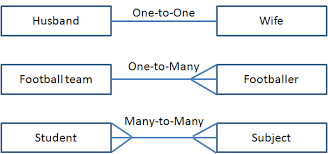
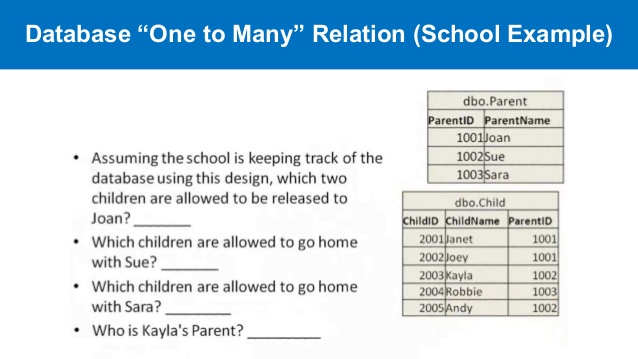
**Week 4 Day 2 Notes**

Today we’re covering how to create relationships between different databases/tables. We’ll not so much be talking about how to do something the correct way, as we are about how to avoid doing something incorrectly.

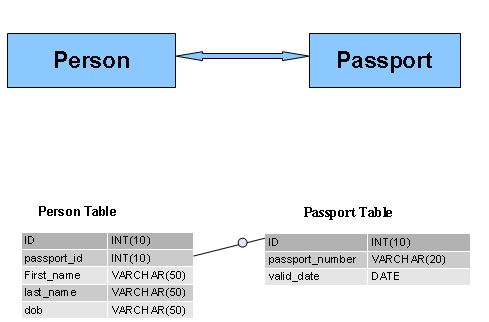
There is no such thing as behind in the course. They intentionally give us more things than we can possibly do. Don’t focus on getting checkmarks in Alexa, focus on learning concepts. Workload is meant to be customizable.

* Data modelling means deciding what are the different types of data (tables, classes, attribute, columns, relationships between them) and their relationships at the start of your project.
* For each of your classes there is a table, the row represent instances, the columns are instance variables or attributes of that class.
* Primary key column is usally called id, and usually starts at one and iterates by one.
* Foreign keys are the mechanism used to create links between databases.
* You can have more than one foreign key to track more than one relationship, because of course there can be more than one relationship between two pieces of data. You have tables that are only tracking foreign keys and the primary key. Foreign keys are the only way to join tables, in this particular system of databases.
* Id should/usually will just be called id while foreign keys will usually be “something”\_id.
* Rows are instances, columns are attributes, tables are classes?
* Sometimes one to many, many to many, or one to one relationships represent an action, but sometimes they are just representing a group relationship (such as a track belonging to a playlist)





* One-to-one relationshps don’t come up very often. Sometimes its convenient to seperate two pieces of data into two different entities rather than merging them, so creating the one-to-one relationship is necessary.



* **If you want to store a one to many, the rule is whichever side is the “many side”, you put a foreign key on that table to connect it to the one side. For a products to reviews database, you would put the foreign key in the review table.**
* If you want to store a many to many relationship, there has to be a third join table that has foreign keys referring to both sides.

Users ------------- Reviews (table w/ tw­o join keys)------------Products.

For each user there can be many reviews, and for each product there can be many reviews. Users to products is one-to-many.

Logically this is the only way to map a many-to-many relationship.

* With a many-to-many relationship (for example students/courses) we describe it by saying:

Student “has\_many” courses and “belongs to “many\_courses”.   
Course “has\_many” students and “belongs to “many\_courses”. Phrases in “-” are used in ActiveRecord rails to describe relationships.

* The rule for mapping a one-to-one relationship is the same as one-to-many, its just the decision to put the foreign key on one side rather than the other is arbitrary. You don’t want to put foreign keys on both sides, because that would be repreating yourself. We want DRY databases.
* Two map different many-to-many relationships for the same two things, you would need two seperate join tables. For example  
  Users -------------------- Films  
  Could have a join table that represents films that have been watched, and also films that the user wants to watch.
* If something has a primary key, then it will be a class by itself.

If multiple lines or columns will need to be changed, you should create a seperate table to represent the relationship of those lines.

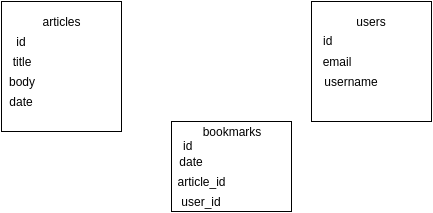
Signs that join table should have its own class:

Is there more info about the relationship than the two things it is linking (the foreign ids for each)?

Is there a name/noun that describes this relationship? Favoriting / retweeting.

Is there there going to be more info about it in the future?

When it doubt give it a class.



In diagraming many-to-many relationships, you have to conceptualize the relationship from the point of view of each side.

Data modeling for an app, you have to make decisions on what relationships to track, it won’t necessarily make sense to create many-to-many relationships for every table in your database. You’ve got to make rational decisions based on what you want your app to do.