Translating ER into RM

The best way to learn this mechanism is to poceed by examples.
Let's start with this classic:

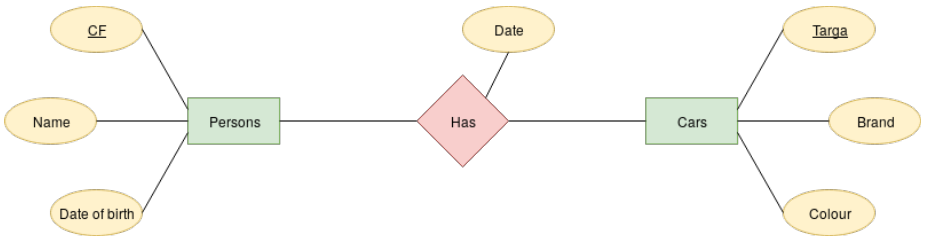

Figure 1: The classic example

When we want to translate an Entity Relationship Diagram (ERD) into a Relation Model (RM) we have to follow certain rules, let's check them out.

Rule #1 Every entity is a table and its attributes are the table's columns.

Rule #2 Each relation is a table and its attributes are part of the table's columns; also the keys of the entities involved in the relation become columns and they are flagged as Foreign Keys (FK). The FKs must be the table Key.

Let's put this in practice:

PERSONS

<u>CF</u>	Name	Birth date
1	John	1978
5	Mary	1974

`PERSONS(CF, Name, Birth date)`

CARS

<u>Targa</u>	Brand	Color
xy	VW	Red
uv	Toyota	Blue

`CARS(Targa, Brand, Color)`

That was pretty easy, wasn't it?
Now it's time to draw the relation table:

HAS

<u>CF</u>	<u>Targa</u>	Date
1	xy	2002
5	uv	2004

`HAS(CF, Targa, Date)`

Well, that's clear. Now it's time to mix things up a little bit. What if a car can't be owned by different people? (see 2)

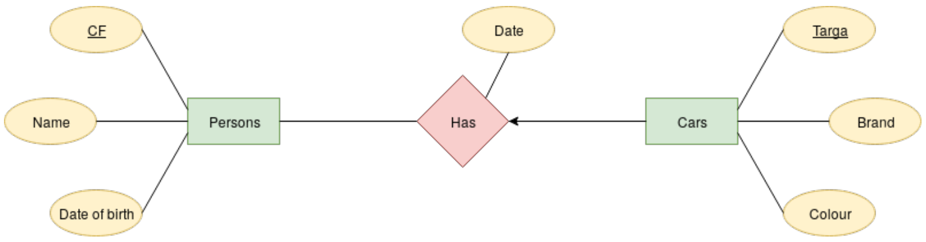

Figure 2: A car now can have at most one owner

We must use only the Targa as key in HAS relation.
And what if we want the car to have also at least one owner? (see 3)

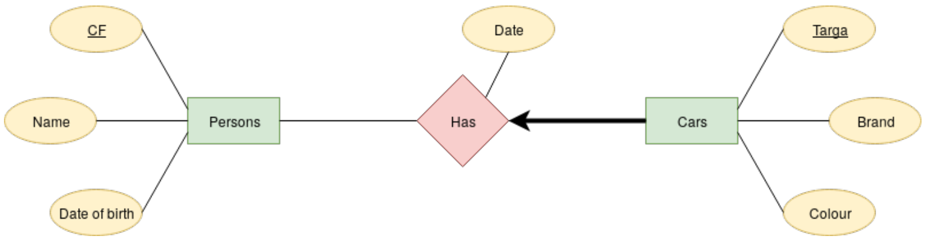

Figure 3: A car now can have at most one owner

Now that a car must be owned by one (and only one) person it makes sense no more to keep HAS table and CARS table divided: the smartest solution is to delete HAS table and introduce in CARS table two columns, one for Date and one for CF, which will be a FK. Last but not least the CF field must be mapped as *not-null*. That's the representation in RM:

CARS

Targa	Brand	Color	CF	Date
xy	VW	Red	1	2002
uv	Toyota	Blue	5	2004

```
CARS(Targa, Brand, Color, CF, Date)
CF not null
CF Foreign Key to PERSONS
```

Going back to diagram 1, what if I want to keep in memory all the times somenoe rent the same car? Now I cannot do that because, being the Key,Targa cannot apear twice in the table. The solution here is to use also Date as a Key. But that's not fasiable because, as we stated in Rule #2, only Keys of the entities are used as Key in the relation Table, and Date is not an entity; hence we have to redesign our diagram as

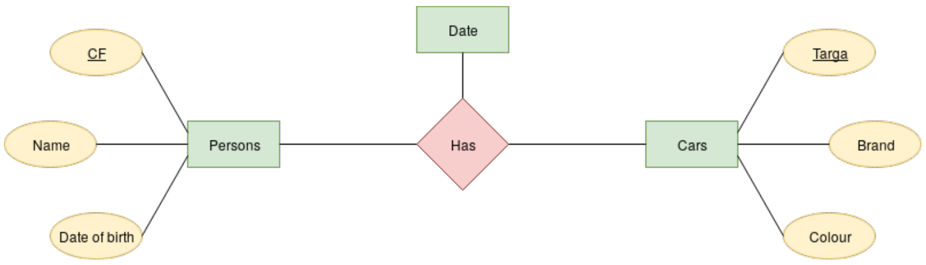

Figure 4: Now Date is an entity

Now we want that every Rent must have an Insurance, so we have to create a new entity called Insurance and link it to the Rent relation as this:

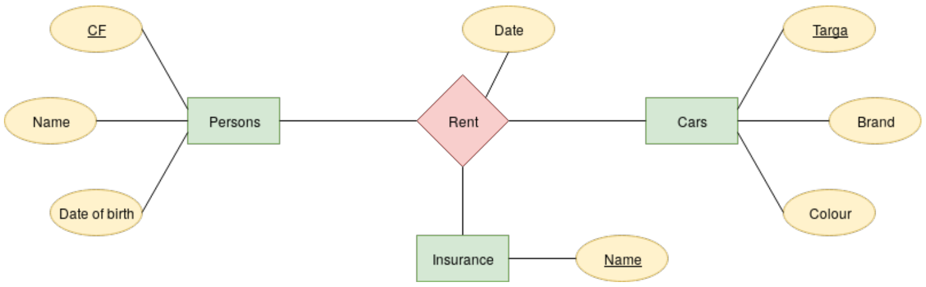

Figure 5: We added Insurance as an entity

```
   RENT(Owner, Car, Name, date)
Owner FK to PERSONS
```

```
Car FK to CARS
Name FK to INSURANCE
```

And now we decide that the Insurance is no more necessary for a Rent, how do we set it as a field hat can be null? We have to link it to Rent with a relation, but we cant link two relations toghether: we must aggregate Persons, Rent and Car so that we can treat this set as an entity and then we can link to this new entity the Insurance with another relation (6).

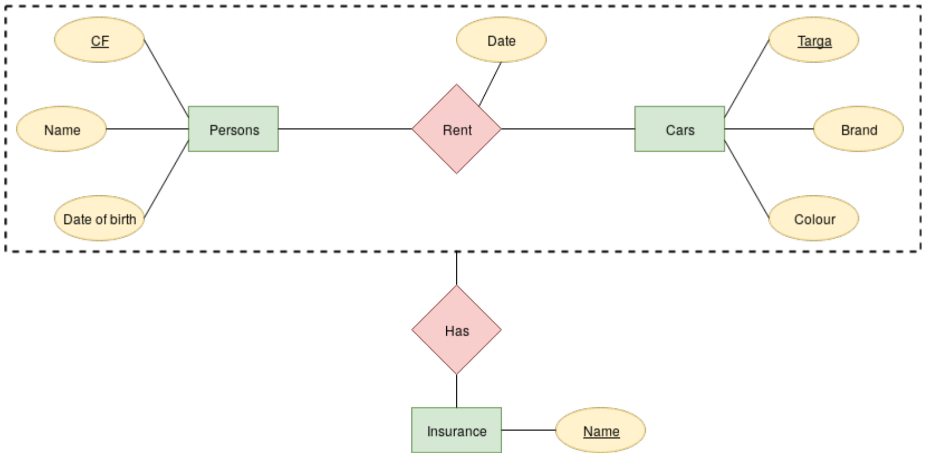

Figure 6: We added Insurance as an entity

In this case what's going to be the key of Rent?

```
   RENT(Owner, Car, date)
Owner FK to PERSONS
Car FK to CARS
```

And which should be the Key of has? Keeping in mind Rule #2 we deduct that the keys of a relation must be the keys of the entities involved,marked as Foreign Key, therefore:

```
   HAS(Owner, Car, Name)
(Owner, Car) FK to RENT
Name FK to INSURANCE
```

And now the last question for this example: what's the difference between 5 and 7?

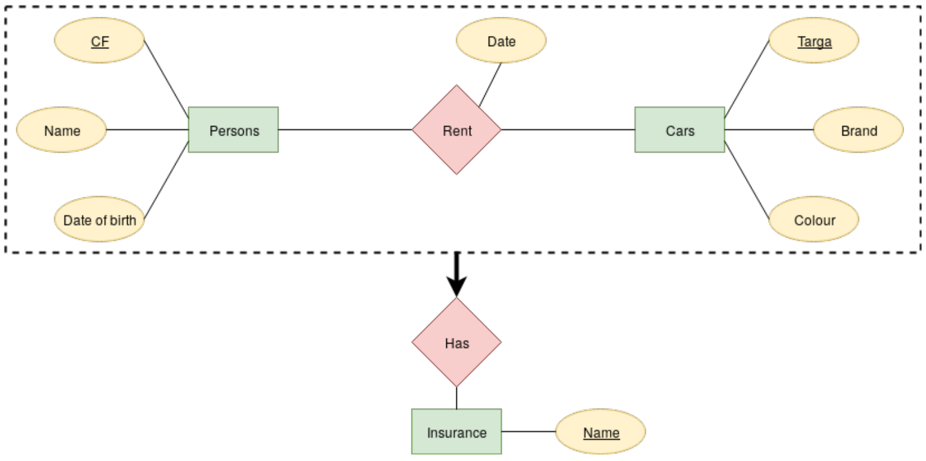

Figure 7: Note the particular form of the diagram

The difference is:

- In 5 you *must* use Name as Key in RENT table, because Insurance is an entity involved in Rent relation (Rule #2); therefore a person can rent the same car multiple times given that he uses a different insurance.

- In 7 you don't have to use Name as Key in RENT table, therefore it is not possible for a person to rent the same car with two different insurances (even if the insurance changes the key remains the same).

0.1 Let's talk about inheritance

It's time to see a little bit of inheritance, as always we're starting with an example:

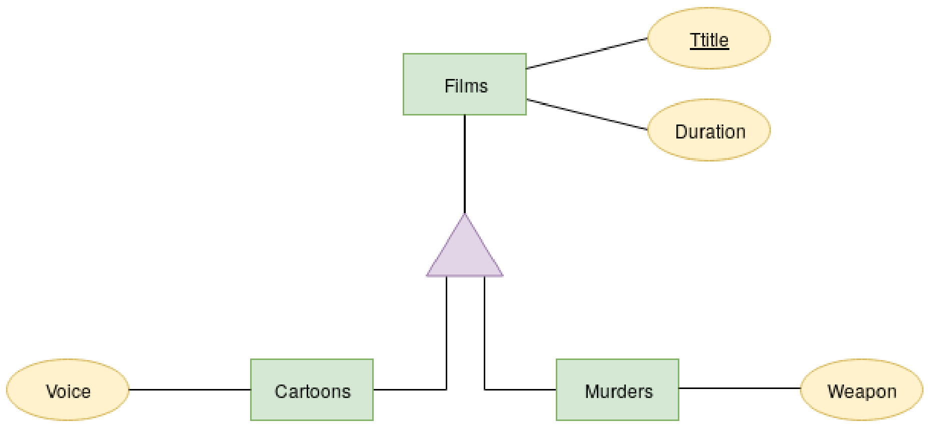

Figure 8: An example of inheritance

There are three ways to translate ER inheritance into RM:

1. The first approach is to have a different table for every entity, the tables of the children-entities will contain also the attributes inherited by the parent-entity.

   ```
   FILMS(Title, Duration)
   CARTOONS(Title, Duration, Voice) Title FK to FILMS
   MURDER(Title, Duration, Weapon) Title FK to FILMS
   ```

   The bad aspect of this approach is that we store the different entity-types in different palces, for instance "Shrek" iss stored into CARTOONS table, therefore if we want to query the database by title and ask for "Shrek" we have to navigate trough all the three tables. Also if we want to query on how many films the database contains we will have to shearch in all the three tables.

2. The second approach is to have a different table for every entity, but the tables of the children-entities contain only their specific attributes and the Key attribute of the parent-entity, which will be also the child-entity Key (Foreign Key).

   ```
   FILMS(Title, Duration)
   CARTOONS(Title, Voice) Title FK to FILMS
   MURDER(Title, Weapon) Title FK to FILMS
   ```

   This approach is better because all the entities are stored in the FILMS table and therefore if we're querying, for instance, title or duration, we have to look only in the FILMS table. The films that are cartoons, like Shrek are going to be registered also in the CARTOONS table, and there we will find their voice attribute. The side effect of this approach is that when we query by, let's say, duration and voice we have to look into two different tables, and that's actually more complex than we need.

3. The third approach is to store all the films iin the same table with all the attributes of the parent-entity and the children-entities, these will be filled with a *null* value if the film stored won't be a cartoon or a murder.

   ```
   FILMS(Title, Duration, Voice, Weapon)
   Duration is not-null
   Voice and Weapon can be null
   ```

   Thinking in a querying way that's the best approach, for every research in the database you have to navigate only trough a table.

0.2 Reverse engineering: from RM to ER

Sometimes you may want to be able to translate an RM back to its ER model, so here we provide some examples and basic rules in order to get through it.

For instance, let's consider the following statements:

```
A(x,y)
B(u, v)
  v FOREIGN KEY to A
  u FOREIGN KEY to C
C(m, n)
```

B has two foreign keys, one points to A and the other one points to C, so B is a relationship between A and C.

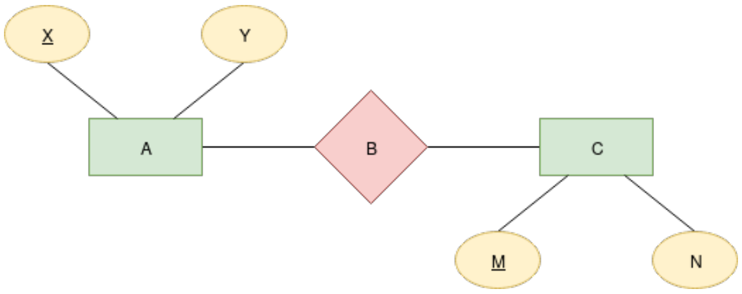

Figure 9:

But now we change it up just a little, and we want something like this:

```
A(x,y)
B(u, v)
  v FOREIGN KEY to A
  u FOREIGN KEY to C
C(m, n)
```

It looks similar, but it's not the same: `v` is not a key for `B` anymore, so we can have multiple `v`s for just one `u`, so the relationship from `C` to `A` becomes a "many-one" relationship.

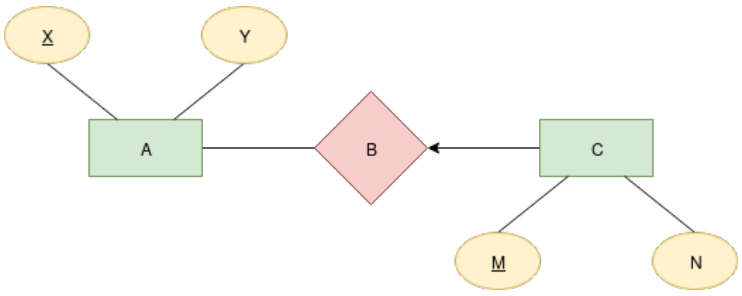

Figure 10:

Here it is another example:

```
C(m, n, v)
  m FOREIGN KEY to A
A(x,y)
```

This is a IS-A and C is child to A, because the foreign key `m` is also the only key in the table.

0.3 Weak Entities and summing it up

Now let's take a quick look at the translation of Weak Entities, i.e. entities that exist only if another one does.

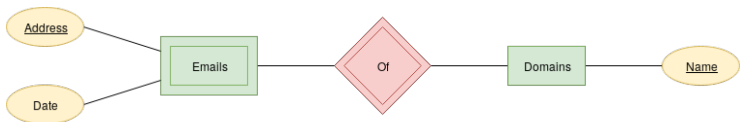

Figure 11: E-mail cannot exist if domain doesn't

The translation is simple: Domains is a straight entity, so we list all its attributes and underline the key, and then we do the same for E-mails, and its composite key is going to be its key (address) plus the key of its straight entity (name), which is also a Foreign Key.

```
DOMAINS(name)
EMAILS(address, date, name)
  name FOREIGN KEY to DOMAINS
```

Now let's point out three basic rule if you are into troubles with reverse engineering and you want to have it done quickly:

Rule #1 If the Foreign Key is not an attribute, what we have is a *relationship*;

Rule #2 if the Foreign Key is an attribute and it is the only key for that table, what we have is an IS-A;

Rule #3 if the Foreign Key is an attribute but it is NOT the only key for that table, what we have is a Weak Entity relationship.