

Hi.  
  
This video will be about the goals of logical design.  
  
What I mean with logical design here is that the move from the diagram that we create towards the actual table structure, and in fact, crucially, how we represent the relationships between entities within the tables.  
  
There are two main goals of logical design.  
  
The first one is we only want to have a minimum number of tables because we do not want to split up the data too much.  
  
This has two reasons.  
  
If we have too many tables, we lose oversight where my data is.  
  
Secondly, and perhaps more importantly, if we have the data stored in several tables, once we query it, we will have to query somehow the join of those tables, or the join at least of a lot of tables.  
  
And whenever we join tables in a query, we pay a price.  
  
Namely, we have to wait longer for the answer.  
  
The more joins you use, the longer you usually have to wait for the answer.  
  
Therefore, if we have a lot of tables, we will have to use a lot of joins to get to our answers, and that's not good.  
  
So, therefore we want to minimise the number of tables.  
  
The second goal is that we want to minimise the number of null values within each table.  
  
Why do we want to minimise the number of null values if there are loads of null values?  
  
It just indicates that the structure of the table doesn't fit well to the structure of the data.  
  
If the structure of the table doesn't fit well with the structure of the data, that just means that you don't understand the structure of the data.  
  
Well, if you just look at the schema of the table and the user will have trouble understanding what the data is about, how to query it and where to look for data in this sea of null values.  
  
So we want to minimise number of tables, minimise number of null values.  
  
So, moving from the diagram to tables, the first step that is the most basic one, but fundamentally important, is turning all the entities into tables.  
  
So we just derived skeleton tables for all the entities.  
  
So the name of the table will just be the name of the entity and the attributes of the entities become the attributes of the table and the identifier of the entity becomes the key of the table.  
  
This is the first basic step.  
  
It's very straightforward.  
  
Hardly anything can go wrong.  
  
However, now we want to turn relationships into tables as well.  
  
And for this we will see in a second, exactly, step by step, how you do this with any possible relationship.  
  
But for now let me focus on one to one relationships.  
  
Let's first look at the following relationship between car and employee.  
  
It's obligatory on both we have two tables here.  
  
One lists the cars and the other one lists the employees.  
  
Now clearly, if you look at the data in those tables, there's no way that you can see which car is assigned to which employee.  
  
Because so far we did not represent this relationship assigned to between cars and employees.  
  
Somehow we have to now represent this.  
  
Well, here we have a relationship that is obligatory on both.  
  
So for each car there will be a corresponding employee, and for each employee there will there will be a corresponding car the employees are assigned to.  
  
So we have exactly as many cars as employees and they are assigned to each other.  
  
So in this case, what we would do is we just copy everything into one table, because there's no need for not doing this basically.  
  
So now we just have all the cars together with the corresponding employees in one row.  
  
And then we just choose the identifier 1 of the 2 as the identifier of the row.  
  
In this case, we choose the identifier of the employee to be the key of the new resulting table.  
  
This is kind of one of the nicer options because what we now have is we represented two entities in one relationship, all in one table.  
  
There are no null values.  
  
This is the best case, in fact.  
  
Now let's go to a slightly changed scenario where we have still cars employees.  
  
But now the assigned to a relationship is non obligatory on employees.  
  
And so the idea here is just that not every employee has a car assigned, but every car has an employee assigned.  
  
So now what could we do?  
  
So if we could attempt to do the same thing that we did just now, that's one option.  
  
In fact, the nice thing about that option is that you always end up with one table.  
  
So what we do is we just copy employees together with their cars into one row.  
  
But the problem now is that not every employee has a car.  
  
So what we end up with is we end up with several rows where the car of the employee just contains null values.  
  
So now we ended up with one table and many null values, which is good on the one hand, but bad on the other hand.  
  
So what could we have done instead?  
  
We could have done the following.  
  
Well, every car has one employee assigned to.  
  
So how do we do that?  
  
Well, we kind of just make a pointer to the employee in the car table.  
  
So what we do is now we copy the identifier of the employee to the car it is assigned to.  
  
So what we do is we add a new attribute into the car table, call it number for the employee number, and we put there the number of the employee that is assigned to that car.  
  
This will never be null, because we know that the relationship is obligatory on car, meaning that for each car there will be one employee, so there must be some.  
  
So we don't have any null values in this design, but now we have two tables instead of one in the previous design.  
  
So there's a certain trade off between these two goals of logical design Minimise the number of tables versus minimise the number of nulls.  
  
And sometimes it's a matter of taste, and it also depends on what queries you usually running on the data to whether to go for one design or for the other design.  
  
Thank you for listening to this video.  
  
See you next time when we discuss in detail how to deal with any possible relationship.