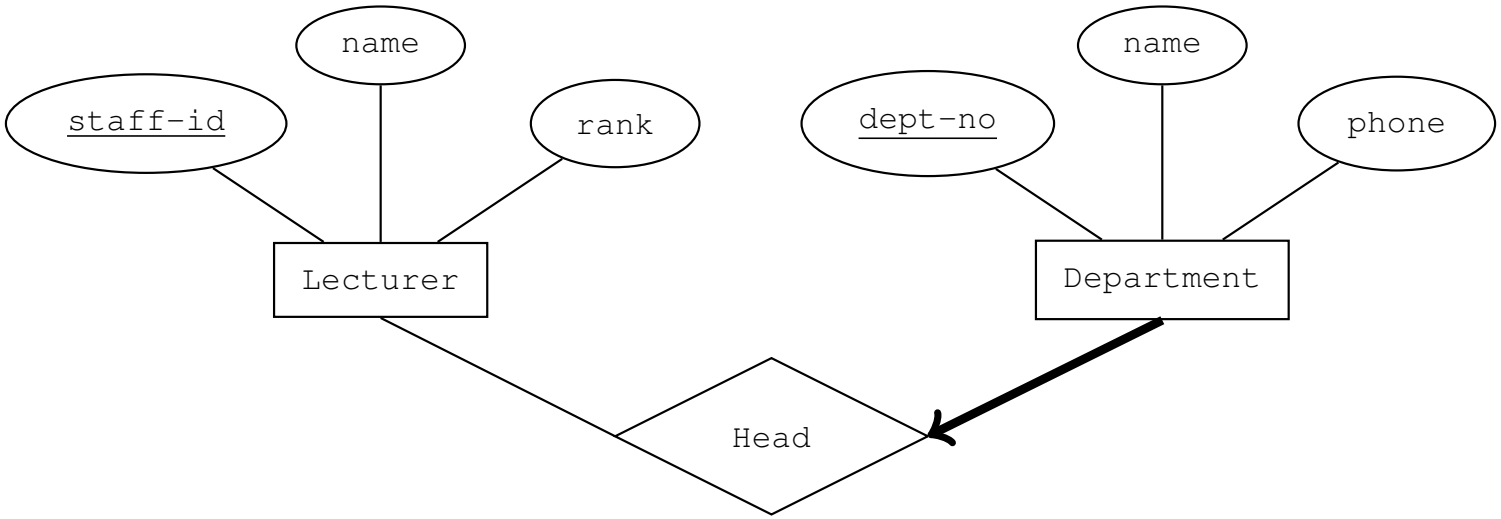
Headship – key constraints

Next we cover the *key constraints*.

Each department has exactly *one* lecturer as head and so here we have a key constraint — each *department* can only appear in one tuple of the relationship set.

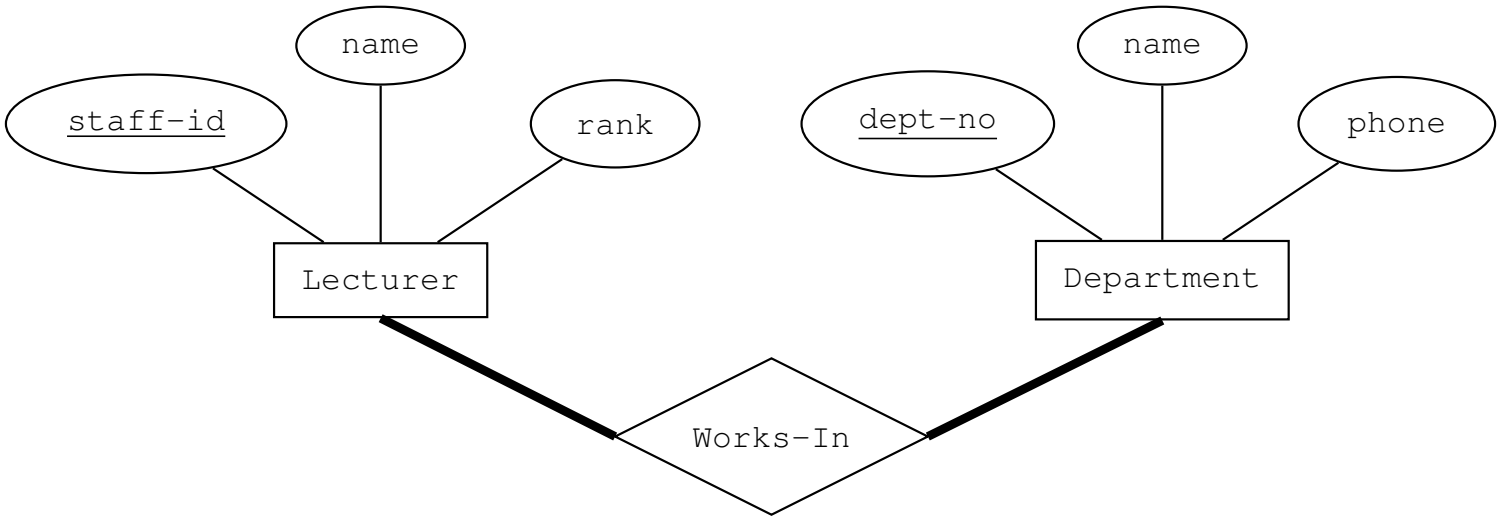
This is indicated by an arrow which goes *from* the entity set that is constrained to appear only once.

ER diagram with participation and key constraints



The Works-In relationship set

At first sight the Works-In relationship set is straightforward:



Note the thick lines indicating that every lecturer works in some department, and every department has some lecturers in it.

What's the problem?

Suppose Associate Professor Smith works 60% of the time for CSSE and 40% of the time for Maths.

The ER diagram gives us no mechanism for storing the percentages at the moment.

We cannot extend the entity set `Lecturer` to contain a field `percent` because that field makes no sense when considering a lecturer in isolation.

staff-id	name	rank	percent
00144238	"J Smith"	"Associate Professor"	??????

Similarly we can not add this field to the entity set `Department`.

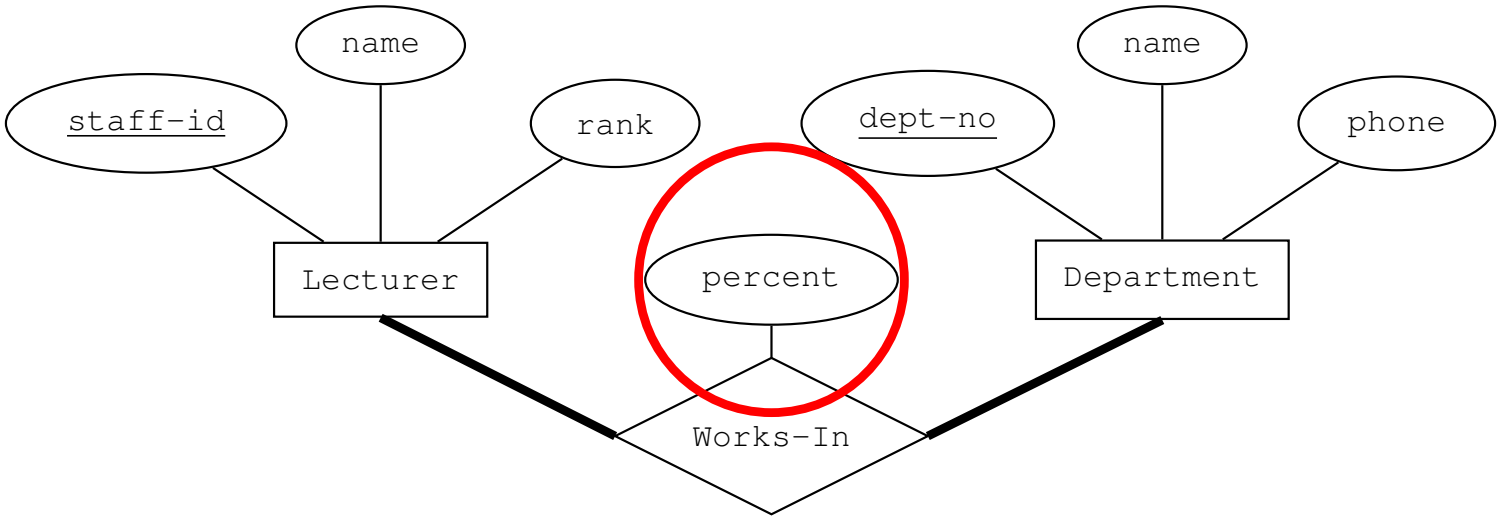
Relationship Attribute

The percentage time is not an attribute of a lecturer alone, nor of a department alone, but in fact is an attribute of a *relationship* between a lecturer and a department.

So in our example, the 60% amount is associated with the *pair* of entities (`Smith`, `CSSE`) and the 40% amount is associated with the *pair* of entities (`Smith`, `Maths`).

So `percent` is actually an attribute of the *relationship set*, not of any of its constituent entities.

Diagramming relationship attributes



An oval-shaped attribute is added to the relationship symbol to indicate that this attribute "belongs to" the relationship itself rather than to its components.

Formally

R & K sidestep this issue, but their formal definition of a relationship set needs to be modified.

If $E_1, E_2, \ldots, E_n$ are entity sets and $A_1, A_2, \ldots, A_m$ are sets (i.e. attribute sets) then a *relationship set* is a subset

$$R \subseteq E_1 \times E_2 \times \cdots \times E_n \times A_1 \times \cdots A_m$$

We think of an element of the relationship set as consisting of n entities (which of course possess their own attributes), together with m values that describe aspects of the relationship.

Example

Thus for our example, if L is the lecturer entity set, and D the department entity set and $P = \{p \mid 0 \leq p \leq 100\}$ then

$$\texttt{Works-In} \subseteq L \times D \times P$$

So now we can say sensible things like

$$(\texttt{Smith}, \texttt{CSSE}, 60) \in \texttt{Works-In}$$

to capture the fact that Smith is 60% associated with CSSE.

Notice however that the ER model is *NOT* strong enough to specify various other types of "business logic" such as ensuring that the percentages for each lecturer add up to 100.

Relationships between a set and itself

In our definition of a relationship set, there was no requirement for the entity sets E_1, E_2, etc to be *different*.

Suppose that the university commences a formal system of *peer review of teaching* and specifies that:

- Each lecturer has another lecturer, called a *peer reviewer* who attends a few of their lectures and provides feedback

Thus we have a binary relationship set in which *both* components come from the `Lecturer` entity set.

Roles

In this context an element of this relationship set is a *pair* of lecturers such as

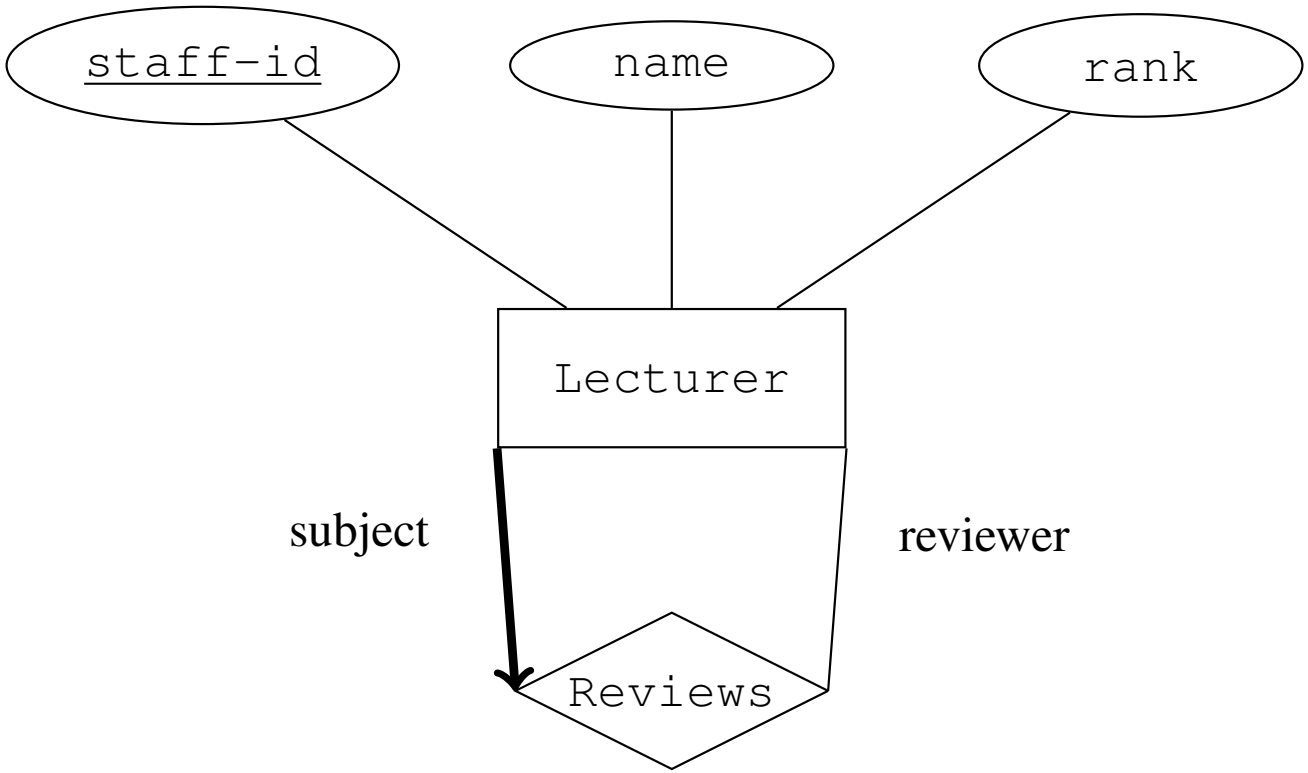
$$(\texttt{Smith}, \texttt{Jones})$$

Although both components are from the `Lecturer` entity set, they occur in this relationship set with different *roles*.

One of the two lecturers is the *reviewer* and the other is the *review subject*.

In an ER diagram this is indicated by a single occurrence of the entity set, but connected multiple times to the relationship diamond with each line labelled with the relevant *role*.

Roles in an ER diagram



Aggregation

So far we have covered relationship sets that involve a number of *entity sets* and that may (or may not) have *relationship attributes*.

However in a number of circumstances we want to model a relationship between an *entity set* and a *relationship set*.

This can be achieved by using a feature of the ER model called *aggregation* which provides a way of indicating that an entire relationship set is participating in another relationship set.

Essentially it allows us to view a relationship set as a “special sort" of entity set.

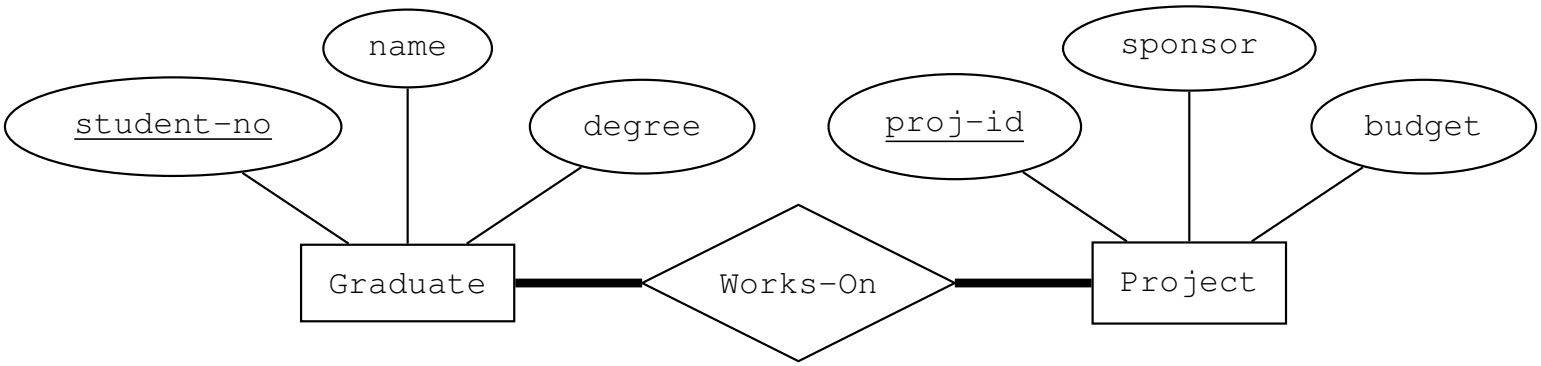
When is it used?

Suppose that we have additional requirements for research projects as follows.

- Graduate students have a student number, a name and a degree course
- Each research project has one or more graduate students working on it
- When graduate students work on a project they have one supervisor for their work *on that project*; they can work on more than one project in which case they may have different supervisors for each one.

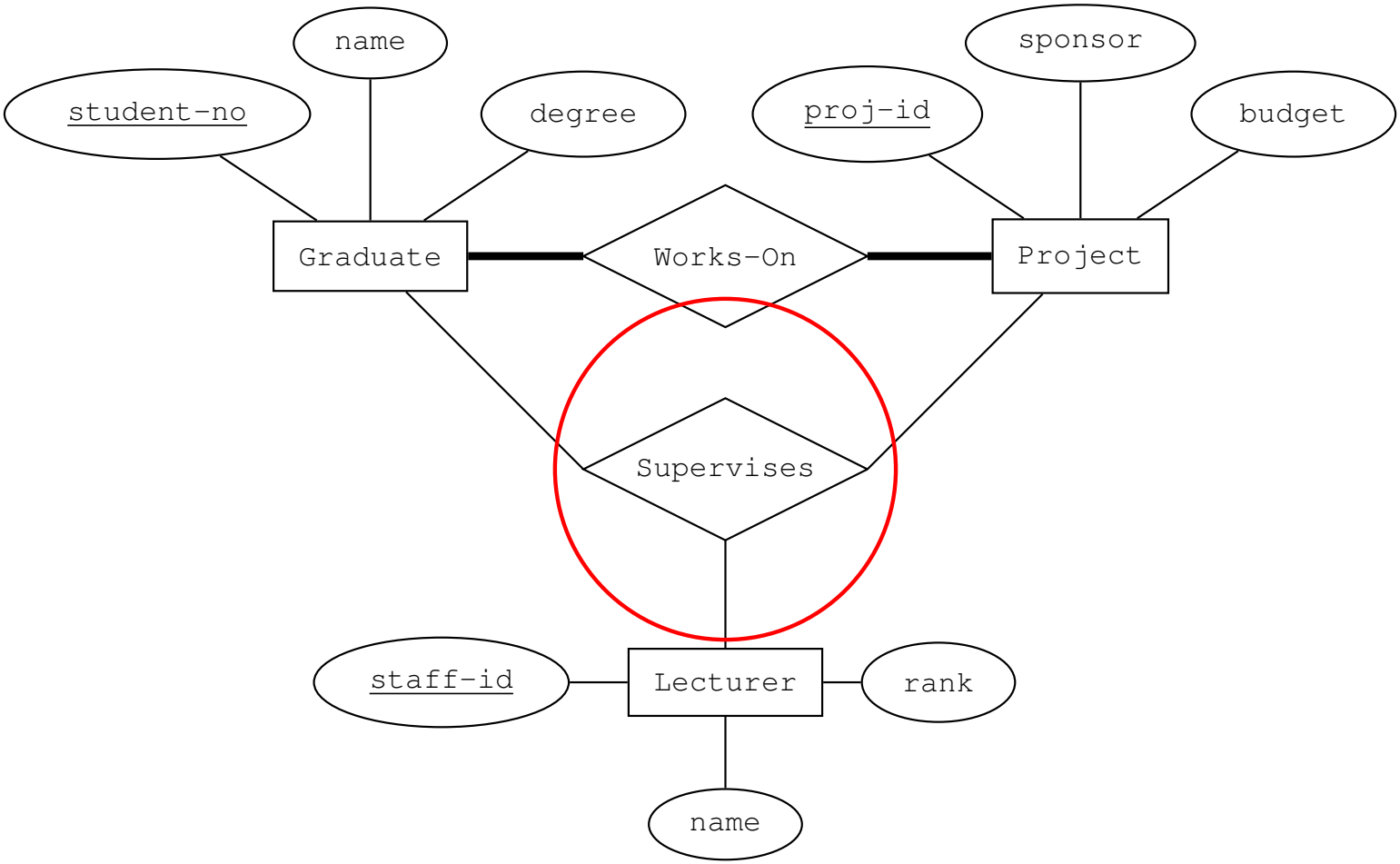
First requirement

Modelling the new entity set `Graduate` and the requirement that they each work on at least one project and every project has at least one research assistant is familiar and straightforward.



But how to add the *supervision* requirement?

First approach



A ternary relationship?

This introduces a *ternary relationship* called `Supervises` that relates a graduate student, a project and a lecturer.

The elements of this relationship set would be *triples* containing one graduate student, one project and a lecturer. So a triple of the form

$$(g, p, l)$$

could be interpreted as meaning

> Graduate g is supervised by lecturer l while working on project p.

This *does* capture some of the requirements — but what about participation and key constraints?

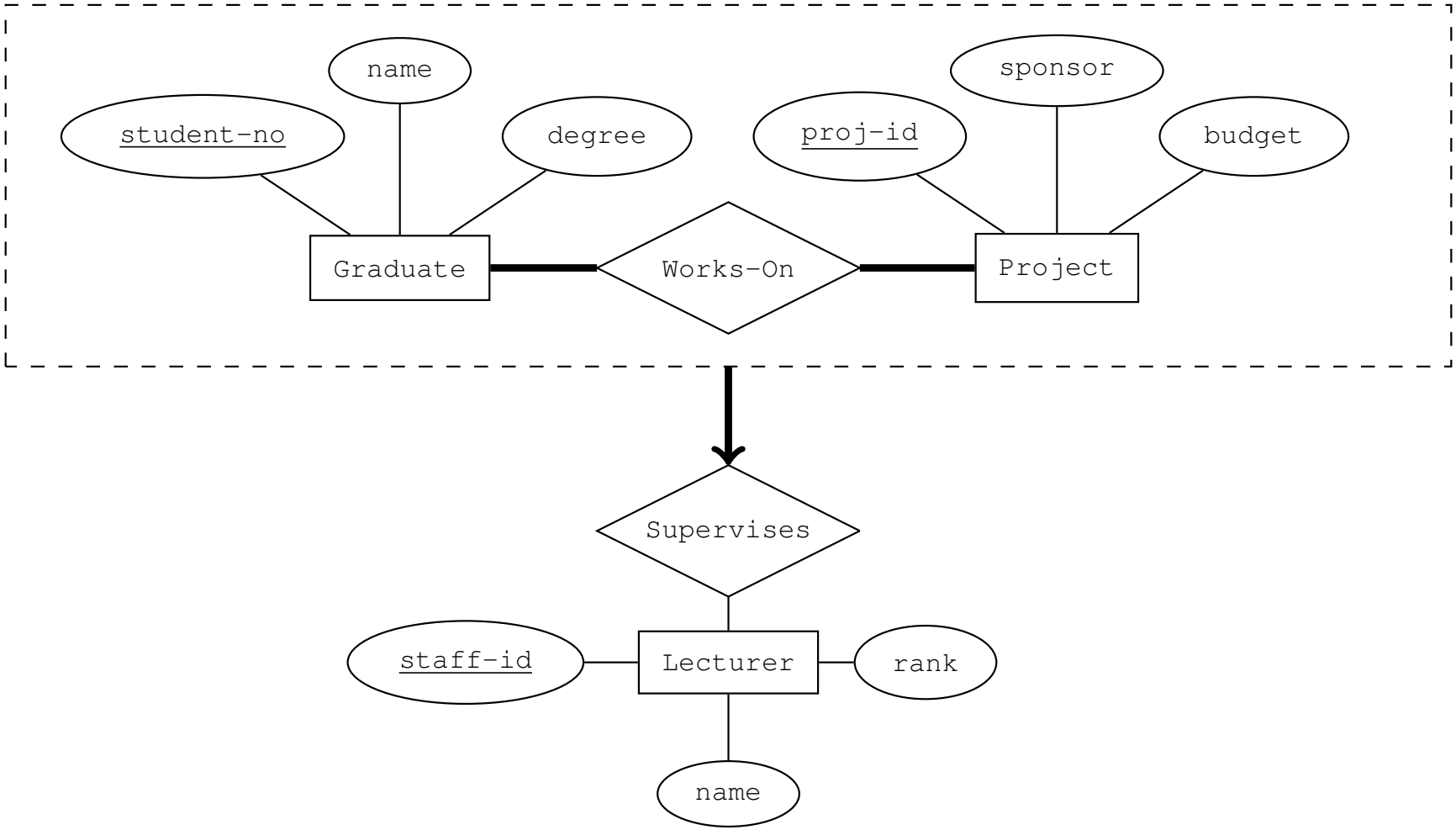
Key constraints

The problem is that we cannot correctly model the key constraints — that each “graduate student working on a project" should have a unique supervisor.

The problem is that we *really* want a *binary* relationship that has (g, p) as one of its components and l as the other.

This is exactly the concept of *aggregation* in that we treat the relationship set `Works-On` as an entity set in its own right.

Aggregation in ER diagrams



Boxes

This now models the exact concept that we wanted

- Every graduate working on a project has exactly one supervisor for *that* piece of work
- If a graduate works on more than one project, then they can have a potentially different supervisor for each project

The dashed box indicates that for the purposes of the `Supervises` relationship set, the contents of the dashed box are to be taken as a single entity set.

A balancing act

Even with these additions, there are many aspects of a normal business that an ER diagram *cannot capture*.

The reason that relational databases have been so very successful is because

- The ER model is *strong enough* to model a *useful proportion* of the activity of a business, yet
- The ER model is *simple enough* to be implemented and mathematically analysed

More expressive models are hard to implement and optimize, while less expressive models omit important functionality.