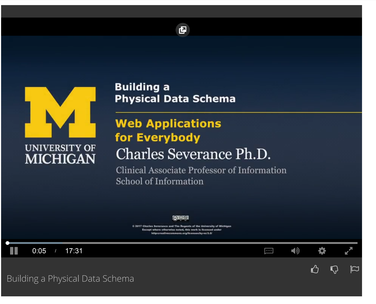
Lecture Materials

- Relational Database Design 14 min
- Normalization and Foreign Keys 9 min
- Building a Physical Data Schema 18 min
- Relational Database Design #4 14 min
- Practice Quiz: Database Models 15 questions

Assignment

Bonus Materials



Building a Physical Data Schema

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:08
So, now, we're going to work through an example, where we're going to put these actual numbers in tables. And if you take a look at what we started with, we drew this picture on the wall in our conference room. We drank a lot of coffee. Now, when we drew this picture, we didn't sort of worry about nerd mechanics of this, right? We didn't worry about the exact name of the columns. We're really focused on this is like what the logical structure. And now, we're going to map this sort of logical structure to a more physical structure. And so, what we're going to do is, this was a part of that picture that we drew. And now, what we're going to do is say, "You know what? We're going to map this to sort of exactly what we're going to name the columns, what we're going to name the tables, et cetera." So, what we're going to do is we're going to start and say, "Okay, we're going to make a track table." And we're going to put a primary key in. You'll see on almost every single thing we do, we just, as first, you just put a primary key in. You'll get cut and paste in. You just do it over and over and over again. Put a primary key and again, so that we just have a little handle for each row, that we can point to later. That's exactly what we're doing. Then, we have a logical key. And I'm just kind of painting that with the green color. And then, we have kind of data columns, like these are just integers. This is string. This is integer, integer, integer. And then, we have to model this, right? And so, what happens is, we add the foreign key to the table, that is the beginning of the arrow. And this is actually, later we'll learn, this is a many-to-one relationship. Many tracks, meaning there will be many rows in the track table that point to the same album table. So, album has many tracks. So, this is a many-to-one, or you might say infinity-to-one, or the little-crow's-feet-to-vertical-bars. So, there's all kinds of ways on these pictures, which we'll take a look at in a bit, to capture these notions. But, we put the foreign key at the beginning of the arrow, that's what we do. We put the foreign key at the beginning of the arrow. And then, on the album table, we just make an album table, album ID, title. Okay, we might look up album's title, so we'll mark that as a logical key. So, if we keep doing this over and over and over again, it's pretty simple because albums belong to artists. So, we make an artist table with primary key, logical key. Albums have primarily key, logical key. Only a foreign key because there is an arrow between albums and artists. And then, we put genre and we got the genre ID and the genre name. And then we have an arrow there. And so, we have two foreign keys here and that's okay, that's perfectly fine. We had two arrows that started the track. And now we have two foreign keys. And we've got this naming convention, that genre, genres. I mean, you just look at it and go like, "Thank heaven." If I call this X, Y, Z, A, B, C, D, E, F, G, H, I, J, K, if I named them with really stupid names, we could still write code. And the program would not care. But naming conventions is really important so that you just don't go crazy. So, unless you have your own naming convention, follow my naming conventions. And you know, if you want to help from the teaching assistants and you're not using the naming conventions, we just kind of say, "Why don't you go back and just rewrite everything you've got using our naming convention because it makes our heads go poof." When you show me stuff that's not used in the naming convention and you ask my help. Okay, so let's make ourselves a database. And as usual, we have got a little hand out that's going to make our lives a little easier. Okay, so let's just get some of the work out of the way. Let's make a database called Music. So we got a Music database. Let me in the database. And we're going to make some tables. So, let's take a look at some of these create statements. So remember what we're doing is we're capturing these pictures. And so, we're going to build from the outside in. So we're going to work on artist first. And we're going to work on outside in because in a sense we have to establish this table before we can establish the table that points to it. So is I do this like one, two, three or four. This might be two. But this has going to have to be the last one because it really depends on the other. So you kind of work from the leaves of our tree inwards. So, we'll do create table artist and so this is from the previous lecture. This is just an auto-increment, not null integer, fine. Then there's a name and we're going to say the primary key is artist ID. Now, this is sort of what we did before, that's all we did. Next, the interesting one now is, we're going to create the table album. And so if you have an artist and the album, we had an arrow here. So we're going to have a primary key for the album. We're looking at this table right now. Primary key for the album, that's done. We say, not null increment, primary key. We have a title and artist ID. We're going to call it, we're going to use the MP3 index, because that's kind of our logical key. And so that stuff we've done before and now here's the new stuff. There's a lot of stuff in constraint. Foreign Key is like mySQL key words. And then what we basically say is, the column artist ID in this table references a column in the artist table named artist ID. So this is the syntax that we use on the create statement that establishes the arrow relationship from here to here. Now it turns out you don't actually need to do this and we'll talk a little bit later about this undelete, unupdate cascade. But what we're doing is super helpful to mySQL to performance tune what you're doing saying, "Look, this isn't just another integer column because up here we just said it's an integer." We're really distinguishing this. We're saying, "Not only is this an integer, but this is an integer that has a special kind of number in it. And it points to another table." So that's how we capture it. So look closely at this, okay? And so, if you look at the next thing, we're going to have the genre, right? The genre has a primary key. Again, you just cut and paste this stuff. When I start new projects, I come back to these lectures and I cut and paste this stuff and then tweak it and change it, right? So we got a integer, we got a index for the name, we're going to call the primary key. That's just another thing. And now we're going to create the table track. Primary key, logical key, data, foreign key, foreign key. We mark that this is the primary key. And we tell this that this is telling what our primary and logical key is within the table, right? So, that's telling us within table. And now, track has a link to album and a link to genre. So now, we've got these two foreign keys but now we have to inform mySQL about the workflow. So, our field album ID, this one here, points to album's album ID. So, this is table name, field name and genre ID. Our genre ID points to the genre table row indexed by genre ID. We're using this as a textual way to draw the picture. So the text doesn't need to be complex. If you know what the picture is, you should be able to directly convert from the picture to the text. The scientific, clever, creative bit was all in making the picture not in typing up the SQL. Once you know what the picture is, it's a rather manual process. And as a matter of fact, there's even tools that let you draw this stuff and then say make the SQL. I don't like those tools because they are too fancy. Because I like to have a look at this and I like to understand this because to me that's a picture. And I think it's actually very beautiful to capture that picture. Okay, so let's run some SQL now. And this is where cutting and pasting is super fun. So, I've got some got semi-colons at the end of these. So I'll start with one. I'll just create the artist table. And let's go in and create the artist table. Poof. Worked. Create the album table. Now, well let me do this, let me show you something. Let me try to create a track table and watch how it blows up. So the track points to the album table and the genre table neither of which have been created yet. Okay, so what I'm trying to do here is I have an artist table but I don't have an album or genre table and so I'm like, "Hey let's link out to these other tables." And it's going to say, "No, that was not making me very happy. Cannot add a foreign key constraint." Now you could even go to a stack overflow. Let's put this in a stack overflow just for Yucks, or just go to Google and see what happens. Reasons that you might not get a foreign constraint, foreign key blah blah blah blah blah. Who knows if that's a good question or not. That's half the promise stack overflows. You find stuff you don't know if it's good or not. So, but the mistake we made is we did not start from the leaves of this database and work our way in. So, we will go back and do this right now. And we will go and we will make, we have the artist table and now we can create the album table. And this part here is going to work, this part here, get rid of that and go like this. Artist exists and so we're going to establish a relationship between this album table we're about to make and the artist table. So, this is going to be happy. Click. And so if you look in artist, well we'll see this. So, if you look at artist, it knows about this stuff, and look at album it knows about this stuff, it knows that that's a foreign key, it's an index, it's a foreign key. There's other stuff over here so it kind of knows all this stuff. We've communicated a picture using text, okay? The next thing we're going to create is the genre table. Genre table just got a primary key because it's a leaf on the end of this. Create a genre to come, create the genre table. Easy money. And then we will create the track table. Now this is the one we tried to do before that blew up. But now that those two tables exist, it's going to be happy as a clam. There's our track. It's got you know foreign key, logical key, primary key. The primary key indexed. This has also got an index on it as well. But there's even more that it knows about that. We have all these values and it's time to insert the data. The other thing that we do is we insert the data from the leaves in because we have to establish those rows. So I'm going to start in the artist table because you've got to establish the rows. Now when we do this in a program like PHP, we'll actually be able to ask the database what these numbers are. So, now we're going to insert some data. And one of the things we have to do when we insert the date is we've got to kind of work outwards because we have to establish the numbers that go with each of these items, because we're going to point to them using the numbers. So, we're going to start with artist and then work our way in. And insert the two artists that we have Led Zeppelin and AC-DC. Now if you recall from the last time we talked about this with auto-increment. We're demonstrating auto-increment. So, now we can go into the artist table and you will notice when you look at the artist data that we have assigned these auto-increment. Now before it was like, "Oh that's magic, it works." But now we know that Led Zeppelin has always won. Now when we're writing PHP, code you'll be able to call the database and say, "Hey you just put in Led Zeppelin. What number did you give it?" And then we can know in the rest of our code to use one. But for now we're going to have to, we don't, we're not writing code. So, I'm going to have to say that Led Zeppelin is one and AC-DC is two. So, I wrote that down. So, now I have a primary key, an integer primary key for every time I want to mentioned Led Zeppelin anywhere in this data model from now on. Same thing is true for AC-DC. Now the next thing we have to insert is we'll put the genres in. And again we're not putting genre ID. We're in fact by inserts we're going to establish the genre ID, right? So I'm insert two genres and let's take a look. And that means that I got to write this down. Rock is one rock, and metal is two. So, now we are going to insert into album. Now this one here is interesting because we haven't had to mention the primary keys because they're auto-increment. But the foreign keys are not. We have to know what these numbers are. So what we're saying here is, let's put an album in of who made who. And that's artist number two and artist number two is AC-DC. Then let's put in an album name Four Ivy, and the artist name is Zeppelin. And that's number one according to my little sheet, right? Okay. So, the foreign keys we as the programmer are responsible for knowing the exact number of the foreign keys. And like I said when we're writing code, you'll see that this is easier. It'll tell you what these numbers are when you insert the artists and the genres. And so now an album, take a look at the album. So you'll notice here that this is what we've got. This is interesting because we have informed my SQL and PHP my admin also knows that this is a link. And so you'll notice that this is a highlighted link and we can actually dive in and we can follow the link to the artist. Not just a matter of work out steps. It did that. And so PHP my admin once we have informed it enough about these connections with those constraints that we typed in that said this column is related to a common in another table. Thank you very much. We know what we can do here and we can actually make use of it. And so the last thing we've got to do. Oh wait. Yeah, the last thing we got to do is we have to insert into the track. And this looks crazy. It's all data. An album ID and genre ID are the only things that matter. Oops, I forgot to write down the album ID. They will be written down on this piece of paper but I forgot to write them. But it doesn't matter. So, all this other stuff this is just data right here. Data, data, data that's data. That's the count of how many times I played it. This is album two and genre one. But again it's no different. So, I can put all these things in. Again like I said, it's easier when we write code to do this. But I'll put all these guys in. Oh by the way see how when I inserted this row, it says that was row 10 one. That's kind of like when the database tells you what the primary key that it chose. So it will know that this first track of Black Dog is number one and Stairway is number two and About to Rock is number three. And again in code we are handed this stuff back. So now if I get go take a look at the tracks and take a look at the track data, you'll see that we've got these two foreign keys. And you can dive through the foreign key into the actual record in the genre table. And again, you can see how now this is a lot easier if you have a really consistent naming and discipline in your naming conventions. Because if you didn't have discipline in your naming conventions, you would be like super crazy. So, if we were look at all of these things now, we have basically drawn, we have informed my SQL what's going on, what's connected to what, we have created foreign keys, we have informed my SQL about what our meaning of the foreign keys are. PHP my admin understands the foreign keys. That's why the little bits are blue. And we've put the right numbers in. That's our job as a programmer to know those numbers and put the right numbers in. But ultimately now you can see that we've created the picture. Now the other thing that you will see is that there is vertical duplication in the albums but these are integers. So, that's okay. It's more than okay. It's super awesome. You'll notice though that who made who only appears once in this entire database and the word rock only appears once. The rule of what we were trying to do was no vertical duplication of columns that has strings in them. It's okay to have vertical duplication of columns that have integers. And we've solved it. We did it. We made it. Now we blew our data into all these tables. And now we're going to do is bring it all back together.

Downloads

- Lecture Video mp4
- Subtitles (English) WebVTT
- Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?