courserd

**Lecture Material** 

Many-to-Many

Relationships

◀ Back to Week 4

Assignment

**Bonus Materials** 

Catalog

14 min

**X** Lessons

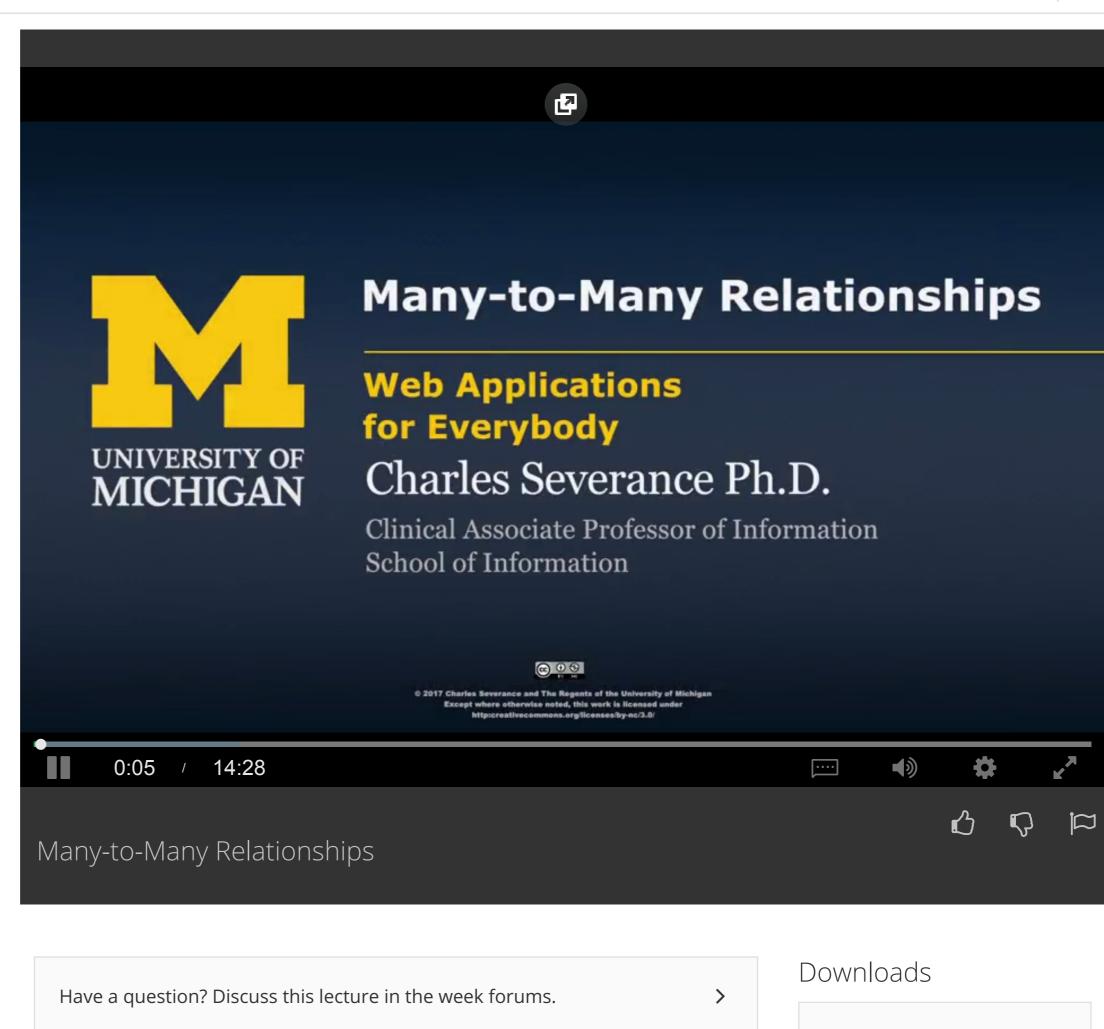
Search catalog

Q For Enterprise



Prev

Next



## Lecture Video mp4 Interactive Transcript Subtitles (English) WebVTT English ▼ Search Transcript **Transcript (English)** txt 0:08 Okay, so everything we've been talking about up till now is what's the one to many relationship. But there's other kinds of shapes of relationships between Would you like to help us translate the transcript and tables and the biggest new one that we haven't talked about is what we call subtitles into additional Many to Many relationships. And so this is my data model for the software languages? that runs the auto grader called the Tsugi software. And one of the more common situations is the course to user mapping. And so the problem is that you can have many users and you can have many courses and many users can

be a member of many courses and many courses can have many users. And so the notion of one album belonging to an artist and it turns out that album artists really should be many to many. So album and artist really should be many to many. But the idea is we have to have a relationship. And so what we can't do is we can't say there's users to courses, we can't have the many on this side and the one on that side. And we can't go users to courses and have the many on one side, the many here and the one on that side, you can't do that. The relationship is truly, between users and courses is many to many. So what you end up having to do is you actually make a table. Table in the middle, right? And so we'll talk about this. So everything we've done so far is many to one. And that just means that you can have lots of rows that are like seven, seven, seven in the track and then there's one seven over here, there's one seven. So lots of foreign keys, lot's of rows. The foreign key, you can do vertical duplications of foreign key but you can't also then have this. And also if you really know music, a track can be on more than one album. We simplified that for the previous lecture. Okay, so the way we did it was a many to one, okay? But, sometimes there are things that are naturally many to one but sometimes there are things that are actually many to many. So lots of authors, a book can have multiple authors and an author can write multiple books. And so we really just have to draw the picture like this. This is crowfoot diagram that stands for many many, it's many to many. And the logical diagram we'll just say it's many to many. It just is, okay? Let's not argue about it. You can't do it any other way. But then we have to turn it into tables, just like we had to add columns for foreign keys. Well, now we instead of just adding columns, we have to add a whole table. You'd call it junction table, a many to many table, a connector table, a relationship table, all kinds of things. And so what we do then is we break this many to many relationship into two one to manys. So this is a many to one, this is many to one, one on this end, one on that end and then there's many. And then, we can put lots of author book combinations inside here. And it's almost easier to take a look at this in a practical example, right? And so, if we think of accounts and users and we have a many to many relationship between people and courses and the courses that they're in et cetera, which is really a roster is a combination of, we might call this thing a roster in the middle, and that is the mapping of what people are in what courses. So again, we're going to break this in the middle and make a little table in the middle and put the manys back and forth, right? So, this is how you do it. So, what you do is you create a table. And so this is the first table, every table we've created till now, has a primary key, right? Because we want to get it that row. These connector tables or join tables or junction tables, they tend not to have a primary key. And we'll see in a second how we create sort of a meta-primary key that has a uniqueness constraint on it. That is the combination, so you can only have the user, an account course combination once but one account can have many courses. So that uniqueness is the combination of the two numbers not the number itself. So the primary key in a sense is like a meta-primary key, which is those two things connected together. And so, we end up with a foreign key, a foreign key pointing to a primary key, primary key. These two tables are just totally normal. Now the interesting thing is you sometimes actually do model some data in here. And in a course situation, you can also model the role, whether they're a teacher or a student. Because they're not, it's not like you're a teacher forever or that whatever, you can't model fact that you're a teacher and say everything you do because sometimes you're in a class that you're a teacher and sometimes your class that's a student. And so the fact that you're a teacher-student gets modeled at the connection, right? So we have this connection, it's a many to many connection, but we kind of have this role that's modeled at the connection point between these two things. And so that's an aspect. You don't always have to have data in the connector table but it's also okay. And sometimes the data in the connector table is the only way to put it. So another thing you might put in here is like grade or something. Because you can't say you get one grade because you're in many classes, so you might have a final grade in lots of classes. And so it's what grade did you get in the class that you're in, okay? So there is data to model sort of here in the middle at times, in the middle of the connection. So in a logical diagram we don't really draw this picture but there is data that lives there. So, the tables that we're going to create are pretty straightforward. We'll start with the outside ones. We've got an account table. By now, that should be familiar. We have a logical key. We're going to put a uniqueness constraint on it, the name, and then primary key is Account ID that's just telling inside here. The same for courses, we got a primary key, autoincrement, yada yada. Copy, paste, copy, paste. Do that stuff all over again. Okay. So those two are are just like we've been doing. The fun one is this next one, okay? And so, in this one, we're building this member table which is the thing. And so, you'll notice there is no primary key. There's no auto increment, right? There's no autoincrement here. We don't have that. So, what we have is we have two foreign keys. We have two arrows that start in this table. So we have two foreign keys, so Account ID and course ID. And then, role is this little bit of data that we're going to model right there. There could be a couple of other things here if we're going to do this. And then we have two constraints. And what all the constraints really are doing is capturing these two arrows. So that says that this account ID inside this thing actually points to a row over here in the account table with a primary key of account ID and this course ID that's in my table points to a corresponding row in course indexed by course ID. And we're going to do an update cascade and undelete cascade. We talked about that before. And so that's most of this but here is the fun part. Look at this primary key, right? And all the other ones, we had a primary key and we indicated that was the primary key. Now, we don't have a single primary key. We have a combination primary key or a concatenated primary key and so we just give it two things. And what this says is the combination is unique. Account ID is not unique. Course ID is not unique. But the combination of any pair of account ID, course ID is unique. And so, we kind of have a primary key but it's both these numbers. Now the thing is, is it sets up an index to look up the combination of the numbers really fast. So this actually drives an index. And so, looking things up as long as you know both numbers is really fast and it turns out, given that you're kind of connecting with joins between all these courses, you know what those numbers are. So I can look these things up really fast. Super fast. Fast. Fast. Numbers not strings. Sorry, there's little ad that pops up everywhere in my head like "Don't use strings. Don't use strings. Just use numbers." Okay. The inserts now kind of feel pretty natural because we're used to these are the leaf tables. So we're going to put some folks in, right? Jane, Ed and Sue, and Jane's e-mail, and then we are establishing cause those have autoincrement primary keys. So we're establishing the internal key for each account. Then we're going to do the same thing for courses, yada yada yada. And then we are going to come up with the primary keys and then we have to write those down on our little piece of paper, right? Because now we have to model at the connection. Okay? And so, now we're going to connect these things in. And so, we're going to put in little bits of information. We're going to insert into the member table. We've got two foreign keys which effectively are the primary key but they're not autoincrement. It's just a primary key so, account ID, course ID, enroll. So that means that account one, Jane is in course one as the instructor, and account two, Ed is in course one as a student, and Sue is in course one as a student, and Jane is in course two as a student, and Ed is in course two as a teacher, and Ed is in course three as a teacher, and Sue is in course three as a student. So what we're doing is we're making these things are like in here and we're putting connections where appropriate, and it doesn't have to just be one connection. Okay. And so, all we have to know all these numbers by knowing these numbers and then knowing how to connect them but this is really just modeled data. That's the role that's there. So we're going to insert those. That's just drawing little lines as how we draw lines inside of databases. And so, if we're going to join this back together, we're going to want to produce a bit of output that Ed is in PHP as a teacher, Sue is in PHP as a student, Jane is in Python as a teacher. So, how to produce that? Well, you write a join statement, right? You write a join, and I think of this as chiny joining through, right? We join through the membership table. So we're kind of taking from two sides and we merge all these things together and we pull this all in. Okay? So we're going to select the account name, the member role and the course title. That's going to get us all this stuff. And then we're going to join from the account, join with a member, join with the course. That's creating the super meta row. It's kind of this big. Remember here is, in there, members going to contribute role basically. And the on clause, three tables, two relationships, arrow, arrow, on clause, member account ID equals the account ID, and the member course ID equals course course ID. Again, you should be able to write these in your sleep if you've been naming your things correctly. Now I got an order by here that basically says, I'd like this to go in ascending order of course title, followed by descending order of member role, and then followed by within a role, so in this case. So, this is the most important sort. This is the second most important sort. And then when these two things match then, Ed and Sue will sort that way. And so now, we can sort of get a course roster list, who's the teacher, who's the student in all of our courses by joining all those things together. And again, what I just showed you is pretty much how we do it in real production systems. Here's a user table. Here is a course table, contacts table. Here is a foreign key, context ID, user ID. Here is some information role that's actually modeled right there. And

look, it knows about the fact that this is a many-to-a-one. And this was in a

many-to-a-one which means that we effectively have created a many-to-many

except that we use this junction table to achieve the many-to-many between

users and courses. This is why I say once you kind of know the stuff that's in

here in this modeling, you can kind of build professional grade to serve the

data model. There will be things you will learn. I mean this is not the end of

your learning. There will be other things that you will learn but, if you can see

this, you actually know a whole lot. And I just need to emphasize that yes, you

system, you can't pretend that the data is going to be small. You got to model

it right. It's so much better to just model it right from the beginning, get good

at what we're learning in these classes, and then if you happen to scale, you're

ready for it. It might seem like a tradeoff. You might like, "Oh boy, there's a lot

doesn't just slow down a little bit. It dies. It doesn't work at all. There comes a

point where your demand causes it to just die and you just can't throw more

resources at it. And so this is where startups, you always hear about them, "Oh

we wrote this cool thing and oh we had to throw it all away and rewrite it." And

that's the end of a whole bunch of lectures talking about database. Databases

speedy. The key is one copy of string data elements don't allow that vertical

know enough of it to be dangerous. The key is, is that storage is smaller and

the amount that scanned is smaller. And so that's really cool. And it is an art

form and literally, you can learn the basics and they're beautiful, and you can

actually be very competent, but you will learn forever. I love sitting down

with somebody smarter than me about databases and say, "What about

normalization. Sometimes you apply them and then sometimes you got to

cross the streams, like "Oh, wait a sec." So, there's a lot of learning. The fun

thing is to learn the basics and then later, you can learn the exceptions.

this? What about that?" And even some of the rules of database

Cheers.

duplication. That's called database normalization. It is a science of its own. We

could store this in a spreadsheet. But, speed is the key. And, in an online

of work there." But the answer is, your application simply dies. I mean it

that's because they didn't think about their data model early enough. So,

allow us to scale to large amounts of data and pull those things in very

first 10,000, 20,000 users of an application. You can build a professional grade