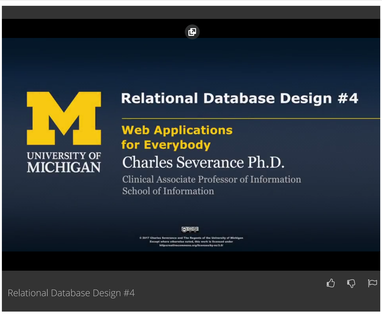
Lecture Materials

Assignment

Bonus Materials



Relational Database Design #4

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

Interactive Transcript

Search Transcript

English

0:08
So we have successfully moved all of our data into tables. And we've successfully made connections with those tables and the databases using primary keys and foreign keys. But literally, you can't go back until our UI person says, look at this new thing we did, this is your new user interface. People will be able to use our software, we'll just have to teach them about foreign keys, isn't that great? We'll just use phpMyAdmin as our user interface because it's so gorgeous and beautiful, or not. So our job is to take what we've done with all this splitting of the data and all this performance tuning, and then bring this all back together. Meaning we have to follow the foreign keys. So now we've established the foreign keys, we have to follow the foreign keys. And this is where we'll ask you to follow foreign keys and this is where it magically does it super fast.

0:55
So, the new SQL concept that we're going to come up with and understand next is the JOIN operation. So the JOIN operation says we're going to pull data from multiple tables simultaneously. And we're going to pull these things together. And there's that ON clause as part of it that says, here are the rules to connect this row to that row. It's that simple. So the JOIN says more than one table and the ON clause says the rules upon which we can do this. And so here's a really basic JOIN.

1:24
So what we have is we have an album table, and we have an artist table. We just want to show the album title and the artist name, we're not interested in these numbers. So what we have to is kind of follow all the corresponding links between these, except that that's what the JOIN does for us. And so we say, SELECT. And then there's a new syntax here, this is a list of columns. But if we're talking about more than one table, we say Album.title, Artist.name FROM, we've done that before and then the table name. So we're pulling from one table. And I intend to do it in the order of the arrows. So if the arrow starts here and goes there, so Album is the first thing I say. And then JOIN, that basically is we're taking data from both tables to the Artist table. So all that's really saying is the super set of data is both those tables. Album JOIN Artist is connecting those two tables and getting data from both of them. The ON clause is the more precise thing that says, when does a row from this table connect to a row in the other table. That's the ON clause.

2:32
So ON says, when Album's artist_id, Album's artist_id, when this column right here is equal to Artist's artist_id, which is that column right there. So I want to join these two rows together when that is true. So ON is the when this is, the actually makes the connection. So don't make the connection between all possible rows. You don't want this row connected to this row and this row connected to this row and this row connected to this row and that one. You could have four connections. I'm only interested in the connections that meet this criteria and that is when they match. And that's kind of always what we say. And again, you kind of see by the fact that I've named these things cleverly, it's easier to write these joins, right? Naming convention makes a lot of difference. And so what we see is it connects these things appropriately, and gives us back the title and the name. And so now we've gone from an nternal data model that's super efficient, and highly scalable, to a user interface that is showing the user what they want. And that's really the name of this album, and what is the name of the artist that does the album? So our job then is to reconstruct all this stuff, right?

3:46
And so if we sort of look at this in a little more detail, you can think of a JOIN as creating a super long meta row.

3:56
That is all combinations of rows, so all combinations, rows. And then you can think of the ON clause as filtering out the combinations that don't match. And so we can show this, this exact same SELECT statement, SELECT, we want the title, but this time we're going to actually show the artist_id in both of the tables. So this is from this table and this is from this table. And so artist_id, so we see the matching and then the artist name. So we see the name, we see the title. But we also see this thing where we create the super long row, that's all the data, and we're only showing the ones where at matches, okay? And that ON reduces this from four up here to two down here. Because there's only two records where this match happens. Now, if we go back, this is the many-to-one. We'll get there. Just because there's one on each one doesn't mean it's like this. There could be many Albums for one artist. But right now, because our data is so small, we're only seeing one. But if you go back, this is a many relationship and this is a one relationship. We'll see where they have more than one on the starting point, okay?

5:13
So here we have a little bit more, where we're going to go between the track title. Now we're going to have some, the are many, right? So if we go like this, this is the track table and this is the genre table, okay?

5:31
And what we've done here is there's no ON clause, okay? So there's no ON clause, and so now we're going to see all combinations of all rows. And if you recall, there were four things in the track table and two things in the genre table. And so every row in the track table, this is the many side, this is the one side. All rows, we're going to duplicate all rows here. So we're going to duplicate that and that, right? So that's one row, and we're going to duplicate again. So there is 4 over here, and 2 over here. So we see 8 total, okay? No ON clause, there's no ON clause. So we're getting rows where these two things don't match. And so you kind of see how the JOIN just sort of glues these things together in all combinations, right? If I had 20 rows here and 10 rows here, I would get 200 combined rows. So the ON clause is really important to throw away non-matching, all right, let me do that a little prettier, throw away the non-matching rows is what the ON clause really does. In this case the non-matching rows are going away. And so then you would only see the matching rows, which is what you wanted. I mean you hook these things together on purpose. It'll be like to say, hey, do an automatic ON clause for me because you know what I meant. because I called on the foreign key, why would I not want the ones where they don't match? Or why would I want the ones that don't match? But whatever, in SQL, you just have to say ON and then tell your matching condition, okay? And so now if we add that on, it just pulls that back down and shows the matches. And then we get rid of the things in the middle and now we see this vertical duplication has reappeared. But the vertical duplication of strings is not in the database, but in the UI, so this is the data that then somehow is sent out to the user, right? And so the replication that we wanted in the first place in our user experience, that's back. So they join for a brief moment. So this is not stored, this is constructed when you asked for it. So it's not wasting space and it's super easy to produce, but then it looks the way you wanted to look like. So it's reconstructing the vertical duplication of string data.

7:49
Now, it can get complex, but as long as you name things right, it's not so bad, right? And so, to reconstruct the things that we wanted before is we're going to select the Track's title, Artist name, the Album title, Genre name, going across four tables. And we're going to go from the Track table, join with the Genre table, join with the Album table, join with the Artist table, right? That's getting us across four rows. These are the fields we want, four rows. There are four tables and three relationships between the tables, right? We had this, we had this and this and this, we had an arrow, arrow, arrow. So there are four tables, three relationships. So there's an ON clause for each of the relationships and there is a JOIN from JOIN for all four. So there's four there and there's three here. And after awhile you type this in your sleep. Track.genre_id = Genre.genre_id, this is a foreign key, this is a primary key. And Track.album.id = Album.album_id, foreign key, primary key. And Album.artist_id = Artist.artist_id. Each one of these things is just the way we capture that relationship and walk into that other table when we want that data in a way that matches. So this isn't all that difficult to write.

9:07
We often sit in phpMyAdmin and type it until we get it right, make a syntax error, but ultimately, poof, out we get, and then we get it. And again, you see all this vertical duplication that's happening again. But again, this output of the SELECT is ephemeral, it's temporary. It's not stored in the data and it's not wasting any time or any space on the database to construct the data. But for the user, we're giving them what they want, okay?

9:34
Woah, so that's a lot of stuff. Remember, it's like a week ago when we started this thing. because we had the UI person come in, we saw some vertical duplication. We made four tables connected to one another, boom, boom, boom, figured that all that out. Then we made some create statements that captured that with some constraints in there. And then we started inserting data and sticking these on numbers and then keeping track of the numbers. And then we had to do a JOIN to bring it all back together. And when we're all done, we're just like, aah! We just started here and we ended up there. Why didn't we just type that into a Google Doc or something? And the answer is speed. Speed, scale, it seemed trivial, and that's because I'm using a tiny little bit of data. But literally, in a web application, you just can't afford. Because if you're successful, you're going to get people using your thing. If it doesn't matter, it doesn't matter at all. You're really going to get nobody or too many. So speed and performance is all the reason that we did all this stuff.

10:46
Okay, so let me pick up one little topic that I didn't talk about, okay? And that is the ON DELETE CASCADE. So let's go back and show you the ON DELETE CASCADE.

10:58
So remember how I said, ON DELETE CASCADE ON UPDATE CASCADE? I just throw these in.

11:07
This is like saying, in this table we're pointing to a row in another table. So here's a row in the table, four here, four here. The question happens is, what happens if this number changes to 9? Do we also want to change this to 9? Or what if we wipe this row out and there's a bunch of rows? because this is a many, this is a one, many, one. If we have a bunch, do we want to get rid of all these corresponding rows? And so what we are saying here is if this row is updated, cascade the changes into this table. Or if this row is deleted, cascade the changes into the table. So let's draw a picture of that, it makes it a little bit easier. So we have trial table and the parent table. These arrows are kind of going in the wrong way but the cascading is going on the other way. The relationship is kind of like the from to. This is the many side, this is the one side. This is the foreign key, this is the primary key. But what we're saying here is, there is this relationship that is this row is kind of a parent row to these rows, called parent-child. And what happens if I wipe this out, right? If I wipe that row out, what should happen? Promise the joins would start breaking. So what we're saying is with ON UPDATE, and there's other versions of this. But what we're saying with ON DELETE CASCADE is, if we wipe this row out, MySQL, you know about this relationship, follow it down and wipe those rows out automatically and instantly, okay? So that's ON DELETE CASCADE. And so after you've done that, those rows are now gone, because that's all that you've got left. And we do that to maintain database consistency. It's also kind of nice. You just know, I'll get rid of this thing, and all the child rows that depend on it, they're gone automatically. I mean if you didn't mean to delete it, then don't delete it. Now you don't have to use CASCADE. although CASCADE is kind of the one of the things if you say over restrict, you don't say ON DELETE CASCADE. It actually blows up when you try to delete the parent row. It says, there are children rows here, you can't do it. CASCADE says, if you delete the parent row, delete the children rows. And SET NULL means that you go up into the child rows and set them all to null which means make them empty. But that does mean that the join, there's nothing for it to join to. So, you as the programmer get to choose this. And this is just an example of you informing the database engine of your intentions. Like, I would like you to do this for me automatically.

13:27
I don't know how hard this is but I don't care, this is what I want you to do. And that's what you're doing with SQL and create statements is you're giving it a lot of information. So that's CASCADE. UPDATE CASCADE, we talked about that. But up to now, we've done everything that's one-to-many relationships. And now we're going to do many-to-many relationships, which is kind of the other major way of connecting tables 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?