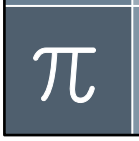
Structuring Data

Relational Modelling, Part III
Strong and Weak Entities



This is the third part of three in our discussion of relational modelling. Weak entities are the real challenge. If you master them, you master relational modelling.

Entity Relationship Design

- Steps to build a conceptual design
  1. Identify the entity types
  2. Identify and associate attributes with the entity types
  3. Identify the relationship types
  4. Determine cardinality and participation constraints
  5. Determine primary and foreign keys
  6. Validate the model

We continue our discussion about ER modelling, still focussing on relationships and – on finding good keys.

Primary Keys of Weak Entities

This is the tricky bit in relational modelling. If you master this, you'll produce useful databases.

Natural Key for Weak Entity "Purchase"

Purchase

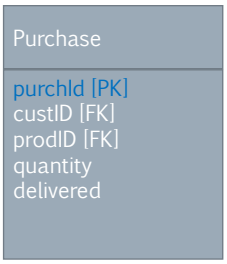
custId	prodId	quantity	delivered
1234	2345	5	05/02/2023
1235	2346	10	10/06/2023

In the last module, we cleaned up the primary keys for our strong entities Customer and Product. We discussed why these changes had to lead to adjustments of the Purchase table: Once the family_name and given_name columns were no longer composite primary key of the Customer table, they are no longer suitable columns for the Purchase table. They are the wrong cardinality, and lead to duplication of values.
As a result of the key change, we now have a natural composite key of custID, prodID and delivered. This is the natural composite key that has a chance of uniquely defining the details of each of our purchases.

Surrogate Key for Purchase

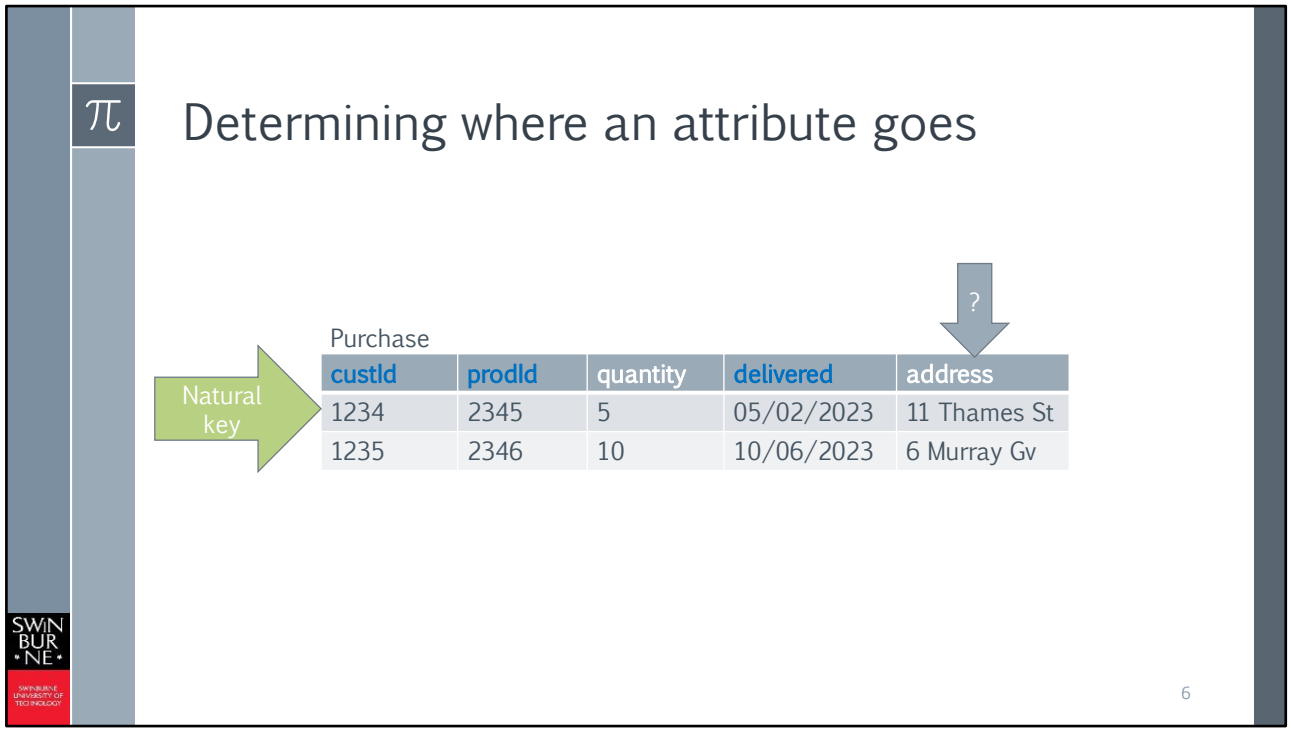
Purchase

purchId	custId	prodId	quantity	delivered
1111	1234	2345	5	05/02/2023
1112	1235	2346	10	10/06/2023



5

What people are usually more comfortable with, is using and ID field. Database products offer an autonumber datatype, which automatically allocates and then increments an integer.
In this way, it is easy to work with a surrogate key.
As you can see, this does not enable us to remove any attributes that belong to the natural key from the table, otherwise we lose information.
Another consideration is, if your natural key is not made primary key, the database will not stop it from repeating. So if you use the purchaseId as a key, you may end up having the same customer having the same product delivered twice on the same day. That may well be what you want. A good modeller knows the consequences of their decisions.



Let's use the natural key for the moment.
We are going to explore if the key can help us decide whether an attribute is in the right table.
We have to record the delivery address somewhere. Is the Purchase table the right place? We are already recording the delivery date there, so maybe it will work.

Let's suppose the customer has several warehouses and decides with every purchase where the deliveries go. The address is clearly purchase –specific. So Purchase would be the right table for it and the full key would determine the address value.

If the customer only has one warehouse, the address never changes for this customer. You realise that the address only depends on part of the key (custId). That will tell you that the address will repeat unnecessarily in this table (whenever the same customer buys something).

However, if you have just one customer who has a choice of warehouse, you have to accommodate a potential change of warehouses per purchase.

Surrogate or not

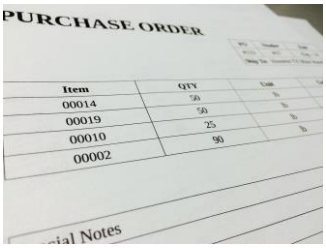
Purchase

purchId	custId	prodId	quantity	delivered	address
1111	1234	2345	5	05/02/2023	11 Thames St
1112	1235	2346	10	10/06/2023	6 Murray Gv

With a surrogate key, the question becomes, does the address depend on the primary key or a non-key column (the custId)?

This ignores the fact that custId is actually a defining column of the Purchase table, but that it can't define the entities in Purchase alone. So it becomes more difficult to examine what defines each column.

Do we need an even weaker entity?



Purchase

purchId	custId	prodId	quantity	purchDate	deliveredDate
1111	1234	2345	5	28/01/2023	05/02/2023
1111	1234	3333	20	28/01/2023	05/02/2023
1111	1234	4444	10	28/01/2023	05/02/2023
1112	1235	2346	10	30/05/2023	10/06/2023

8

First, let's go a bit more professional with our table and give it a purchase date. It is the norm for businesses to record when someone orders from them. If we want a natural composite key, the purchDate is a much better candidate than deliveredDate. (We can't be sure all goods are in stock and ready to be delivered on the given date.)

Let's make another change to Purchase. It is also common for businesses to include several products in the same purchase order. The purchase order looks like in the picture – one customer, one order date, but several line items. Our first instinct is to add more rows for the purchase into the purchase table. As you can see, there is now a mismatch in cardinality. We have repeated the purchId, custId and purchDate as well as the deliveredDate of purchase order 1111. The product Id and quantity, however, don't repeat.

From this you can see that there is a mismatch in cardinality. This means we have no choice but to introduce another one-to-many relationship. We retain the Purchase table, but we create an even weaker entity for the purchase items.

‘Weak and Weaker’ Entity

Purchase

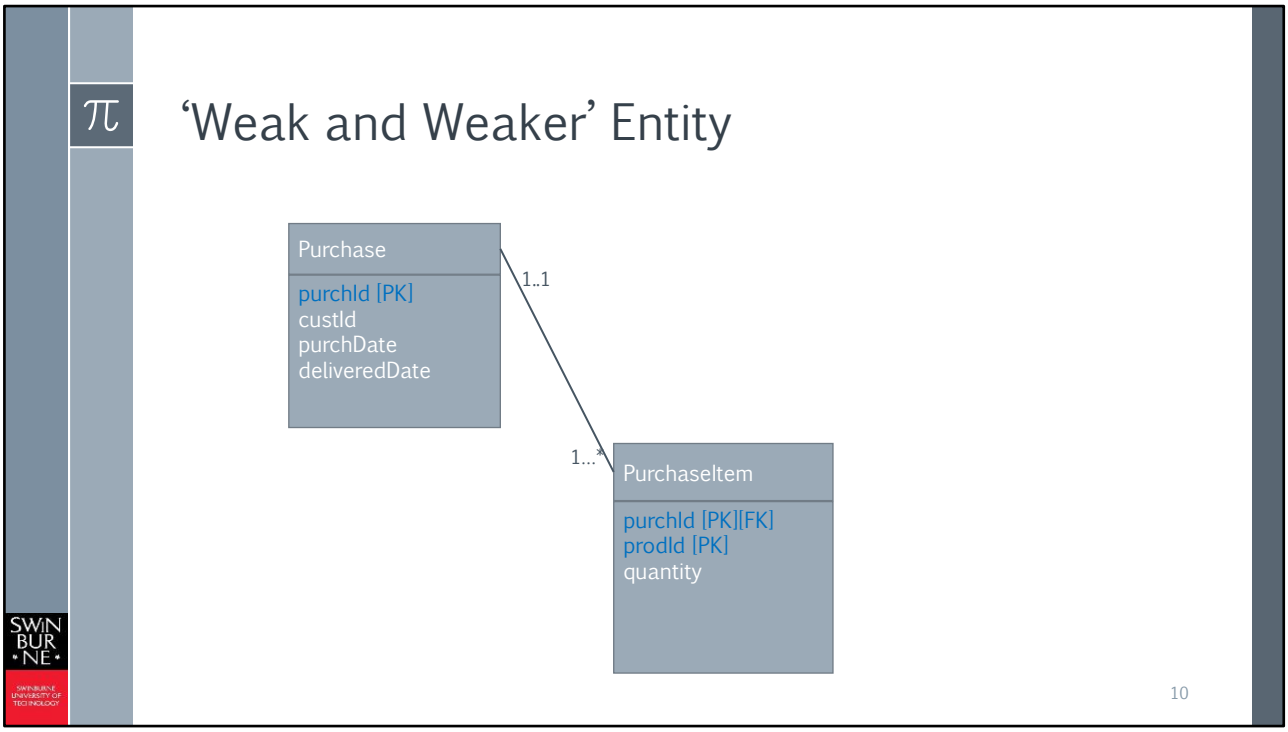
purchId	custId	purchDate	deliveredDate
1111	1234	28/01/2023	05/02/2023
1112	1234	30/05/2023	10/06/2023

PurchaseItem

purchId	prodId	quantity
1111	2345	5
1111	3333	20
1111	4444	10
1112	2346	10

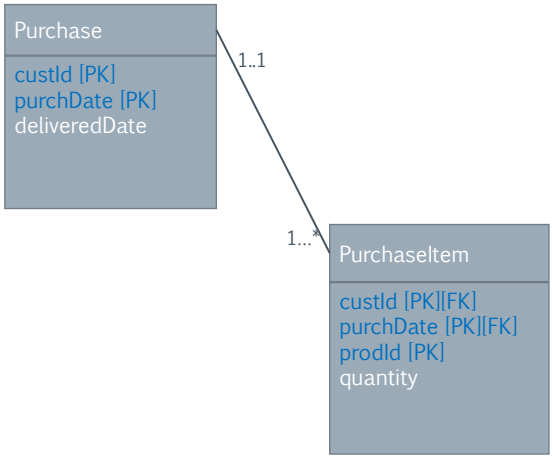
9

Now the tables are compliant with relational modelling rules. No unnecessary redundancy. Question: Which attributes are the primary keys for each table? Which attribute is the foreign key? Imagine what the UML diagram would look like before you continue.



In UML, our new Purchase / PurchaseItem relations look like this. For each entry in the Purchase table, we have at least one purchase item, but we can have many. For each entry in the PurchaseItem relation, we must have exactly one matching entry in the Purchase table.
PurchaseItem now needs an additional attribute in the composite primary key to identify the quantity attribute. And as the PurchaseItem table is the child table, it is the one that has the foreign key.

Weak Entities and Composite Keys

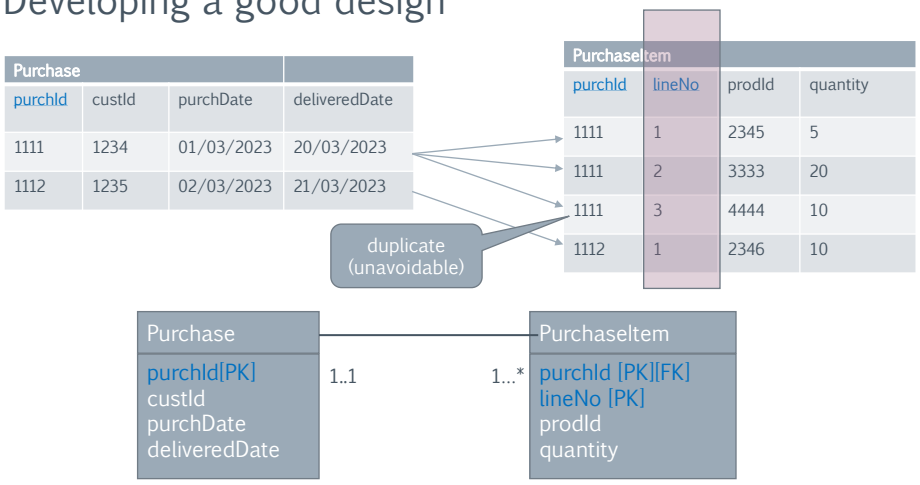


11

This is what the same relationship would look like if we did not have an ID for purchase (which is unlikely in the case of purchase orders). We have one fewer attribute in Purchase, but one more in PurchaseItem, to accommodate the extra field of the foreign key. It is worth mulling over this for a while – if you can explain to yourself why purchase date has to be part of the key in PurchaseItem, you are getting the hang of this.

A Better Key Option

› Developing a good design

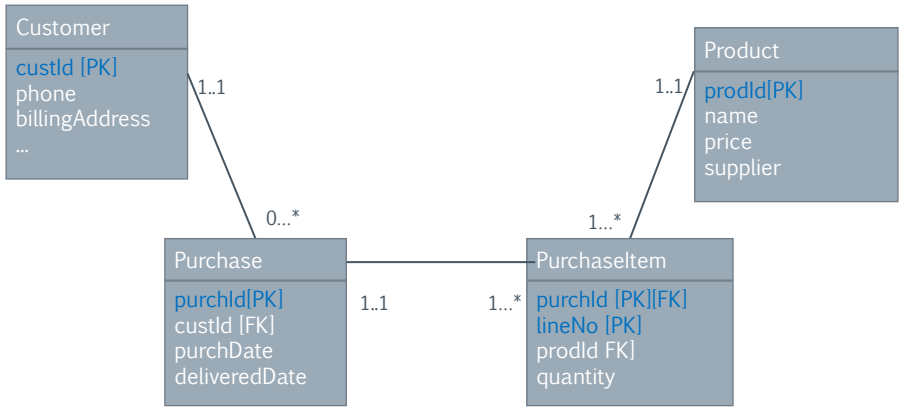


12

Would it help us if we introduced a surrogate key for the PurchaseItem table? I don't think so – because typically, this kind of table is the 'end of the line' – no other entity links to it. The surrogate key means nothing, so it just adds another column. What people tend to do is add an additional line number, which is unique within the purchId, but not across the whole table. The advantage of this is that we now have a natural ordering of the items of each purchase in case we need to print a purchase order on paper.
The other advantage is that the prodId is no longer part of the primary key. It may sometimes be necessary to replace a product for some reason. That would then change the key of the tuple. This is not really a problem, unless the same productid appears twice in the same Purchase. If we use line numbers as part of the key, this wouldn't be a problem either. Many issues to consider when modelling.

A More Complete Model

› So far, we have neglected parts of our database…



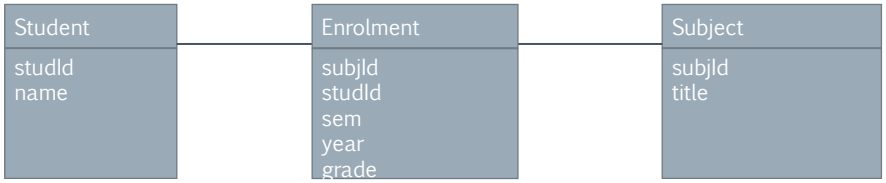
So far, we have omitted the strong entities to focus on the weak entities, which are always trickier to model. So here is our complete database so far.

Another Weak Entity – Test Yourself

Student	
studId	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment				
studId	subjId	sem	year	grade
101	MGMT101	1	2024	85HD
101	ICT402	2	2024	77D
103	ICT402	1	2023	60C
104	PHI787	2	2024	65C

Subject	
subjId	title
MGMT101	Management
ICT402	Info Tech
FIN394	Finance
PHI787	Philosophy



14

Create a database for your university where they can store your grades! You're in luck, the tables have already been created, and the attributes are in the correct entities! All you have to do is mark the primary and foreign keys in the UML diagram, then add the cardinalities next to the relationship. Use a piece of paper or notepad if you can't remember.
Don't add any more attributes for now.

Here is a hint: Sometimes students don't pass a subject, so you have to provide the opportunity to take a subject again.

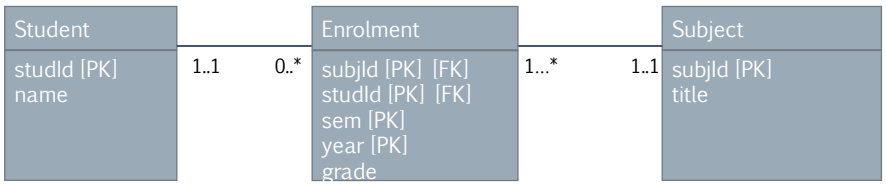
Now stop the recording and think about it before you continue.

Another Weak Entity – The Answer

Student	
studId	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment				
studId	subjId	sem	year	grade
101	MGMT101	1	2024	85HD
101	ICT402	2	2024	77D
103	ICT402	1	2023	60C
104	PHI787	2	2024	65C

Subject	
subjId	title
MGMT101	Management
ICT402	Info Tech
FIN394	Finance
PHI787	Philosophy



15

Is this what you expected? If not, and you cannot work out why, please discuss with the tutor, or ask questions on the discussion board, or in the online discussion session.

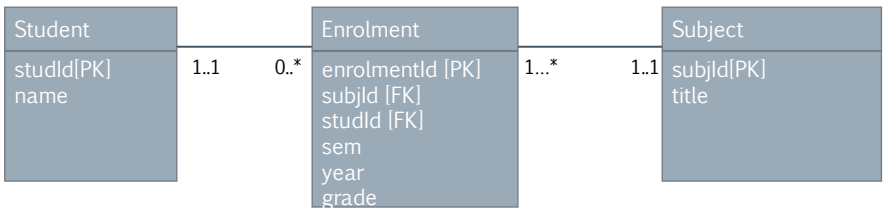
Most students don't like composite keys – what would the UML schema look like if we replaces the composite key with a surrogate key?

Another Weak Entity – With Surrogate keys

Student	
studId	name
101	Peter
102	Anh
103	Rajiv
104	Anna

Enrolment				
studId	subjId	sem	year	grade
101	MGMT101	1	2024	85HD
101	ICT402	2	2024	77D
103	ICT402	1	2023	60C
104	PHI787	2	2024	65C

Subject	
subjId	title
MGMT101	Management
ICT402	Info Tech
FIN394	Finance
PHI787	Philosophy



16

The enrolmentId is now the primary key of Enrolment. The consequence is that now we could theoretically enrol a student twice in the same subject in the exact same semester. So there is less protection from error.
There is no point in adding surrogate keys to the strong entities.

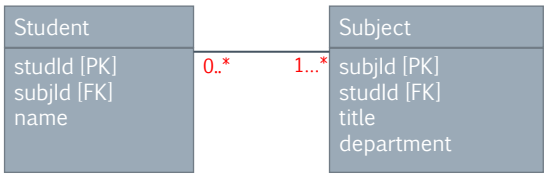
Many-to-Many

Many-to-Many relationships CANNOT be modelled in the relational world. Many students miss this basic fact. Let's look at why many-to-many doesn't work.

Why Many-to-Many Does Not Work

Student		
studId	name	subjId
101	Peter	MGMT101
101	Peter	ICT402
103	Rajiv	ICT402
104	Anna	PHI787

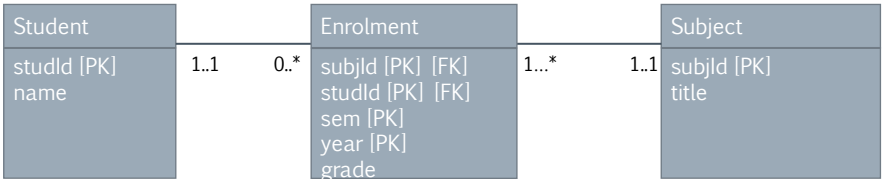
Subject			
subjId	title	studId	department
MGMT101	Management	101	Business
ICT402	Info Tech	101	IT
ICT402	Info Tech	103	IT
PHI787	Philosophy	104	Social Sciences



18

A many-to-many relationship between two tables is a situation where one row in the first table matches several rows in the second table and one row in the second table matches several rows in the first table.
Here, one student can be enrolled in several subjects, and one subject has several enrolled students. In a many-to-many model, neither table can have unique primary key values, because both have to repeat to match all the other table's rows. A student entry has to repeat as many times as they are (or have been) enrolled in a subject to accommodate different subjectIds. A subject entry has to repeat as many times as there are enrolled students to accommodate different studentIds. This leads to lots of duplication. There is also the problem that we can't be sure which entry the foreign key points to, as the primary keys are not unique.

Many-to-Many Resolved

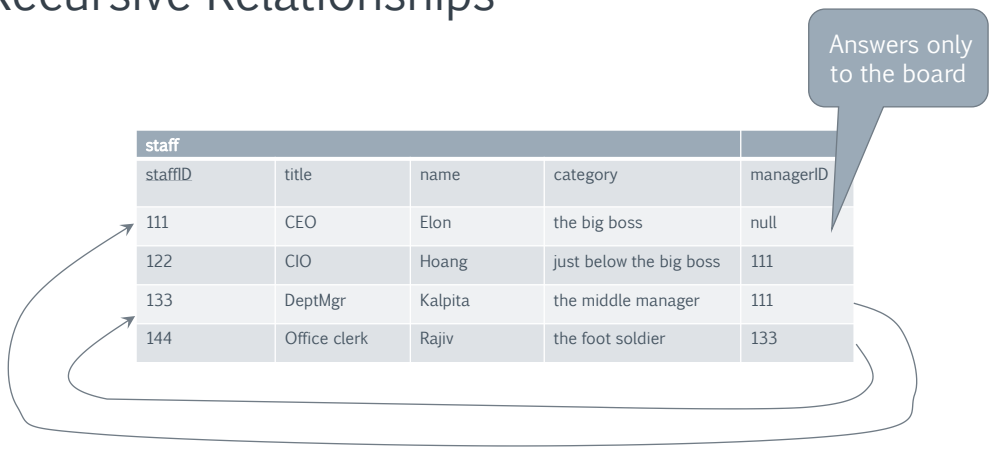


The way to resolve this is simply to put a weak entity in between. Due to the composite key in the weak entity, the primary key has unique values and there is no redundancy.

Recursive Relationships

A recursive relationship models a relationship between a table and itself. Sounds weird, but it can be useful.

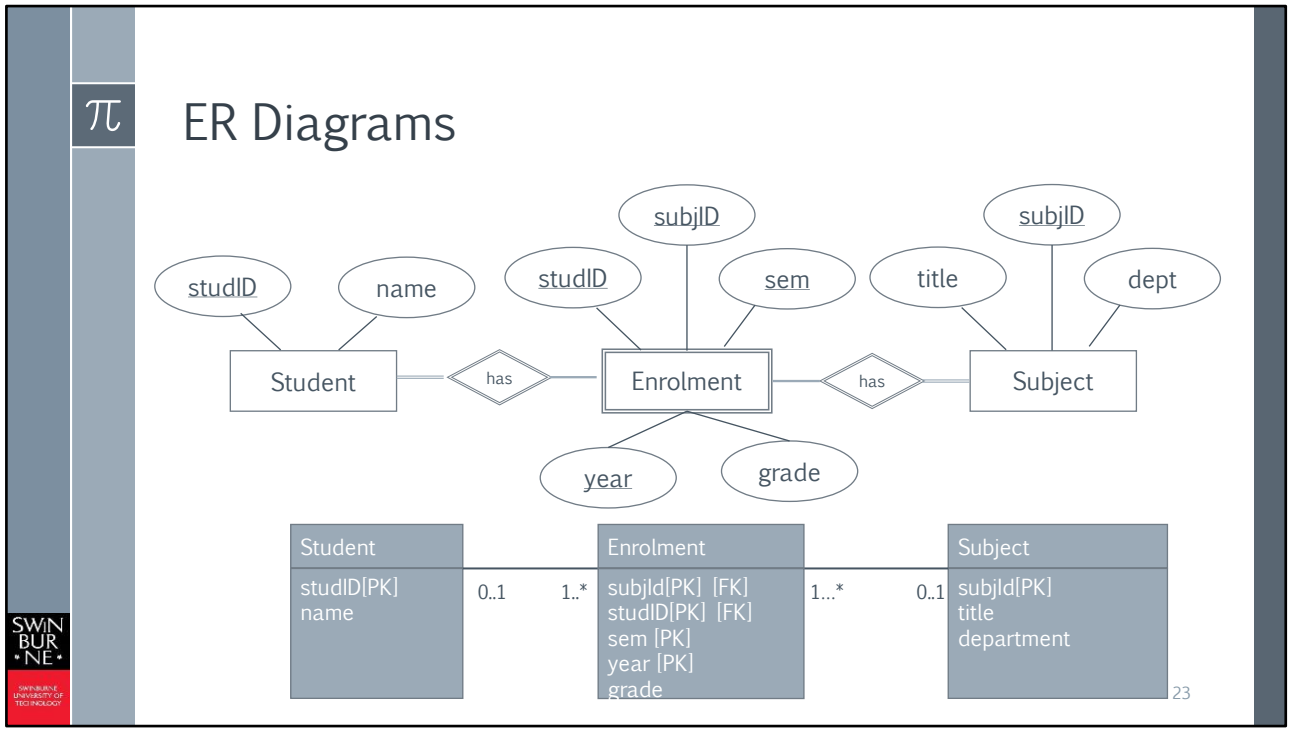
Recursive Relationships



Every staff has a manager, and the manager is naturally also staff. So we can have a foreign key of managerId in the same table, and it references the staffId which is the primary key.

Notation, again

And finally, let's look at another notation – ER diagrams. You don't have to be able to make one in this subject. It's enough if you recognise it when you see it.



ER diagrams take a lot of space, because every attribute goes in its own separate oval shape. If you have a large database (and people do), this gets a bit messy. Having said that, if you omit the attributes, you end up with a similar diagram as when you omit the attributes in an UML diagram ☺

To explain: The attributes are shown in oval shapes and connected to the entity, which is a square shape. The double line around enrolment means that it is a weak entity. The relationship between enrolment and student, the diamond shape, also has a double line because it is a relationship with a weak entity.
The double line that connects the diamond to the student entity shows that participation is mandatory. There cannot be enrolment tuples without a link to a student. The solid line between the has relationship and the enrolment means that there can be students without enrolments. The participation of enrolment is optional. We have the same situation between enrolment and subject – the subject's participation in the enrolment is mandatory, whereas the participation of enrolment in subject is not – we can have subjects that have never been offered, so there are no enrolments yet.

Summary

- Understanding the natural cardinality of an attribute makes for good design.
- The ability to identify a unique key makes for good entities.
- When there are several candidate keys, it's important to know what the potential limitations of each key are.
- Introducing surrogate keys often simplifies relationships, but not in every case.
- You can have recursive relationships.
- Many-to-many relationships **are not possible**.They are resolved using weak entities.
- ER diagrams are used less frequently than UML diagrams.