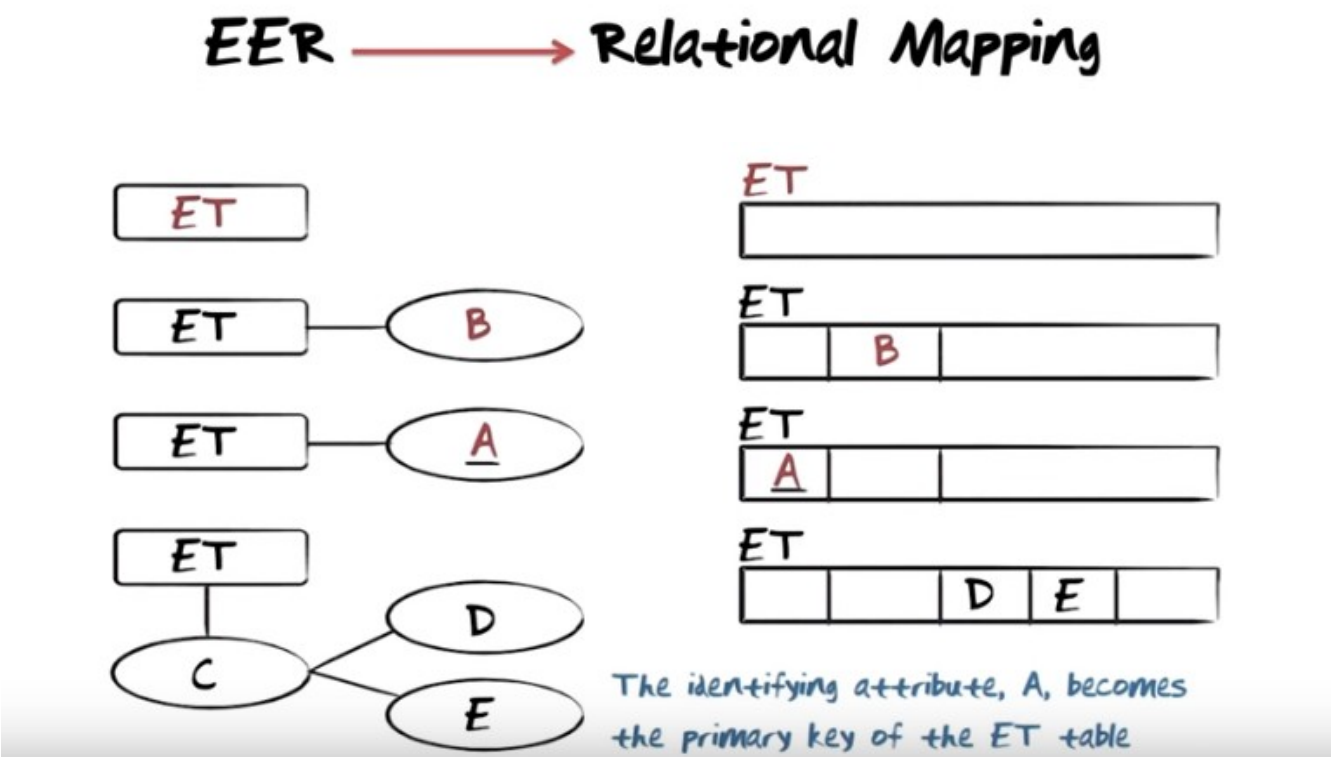
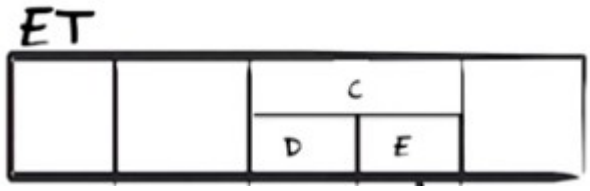
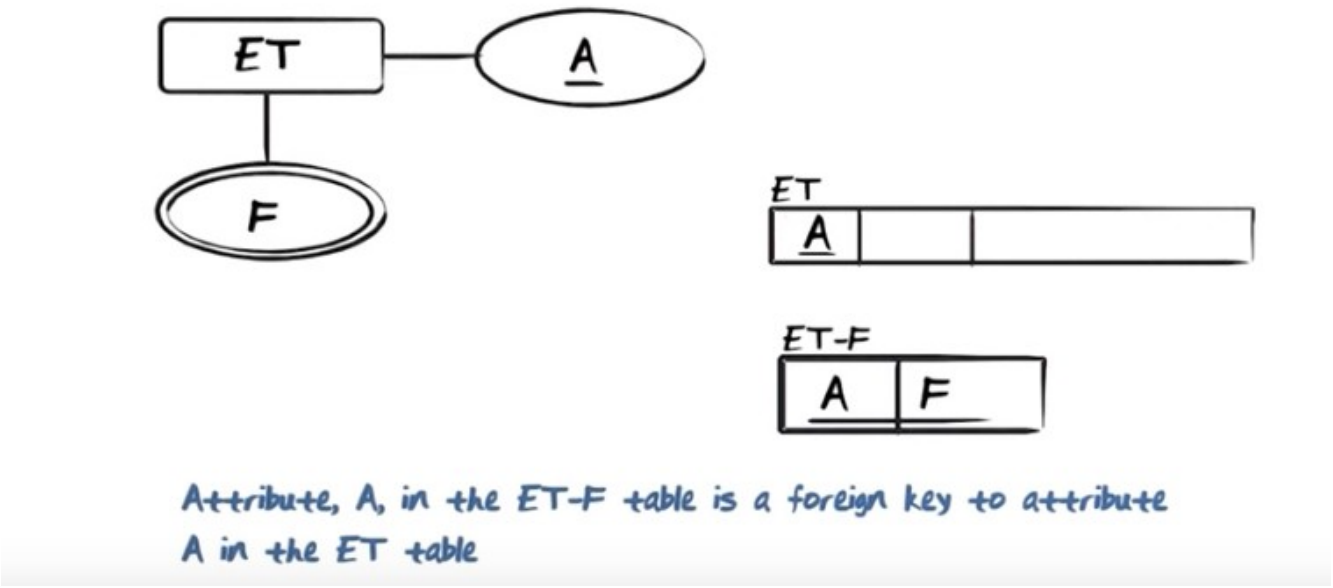


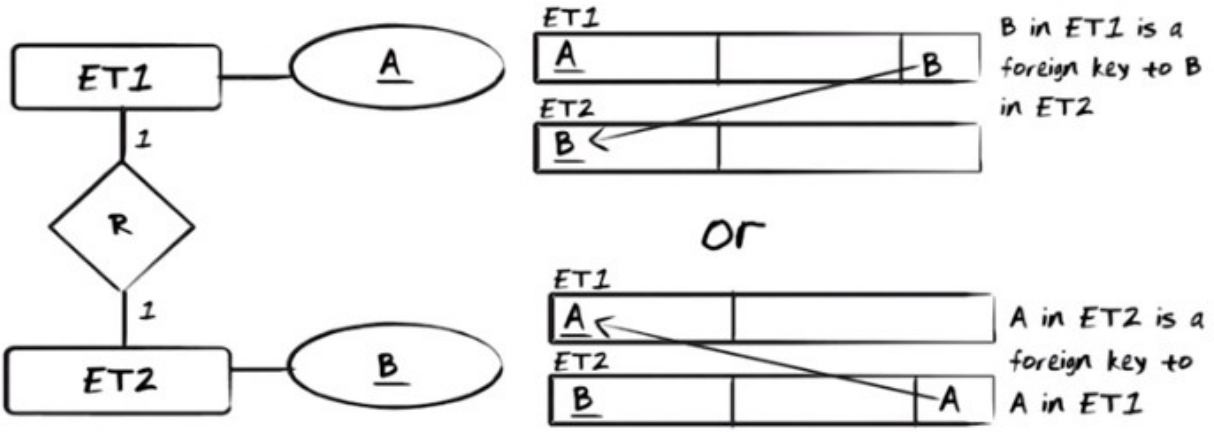

The above figure: Relation 的名字就是 entity 的名字, relation 中的 atribute(一列)就是一個 property. 對於 composite property type C (composed from property type D and E), 在 relation 中只能放入 D 和 E. You might ask what happens to property C. Unfortunately in the translation, property type C gets lost. That means that the meaning that's expressed by the relation will not be as rich as it is here. One could consider the following alternative way of mapping it:



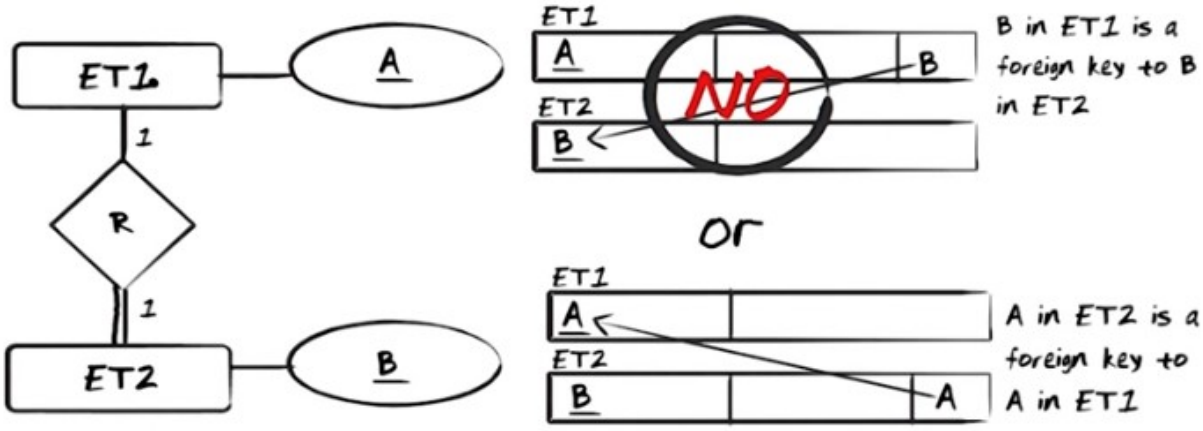

The above figure: However, we said that a relation was defined as a subset of tuples that were constructed from domeains with atomic values. Now this would not be be a domain with atomic values. This would actually be a domain where we would have a name (C) and two sub components of it. The realational model is inherently flat, and the languages we're gonna look at, relatinal algebra, calculus and SQL only work when that't the case. So unfortunately this is the solution that we have. We have lost C.



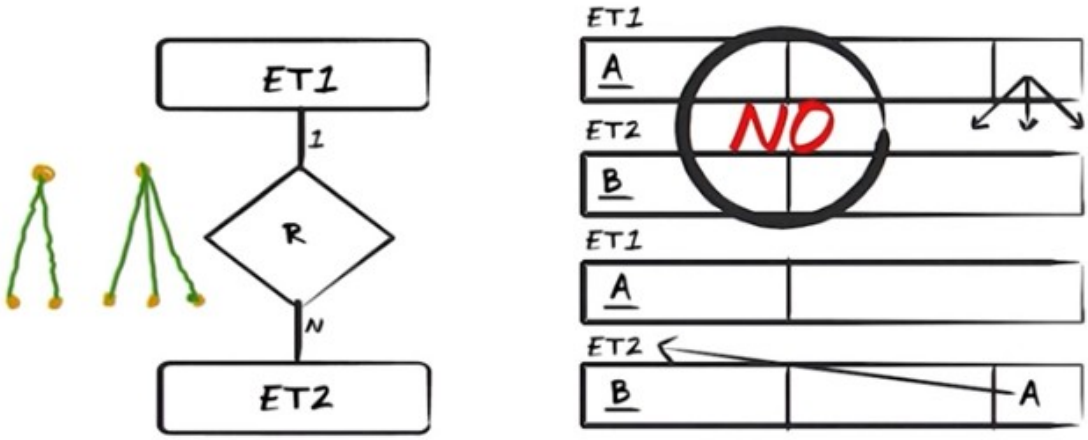

The above figure: 對於 multi-valued propery type, a brand-new relation is created. The name of that relation is the name of the entity ET hyphen the name of the particular multi-valued property type we are dealing with, in this case F. The combination of A and F become a composite key in ET-F. We don't have multi-values in the relational model, so we put it (F) in a seperate relation. 由後面的課中知, 也可以像下面那樣, 用一個箭頭 由 ET-F 中的 A 指向 ET 中的 A.



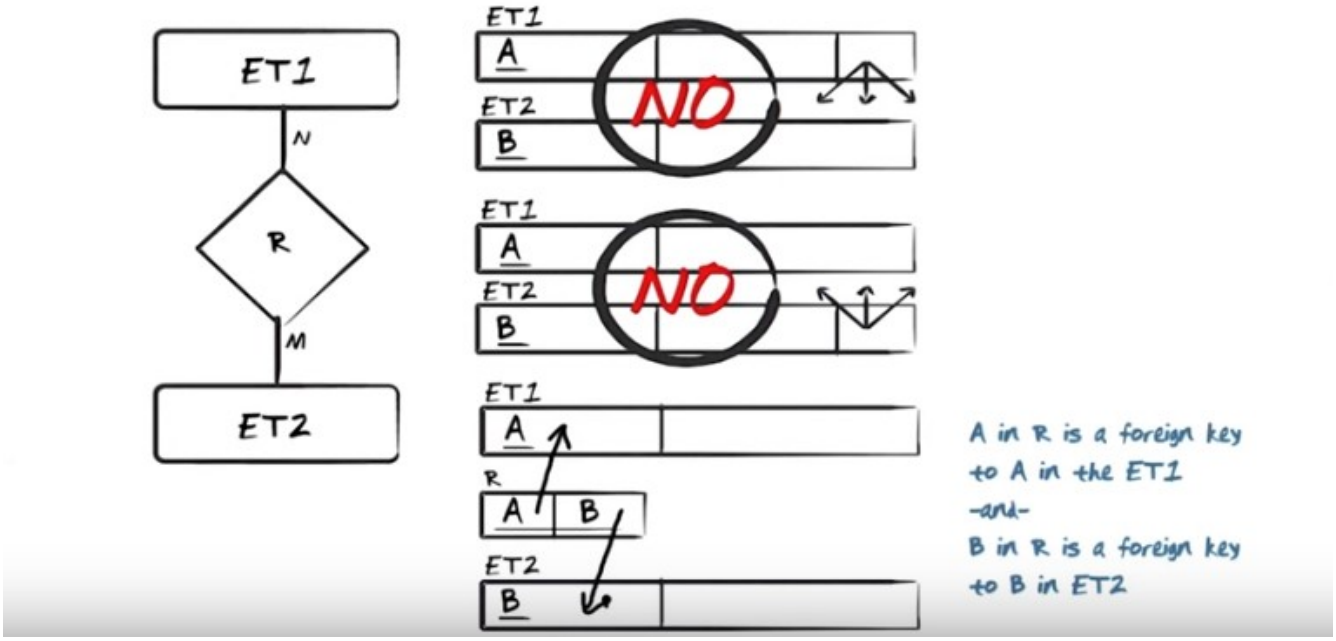

The above figure: Map one-to-one relationship types to the relational model. One way is that in ET1 we insert the primary key attribute of ET2, as sort of a pointer or reference from ET1 to ET2.



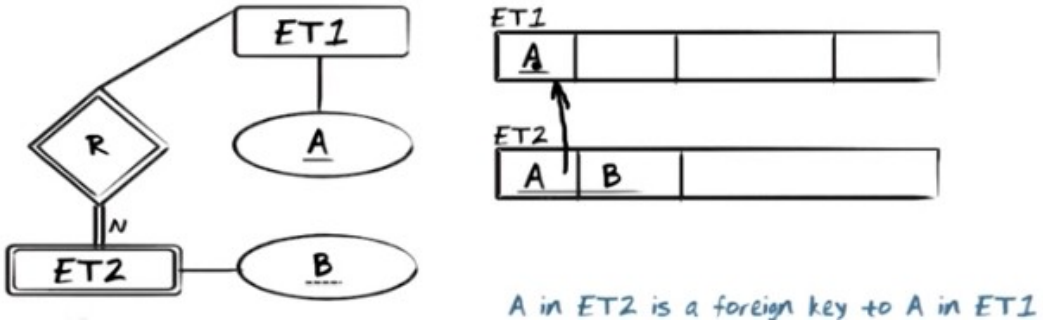

The above figure: ET2 是 mandatory 的. 第一種 mapping 方法不行, 因為 ET1 不是 mandatory 的, so there might be some ET1s that are not related. 上上圖中也可能不是 mandatory 的, 怎麼辦? MO: 沒辦法, ET1 和 ET2 都一樣, 所以就不講究了.



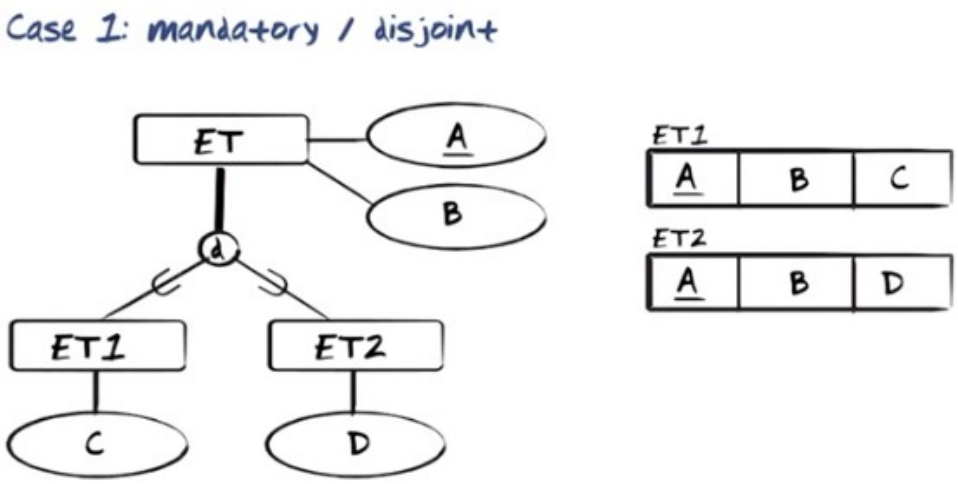

The above figure: Map one-to-many relationship types. 第一種 mapping 方法不行, 因為 remember again you cannot have a multi-valued in a field in a relation, that was exactly the reason we had to deal in such a strange way with multi-valued atrribute. So this solution here having mutliple pointers as a value of the atribute just does not work. 第一種 works. It's OK here to insert the ET1 side reference all to ET1 because it's unique for every single value of B.



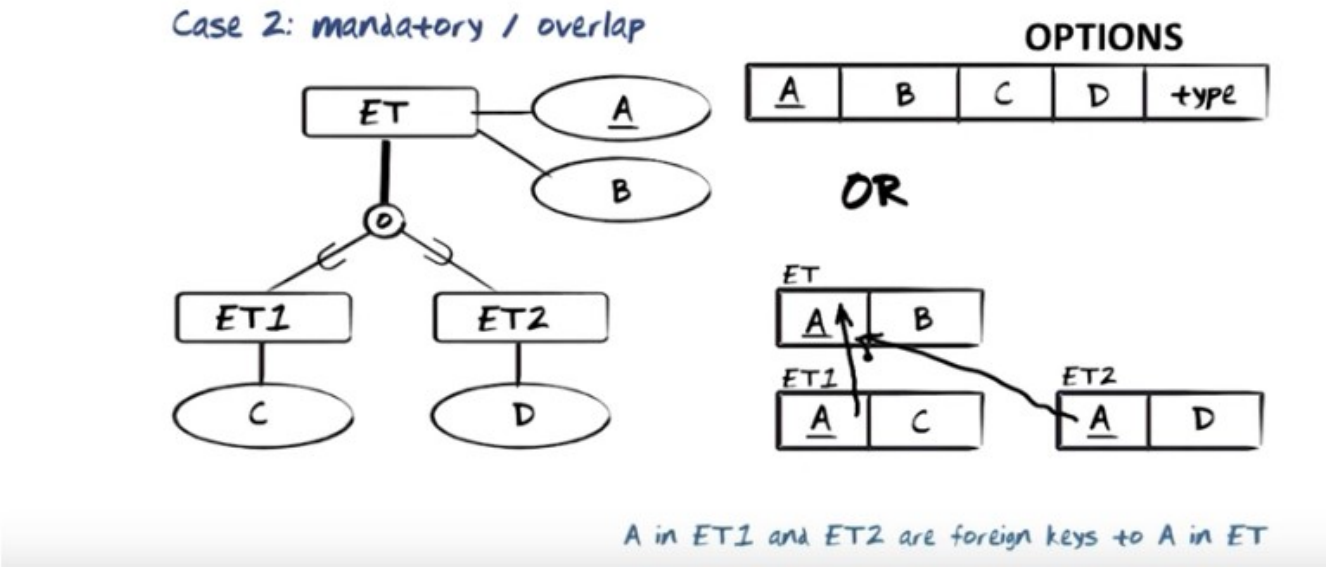

The above figure: many to many relationship types. We actually need to generate a separate relation(注意 relation 就是一個表) for R. So basically the idea is that from R we would point to the ET1 side, and the same for the ET2 side. It is important to notice that it is the combination of attribute A and B that constitute the key of relation R together. Why is that important? Well, it's important because that is exatcly what enforces a many-to-many relationship type. MO: 首先 R 本身已經是一個表了, 這個表也有個 primary key, 即 A 和 B 聯合起來的 key. 如何通過 A-B 來 identify R 的一行? 要給定俱體的 A 的值和 B 的值, 因為原 EER 圖中是 N:M relationship, 所以 A 的值和 B 的值都要給定, 然後就可以在整個 relational 表中定出哪個 ET1(A 為給定值的的)對應哪個 ET2(B 為給定值的的)了.



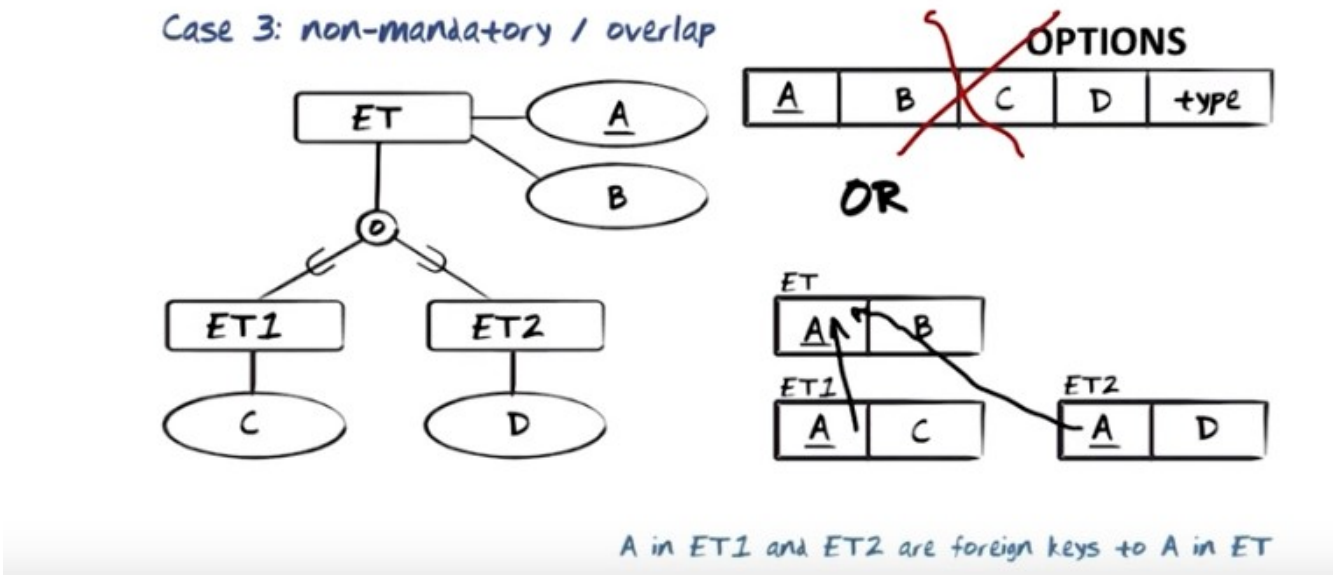

The above figure: map identifying relationship types R with weak entity type ET2 (雙線的 relation type 即 identifying relationship, textbook 252). Insert in ET2 a reference to ET1. That is, the A attribute in ET2 becomes a foreign key on the atrribute in ET1. Somewhat similar to what we saw in the many-to-many relationship type, the combination of A and B here constitute the key for ET2. The reason is as follows. When you look at an instance of ET1 identified by A, then through the one-to-many relationship type here, that will identify multiple values of ET2. Each one of those values has a value B. So it takes that A and B in combination to distinguish between the instances of ET2.
我的理解: foreign key 就是這樣一個東西: 它本來不是這個 relation 的, 而是從別的 relation 中引入的



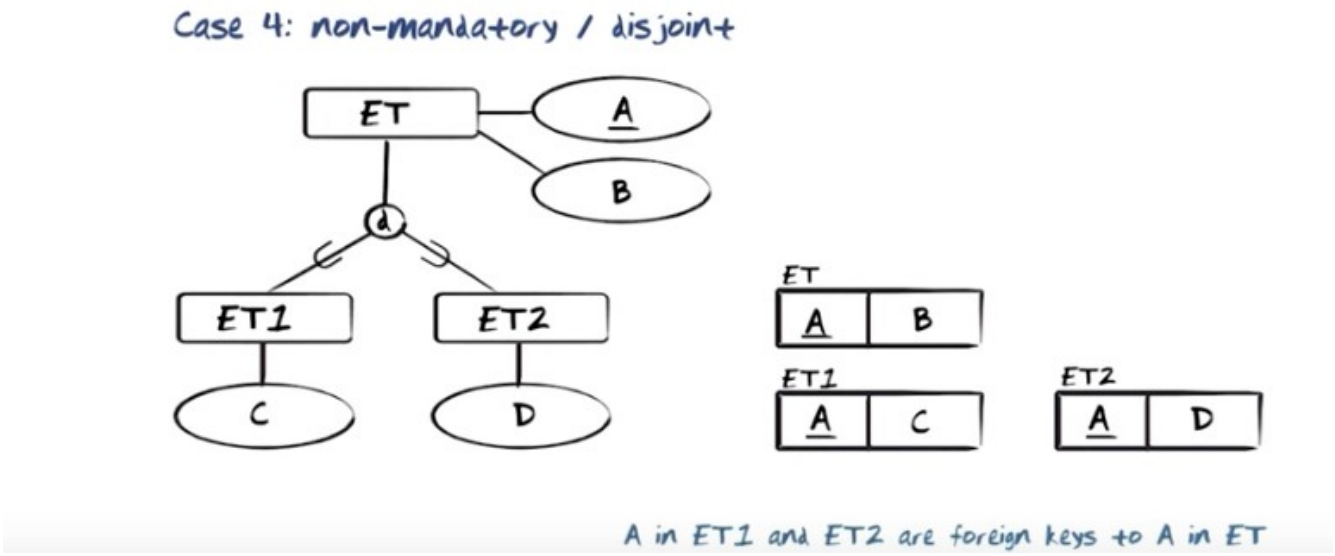

The above figure: mapping super-sub type relationships is a little bit more complicated. There are four cases I would liket to consider. In the first case, mandtatory and disjoint. We only create a relation for ET1 and for ET2, we do not also create relation for ET. We will find every single ET instance with a corresponding A and B value either in ET1 or ET2.



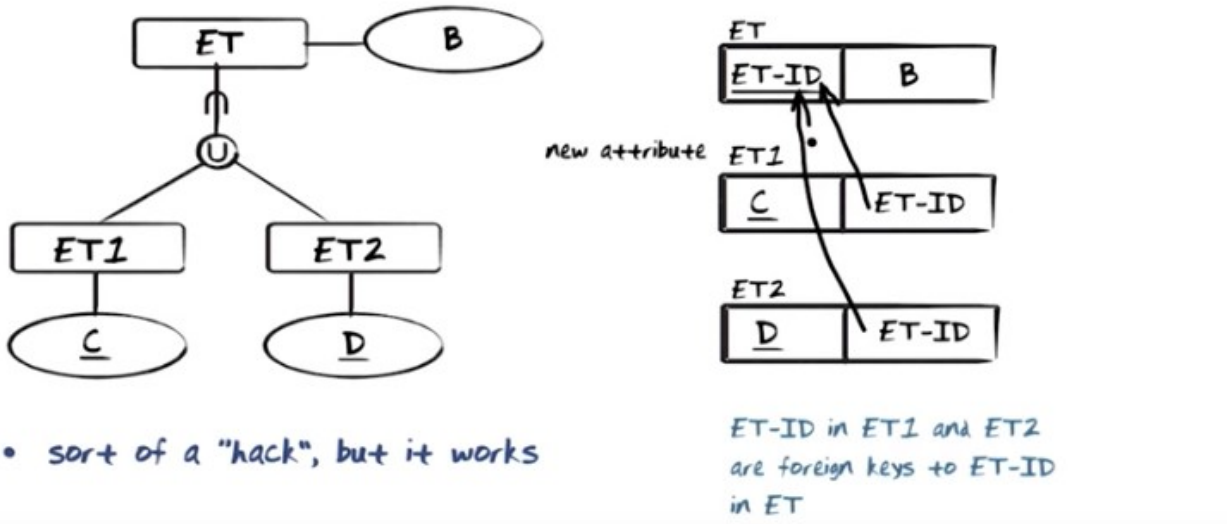

The above figure: madatory and overlap. 第一種的 type 的作用為: type will say whether this is an instance that is in ET1, or whether it's an instance in ET2, or whether it's an instance of both ET1 and ET2. There are several reasons that I do not likt this option. One is that when ther are instances of ET that is either in ET1 or in ET2, on of those (C and D) automatically will be a null value. The second reason is.... 第二種: every sinlge tuple in the ET relation will either have a tuple in ET1 with matching A value, or a tuple in ET2 with matching A value, or a tuple in both with matching A value.




The above figue: non-mandatory and overlap. 第一種畫叉不是說這種方法不行, 而是說不喜歡這種方法, 理由跟上圖的一樣: now you may actually have tuples in this single relation (ET relation) that has B values but no C or D values, because that's not mandatory anymore. 第二種 is very similar to the one we used in case two. It is now possible to have instances of ET with A and B values, where the corresponding A value does not exist in ET1 or ET2, simply because that's not mandatory.




The above figue: non-madatory and disjoint. Since this is not mandatory, you can have instances of ET that have no corresponding tuples in ET1 and ET2. However, when there are tuples in ET1 or ET2 because of the disjointness, a tuple will only be in one of them.




The above figure: we insert an artificial identifier in the relation ET. That identifier will then consist either of what corresponds to C or corresponds to D. Remember that every single element in there (左圖中的 ET) comes from there (ET1) or from there (ET2), so it will be identified by C or by D. So the ET identifier is a foreign key on the ET-ID in ET both from ET1 and from ET2.