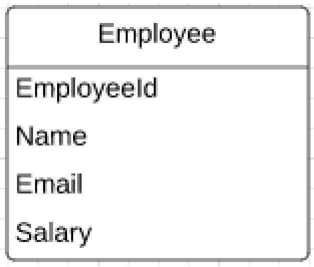
ERD Howto

1Entities

So here we are trying to make sense of this ERD modelling thing. Let's see how it works. So now, we are going to model the structure of a company. This is an application that almost every company has.

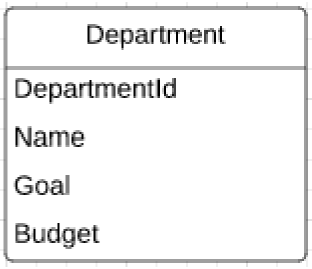
So let's start identifying concepts, it is obvious that a company has Employees. So we define an Entity called **Employee**. There you can keep whatever information about your employees, their name, their birthdate, their email address, their Id (passport or companyId number) etc etc. As usual and by default we will assign him our own identifier or primary key called EmployeeId.

So as we saw, this is how it looks like in an ERD.



Employees			
EmployeeId	*Name*	*Email*	*Superpower*
1	P. Xavier	xavi@xmen.com	Brain
2	Logan	wolverine@xmen.com	Nails
3	Rogue	rogue@xmen.com	Drains your life
4	Storm	Halle.berry@xmen.com	Weather forecast

Another entity that easily comes to your mind is a **Department**. As usual, the first thing we do with an entity is to assign it an id (Because I am not especially original I will call it DepartmentId)



Department			
DepartmenId	*Name*	*Goal*	*Budget*
1	Directorate	Run the school	1000000
2	Dep. A	Learn to control my powers	2000000
3	Operations	Save the world	123456

So far so good. Identifying entities should be your first step. Try also to think of the attributes the table has. At the beginning it will be a bit more difficult, sometimes you will mix two entities, sometimes you will add an attribute that later will become a relationship. Do not worry. Practice makes the master.

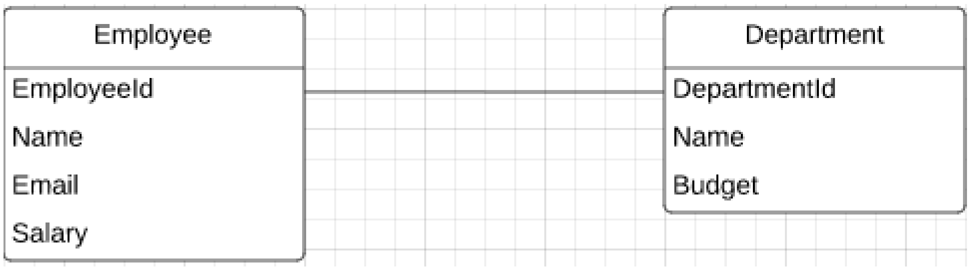
Can you think of any other entity right now? Think a bit about it before continuing. Grab a piece of paper (or open LucidChart, draw.io or Gimp) and paint the box and populate it with 4 or 5 elements.

2 Relationships

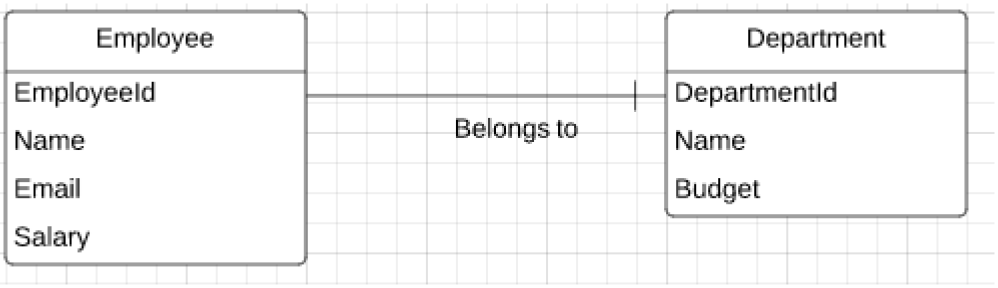
Ok, what we are studying is called **Entity**… Relationship Diagram. That should give us a hint that we have done half of our work. Specifically the Entity part … But we have to do the relationships right? So Relationships model how entities *relate* between themselves.

Thank you Captain Obvious, we imagined that

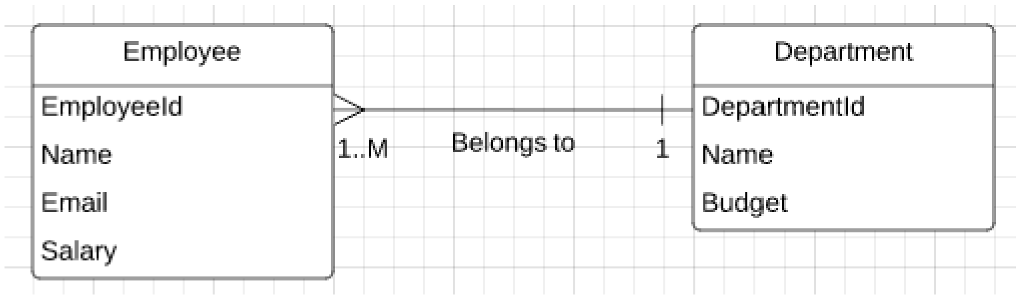
Yeah, I know it is obvious but it had to be said. So the most obvious relationship between an employee and a department is an *Employee* **belongs to** a *Department.* So as I said sometimes, when you are modelling your business, your application, your notebook…. A good hint would be to locate your *Nouns* (will become entities), the adjectives (will become attributes) and the **Verbs** (will become relationships). So back to what we said, an Employee belongs to a Department. There is a relationship there so I will draw a line (first I will draw it without cardinality.



Not very difficult right? Cardinality , as we have seen, will determine how do I create my tables, so it is worth to think carefully. An employee belongs to A/ONE department. (I know there are companies with estrange hierarchies… but shame on them. I will stay on the NORMAL behaviour when an Employee belongs to ONE and ONLY one department. So basically that means that the arrow that connects employee to department on the department side can stay as a single line, or to be even more specific we could add a 1 that will state that an employee has to have a department. We can also add the text of the relationship so our colleagues understand a bit more. Like this:



Well, that makes sense. But how is the relationship in the other direction? Well , a department should have 1 or many employees. So that means you have to add a trident… three lines .. an arrow. Or even you will see it with "numbers" 1.. M to indicate it.



So we have just finished our first relationship in our new ERD diagram. Bravo!

But at this point you could tell me. Ok, we saw a lot of these drawings in class and then I do not see how they are used. So let's take it to real world. How do we implement this:

So we have a table:

```
CREATE TABLE IF NOT EXISTS Employee (
        Employee_Id INT NOT NULL,
        Name TEXT not null, EMAIL, varchar(128), SALARY int,
        PRIMARY KEY (EMPLOYEE_ID)
);
```

And we also have the department table:

```
CREATE TABLE IF NOT EXISTS Department (
        Department_Id INT NOT NULL,
        Name TEXT not null, Budget double,
        PRIMARY KEY (Department_ID)
);
```

So we are in the critical moment, how do we create the relationship? Where would you put it? We said that many employees will work in one department. So that makes it difficult to store it in this table right?

Department			
DepartmenId	*Name*	*Goal*	*Budget*
1	Directorate	Run the school	1000000
2	Dep. A	Learn to control my powers	2000000
3	Operations	Save the world	123456

How? You coud separate it with commas but that would be a string, not a reference to the users:

Department				
DepartmenId	*Name*	*Goal*	*Budget*	*People*
1	Directorate	Run the school	1000000	1,100
2	Dep. A	Learn to control my powers	2000000	2,3,4
3	Operations	Save the world	123456	5,6

We could try to repeat the row:

Department				
DepartmenId	*Name*	*Goal*	*Budget*	*People*

1	Directorate	Run the school	1000000	1
1	Directorate	Run the school	1000000	100
2	Dep. A	Learn to control my powers	2000000	2
2	Dep. A	Learn to control my powers	2000000	3
2	Dep. A	Learn to control my powers	2000000	4

However, that has several problems:
1) Department ID is not an Id anymore. It does not identify uniquely 1 row of our table.
2) We are repeating a lot of the text. That is boring and expensive. Bad business!

So it becomes obvious that we cannot store it there, what if we store it in the other place of the 1 to M relationship:

Employees				
EmployeeId	*Name*	*Email*	*Superpower*	***DepartmentFk***
1	P. Xavier	xavi@xmen.com	Brain	**1**
2	Logan	wolverine@xmen.com	Nails	**3**
3	Rogue	rogue@xmen.com	Drains your life	**2**
4	Storm	Halle.berry@xmen.com	Weather forecast	**2**

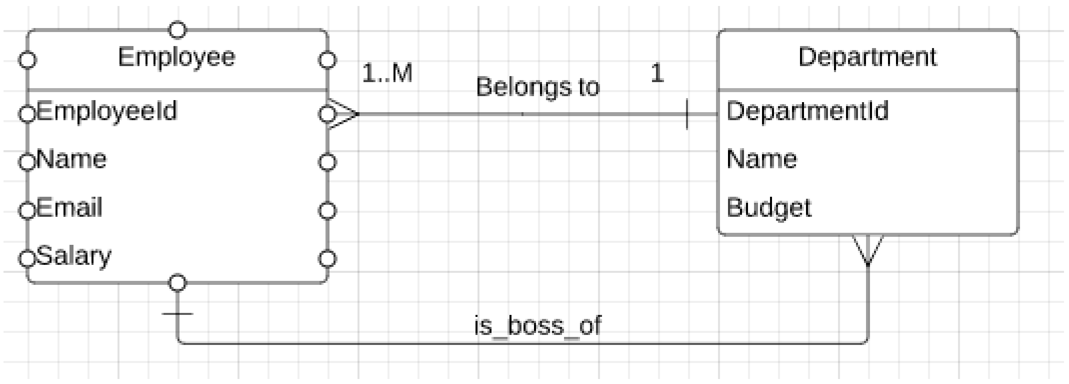
Well, this looks better, we do not have to repeat the values, employeeId is not repeated, department id is an integer that references uniquely to which department they belong.

So it looks like we have a solution!!!
How that would look like in our table SQL definition?

```
CREATE TABLE IF NOT EXISTS Employee (
        Employee_Id INT NOT NULL,
        Name TEXT not null, EMAIL, varchar(128), SALARY int,
        Department_Fk int,
        FOREIGN KEY (Department_FK) REFERENCES DEPARTMENT (DEPARTMENT_ID)
        PRIMARY KEY (EMPLOYEE_ID)
);
```

You should be capable of populating this two tables. However remember that order is important. We cannot insert Department FK = 1 if the database does not have a Department with Department ID = 1. The database will complain and will not allow us to create that row. So the easiest way to go is create first the departments and then create the employees. And voilà, you are done!

So let's practice a bit with that. We are interested in knowing who the boss of a department is. So a department should have only one boss and we will make possible that one person is the chief of many departments (because at the beginning the departments were quite small. So let's paint that:



Do you see the relationship in the bottom? Ok. So what have we learned? The foreign key should go always in the "Many side" of the relationship in a 1 to M relationship.. So in this case it has to go into department.

Department				
DepartmenId	*Name*	*Goal*	*Budget*	*ManagerFk*
1	Directorate	Run the school	1000000	1
2	Dep. A	Learn to control my powers	2000000	1
3	Operations	Save the world	123456	2

So here we see that Professor Xavier is the boss of Departments 1 and 2 and Logan is the boss of Operations. You see that I did not write EmployeeId as name of the column. You can name it whatever you want, as long as later it is easy to know what are you storing there. If I had added only EmployeeId it would be difficult to know what I meant, so in Employee we added DepartmentFK because it is obvious that we refer that the employee works for that department but as you can see here it is better to say that is the manager. So how does that look in SQL

```
CREATE TABLE IF NOT EXISTS Department (
        Department_Id INT NOT NULL,
        Name TEXT not null, Budget double,
        Manager_Fk int,
        FOREIGN KEY (Manager_FK) REFERENCES DEPARTMENT (DEPARTMENT_ID)
        PRIMARY KEY (Department_ID)
);
```

As you can see the name is something to query later and to see in one sight what does that column refer to. The real information is stored in the yellow line. It is in the yellow line that we tell the database that Manager_Fk is not a simple INT column but a Foreign Key.

Now let's go to insert the data! Yessss!! "But wait a minute… You told us that order mattered in inserting. And that we should first insert in the table that does not hold the foreign key, so the element created can reference the other table…. And now, you have just created a circular dependency. I will never be able to insert in those table :'("

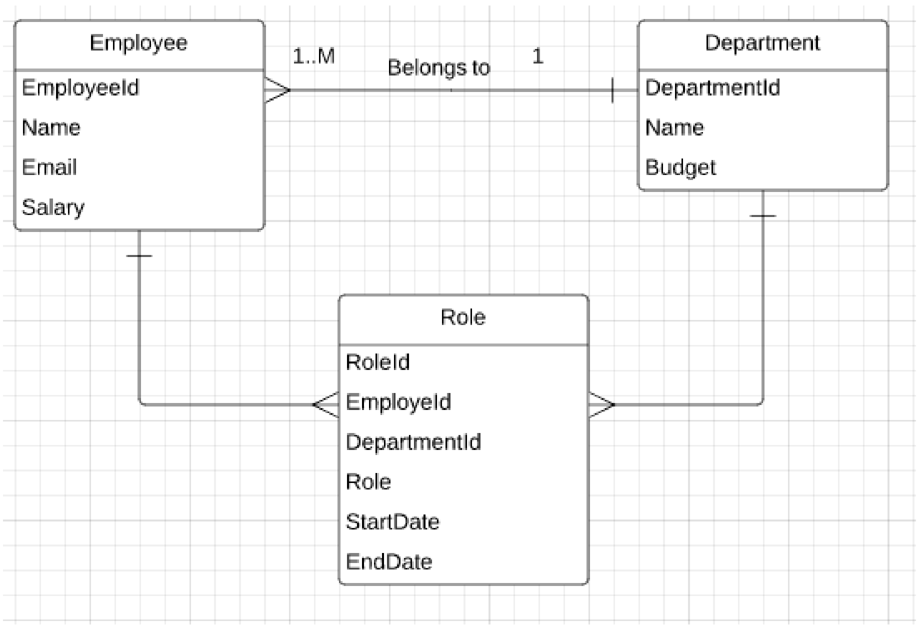
Don't panic! There is always a solution…

Let's see:

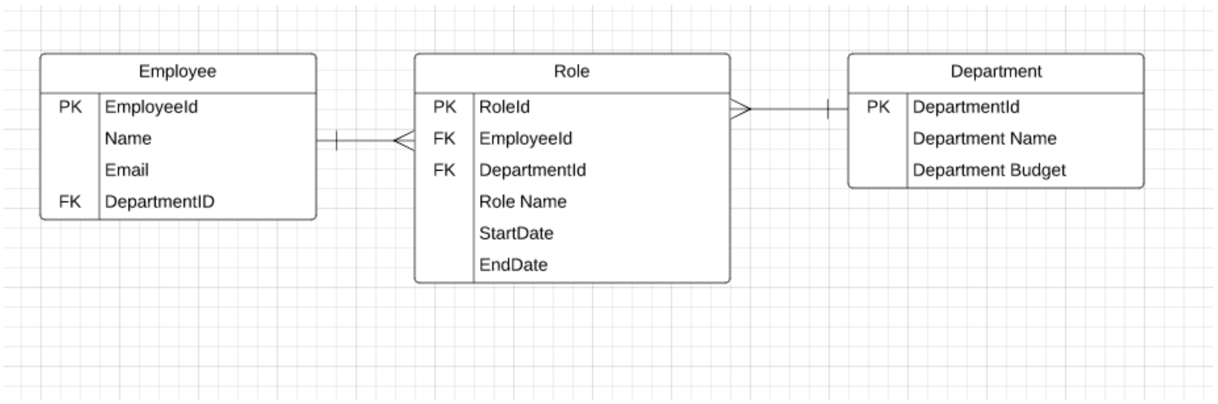
```
-- we insert the departments without boss:
INSERT INTO DEPARTMENTS (1, 'Directorate', 'Run the school', 10, NULL) ;
INSERT INTO DEPARTMENTS (2, Dep. A, 'Learn to control the powers', 20, NULL) ;
INSERT INTO DEPARTMENTS (3, 'Operations', 'Save the world', 40, NULL) ;

-- we insert the employees giving them a department (that already exists)
INSERT INTO EMPLOYEE (1, ' P. Xavier ', xavi@xmen.com', 'Brain', 1) ;
INSERT INTO EMPLOYEE (2, 'Logan', 'wolverine@xmen.com', 'Nails', 3) ;
INSERT INTO EMPLOYEE (3, 'Rogue', 'rogue@xmen.com', 'Drains your life', 2) ;
INSERT INTO EMPLOYEE (3, 'Storm', 'hberry@xmen.com', 'Weather forecast', 2) ;

-- we update the departments with the ID of the employee that now does exists
UPDATE departments set manager_fk = 1 where department_id = 1 or department_id = 3;
UPDATE departments set manager_fk = 2 where department_id = 3;
```

It is done!! Congratulations. We advance.

Ok but… there is always something more. My manager came and told me that we do not want only to store who is the manager but also many other roles and jobs in the system. Suddenly our relationship became an MxN relationship. Because an employee can have different "roles" in different departments and in a department you will have several people with different roles. So an MxN relationship appears, and when an M to N relationship appears it means that we need an extra table to hold than information. We know how to implement 1 to N relationships but we do not know how to encode a MxN, so what we will do is splitting the MxN relationship in two 1 to M relationships like this.



So now we can remove the ManagerFk from department and then we have something like this in SQL:



As you can see in this graphical representation of the database, the manager disappeared from the department because now we can store that information in the Role table. So let's do it:

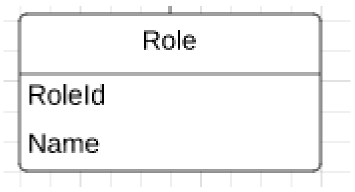
```
-- Prof. Xavier
-- He is the manager for departments 1 and 3.
INSERT INTO ROLE (ROLE_ID, EMPLOYEE_ID, DEPARTMENT_ID, ROLE_NAME, Start_date, End_Date)
VALUES ( 1, 1, 1 , 'Manager', 01-01-1990,  22-07-2019);
INSERT INTO ROLE (ROLE_ID, EMPLOYEE_ID, DEPARTMENT_ID, ROLE_NAME, Start_date, End_Date)
VALUES ( 2, 1, 3 , 'Manager', 01-01-1990,  22-07-2019);
-- he is also director for department 2
INSERT INTO ROLE (ROLE_ID, EMPLOYEE_ID, DEPARTMENT_ID, ROLE_NAME, Start_date, End_Date)
VALUES ( 3, 1, 2 , 'Director', 01-01-1990,  22-07-2019);

-- Logan is manager of Department A and Operator in Operations.
INSERT INTO ROLE (ROLE_ID, EMPLOYEE_ID, DEPARTMENT_ID, ROLE_NAME, Start_date, End_Date)
VALUES ( 4, 2, 2 , 'Manager', 01-01-1990,  22-07-2019);
INSERT INTO ROLE (ROLE_ID, EMPLOYEE_ID, DEPARTMENT_ID, ROLE_NAME, Start_date, End_Date)
VALUES ( 5, 2, 3 , 'Operator', 01-01-1990,  22-07-2019);
```

So as you can see, we have just implemented the solution. Besides another good effect is that now, we do not have the circular reference, so to recreate the database you could start by the Departments, continue with the employees and finish with Role.

However, I have seen that I am repeating the word Manager many times. And I have always told you that we try to reduce copy/paste , repetition etc. right? How is that possible that there is that "Manager" string repeating itself?

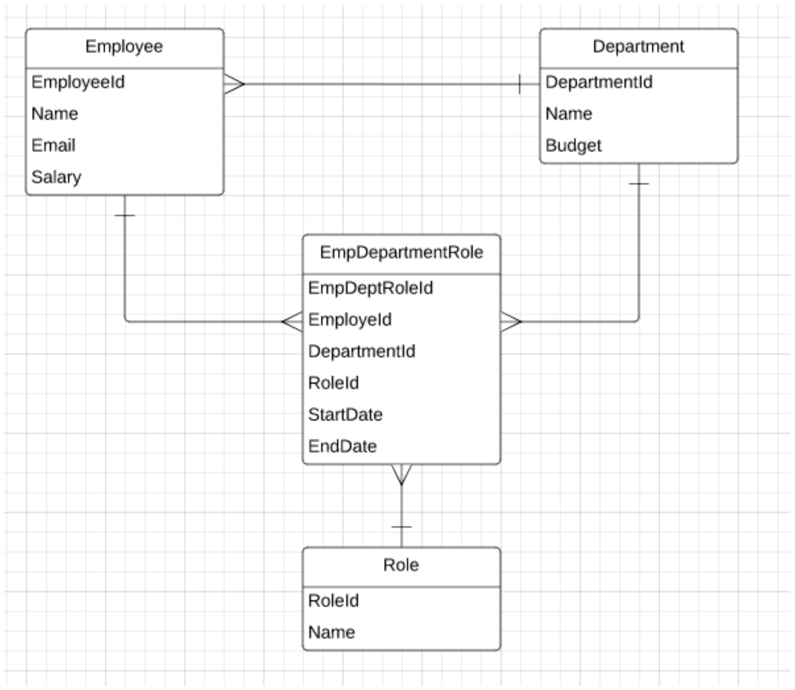
Ok, so let's extract it from there to a table and we will have a beautiful example of a *'ternary relationship'* Do not be scared by the name, it is only a relationship between three tables.

We have very small table:



Role	
Role Id	Name
1	Manager
2	Operator

And the relationship would look like this.



So to create it it would be

```
CREATE TABLE IF NOT EXISTS Emp_Department_Role  (
        Emp_Department_Role_Id INT NOT NULL,
        Employee_Fk int,
        Department_Fk int,
        Role_Fk int,
        FOREIGN KEY (Manager Employee_Fk) REFERENCES EMPLOYEE (EMPLOYEE_ID),
        FOREIGN KEY (Manager Department FK) REFERENCES DEPARTMENT
(DEPARTMENT_ID),
        FOREIGN KEY (Manager Role_Fk) REFERENCES ROLE (ROLE_ID)
        PRIMARY KEY (Emp_Department_Role_Id) );
```

And now you can populate it

-- we create the roles

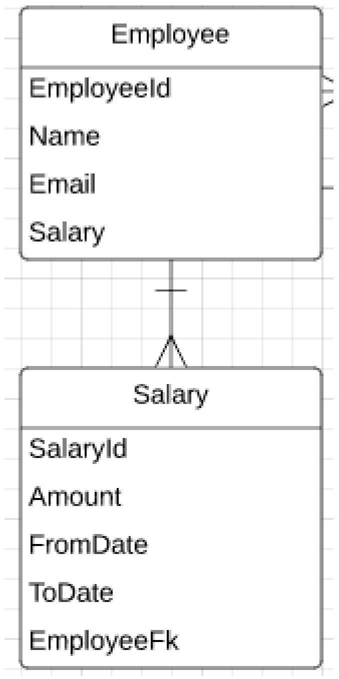
INSERT INTO ROLE (1 , 'Manager'), (2,'Director'), (3,'Operator');

And then we add Logan's roles

```
-- Logan is manager of Department A and Operator in Operations.
INSERT INTO Emp_Department_Role  (Emp_Department_role_ID, Role_id, EMPLOYEE_ID, DEPARTMENT_ID,
Start_date, End_Date)
VALUES ( 4,1, 2, 2, 01-01-1990,  22-07-2019);
INSERT INTO Emp_Department_Role  (Emp_Department_role_ID, Role_id, EMPLOYEE_ID, DEPARTMENT_ID,
Start_date, End_Date)
VALUES ( 5, 3, 2, 3 ,  01-01-1990,  22-07-2019);
```

Perfect!!! Let's continue practicing. My boss has asked me to keep track of the salaries of my employees during time. So before continue reading try to think of how would you do it.
Hint: The salary will have to disappear from the Employee and become an Entity.

Let's go with it



This is what I came up with. An employee can have many salaries over time but a specific salary will only belong to a specific employee.

SalaryId	Amount	Period	EmployeeFk
1	30.000	1/1/18 – 31/12/18	1
2	32.000	1/1/19 – 31/12/19	1

So now we see that Prof. Xavier received an increase of 2.000 euros last year.

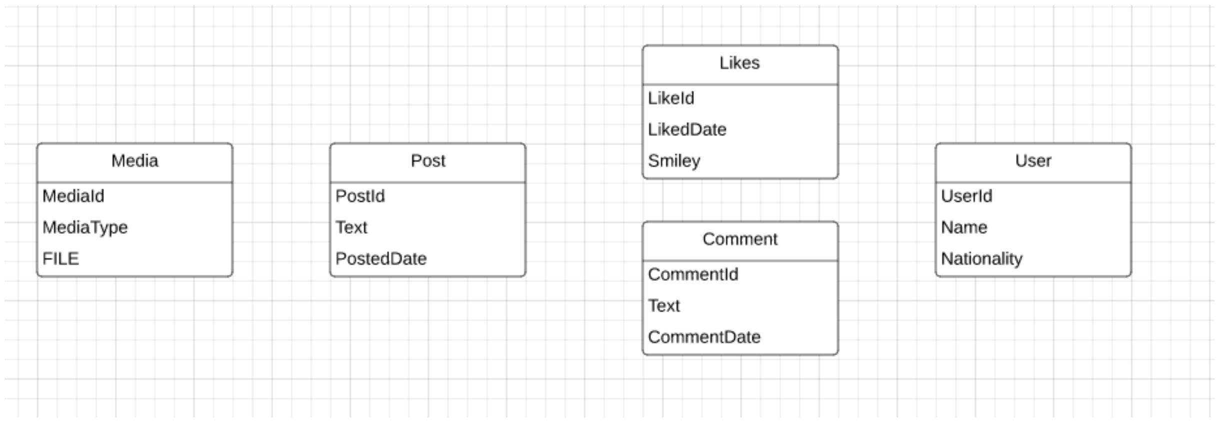
Someone could have asked to put salaries in another table and then a relationship MxN with employee. You could do that. In my opinion there is not much repetition (even if there are two employees with 30.000 salary is a very broad value to differentiate a salary per employee. Besides, here I could add if that Salary has bonuses, or any other condition related to the contract with my employee. But if you want to implement it with a MxN relationship and you explain it nobody could tell you that you are wrong.

Try now to do the next exercise. First add a relationship 1 to N between a department and a project. Like "One department works in many projects. And a project belongs to only one department.

Once you have done that, evolve the diagram / SQL in a way that the different departments can collaborate in a project and each Department will play a role for a specific project. Hint: MxN relationship like a DepartmentRole.

Another exercise:

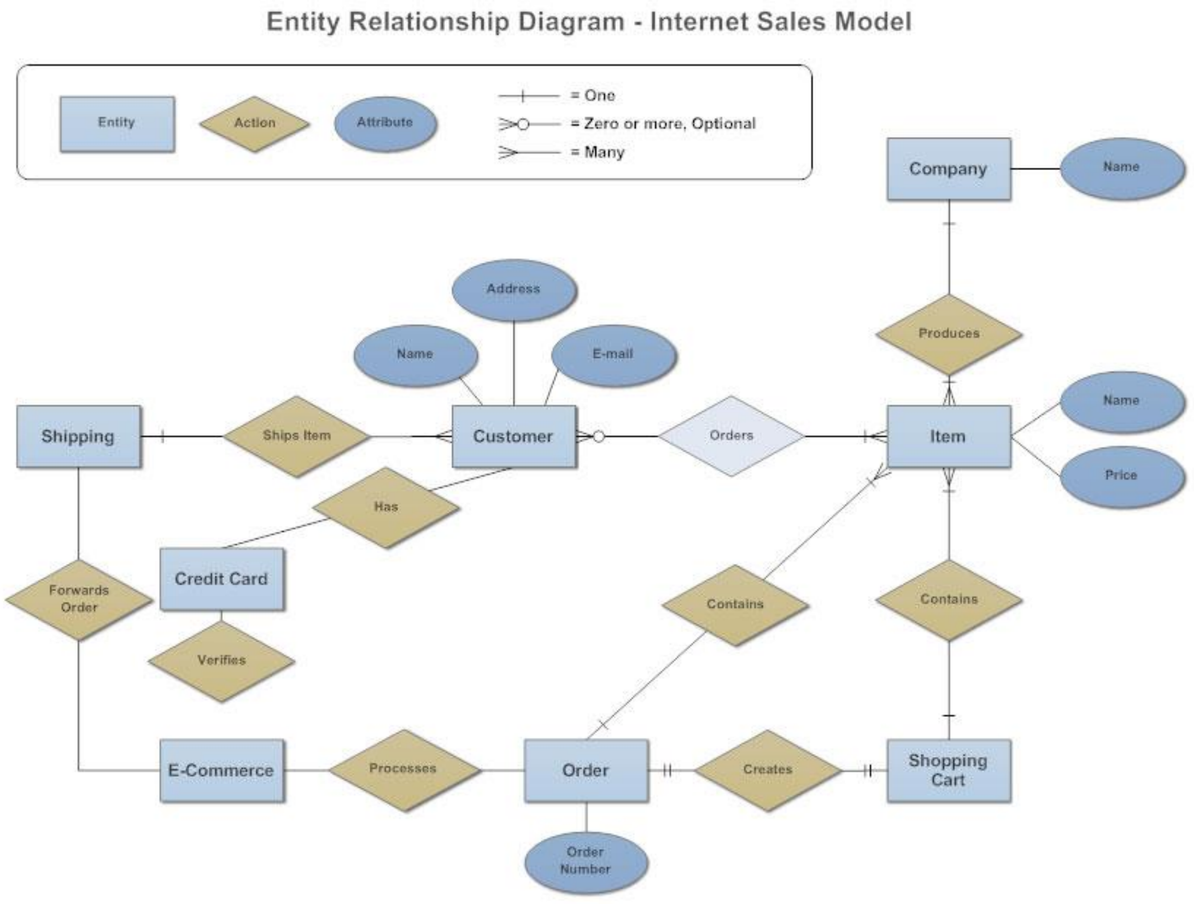
I offer you these entities to start your own Facebook database. Try to connect the different entities offered. Think of the cardinality. Try to create the tables in a new database called MyFacebook;



Add whichever entity or extra relationship you consider it is missing.

One more exercise, think of your favourite website or application (Google Mail, Whatsapp, Amazon, …choosing Facebook or Instagram is cheating a bit.. but in that case expand a lot the model above). Try to locate the important entities of it and create your own model for it.

APPENDIX:



Sometimes you will see this type of Diagram. Do not worry, it is the same of what we learned the only thing is that it is a bit more conceptual.

Squares are entities = The same we were using
Circles are attributes = Normally we were adding the attributes in the square.
Rhombus = represent relationships.

GOOD LUCK!