

Hi.  
  
Last time we introduced the relational data model in detail, but we've not introduced all the constraints yet.  
  
So we finished by discussing so called integrity constraints related to the key of the table or the primary key of the table.  
  
And so there the idea was, the key gives us a way of addressing exactly the rows and the constraints.  
  
The integrity constraints related to it is that every row has to contain actually values for the key.  
  
And also each row has to have a unique value for the key.  
  
So there can't be two rows that have the same values for the key attributes.  
  
Let's have a look at this example here.  
  
That violates in fact integrity constraint.  
  
So we have this table again with films title and release here, and we've seen that the film ID is in fact the primary key of the table.  
  
And then we have here a fifth row which just says film ID null.  
  
And so clearly this is a violation of the integrity constraint.  
  
Another violation would be if in this table, the third video wouldn't have ID F3, but it would have IDF2, because already the one in the second row has F2.  
  
So we couldn't have two rows which have both film ID F2, because otherwise we couldn't anymore identify the row just looking at the film id.  
  
So this is an illegal structure.  
  
We have to maintain the integrity constraint.  
  
What other constraints are there?  
  
Well, very important one is the so called referential constraints.  
  
And so this brings us to a constraint where we have to make sure that value in one table points actually to a row in another table.  
  
So it often happens in relational database that we have an attribute of one table.  
  
For example, in the film table there is something called the director id.  
  
And this points to a key of the director table, which is indeed the director id.  
  
And so this is what's called a foreign key.  
  
So a foreign key in one table uniquely identifies a tuple in another table.  
  
So indeed in the example director ID is a foreign key for film, because it tells me, look there for the director.  
  
And so the relationship between in fact film and director is many to one.  
  
And we will see that foreign keys are exactly a way of representing those relationships.  
  
So in this example indeed you see the foreign key in the director ID referring to the director in another table.  
  
Now you might ask, why not just store everything in one table?  
  
Why have this problem with pointers to other tables?  
  
And then you will realise quickly that this would lead to massive redundancies, because one director might have 10 movies.  
  
Now, if you would always store all the information in one table, you would always have to repeat all the information about the director for each movie that he made.  
  
Rather than doing this, he's saying we just say this movie is made by the director and all the information about the director is in another table.  
  
To do exactly this always properly, meaning distribute the right data over the tables, is also a big part of what this course will be about.  
  
It's especially when we come in the last part of this class to normalisation.  
  
Okay, so the referential constraint associated with the foreign key is that whenever there is a row that has a value in the foreign key, that value should have its correspondent row in the table it points to.  
  
So if we have a row with a director id, it has to have in fact the row in the director table with this director id.  
  
So what happens now if this thing is violated?  
  
Well, the relational database will somehow maintain the referential constraint.  
  
So if I try as a user to enter a row in the film ID and then specify an ID that doesn't point to any director, SQL will just prevent me from entering it.  
  
On the other hand, what happens if there are already like 10 movies by a director, the director ID D20 and I would just try to delete that director in the directors table, Then suddenly all those movies would point to nowhere.  
  
So what we can do then is we can tell SQL how it should act in this case.  
  
Either it should prevent the user from deleting the director to which something points, or we can just say cascade.  
  
So cascade would mean that, okay, if you direct, if you delete the director, then SQL also deletes all the movies of the director in the other table.  
  
But you see already this can have unforeseen consequences.  
  
So cascade is hardly used.  
  
The normal thing is always that SQL will prevent you from violating referential constraints.  
  
So as a rule of thumb, I can already tell you as a kind of look ahead, whenever we have a one to many relationship between two entities, like in this case, we have one director has many movies, we will resolve this using what is called a foreign key, which leads to these referential constraints, meaning that the values of the foreign key have to point actually to something that exists in the other table.  
  
And so in general this will be easy.  
  
The big challenge will be then to deal with many to many relationships and will be discussed soon.  
  
Thanks for watching, see you next time.